Hardware Oriented Content-Adaptive Fast Algorithm for Variable Block-Size Integer Motion Estimation in H.264

Yu-Han Chen, Tung-Chien Chen, and Liang-Gee Chen
DSP/IC Design Lab, Graduate Institute of Electronics Engineering and
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
Email:{doliamo, djchen, lgchen}@video.ee.ntu.edu.tw

Abstract— **Motion estimation can reduce temporal redundancy and achieve high compression capability in video coding standards. In H.264, the coding gain is further improved by variable block-size motion estimation (VBSME). In order to reduce the complexity, many fast algorithms have been proposed. Though previous works can reduce a large amount of computation, most of them are not suitable for hardware implementation and not robust for complex motion videos. A content-adaptive fast algorithm for variable block-size integer motion estimation (VBSIME) is proposed in this paper. Motion activity is exploited to improve the coding efficiency. Because of the good data reuse scheme and simple control flow, the proposed algorithm is applicable to hardware implementation. According to the simulation results, about 98% searching candidates and 86% encoding time are reduced with at most 0.05dB quality drop in average compared with full search even for complex motion videos.**

I. Introduction

H.264 is an advanced video coding standard co-developed by ITU-T Video Coding Experts Group and ISO/IEC Moving Picture Experts Group [1]. This standard provides superior coding tools to upgrade the coding efficiency and video quality when compared with previous standards. Due to its high compression capability, H.264 is potential to be adopted in many emerging applications.

Motion estimation is the core technique to remove temporal redundancy and to achieve high compression ratio in video coding standards. VBSME largely enhances the ME performance in H.264. For coding a macroblock (MB), 7 kinds of block-sizes (16×16, 16×8, 8×16, 8×8, 8×4, 4×8, and 4×4) are allowed. In reference software [2], matching costs of different block-sizes are calculated individually. Without good data reuse, 7 times of computational resources are required. To solve this problem, all SAD (Sum of Absolute Difference) costs of the smallest 4x4 blocks are computed first. Next, other costs of larger block-sizes are generated from the 4x4 costs. Finally, full search is applied to all kinds of partitions by means of the pre-computed costs. Though computation is greatly reduced, a large amount of memory is required to store all SAD data. For hardware consideration, it is infeasible. In [3], a full search algorithm and its architecture for VBSME has been proposed. Unlike the sequential flow in reference software, all costs of different block-sizes are computed in parallel, and good data reuse is attained. However, in resource-constrained systems such as mobile devices, fast search algorithm is a must. Therefore, a fast search algorithm with a good data reuse scheme is required.

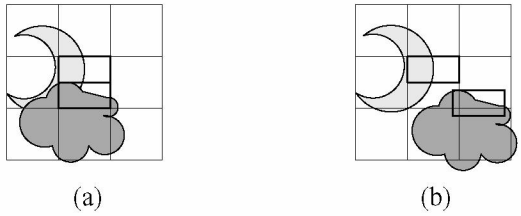


Fig. 1. Complex motion scene. (a) The current frame; (b) The previous frame.

VBSME is useful for videos with complex motion. An example is depicted in Fig. 1. In this scene, the moon is still but the cloud is moving. We can't get good matching block for whole 16×16 MB in the center unless the 16×8 partition is used. Traditional fast algorithms are developed for single block-size and easy to be trapped in local minimum. That is to say, they can't provide robust coding efficiency especially in complex motion videos. For VBSME, motion activity in the neighboring area should be exploited well. In the complex motion area, more computation is needed to search for the best matching candidate. On the other hand, less computation is consumed while the motion is simple. A motion-adaptive fast algorithm for VBSME is beneficial to computation reduction and quality maintenance.

In this paper, a hardware oriented content-adaptive fast algorithm for VBSME is proposed. Motion activity is exploited well to improve the coding efficiency. Because of the good data reuse scheme and simple control flow, it is suitable for hardware implementation. The rest of this paper is organized as follows. For a start, problem analysis is illustrated in section II. Then, the proposed fast algorithm followed by the hardware architecture is introduced in section III. The performance is shown in section IV. Finally, we will give a conclusion in section V.

