ALTIVEC EXTENSION TO POWERPC ACCELERATES MEDIA PROCESSING

DESIGNED AROUND THE PREMISE THAT MULTIMEDIA WILL BE THE PRIMARY CONSUMER OF PROCESSING CYCLES IN FUTURE PCS, ALTIVEC—WHICH APPLE CALLS THE VELOCITY ENGINE—INCREASES PERFORMANCE ACROSS A BROAD SPECTRUM OF MEDIA PROCESSING APPLICATIONS.

Keith Diefendorff
Microprocessor Report

Pradeep K. Dubey
IBM Research Division

Ron Hochsprung
Apple Computer

Hunter Scales
Motorola Corporation

There is a clear trend in personal computing toward multimedia-rich applications. These applications will incorporate a wide variety of multimedia technologies, including audio and video compression, 2D image processing, 3D graphics, speech and handwriting recognition, media mining, and narrow-/broadband signal processing for communication.

In response to this demand, major microprocessor vendors have announced architectural extensions to their general-purpose processors in an effort to improve their multimedia performance. Intel extended IA-32 with MMX[1] and SSE (alias KNI),[2] Sun enhanced Sparc with VIS,[3] Hewlett-Packard added MAX[4] to its PA-RISC architecture, Silicon Graphics extended the MIPS architecture with MDMX,[5] and Digital (now Compaq) added MVI to Alpha. This article describes the most recent, and what we believe to be the most comprehensive, addition to this list: PowerPC's AltiVec.[6,7] AltiVec speeds not only media processing but also nearly any application in which data parallelism exists, as demonstrated by a cycle-accurate simulation of Motorola's MPC 7400, the heart of Apple G4 systems.

Highlights and performance summary

Like all the other extensions, AltiVec is a SIMD (single-instruction, multiple-data) extension to a general-purpose architecture. But the similarity ends there. Whereas the other extensions were obviously constrained by backward compatibility and a desire to limit silicon investment to a small fraction of the processor die area, the primary goal for AltiVec was high functionality. It was designed from scratch around the premise that multimedia will become the primary consumer of processing cycles[8] in future PCs and therefore deserves first-class treatment in the CPU.

Unlike most other extensions, which overload their floating-point (FP) registers to accommodate multimedia data, AltiVec dedicates a large new register file exclusively to it. Although overloading the FP registers avoids new architectural state, eliminating the need to modify the operating system, it also significantly compromises performance, which was not acceptable for AltiVec.

AltiVec treats multimedia data as first-class data in the form of vectors. Vector elements include all of the major data types found in 3D graphics, image processing, digital audio and video, speech recognition, data mining, and other multimedia applications.

AltiVec's powerful data reorganization capability goes far beyond that of any previous SIMD engine, making AltiVec uniquely well suited to the bit-parallel algorithms found in

0272-1732/00/$10.00 © 2000 IEEE

Table 1. Data types for various media tasks.

	Data type				
	8-bit integer		16-bit integer		Single-precision float
Task	Unsigned	Signed	Unsigned	Signed	Signed
Video		Low quality		High quality	
Audio				Low quality	High quality
Image processing		Low quality		High quality	
3D graphics				Low quality	High quality
Speech recognition				Low quality	High quality
Communication	Crypto		Crypto		
Media mining					High quality

digital signal processing (DSP) domains. These include error correction, bit-packing kernels, and many others.

AltiVec extends the scalar PowerPC architecture with a powerful new set of SIMD instructions. These instructions execute from the same instruction stream as the PowerPC's scalar integer, floating-point, and branch instructions.

AltiVec's major architectural characteristics include

- fixed-length 128-bit vectors, each comprising four, eight, or 16 data elements;
- a separate vector register file with a 32-register namespace, each register holding one 128-bit vector;
- vector-element data types of 8-, 16-, and 32-bit signed or unsigned integers, as well as IEEE single-precision floats;
- 162 new SIMD-style instructions optimized for digital signal processing;
- saturation or modulo arithmetic;
- a four-operand, nondestructive instruction format (three sources, one destination); and
- modeless operation for zero overhead use of AltiVec instructions.

SIMD parallelism is well matched to the parallelism found in the packed-data streams of media applications. To use SIMD processing, algorithms typically break long data streams into sequences of short fixed-length vector operands. SIMD instructions then process these vectors iteratively in loops, each instruction performing the same operation on all corresponding elements in the source-operand vectors in parallel. With AltiVec's long 128-bit vector, loop overheads tend to be small, giving AltiVec processors performance approaching that of true vector machines.

On the basis of cycle-accurate simulations of more than 40 media processing kernels, we found that AltiVec delivered an average speedup of 6.5 on integer kernels and 5.1 on floating-point kernels, over the same PowerPC processor without AltiVec. The speedups often approach—and sometimes even exceed—the theoretical SIMD parallelism, which is 16 on 8-bit data (for example, video), eight on 16-bit data (for example, modem filters), and four on 32-bit integers and floats (for example, 3D graphics and high-fidelity audio). Speedups greater than the theoretical parallelism arise from the ability to use new algorithms that are inappropriate for scalar processors or for less capable SIMD processors.

AltiVec architecture

One of the attributes that enable large speedups across such a broad spectrum of media processing applications is AltiVec's support for all of the important media data types. Table 1 shows the various data types that a processor must support if it is to perform well on media processing tasks. To date, AltiVec is the only SIMD architectural extension to support all these types.

AltiVec's large vector register file provides quick access to a large number of values, such as the transform or filter coefficients that are accessed frequently in signal processing loops. The large register namespace facilitates software pipelining and loop unrolling necessary to cover the long latencies associated with media streams. With a separate register file, the general-purpose and floating-point registers are not encumbered with multimedia data, so media processing doesn't interfere with scalar processing. The separate file also permits the vector registers to be physically optimized for the wide SIMD execution units.

