MULTIMEDIA INSTRUCTIONS IN IA-64

Ruby B. Lee, A. Murat Fiskiran and Abdulla Bubshait

Department of Electrical Engineering
Princeton University, Princeton, NJ 08540, USA
rblee@ee.princeton.edu

ABSTRACT

We discuss the integer and floating-point multimedia instructions in the IA-64 instruction-set architecture (ISA). These multimedia instructions implement subword parallelism, also called packed parallelism or microSIMD parallelism. They are both a subset and a superset of the multimedia instructions from the predecessor architectures: MMX, SSE and SSE-2 from the IA-32 architecture, and MAX and MAX-2 from the PA-RISC architecture. We discuss the novel subword permutation instructions that are new in the IA-64, and their effectiveness, in combination with the subword arithmetic instructions, for speeding up multimedia programs. These packed arithmetic and permutation instructions can also be used in media processors and DSPs for very fast and cost-effective multimedia processing.

1. INTRODUCTION

The IA-64 instruction-set architecture (ISA) includes a rich set of multimedia instructions to accelerate the processing of different forms of multimedia data [1]. These instructions implement the ISA concept of *subword parallelism* [2,3], also called *packed parallelism* [4] or *microSIMD parallelism* [5]. In microSIMD parallelism, data is partitioned into smaller units called *subwords*. In IA-64, a subword can be 8, 16 or 32-bits long, thus, multiple subwords can be accommodated in a single register, which is 64-bits long.

Packed subwords permit faster multimedia processing because of two main reasons. Multimedia data is typically low-precision data, such as 8-bit pixels in a video application, or 16-bit samples in an audio application [2,3,4]. Representation, storage and processing of these data is more efficient when microSIMD parallelism is used, since multiple low-precision data elements can be loaded into a single register, and processed in a single cycle. Secondly, multimedia applications have a great deal of data parallelism, and microSIMD parallelism exploits this by allowing parallel processing of subwords, using the standard word-oriented registers, datapaths, and pipeline control in microprocessors. A `packed add` instruction on 8-bit subwords performs eight subword add operations on the two 64-bit source registers in a single cycle, with only slight modifications needed to the 64-bit ALU (Figure 1).

MicroSIMD parallelism can be extended to include floating-point (FP) data [1,6,11]. A 64-bit FP register can be partitioned to include two single-precision FP numbers. Packed FP instructions can operate on this data similar to packed integer operations. This is useful for multimedia applications that use FP data intensively, such as graphics geometry processing and high-fidelity audio [7].

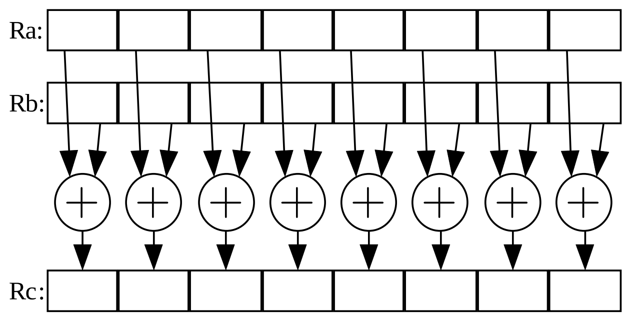

Figure 1. `Packed add` instruction on 8-bit subwords.

Section 2 describes the integer and FP multimedia instructions in Intel IA-64 architecture. Section 3 illustrates the performance potential of these multimedia instructions with some typical examples. Section 4 concludes the paper.

2. IA-64 MULTIMEDIA INSTRUCTIONS

IA-64 multimedia is both a subset and superset of the MAX-1 [2] and MAX-2 [3] multimedia instructions in the PA-RISC processors from Hewlett-Packard [8] and the MMX [4], SSE-1 and SSE-2 from Intel [6,9,11]. In chronological development, IA-64 multimedia instructions were defined after MAX-1, MAX-2 and MMX, around the same time as SSE-1 and before SSE-2. Comprehensive survey of first generation multimedia ISAs is given in [10], and of second generation ones, including Altivec and 3DNow! in [11].

