AN EFFICIENT AND LOW POWER ARCHITECTURE DESIGN FOR MOTION ESTIMATION USING GLOBAL ELIMINATION ALGORITHM

Yu-Wen Huang, Shao-Yi Chien, Bing-Yu Hsieh, and Liang-Gee Chen

DSP/IC Design Lab
Graduate Institute of Electronics Engineering, National Taiwan University
No. 1, Sec. 4, Roosevelt Road, Taipei 106, Taiwan
{yuwen, shoayi, bingyu, lgchen}@video.ee.ntu.edu.tw

ABSTRACT

This paper presents a new algorithm and architecture for motion estimation. The proposed global elimination algorithm (GEA) is derived from successive elimination algorithm (SEA). The main idea is to remove the branches of SEA to make data flow more regular and suitable for hardware. Besides, the processing time per motion vector for GEA is fixed, no initial guess is required, and the skipping ratio of search positions can be fixed within frames and is even higher than 99%. The average PSNR of compensated frames is almost the same (within 0.1dB) as that of full-search block matching algorithm (FBMA). An architecture composed of a systolic part, an adder tree, and a comparator tree is also developed for GEA. Simulation results show our design outperforms many FBMA architectures in normalized processing capability per gate and normalized power at gate level.

1. INTRODUCTION

Motion-compensated transform coding has been adopted by all of the existing standards related to video coding, such as the MPEG series and the H.26x series. Motion estimation removes temporal redundancy within frames and thus provides the coding system with high compression ratio. Full-search block matching algorithm (FBMA) is the most popular but demands the most computation. Recently, successive elimination algorithm (SEA) [1][2] is well known for its capabilities to reduce the heavy computation of FBMA and maintain the same results as FBMA. It is more attractive than other fast algorithms that cause PSNR loss, such as three-step search, diamond search, ...etc. However, it is critical for SEA to find good initial guesses of motion vectors, which is a difficult task for the regions where the motion field is not smooth. Besides, due to the irregular data flow, systolic mapping of SEA for hardware is never an easy task.

A new search algorithm called global elimination algorithm (GEA) is proposed in this paper. In contrast to SEA, the branches are removed, the data flow is regular, the processing time per motion vector is fixed, no initial guess is needed, and the skipping ratio is also fixed within frames and is higher than 99%. Experimental results show that GEA achieves almost the same coding gain as FBMA, and sometimes the PSNR of the compensated frame obtained by GEA is even higher than that of FBMA. An efficient architecture design using GEA is also proposed. The core consists of a systolic part, an adder tree, and a comparator tree. Compared to many FBMA architectures [3]-[9], the normalized processing capability per gate is the best, and the normalized power at gate level is the lowest.

In Section 2, SEA is reviewed, and then the proposed GEA is described. Next, GEA architecture is presented in Section 3. Section 4 compares GEA architecture to FBMA architectures. Finally, Section 5 gives a conclusion.

2. ALGORITHMS

FBMA can be described by the following equations:

$$SAD(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |c(i,j) - s(i+m, j+n)| \qquad (1)$$
$$MV = \{(u,v) \mid SAD(u,v) \le SAD(m,n), -p \le m \le p-1, -p \le n \le p-1\}$$

where current block data are $\{c(x,y) | 0 \le x \le N\text{-}1,\ 0 \le y \le N\text{-}1\}$, search area data are $\{s(x,y) | -p \le x \le p+N\text{-}2,\ -p \le y \le p+N\text{-}2\}$, block size is NxN, search range is $-p$~p-1, and MV is the motion vector of current block with minimal SAD among $(2p)^2$ search positions.