Another important AltiVec feature is its four-operand instruction format (three source operands, one destination). This feature gives each instruction extraordinarily high operand

Table 2. AltiVec instruction-set summary.*

	Arithmetic				Source elements						Destination elements				
Instruction class	**Signed**	**Unsigned**	**Modulo**	**Saturate**	**Operands**	**Bytes**	**Halfwords**	**Words**	**Floats**	**Vectors**	**Bytes**	**Halfwords**	**Words**	**Floats**	**Vectors**
Load/store											X	X	X	X	X
Stream prefetch															X
Add/sub	X	X	X	X	2	X	X	X	X		X	X	X	X	
Multiply	X	X	X		2	X	X		X			X	X	X	
Multiply-add	X	X	X	X	3		X		X			X		X	
Multiply-sum	X	X	X	X	3	X	X						X		
Sum across	X			X	2										
Partial sum across	X	X		X	2	X	X	X			X	X	X		
Average	X	X	X		2	X	X	X			X	X	X		
Logicals			X		2					X					X
Rotate/shift	X	X	X		2	X	X	X			X	X	X		
Compare	X	X			2	X	X	X	X		X	X	X	X	
Select					2					X					X
Pack	X	X	X	X	2		X	X			X	X			
Unpack/merge	X		X		2	X	X					X	X		
Splat	X	X	X		2	X	X	X			X	X	X		
Permute		X	X		3	X					X				
Shift elements					2	X					X				
Round to integer	X				1				X				X		
Convert w/scale	X	X			1			X	X				X	X	
Max/min	X	X	X		2	X	X	X	X		X	X	X	X	
1/*x* estimate	X				1				X					X	
1/sqrt(*x*) estimate	X				1				X					X	
Log/power estimates	X				1				X					X	

*This table summarizes AltiVec capabilities in a concise form. Not all combinations shown are available for every instruction in a given class.

bandwidth and supports the encoding of powerful instructions such as multiply-add, permute, and select (described later). Since the four-operand format is nondestructive, it also eliminates the excess register shuffling and copying that comes with destructive two-operand formats like that of the x86 architecture. Thus, AltiVec's instruction format allows programs to use registers efficiently, minimizing spill/fill traffic to memory and producing a short instruction path, which are both important for efficient signal processing loops.

AltiVec is based on a simple RISC-style load/store architecture, but instructions operate on vector operands rather than on the simple scalar operands of classical RISC engines. The AltiVec instruction set was distilled from many digital-media-processing algorithms into a set of generalized primitives that support common operations such as saturation arithmetic. Using this approach, the design can support a wide spectrum of media applications while avoiding the highly specialized instructions commonly found in traditional DSPs. Counting all variations of data types and arithmetic (modulo, saturation, signed, and unsigned), AltiVec adds 162 new instructions to the PowerPC architecture, as summarized in Table 2.

The AltiVec design criteria called for all instructions to be easily pipelined and suitable for superscalar, out-of-order dispatch. All AltiVec processors are expected to implement the full architectural vector width and to fully

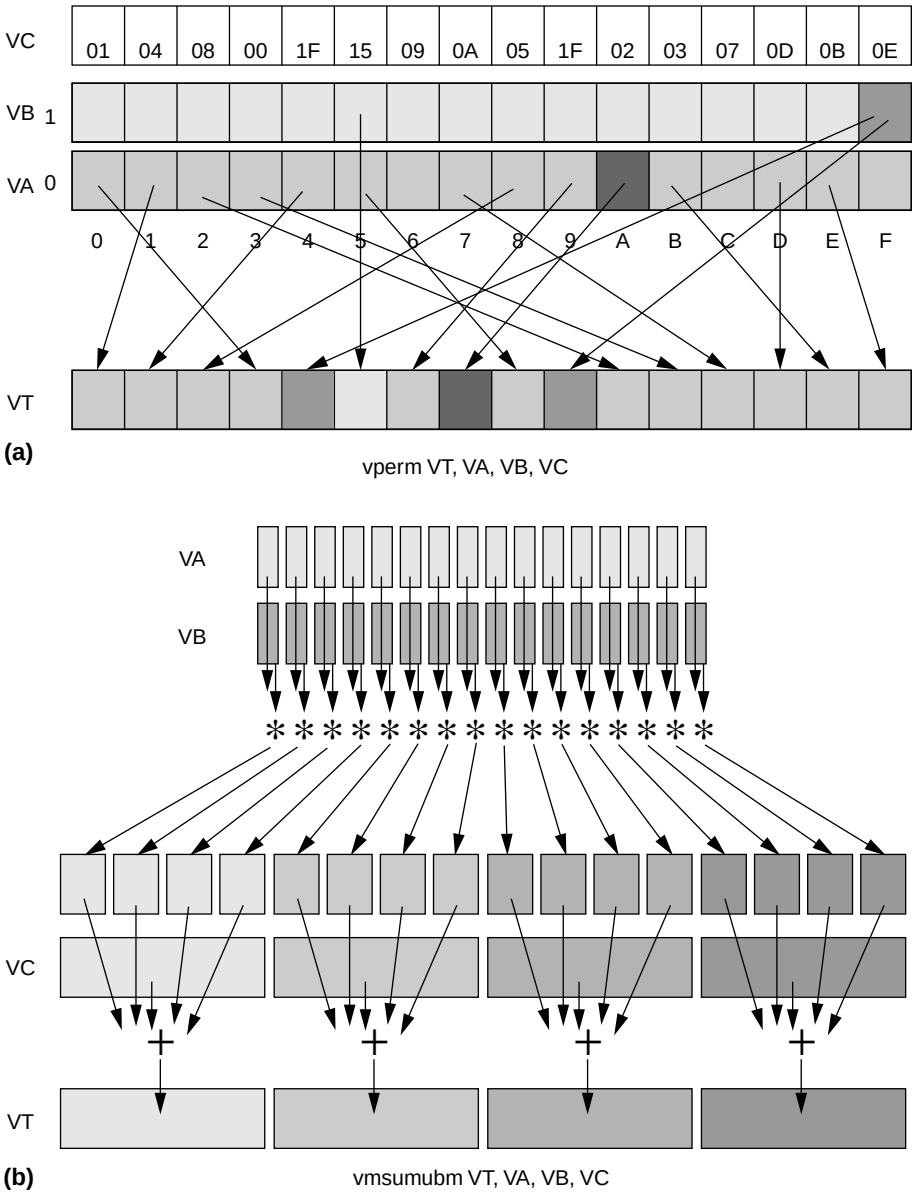

Figure 1. Vector permute (a) and vector dot-product (b) primitives in AltiVec.

