

Parallel Architectures of 3-Step Search Block-Matching Algorithm for Video Coding

Her-Ming Jong, Liang-Gee Chen*, and Tzi-Dar Chiueh
 Department of Electrical Engineering,
 National Taiwan University, Taipei, Taiwan, R.O.C.
 email: d80047@cc.ee.ntu.edu.tw

ABSTRACT

This paper describes fully pipelined parallel architectures for the 3-step search block-matching motion estimation algorithm. Difficulties of this algorithm in hardware implementation were overcome by use of intelligent data arrangement and memory configuration. Techniques for reducing interconnections and external memory accesses were also developed. Because of their low costs, high speeds, and low memory bandwidth requirements, the proposed architectures provide efficient solutions for real-time motion estimations required by various video applications.

I. INTRODUCTION

Among various video compression techniques, the motion-compensated hybrid coding is the most popular one and is adopted by several international standards. The block-matching motion estimation/compensation provides these coding systems with significant bit-rate reductions. However, it also requires a large amount of computation and a heavy memory bandwidth.

Many fast block-matching algorithms (BMA's) have been developed to reduce the extremely high computational complexity of the optimal full search (FS) procedure. Among all these BMA's, the 3-step hierarchical search (3SHS) [1] is considered as one of the best algorithms and is recommended by MPEG and RM8 of H.261. In spite of the good performance and significant complexity reduction provided by 3SHS, almost all reported BMA architectures and VLSI implementations select FS because of its regular data flow and low control overhead [2]-[3]. These architectures efficiently reuse data to decrease external memory accesses, and they speed up the computation by highly parallel processing and pipelining. However, because of the inherent high complexity of FS, high-speed motion estimators can only be provided by large arrays of processing elements (PE's).

The low complexity of 3SHS algorithm makes it potential to implement high-speed motion estimators at low hardware costs. To fully utilize this advantage, we developed a dedicated architecture which overcomes the irregular data flow of 3SHS and thus achieves an efficiency

close to 100 percent. It also successively reuses data and reduces the control overhead. Depending on the trade-off between cost and speed, this scheme can be reduced or expanded to meet requirements of various video coding systems, from low bit-rate video to HDTV systems.

II. THE 3-STEP SEARCH ALGORITHM

The procedure of a block-matching algorithm is to find a best matched displaced block from the previous frame, within a search range, for each block in the present frame. A straightforward method, the full search, exhaustively matches all possible candidates to find the displacement (called motion vector) with a minimal distortion. To reduce the heavy computational cost resulted from the massive number of candidate locations, the 3SHS algorithm searches for the best motion vector in a coarse-to-fine manner. For the commonly used search range of -7 to $+7$ pixels, the hierarchical search procedure decreases the number of searched locations to $1/9$ of the exhaustive approach.

Experimental results show that the 3SHS provides a robust near-optimal performance [1], and the difference between performances of 3SHS and FS can be further reduced, without increasing the computational cost, by a modification proposed in [4]. The low complexity and the high accuracy make 3SHS a potential solution for high-speed video applications; However, some architectural considerations prevent this algorithm from being widely used in real-time systems: First, the variable distances between candidate locations and the unpredictable data requirement complicate the control scheme, lower the efficiency of computation kernel, and make it difficult to reuse data for reducing the number of external memory accesses; Second, the dependency between steps entails sequential execution, so the latency of each step need be short for high-speed operation. This imposes an additional limitation in architecture design. Architectures presented in the following sections solve these problems and provide efficient implementations of the 3SHS algorithm.

III. THE PROPOSED 3SHS ARCHITECTURES

A. Algorithm Mapping and Basic Structure

¹now is Research Consultant in AT&T Bell Lab.

²This work is supported by National Science Council, Republic of China, under Grant NSC 83-0404-E-002-008.

The following loops describe the main computations in 3SHS:

Loop 1: For each of the 3 steps

Loop 2: For each of the 9 candidate locations

*Loop 3: For each of the 256 pixel-pairs in a pair of blocks
Accumulate the absolute pixel difference.*

Considering the cost of computation and data access, we proposed the following algorithm mapping: 9 PE's evaluate the 9 locations of *Loop 2* in parallel and sequentially execute operations in *Loop 1* and *Loop 3*. An additional advantage of this approach is the flexibility in search range and block size. These cases can be handled by only changing the contents of counters in *Loop 1* and *Loop 3*, which don't affect the hardware structure. Furthermore, because the required pixels are sent to PE's in parallel without data skewing, the latency delay is also very short.

In principle, this architecture provides a flexible high-speed motion estimator at a low cost. On-chip buffers are suggested to reuse data and thus to reduce the required I/O memory bandwidth. For parallelizing the data accesses of PE's and reducing the complexity of interconnection between buffer and PE's, techniques called residual memory interleaving and PE function redistribution are described in the following subsections.

