A PROGRAMMABLE, HIGH PERFORMANCE VECTOR ARRAY UNIT USED FOR REAL-TIME MOTION ESTIMATION

Nikos Bellas *Malcolm Dwyer*

Multimedia Architectures Lab, Motorola Inc., Schaumburg, IL

ABSTRACT

The MPEG-4 and H.263 video standards are enabling technologies for the proliferation of wireless multimedia applications in 3G systems. For video encoding, the Motion Estimation (ME) stage is typically the most demanding in terms of performance and bandwidth requirements, and is usually implemented through dedicated hardware, especially in systems with stringent power requirements. This approach, however, cannot exploit any algorithm advances on Motion Estimation algorithms, and requires major hardware re-design in case of modified specifications or standards. This paper describes the architecture of a programmable Motion Estimation unit that is used as part of a larger wireless video encoding system. An Instruction Set Architecture (ISA) allows the development of various ME algorithms in software without the need to re-design portion of the chip.

1. INTRODUCTION

There are a large number of different techniques to perform ME in a video encoding system, since this process is not part of the MPEG standard. Search window size and shape, comparison computation, convergence criteria, pixel accuracy, and block size are some of the parameters that vary between different motion estimation algorithms, and make a programmable solution desirable.

The search window is the rectangular area of the previous frame within which the search for the best matching block takes place. All the ME algorithms constrain their search for the best match in a rectangular area around the current macroblock. For videoconferencing applications, this constraint does not create compression inefficiencies because the amount of motion between two successive frames is usually very small. The value p defines the search window size, where the search window extends p pixels to the top, bottom, left, and right of the current macroblock's location (Figure 1). For a displacement vector (i,j), the distortion between the two 16x16 macroblocks is defined by the following equation:

$$\text{SoAD}\ (i,j) = \sum_{m=0}^{15}\sum_{n=0}^{15} \left|\text{Curr}\ (m,n) - \text{Prev}\ (m+i, n+j)\right|$$
$$i, j \in [-p, p]$$

The proliferation of video capturing wireless devices like video cameras, PDAs, cell-phones, and digital cameras and the shrinking time to market present two conflicting goals to the designers of such systems. The demand for high MIPS/W has to be traded-off with the high cost of developing new ASICs every time the performance requirements go up, or the algorithms change. A design that is optimized for ME, yet it is programmable is a good compromise between the two worlds.

The ME module consists of a vector array optimized to perform fast SoAD computations, and half-pixel interpolations which are the building blocks of most ME algorithms. It is also equipped with a scalar pipeline to perform scalar arithmetic, and control operations as well as to control the functionality of the vector unit. The architecture follows a VLIW mechanism and up to three instructions can be issued per cycle.

The rest of the paper is organized as follows: Section 2 outlines previous work on the subject of ME algorithms and hardware implementations. Section 3 presents the architectural details of the proposed vector unit and Section 4 gives details on the ISA that is implemented by the vector array and explains how it can be used to perform various ME algorithms. Section 5 gives some technology implementation and performance results, and Section 6 summarizes the paper.

2. RELATED WORK

A large number of papers have focused on algorithms that make trade-offs between computational complexity and compression quality ([1], [2], [3]). There has also been a considerable amount of work on the hardware implementation of ME modules. In [4][5], two data paths to perform sum of absolute differences and pixel interpolation for half and quarter pixel interpolation are described. No general-purpose control code can be executed, and the modules are not programmable. The paper in [6] uses a formal methodology for mapping the full-search ME algorithm into systolic arrays, and the paper in [7] derives systolic architectures for the three-step hierarchical search algorithm. Extensive research has focused on low-power implementations of ME modules [8]. New multimedia-oriented instructions have been added to processors to speed up applications that require fast processing of large data sets. The Intel's MMX and Motorola's Altivec technologies are among the most well known for the desktop processing [9][10][11]. The same trend has pushed for SIMD extensions on embedded processors like ARM [12]. Finally, the push towards data-intensive multimedia workloads have pushed designers towards high performance media processors such as Equator's MAP-CA, BOPS's Manta, Philips's Trimedia, etc.

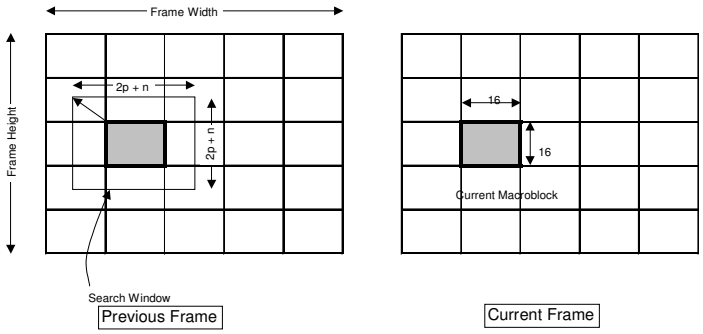

Figure 1 Motion Estimation Search Window

These approaches do not address the issue of programmable ME. They focus either on implementing a particular algorithm, or on just providing extensions to instructions that can be used for ME software development. Our approach can be termed as a programmable ME accelerator unit.