pipeline all instructions; that is, all instructions will issue back-to-back with a throughput of at least one instruction per cycle. Most implementations will support simultaneous dispatch of one ALU-class vector instruction along with one permute-class vector instruction, or either of these paired with a vector load or store. Thus, the peak throughput of AltiVec instructions will typically be two per cycle, as it is in Motorola's MPC 7400 processor. No AltiVec implementations will ever impose any restriction on, or suffer any penalty for, mixing vector instructions with PowerPC scalar instructions.

Permute power

Much of AltiVec's performance and flexibility derives from the permute instruction (vperm), illustrated in Figure 1a. This instruction performs two essential functions: data reorganization and table lookup. One of the historical problems with SIMD architectures is that if the data structures do not precisely match the hardware organization, the program must preprocess the data to conform to the hardware. This preprocessing overhead severely reduces the potential SIMD speedup. Permute eliminates this problem by providing a method for arbitrarily rearranging vector elements with a single instruction. Permute's dual source-vector operands (VA and VB) enhance this capability by allowing rearrangement of data across vector boundaries.

The other important use for permute (table lookup) is discussed later. Although permute requires a full 32×16 bytewise hardware crossbar, the enormous value of permute's table lookup function alone makes it worth the incremental hardware cost over that of the more restrictive data shuffling operations found in other SIMD machines.

In addition to the full generality of the permute instruction, AltiVec provides specific variants for unpacking (expanding) small elements into larger fields, packing (truncating) large elements into smaller, tightly packed fields, merging (shuffling) elements from two vectors into one vector, replicating an element across a vector (splat), and double-vector shifting and rotating. These variations specify the permute-control vector implicitly or as an

explicit literal within the instruction opcode, thus avoiding the overhead of creating and storing the permute-control vector in these common cases. Special forms of pack and unpack are provided for 16-bit pixels in a 1/5/5/5 format.

Vector dot product (multiply-sum)

One of the most common DSP operations is the vector dot product. AltiVec accomplishes this operation with two instructions: multiply-sum and sum-across. Multiply-sum, illustrated in Figure 1b, multiplies corresponding elements in two vector registers (VA and VB), sums those products with four values from a third vector register (VC), and deposits the four 32-bit partial sums into the destination vector register (VT). VC serves to accumulate partial sums for taking the dot product of long vectors. AltiVec processors carry out the multiplications to full precision and then subject the individual partial sums to saturation (clamp to max on overflow, min on underflow). As a final step, the sum-across instruction can be used to sum the four accumulated partial sums into a single 32-bit scalar result.

AltiVec provides multiply-sum in byte- and halfword-element forms. The byte-element forms support motion estimation in video compression. During motion search, the multiply-sum instruction is used to locate the closest matching macroblock using the sum-of-differences-squared $(a_i - b_j)^2$ measure. This approach produces a higher quality comparison than one based on the sum-of-absolute-difference ($|a_i - b_j|$) instruction used by architectures such as VIS and SSE, while still achieving a throughput of 16 pixels per cycle.

Multiply accumulate

Another common DSP operation is multiply-accumulate. This operation underlies many digital filters, mathematical transforms, matrix-arithmetic operations, and so on. AltiVec provides multiply-add, a more powerful version of multiply-accumulate, with three source operands and one destination. With multiply-add, the respective elements in two source vectors are multiplied and the products added to corresponding elements of a third source vector. The intermediate calculations are carried out to infinite precision, and the final product sums are truncated to fit into destination-register fields the same size as the source elements.

For halfword elements, AltiVec's multiply-add has two forms: multiply-add-low and multiply-add-high. In the low form, the unsigned accumulator elements are added to the full product, and the intermediate product-sums are truncated modulo 2^{16}. In the high form, the signed accumulator elements are left justified (7-bit left shift), added to the 32-bit intermediate products, and then saturated to fit into the 16-bit destination element fields. The 7-bit left shift (as opposed to 8-bit) results in one extra bit of precision by taking advantage of the fact that the most significant bit of each signed 32-bit intermediate product is redundant. A variant of this form rounds the intermediate product-sums to squeeze out an additional half bit of precision. These tricks provide additional precision that is important to several algorithms, especially audio processing.

Multiply-add is not provided for byte elements because 8 bits of precision is not sufficient for most DSP algorithms. In most algorithms involving 8-bit data, the elements are first expanded to 16 bits, where most computations are carried out, and the final results are truncated back to 8 bits. Video compression and decompression, which use 8-bit values throughout, are exceptions, but the operations involved in those algorithms are more suited to multiply-sum, which does have 8-bit forms. In the few cases that require multiply-add of byte elements, vector-multiply and vector-add instructions are provided.

As a concession to silicon area, AltiVec does not provide a vector-multiply or multiply-add for 32-bit integers. For applications requiring more than 16 bits of precision, such as high-fidelity digital audio, 24-bit precision is usually sufficient. AltiVec provides this level of precision with its floating-point instructions, which do include a multiply-add. Floating point also provides the wide dynamic range that is often the real motivation to use integers larger than 24 bits.