B. Residual Memory Interleaving

On-chip buffers reduce the load for chip I/O and memory system. The next problem is: how to provide all required data for the 9 PE's simultaneously? Our approach is dividing the buffer into $3^2 = 9$ memory modules, and interleaving search area pixels to these 9 modules as shown in Fig. 1. The label (0 ~ 8) for each pixel indicates the module that stores this pixel. Because of the hierarchical characteristic of 3SHS, the distance between adjacent candidate locations is always 2^k ($k=2, 1, \text{ or } 0$). Since the residue of dividing 2^k by 3 is never zero, the 9 required pixels always reside in 9 different modules. This is illustrated in Fig. 1 by marking the simultaneously accessed pixels at some instant in each step. This memory interleaving provides a solution for parallel data accesses, but it asks every PE to be able to access each of the 9 memory modules. The PE function redistribution, described in the following, can significantly reduce the interconnection overheads.

C. Redistribution of PE functions

The basic structure introduced in Sec.3.1 maps operations for evaluating a certain candidate location to a fixed PE. The proposed function redistribution allows operations belonging to a candidate location to be performed by several PE's, then it combines partial-results from these PE's to form the MAD of this location.

Consider the example in Fig. 2(a): If PE0 always evaluates candidate locations 0, it has to access memory module 0 ~ 2 during clock cycle 0 ~ 15, and to access module 3 ~ 5 and 6 ~ 8 during clock cycle 16 ~ 31 and 32 ~ 47 respectively. That is, all of the 9 memory

modules store data for this PE, which demands a fully connected nonblocking switching network from 9 sources to 9 destinations.

Rather than fix its job, we allow a PE to calculate partial-results of different candidate locations at different moments. In the case mentioned above, we let PE0 evaluate location 0 in the first 16 clock cycles, and then let it evaluate location 3 and then location 6 for the next two 16-clock-cycles, as shown in Fig. 2(b). From the other viewpoint, the evaluation of location 0 is completed by a cooperation between PE0, PE3, and PE6, where jobs are cyclically redistributed every 16 clock cycles. By use of this method, each PE only has to connect to 3 memory modules. The PE-location mapping table in Fig. 3 illustrates the whole procedure by showing every PE's corresponding candidate locations at different instants. Now consider the integration of partial-results from different PE's: For every 16 clock cycles, the 9 PE's produce 9 partial-results and transfer them to PE's output latches. Therefore PE's can immediately begin to compute the next set of partial-results. These 9 latched partial results can then be accumulated by a queue and a time-sharing common bus, one at a time by a proper selecting signal *sel*. The queue provides space for storing 9 intermediate values, one for each MAD. After 256 clock cycles, each MAD accumulates 16 partial-results that are collected from appropriate PE's.

The above method redistributes PE functions only in the vertical direction. In principle, the same concept can also be applied horizontally to provide further interconnection reduction. However, directly applying this method in the horizontal direction causes massive interchanges of partial-results. This drawback can be overcome by rearranging the data processing order within each row to concatenate operations belonged to the same candidate location. As a result, the number of partial-result interchanges is reduced from 16 to 3 per row, and thus the accumulation can be handled by a two-level configuration. This scheme is a direct expansion of the one-level partial-result queue mentioned above, and it is described in Sec.3.4 together with our final proposed architecture. Fig. 4 shows the PE-location mapping when the horizontal process rearrangement and function redistribution are also applied. By use of this method, each PE has to connect to only a certain memory module. This eliminates the complicated interconnection and switching circuitry between memory modules and PE's.

D. Implementation Details

Fig. 5 shows the proposed 3SHS architecture, which combines the basic 9-PE structure and techniques presented above. According to the principle of residual memory interleaving, the search area buffer is divided into 9 memory modules. The address for writing (*wa*) is steadily broadcasted to all memory modules independent of the previous result. In contrast, 9 different addresses for reading (*ra*), which depend on the result of the previous step, are required by PE's simultaneously. Fortunately, each of these addresses differs from others by predictable offsets and can be locally calculated from a

base address that is the only one the address generator has to provide.