3. MOTION ESTIMATION ARCHITECTURE

Figure 2 shows the top-level diagram of the vector unit. The data path of the ME unit is composed of a systolic/vector array which is primarily used for the computation of the Sum of Absolute differences (SoAD) and a scalar part which is equipped with register files and scalar functional units. This part is mainly used for simpler scalar computations and program flow. The systolic array consists of 16 Processing Elements (PEs). The programmer controls the functionality of the memory and the systolic array through bitmasks that can be manipulated using the instructions of the ME module.

The machine is organized as a three-stage pipeline with Instruction Fetch (IF), Instructions Decode (ID), and Execution and Write Back (EX). The EX stage is where the vector array and the ALUs are used to compute results and write them back to the register file or to the PEs. Since there are only three stages, an instruction that reads a register operand can be issued immediately before the instruction that produced the operand. The only exception to that is the conditional branch instruction which can be issued only two clock cycles after the conditional flags have been set (via the Cmp instruction). The vector array is also part of the EX stage, but it cannot function simultaneously with the ALUs of the scalar part. Two separate 16-bit register files are used as scratchpad memory, while some registers have a dedicated functionality. This functionality is mainly used to control the vector array by the instructions vec_Soad, vec_SoadHP, etc. described in the next section.

The memory subsystem has three input/output channels from which it can provide data to the vector array, and can receive up to three independent addresses to read data from. The memory system is working on two modes: the “pixel” mode in which the incoming addresses are the (x, y) coordinates of the desired pixel, and the “linear” mode in which the incoming addresses are absolute. The memory can be set to the appropriate mode via control registers in the register files. A crossbar switch mechanism provides different paths to link the memory to the PEs.

A DMA unit is used to automatically generate the addresses of the incoming pixels so that the programmer of the module does not do that explicitly. The DMA is programmable, and can generate addresses in different patterns, such as a linear scan of the memory, or a two-dimensional scan, etc. The functionality of the DMA is controlled through variables such as the initial memory address, the size of the stride, the size of the skip, etc. The result is that the DMA can generate a variety of useful access patterns, and offload the programmer from the tedious task of providing a new memory address every clock cycle.

A layer of logic between the actual SRAM modules and the crossbar switches called Virtual Memory Translation Unit (VMTU), implements the conversion between the 2-D (x, y) addresses from the DMA and the absolute addresses that the SRAMs can use. In case of a memory access outside the search window, the VMTU intercepts the access and replaces it with an access at the edge of the search window. This is useful when the target MB is close to the edge of the search window and some of the accesses will be outside.

The core of the PE is the |a-b| block, and the accumulator. The computation that is done in one clock cycle by the PE is given by the following equation:

$$r = r + |a - b|$$

The PEs are used to compute the sum of absolute differences between a series of current pixels, and a series of search window pixels. They are connected in a pipeline chain to allow for pixel re-use once the pixels have been read out of the memory. This lowers the memory bandwidth requirements and allows up to 16 SoAD computations/cycle once the pipeline fills up. At the end of a number of cycles, the accumulator will contain the sum of absolute differences between a current macroblock, and a search window macroblock

Each PE can also perform bilinear interpolation before the SoAD to facilitate half-pixel interpolated searches. Figure 3 shows the interpolated pixel for every case. The A,B,C, and D pixels are fed into the PEs and the interpolated pixels are computed before its SoAD with a pixel in the current MB is evaluated.