Conversion between integer and floating-point formats is very fast in AltiVec. A single instruction converts and scales a vector of four signed or unsigned 32-bit fixed-point words to a vector of four single-precision floating-point values, or vice versa. An instruction for round-

ing floating-point values to integral values via any of the four IEEE-754 rounding modes[9] is also provided. Since all AltiVec instructions are fully pipelined, SIMD floating-point arithmetic throughput is similar to that of SIMD integer arithmetic as long as the floating-point unit's pipeline latency can be covered, which it usually can. With these floating-point features, AltiVec sacrifices little by not directly supporting 32-bit-integer multiply.

For division and square root, AltiVec uses Newton-Raphson refinement of a reciprocal seed. The high accuracy achieved with AltiVec's fused multiply-add instruction (single rounding after the add operation) allows rapid convergence to an IEEE-754–accurate reciprocal value. This approach provides divide performance equaling or exceeding that of processors with expensive division hardware. Unlike with hardware dividers, AltiVec's approach carries no hidden-state information from cycle to cycle; thus, division can be fully pipelined and intermediate instructions can be easily rescheduled.

Conditionals

Changes in control flow present a serious performance problem for any processor, especially those running DSP applications where loops tend to be tight and the data-dependent decisions difficult to predict. The latter are nearly intractable in SIMD architectures, which process multiple data elements in a single instruction.

Although AltiVec provides a means for conditional branching, it places more emphasis on avoiding branches. To this end, AltiVec offers a type of conditional-move instruction, called select (vsel), which, in concert with vector-compare instructions, operates efficiently on SIMD vectors. The vector-compare instructions generate a predicate vector that can be stored in any vector register and used by subsequent vsel instructions to choose elements from one of two registers, depending on the value of the corresponding predicate elements. With this mechanism, data-dependent decisions can be made on all elements in a vector in parallel, making it unnecessary to test and branch on each element individually. AltiVec can use the vsel instruction to simulate predicated execution, which can eliminate many branches. Since vsel selects values on a bit-by-bit basis, it is also useful for selecting and merging vector subfields that do not fall on element boundaries.

In cases where data-dependent redirection of program control flow cannot be avoided, AltiVec's vector-compare instructions optionally update PowerPC's condition register (CR_{06}) field. Subsequent PowerPC conditional-branch instructions can test this register in the normal manner. A special form of vector compare—vector compare-bounds—speeds up 3D-graphics clipping operations.

Loads and stores

The philosophy behind AltiVec's memory operations is to support only basic load and store primitives in an effort to keep the memory path as fast as possible. The load-vector and store-vector instructions transfer full quadword vectors between memory and the vector registers. Load-vector-element and store-vector-element instructions transfer individual byte, halfword, and word scalar elements between memory and the vector registers. Vector loads and stores use the index-addressing mode (RA|0 + RB) only.

All memory accesses are aligned on their natural size boundary. If a load or store's address is not size aligned, the appropriate number of least-significant address bits is ignored and an aligned transfer occurs. AltiVec provides assistance for extracting misaligned data once it is in the registers. Special load-vector-for-shift-left/right instructions assist in this process by computing a permute-control vector based on the misaligned memory address. Random isolated unaligned-vector loads can be simulated with just four instructions, but the average cost of unaligned-vector loads in a long linear sequence, which is the more important case, approaches an average of only two instructions (one load vector and one vector permute). These two instructions will issue simultaneously in most AltiVec implementations.

Software-directed prefetch streams

AltiVec allows software to manage the bandwidth between processor and memory with explicit cache management instructions. With these instructions, software can indicate to the cache hardware how it should prefetch data and prioritize replacement. The principal instruction for this purpose is the cache-prefetch

instruction, called data-stream touch. This instruction specifies the starting address, a block size (one to three vectors), the number of blocks to prefetch (one to 256 blocks), a signed stride (± 32,768 bytes), and a 2-bit tag that uniquely identifies one of four prefetch streams that can run simultaneously. Other forms of data-stream touch are provided for writing data into streams and marking data as transient, that is, as having poor temporal locality.

AltiVec coding examples

AltiVec has proven adept at accelerating a wide range of multimedia and DSP applications. To illustrate its utility, we present coding examples from the areas of image processing, wavelet signal processing, and Galois-field arithmetic.

Median filter

Median filter is an algorithm commonly used in image processing to smooth out noise. The support region of the pixel neighborhood in such a filter is normally a sliding filter window. Since the data type is usually 8-bit unsigned (intensity) values, the sequence of scalar memory accesses typically involves numerous misaligned memory accesses. Figure 2 provides a high-level description of a 5×5 median filter based on a simple algorithm. The two most important functions are pixel load and compare-and-swap.

In Figure 2, one pixel neighborhood appears in the dashed box. Twenty-five vectors are needed to calculate 16 output pixels in the third row. Even though pixel sets are misaligned, the use of AltiVec's load and shift features completely avoids expensive unaligned memory accesses. Additionally, with this approach each pixel is fetched only once, thus minimizing the number of memory accesses.

This example illustrates AltiVec's double-shift instruction, vsldi. The critical innermost loop of compare-and-swap (V1, V2) can be implemented using a simple three-instruction sequence of vmaxub Vx, V1, V2; vminub V1, V1, V2; vor V2, V1, Vx. This sequence, which accumulates the minimum pixel value in V1, can be executed in just two cycles on a typical AltiVec superscalar implementation, resulting in a significant speedup over scalar processors for median filters, even with this simple algorithm. With a more sophisticated algorithm, AltiVec can execute median filter at 1.2 cycles per output pixel.