2.1 Packed Integer Arithmetic Instructions

Table 2 includes a summary of IA-64 packed integer arithmetic instructions, describing the mathematical operations performed. `Packed add` and `packed subtract` instructions operate with either saturation arithmetic or modular arithmetic. `Packed average` and `packed negated average` are especially useful in multimedia applications involving images and video, such as MPEG [12]. `SAD`, which stands for "Sum of Absolute Differences", is a multi-cycle operation (usually pipelined) for accelerating motion estimation in MPEG video compression. `Packed compare` instructions write a bit string of 1's to the target register for *true* comparison results, and a bit string of 0's for *false* results. `Packed maximum` and `packed minimum` instructions write the greater and smaller of the source subwords to the target register respectively.

`Packed multiply` instructions have different variants because subword-wise multiplication produces intermediate products twice the size of the target register. Thus, only half of the bits of the intermediate products can be written to the target register. `Packed multiply high` instruction chooses the high order bits of the intermediate products, whereas `packed multiply low` chooses the low order bits of the intermediate products. `Packed multiply shift` permits a right shift of the intermediate products before writing the low order bits to the target register. `Packed multiply odd` and `packed multiply even` only multiply the odd or even indexed subwords of the source registers. This allows the full product to be written to the target register.

Other packed integer instructions are `packed shift` and `packed shift and add` instructions. The shift amounts can be specified either as a constant in an immediate field (*n*) or as a variable in a register (*b*).

2.2 Packed FP Arithmetic Instructions

Table 3 summarizes IA-64 packed FP instructions. In `the packed FP multiply` of two single-precision (SP) operands, each product has the same number of bits as the operands since it is also a single-precision FP number. This is simpler than integer multiply, where the product is twice as long in number of bits.

The basic FP microSIMD instruction in IA-64 is the `packed FP multiply add` operation, often cited as the most frequently used operation in digital signal processing and geometry processing. IA-64 has three variants of this. In `packed FP multiply add` and `packed FP multiply subtract`, each pair of subwords in the first two source registers is multiplied together, then the corresponding subwords in the third source register are added to (or subtracted from) these products. A third variant, `packed FP multiply negate add`, performs `packed FP multiply add` then negates the result. These operations are used to achieve basic `packed FP add` or `packed FP subtract` (by using FR1 as one of the first two source registers), and `packed FP multiply` (by using FR0 as the third source register). When used as source registers in IA-64 instructions, FR1 gives a constant value of 1 and FR0 gives a constant value of 0.

2.3 Subword Permutation Instructions

Subword permutation instructions exist for both integers and FP numbers (Table 4). `Pack` instructions, which also perform saturation functions, are used to create smaller data types from larger ones (Figure 2a). `Unpack` instructions are inverse of `pack` instructions (Figure 2b). `Mix` instructions come in two variants and write alternating subwords from the two source registers to the target register (Figures 2c and 2d). Either `mix` or `unpack` instructions are very useful for performing matrix transpose of subwords packed into multiple registers. The `permute` instruction performs any arbitrary permutation of four 16-bit integer subwords, with or without repetition of any subword. It is called `mux2` in IA-64, where `2` refers to 2-byte subwords. The 5 variants of the `mux1` instruction (Figure 3) are carefully selected permutation primitives, new to IA-64. The `mix` and `pack` operations are also extended to packed FP instructions. Here a new `FP mix left right` version is introduced, which concatenates the left subword of the first source register with the right subword of the second source register. Since IA-64 packs only two SP subwords in its extended FP precision registers (82 bits), permuting the two subwords in a register reduces to swapping them, as in the `FP swap` instruction.

