Programmable Motion Estimation Architecture

Anargyros Drolapas, George Lentaris and Dionysios Reisis
Electronics Lab, Dpt of Physics, National and Kapodistrian University of Athens, Athens, Greece
Email: dreisis@phys.uoa.gr

***Abstract*— This paper presents a real-time Motion Estimation architecture with improved hardware cost. The design bases on a parallel memory organization minimizing the resources required to support integer and sub-pixel modes of search, while it sustains the required throughput of pixels to the SAD calculator. A speculative execution technique improves the number of cycles required by the search process. The architecture is programmable including an instruction set for actions common to all the block-matching techniques, while circuits introduced at compile time accommodate individual actions of the most demanding algorithms such as MVFAST and PMVFAST. A FPGA implementation validates HDTV video performance.**

I. Introduction

A key role in video encoder architectures plays the design of the Motion Estimation (ME) module due to the requirements for memory space and computational power. Research focuses on either improving the computational complexity of the ME algorithm or on designing parallel architectures to speed up its execution. Fast ME algorithms [1], [2], reduce the complexity and keep the resulting QoS close to that of the Full Search algorithm. Designers of ME architectures concentrate on solving the problem of real-time performance and also on optimizing the hardware resources of Full Search architectures [3], [4], [5], specific fast ME implementations [6], [7], or programmable organizations [8], [9], [10] allowing the choice of the most suitable ME algorithm to each application.

Aiming at providing a solution to the real-time execution of the ME process and also at keeping the hardware cost low, this paper presents a programmable ME architecture. The organization benefits from a non-linear skew memory scheme using eight (8) banks and letting the device to access eight (8) pixels of the candidate block at each cycle. The design is pipelined to operate at increased frequency. To overcome the problem of an empty pipeline, the architecture involves a speculative execution technique to allow the prefetching of pixels. The proposed organization features a set of instructions used by all the block-matching algorithms. Individual actions of more demanding algorithms, such as the MVFAST and PMVFAST [2], are supported by circuits integrated at compile time and triggered by the corresponding specific instructions of each algorithm.

The advantages of the proposed ME design compared to other programmable ME architectures [8], [9], [10], include first, the cost efficient memory scheme improving not only the required hardware resources but also the throughput. Second, the pipeline structure and the application of the speculative execution technique achieve real-time performance even for high definition (HDTV) video as shown by a Xilinx Virtex-II Pro implementation operating at 80 MHz. Third, the organization can speed up the ME process because it is programmable with respect to the sub-sampling of the candidate block and the sub-pixel motion vector refinement.

II. The ME Architecture

The proposed ME architecture consists of three modules: the Data Cache, the SAD Calculator and the Control (fig. 1). The Data Cache module buffers the pixels of the current Macro-Block (MB) and its corresponding search area. During the search process, it forwards 8-tuples of these pixels to the SAD module, where each candidate block is compared against the current MB. The Control executes the search algorithm and determines the sequence of the candidate blocks.

A. Data Cache

The Data Cache functions as a local buffer for the 16×16 pixels of the current MB and also for the 48×48 pixels of the corresponding search area to eliminate repetitive requests for the same pixels to the host encoder. The pixels stored in the Data Cache can be accessed from any location on the search area in three distinct patterns: row, column, or a sparse pattern sub-sampling a 8×4 block (fig. 2). The sparse pattern is useful in scanning a part or the whole of a candidate block, while rows and columns in generating sub-pixels.

Fig. 1 shows the RAM storing the current MB and the *Search Area* RAM, which is a parallel memory with 8 banks and –without redundant pixel storage– it provides the requested row, column and sparse patterns in a single cycle. Each pixel is referenced by a (x, y) vector denoting its position on the area and it is stored in the bank $B = (4 \cdot y + x + \lfloor \frac{y}{2} \rfloor) \bmod (8)$, at the address $A = 12 \cdot \lfloor \frac{y}{2} \rfloor + \lfloor \frac{x}{4} \rfloor$. This mapping (fig. 2) allows the access of any of the above patterns without conflicts. The Data Cache includes the *Interpolator* component, which uses 7 pixels of a row/column pattern for the generation of a row/column of 4 half-pels (using 4-tap filters with coefficients [-1,5,5,-1]), or 4 quarter-pels (using 2-tap averaging filters). Each access pattern and its (x_p, y_p) origin is specified by the SAD module, which will issue successive requests in order to complete the scan of the candidate block under inspection.