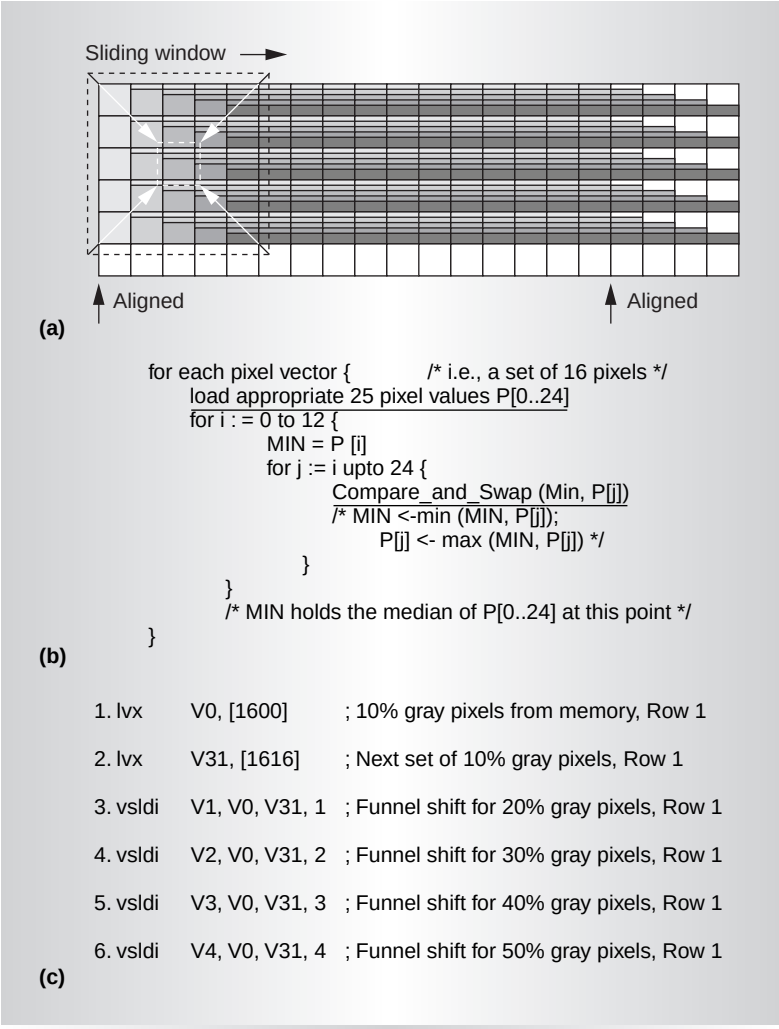

Figure 2. Pixel access during 5×5 median filtering.

Haar transform

The applications of wavelet technology have increased markedly in areas such as digital image processing, data compression, astronomy, acoustics, sub-band coding, imaging, speech discrimination, and fractal compression. The Haar transform, introduced in 1910, is the oldest wavelet transform. The 2×2 transform decomposes an image into four bands whose spatial frequencies and spatial contents differ.

Figure 3 (next page) illustrates this transform, along with its implementation in AltiVec code. This example illustrates the use of the multiply-sum operation. Support for

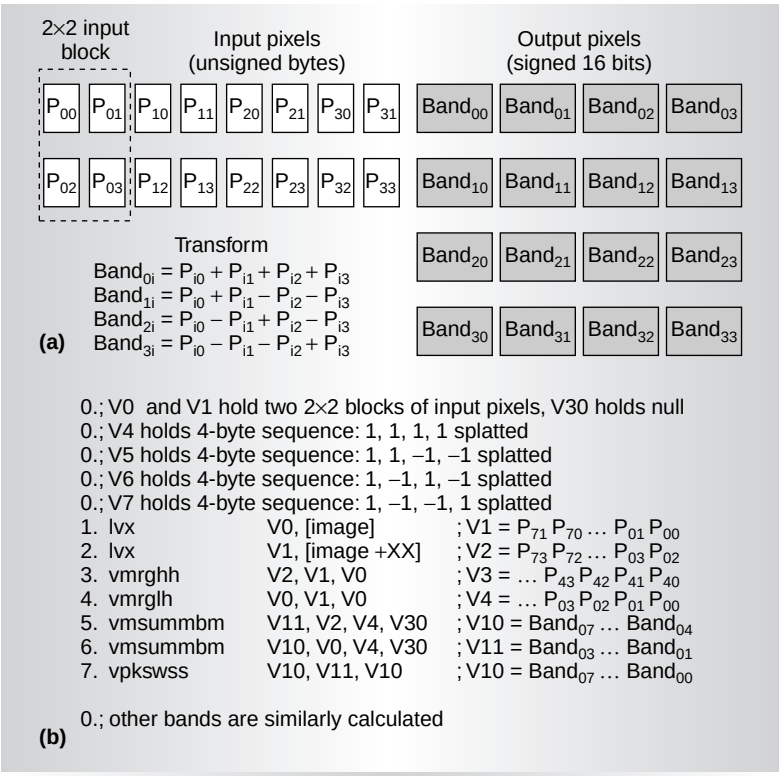

Figure 3. A 2 × 2 Haar transform (a) implemented in AltiVec code (b).

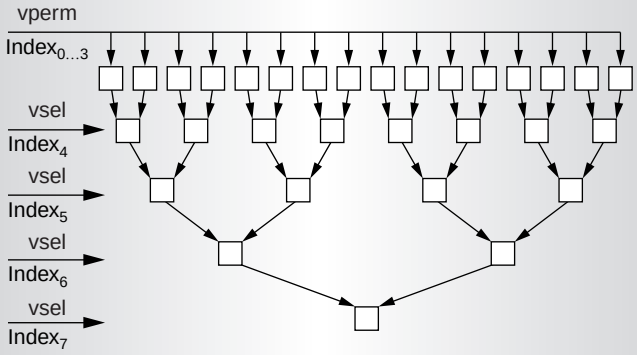

Figure 4. 256-entry, 8-bit-element table lookup using AltiVec.

one signed and one unsigned operand is crucial for filtering applications like this, where the signed filter coefficients are small enough to fit in a byte and the input is unsigned 8-bit data. A video-loop filter, which performs smoothing in YUV color space, is another such application. Here the convolution kernel is separable and has coefficients small enough to fit in a byte. In the absence of architectural support for byte data types, the program would have to unpack the data and extend the coefficients to 16 bits, losing half the potential parallelism.[1,3]

Galois-field arithmetic

Table lookups involving small tables (256 or fewer elements) and small elements (8 or fewer bits) are common in multimedia and communication kernels such as cyclic redundancy check (CRC) code generation, convolution encoding, Galois-field (GF) multiplication, and many nonlinear functions, such as gamma correction. Also, bit-level manipulations, including various simple functions such as bit-stream interleaving or bit (un)packing, are often programmed using table lookup functions. Most traditional scalar and SIMD architectures do not support parallel implementation of such functions, so, for example, a sequence of 16 index extractions, 16 address calculations, and 16 data loads would be needed to load 16 data items based on 16 indices. AltiVec accomplishes all of these operations with one vperm instruction.