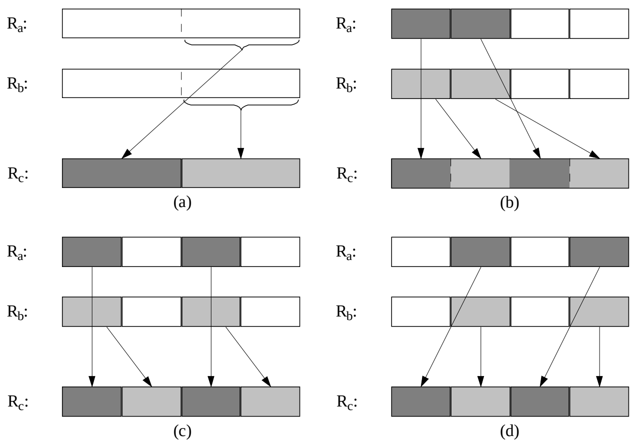

Figure 2. (a) `Pack` (b) `Unpack high. Unpack low` is symmetric. (c) `Mix left` (d) `Mix right`

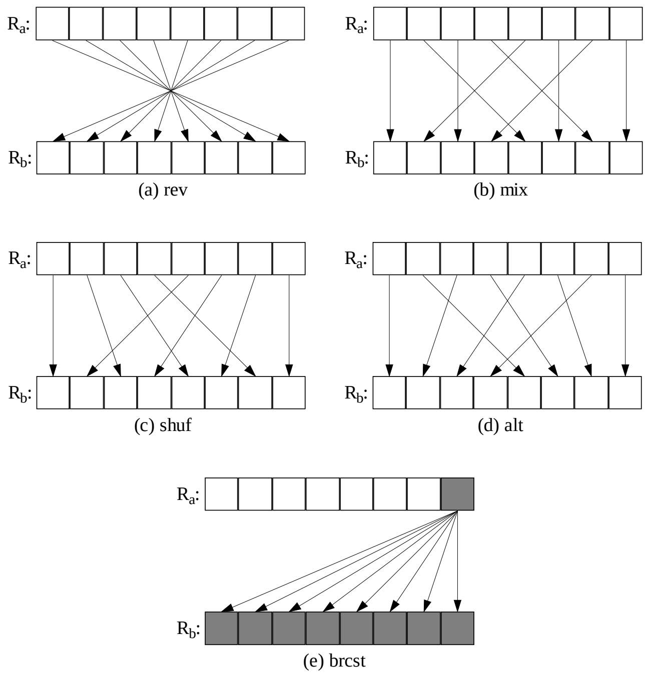

Figure 3. `Mux byte` instruction with its five variants.

3. EXAMPLES AND PERFORMANCE

3.1 Pixel Padding

Pixel padding is used to improve the accuracy of motion estimation and compensation routines in the MPEG-4 video compression. The data objects used in calculations are 8x8 macro-blocks, composed of 8-bit pixels. In our optimizations, IA-64 subword permutation instructions are used to create packed data, which are then efficiently manipulated by packed arithmetic instructions. Other common operations in pixel padding are matrix transposition and parallel averaging. Transposition is efficiently handled by the `mix` instruction [10] and parallel averaging is performed by the `packed average` instruction.

3.2 Shape Adaptive DCT

Shape Adaptive Discrete Cosine Transform (SA-DCT) is another commonly used sub-routine in the MPEG-4 algorithm. In SA-DCT, Discrete Cosine Transform is applied to arbitrary shaped areas on the video object plane, as opposed to the rectangular shaped frames as in MPEG-2. SA-DCT involves organizing pixels using vertical shifts to make them suitable for a vertical DCT pass. After the vertical DCT, pixels are re-organized using horizontal shifts, and a final horizontal DCT is applied. IA-64 subword permutation instructions are used to prepare the arbitrary shaped areas for subsequent DCT passes. In addition, the DCT computation itself is also optimized. These computations normally involve multiplication of pixels by fractional integer constants such as 1.4140625, which is an approximation to $\sqrt{2}$. Fractional integer multiplies are optimized using `packed shift right add` instructions [10].

3.3 Split Radix FFT

Fast Fourier Transform (FFT) is a DSP function that is widely used in DSP chips as well as in general-purpose processors for applications such as audio decoding in MPEG video. We focus on the Split Radix FFT (SR-FFT) which is an efficient FFT implementation requiring relatively fewer FP multiplications. Our optimizations are used on 16-element SR-FFT that operates on single-precision FP numbers. Such precision is necessary to produce the high-fidelity audio output. First, SP data is packed into IA-64 FP registers to allow using packed FP instructions. Additions on the complex plane are performed using `packed FP add`. Multiplication by the imaginary constant *j* is realized using the `FP swap` instruction. Complex multiplication is performed by using two `packed FP multiply and add` instructions, and one `FP swap` instruction.