II. Analysis

Though VBSME contributes high coding efficiency, it also occupies a major part of computational load in H.264 encoder.

0-7803-9266-3/05/$20.00 ©2005 IEEE.

According to the run time profile, about 60% computation time is spent in integer motion estimation (IME) when the searching range (SR) is set to 16. With SR larger than 32, IME will dominate whole encoding system (more than 90%). In order to reduce the complexity and meet the real-time constraint, a fast search algorithm for VBSIME is needed.

Conventional fast block matching algorithm (BMA), such as four step search (4SS) [4] and diamond search (DS) [5] are developed for previous standards with single block size. If we directly adopt them for VBSME with the sequential procedure in reference software, computation reduction is limited. The SAD reuse scheme in reference software is not suitable here because pre-computing all SAD costs is too expensive for fast search algorithms. Without good data reuse, the computation will increase proportional to the number of block-sizes. For example, the minimum number of searching candidates for DS is 13 in previous standards but $91(13 \times 7)$ in H.264. That is to say, a fast search algorithm with good data reuse scheme is important.

Several fast algorithms for VBSME have been proposed. In [6], authors propose a top-down procedure to process the largest 16x16 block first. In [7], a bottom-up approach starting from the smallest 4x4 blocks is suggested. By combining the above two ideas, a merge-and-split scheme is proposed in [8]. In these algorithms, motion estimation for different block-sizes are performed sequentially in defined criteria, and computation is reduced by early termination scheme. However, the control is complex ,and the sequential flow still leads to poor data reuse.

In [9], a data-adaptive motion estimation algorithm is proposed. According to the motion activity, the proposed algorithm adjusts the size of searching window to reduce unnecessary computation. Because full search is applied within the adaptive window, it can achieve good data reuse. However, in complex motion areas, the SR should be large enough to maintain video quality which leads to a large amount of computation.

According to these considerations, a fast search algorithm with a good data reuse scheme and a simple control flow is required. Besides, motion activity needs to be exploited well to improve the coding efficiency especially in complex motion areas. In the following section, our proposed algorithm satisfying all the requirements above will be introduced.

III. Proposed Content-Adaptive Fast Algorithm

A. Concepts

As mentioned before, a good data reuse scheme is important for fast search algorithms. In fact, if we compute all corresponding 4x4 SAD costs for a search point, all other costs of larger block-sizes can be directly merged from them. In this way, the SAD costs can be reused well without additional memory usage. This scheme seems similar to that in reference software, but they are different in several points. First, our scheme reuse the SAD costs immediately for one searching candidate. We don't need to store them in the memory and it's very efficient for hardware consideration. Secondly, all the costs of different block-sizes are generated simultaneously. Unlike the sequential flow in reference software, motion estimation for different block-sizes needs to be processed in parallel for this scheme. However, this parallel flow leads to inaccurate motion vector (MV) costs which will induce some quality loss in low bit-rate condition. We will show the quality degradation in section IV.

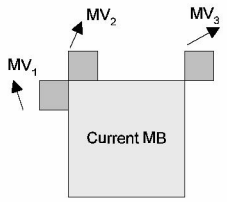


Fig. 2. Neighboring predicted motion vectors.

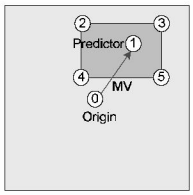


Fig. 3. The proposed moving window fast search algorithm.

Using MV predictors is the general idea to exploit the spatial correlation between neighboring MBs. Conventional video coding standards use the median MV of left, up, and up-right MBs (as depicted in Fig. 2) as the MV predictor of the current MB. But in a complex motion area, the predictor is not accurate. If we only search the area around the predictor, coding efficiency may drop severely. To solve this problem, a moving window fast algorithm is proposed (as depicted in Fig. 3). First, the adaptive moving window is generated according to the neighboring motion vectors ,and motion activity is predicted accurately. Secondly, fast search is applied to not only the predictor but also the vertices of the moving window. It can catch the complex motion better and contribute high coding efficiency.