This powerful capability enables sophisticated techniques like GF arithmetic, unlocking entirely new algorithms that are computationally intractable on conventional scalar and SIMD engines. Figure 4 illustrates the steps in performing a 16-way parallel lookup in a 256-element table (with byte elements).[1,3]

GF arithmetic, that is, arithmetic over a finite field, has many important applications. One example is Reed-Solomon coding, which is a type of block code commonly used for error correction in broadband communications. With add-form representation, GF addition is just an exclusive-or operation. GF multiplication, however, is expensive. For example, a GF (2^4) multiplication involves three table lookups into two 16-element tables and a modulo-15 addition, as shown in Figure 5. Note that the more commonly used GF (2^8) multiplication can be carried out using three GF (2^4) multiplications and four GF (2^4) additions.[10]

Performance results

The AltiVec performance data presented in this article is based on cycle-accurate simulation of Motorola's MPC 7400 microprocessor with AltiVec. The simulated processor has several important microarchitecture characteristics:

- all instructions are fully pipelined with single-cycle throughput;
- simple operations (simple arithmetic, logical, permute) have a 1-cycle latency;
- compound operations (for example, multiply-sum) have 3 to 4 cycles of latency;
- the processor can issue one ALU-class and one permute-class instruction each cycle (permute class includes most nonarithmetic operations such as pack, unpack, splat, permute, and so on); and
- there is no restriction on issuing AltiVec instructions in the same cycle as PowerPC scalar instructions.

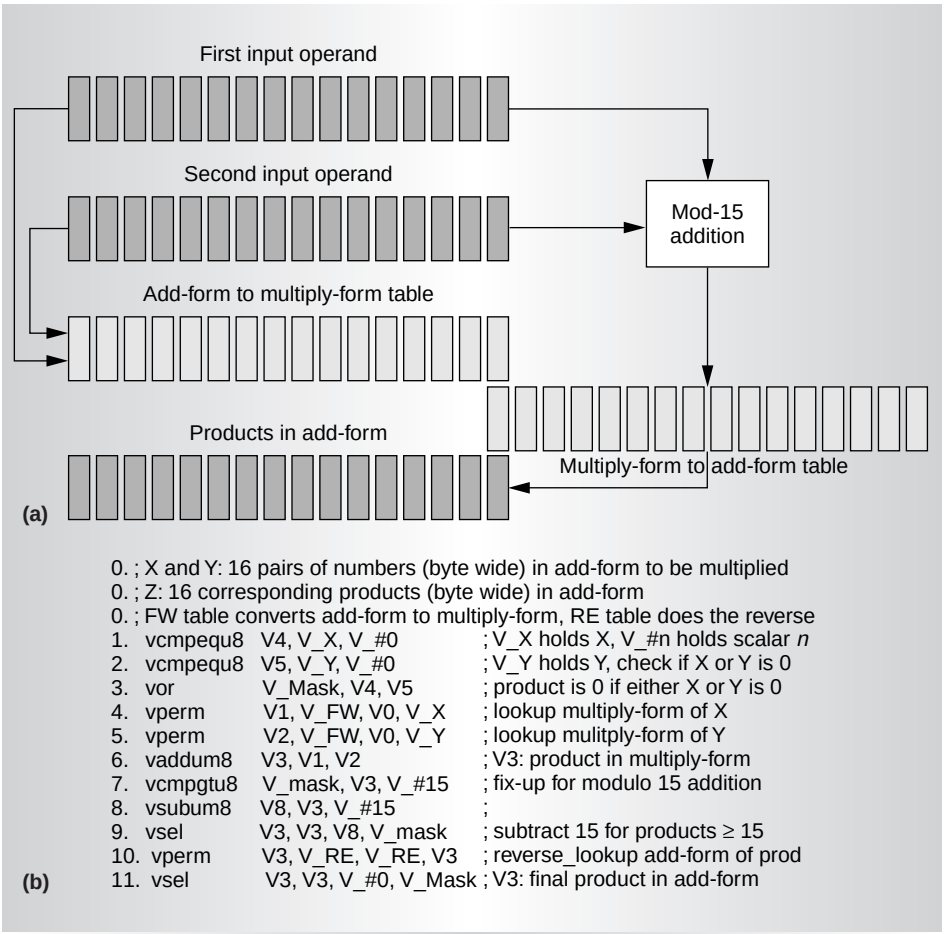

Figure 5. Galois-field (16) multiplication (a) implemented in AltiVec code (b).

Table 3 (next page) summarizes performance on a variety of media functions, chosen to cover a wide spectrum of media processing, including audio, video, graphics, speech recognition, and communication. The optimal scalar algorithm for a media function is often different from the optimal AltiVec algorithm. Therefore, to present a fair comparison, we made every effort to pick the best algorithm for each case.