Simulation results are presented in Table 1 for four different cases. In the **unoptimized** case, the Reference C code for the algorithm is compiled with gcc without using any compiler optimizations. In the **basic optimizations** case, SGI compiler is used with simple compiler optimizations such as loop unrolling and instruction rescheduling. In the **manually optimized** setting, assembly code is hand coded to employ multimedia instructions for subroutines such as matrix transposition or packed averaging. To simulate code for these three cases, we use Hewlett-Packard's IA-64 simulator, SKI. Since SKI is an instruction-level simulator, these results reflect the use of an infinitely wide machine with unlimited resources. We report performance data for a more restricted machine in our fourth simulation case, shown as **manually optimized 4G2M**. Results for this case are for a hypothetical IA-64 implementation with four general functional units (4G) and two memory ports (2M). Instructions are issued in superscalar fashion, rather than in bundles of three as in IA-64 processors like Itanium. Integer and FP multiply instructions are assumed to have a latency of three cycles.

TABLE 1. Pathlength (PL), execution cycles (CYC), instructions per cycle (IPC) and speedup (S).

Pixel Padding	**PL**	**CYC**	**IPC**	**S**
Unoptimized (gcc)	49716	31431	1.58	1
Basic optimizations (SGI)	11697	2896	4.04	10.9
Manually optimized	360	82	4.39	383.3
Manually optimized 4G2M	360	107	3.36	293.7
4-Point SA-DCT	**PL**	**CYC**	**IPC**	**S**
Unoptimized (gcc)	320	116	2.76	1
Basic optimizations (SGI)	173	97	1.78	1.2
Manually optimized	21	9	2.33	12.8
Manually optimized 4G2M	21	13	1.62	8.9
Split Radix FFT	**PL**	**CYC**	**IPC**	**S**
Unoptimized (gcc)	1256	435	1.95	1
Basic optimizations (SGI)	175	134	1.31	3.2
Manually optimized	106	16	6.63	27.2
Manually optimized 4G2M	106	45	2.36	9.6

4. CONCLUSION

We have described the multimedia instructions in the Intel IA-64 architecture under three broad classes: packed integer arithmetic instructions, packed FP arithmetic instructions, and subword permutation instructions. Multimedia information processing has been shown to benefit from all these three instruction classes. For instance, applications that involve processing image files like bitmaps or JPEG, primarily operate on low-precision integer data. For such applications, integer microSIMD extensions prove most useful. On the other hand, applications involving audio or graphics operate on single-precision FP data, which require FP microSIMD instructions. Subword permutation instructions are useful whenever data needs to be packed, unpacked or reorganized, such as in matrix transposition. In order to illustrate the benefits of microSIMD parallelism, we considered three common multimedia algorithms, and optimized them to use IA-64 multimedia instructions. Our simulations demonstrate the impressive speedups that can be obtained through using microSIMD extensions in multimedia applications.

TABLE 2. Packed integer arithmetic instructions in IA-64.

Integer Instruction	**Description**[1]
`P.add`	$d_i = a_i + b_i$
`P.add w/ sat.`	$d_i = a_i + b_i$
`P.subtract`	$d_i = a_i - b_i$
`P.subtract w/ sat.`	$d_i = a_i - b_i$
`P.average`	$d_i = avg(a_i, b_i)$
`P.negated average`	$d_i = -avg(a_i, b_i)$
`P.compare`	$d_i = compare(a_i, b_i)$
`P.maximum`	$d_i = \max(a_i, b_i)$
`P.minimum`	$d_i = \min(a_i, b_i)$
`SAD`	$d = \sum \lvert a_i - b_i \rvert$
`P.multiply low`	$d_i = (a_i * b_i)_{low}$
`P.multiply high`	$d_i = (a_i * b_i)_{high}$
`P.multiply shift`	$d_i = [(a_i * b_i) >> n]_{low}$
`P.multiply even`	$[d_{2i}, d_{2i+1}] = a_{2i} * b_{2i}$
`P.multiply odd`	$[d_{2i}, d_{2i+1}] = a_{2i+1} * b_{2i+1}$
`P.shift left add`	$d_i = (a_i << n) + b_i$
`P.shift right add`	$d_i = (a_i >> n) + b_i$
`P.shift left`	$d_i = a_i << n$
`P.shift left var.`	$d_i = a_i << b$
`P.shift right`	$d_i = a_i >> n$
`P.shift right var.`	$d_i = a_i >> b$

TABLE 3. Packed FP arithmetic instructions in IA-64.