2.1. Successive elimination algorithm

The main idea of SEA can be shown in the following equation:

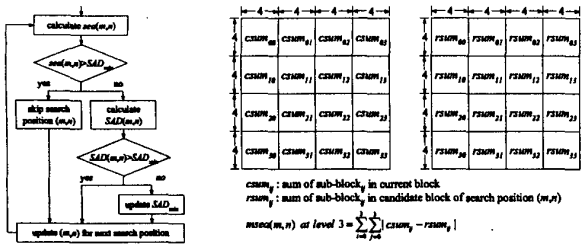
$$SAD(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |c(i,j) - s(i+m, j+n)| \qquad (2)$$
$$\ge \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |c(i,j)| - \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |s(i+m, j+n)| \equiv K - SB(m,n) \equiv sea(m,n)$$

For every search position, the computation of sea is much easier than that of SAD due to the fact that K is the sum of current block and only has to be calculated one time, and the sum of candidate block $SB(m, n)$ can be derived from $SB(m\text{-}1, n)$:

$$SB(m,n) = SB(m-1,n) + \sum_{a=0}^{N-1} s(m+N-1, n+a) - \sum_{b=0}^{N-1} s(m-1, n+b) \qquad (3)$$

If $sea(m,n)$ is larger than current minimal SAD (SAD_{min}), it is guaranteed by (2) that $SAD(m,n)$ will be larger than SAD_{min}, and thus the search position (m,n) can be skipped. Otherwise, $SAD(m,n)$ needs to be calculated and compared with SAD_{min}. It is clear that a good initial guess of MV with small SAD is critical for SEA to increase the skipping ratio. Fig. 1(a) shows the flow-chart of SEA. A conditional branch exists after sea calculation for each search position, which makes data flow irregular.

0-7803-7402-9/02/$17.00 ©2002 IEEE



(a) (b)

Fig. 1. (a) Flowchart of SEA, (b) *msea* at level 3 with N=16.

2.2. Multilevel successive elimination algorithm

Let us modify (2) to form the following equation:

$$SAD(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |c(i,j) - s(i+m, j+n)| \quad (4)$$

$$\geq \sum_{q=0}^{L-1} |K_q - SB_q(m,n)| \equiv msea(m,n)$$

$$\geq \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |c(i,j)| - \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |s(i+m, j+n)| \equiv K - SB(m,n) \equiv sea(m,n)$$

In (4), an NxN block is divided into L sub-blocks, K_q is the sum of q-th sub-block in current block, and $SB_q(m,n)$ is the sum of q-th sub-block in candidate block at search position (m,n). For each search position, $msea(m,n)$ is calculated to decide if search position (m,n) can be skipped. When L=1, (4) is reduced to (2) and is called *msea* at level 1. If the sub-blocks are of the same size and L=4,16..., it is called *msea* at level 2,3,..., respectively. In Fig. 1(b), *msea* at level 3 with N=16 is taken as an example. The skipping ratio of multilevel SEA (MSEA) is higher than that of SEA, but to calculate *msea* requires more computation than to calculate *sea*. When spiral scan or initial guess at MV predictor is adopted, skipping ratio is 50%~90% depending on sequences.

2.3. Global elimination algorithm

The data flow of SEA or MSEA is not regular due to the conditional branches after the *sea* or *msea* calculation to determine the skipping process. This makes hardware design a tough task. Furthermore, when true motion vectors are beyond search range, good initial guesses can never be found, and the skipping ratio may be so low that even the processing time of SEA or MSEA is longer than that of FBMA. In our GEA, these problems can be solved as the following steps:

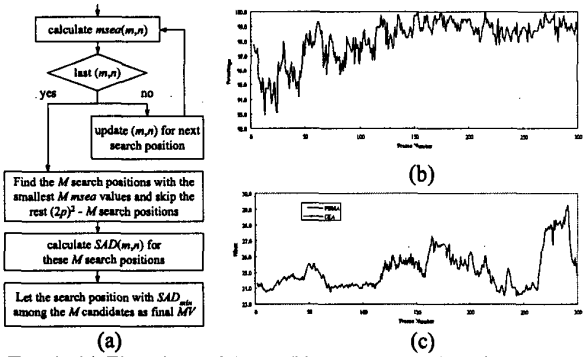
1) Calculate *msea* for every search position in raster scan order.
2) Find the M search positions with the smallest M *msea* values and skip the rest $(2p)^2 - M$ search positions.
3) Calculate the SAD values for these M search positions.
4) Let (u,v) with SAD_{min} among the M candidates as final MV.