The AltiVec extension to the PowerPC architecture was designed to offer architectural support for the entire range of multimedia processing. Multimedia processing has its roots in signal and image processing, where overall performance is often determined by the performance of critical loops iterating over a large input data set. AltiVec's large vector register file, full-range data-type support, four-operand nondestructive instruction format, unmatched data-reorganization capability (permute), and powerful SIMD instruction set enable it to significantly speed processing in these critical loops. At the same time, AltiVec's data-prefetch streams can keep the SIMD processing engine fed with data. The speedups achievable with AltiVec are sufficient to enable media-rich applications to run entirely on general-purpose PowerPC microprocessors without the aid of hard-to-use special-purpose media processors or dedicated hardware accelerators. MICRO

References

1. A. Peleg and U. Weiser, "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, July/Aug. 1996, pp. 42-50.
2. K. Diefendorff, "Katmai Enhances MMX," *Microprocessor Report*, Oct. 5, 1998, pp. 1, 6-9.
3. M. Tremblay et al., "VIS Speeds New Media Processing," *IEEE Micro*, July/Aug. 1996, pp. 10-20.
4. R.B. Lee, "Subword Parallelism with MAX-2," *IEEE Micro*, July/Aug. 1996, pp. 51-59.

Table 3. AltiVec performance on various media functions.

Function	Data set	Cycles using AltiVec	Speedup over optimized scalar code on same processor
Motion estimation	176 × 144, 8-bit	90.7/macroblock	16
Quantization	8 × 8 DCT output block, 16-bit → 8-bit	96.8	12.5
8 × 8 inverse discrete cosine transform	8 × 8 image block, 16-bit	101.7/block	12.3
Inverse fast Fourier transform	128 complex taps, floating point (FP)	1,700	3.5
Windowing (Dolby AC3 function)	256 elements, FP → 16-bit	834/kernel	4.9
Matrix-matrix multiplication	Two 4 × 4 → one 4 × 4, FP	36.5	6.2
Matrix-vector multiplication	4 × 4 arrays times 4-element vectors, FP	5.6/vector	Not available (n/a)
Transform, perspective, projection	800 vertices, 400 normals, FP	28,800	2.5
Buffer accumulation	[RGBA] (8-bit), FP scale factors	5.3/pixel	17.5
Bezier curve drawing	Four 16-bit → 64 16-bit points	2.48/point	6.3
3 × 3 median filter	128 × 128 pixels	1.23/pixel	n/a
Separable convolution	512 × 512 pixels	1.94/pixel	n/a
Bilinear interpolation	128 × 128 pixels	26.7/pixel	6.4
Color-space conversion	4,800 pixels	2.25/pixel	n/a
L-filter[11]	128 × 128 pixels	5.23/pixel	n/a
13-tap, real finite impulse response	AltiVec: 16-bit; scalar PPC: FP	4.25/output	9.4
Linear prediction, Levinson-Durbin	12 LP coefficients, FP	388	3.2
Linear prediction, Schur recursion	12 LP coefficients, FP	238	7.1
Bit-packing in 64-QAM (quadrature amplitude modulation) demodulation[12]	32 16-bit → 384 bits	35	10.5
CRC-32	128-bit → 32-bit	1.312/byte	n/a
Autocorrelation	256 8-bit samples → 16 32-bit coefficients	676	30.7
GSM (global system for mobile communications) module 4.2.11	Signed 16-bit, 60,600 samples	1,034	12.5
2 × 2 forward Haar transform	Eight 2 × 2-pixel blocks → 16-bit frequency bands	0.375/pixel	n/a
Sorting using Batcher sort	16 8-bit elements	76	10.0
(2,1,3) convolution encoder	Two sets of 125 bits → 256 bits	19	n/a
Gamma correction,[13] ITU-R 709	8 bit → 8 bit	0.625/pixel	n/a
Arbitrary permutation	128-bit	20	n/a
Associative search	Two 32-entry tables, 16-bit keys and tags → 16-bit tag	13	5.8
Gaussian elimination	FP, 16 variables	9,824	1.42

5. "MIPS Digital Media Extension," *Instruction Set Architecture Specification*, http://www.mips.com/MDMXspec.ps (Oct. 21, 1997).
6. "AltiVec Extension to PowerPC," *Instruction Set Architecture Specification*, http://www.motorola.com/AltiVec (May 7, 1998).
7. M. Phillip et al., "AltiVec Technology: Accelerating Media Processing Across the Spectrum," *Proc. Hot Chips 10*, 1998, available at http://www.hotchips.org.
8. K. Diefendorff and P. Dubey, "How Multimedia Workloads Will Change Processor Design," *Computer*, Sept. 1997, pp. 43-45.
9. ANSI/IEEE Std 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE Press, Piscataway, N.J.
10. E.R. Berlekamp, *Algebraic Coding Theory*, Aegean Park Press, Laguna Hills, Calif., 1984.
11. I. Pitas, *Digital Image Processing Algorithms*, Prentice Hall, New York, 1993.
12. C. Abbott et al., "Broadband Algorithms with the MicroUnity MediaProcessor," *Proc. Compcon 96*, IEEE Computer Soc. Press,

Los Alamitos, Calif., 1996, pp. 349-354.
13. C. Hansen, "MicroUnity's MediaProcessor Architecture," *IEEE Micro*, July/Aug. 1996, pp. 34-41.

Keith Diefendorff is editor-in-chief of *Microprocessor Report*. He worked on the AltiVec architecture while at Apple Computer, where he was a Distinguished Scientist and the director of microprocessor architecture and strategy. He received an MSEE from the University of Akron, Ohio, and is a member of the IEEE and the Computer Society.

Pradeep K. Dubey is a senior manager at the IBM India Research Lab, where he works on topics related to multimedia applications. His work includes design, architecture, and performance issues for Intel's x86 and the PowerPC processors. He received a BS from Birla Institute of Technology, India, an MSEE from the University of Massachusetts, and a PhD from Purdue University. He is a senior member of the IEEE.

Ron Hochsprung is a Distinguished Engineer at Apple Computer. He worked on the system architecture of Apple's platforms and was a member of the PowerPC architecture development team. He graduated from the Illinois Institute of Technology and is a member of the IEEE.

