LOW-PASS FILTER BASED VLSI ORIENTED VARIABLE BLOCK SIZE MOTION ESTIMATION ALGORITHM FOR H.264

*Zhenyu Liu**

Kitakyushu Foundation for the Advancement
of Industry Science and Technology
Kitakyushu, 808-0135, Japan

Yang Song, Takeshi Ikenaga, Satoshi Goto

The Graduate School of IPS
Waseda University
Kitakyushu, 808-0135, Japan

ABSTRACT

In this paper, a fast motion estimation algorithm, which is friendly to VLSI hardware implementation is proposed. This algorithm has such features: First, through "Haar" low-pass filter based subsampling, the computation complexity at each search position is reduced to about 25% of the original algorithm; Second, one modified motion vector prediction is provided to eliminate the data dependence among sub-partitions in the same macro block(MB). Based on this approach, parallel processing for variable block size motion estimation(VBSME) with integer pixel accuracy can be realized; Third, one "adaptive sub-search window" scheme is proposed to further reduce computation cost and it also can facilitate reference frame data reusing to reduce memory transfer from the external RAM to the on-chip SRAM. The proposed VBSME algorithm is very suitable for parallel VLSI implementation.

1. INTRODUCTION

H.264/AVC is the newest international video coding standard, which can provide much better peak signal-to-noise (PSNR) and visual quality [1]. In H.264/AVC, motion estimation (ME) is conducted on different blocks sizes named as VBSME. During VBSME, all the block sizes inside one MB are processed and the block mode with the best R-D cost is then chosen. Compared with previous fixed block size ME process, VBSME can achieve higher compression ratio and better video quality, but it accounts for more than 50% computation cost of the encoding algorithm. Therefore, hardware acceleration is a must for real-time applications. How to reduce the processing time and power dissipation of VBSME is a vital issue for real-time encoder and this needs hardware-software co-optimization. Computation complexity reduction, simple control logic and regular memory access are three important issues for hardware architecture design. Many fast motion estimation algorithms have be provided to reduced the algorithm computation through reducing search positions. These algorithms are efficient for fixed block ME. But for H.264, because MB is partitioned into many blocks and each block has its own search path, it is hard for fast ME to realize SAD reusing scheme as full search (FS) algorithm [2]. Partition mode in H.264 multiplies the computation complexity of fast ME. In real-time applications, worst-case complexity of a ME method becomes more important than its average complexity. Despite improving average complexity, fast ME techniques won't have much better worst-case complexity than FS with SAD reuse. Moreover, the control logic and memory access of fast ME are complex and irregular, so they are not suitable for hardware designs. In fact, FS is more preferred in hardware ME accelerator designs, because (1) Process elements (PE) are scheduled to work in parallel and fully utilized, so its throughput is in direct ratio to its PE number; (2) Its memory access and control logic are regular and simple; (3) It can realize SAD reusing scheme in H.264, so the computation complexity is almost reduced to the level of single block type case. In order to apply FS to built parallel hardware accelerator, two problems must be resolved. First, the data dependency in one MB must be eliminated to make parallel processing feasible. Second, the computation complexity must be reduced to save its hardware cost, power dissipation and processing latency.

In this paper, we provide a low-pass filter based subsampling algorithm and sub-search window approach to reduce the computation. In order to eliminate the data dependency, one simplified motion vector prediction (SMVP) algorithm is proposed to facilitate parallel processing. The details of this algorithm are described in Section 2. The simulation results in Section 3 show that there is little quality loss, despite a major reduction in complexity. Conclusions are given in Section 4.

2. LOW-PASS FILTER BASED VLSI ORIENTED VBSME ALGORITHM

2.1. Low-Pass Filter Based Subsampling Algorithm

Subsampling is one effective approach for arithmetic computation reduction. During ME procedure, the search area data and current MB are both subsampled in vertical direction [3], so 50% arithmetic computation can be reduced. What is more, because this approach does not increase the complexity of control logic and memory access, it is widely applied in hardware accelerator. But directly subsampling algorithm also brings some problems. First it causes signal distortion. For 1-D signal $x(n)$, its frequency spectrum is denoted as $X(e^{j\omega})$. The subsampled signal $y(n)$ and its frequency spectrum $Y(e^{j\omega})$ can be represented as Eq. 1 and Eq. 2.

$$y(n) = x(2n) \tag{1}$$

$$Y(e^{j\omega}) = \frac{1}{2}[X(e^{j\frac{\omega}{2}}) + X(e^{j(\frac{\omega}{2}-\pi)})] \tag{2}$$

This procedure is also clearly illustrated in Figure 1. We can see that the frequency aliasing after subsampling causes the distortion of the signal.

