IMPROVED FFSBM ALGORITHM AND ITS VLSI ARCHITECTURE FOR VARIABLE BLOCK SIZE MOTION ESTIMATION OF H.264

Li Zhang[1,2], *Wen Gao*[1,2]

[1] Institute of Computing Technology, Chinese Academy of Science
[2] Graduate School of Chinese Academy of Science
{zhangli,wgao}@jdl.ac.cn

ABSTRACT

The video coding standard H.264/AVC has adopted variable block size motion estimation to improve coding efficiency, which has brought heavy computation burden. The FFSBM (fast full search block matching) algorithm has been proposed to reduce the complexity. This paper proposes an improved FFSBM to adaptively reduce the complexity of FFSBM according to the degree of motion activity. A modular 2-D VLSI architecture to implement the improved algorithm is also proposed, the size of the PE array is carefully selected to reduce the gate count. Experimental result shows that this algorithm-hardware co-design gives better area/throughput tradeoff than the existing ones and is a proper solution for H.264's variable block size motion estimation.

1. INTRODUCTION

An important coding tool of video coding standard H.264 [1] is the variable block size matching algorithm for the ME (motion estimation). As Fig.1 shows, a macroblock can be partitioned into sub-blocks with variable size, each sub-block will search its own best matched sub-block in the previous frame. This variable block size motion estimation brings performance improvement and great computation complexity also. To reduce computation complexity, an intuitive idea is to get the SAD (sum of absolute difference) of large sub-blocks by merging the SAD of the small sub-blocks. This idea is called the FFSBM (fast full search block matching) and has been adopted in the reference software of H.264 [1]. The modularity of FFSBM makes it easy to be implemented by hardware. But the full search range is not necessary at most cases, this will waste computation power and adds the silicon area. In this paper, an improved FFSBM algorithm is proposed and a proper hardware architecture is also implemented for the proposed algorithm.

The following of this paper is as follows: section 2 describes the improved algorithm, the VLSI architecture is presented in section 3, the implementation results are in section 4 and conclusions are in section 5.

2. IMROVED FFSBM ALGORITHM

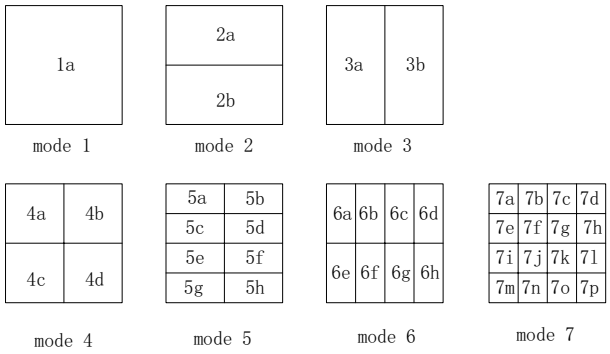

Fig.1 Seven partition modes of one macroblock

In H.264, there are 7 partition modes for a macroblock like Fig.1 shows, so there are totally 41 sub-blocks in one macroblock, the normal FFSBM takes a merging strategy to get the SAD of these sub-blocks, the SAD of 16 4x4 sub-blocks are calculated directly, the larger blocks' SAD are merged in a hierarchical pattern, such as 6a is added up by 7a and 7e, 4c is added up by 5e and 5g, let the search range be R, for the ME of a sub-block, the candidate MVs (motion vectors) are constrained in a (2R+1)x(2R+1) window, the center of the window is the predicted MV. To guarantee the SAD merging process, all 41 sub-blocks share the same search-window, the center of the window is the median predicted MV of the 16x16 block 1a.

We can see the complexity of FFSBM is still too high for real-time requirement. Many fast ME algorithms have been proposed. But they are not efficient for the hierarchical feature of FFSBM. A very plain but efficient way to reduce the complexity is to reduce the size of search window. We know that the distribution of MV in real-world video is center-biased, so a constant large

search window is not always necessary. The window size should be adaptively decided.

When we design the improved FFSBM algorithm, several principles are considered:

a) If the search window contains all the 41 best MVs of the sub-blocks, there will be no search accuracy loss. So the window size should be adjusted according to the largest MVD (motion vector difference between the final MV and the predicted MV).
b) To give an easy hardware implementation. It is better to adjust the search window size at the frame level, for the macroblock level adjustment inserts extra cycles between macroblocks.
c) In normal full search ME algorithm, the shape of the search window is a square which has equal width and height. But it is a well-known fact that usually the horizontal motion is heavier than the vertical motion in real world video. So we should adjust the search window's width and height separately.