The design achieves a constant rate of 16 integer pixels per cycle when forwarding data to the SAD module (8 current MB pixels and 8 search area pixels) by implementing the *Search Area* RAM component as a 5-stage pipeline, consisting of: I) The *Request Translator*, which transforms the $(x_p, y_p, pattern)$ request to 8 pixel addresses and a bank

978-1-4244-5091-6/09/$25.00 ©2009 IEEE

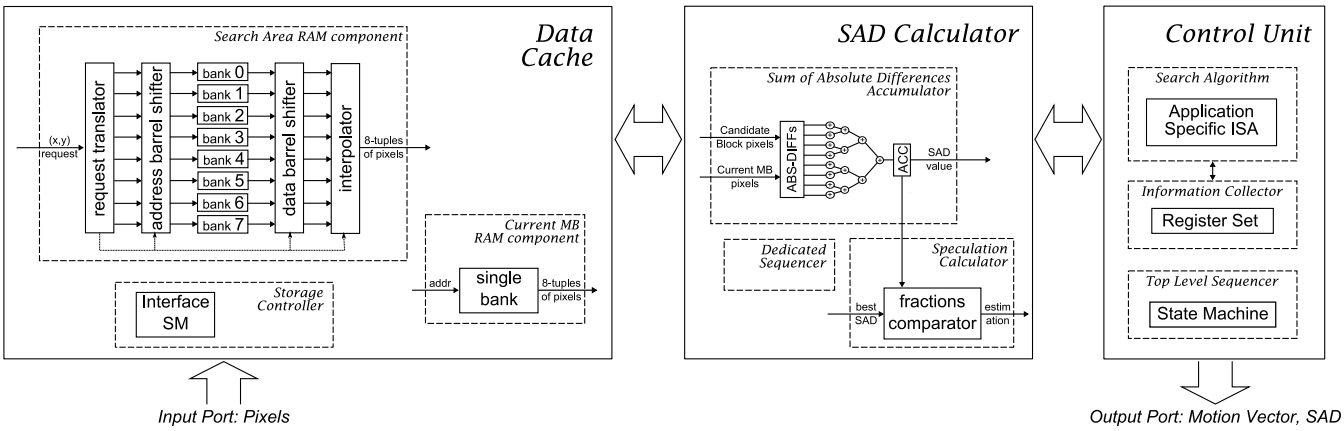

Fig. 1. The Architecture of the Motion Estimation Engine

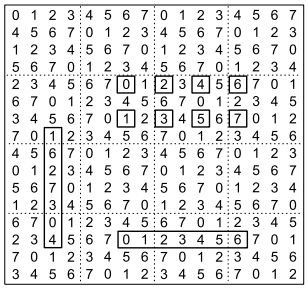

Fig. 2. Mapping of a MB to the 8 RAM banks and three example requests

key by using the aforementioned A and B mappings. II) The address barrel shifter, which forwards the 8 addresses to the banks according to the generated bank key. III) The 8 memory banks. IV) The data barrel shifter, which aligns the candidate pixels to the current MB pixels by using the bank key. V) The 4 filters of the Interpolator component (this stage is bypassed depending on the requested pattern). Stages I, II, III and IV are augmented with the processing elements of the SAD module to form the Memory-to-SAD pipeline, which constitutes the data pipeline of the ME engine.

The Data Cache includes a state machine for refreshing the pixels –through the host encoder's interface– required at the start of each MB process. Note that, the MBs are processed in a raster-scan order and only a part of the search area needs to be fetched: the three MBs located on the right side of the recently processed area.

B. SAD Calculator

The SAD module compares a 16×16 region of the search area (candidate block) with the current MB and it forwards the result to the Control Unit for further processing. This process results in a metric illustrating the resemblance between the two pixel blocks, i.e. the Sum of Absolute Differences.

The Control Unit provides the SAD with a (x_c, y_c) vector specifying the position of the candidate block on the search area. Based on this input, the SAD module initiates up to 32 successive requests to the Data Cache module for fetching the pixels required for calculating the SAD. Each request to the Data Cache addresses two distinct 8-tuples of pixels: one corresponding to Current MB pixels and one corresponding to candidate block pixels. The requested tuples form 8 pairs of pixels, which are directly forwarded to 8 subtractors. The absolute values of their differences are summed within a depth-3 adder tree (the first stage of this tree utilizes add/subtract elements to reduce the number of the absolute converters to 4). The root of this tree is a 16-bit accumulator terminating the flow of the Memory-to-SAD pipeline.