The second drawback of subsampling is that small blocks are more prone to be trapped into false optimal points. In H.264, there

*Thanks to Kitakyushu knowledge-based cluster project for the Japanese Ministry of ECSST funding.

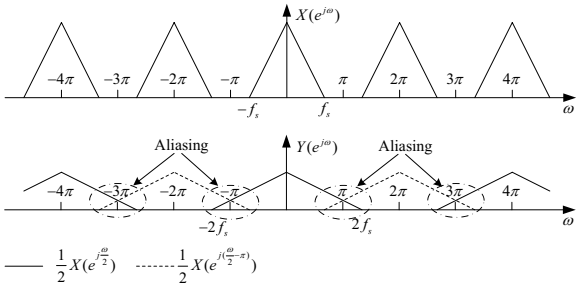

Fig. 1. Frequency Aliasing after Subsampling

exists small size blocks, such as 4×4 and 4×8 blocks. After subsampling, most characters of these small blocks are lost. For example, if the picture is decimated in both vertical and horizonal directions, each 4×4 block has just 4 pixels. During ME, 4-pixel data character makes 4×4 blocks more prone to false optimal points.

For one image, low-frequency signals contain more information than high frequency signals. Moreover, after motion-compensated prediction, the low-frequency DCT coefficients of the residual data are more significant, because high frequency coefficients are always discarded during quantization procedure. Based on this hypothesis, we could applied subsampling approach on the low-frequency signals of the video sequence. In details, the reference picture and the current picture first pass through a low-pass filter to get low frequency signals, which are labelled as low-frequency reference (LR) and low-frequency current (LC). Subsampling ME algorithm is processed on these low-frequency pictures. Because the high frequency signals are filtered, the frequency aliasing can be greatly alleviated. Moreover, through the low-pass filter, the word-length of each pixel is increased, so more characters are preserved. Consequently, Low-pass filter based subsampling algorithm can efficiently resolve those problems caused by directly subsampling.

In order to reduce the computation burden, we apply "Haar" low-pass filter, which means that each pixel in the generated picture is the sum of neighboring four pixels in the source picture. For example, the pixel $LR(i,j)$ can be expressed as Eq. 3.

$$LR(i,j)=R(i,j)+R(i+1,j)+R(i,j+1)+R(i+1,j+1) \quad (3)$$

In SAD calculation, LR and LC are subsampled in both horizonal and vertical directions. The SAD computation of a $B_x\times B_y$ size block with the (l_x,l_y) coordinate in the frame can be expressed as Eq. 4, where (u,v) represents the motion vector. So, at each search position, the processed pixel number is reduced to 25% of full-search algorithm. Different from wavelet based ME, we do not decimate the LR frame, so (u,v) traverses each search position in the search window. Though wavelet based ME can further reduce the computation through decreasing search positions, it has two drawbacks for VBSME in H.264: (1) The search step based on decimation is 2-pixel. After coarse search, each block must search around its coarse search result to find its 1-pixel accuracy optimal position. Because there are 41 blocks in one MB, it is not easy to realize one efficient hardware for this refining search. (2) Decimating both LR and LC causes shift variance problem [4]. This problem is more serious for small size blocks.

$$SAD(u,v)=\sum_{i=0}^{B_x/2-1}\sum_{j=0}^{B_y/2-1}|LC(l_x+2i,l_y+2j) - LR(l_x+2i+u,l_y+2j+v)| \quad (4)$$