Let the (MVdx, MVdy) be the MVD of one sub-block, according to consideration a, we will make decision based on the largest MVD of the 41 sub-blocks, let it be (MaxMVdx, MaxMVdy). Because of the center biased character of the MV, we suppose the distribution of MaxMVdx and MaxMVdy can be modeled by a Laplacian distribution, which has a peak value at zero. According to the consideration c, the MaxMVdx and MaxMVdy should obey two independent Laplacian distributions respectively.

Due to the continuity of motion field, we suppose the distribution of current frame's MVD is similar to that of the previous frame. Let the variance of a zero-mean Laplacian distribution be σ, the probability that a value will fall within $(-3\sigma, 3\sigma)$ is about 99%. But the calculation of variance is troublesome for a hardware implementation. Since the expected mean absolute value of a signal with Laplacian distribution and zero-mean is $\sigma/\sqrt{2}$, we can get the variance by calculating the expected mean absolute value, that is:

$$3\sigma = 3\sqrt{2} * MeanAbsoluteValue \approx 4 * MeanAbsoluteValue \quad (1)$$

Before encoding one frame we should decide the search window's width and height, if this frame is the immediate one after an I frame, the search window will take the full size, else suppose the mean absolute value of the previous frame's MaxMVdx(MaxMVdy) is AveWidth(AveHeight), then we have the following rule:

$$CurrentSearchWidth = 4*AveWidth + 1 \quad (2)$$

$$CurrentSearchHeight = 4*AveHeight + 1 \quad (3)$$

Note that if the resulted width or height is larger than the predefined max range, then it is truncated to the max range.

In H.264, the ME of a sub-block selects the MV that minimizes the following object function:

$$J(m, \lambda_{motion}) = SAD(m) + \lambda_{motion}(m - p) \quad (4)$$

with m being the candidate MV and p being the predicted MV, the second item in the right of the above the equation represents the bits cost for the MVD. In the original FFSBM, even the predicted MV used as the search window center is the same to all the sub-blocks, each sub-block still uses their own predicted MV to calculate the MVD cost, in H.264 the MV is median predicted by the top, left, top-left neighbors' MV, this sequential data dependency is not suitable for the hardware implementation. In the proposed algorithm, all the sub-blocks' predicted MV for the MVD cost is the predicted MV of 16x16 block 1a.

To test the coding efficiency of the improved FFSBM, we compare that with the FS(full search) ME algorithm (all the sub blocks take their own full search), the experiment is done on the platform of reference software JM7.3, the QP is 32, only one reference frame is used, the max search range is 16 for both for height and width, RDO and CAVLC is used, four CIF sequences are chosen as the test cases, only the first frame is I frame, and the following are P frames.

Table 1 Performance comparison

	FS		Improved FFSBM		
	PSNR	Bit rate (Kb/s)	PSNR	Bit rate (Kb/s)	Average window size
Foreman	33.50	229.01	33.45	231.41	314.47
Coast guard	31.48	514.31	31.46	516.65	103.04
Flower	30.99	946.19	30.97	949.70	50.98
Mobile	30.14	892.14	30.12	891.71	151.02

In the FS, the search window's size of each sub-block is constantly (2*16+1)*(2*16+1) = 1089, total computation will be seven times of that, we can see that in the improved FFSBM, the complexity is greatly reduced while the performance loss is minor.

3. VLSI ARCHITECTURE

The modularity of the improved FFSBM algorithm makes it easy to be implemented by VLSI. In recent years, a lot of VLSI architectures for ME have been proposed in literature such as [2],[3]. These architectures can be classified into two types, for the first type, the AD (absolute difference) operations of one SAD are done in parallel, so the number of PEs increases with the block matching size, for the second type the AD operations of different candidate position's SADs are done in parallel, so the PE number increases with the search range. Since in the improved FFSBM the search range is adaptive, we would choose the architecture in the first type. So the