As readers can see the flow chart of GEA in Fig. 2(a), global elimination of search positions is in step 2 after all *msea* values are calculated. No conditional branch is necessary within the calculation of *msea* for all search positions, i.e. the control of GEA is easier than that of SEA or MSEA. The selection of M is a trade-off between speed and coding gain. Generally speaking, larger M makes the results more reliable while smaller M saves more computation. Regardless of M value, the processing time per MV is now fixed, which is another good feature for hardware.

Although GEA does not guarantee the same results as FBMA, GEA is very reliable. Experimental results of typical applications with 16x16 block size, 176x144 QCIF (352x288 CIF) video sequences, –16~15 (-32~31) search range, *msea* at level 3, and M=7, which corresponds to a 99.31% (99.83%) skipping ratio, are shown in Table 1. Many standard sequences are tested. Sometimes, the average PSNR of the compensated frames obtained by GEA is even higher than that generated by FBMA. (Note that a minimal SAD does not guarantee a minimal mean square error, e.g. $1+9<5+6$, $1^2+9^2>5^2+6^2$.) The maximal drop of average PSNR of GEA is only about 0.08dB for Hall Monitor CIF. At most of the time, the results of GEA are close to those of FBMA. Fig. 2(b)(c) illustrates Mobile Calendar CIF as an example. In average, 98.1% motion vectors are the same between FBMA and GEA for Mobile Calendar CIF. The PSNR curves are very close, so it is very hard to distinguish them.

Table 1. Comparison of FBMA and GEA with parameters as follows: N=16, *msea* at level 3, M=7, (a) QCIF, p=16, skipping ratio=99.31%, (b) CIF, p=32, skipping ratio=99.83%. The average PSNR of compensated frames is shown in dB.

Video Sequence	(a)		(b)	
	FBMA	GEA	FBMA	GEA
Coastguard	32.93	32.93	31.59	31.55
Container	43.11	43.11	38.53	38.53
Foreman	32.21	32.22	32.85	32.82
Hall Monitor	32.98	32.97	34.90	34.82
Mobile Calendar	26.15	26.15	25.20	25.16
Silent	35.14	35.16	36.12	36.11
Stefan	24.71	24.67	25.73	25.71
Table Tennis	32.10	32.11	33.03	32.96
Weather	38.42	38.42	37.45	37.45



(a) (b) (c)

Fig. 2. (a) Flowchart of GEA, (b) percentage of motion vectors same as FBMA, Mobile Calendar CIF, and (c) PSNR curves of compensated frames for FBMA and GEA.

3. ARCHITECTURE DESIGN BASED ON GEA

In this section, N=16, *msea* at level 3, and M=7 are taken as parameters to develop an architecture design based GEA.

3.1 Systolic part

The data flow of systolic part that calculates the sum of each 4x4 sub-block is shown in Fig. 3. Note that $c_{l,k}$ and $s_{l,k}$ denotes $c(k,l)$ and $s(k,l)$ in (1), respectively, the rectangles are shift registers, and the search range is –16~15 (p=16). A column of block data is loaded in parallel at each cycle. When t=0~15, c data are loaded. The sums of sub-blocks in current block is computed at

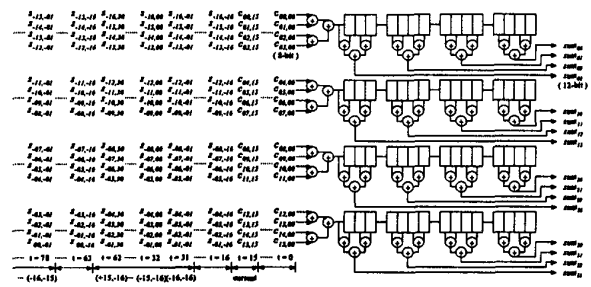


Fig. 3. The data flow of systolic part. A column of block data is loaded, and $sum_{00} \sim sum_{33}$ are calculated in parallel.

