Efficient architecture for global elimination algorithm for H.264 motion estimation

P MURALIDHAR* and C B RAMARAO

Department of Electronics and communication Engineering, N.I.T. Warangal, India
e-mail: pmurali_nitw@yahoo.co.in; cbrr@nitw.ac.in

MS received 29 September 2013; revised 11 June 2015; accepted 23 October 2015

Abstract. This paper presents a fast block matching motion estimation algorithm and its architecture. The proposed architecture is based on Global Elimination (GE) Algorithm, which uses pixel averaging to reduce complexity of motion search while keeping performance close to that of full search. GE uses a preprocessing stage which can skip unnecessary Sum Absolute Difference (SAD) calculations by comparing minimum SAD with sub-sampled SAD (SSAD). In the second stage SAD is computed at roughly matched candidate positions. GE algorithm uses fixed sub-block sizes and shapes to compute SSAD values in preprocessing stage. Complexity of this GE algorithm is further reduced by adaptively changing the sub-block sizes depending on the macro-block features. In this paper adaptive Global Elimination algorithm has been implemented which reduces the computational complexity of motion estimation algorithm and thus resulted in low power dissipation. Proposed architecture achieved 60% less number of computations compared to existing full search architecture and 50% high throughput compared to existing fixed Global Elimination Architecture.

Keywords. Fast block matching motion estimation; global elimination; matching complexity reduction; power reduction.

1. Introduction

Digital video is to be compressed for efficient storage or transmission due to its massive amount of data. To achieve a high compression ratio, a Block Matching Motion Estimation (BMME) algorithm is used to remove temporal redundancy in video coding standards such as H.263, H.264, and MPEG-4. For each macro block (MB) of size 16×16 in the current frame, the algorithm searches for the best matching block within a search region around the current block position in the previous frame. Sum of absolute differences (SAD) is the commonly used matching criterion due to its simple operations. Speed of motion estimation depends on the computational complexity of the motion estimation algorithm used.

Three Step Search (TSS) [1], Four Step Search (FSS) [2], and Hexagon Search (HS) [3] are some of the algorithms that reduce computational complexity by reducing the number of search points in search window.

These algorithms assume ‘Uni-modal Error Surface’ in which the matching error would increase with the distance from the global minimum, thus results in quality loss compared to full search in high resolution applications because of local minimum. Apart from the search point reduction,

there are many algorithms that optimize the motion estimation task by reducing matching complexity at each search position. Some algorithms simplify the SAD calculation by pixel sub-sampling, exploiting spatial correlation of pixels in alternate pixel pattern [4], quincunx patterns, etc. since only a subset of pixels is used for SAD calculation in sub-sampling based algorithms, the quality loss can be large for large sub-sampling ratio. Averaging of pixels is also quite commonly used to reduce matching complexity. This prevents elimination of large number of pixels from distortion measurement. Successive Elimination Algorithm (SEA) [5], Multilevel Successive Elimination Algorithm (MSEA) [6], and Global Elimination Algorithm (GEA) [7] use averaging of pixels and achieve quality comparable with the full search algorithm.

Successive Elimination Algorithm works on the principle that the absolute difference of sum of pixels of current block and sum of pixels of candidate block cannot be greater than the minimum SAD. A large number of search points can be eliminated early if the Absolute Difference of sum-of-pixels at these points is larger than the best SAD till that point. Thus complete SAD calculation can be avoided at many of the search points by using this method. In this at each search point, SEA (sum of current pixels – sum of reference pixels) is computed to know whether the search point is to be eliminated or not.

*For correspondence

Multilevel SEA algorithms help in eliminating more candidates quickly by creating successively tighter bounds and hence avoiding full SAD calculation at most of the points. In MSEA, a series of such bounds is constructed by using the fact that sum of absolute difference of larger block sums would be smaller than sum of absolute difference of smaller block-sums within a macro-block. In the above two algorithms, computational complexity is reduced compared to traditional full search algorithm but they are not suitable for hardware implementation due to irregular data flow of the algorithm and these algorithms require initial threshold. Choosing the threshold value will affect the speed of motion estimation. Hence to overcome these problems Global Elimination algorithm is proposed.