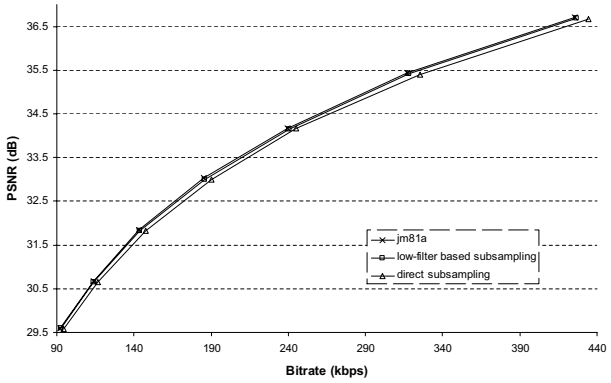

Fig. 2. RD performances for Foreman CIF sequence at 30fps

In order to investigate the low-pass filter based subsampling VBSME algorithm, it is compared with FS and directly subsampling algorithm. The first 255 frames in "foreman" CIF sequence are used in the simulation. The test conditions are I-P-P-P..., CAVLC, Hadamard transform, 32×32 search range, R-D optimization and 5 reference frames. The RD-curves are shown in Figure 2. Compared with FS algorithm, directly subsampling algorithm has about 0.15dB PSNR loss. In contrast, the subsampling based on "Haar" low-pass filter has the similar performance as FS algorithm.

If the picture width is W, the picture hight is H and the search range is $M\times N$, in the original FS algorithm for each reference frame, the absolute difference (AD) number is $W\times H\times M\times N$ and the addition operation (ADD) number is $W\times H\times 265\times M\times N/256$. In our algorithm, the ADD number for low-pass filter of each reference frame is $4\times W\times H$. After subsampling in vertical and horizonal directions, for each search position's SAD calculation, the AD number is 64 and the ADD number is 73. In SAD cost processing, the AD number is $(W\times H\times M\times N\times 64)/256$. Taking the low-pass filter processing into account, the total ADD operation is $W\times H\times(73\times M\times N/256+4)$. If search rang is 32×32, compared with FS VBSME, 75% AD and 72% ADD operations can be saved by our algorithm during SAD cost computation.

2.2. Simplified Motion Vector Prediction Algorithm

In the reference encoder software, RD cost is the sum of SAD and encoding cost of motion vector difference (MVD). MVD calculation incurs data dependency among not only adjacent MBs but also sub-partitions within one MB. This makes parallel processing unfeasible. To eliminate the data correlation within MB, the approach in reference [2] is directly omitting the MVD cost during the integer pixel precision VBSME, because SAD cost is much more important than MVD cost. Based on this modified algorithm, high performance 2-D [2] and 1-D [5] array processing VBSME architectures are proposed. Unfortunately, this scheme is not suitable for our algorithm. First, the low-pass filter makes the video lose its high frequency signals. Second, subsampling incurs the frequency aliasing noise. Third, after subsampling, feature loss is serious for small blocks,

which makes they prone to be trapped in false optimize search positions. Through experiments, we find that without MVD cost our algorithm has 0.3dB-0.6dB PSNR loss.

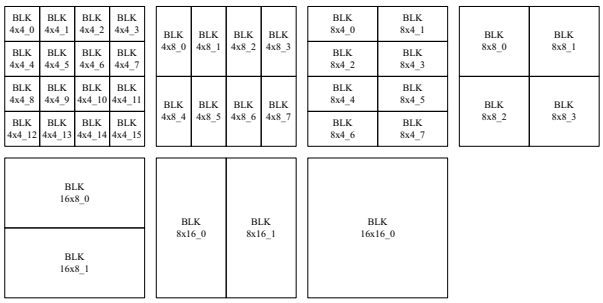

Fig. 3. Lables of Macroblock and sub-macroblock partitions

One modified MVP algorithm, which avoids the data dependency within MB, is provided in this paper to facilitate VBSME parallel processing. In order to clarify the algorithm, 41 blocks in one MB are labelled, as shown in Figure 3. Considering pipeline processing and hardware complexity, we just use MVs of the four 4×4 blocks in the upper neighboring MB to derive MVPs of all blocks in current MB, as shown in Figure 4.