Motion vectors in a simple motion region have a strong correlation with the predictor. Besides, MBs in the zero motion background usually have motion vectors around the origin. Hence, an adaptive algorithm is needed to search more initial candidates in complex motion videos and less in simple ones. That is to say, the searching effort should be adapted to motion activity.

B. Procedures

The flow of the proposed algorithm is shown in Fig. 4. At first, motion activity is exploited to generate the moving window and the initial searching candidates. Then, the fast search is applied from the initial candidates, and all the costs of different block-sizes are computed in parallel. After several passes of iterations, the 41 best integer MVs and costs are generated.

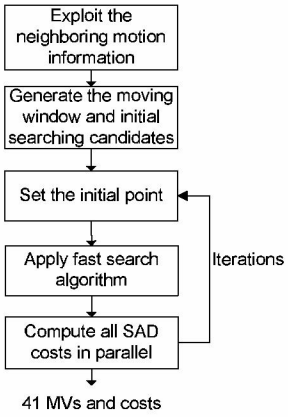


Fig. 4. The flow of the proposed content-adaptive algorithm.

Four boundaries of the moving window are generated from neighboring MVs (MV_1, MV_2, and MV_3 in Fig. 2) as follows.

$$Bound_{up} = max(MV_{y1}, MV_{y2}, MV_{y3})$$
$$Bound_{down} = min(MV_{y1}, MV_{y2}, MV_{y3})$$
$$Bound_{left} = min(MV_{x1}, MV_{x2}, MV_{x3})$$
$$Bound_{right} = max(MV_{x1}, MV_{x2}, MV_{x3})$$

Next, the number of the initial searching candidates should be adjusted by the motion information. If the horizontal components of motion vectors MV_1, MV_2, and MV_3 (in Fig. 2) are all the same, it means horizontal motion is well predicted in this area. We can shrink the moving window in the horizontal direction to save unnecessary computation. For the vertical direction, it's in the same manner. The conditions are shown as follows.

$$If \quad (MV_{x1} = MV_{x2} = MV_{x3})$$
$$Then \quad Shrink\ horizontal\ moving\ window$$
$$If \quad (MV_{y1} = MV_{y2} = MV_{y3})$$
$$Then \quad Shrink\ vertical\ moving\ window$$

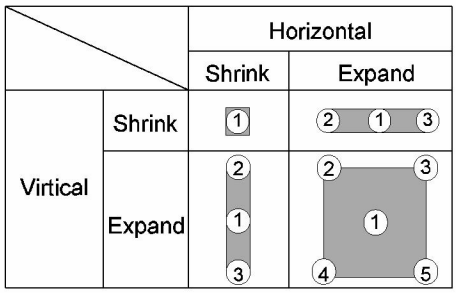
In Table I, we show the number of searching passes and the states of moving window for different conditions. Because background with zero motion may usually occur, we always need to add the origin as another initial candidate. At last, 2, 4, or 6 passes of fast search will be applied according to the motion activity in the video content.

C. Hardware Architecture

Due to the good data reuse scheme and simple control flow, the proposed algorithm is suitable for hardware implementation. The hardware architecture is shown in Fig. 5. The searching window SRAMs are used to stored all the reference pixels inside SR. The data are loaded from external SDRAM through the system bus. In order to compute all costs in parallel, we need to generate the 16x16 absolute difference values simultaneously. Therefore, 16x16 registers are used as the current MB buffer, and 16x16 shift register array is needed to store the reference pixels. The reference data are shifted

TABLE I

LIST OF THE NUMBER OF INITIAL CANDIDATES AND THE STATES OF MOVING WINDOW EXPANSION.