2. Global elimination algorithm

2.1 Global elimination Algorithm

Global Elimination is a two-stage motion estimation algorithm. Due to the regular data flow structure, and simple two stage control flow, the global elimination algorithm is of interest in VLSI implementations. The global elimination algorithm (GE) carries out motion estimation in two stages. In the first stage each current MB is divided into equal sized sub-blocks and the sub-block averages are calculated. A reference MB at each search point is also identically divided into equal number of sub-blocks. The SAD in sub-block averages (SSAD) between current MB and reference MB is used as the matching measure rather than a complete SAD. The complexity reduction is achieved both because SSAD is cheaper to calculate than SAD and also because the averages at successive search points can be incrementally calculated from some of the information saved from the previous reference point. The top ' N ' MB candidates with least cost function based on SSAD values are selected for the 2nd stage search. During the 2nd stage, a full search (FS) is performed on the entire ' N ' candidate MBs to find the best integer pixel motion vector.

GE algorithm uses fixed number of partitions to compute SSAD values in preprocessing stage. In many video sequences, there are many plain MB's i.e., pixels values within the macro-block are almost same. For such MBs, a reduced partition count can also give similar PSNR quality at a given bit rate, but with a reduced computational cost.

In this paper we propose solution to address these problems. We calculate the pixel variance of each MB to determine the right amount of partitioning to use. Lower variance MBs will have smaller number of partitions. By making use of variance, we obtain more optimal macro-block partitioning to reduce complexity of SAD calculations.

2.2 Need for partitioning

Partitioning the MB into small sub-blocks in first stage will result in more calculations in the matching operation

compared to partitioning into larger sub-blocks. Small number of partitions would greatly reduce the complexity of first stage. However, a cruder partition will also result in poorer match compared to a finer partition because a larger number of pixels are merged to find the mean values and a lot of details are lost. A larger number of top best candidates need to be then selected for the 2nd stage search. Thus it is desirable to limit the number of top candidates in the 2nd stage because of the number of comparisons required to find the worst candidate at each reference point in the 1st stage increases with the number of top candidates. Also, calculating full SAD (in 2nd stage) on a large number of candidates located at arbitrary positions in the search window is costly. Thus, there is a scope to reduce the computation cost by adaptively partitioning macro-blocks based on the contents of individual current macro-blocks. Plain MBs can be partitioned into smaller number of sub-blocks with not much loss in quality, but significant reduction in complexity.

2.3 Adaptive GE with variance based partitioning

In this partitioning of MB into sub-blocks is accomplished based on variance of current block. Many video sequences contain spatial regions which are nearly plain. Since plain macro-blocks are typically embedded in plain regions, if we look at the SAD values for plain MBs with different motion vectors, we would get a gradual error surface with many motion vector candidates resulting in very small and similar SAD values. The resulting bit rate due to these MBs will also be very small due to the absence of high frequencies and small error in motion vector estimation will not result in large increase in bit rate. We use this observation to adapt GE partitioning for plain regions. We measure the pixel variance within current MB. If the variance exceeds a certain threshold, then the MB is partitioned into $4 \times 4 = 16$ sub-blocks, each containing 16 pixels. If the variance is smaller than threshold value, then the MB is partitioned into $2 \times 2 = 4$ sub-blocks of 64 pixels each. The variance measure that is used is sum of absolute difference between pixel values and mean value of the MB.

Variance-based partitioning method results in cost reduction at relatively low quality loss. However, the method is more effective for low variance threshold values and hence the cost reduction that can be obtained is small in case of video sequences with small percentage of low variance regions.

3. Architecture design

In this section architecture for proposed GE algorithm with variance and Hadamard transform based partitioning is described. The architecture is as shown in below figure 1. It consists of Datapath, Controller, Current block memories, and Reference block memories. Current block memories are

Figure 1. Architecture diagram.

used to store the current block data. Sixteen number of current block memories are used to store current block data. Depth of each current memory is 16. Sixteen Reference block memories are used to store search area data. Depth of each reference memory is 1504 for a search range of 32×32 and 496 for a search range of 16×16 .

Datapath shown in figure 2 consists of block sum calculator, shift registers and multiplexer logic, current register file, Multiplexers, SSAD/SAD adder, SSAD comparator and SAD comparator. Since the GE algorithm does matching operation at each position in the search range, it is important to minimize the number of computations at each search position.

In the proposed architecture, computations are reduced using block sum calculator to find various sub-block sums. One way to reduce the number of computations is by reusing as much computations as possible. In Huang *et al* [7], the architecture described reused the pixel additions that result in column sums of four pixels. However, at each reference position, the column sums were added together to get the sum of 4×4 blocks. In this section, we demonstrate an architecture, that is suitable for AGE algorithm. We also reduce the number of computations further compared to Fan & Lin [8] by calculating new block sum by subtracting first column sum and adding new column sum, rather than adding all column sums afresh. In case of 4×4 GE, this would reduce the number of computations from 91 to 67. In AGE, where a block-sum may be formed using 4×8 or 8×8 pixels,

this method gives larger benefit by reducing number of column sums from 7 to 2 for such block-sum calculations.