A two-level scheme of partial-result queues was developed to accumulate the partial-results to proper MAD's. In this scheme, the first level handles the horizontal PE function redistribution. It consists of three partial-result queues of length 3. Because the horizontal function redistributions occur for every 5 or 6 clock cycles, the 3 queues can process all the 9 partial-results latched in PE's before the next 9 partial-results are produced. Similarly, the second level executes the vertical function redistribution by selecting appropriate partial-results from the three level-1 queues. Three additional queues of length 3 provide spaces for storing horizontally processed partial-results. They make level-1 queues able to process the next set of data when the horizontally integrated data are waiting for vertical processing. After 256 clock cycles, the last set of partial-results is produced by PE's. Another 3 clock cycles are spent to complete the horizontal function redistribution. In the next 9 clock cycles, the partial-result queue provides the 9 MAD's and a comparator finds the minimal one, then the address generator uses this information to select the base address of interested locations in the next step. Although PE's are idle during this period, the overhead is insignificant compared with the total execution time, and the utilization of PE's is 96.7%.

In summary, the architecture presented above combines the computing power of parallel PE's and the flexibility of random-access buffers. It also effectively utilizes the key features of hierarchical BMA's to manage data, and thus it can be applied to implement other algorithms of this type. This architecture can be directly reduced to a 3-PE structure for low-cost applications. It can also be expanded to a 27-PE scheme as shown in Fig.6. This pipelined 3-stage structure provides high-speed motion estimators required for future high-definition video media. Compare with FS architectures with the same throughputs, the proposed approaches requires only about 1/9 of PE's.

IV. CONCLUSION

In this paper, we presented a family of efficient architectures for the 3-step hierarchical search block-matching algorithm: a 9-PE novel design and its 3-PE low-cost and 27-PE pipelined high-speed variants. The random-access on-chip buffer and input data are arranged by a principle called residual memory interleaving. Combined with a technique of PE function redistribution, the interleaved buffer provides every PE with its required data simultaneously without introducing complicated interconnections and switching circuitry. In summary, the proposed architectures have the following desirable features: (1) very few PE's and low interconnection overhead, (2) high throughput rate, (3) low latency delay, (4) low I/O and memory bandwidth requirements, and (5) close to 100 percent efficiency. As shown in Table 1, the proposed approach provides efficient motion estimators suitable for various video applications.

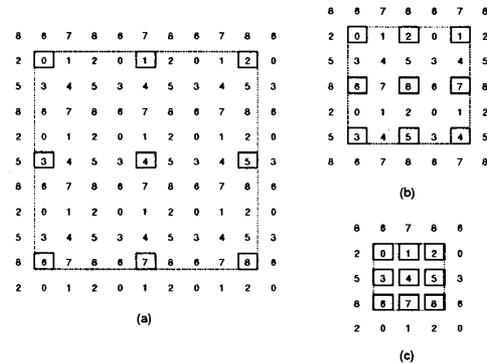


Figure 1: The residual memory interleaving and the accessed data (marked by small frames) at the first clock cycle of: (a) step 1; (b) step 2; (c) step 3.

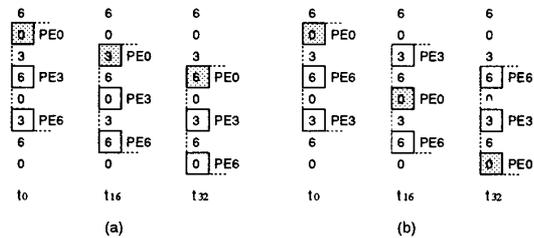


Figure 2: An illustration of PE function redistribution: (a) original fixed PE-location mapping; (b) dynamic PE-location mapping via function redistribution (t_i : clock cycle i of step 2).

References

- [1] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, Nov.29-Dec.3, 1981, pp. G5.3.1-5.3.5.
- [2] K. M. Yang, M. T. Sun and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," *IEEE Trans. Circuits Syst.*, Vol. 36, No. 10, pp. 1317-1325, Oct. 1989.
- [3] T. Komarek and P. Pirsch, "Array architectures for block-matching algorithms," *IEEE Trans. Circuits Syst.*, vol. 36, no. 10, pp. 1301-1308, Oct. 1989.
- [4] H. M. Jong, L. G. Chen, and T. D. Chiueh, "Accuracy improvement and cost reduction of 3-step search block-matching algorithm for video coding", to be appeared on *IEEE Trans. on Circuits and Systems for Video Technology*.

clock cycle (n x 10 + m)		candidate locations evaluated by PE's								
n	m	PE0	PE1	PE2	PE3	PE4	PE5	PE6	PE7	PE8
step 1	0 0-15	0	1	2	3	4	5	6	7	8
	1 0-15	6	7	8	0	1	2	3	4	5
	2 0-15	3	4	5	6	7	8	0	1	2
	3 0-15	0	1	2	3	4	5	6	7	8
⋮										
step 2	14 0-15	3	4	5	6	7	8	0	1	2
	15 0-15	0	1	2	3	4	5	6	7	8
	16 0-15	0	2	1	6	8	7	3	5	4
	17 0-15	3	5	4	0	2	1	6	8	7
18 0-15	6	8	7	3	5	4	0	2	1	
19 0-15	0	2	1	6	8	7	3	5	4	
⋮										
step 3	30 0-15	6	8	7	3	5	4	0	2	1
	31 0-15	0	2	1	6	8	7	3	5	4
	32 0-15	0	1	2	3	4	5	6	7	8
	33 0-15	6	7	8	0	1	2	3	4	5
34 0-15	3	4	5	6	7	8	0	1	2	
35 0-15	0	1	2	3	4	5	6	7	8	
⋮										
step 3	46 0-15	3	4	5	6	7	8	0	1	2
	47 0-15	0	1	2	3	4	5	6	7	8

