May 2006, Vol.21, No.3, pp.378–382 J. Comput. Sci. & Technol.

Improved FFSBM Algorithm and Its VLSI Architecture for AVS Video Standard

Li Zhang[1,2] (张 力), Don Xie[3] (解晓东), and Di Wu[1] (吴 迪)

[1] *Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, P.R. China*

[2] *Graduate University of Chinese Academy of Sciences, Beijing 100039, P.R. China*

[3] *Grandview Semiconductor (Beijing) Corporation, Beijing 100039, P.R. China*

E-mail: {zhangli, dwu}@jdl.ac.cn; don.xie@grandviewsemi.com

Received October 24, 2005; revised March 27, 2006.

Abstract The Video part of AVS (Audio Video Coding Standard) has been finalized recently. It has adopted variable block size motion compensation to improve its coding efficiency. This has brought heavy computation burden when it is applied to compress the HDTV (high definition television) content. Based on the original FFSBM (fast full search blocking matching), this paper proposes an improved FFSBM algorithm to adaptively reduce the complexity of motion estimation according to the actual motion intensity. The main idea of the proposed algorithm is to use the statistical distribution of MVD (motion vector difference). A VLSI (very large scale integration) architecture is also proposed to implement the improved motion estimation algorithm. Experimental results show that this algorithm-hardware co-design gives better tradeoff of gate-count and throughput than the existing ones and is a proper solution for the variable block size motion estimation in AVS.

Keywords AVS, FFSBM, motion estimation, VLSI

1 Introduction

China AVS[1] has been finalized recently. It aims at the application of high definition TV coding. AVS can get a similar coding efficiency to H.264[2] with less complexity. But considering the picture size of HDTV is large, the real-time encoding of AVS is still a difficult task. The most computation-intensive part of AVS is the variable block size ME (motion estimation). One MB (macroblock) in AVS can be further divided into blocks with 4 different block sizes: 16×16, 16×8, 8×16 and 8×8. So there are totally 9 different partitions in one MB, where each block has its own ME independently. The computation complexity is quite large for the HDTV coding. To get the real-time encoding for HDTV, the hardware acceleration is necessary.

There have been many fast ME algorithms in literature, such as the NTSS (new three-step search) in [3], the BBGDS (block-based gradient descent search) in [4], the DS (diamond search) in [5], and the HEXS (hexagon-based search) in [6]. These algorithms are based on the assumption that the distribution of motion vector is center-biased. They perform well for small motion intensity and small picture size. But in many applications such as SDTV or HDTV, the picture size is large and the motion is heavier. The above algorithms are easy to be trapped into the local minimum. To avoid being trapped into local minimum, the Unsymmetrical Cross Multi Hexagon-grid Search (UMHexagonS) algorithm in [7] is proposed. The UMHexagonS consists of hybrid search patterns to keep search accuracy in all the video sequences. It has been adopted by the AVS reference software.

Most of the existing VLSI architectures for ME such as [8, 9] are array-based. They make use of the regularity of full search to get data reuse and high parallelism. But for the fast ME algorithms such as UMHexagonS, the data flow is irregular and the pipeline operation cannot be achieved, so they are unsuitable for hardware implementation. To meet the requirement of hardware, the FFSBM in [10] is proposed for the variable block size ME. It derives the SAD (sum of absolute difference) of larger blocks by merging the SAD of smaller blocks, thus the SAD can be reused to reduce complexity. Note that the search pattern of FFSBM is still in raster scan order within a rectangle search window. Thus the regular data flow makes it suitable for hardware implementation.

After all, the FFSBM is still a full search algorithm. It searches all the candidate points in a search window which has constant size. The complexity is still too high for real time video encoding. An improved FFSBM algorithm is proposed in this paper. It can not only keep the regularity of the FFSBM but also efficiently reduce the complexity according to the local motion field's feature. This paper also proposes a VLSI architecture to implement the improved FFSBM algorithm.

The following of this paper is organized as follows. Section 2 describes the proposed algorithm. The VLSI architecture is presented in Section 3. Experimental results are presented in Section 4 and finally Section 5 concludes this paper.