FP Instruction	**Description**[1]
`P.multiply add`	$d_i = a_i * b_i + c_i$
`P.multiply subtract`	$d_i = a_i * b_i - c_i$
`P.multiply negate add`	$d_i = -a_i * b_i + c_i$
`P.negate`	$d_i = -a_i$
`P.absolute value`	$d_i = \lvert a_i \rvert$
`P.negated absolute value`	$d_i = -\lvert a_i \rvert$
`P.compare`	$d_i = compare(a_i, b_i)$
`P.maximum`	$d_i = \max(a_i, b_i)$
`P.minimum`	$d_i = \min(a_i, b_i)$
`P.maximum of absolute values`	$d_i = \max(\lvert a_i \rvert, \lvert b_i \rvert)$
`P.minimum of absolute values`	$d_i = \min(\lvert a_i \rvert, \lvert b_i \rvert)$
`P.reciprocal square root approximate`	$d_i = 1 / \sqrt{a_i}$
`P.reciprocal approximate`	$d_i = 1 / a_i$

[1] In Tables 2 and 3, a_i, b_i and c_i denote subwords in the source registers, while d_i denote subwords in the target register.

TABLE 4. Subword permutation instructions in IA-64.

Instruction
`Mux2 or Permute n subwords`
`Mux1.rev`
`Mux1.mix`
`Mux1.shuffle`
`Mux1.alt`
`Mux1.broadcast`
`Mix {left, right}`
`Unpack {high, low}`
`Pack`
`FP mix {left, right, left_right}`
`FP swap`
`FP pack`

5. REFERENCES

[1] Intel, "IA-64 Architecture Software Developer's Manual, Vol. 3: ISA Reference," Rev 1.1, July 2000, ID# 245319-002.

[2] R.B. Lee, "Accelerating Multimedia with enhanced Microprocessors," *IEEE Micro*, Vol. 15, No. 2, April 1995, pp. 22-32.

[3] R.B. Lee, "Subword Parallelism with MAX-2," *IEEE Micro*, Vol. 16, No. 4, August 1996, pp. 51-59.

[4] A. Peleg, U. Weiser, "MMX Technology Extension to the Intel Architecture," *IEEE Micro*, Vol. 16, No. 4, August 1996, pp. 42-50.

[5] R.B. Lee, "Efficiency of MicroSIMD Architectures and Index-Mapped Data for Media Processors," *Proceedings of. IS&T/SPIE Symposium on Electric Imaging: Media Processors 99*, January 25-29, 1999, pp. 34-46.

[6] Intel, "IA-32 Intel Architecture Software Developer's Manual", Volume 2: ISA Reference," 2000.

[7] R.B. Lee and M. Smith, "Media Processing: A New Design Target," *IEEE Micro*, Vol. 16, No. 4, August 1996, pp. 6-9.

[8] G. Kane, "PA-RISC 2.0 Architecture," 1996, Prentice Hall, ISBN 0-13-182734-0.

[9] Intel, "Intel Architecture Software Developer's Manual, Volume 2: ISA Reference," 1999, Order Code 243191.

[10] R.B. Lee, "Multimedia Extensions for General-Purpose Processors," *Proc. IEEE SIPS 97*, Nov. 1997, pp. 9-23.

[11] R.B. Lee, A.M. Fiskiran, "Multimedia Instructions in Microprocessors for Native Signal Processing", invited book chapter in press.

[12] V. Bhaskaran, K. Konstantinides, R.B. Lee and J.P. Beck, "Algorithmic and Architectural Enhancements for Real-Time MPEG-1 Decoding on a General Purpose RISC Workstation," *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 5, No. 5, Oct. 1995, pp. 380-386.