Computational Requirements for Media Signal Processing

Vojin G. Oklobdzija

Advanced Computer System Engineering Laboratory
Electrical and Computer Engineering Department
University of California
Davis, CA 95616
(510) 486-8171
vojin@ece.ucdavis.edu
http://www.ece.ucdavis.edu/acsel

Abstract

Media signal processing is characterized by high computing power and a great deal of parallelism on low precision data, which usually consists of simple integers with 8, 12, or 16 bits of precision. Efficient processing of media signals requires modifications of the traditional data-path found in general purpose processors. In this paper, we address those differences and propose adequate solutions for efficient implementation of data-path that support main classes of media instructions.

1. INTRODUCTION

Media signal processing involves real time processing of audio and video signals. .It has been used increasingly in consumer electronic products, such as personal digital assistants, cellular phones, video games, digital cameras, and high-end communication products, IP-telephony gateways, multi-channel modems, speech-processing systems, echo cancellers, image and video processing systems, internet routers, and virtual private network servers. Typically multimedia communication involves the transfer of large amounts of data so that data compression is essential for a cost-efficient use of existing communication channels and storage media[1]. To achieve real-time processing of media signals architectural extensions are necessary in order to alleviate the pressure for performance exhorted on the system and technology [3-10]. All of those extensions operate on the multiple-data values under the control of a single instruction. In most of those processors data is packed in 64-bit registers in one of the general-purpose register files reflecting their 64-bit adherence to the 64-bit architecture world [9]. The goal of media extensions is to realize the full potential of the SIMD approach as applied to multimedia applications [2]. Various studies of medial algorithms showed that a departure from the 64-bit data-path width is necessary and that 128, even 256 bit data-path widths would be more adequate [3].

2. REQUIREMENTS FOR PROCESSING OF MEDIA SIGNALS

All real-time signal processing algorithms such as, coding/decoding, and compression / decompression of audio and video signals, are based around certain computation demanding functions such as the Discrete Cosine Transform (DCT), the Inverse Discrete Cosine Transform (IDCT), the Fast Fourier Transform (FFT). These functions are computationally intensive so that special dedicated VLSI circuits or Digital Signal Processors (DSPs) are used in conjunction with the main processor [1-2]. Conventional DSP architectures lack the computing power and bandwidth to perform more than one multimedia task at a time. On the other hand, general-purpose microprocessors support media processing by introducing SIMD multimedia enhancements in their instruction sets [4-10]. In these processors a long-word ALU is divided into several small-word ALUs, and several pieces of independent short-word data are processed by a single instruction to perform SIMD operation.

2.1 Word Length and Data Formats

To evaluate performance gains of the multimedia technology compared to the scalar one and the effectiveness of extending widths of media paths from classic 64 bits to 128 or 256 bits we performed analysis on some typical media processing steps. The analyses use 16-bit data elements, so that four elements can be processed in parallel using operations on packed words in MMX-64, eight in EMP-128, and sixteen in EMP-256. We assume that all accesses to packed data are aligned. An instruction issue rate of one instruction/cycle, latency

0-7803-6514-3/00/$10.00©2000 IEEE

of 1 cycle for all non-load/store instructions, a cache line size of 512 bits, and a cache miss penalty of 8 cycles was assumed in this analysis. Under these assumptions, multimedia architectures can speedup the execution of the motion compensation from 2.3 to 7.2 times depending on widths of media paths and types of video frames as illustrated in Fig.1. and Fig.2.

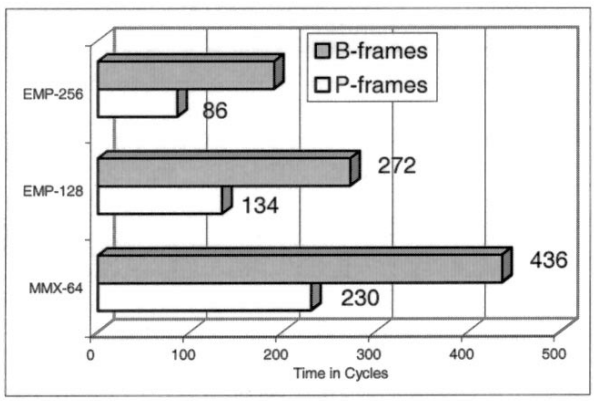

Fig. 1. Execution times of the motion compensation of an 8x8 blocks from P- and B- frames