2 Improved FFSBM Algorithm

As Fig.1 shows, the normal FFSBM takes a merging strategy to reuse SAD of the smaller blocks. Since different block modes are overlapped in spatial domain, the

Short Paper
Supported by the National High Technology Development 863 Program of China under Grant No. 2003AA1Z1290.

SAD can be calculated only for the 8×8 block mode, a sequence of merging steps is taken to get the SAD of larger block modes. For example, when checking for one MV (motion vector) of the 16×8 mode, the SAD can be got by summing up the SAD of two 8×8 blocks. To guarantee the SAD merging process, all 9 blocks should share the same search window, i.e., the center of all blocks' search windows should be same. In AVS standard, a block uses its neighboring block's MV to predict its own MV. In FFSBM the center of the search window is the position which is pointed by the predicted MV of the 16×16 block.

It should be pointed out that the FFSBM is essentially a full search algorithm with data reuse. All the candidate MVs in a rectangle window are searched in a raster scan order. So the complexity is still too high. One method to reduce the complexity while keeping the regularity is to adaptively reduce the search window size.

2.1 Distribution of the MVD

The main idea of the proposed algorithm is to adaptively select the search window size according to the characteristic of the motion field. Since FFSBM uses the predicted MV as the center of the search window, the coding efficiency will not be lost if the search window can cover all of the block's MVD (the difference between the predicted MV and the final best MV). To get the minimal complexity, it is desirable to get the accurate upper bound of the search window size.

Since MVD is the error of MV prediction, we can regard that it complies with a zero-mean Laplacian distribution. According to [11], let the variance of a zero-mean Laplacian distribution be σ, then the probability that a value will fall within $(-3\sigma, 3\sigma)$ is about 99%, thus we can select 3σ as the upper bound. For a variable which complies with the zero-mean Laplacian distribution with the variance being σ, its mean absolute value is $\sigma/\sqrt{2}$. Based on that, we can get the upper bound 3σ by calculating the mean absolute value, that is:

$$3\sigma \approx 4 \times MeanAbsoluteValue. \tag{1}$$

The approximate value 4 instead of $3\sqrt{2}$ is used to replace the floating point multiplying by a right shifting. The *MeanAbsolutevalue* is the mean of the variable's absolute value.

Since there exists correlation between motion fields of two successive frames, the proposed algorithm uses the MVD distribution in the previous frame to predict the current MB's search window size. It should be pointed out that the distribution of MVD in one video frame varies with different local regions. Since the MV is predicted by the neighboring blocks, the distribution of MVD is decided by the complexity of the local motion field. Fig.2 shows the 16×16 blocks' horizontal MVD (absolute value) of the third frame in "Foreman". We can see that the blocks in the moving object have larger MVD because the motion here is complex, while the blocks in the still background often have zero MVD. So when deciding the search window size by MVD, the MVD distribution of a local region but not the whole frame is considered.

2.2 Proposed Algorithm

To clearly present our algorithm, we assign each MB an index (t, i, j), where t is the frame index, (i, j) is the MB's 2-D coordinate in one frame. Suppose one frame consists of $M \times N$ MBs, then $i \in [0, M-1]$ and $j \in [0, N-1]$. In $MB(t, i, j)$, the max absolute value among the 9 blocks' horizontal/vertical MVD is $MVDx(t, i, j)/MVDy(t, i, j)$. We have the following equations:

$$MeanMVDx(t,i,j) = \frac{1}{25}\Big(\sum_{m=i-2}^{i+2}\sum_{n=j-2}^{j+2} MVDx(t,m,n)\Big), \tag{2}$$

$$MeanMVDy(t,i,j) = \frac{1}{25}\Big(\sum_{m=i-2}^{i+2}\sum_{n=j-2}^{j+2} MVDy(t,m,n)\Big). \tag{3}$$

The (*MeanMVDx, MeanMVDy*) is the mean MVD value of a 5×5 MB window, it can reflect the MVD distribution of the local region that contains $MB(t, i, j)$. Based on (1), the following equations are used to decide the search window size of $MB(t+1, i, j)$.

$$WindowWidth(t+1,i,j) = 4 \times MeanMVDx(t,i,j), \tag{4}$$

$$WindowHeight(t+1,i,j) = 4 \times MeanMVDy(t,i,j). \tag{5}$$

From the above equations, it can be seen that the search window size will fluctuate according to the motion intensity of the co-located region in previous frame.