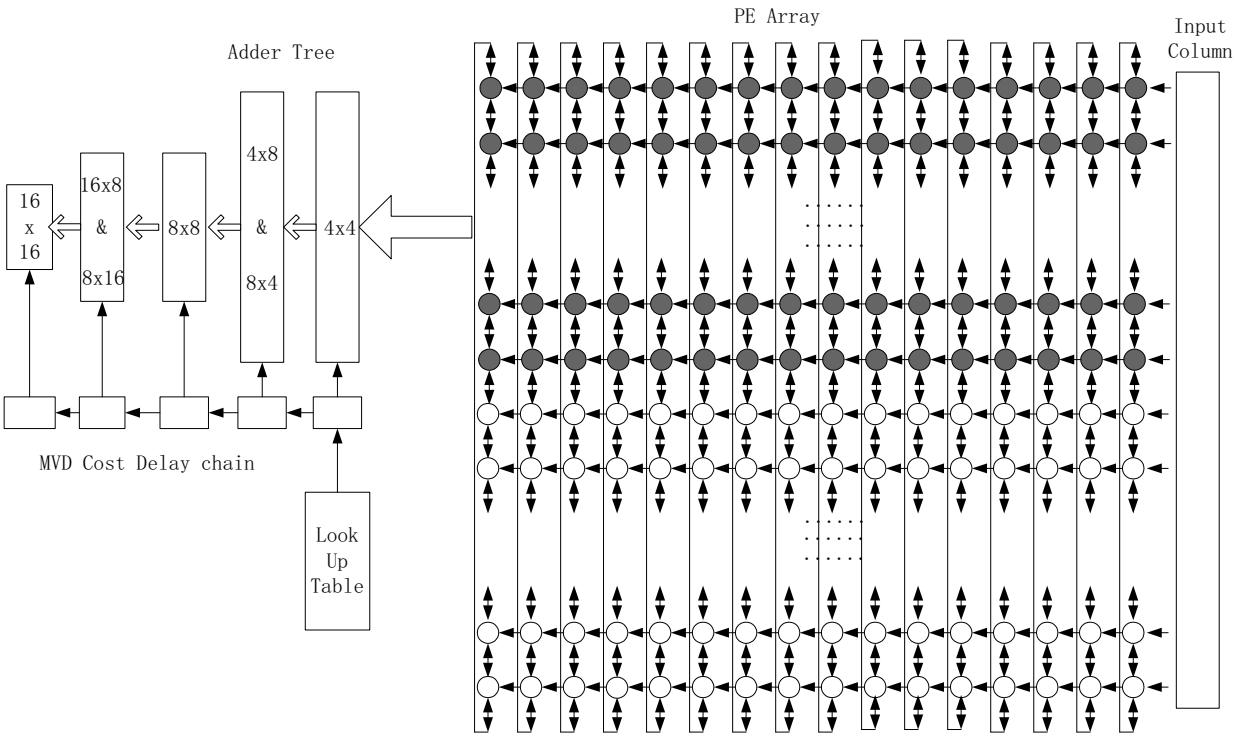

Fig. 2 VLSI Architecture

number of PE is constant and the computation time fluctuates with the degree of motion activity.

The architecture in [3] is very suitable for the 2-D data transfer of the improved FFSBM. Based on it the new architecture like Fig.2 is designed. It has two main parts: the PE array and the adder tree. The PE array calculates 256 ADs of 16 4x4 blocks in parallel. The adder tree will merge the 41 SADs and selects the best MV.

The size of the PE array is 16x32. These PEs can be classified into two types, the 16x16 shadowed PEs in the above part are the active ones, each of them contains one pel of current macroblock and one corresponding pel in the search area. Thus each cycle the 256 ADs of 16 4x4 sub-blocks can be produced in parallel. The 16x16 PEs in the bottom part are inactive ones, each of them just contains one pel of the search area, the search area pels in all the PEs (inactive and active) have three shifting directions: leftwards, upwards and downwards, note that the upwards and downward shifts are circular.

In the improved FFSBM, the current search width is (2W+1) and height is (2H+1), the total search area is (2W+16)x(2H+16), let the max value of W and H be 16, if H is no lager than 8, one column of search area can be stored in one column of the PE array. 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.3-a. Then the data flow can be depicted as following code:

```
for(x= 0 ; x <  (2*W+1) ; x= x+1)
{
 for(y= 0 ; y <  2*H ; y = y+1)
 {
   if x is even
     All the search area pel data shift upwards
   else
    All the search area pel data shift downwards
 }
 if x is odd
  All the search area pel data shift leftwards, one new
  column is input like Fig. 3-a
 else
  All the search area pel data shift leftwards, one new
  column is input like Fig. 3-b
}
```

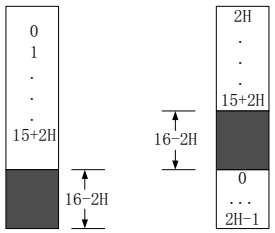

Fig.3-a Fig.3-b

When one column is input like Fig. 3-a, the pels are arranged in normal order from top to bottom, the bottom (16 - 2H) pels can be padded with any data. But when the column is input like Fig. 3-b, every pel in the column is

previously shifted upwards by 2H positions. As a result, we can see that at each cycle, the above 256 active PEs always store the 256 pels of the current candidate macroblock in the search area. The (2W+1)x(2H+1) candidate positions are iterated in (2W+1)x(2H+1) cycles. The search order is from top to bottom for the even columns of the search window, and from bottom to top for odd columns.