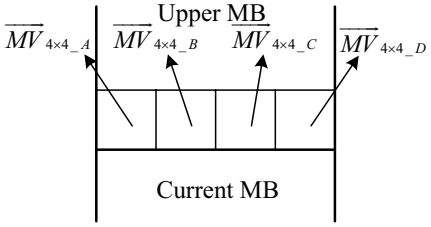

Fig. 4. Motion Vector Prediction Data Dependency

If "Upper MB" exits, the MVP for blocks "4×4_0", "4×4_4", "4×4_8", "4×4_12", "4×8_0" and "4×8_4" is $\overrightarrow{MV}_{4\times4\_A}$; the MVP for blocks "4×4_1", "4×4_5", "4×4_9", "4×4_13", "4×8_1" and "4×8_5" is $\overrightarrow{MV}_{4\times4\_B}$; the MVP for blocks "4×4_2", "4×4_6", "4×4_10", "4×4_14", "4×8_2" and "4×8_6" is $\overrightarrow{MV}_{4\times4\_C}$; the MVP for blocks "4×4_3", "4×4_7", "4×4_11", "4×4_15", "4×8_3" and "4×8_7" is $\overrightarrow{MV}_{4\times4\_D}$; the MVP for blocks "8×4_0", "8×4_2", "8×4_4", "8×4_6", "8×8_0", "8×8_2" and "8×16_0" is $(\overrightarrow{MV}_{4\times4\_A}+\overrightarrow{MV}_{4\times4\_B})/2$; the MVP for blocks "8×4_1", "8×4_3", "8×4_5", "8×4_7", "8×8_1", "8×8_3" and "8×16_1" is $(\overrightarrow{MV}_{4\times4\_C}+\overrightarrow{MV}_{4\times4\_D})/2$; the MVP for blocks "16×8_0", "16×8_1" and "16×16_0" is $(\overrightarrow{MV}_{4\times4\_A}+\overrightarrow{MV}_{4\times4\_B}+\overrightarrow{MV}_{4\times4\_C}+\overrightarrow{MV}_{4\times4\_D})/4$. Otherwise, MVPs for all blocks are set as zero.

2.3. Adaptive Sub-Search Window

In H.264 encoder reference software, each block uses its MVP as search window center. That means each block has its own dedicated search window. In the view point of the hardware implementation, it is highly preferred to the uniform search window for all blocks in MB, because SAD reusing scheme can be fully utilized. In order to simplify the control logic and the data transfer, rectangle search area is preferred. In reference [2], $[0,0]$ is adopted as uniform search window center because this approach can facilitate data reusing to reduce data transfer. For those sequences which have large MVs, the real MVs will be out of the range of search window. Certainly, we can enlarge the search window to resolve this problem, but this approach incurs the increase of computation complexity, internal memory access and power dissipation. These items increase in direct ratio to the search area.

In order to resolve this problem, we provide an adaptive sub-search window scheme. First we define the search area frame for each MB. The search frame center is $[0,0]$, so its overlapped area for adjacent MBs can be reused as reference [2]. The search frame width is denoted as F_w and its hight is denoted as F_h. F_w and F_h could be defined large enough to cover the real MV. VBSME is just performed in a sub-area in the search frame, which is derived from $\overrightarrow{MV}_{4\times4\_A}$ $\overrightarrow{MV}_{4\times4\_B}$ $\overrightarrow{MV}_{4\times4\_C}$ and $\overrightarrow{MV}_{4\times4\_D}$. The bottom left corner of this sub-area is $[x_{bl}, y_{bl}]$ and the top right corner is $[x_{tr}, y_{tr}]$, which are defined in Eq. 5. This scheme dramatically reduces the search positions even the video sequence has large MVs. For example, using "stefan" CIF first 255 frames, if $F_w = F_h = 64$, $x_e = y_e = 16$, the average searched positions just account for 28.7% of the search frame.