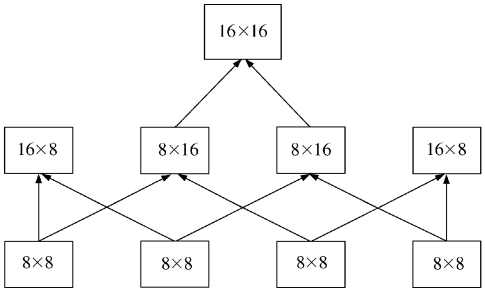

Fig.1. Merging process of SAD.

Fig.2. Local distribution of MVD in Foreman's third frame.

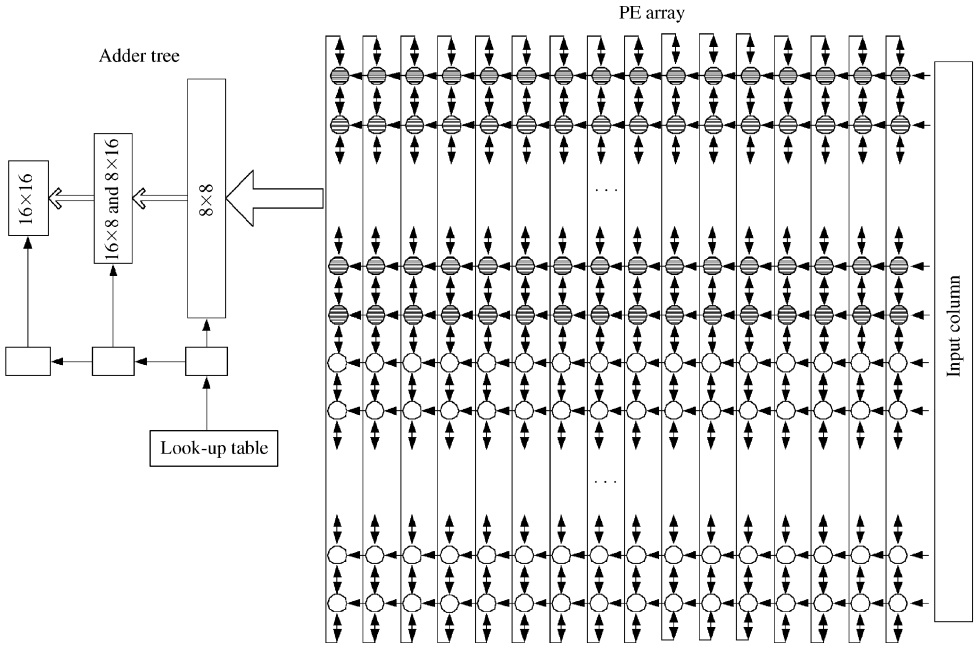

Fig.3. VLSI architecture.

Besides the above improvement for FFSBM to reduce complexity, we also make another modification for the consideration of hardware implementation. The AVS takes a rate-distortion criterion for ME: a block selects the MV which minimizes the following object function as the best MV:

$$J(m) = SAD(m) + \lambda_{motion}(m - p), \qquad (6)$$

with m being the candidate MV and p being the predicted MV, $\lambda_{motion}(m - p)$ in the above equation is the MVD cost. It represents the bit cost for the MVD. In the original FFSBM, even all the blocks use the 16×16 block's predicted MV as the window center, each block still uses its own predicted MV to calculate the MVD cost. This data-dependency of motion vector prediction is unsuitable for the parallel implementation. For example, when the four 8×8 blocks are processed in parallel, the right-bottom 8×8 block cannot get its predicted MV since its left and top blocks' MVs are not known yet. In the proposed algorithm, the p in (6) for all the 9 blocks is the 16×16 block's predicted MV, which is known before the encoding of the current MB, so the process of all the 9 blocks can be done in parallel.