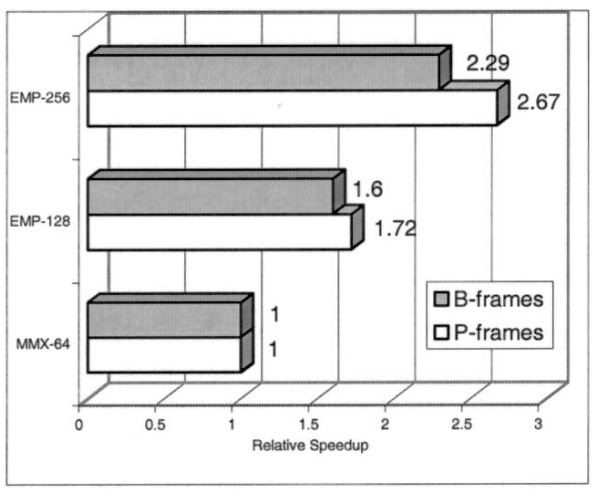

Fig. 2. Relative performance of multimedia processors in the motion compensation of an 8x8 blocks from P- and B-frames

If we assume that a linear approximation of these results down to scalar implementations is valid, we can estimate that the MMX-64 implementation must be about 2.3 and 2.7 times faster than the corresponding scalar, non-MMX, implementation of the motion compensation for B- and P-frames, respectively.

2.2 Data Types and Sub-Word Parallelism

Sub-word parallelism is a highly effective technique in block-level processing where complete data blocks are processed in a uniform manner. For these parts of the algorithm an almost linear speed-up is achieved. The concept of sub-word parallelism is almost universally employed in many state-of-the-art media processors and programmable architectures. A split-ALU implementation is required that supports processing on a variety of different sub-word sizes.

New data types needed for media processing, assuming 128-bit media word, contain packed fixed-point integers. The decimal point for the fixed-point values is assumed to be to the right of the least significant bit (LSB) for the fixed-point integers. When the fixed-point fractions are assumed, decimal point is assumed to be to the right of the MSB (Q-format).
The new media data types are shown in Fig.3.

1. Packed byte: 16 bytes packed into a 128-bit quantity
2. Packed half-words: 8 half-words (16-bit) packed into a 128-bit quantity
3. Packed words: 4 (32-bit) words packed into a 128-bit quantity
4. Packed double-words: two 64-bit double-words packed into a 128-bit quantity

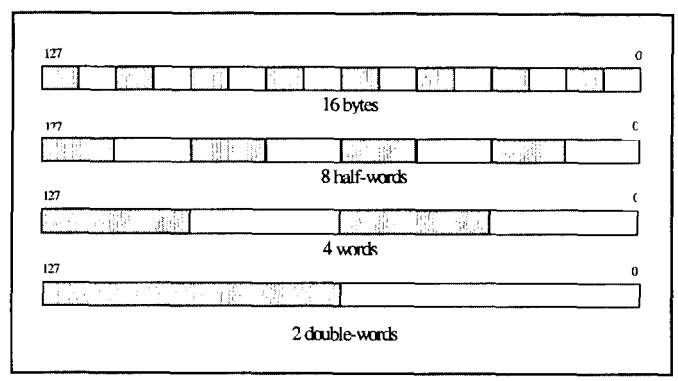

Fig. 3. Packed Data Types

When dealing with sub-words, a PERM instruction allowing arbitrary permutations of two source operands into a target operand is highly useful. This operation may be used for broadcasting one data item to all elements of a vector register, for aligning data after memory access (e.g., in motion compensation), and for extended shift operations. An example of an arbitrary permute operation performed under control of a special permute mask register is shown in Fig.4.

SIMD-style processing assumes the same operation to be applied uniformly to all data items. This limitation is removed in media processors by introduction of field-wise conditional execution, as shown in Fig. 5 by the example of a conditional move instruction. The result of an operation specified for the complete vector register

depends on condition states that have been set individually for each data field by previous operations [2].