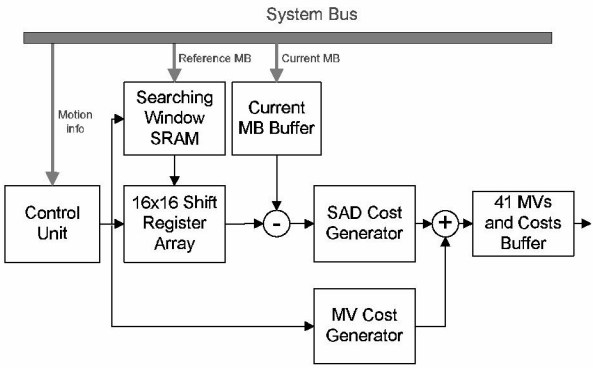


Fig. 5. The hardware architecture of the proposed algorithm.

according to the movement (upwards, downwards, leftwards, and rightwards) of the searching candidate. Because all data needed are stored in registers, we can compute all the absolute difference values simultaneously and accumulate them to the 41 SAD costs. After these SAD costs are added by the MV costs, the final costs are compared to the previous best costs and stored into buffer if the current results are better. A control unit is needed here. It loads the motion information and generates the initial searching candidates at the start. During block matching process, it generates the control signals for reading the required data from searching window SRAMs, shifting the reference register array in the proper direction, and generating the MV costs.

IV. Simulation Results

The proposed content-adaptive algorithm is embedded into reference software JM8.2 encoder. We have tested 1 QCIF and 5 CIF sequences with low, medium, and high motion activity. FSS is chosen as the fast search algorithm in our proposed algorithm. Because its square search pattern is similar to full search and suitable for hardware implementation. Table II shows the PSNR drop, bitrate increasing, and the number of search points per MB. The performance of 6FSS, 2FSS, and the proposed content-adaptive algorithm are compared with full search in JM8.2. 6FSS algorithm fixes 6 passes of iterations for the fast search algorithm (the origin, the predictor, and the four vertices of the moving window). 2FSS

TABLE II

LIST OF PSNR DROP, BITRATE INCREASING, AND SEARCH POINTS PER MB FOR 6FSS, 2FSS, AND THE PROPOSED CONTENT-ADAPTIVE ALGORITHM COMPARED TO FULL SEARCH IN REFERENCE SOFTWARE.

H.264 Baseline Profile, JM8.2, 1 reference frame, IPPP... QP = 15, 18, 21, 24, 27, 30, 33, 36, 39, 42			
	PSRN drop (dB)	Bitrate increase (%)	Search Point per MB
Foreman, QCIF, SR = ±16			
6FSS	0.039	-0.04%	105.5
Proposed	**0.048**	**0.14%**	**55.29**
2FSS	0.067	1.54%	20.4
Silent, CIF, SR = ±32			
6FSS	0.012	-1.14%	103.02
Proposed	**0.023**	**-1.07%**	**33.82**
2FSS	0.026	-0.61%	18.24
Stefan, CIF, SR = ±32			
6FSS	0.037	0.40%	114.49
Proposed	**0.048**	**0.98%**	**61.1**
2FSS	0.149	14.12%	21.18
Mobile, CIF, SR = ±32			
6FSS	0.013	-0.87%	103.9
Proposed	**0.018**	**-0.91%**	**51.9**
2FSS	0.019	-0.87%	17.34
Foreman, CIF, SR = ±32			
6FSS	0.039	-0.17%	112.02
Proposed	**0.05**	**0.03%**	**67.5**
2FSS	0.078	2.17%	23.68
Coastguard, CIF, SR = ±32			
6FSS	0.016	-1.22%	102.8
Proposed	**0.021**	**-1.32%**	**43.63**
2FSS	0.025	-1.36%	17.95

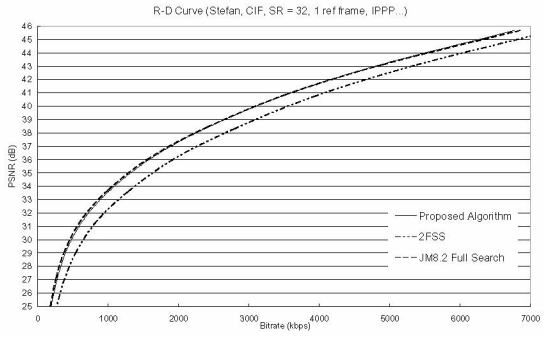


Fig. 6. R-D curve of $Stefan$ sequence.