3 VLSI Architecture

For the hardware implementation, the candidate MVs in the search window are checked one by one, but the process of all the 9 blocks in one MB is processed in pipeline. There are three stages in the process: 8×8, 8×16 and 16×8, 16×16. The four 8×8 blocks are processed in pipeline stage 1, then the 16×8 blocks and the 8×16 blocks are processed in pipeline stage 2, and the 16×16 block is processed in pipeline stage 3. The pipeline depth is 3. We should select an appropriate architecture for this pipeline data flow.

In recent years, a lot of VLSI architectures for ME have been proposed in literature such as [8, 9]. These architectures can be classified into two types. For the first type, the SAD of different candidate MVs are calculated in parallel and the number of PE (processing element) is decided by the search window size. For the second type, the SAD of one candidate MV is calculated in parallel and the number of PEs is decided by the block size. In the proposed algorithm, the block size is constant while the search window size is adaptive, so we would choose the architecture in the second type. The architecture in [9] is very suitable for the parallel process of the proposed algorithm. Based on [9], the new architecture like Fig.3 is designed. The architecture is platform-based, i.e., we assume that it is controlled by a processor. The size and center of search window are calculated by the processor and transferred to the proposed architecture as commands.

The architecture has two main parts: the PE array and the adder tree. The PE array calculates 256 AD (absolute value) in parallel. The adder tree will merge the 9 SADs and selects the best MV. The size of the PE array is 48(height) $\times$ 16(width). These PEs can be classified into two types. The 16×16 shadowed PEs in the above part are active ones. Each of them contains one pixel of current MB and one corresponding pixel in the search area and each PE produces one pixel's AD per cycle. The 32×16 PEs in the bottom part are inactive ones. Each of them just contains one pixel of the search area. The search area pixels in all the PEs (inactive and active) have three shifting directions: leftward, upward and downward. The upward and downward shifts are circular.

When the width and height of the current MB's

search window are $(2W+1)$ and $(2H+1)$ respectively, the total search area that is accessed by the ME is $(2W+16)\times(2H+16)$. The max values of W and H are 32 in the proposed algorithm.

If H is not more than 16, one column of search area pixels can be stored in one column of the PE array because $(2H+16)$ is no more than 48. In the first 16 cycles, the left 16 columns of the search area are input into the PE array, the input mode is like Fig.4(a). Then the data flow can be depicted as the following code:

```
for (x = 0; x < (2W + 1); x = x + 1)
{
    for (y = 0; y < 2H; y = y + 1)
    {
        if x is even
            All the search area pixel shift upwards
        else
            All the search area pixel shift downwards
        }
        if x is odd
            All the search area pixel shift leftwards, and one
            new column is input like Fig.4(a)
        else
            All the search area pixel shift leftwards, and one
            new column is input like Fig.4(b)
}
```

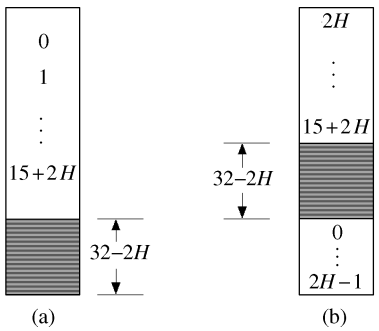

Fig.4. Two input modes of one new column.

When one column is the input as shown in Fig.4(a), the pixels are arranged in normal order from top to bottom, the bottom $(32-2H)$ positions can be padded with any data. But when the column is the input as shown in Fig.4(b), every pixel in the column is previously shifted upwards by $2H$ positions. As a result, we can see that at each cycle, the 256 active PEs always store the 256 pixels of a candidate MB in the search area, one searching point in the search window is processed per cycle.

If H is more than 16, one column of the search area cannot be stored in the PE array completely. In this case, we can divide the search window into two parts. The top $(2W+1)\times 33$ sub-window is processed firstly and then the bottom $(2W+1)\times(2H-32)$ sub-window is processed in the second pass. There will be 16 extra cycles inserted between the two passes to move the search area pixel into the PE array. Considering the fact that horizontal motion is much heavier than vertical motion for natural video, the needed H is rarely larger than 16. The experimental results in the next section will show that. So we truncate the height of the PE array to get a better tradeoff between gate count and throughput.