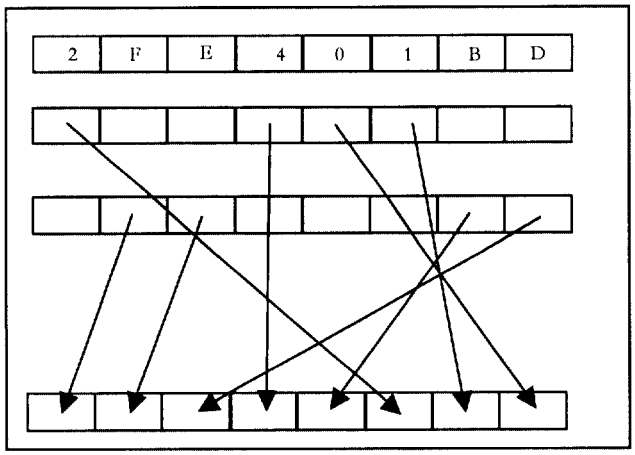

Fig.4. Permute operation

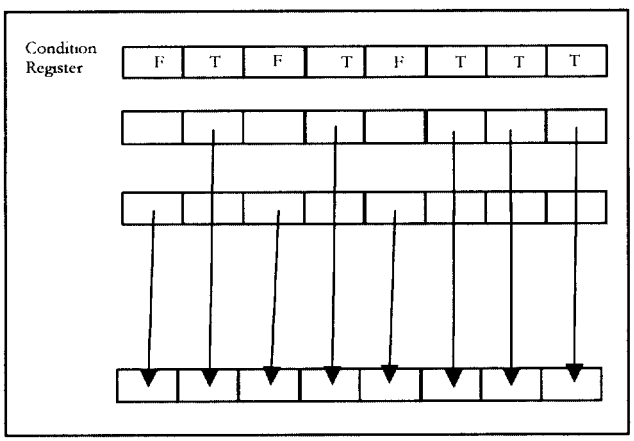

Fig.5. Conditional Move operation

3. OPERATIONS SUPPORTING MEDIA SIGNAL PROCESSING

3.1 MPEG Decoding

MPEG decoding requires the following six basic steps: (a) Header Decode, (b) Huffman and Run-length decode, (c) Inverse Quantization, (d) Inverse Discrete Cosine transform, (e) Motion compensation, (f) YUB to RGB conversion.

The major arithmetic operations in MPEG decoding are addition, and multiplication thus, in order to speed-up the IDCT, motion compensation, and color conversion processing, new instructions are needed to support these operations in multiple data formats efficiently.

3.2 Support for DCT / IDCT

The two-dimensional DCT/IDCT, heavily relies on continued accumulation of multiplication results as the core operation. Obviously an efficient support for MAC (multiply-accumulate) instruction is important.

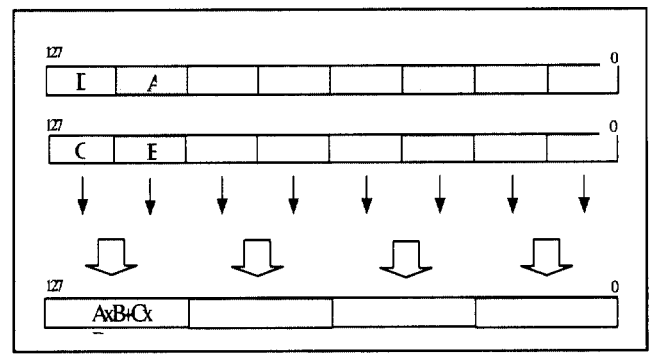

Fig. 6. Parallel MAC operation

MAC operation, when using sub-word parallelism produces the result, which becomes the double of the input operand length when using full multiplication and increases further with continued accumulations. The results have either to be truncated during the MAC operation, or the sub-word parallelism of subsequent operations will be cut by half when allocating double-sized sub-words for the result data. This is the issue of accuracy requirements, as specified for the IDCT, which requires computations to be performed with a specific minimum word length and shows sensitivity to rounding of intermediate results. MAC operation can be defined to perform rounding operation or a MAC instruction preserving the full word length of the results can be complemented with an instruction (PACK) that converts full word length into a sub-word thus combining a round facility and saturation. The best use of such instruction is when two media words are combined into one consisting of sub-word data (PACKD), Fig. 5. This instruction can be used to perform format conversions of intermediate and final results while complying to the accuracy requirements [1-2].