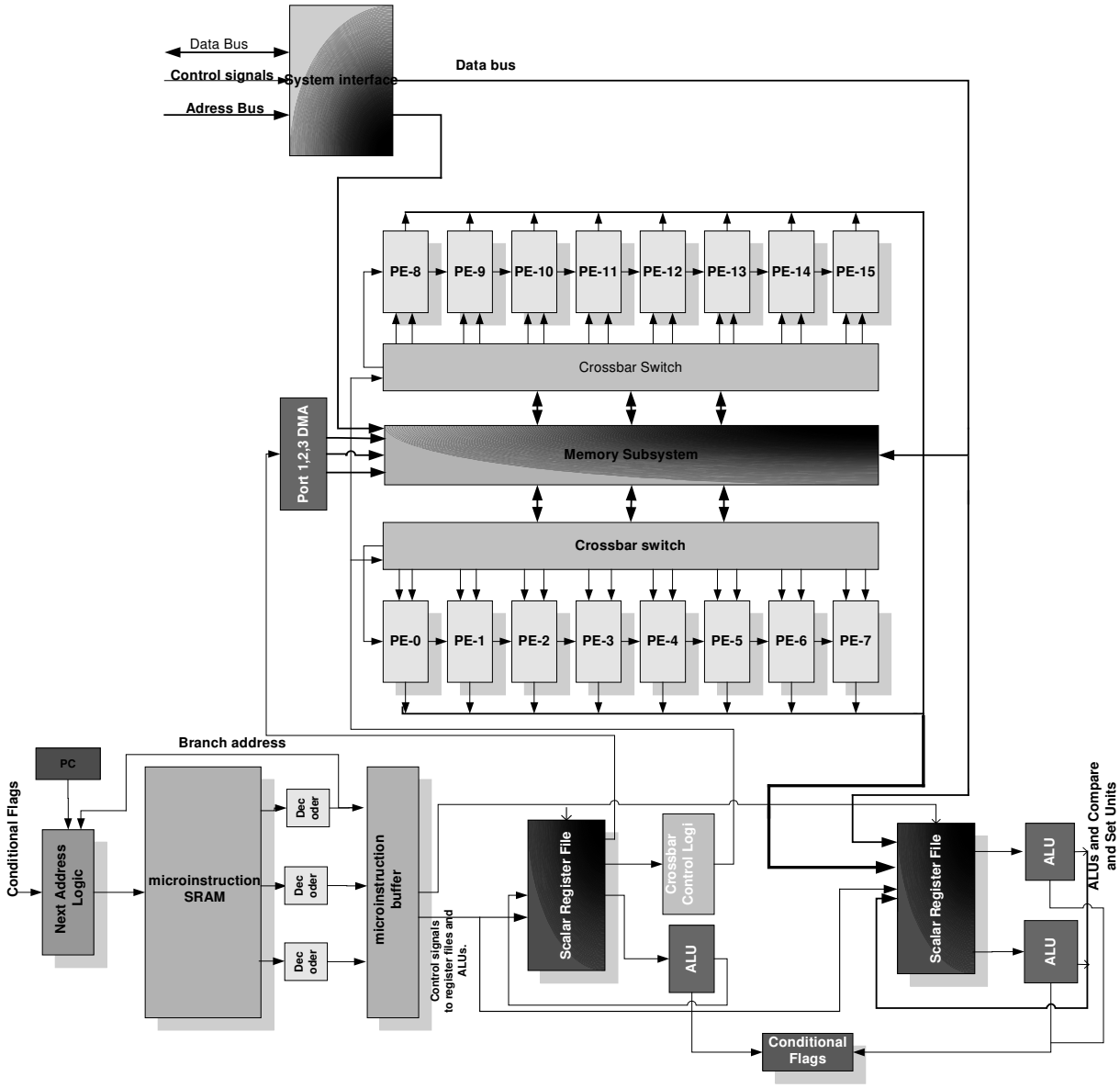

Figure 2 Block diagram of the programmable ME unit

4. INSTRUCTION SET ARCHITECTURE

The ME unit interprets and executes a well-defined ISA optimized for the computations typical in a ME algorithm. The following set of operations are supported :

- **Computational instructions**

These instructions perform the vector operations in the systolic array of the ME unit, as well as scalar operations. Typical computational instructions are:

- **Vec_Soad, Vec_SoadHP, Vec_Acc**

They are used to trigger a SoAD or SoAD with HP interpolation, or just pixel value accumulation on the vector array. They selectively enable/disable individual processing elements, and direct the output of the memory subsystem to particular PEs. They use specific registers (not part of the opcode because they are always the same) to set-up the crossbar switch before the ME execution loop is entered.

- **Add/AddC/Inc/Shift/Abs/AbsC/Min/MinC/Max/ MaxC/Cmp/CmpC**

These are standard scalar arithmetic operations. They support operations on bytes or words (16-bits). The Cmp/CmpC instructions are the only ones that can modify the conditional flags. The Add instruction is using a triadic addressing mode, and the AddC instruction is adding a 16-bit constant to a register $r and deposits the sum to $r. Similarly for all the other operations.

- **CondAdd $r1, const1, const2, const3**

It executes:

```
If ($r1 == const1)
    $r1 += const2;
else
    $r1 += const3;
```

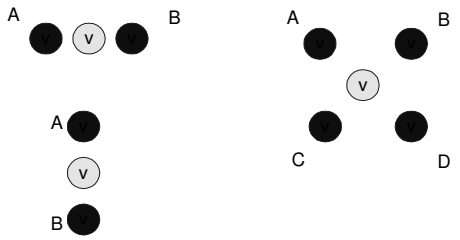

Figure 3 Half-pixel interpolation cases

It used to accelerate setting of pointers in the 2-D space of the search window.