Each node in the adder tree calculates (6) for one block. The predicted MV p in (6) is the same for all the nodes in the adder tree, and all the nodes (such as the 4 nodes for the 8×8 block) in the same level of adder tree share the same candidate MV m. Therefore, all the nodes in the same level can share the same MVD cost. Furthermore, the MVD cost value can be reused by different levels. For example, the MVD cost which is valid for the 8×8 level at cycle T will be valid for the 8×16 and 16×8 level in cycle $T+1$. In our design, we use a look-up table to generate the value of MVD cost. A delay chain is used to serially shift the value to each level of the adder tree. At each level the value will be broadcast to all the nodes.

4 Experimental Results

To evaluate the performance of the improved FFSBM algorithm, we have tested it in two typical 720P sequences with different motion characteristics: Harbour (static camera, complex motion) and Night (fast horizontal motion). The proposed algorithm is compared with the original FFSBM algorithm. The search window size is $[-32,32]$ for the FFSBM. The coding sequence is "IPPPP". There are totally 150 frames in each sequence. The first P frame for the proposed algorithm still takes constant search window size $[-32,32]$. Only one reference frame is used. 4 QPs are selected as 32, 36, 40, and 44. The experiment is done on the AVS reference software RM52C[12]. The experimental results are as follows.

Table 1 has illustrated the speedup ratio of the improved FFSBM compared with the original FFSBM. The speedup ratio is calculated by comparing the full search window size to the actual search window size. We can see that the speedup ratio is larger. From Figs.5 and 6, it can be seen that the coding efficiency of the improved FFSBM algorithm is nearly the same as that of the original FFSBM. As we have mentioned in Section 3, when the height of the search window is more than 16, it needs 16 extra cycles to fill the PE array before the second pass. This will interrupt the pipeline operation. Fortunately, Table 1 has illustrated that the chance of H exceeding 16 is relatively small. Thus limiting on the height of the PE array is reasonable to reduce the gate count of the chip.

Table 1. Speedup Ratio and Exceeding Percentage

		Speedup ratio	Exceeding percentage
Harbour	32	44.02	2.54%
	36	40.81	3.16%
	40	31.47	5.41%
	44	14.51	21.43%
Night	32	16.97	4%
	36	15.21	4.69%
	40	11.02	7.82%
	44	6.15	19.39%

An interesting point can be found in Table 1. When quantization step is smaller, the speedup ratio is larger

and the exceeding percentage is smaller. This can be explained as follows. When the quantization step is smaller, the quality of the reconstructed reference frame is better, thus the ME can get a more accurate result and the predicted MV will be nearer to the real best MV. So the range of MVD will become smaller.

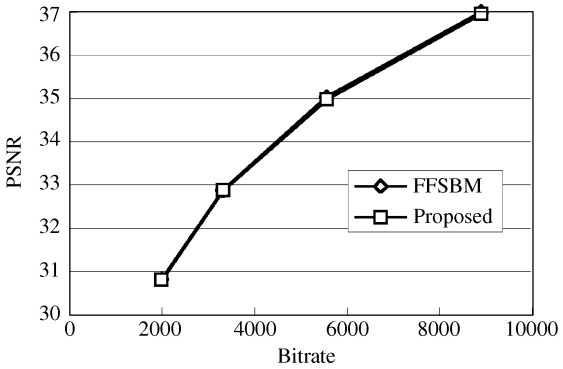

Fig.5. RD curve of "Harbour".

Table 2. Performance Comparisons on the Two Architectures

	Proposed	[13]
Search window size	$[-32, 32]$	$[-8, +7]$
Cycle per MB	4257 (max)	4496
Technology	0.18μm	0.13μm
Gate count	212K	108K
Clock frequency	150MHz	100MHz

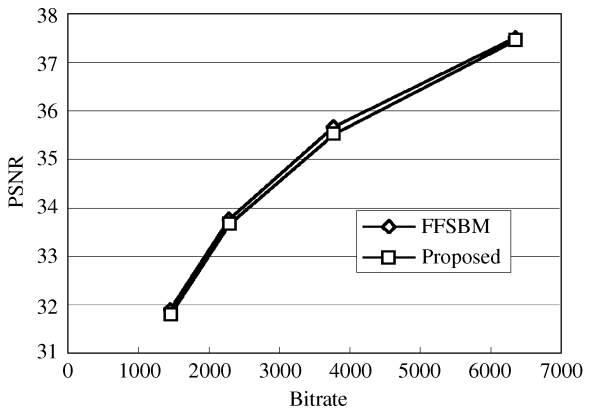

Fig.6. RD curve of "Night".