3.3 Support for Motion Compensation

Motion compensation involves, memory accesses to the macro-block pointed to by motion vectors. Memory accesses are unaligned in the general case, thus we need to align register contents prior to macro-block reconstruction.

Pixel interpolation operations become necessary for each pixel of a macro-block, amounting to a significant number of arithmetic operations. This can effectively be supported by an AVG instruction that computes the average of two

source operands by calculating their sum followed by a right-shift. Rounding may also be employed as defined by MPEG-4.

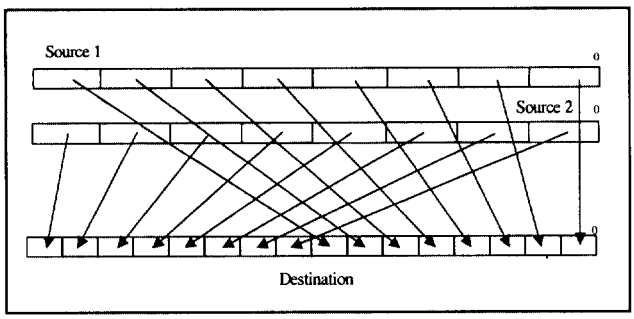

Fig.7. PACK Double operation

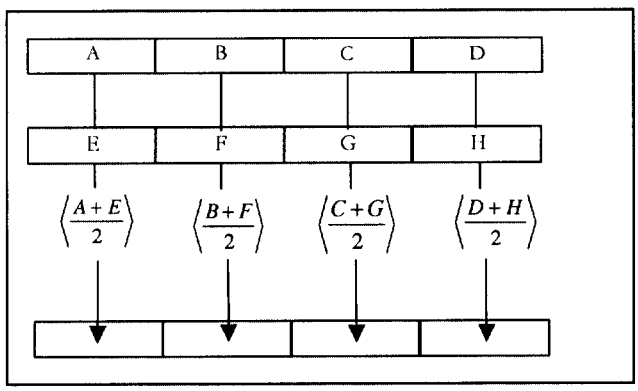

Fig.8. Average operation

Sum of Absolute Differences: SAD operation is important is comparing pixels of an image. SAD operation is shown in Fig. 9.

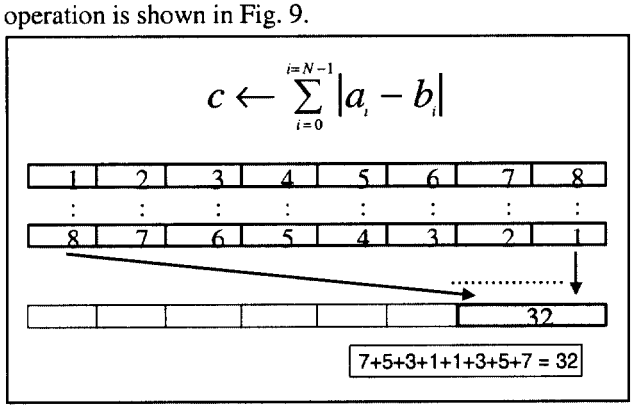

Fig. 9. SAD instruction supporting pixel error calculation

Implementation of SAD operation can be very complex in the traditional data-path. Efficient implementation can be achieved by using partial product reduction tree of a media multiplier.

3.4 Support for Inverse Quantisation

An operation sequence used in the inverse quantisation is the alternative addition/subtraction of a constant value, which is dependent on the sign of the input data. An ADDSUB instruction, conditionally executed, helps to implement this operation efficiently. This operation is be executed in parallel for all sub-words of a vector register with conditions individually set for each sub-word.