- **Data transfer instructions**

They are used to transfer data between the register files, the memory and the accumulators of the PEs of the vector array.

- **Control flow instructions**

This set includes jump and return from function instructions as well as a BrMask conditional branch, which allows the PC of the microcontroller to jump to a new location depending on a number of different combinations of the conditional flags.

5. IMPLEMENTATION AND RESULTS

The ME unit was implemented as part of a larger MPEG-4 and JPEG compression chip. It is implemented at 0.18um CMOS technology, and is clocked at 100 MHz. Its area is 4.21 mm2. For this implementation, there is enough memory to store a 48x48 search window, which is enough to support a +-16 Motion Vector range, and also memory to store the 16x16 current MB.

The ME unit is able to support 30 fps CIF encoding at 100 MHz. The algorithm used is less computationally expensive than full search. Instead of computing the best match using all the 256 pixels of a macroblock, only 64 pixels are used by a vertical and horizontal sub-sampling of two. The steps of the algorithm are the following in summary:

1. A sub-sampled full-search algorithm in a range of +-16 full pixels is performed around the location (0,0) of the current MB. The pixels of the current MB and the target MB are partitioned in four classes: A, B, C, and D. For a particular target MB, only pixels from the same class are used to perform SoAD computations with the pixels of the same class in the current MB. The output of this stage are the four optimal motion vectors (MVs) and the SoADs for the four pixel classes (A, B, C, D), and the (0,0) location.
2. A full search algorithm is performed on the five locations from step 1, and the optimal integer MV is found.
3. Half pixel interpolation is performed around the optimal integer MV to compute the optimal half-pixel MV.
4. An extra step is taken to compute the average pixel value P of the current MB (using the vec_Acc instruction), and, then, the variation (SoAD) of every pixel in the current MB with respect to P. If the variation is smaller than a threshold, the current MB can be coded intra.

All these steps are carried out in the ME unit without any Host intervention other than to send the pixel data and receive the resulting MVs at the end. The performance of the ME unit is approximately 600 MSADs / sec

5. CONCLUSION

This paper describes a programmable architecture for fast ME execution. Various ME algorithms can be developed using an assembly programming interface. The vector array can be used for the computational intensive part of the code, while the scalar part is used for control and non-critical computations. This module eliminates unnecessary communication with the Host unit since all the steps of a potentially complex ME algorithm can be performed in place without the Host intervention.

11. REFERENCES

[1] K.Xie, L.V Eycken, and A. Oosterlinck. A new block-based motion estimation algorithm. *Signal Processing: Image Communication,* 4:507-517, May 1992.

[2] J.R Jain and A.K. Jain. Displacement measurement and its application in interframe coding. *IEEE Transactions on Communications,* 29(12):1799-1808, Dec. 1981.

[3] B. Liu and A. Zaccarin. New fast algorithms for the estimation of block motion vectors. *IEEE Transactions on Circuits and Systems,* 3(2):148-157, Apr. 1993

[4] J. Fandrianto, et al. Programmable architecture and methods for motion estimation. *US Patent 5,594,813*, February 1992

[5] S.C. Purcell, et al. Structure and method for motion estimation of a digital image by matching derived scores,. *US Patent 6,122,442*, August 1995

[6] T. Komarek and P. Pirsch. Array architectures for block matching algorithms. *IEEE Transactions on Circuits and Systems,* 36(10):1301-1308, October 1989

[7] Y-S. Jehng, L-G Chen, and T-D. Chiueh. A motion estimator for low bit-rate codec. *IEEE Transactions on Consumer Electronics,* 38(2):60-69, May 1992.

[8] Z. L. He, C.Y Chui, K. K. Chan, and M.L. Liu. Low-power VLSI design for motion estimation using adaptive pixel truncation. *IEEE Transactions on Circuits and Systems for Video Technology,* 10(5): 669-678, August 2000.

[9] Alex Pegel and Uri Weiser. MMX technology extension to the Intel architecture. *IEEE Micro*, 16(4): 42-50, August 1996

[10] Keith Diefendorf, et. al. Altivec extension to PowerPC accelerates multimedia processing. *IEEE Micro*, 20(2): 85-95, March 2000

[11] F. Moschetti and E. Debes. A fast block matching for SIMD processors using subsampling. *Proceedings of ISCAS,* vol. 4: 321-324, 2000

[12] S. Segars, et. al. First members of the ARM11 product family. *Microprocessor Forum*, October 2002.