We describe the proposed architecture by Verilog-HDL and synthesize it using 0.18μm standard CMOS cell library by Synopsys Design Compiler. The clock frequency is 150MHZ and the total gate count is 212K. The simulation is done at the register transfer level (RTL), and the simulation platform is Synopsys VCS. A 32-bit RISC processor is selected to calculate and transfer the parameters to the proposed architecture.

There have been previous architectures for the variable block size ME, such as the 1-D architecture in [13]. The comparison result is given in Table 2. An exact comparison is difficult because these architectures are designed for different algorithms and use different technologies. But we still can see that the proposed architecture has a larger search window size to get better coding efficiency. The max cycle count for one MB is $(2 \times 32 + 1) \times (2 \times 32 + 1) + 32 = 4257$ cycles. The cycle count can be further reduced according to Table 1. Even if the gate count is larger, the proposed architecture has better tradeoff among silicon area, throughput and coding efficiency.

5 Conclusions

An improved FFSBM algorithm is proposed in this paper. The complexity is adaptively reduced according to the motion intensity while the lost coding efficiency is marginal. The proposed architecture uses a 2-D PE array to get high throughput, and the size of PE array is carefully selected to reduce the gate count. An efficient method for the MVD cost calculation and the implemented architecture are also proposed. The experimental results show that this algorithm and hardware co-design is a right solution for AVS's variable-block size ME.

References

[1] Audio Video Coding Standard Working Group of China. Video coding standard 1.0. November, 2003, Beijing, China.

[2] Draft ITU-T recommendation and final draft international standard for joint video specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC). ITU-T SG16 Q.6 (VCEG) and ISO/IEC JTC 1/SC 29/WG 11 (MPEG), JVT-G050r1, May 2003, Pattaya.

[3] Li R, Zeng B, Liou M L. A new three-step search algorithm for block motion estimation. *IEEE Trans. Circuits Syst. Video Technol.*, Aug. 1994, 4(8): 438–442.

[4] Liu L K, Peig E. A block-based gradient descent search algorithm for block motion estimation in video coding. *IEEE Trans. Circuits Syst. Video Technol.*, Aug. 1996, 6(8): 419–423.

[5] Tham J Y, Ranganath S, Ranganath M, Kassim A A. A novel unrestricted center biased diamond search algorithm for block motion estimation. *IEEE Trans. Circuits Syst. Video Technol.*, Aug. 1998, 8(8): 369–377.

[6] Ce Zhu, Xiao Lin, Lap-Pui Chau. Hexagon-based search pattern for fast block motion estimation. *IEEE Trans. Circuits Syst. Video Technol*, May 2002, 12(5): 349–355.

[7] Zhibo Chen, Zhou Peng, Yun He. Fast integer pel and fractional pel motion estimation for JVT. JVT-F017, December 2002.

[8] Yang K M, Sun M T, Wu L. A family of VLSI design for the motion compensation block matching algorithm. *IEEE Trans. Circuits Syst. Video Technol.*, Oct. 1989, 36(10): 1317–1325.

[9] K Nuno Roma, Leonel Sousa. A new efficient VLSI architecture for full search block matching motion estimation. In *VLSI-SOC*, Montpellier, France, 2001, pp.253–264.

[10] Chuan-Yu Cho, Shiang-Yang Huang, Jia-Shung Wang. An embedded merging scheme for H.264/AVC motion estimation. In *International Conference on Image Processing,* Barcelona, Spain, September 14–17, 2003, 1: 909–912.

[11] Sheng Zhou, Xie Shiqian, Pan Chengyi (eds.). Probability and Statistics. Higher Education Press, 1997.

[12] AVS reference software RM52C.

[13] SWee Yeow Yap, John V Mcanny. A VLSI architecture for advanced video coding motion estimation. In *IEEE 14th Int. Conf. Application-specific Systems, Architectures and Processors*, Hague, Netherlands, June 24–26, 2003, pp.293–301.