If H is lager than 8, then one column can not be stored in the PE array completely for (2H+16) now is larger than 32, we can divide the search window into two parts, the top (2W+1)x17 sub window is processed firstly, then the bottom (2W+1)x(2H-16) sub window is processed in the second pass, obviously this will insert extra cycles between the two passes, there are two reasons that we set the height of inactive PE array to be 16 not 32: firstly, due to horizontally-biased character, the needed H is rarely larger than 8, secondly, with the highly parallel architecture and reduced search range, the throughput is not the bottleneck, but the area of the 2-D architecture is quite large, to get a better tradeoff, we decrease the area on the condition that the throughput is enough. Before a search pass there are 16 extra cycles to input the search area data, when H is no larger than 8, the total needed cycles for a marcroblock is (2W+1)x(2H+1)+16, else the total cycles is (2W+1)x(2H+1)+32. The average cycles needed per macroblock of four CIF sequences is in the Table 2.

Table 2 Resulted throughput

	Percentage of second pass	Average cycles per macroblock
Foreman	28.4%	335
Coastguard	3%	120
Flower	0.4%	68
Mobile	0.7%	167

We use a look up table to store the MVD cost $\lambda_{motion}(m-p)$. In each node of the adder tree, one sub-block's best MV is selected with the criterion of equation 4, since all the nodes use the same predicted MV, the MVD cost of the function is only decided by the current candidate MV, in the adder tree, all the nodes in the same level share the same candidate MV, so the MVD cost value can be broadcast to all the nodes in the same level of adder tree. Because of the pipelining operation, the MVD cost value which is valid for level0 at cycle T will be valid for the level1 in cycle T+1, so a delay chain is used to serially shift the MVD cost value to each level of the adder tree.

4. EXPERIMENTAL RESULTS

We describe our design by Verilog-HDL and synthesize it using 0.18 standard CMOS cell library by Synopsys's Design Compiler, the clock frequency is 150 MHZ, the total gate counts is 174K.

There have been previous architectures for the normal FFSBM algorithm, a 1-D architecture is given in [4], the comparison result is given in Table 3.

An exact comparison is difficult for these architectures are designed for different algorithms and use different technology, but we still can see that the proposed architecture has a larger and more flexible search window. For the sequence "foreman" with fast motion, the average cycle counts per macroblock needed is still only 335, much less than [4] , Even the gate count is larger, the proposed architecture has better tradeoff between silicon area , throughput and coding efficiency.

Table 3 Performance comparison

	Proposed	[4]
Search window size	33x33 (max)	16x16 (constant)
Average Cycles for one macroblock	Adaptive	4496
Technology	0.18 μm	0.13 μm
Gate count	174K	108K
Clock frequency	150MHZ	100MHZ

5. CONCLUSIONS

An improved FFSBM algorithm is proposed in this paper. By fully exploiting the character of the motion in real-world video, the search window of FFSBM is adaptively reduced. The coding efficiency loss is minor. The proposed architecture uses a 2-D PE array to get high throughput, and size of PE array is carefully selected to reduce the area. The experimental result shows that this algorithm and hardware co-design has better performance than the existing architectures. The proposed design is a proper solution for H.264's variable-block size ME.

6. REFERENCES

[1] JVT Reference Software JM 7.3, http://iphome.hhi.de/suehring/tml/download/

[2] K.M.Yang , M.T.Sun, and L. Wu, "A family of VLSI design for the motion compensation block matching algorithm", IEEE Transactions on Circuit and System for Video Technology, vol.36, no.10 pp.1317-1325,Oct.1989.

[3] Nuno Roma, Leonel Sousa: "A New Efficient VLSI Architecture for Full Search Block Matching Motion Estimation" VLSI-SOC, pp:253-264, 2001.

[4] SWee Yeow Yap, John V Mcanny, "A VLSI architecture for advanced video coding motion estimation"IEEE Proceeding of Application-Specific Systems, Architecture, and Processors. 2003.