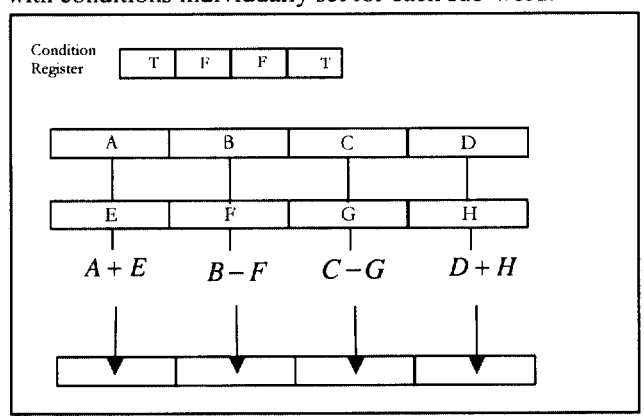

Fig.10. Conditional ADD/SUB operation

3.5 Bit Stream Parsing

Effective reordering of the incoming bit-stream, requires instruction that supports bit reversal. The value of the source register is forwarded to the destination register in the reverse order in one cycle as shown in Fig.10.

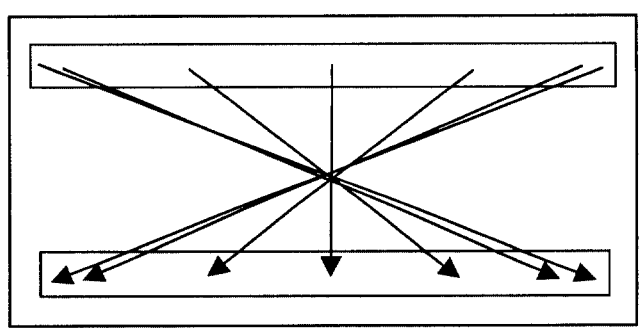

Fig.10. Bit Reversal operation

4. DATA-PATH IMPLEMENTATION

A programmable data-path that capable of efficient processing of media signal is presented in [12]. Media data-path is built to support a maximum of 16 parallel SIMD integer operations as shown in Fig.11. These operations are implemented by re-using the hardware without significant increase in area and delay. This data-path has been modeled in Verilog using 0.25u CMOS library and synthesized using Synopsys. It is capable of supporting media-operations running at 200 MHz.

5. CONCLUSION

Signal processing algorithms found in multimedia applications have demanding computational requirements. The increased functional complexity of basic processor instructions results in a significant speedup however, it requires flexible and complex data path solutions at little silicon cost.