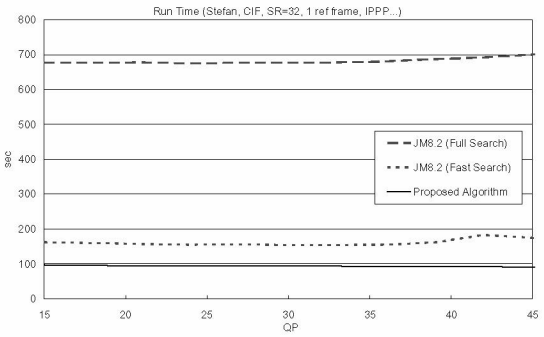


Fig. 7. Run time of $Stefan$ sequence for whole H.264 encoder.

algorithm uses only 2 initial candidates (the origin and the predictor) without expanding the moving window. From Table II, our proposed content-adaptive algorithm finds a good trade-off between quality and computation. For low motion sequences (like "Silent"), most computation is saved. On the other hand, more search points are consumed to maintain coding efficiency in high motion videos. For example, bitrate increase a lot in 2FSS but less in the proposed content-adaptive algorithm for $Stefan$ and $Foreman$ sequences. The R-D curve of $Stefan$ sequences is shown in Fig. 6. The coding efficiency of the proposed algorithm is close to that of JM8.2. A little performance degradation in low bitrate range is due to inaccurate motion vector cost. However, the average PSNR drop is less than 0.05dB as shown in Table II. In addition, the performance is much better than 2FSS algorithm. This means our proposed algorithm with moving window expansion can accurately catch the complex motion in $Stefan$.

Fig. 7 is the run time data of $Stefan$ sequence. It shows our proposed content-adaptive fast algorithm can greatly reduce the complexity of integer motion estimation and contribute to respectively 86% and 43% run time reduction compared to full search and fast search algorithm in reference software JM8.2.

V. CONCLUSIONS

We propose a content-adaptive fast algorithm for variable block-size integer motion estimation in H.264. Our algorithm can adjust the searching effort according to the motion activity, and find a good trade-off between quality and computation. About 98% searching candidates and 86% encoding time are reduced with at most 0.05dB quality drop in average compared with full search even for complex motion videos. Because of the good data reuse scheme and simple control flow, the proposed algorithm is suitable for hardware implementation.

REFERENCES

[1] Joint Video Team of ITU-T and ISO/IEC JTC 1, "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 — ISO/IEC 14496-10 AVC)," Mar. 2003.

[2] "H.264/AVC reference sofware JM8.2," July 2004.

[3] Y.-W. Huang, T.-C. Wang, B.-Y. Hsieh, and L.-G. Chen, "Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264," *Proc. IEEE Int'l Symposium on Circuits and Systems*, vol. 2, pp. 796–799, 2003.

[4] L.-M. Po and W.-C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 313–317, June 1996.

[5] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 4, pp. 369–377, Aug. 1998.

[6] M. Chan, Y. Yu, and A. Constantinides, "Variable size block matching motion compensation with applicatoins to video coding," *Proc. IEE on Communications, Speech and Vision*, vol. 137, no. 4, pp. 205–212, Aug. 1990.

[7] I. Rhee, G. R. Martin, S. Muthukrishnan, and R. A. Packwood, "Quadtree-structured variable-size block-matching motion estimation with minimal error," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 1, pp. 42–50, Feb. 2000.

[8] Z. Zhou, M.-T. Sun, and Y.-F. Hsu, "Fast variable block-size motion estimation algorithm based on merge and slit procedures for H.264/MPEG-4 AVC," *Proc. IEEE Int'l Symposium on Circuits and Systems*, vol. 3, pp. 725–728, 2004.

[9] S. Saponara and L. Fanucci, "Data-adaptive motion estimation algorithm and VLSI architecture design for low-power video systems," *Proc. IEE on Computers and Digital Techniques*, vol. 151, pp. 51–59, 2004.