Figure 3 shows the incremental blocksum calculation engine that processes eight lines of pixels. Each block sum engine receives eight incoming pixels as shown on the left-hand side. Two such engines are needed to calculate block-sums with 16 new incoming pixels at each reference position. In each blocksum engine, sum1 and sum3 give 4×4 block sums of top 4 rows and bottom 4 rows respectively. Sum2 and sum4 give the two 4×8 pixel sums. Sum5 and sum6 give 8×4 and 8×8 sums respectively.

Figure 4 shows the shift register implementation that stores the block-sums. A total of 12 shift registers are required to store six sums from top and bottom eight pixel rows. Only a subset of shift registers are required to be active based on macro-block partitioning. Unwanted shift registers are turned off to save power.

Also, some multiplexing logic is necessary to select appropriate block sum for each sub-block position as shown. A maximum of 16 sub-block sums are required for each macro-block (when all sub-blocks are 4×4 pixel partitions). The multiplexing is designed to minimize the multiplexing stages. For each 8×8 block, there are up to four sub-block sums and minimum one sub-block sum. Figure 5 shows all possible partitioning options for an 8×8 block and corresponding block sum variable assignment to minimize multiplexing logic.

The select signals for the multiplexers are determined by the partition calculator block and are static for entire

Figure 2. Datapath block diagram.

Figure 3. Blocksum calculator.

Figure 4. Shift register and multiplexer logic.

Figure 5. Possible partitioning options for 8×8 sub-block.

search window for a macro-block. All the sub-block sums are applied to SSAD/SAD adder to compute SSAD at each search position. SSAD adder is used to compute SSAD in the first stage and SAD in the second stage of GE algorithm.

Result of the SSAD calculator is fed to the SSAD comparator block that maintains the top '7' candidates' SSAD value, motion vector and search point address information in a register file. Figure 6 shows the SSAD comparator. SSAD

Figure 6. SSAD comparator.

values are compared using the max logic tree to find the current worst SSAD candidate. If the new incoming SSAD value is smaller than the worst, then the worst SSAD entry, corresponding motion vectors and search point addresses are replaced by the incoming candidate. In Huang *et al* [7], after calculating the worst SSAD value, a parallel comparator was used to find matching SSAD value in the register file. This gives the candidate index to be replaced. Also, a checking hardware was necessary to break the tie when there is more than one worst candidate. In figure 6, we eliminate the requirement of tag logic. Instead, we maintain an additional bit of information for each max logic finder. This way, we simplify the max logic tree by avoiding the maintenance of tag bits and also avoid the tag comparisons, which increase linearly with number of SSAD candidates.

After extracting seven minimum SSAD positions using SSAD comparator, in the second stage at each extracted position SAD is computed using SSAD/SAD adder. Among these seven search points, minimum SAD position is found to compute the motion vector.

4. Performance analysis and comparison to existing architectures

4.1 Computational cost analysis

Computations of the GE algorithms are dominated by the first stage of GE. Consider a search range of $[-16$ to

$+15]$ (i.e., $p=16$). Then for processing single macro-block, matching is to be performed at 1024 reference positions present in the search window.

For GE4x4 having 16 number of 4×4 sub-blocks will take 67 (ssad takes 15 add/16 sub/16 abs and block sum calculation takes 20 add) operations for single matching operation which is much less than 767 (255 add/ 256 sub/256 abs) needed for single SAD computation at each reference position. Hence total of 67×1024 operations need to be performed for processing single macro-block in the first stage of GE4x4.

In AGE, all 8×8 sub-blocks may not be partitioned into 4×4 sub-blocks. If all the 8×8 sub-blocks are not further partitioned, it takes 33 (ssad takes 3 add/4 sub/4 abs and block sum calculation takes 22 add) operations for each matching operation. For all other partitioning options it takes operations between 33 and 67. Thus the number of computations will depend on the partitioning option of the macro-block. The operations other than first stage of GE include:

- Every current macro-block is first analyzed and its partitioning is decided. This takes less than 400 add/sub/abs calculations.
- At the beginning of the search row, some additional pixel sums need to take place before the first block sum is available. So, for example, to get a block sum of 4×4 block for the 1st block in a search row, 15 sums

are required instead of 3 that are needed for subsequent positions in the same search row.