$$\begin{cases} x_{bl} = \max(-F_w/2, \min(\overrightarrow{MV}_{4\times4\_A}[x], \overrightarrow{MV}_{4\times4\_B}[x], \\ \quad \overrightarrow{MV}_{4\times4\_C}[x], \overrightarrow{MV}_{4\times4\_D}[x]) - x_e) \\ y_{bl} = \max(-F_h/2, \min(\overrightarrow{MV}_{4\times4\_A}[y], \overrightarrow{MV}_{4\times4\_B}[y], \\ \quad \overrightarrow{MV}_{4\times4\_C}[y], \overrightarrow{MV}_{4\times4\_D}[y]) - y_e) \\ x_{tr} = \min(F_w/2, \max(\overrightarrow{MV}_{4\times4\_A}[x], \overrightarrow{MV}_{4\times4\_B}[x], \\ \quad \overrightarrow{MV}_{4\times4\_C}[x], \overrightarrow{MV}_{4\times4\_D}[x]) + x_e) \\ y_{tr} = \min(F_h/2, \max(\overrightarrow{MV}_{4\times4\_A}[y], \overrightarrow{MV}_{4\times4\_B}[y], \\ \quad \overrightarrow{MV}_{4\times4\_C}[y], \overrightarrow{MV}_{4\times4\_D}[y]) + y_e) \end{cases} \quad (5)$$

Table 1. Simulation Conditions

QP	28, 30, 32, 34, 36, 38, 40		
Search Range	Jm81a	SFS[2]	Proposed
	QCIF:±16 CIF:±24	±32	$F_w = F_h = 64$ $x_e = y_e = 16$
Image (Format)	foreman(QCIF),carphone(QCIF),news(QCIF), coastguard(QCIF),stefan(CIF),tempete(CIF)		
etc	no B slice, CAVLC, 5 references R-D optimization, Hadamard Transform		

3. EXPERIMENTAL RESULTS

The experimental conditions are shown in Table 1 and the relative results are shown in Table 2. For the evaluation of the proposed algorithm performance, we test the FS algorithm in reference software (Jm81a) and the simplified full search (SFS) algorithm provided in [2]. BDBR (Bjonteggard Delta BitRate) and BDPSNR (Bjonteggard Delta PSNR) [6], which are respectively average difference of bitrate and PSNR between two methods, are used and they are derived from the simulation results when QP $= 28, 32, 36, 40$. FS of Jm81a is adopted as norm. The BDBR and BDPSNR results are shown in Table 3. The $(+)$ sign in BDBR and $(-)$ sign in BDPSNR indicate the coding loss. From Table 3 we observe that our

Table 2. Performance Comparison

Sequence	Algorithm	Rate (kbps)							PSNR (dB)						
		28	30	32	34	36	38	40	28	30	32	34	36	38	40
foreman	FS	124.156	92.716	70.247	54.67	42.81	34.194	27.312	35.782	34.425	33.135	31.883	30.647	29.379	28.137
	SFS	127.130	95.448	72.693	57.14	44.932	36.124	29.821	35.746	34.391	33.144	31.951	30.734	29.549	28.418
	OUR	124.985	93.587	70.892	55.317	42.969	34.409	27.676	35.765	34.407	33.113	31.904	30.618	29.363	28.076
carphone	FS	129.907	97.236	73.084	55.472	41.370	31.301	24.004	36.926	35.410	34.015	32.653	31.304	29.873	28.720
	SFS	131.301	98.336	74.012	56.173	42.010	32.176	24.611	36.901	35.398	33.926	32.634	31.279	29.858	28.697
	OUR	130.830	98.025	73.596	55.915	41.485	31.543	24.334	36.899	35.402	33.960	32.599	31.278	29.865	28.708
news	FS	73.273	56.821	43.930	34.632	26.676	20.79	16.399	36.628	35.08	33.575	32.208	30.692	29.242	28.032
	SFS	74.404	57.869	44.438	35.337	26.974	21.018	16.525	36.612	35.062	33.523	32.175	30.648	29.167	28.016
	OUR	74.008	57.362	44.460	34.721	26.651	20.825	16.398	36.620	35.065	33.592	32.145	30.635	29.218	28.006
coastguard	FS	214.408	145.411	97.529	67.300	45.618	32.075	23.146	34.07	32.504	31.032	29.722	28.418	27.207	26.127
	SFS	215.035	145.537	97.833	67.271	45.631	32.037	23.547	34.055	32.483	31.027	29.718	28.405	27.198	26.131
	OUR	215.570	145.491	98.629	67.512	45.746	32.134	23.507	34.061	32.47	31.019	29.707	28.401	27.188	26.155
stefan	FS	1115.61	802.705	576.050	426.251	314.191	240.706	188.099	35.622	33.960	32.394	30.950	29.411	27.914	26.486
	SFS	1120.79	808.614	581.270	431.496	320.579	246.943	193.877	35.594	33.929	32.375	30.923	29.389	27.905	26.492
	OUR	1133.33	816.545	585.293	434.374	321.490	245.636	191.583	35.577	33.909	32.347	30.892	29.378	27.881	26.446
tempete	FS	1122.55	785.151	541.218	382.631	266.319	192.795	142.914	34.884	33.253	31.689	30.287	28.844	27.448	26.158
	SFS	1127.45	788.035	544.257	385.817	269.340	196.397	146.563	34.861	33.218	31.666	30.267	28.820	27.436	26.152
	OUR	1126.11	787.091	542.813	383.686	267.456	193.841	143.729	34.857	33.219	31.667	30.267	28.836	27.425	26.137