To accommodate the proposed *speculative execution* technique, the SAD module provides the Control Unit with an estimation of whether the outcome of the SAD computation is going to be smaller than the best SAD value discovered so far. The estimation is computed by the *speculation calculator* as follows: it compares the dynamically growing SAD value with the corresponding fraction of the *best_sad* ($\frac{1}{4}$, $\frac{1}{2}$). The result of this comparison is sent to the Control Unit along with an exception signal, which is generated when a notification differs from the preceding notifications (false estimation). Note that, the exception signal does not affect the accumulation procedure and thus, it causes no delay to the SAD module.

In the sub-pixel mode of operation, the SAD calculation requires up to 64 cycles for each candidate block. The requests address rows/columns of 4 sub-pixels instead of sparse blocks and half of the adder tree is deactivated. All of the above SAD operations are sequenced by an internal state machine.

C. Control

The Control unit consists of three components (fig. 1): the *Top Level Sequencer*, the *Information Collector* and the *Search Algorithm*. The Top Level triggers the main procedures of the ME engine. The Information Collector provides the Search Algorithm with the motion vectors and the SAD values of the MBs, which are adjacent to the current MB. The Search Algorithm component –which is the core of the Control Unit– handles the SAD module and implements the branching steps of the search algorithm.

The Search Algorithm component, depicted in figure 3, provides the (x_c, y_c) coordinates of a candidate block. The

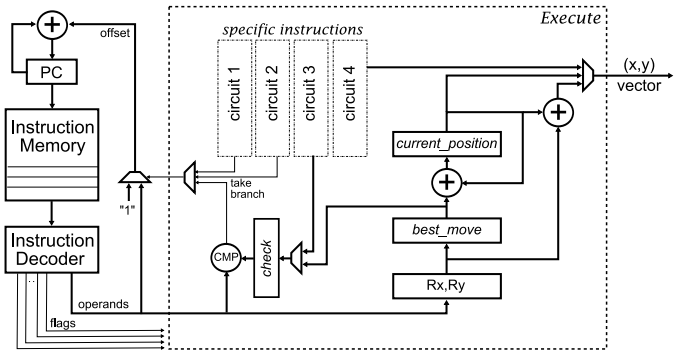

Fig. 3. Search Algorithm component architecture

coordinates are forwarded either to the SAD module during the search process or to the host encoder when the candidate block becomes the final MB predictor. The main feature of this component is its programmability. We developed an Instruction Set Architecture (ISA) to let the Instruction Memory (16-bits wide ROM) store multiple block-matching programs, one of which is selected at the start of each MB process.

To design the ISA, we identified the commonalities of a large number of ME algorithms. Consequently, the organization of the Search Algorithm component (fig. 3) reflects the recursion that almost every ME algorithm uses: "find a good candidate block and improve the result, if possible, with respect to the location of that particular block". Driven by this observation, we use two 16-bit registers (*current_position* and *best_move*) to track the path of the candidate block on the search area. These registers store (x_c, y_c) pointers corresponding to good candidates: *current_position* points to the best block discovered till the current algorithm iteration, while *best_move* points to the best block discovered during the current iteration. During each iteration, the target moves through the search points of the pattern of the algorithm by loading the R_x,R_y register with the relative coordinates of each search point. The R_x,R_y values are provided by the `SAD` instructions of the program (a limited number, stored in the Instruction Memory). At the end of each iteration, the `UPDATE` instruction refines the motion vector stored in the *current_position* register and loads the *best_move* to the *check* register. In this way, the branching steps can be implemented by using the `CMP` instruction to examine the relative position of the best candidate block (most common search strategy) and then, by using the `JUMPC` instruction to determine the flow of the program. Moreover, this approach allows for the evasion of candidate block reinspections because the branches follow the direction of the target block as it moves on the search area.

Apart the instructions described above, the proposed ISA features instructions accommodating the functionality of specific algorithms. For instance, `TSTOP` terminates the search procedure when the SAD value reaches the specified threshold, `MBACT` calculates the motion activity of the current MB and allows for different program flows, etc. These algorithm specific instructions correspond to modular circuits (fig. 3), which can be omitted at compile time to minimize the hardware resources.