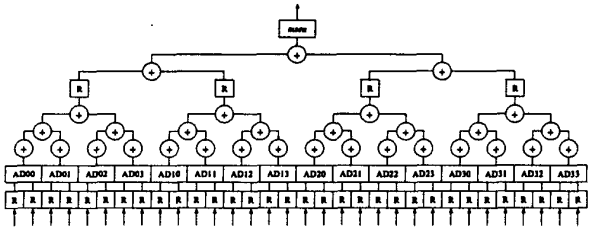


Fig. 4. Parallel adder tree to calculate *msea* in (4).

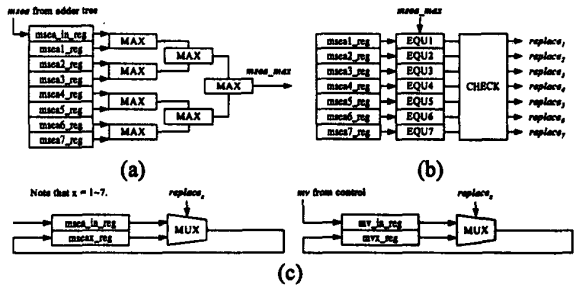


Fig. 5. Parallel comparator tree, (a) forward part to compute *msea_max*, (b) one feedback part to find the mseax_reg to be replaced, (c) the other feedback part to replace the mseax_reg with msea_in_reg.

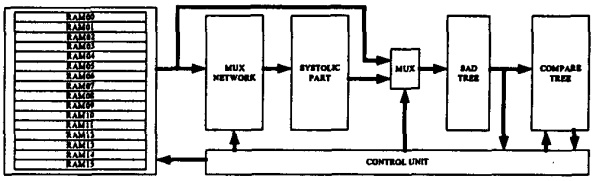


Fig. 6. Overall architecture design for GEA.

t=15 and should be stored into registers at the rising edge of t=16. Next, *s* data are loaded column by column. When t=16~62, *s* data of search positions (-16, -16) ~ (+15, -16) are loaded, and the sums of sub-blocks for search positions (-16, -16)~ (+15, -16) are available at t=31~62, respectively. Then, *s* data of search positions (-16, -15) ~ (+15, -15) are loaded at t=63~109 and vice versa. Thus, $N + 2p$ ($2p+N$-1) clock cycles are needed to load all data into the systolic module.

3.2 Parallel adder tree

The purpose of parallel adder tree is to compute *msea* in (4). In Fig. 4, R is pipeline register. ADxx calculates absolute difference between $csum_{xx}$ and $rsum_{xx}$ denoted in Fig. 1(b). The parallel adder tree adds the results from AD00~AD33 and outputs *msea*.

3.3 Parallel comparator tree

The comparator tree is composed of a forward part shown in Fig. 5(a) and feedback parts shown in Fig. 5(b)(c). Note that the symbols with "_reg" are registers. In Fig. 5(a), mseal_reg ~ msea7_reg should be properly initialized as 0xFFFF before the first valid *msea* value from parallel adder tree comes to msea_in_reg. The maximal value *msea_max* among eight *msea* registers is computed by the forward part. In Fig. 5(b), EQUx compares whether mseax_reg equals to *msea_max* or not. The CHECK functions as states: If no EQU is active, no $replace_x$ is active. If only EQUx is active, only $replace_x$ is active. If two or more EQU units are active, only one of the corresponding $replace_x$ can be active. For example, when EQU1 and EQU2 are active, only $replace_1$ will be active. In Fig. 5(c), if $replace_x$ is active, mseax_reg and mvx_reg will be replaced by the current values in msea_in_reg and mv_in_reg, respectively, at the rising edge of next clock cycle.

To sum up, the comparator tree is to keep the *M* smallest *msea* values and their corresponding motion vectors in mseax_reg and mvx_reg registers, respectively. However, as shown in Fig. 3, invalid *msea* values are generated periodically during the loading of *s* data. At these cycles, the *msea* inputted to comparator tree should be replaced by 0xFFFF so that the results are not affected.