algorithm and SFS have the similar performance as reference FS. In most sequences, our scheme is better than SFS except for "coastguard" and "stefan". The higher coding loss in "stefan" sequence is caused by the limitation of sub-search area. If x_e and y_e are extend to $x_e = y_e = 24$, significant quality gain can be observed. For "coastguard", it contains a lot of high frequency signals because its background includes a river. So the low-pass filter reduces the VBSME's quality.

In these simulations, our algorithm needs much less computation operations in integer pixel accuracy VBSME. For example, in "tempete" sequence, the searched points of our algorithm is 28.6% of SFS's and at each position the "AD" operation number is just 25% of SFS's, so the total "AD" operation of our algorithm is 7.15% of SFS's.

Table 3. Performance Comparison in BDBR and BDPSNR

	SFS[2]		Proposed	
	BDBR	BDPSNR	BDBR	BDPSNR
foreman	+3.36%	-0.167dB	+1.26%	-0.064dB
carphone	+2.57%	-0.122dB	+1.40%	-0.067dB
news	+1.86%	-0.107dB	+0.90%	-0.053dB
coastguard	+0.60%	-0.021dB	+1.10%	-0.038dB
stefan	+1.87%	-0.094dB	+2.70%	-0.136dB
tempete	+1.48%	-0.062dB	+0.80%	-0.032dB

4. CONCLUSIONS

One VLSI parallel processing oriented integer pixel accuracy VBSME algorithm is provided in this paper. Three major approaches are proposed. First, low-pass filter based subsampling scheme effectively reduces about 75% arithmetic operations in every "SAD" calculation with trivial coding efficiency loss. Similar to FS algorithm, SAD reusing scheme can be adopted by our algorithm to reduce the computation cost more. Second, the SMVP algorithm avoids the data dependency among sub-partitions within one MB and this makes VLSI parallel processing feasible. At last, the proposed sub-search window could efficiently reduces the search positions. Consequently, the power consumption for integer pixel accuracy VBSME could be reduced in ratio to the average search area. Compared with the original FS, this algorithm has considerably lower complexity and it is much more suitable for parallel processing VLSI implementation.

5. REFERENCES

[1] J. Ostermann, J. Bormans, P. List, D. Marpe, M. Narroschke, F. Pereira, T. Stockhammer, and T. Wedi, "Video coding with H.264/AVC: Tools, performance, and complexity," *IEEE Circuits and Systems Magazine*, vol. 4, no. 1, pp. 7–28, First Quarter 2004.

[2] Y.W. Huang et.al., "Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264," in *ISCAS '03. Proceedings of the 2003 International Symposium on Circuits and Systems*, May 2003, vol. 2, pp. 796–799.

[3] Y.W. Huang et.al., "A 1.3TOPS H.264/AVC single-chip encoder for HDTV applications," in *IEEE International Solid-State Circuits Conference 2005*, Febrary 2005, pp. 128–130.

[4] H.W. Park and H.S. Kim, "Motion estimation using low-band-shift method for wavelet-based moving-picture coding," *IEEE Transactions on Image Processing*, vol. 9, no. 4, pp. 577–587, April 2000.

[5] S.Y. Yap and J. V. McCanny, "A VLSI architecture for variable block size video motion estimation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 51, no. 7, pp. 384–389, October 2004.

[6] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," *ITU-T Q.6/16, Doc. #VCEG-M33*, Mar. 2001