To avoid stalling the Control module during the execution of the `SAD` instruction (which invokes the SAD calculator) we use the *speculative execution* technique. The key idea of this technique is that the program flow does not depend on the returned SAD value itself, but rather on the comparison of this value with the best SAD value discovered so far. Further, the outcome of this comparison can be estimated during the SAD procedure based on the partially accumulated SAD value. Hence, the *speculation calculator* forwards its estimations to the Control module, which in turn continues with the program execution working in parallel with the SAD module. In case of a false estimation, an exception signal will result in a rollback restoring the state of the algorithm at the time of the specific SAD initiation (we temporarily store the values of the three basic registers and the PC for this exception handling). Note that, a false estimation does not result in a SAD re-initiation and thus, the only time loss is in terms of speculative operations. The *speculative execution* is blocked when a new `SAD` instruction is decoded. In this case, the newly formed (x'_c, y'_c) pointer is sent to the SAD module, allowing the module to start loading the pixels of the new candidate block immediately after the termination of the pending accumulation. Thus, the speculative execution eliminates the empty slots in the Memory-to-SAD pipe (almost 100% utilization) and improves the utilization of the instruction pipe.

The `SUBPEL` instruction activates the sub-pixel mode of operation. In a predefined sequence, the Search Algorithm executes an iteration to find a horizontal or vertical displacement of the integer result by $1/2$ (four candidates) and an iteration to further refine the MV by $1/4$ or $3/4$ (two candidates).

Finally, we note that the ISA involves parameters such as the sub-sampling factor of the MB ($\frac{1}{4}$, $\frac{1}{2}$ or $\frac{1}{1}$ of a chess-board) and the type of the sub-pixel refinement (half-pels, quarter-pels or none) for time or energy conservation reasons.

III. Implementation and Evaluation

The proposed ME architecture was implemented in a Xilinx FPGA, Virtex-II Pro 40. It utilizes 1579 slices (34K eq. gates) and 9 block RAMs operating at 80 MHz: 691 slices for the Data Cache, 192 for the SAD calculator and 696 for the Control. A lighter version of the architecture, not supporting the advanced MVFAST and PMVFAST algorithms, occupies 1176 slices and saves 25% on the hardware resources. Overall, the proposed architecture utilizes 23% less slices than [8], which achieves real-time performance only for low resolution videos. Compared to [9], the architecture presented in this paper reduces the resources by 26% and improves the MB throughput by up to 33% when giving the same PSNR results. In [10], a straightforward memory design accesses only rows of pixels and it cannot speed up the execution of the algorithm by allowing on-the-fly generation of sub-pixels or pixel subsampling; as a result, [10] constitutes a much costlier solution for real-time applications processing HDTV videos and/or subpixels. Even when compared to some VLSI non-programmable

TABLE I

PERFORMANCE RESULTS FOR VARIOUS ME ALGORITHM CONFIGURATIONS

		Diamond Search						PMVFAST Search						*RT limit*
		without sub-pixels			with sub-pixels			without sub-pixels			with sub-pixels			@25fps,
Frame Size	*Statistics*	ssf=1	ssf=2	ssf=4	ssf=1	ssf=2	ssf=4	ssf=1	ssf=2	ssf=4	ssf=1	ssf=2	ssf=4	80MHz
352x288	PSNR	32.71	32.69	32.34	33.42	33.31	32.87	32.91	32.88	32.60	33.67	33.57	33.18	
	cycles/MB	502	383	341	713	499	410	282	238	220	501	359	295	8080
720x576	PSNR	33.31	33.29	33.13	33.59	33.56	33.33	33.54	33.53	33.41	33.92	33.87	33.67	
	cycles/MB	684	479	410	982	637	498	334	269	242	640	431	335	1975
1280x720	PSNR	33.44	33.42	33.28	33.68	33.64	33.44	34.44	34.40	34.24	34.77	34.70	34.47	
	cycles/MB	786	535	446	1115	708	538	355	281	251	692	459	349	889
1920x1088	PSNR	32.60	32.57	32.45	32.82	32.77	32.58	34.53	34.46	34.32	34.82	34.74	34.52	
	cycles/MB	890	587	481	1246	772	580	408	310	273	772	500	377	392

solutions, our organization requires less hardware resources: it uses almost half the logic gates of [7] without degrading the MB throughput.

To evaluate the performance of the proposed ME, we ran several tests on various videos (foreman, pedestrian, etc) of four distinct resolutions (CIF, SDTV and two HDTV). We used two well-known search algorithms, Diamond and PMVFAST, parameterized respectfully to the sub-sampling factor of the MB (ssf=$1|2|4$) and the fractional refinement of the motion vector. Table I summarizes the results: it shows the average number of clock cycles per MB process (including pixel storing) versus the quality of the predicted frame (PSNR). The last column shows the cycle budget that should be allocated at each MB process for *Real-Time* performance (*RT cycle limit*).