3.4 Overall architecture design

In Fig. 6, GEA architecture is presented in a whole view. Sixteen RAM modules are used in order to output a column of block data in parallel. MUX NETWORK selects the right data into SYSTOLIC PART. It contains sixteen 4-to-1 8-bit multiplexers. The control signals for MUX NETWORK must be properly adapted for the different rows of search positions. After the *M* motion vectors with smallest *msea* values among $(2p)^2$ search positions are ready, *SAD* of these *M* search positions must be computed next. Now SAD TREE that calculates *msea* before can be reused to do this job. The calculation of *SAD* for a candidate block requires *N* cycles. Thus, GEA architecture totally needs $N + 2p$ ($2p+N$-1) + 3 + *MN* cycles to find a motion vector. *N* is responsible for the loading of *c* data, $2p$ ($2p+N$-1) is for the loading of *s* data, 3 is due to pipelines, and *MN* is the clock cycles needed to calculate the *SAD* values of *M* search positions.

4. COMPARISON WITH OTHER ARCHITECTURES

The comparison of GEA architecture to other architectures based on FBMA is shown in Table 2 and 3. All designs require memories of the same bits to store current block data and search area data. Only the PE array is synthesized because the control unit is a very small part of the whole design. The PE array of each design is synthesized and reported by SYNOPSYS Design Analyzer with 50 MHz constrain using AVANT! 0.35μm cell library. Architectures with symbol * need many shift registers to avoid data hazards, but these are not implemented. The real gate count and power at gate level for these designs should be higher than reported. The processing capability per gate and gate-level power should be compared under the same throughput of motion

Table 2. Comparison of architectures with $N = 16$, $p = 16$, *msea* at level 3, and $M = 7$. Only the PE array is synthesized.

Architecture	Description	No. of PE	Cycles per MV	Required Memory I/O	Required Freq. for CIF 30 fps	Gate Count @50MHz	Normalized Processing Capability per Gate	Gate-Level Power @50MHz	Normalized Power
[3] Yang	1-D semi-systolic	32	8192	24 bits	97.32 MHz	28.0K	0.13	26.0 mW	2.99
[4] AB1	1-D systolic	16	24064	256 bits	285.88 MHz	3.8K	0.32	11.7 mW	3.95
[4] AB2	2-D systolic	256	1504	128 bits	17.87 MHz	95.1K	0.20	227.8 mW	4.82
[5] Hsieh*	2-D systolic	256	2209	8 bits	26.24 MHz	100.6K	0.13	147.2 mW	4.57
[6] Tree	Tree structure	256	1024	2048 bits	12.17 MHz	56.1K	0.51	179.5 mW	2.59
[7] Yeo	2-D semi-systolic	1024	256	24 bits	3.04 MHz	447.4K	0.26	1052.6 mW	3.79
[8] Lai	1-D semi-systolic	1024	256	24 bits	3.04 MHz	387.6K	0.30	845.6 mW	3.04
[9] SA*	2-D systolic	256	1024	16 bits	12.17 MHz	126.5K	0.23	258.0 mW	3.72
[9] SSA*	2-D semi-systolic	256	1024	16 bits	12.17 MHz	106.0K	0.27	280.1 mW	4.04
Ours	Based on GEA	16	1635	128 bits	19.42 MHz	17.9K	1.00	43.4 mW	1.00

Table 3. Comparison of architectures with $N = 16$, $p = 32$, *msea* at level 3, and $M = 7$. Only the PE array is synthesized.