Hunter Scales has been with Motorola Semiconductor since 1978, working in applications engineering and microprocessor design and architecture. He worked on the 68000, 88000, and PowerPC microprocessors. He was educated at the University of Austin, Texas.

Contact Keith Diefendorff, Cahners MicroDesign Resources, 298 S. Sunnyvale Ave., Suite 101, Sunnyvale, CA 94086-6245; keithd@mdr.cahners.com.

HOW TO CONTACT IEEE MICRO

IEEE Computer Society
PO Box 3014
Los Alamitos, CA 90720

Paper, electronic, or combination subscriptions to *IEEE Micro* are available. See the Reader Service Card in this magazine for a subscription form. Send subscription change-of-address requests to address.change@ieee.org. Be sure to specify *IEEE Micro.*

Membership Change of Address
Send change-of-address requests for the membership directory to directory.updates@computer.org.

IEEE Micro on the Web
Visit our Web site at http://computer.org/micro for article abstracts, access to back issues, and information about *IEEE Micro*.

Writers
Author guidelines and IEEE copyright forms are available at http://computer.org/micro/author resources.

Letters to the Editor
Send letters to Managing Editor, *IEEE Micro,* 10662 Los Vaqueros Circle, PO Box 3014, Los Alamitos, CA 90720. Please provide an e-mail address with your letter.

Reprints of Articles
For price information or to order reprints, send e-mail to micro-ma@computer.org, or fax to (714) 821-4010.

Reprint Permission
To obtain permission to reprint an article or column, contact William Hagen, IEEE Copyrights and Trademarks Manager, at w.hagen@computer.org.

Subscription Questions, Missing or Damaged Copies
If you did not receive an issue or received a damaged copy, contact membership@computer.org.

New Product releases
Mail microprocessor, microcontroller, operating system, microsystem, embedded system, and related systems announcements to *IEEE Micro* at above address.

Micro Law

continued from p. 11

unnecessary to address the doctrine. This failure may have been a tactical error, since an argument could be made that Xerox's diagnostic software meets all of the requirements of the doctrine. That in turn could have precipitated a decision on the applicability of the doctrine to intellectual property. (Probably, however, a majority of the judges of the Federal Circuit would hold that intellectual property is outside the doctrine.)

Is copyright different?

The discussion so far has centered on patents. Do different technical or legal considerations also apply to copyrights?

Different technical considerations can apply. But that is usually simply a matter of a particular case's fact pattern. It may well be that diagnostic software is difficult to duplicate. Among other things, an ISO competitor may lack the necessary technical knowledge of the engineering of the system, so that the cost of developing diagnostic software is prohibitive. (The *Chicago Bulls* case recognized that as a legitimate consideration.) In the *Lotus v. Borland* case, it was impossible to provide a spreadsheet file conversion utility for Lotus 1-2-3 without embodying in the computer program Lotus' command structure. The court of appeals sidestepped the copyright infringement issue by holding such subject matter uncopyrightable, on what appeared to be antitrust policy grounds.

In other cases, a copyright may be easier to avoid than a patent, particularly because copyright law has a fair-use defense and patent law does not. In addition, software patents and business method patents protect their owners at a higher level of generality or abstraction than software copyrights do.

At the policy level, however, different considerations don't apply. Refusal to deal in or license copyrighted subject matter should be neither more nor less privileged than in the case of patents. Arguably, the antitrust status of a refusal to sell or license copyrighted software is more problematic because the copyright law lacks an explicit counterpart to the patent law's section 271(d). Indeed, when Congress enacted section §271(d)(4), it failed to adopt proposals for more pervasive patent *and copyright* immunities. Nonetheless, analysis of the total structure of the copyright laws better supports treating copyright-related refusals to deal on a par with patent-related refusals to deal than would according them disparate treatment.

In the *Data General* case such considerations led the federal appeals court in Boston to conclude that copyright owners were entitled to a presumption of entitlement to refuse to deal absent a showing of privilege-dissipating conduct. The copyright law gives copyright owners comparable exclusionary rights as to reproduction and distribution of software to what patentees have for manufacture, sale, and use. The existence of such rights implies a corollary right not to relinquish these exclusionary rights—in other words, a limited right to refuse to deal—unless the copyright owner is willing voluntarily to do so. The incentives that the copyright system offers would be significantly diminished if copyright owners had no right at all to refuse to deal. That in turn implies such a right. (This conclusion does not compel, for its recognition, any resort to a theory of an absolute right to refuse to deal, for the same reasons discussed earlier regarding patents.)

For all these reasons, the better view is that patent and copyright owners alike have a qualified right not to sell patented or copyrighted products, or license the use of patented or copyrighted material. The right is not absolute. Rather, there is a presumption favoring an intellectual property owner's right not to deal. The presumption is overcome by a showing of appropriate facts. Such facts are use or acquisition of the intellectual property in a manner otherwise unlawful because the use falls within one or more of several categories of condemned behavior: conspiracy, coercion to commit unlawful acts, and purchase of market control. Perhaps (but this is very, very iffy) another exception exists in which intellectual property ownership invokes the essential facilities doctrine.

These comments end this two-part series on IP-related refusals to deal. However, as I mentioned in Part 1, we may very well see this controversy reach the US Supreme Court because of the conflicting decisions in different appeal courts.

Micro Review

continued from p. 13

see is your name on the screen, you know you have work to do. Steve Gibson leads you through some simple steps that will greatly reduce your vulnerability.

One of Gibson's recommendations is to use the Zone Alarm 2.0 personal firewall, which you can obtain for free from Zonelabs.com of San Francisco. I downloaded this product and have been using it more or less successfully. Zonelabs does not certify the product for the configuration I'm running, and so I don't blame them completely for the occasional blue screens of death that occur when the program fails to handle kernel mode exceptions properly.

Using Zone Alarm has been very revealing. It quite regularly reports probes from unauthorized IP addresses. I'm still tweaking the settings and learning to use it effectively, but I'm happy to have it running, despite the