According to table I, searching for an integer MV by using half of the MB pixels (ssf=2) results in only 0.01-0.07 dB PSNR degradation, while it saves 16-34% of the MB process time. Such improvement on the cycle count is crucial in achieving real-time performance in HDTV videos (see RT limit in table I). For ssf=2, PSNR degradation shows a slight increase when working with sub-pixels (where the MB cannot be sub-sampled as an exact chess-board), but the time savings increase up to 38%. In the case of ssf=4, both the quality degradation and the time savings increase even further. The vertical/horizontal fractional MV refinement improves the PSNR by 0.2-0.8 dB. However, this improvement comes at the expense of 20-95% more cycles. Table I shows that in most cases, choosing ssf=4 with sub-pixels prevails over ssf=1 without sub-pixels in both time and PSNR results.

Other tests have verified the following: I) The speculative execution (SE) technique reduces the required cycle count by up to 20% for the Diamond and up to 10% for the PMVFAST. This reduction of the time requirements is similar to that of the Early Escape (EE) technique (stop accumulating when the partial SAD value exceeds the best SAD value) in low resolution videos. However, when processing HDTV videos with sub-pixels, the EE technique saves less time, rendering the SE more important by a factor of 2. The combination of these techniques leads to the results of table I with savings up to 34%. II) Compared to the 6-tap interpolation filters of the H.264 standard, the use of 4-tap FIRs can save up to 32% processing time (they generate 4 sub-pels per 7 pixel row/column, not only 2). This speedup comes at the cost of decreasing the quality of our results only by 0.004 dB. III) When ssf=1, using sparse patterns instead of rows/columns can save up to 6% of the processing time, because it improves the speculative execution technique: we consider a wider area of the candidate block when computing each speculation.

IV. CONCLUSION

The current paper presented a programmable, cost efficient, real-time ME architecture. The organization includes an instruction set common in executing most ME algorithms and it can be configured to support specific instructions for more advanced algorithms such as MVFAST and PMVFAST. The proposed design bases on a parallel memory improving the required hardware resources compared to hitherto published results and by implementing the speculative execution technique it sustains real-time performance even for HDTV video.

REFERENCES

[1] Y. Liu and S. Oraintara, "Complexity Comparison of Fast Block-Matching Motion Estimation Algorithms," IEEE Int. Conf. on Acoustics Speech and Signal Processing, May 2004.

[2] A. M. Tourapis, O. C. Au and M. L. Liou, "Highly Efficient Predictive Zonal Algorithms for Fast Block-Matching Motion Estimation," IEEE Tr. on Circ. and Syst. for Video Technology, vol. 12, no.10, Oct 2002.

[3] C. A. Rahman and W. Badawy, "A Quarter Pel Full Search Block Motion Estimation Architecture For H.264/AVC," IEEE Int. Conf. on Multimedia and Expo, July 2005.

[4] A. M. Campos, F. J. Ballester Merelo, M. A. Martinez Peiro and J. A. Canals Esteve, "Integer-pixel motion estimation H.264/AVC accelerator architecture with optimal memory management," Microprocessors and Microsystems, vol. 32, iss. 2, pp. 68-78, Mar 2008.

[5] T. Moorthy and A. Ye, "A scalable architecture for variable block size motion estimation on Field-Programmable Gate Arrays," Canadian Conf. on Electrical and Computer Engineering, IEEE, May 2008.

[6] J. Choi, N. Togawa, M. Yanagisawa and T. Ohtsuki, "VLSI Architecture for a Flexible Motion Estimation with Parameters," IEEE Proc. of the 15th Int. Conf. on VLSI Design, pp. 452-457, 2002.

[7] B.F. Wu, H.Y. Peng and T.L. Yu, "Efficient Hierarchical Motion Estimation Algorithm and Its VLSI Architecture," IEEE Tr. on Very Large Scale Integration Systems, vol. 16, iss. 10, pp. 1385-1398, Oct 2008.

[8] T. Dias, N. Roma, L. Sousa and M. Ribeiro, "Reconfigurable architectures and processors for real-time video motion estimation," J. of Real-Time Image Processing, vol. 2, no. 4, pp. 191-205, Dec 2007.

[9] K. Babionitakis, G. Lentaris, K. Nakos, N. Vlassopoulos, D. Reisis, G. Doumenis, G. Georgakarakos and I. Sifnaios, "A real-time motion estimation FPGA architecture," J. of Real-Time Image Processing, vol. 3, no. 1-2, pp. 3-20, Mar 2008.

[10] J. L. Nunez-Yanez, E. Hung and V. Chouliaras, "A configurable and programmable motion estimation processor for the H.264 video codec," IEEE Conf. on Field Prog. Logic Applications, pp. 149-154, Sep 2008.