Architecture	Description	No. of PE	Cycles per MV	Required Memory I/O	Required Freq. for CIF 30 fps	Gate Count @50MHz	Normalized Processing Capability per Gate	Gate-Level Power @50MHz	Normalized Power
[3] Yang	1-D semi-systolic	64	16384	24 bits	194.64 MHz	56.0K	0.10	52.0 mW	3.78
[4] AB1	1-D systolic	16	80896	256 bits	961.04 MHz	3.8K	0.30	11.7 mW	4.20
[4] AB2	2-D systolic	256	5056	128 bits	60.07 MHz	95.1K	0.19	227.8 mW	5.12
[5] Hsieh*	2-D systolic	256	6241	8 bits	74.14 MHz	100.6K	0.15	147.2 mW	4.08
[6] Tree	Tree structure	256	4096	2048 bits	48.66 MHz	56.1K	0.40	179.5 mW	3.27
[7] Yeo	2-D semi-systolic	4096	256	24 bits	3.04 MHz	1790.0K	0.20	4210.3 mW	4.79
[8] Lai	1-D semi-systolic	4096	256	24 bits	3.04 MHz	1550.4K	0.23	3382.4 mW	3.84
[9] SA*	2-D systolic	256	4096	16 bits	48.66 MHz	126.5K	0.18	258.0 mW	4.69
[9] SSA*	2-D semi-systolic	256	4096	16 bits	48.66 MHz	106.0K	0.21	280.1 mW	5.09
Ours	Based on GEA	16	5187	128 bits	61.62 MHz	17.9K	1.00	43.4 mW	1.00

vectors, so the normalized processing capability per gate (NPCPG) and the normalized power (NP) are defined as:

$$NPCPG_{XXX} = \frac{[(\text{Required Freq. for CIF 30 fps})^{-1} / (\text{Gate Count @50MHz})] \text{ for XXX}}{[(\text{Required Freq. for CIF 30 fps})^{-1} / (\text{Gate Count @50MHz})] \text{ for GEA}} \quad (5)$$

$$NP_{XXX} = \frac{[(\text{Power @50MHz}) \times (\text{Required Freq. for CIF 30 fps / 50MHz})] \text{ for XXX}}{[(\text{Power @50MHz}) \times (\text{Required Freq. for CIF 30 fps / 50MHz})] \text{ for GEA}} \quad (6)$$

Simulation results show that GEA architecture outperforms many FBMA architectures in normalized processing capability per gate and normalized power at gate level. In addition, it is easy for GEA architecture to support advanced prediction mode, i.e. to allow four motion vectors of 8×8 blocks in a 16×16 macro-block. Only four extra parallel comparator trees and some changes for control are required in this case.

5. CONCLUSION

In this paper, a novel motion estimation algorithm called GEA and an architecture for GEA are proposed. Several problems of SEA are solved by GEA. No initial guess is needed, the data flow is regular, the processing time per motion vector is fixed, the skipping ratio is high, and the estimation results of GEA are almost the same as those of FBMA. The architecture design for GEA is also efficient in area, speed, and power, compared with many existing FBMA architectures.

6. REFERENCES

[1] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Trans. on Image Processing*, vol. 4, no. 1, pp. 105-107, Jan. 1995.

[2] X.Q. Gao, C.J. Duanmu, and C.R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Trans. on Image Processing*, vol. 9, no. 3, pp. 501-504, Mar. 2000.

[3] K.M. Yang, M.T. Sun, and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," *IEEE Trans. on Circuits and Systems*, vol. 36, no. 2, pp. 1317-1358, Oct. 1989.

[4] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Trans. on Circuits and Systems*, vol. 36, no. 2, pp. 1301-1308, Oct. 1989.

[5] C.H. Hsieh and T.P. Lin, "VLSI architecture for block-matching motion estimation algorithm," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 2, no. 2, pp. 169-175, Jun. 1992.

[6] Y.S. Jehng, L.G. Chen and T.D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. on Signal Processing*, vol. 41, no. 2, pp. 889-900, Feb. 1993.

[7] H. Yeo and Y.H. Hu, "A novel modular systolic array architecture for full-search block matching motion estimation," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 5, no. 5, pp. 407-416, Oct. 1995.

[8] Y.K. Lai and L.G. Chen, "A data-interlacing architecture with two-dimensional data-reuse for full-search block-matching algorithm," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 8, no. 2, pp. 124-127, Apr. 1998.

[9] Y.H. Yeh and C.Y. Lee, "Cost-effective VLSI architectures and buffer size optimization for full-search block matching algorithms," *IEEE Trans. on VLSI Systems*, vol. 7, no. 3, pp. 345-358, Sep. 1999.