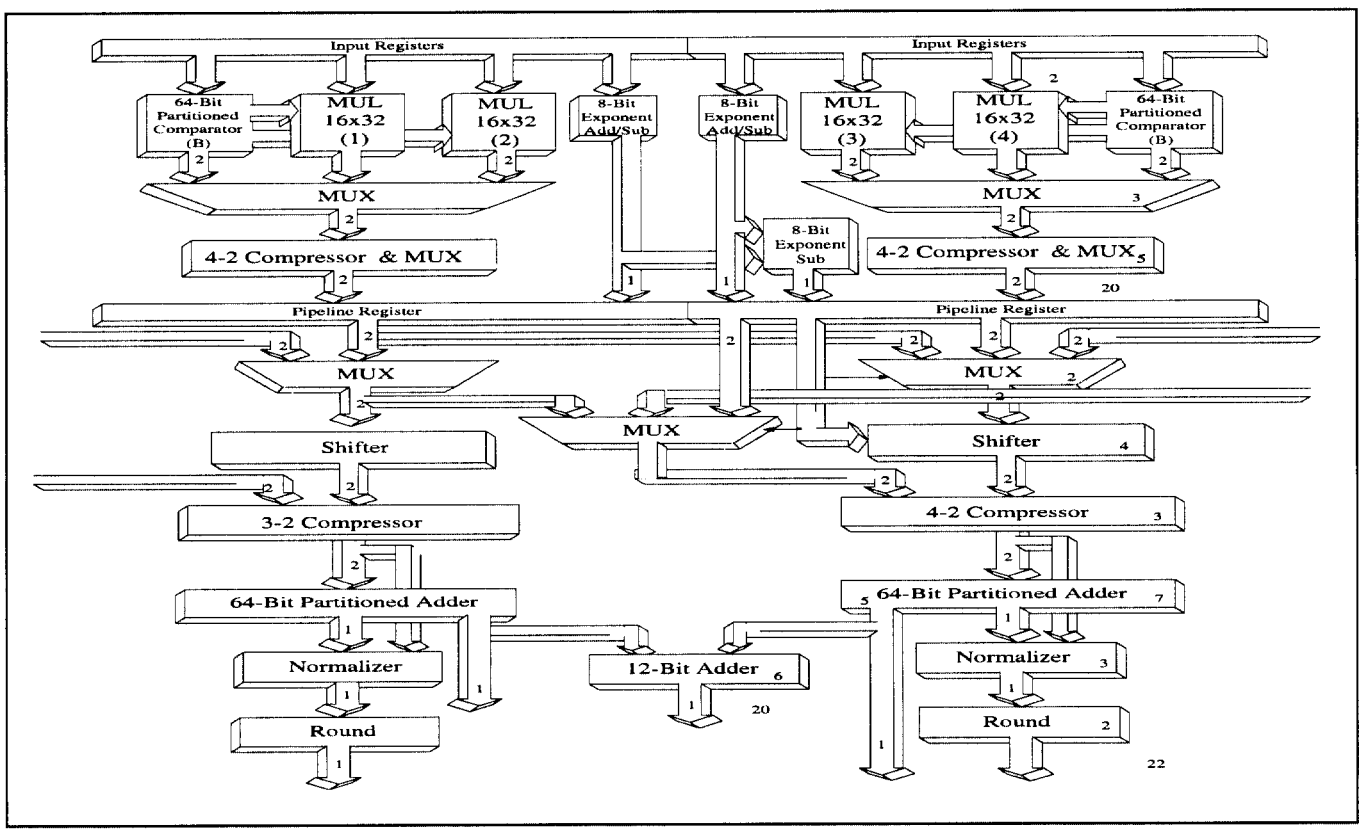

Fig.11. Media Data-Path organization

REFERENCES

1. I. Kuroda and T. Nishitani, "Multimedia processors," Proc. IEEE, Vol. 86, No. 6, pp. 1203-1221, 1998.
2. M. Berekovic, H-J. Stolberg, M.B. Kulaczewski, P. Pirsch, H. Moller, H. Runge, J. Kneip, B. Stabernack, "Instruction Set Extensions for MPEG-4 Video," Journal of VLSI Signal Processing Systems, Vol. 23, No. 1, pp. 27-49, October 1999.
3. Schmookler, M.S., et. al., "A low-power, high-speed implementation of a PowerPC microprocessor vector extension", Proceedings 14th IEEE Symp. on Comp. Arith., 1999 , Page(s): 12 -19.
4. Lee. R., "Accelerating multimedia with enhanced microprocessor", IEEE Micro vol.15, No. 2, pp. 22, April 1995.
5. M. Tremblay, J. M. O'Connor, V. Narayanan, and L. He, "VIS speeds new media processing", IEEE Micro Mag., vol. 16, pp. 10-20, Aug. 1996.
6. M. Tremblay, D. Greenley, and K. Normoyle, "The design of the microarchitecture of UltraSparc TM-I", Proc. IEEE, vol. 83, pp. 1653-1663, Dec. 1995.
7. S. Rathnam and G. Slavenburg, "An architectural overview of the programmable multimedia processor, TM-1", in Proceedings of Compcon. New York: IEEE Computer Sc. Press, 1996, pp. 319-326.
8. R. B. Lee, "Subword parallelism with MAX-2", IEEE Micro Mag., vol. 16, pp. 51-59, Aug. 1996.
9. Peleg, A. and Weiser, U.,"MMX technology extension to the Intel architecture", IEEE Micro Mag., vol. 16, pp. 42-50, Aug. 1996.
10. L. Gwennap, "Digital, MIPS add multimedia extensions", Microprocessor Rep., vol. 10, no. 15, pp. 24-28, Nov. 1996.
11. A. Farooqui, V.G. Oklobdzija, "Impact of Architecture Extensions for Media Signal Processing on Data-Path Organization", Proceedings of the 34th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, October 29-31,2000.
12. A. Farooqui, V.G. Oklobdzija, "A Programmable Data-Path for MPEG-4 and Natural Hybrid Video Coding", Proceedings of the 34th Asilomar Conference on Signals, Systems and Computers, Pacific Grove, October 29-31,2000.