Figure 3: The PE-location mapping table when only vertical function redistribution is applied.

clock cycle (n x 16 + m)		candidate locations evaluated by PE's								
n	m	PE0	PE1	PE2	PE3	PE4	PE5	PE6	PE7	PE8
step 1	0 0-5	0	1	2	3	4	5	6	7	8
	0 6-10	2	0	1	5	3	4	8	6	7
	0 11-15	1	2	0	4	5	3	7	8	6
	1 0-5	6	7	8	0	1	2	3	4	5
	1 6-10	8	6	7	2	0	1	5	3	4
	1 11-15	7	8	6	1	2	0	4	5	3
	2 0-5	3	4	5	6	7	8	0	1	2
	2 6-10	5	3	4	8	6	7	2	0	1
	2 11-15	4	5	3	7	8	6	1	2	0
	3 0-5	0	1	2	3	4	5	6	7	8
	3 6-10	2	0	1	5	3	4	8	6	7
	3 11-15	1	2	0	4	5	3	7	8	6
⋮										
step 2	15 0-5	0	1	2	3	4	5	6	7	8
	15 6-10	2	0	1	5	3	4	8	6	7
	15 11-15	1	2	0	4	5	3	7	8	6
	16 0-5	0	2	1	6	8	7	3	5	4
16 6-10	1	0	2	7	6	8	4	3	5	
16 11-15	2	1	0	8	7	6	5	4	3	
⋮										
step 3	31 11-15	2	1	0	8	7	6	5	4	3
	32 0-5	0	1	2	3	4	5	6	7	8
	32 6-10	2	0	1	5	3	4	8	6	7
32 11-15	1	2	0	4	5	3	7	8	6	
⋮										
step 3	47 11-15	1	2	0	4	5	3	7	8	6

Figure 4: The PE-location mapping table when both vertical and horizontal function redistribution are applied.

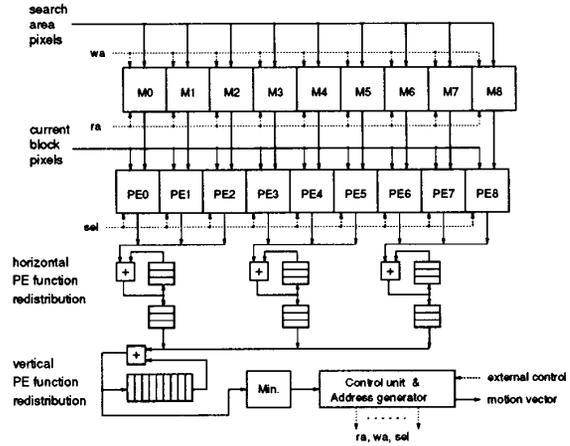


Figure 5: The proposed 9-PE 3SHS architecture.

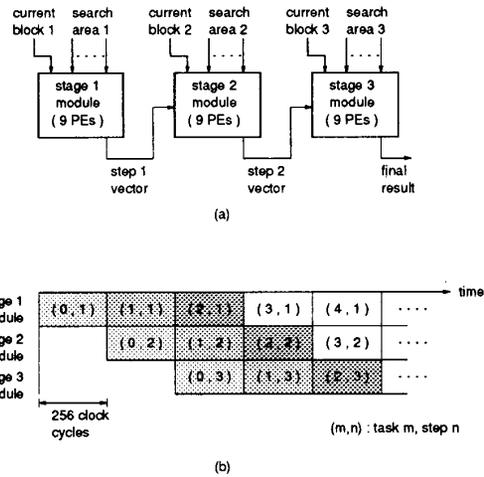


Figure 6: The pipelined 3-stage 27-PE architecture: (a) block diagram; (b) reservation table.

No. of PE's	No. of input ports	Buffer size (KB) ^a	Clock cycles per block ^b
3	2	0.64	2330
9	2	1.44	794
27	3	4.83	269

^aThe sizes can be further reduced by half, by an algorithm proposed in [4].

^bThe overhead between steps for deciding the next step address is included.

Table 1: Performance and costs of proposed architectures and suitable applications.