- In our experiments, we selected only few best candidates from the 1st stage of GE or adaptive GE. The SSAD comparator thus requires only 2–7 comparisons per reference point.
- Also, 2nd stage of GE involves full SAD calculation on top seven candidates, amounting to $767 \times 7 = 5369$

calculations. This number is still small compared to total cost of GE or adaptive GE algorithm, and is ignored.

Table 1 shows the comparison of proposed architecture for two different search ranges. From the results it is clear that by reducing the search range throughput can be improved. Proposed architecture is compared with the existing Global elimination Architectures as shown in

Table 1. Comparison of two implemented architectures.

Property	Proposed architecture1 (32×32) Search range	Proposed architecture2 (16×16) Search range
Search range	32×32 (or) [−16, +15]	16×16 (or) [−8, +7]
No. of clocks per MV	1664	656
Size of memory (bytes)	24,320	8192
No. of slices	7033	6341
No. of slice flip flops	3430	3381
No. of 4 i/p LUT's	12,745	11,469
Max. operating freq (MHz)	32.041	37.77
Throughput (MB/s)	19,256	57,576
Total power (W)	0.408	0.506
CIF frame rate (fps)	48.6	145.39
HD frame rate (fps)	5.34	16
SD frame rate (fps)	16	48
No. of operations per motion vector (max)	81,645	24,813

Table 2. Comparison with existing GE architectures.

Property	Global elimination [7]	Parallel GE [9]	Proposed architecture1
Search range	[−16, +15]	[−16, +15]	[−16, +15]
No. of clock cycles per MV	1635	1730	1664
Subblock sizes possible	Only 4×4	Only 4×4	All sub-block sizes
Memory size	24,320 bytes	13,312 bytes	24,320 bytes
Throughput (MB/s)	12,276	–	19,256
No. of computations	105,721	85,241	81,645 (all 4×4 sub-blocks)

Table 3. Comparison with existing FPGA based implementations.

Design property	1-D Systolic (Li Man Ho)	1-D Tree (Li Man Ho)	Compact systolic [10]	SAD reuse architecture [12]	Proposed
Board used	Virtex 2 pro	Virtex 2 pro	Virtex E	Virtex 2	Virtex 4
Search range	[−16, +15]	[−16, +15]	[−16, +15]	[−4, +4]	[−16, +15]
No. of cycles/MV	16,401	16,393	1983	2086	1664
No. of slices	836	350	14,728	7253	7033
Max. operating freq	239 MHz	240 MHz	55 MHz	68.02 MHz	32.04 MHz
Total power	1.344 W	1.16 W	2 W	–	0.408 W
Throughput (MB/s)	14,424	14,643	27735	32,611	19,256
CIF frame rate (fps)	36	37	70	82	48.6
SD frame rate (fps)	12	12.2	23.11	27.17	16
No. of operations per MV (max)	785,408	785,408	785,408	–	81,645

Table 4. Performance analysis of implemented architecture.

Video name	Addition operations 16×16 block sum (X1)	Absolute operations subblock (4×4) (X2)	Difference operations subblock (4×4) (X3)	Addition operations subblock (4×4) (X4)	No. of search points (X5)	7 min SAD operations per MB =[(767*7)+(X1 +X2+X3 +X4)*218]	% Reduction in computations
salesman	256	12.75	12.75	11.75	396	69,297.5	59.84
caltrain	256	11.20	11.20	10.20	800	68,283.8	60.43
Missa	256	7	7	6	396	65,537	62.02
Football	256	14	14	13	330	70,115	59.37
Susie	256	11	11	10	330	68,153	60.50

table 2. The number of computations for macro-block is reduced compared to Huang *et al* [7, 9] implementations and throughput also improved. Table 3 indicates the comparison of existing FPGA implementation of Motion Estimation Architectures. From table 3 results proposed architecture is outperformed compared to Saldaha & Miguel [10], Li Man Ho [11] and Yalcin *et al* [12].

4.2 Comparing proposed architecture of two implemented architectures

4.3 Comparison with existing existing GE architectures

4.4 Comparison with existing Fullsearch architectures

Table 3 shows that proposed architecture uses less resources, less number of computations and low power compared to the existing full search architecture. Table 4 shows the proposed architecture results in 60% less number of computations compared to Full search block matching algorithm implementations.

4.5 Conclusion

Adaptive Global elimination architecture has been designed which can optimize the matching complexity of motion estimation block in video encoders. Table 4 shows the proposed architecture results in 60% less number of computations compared to Full search block matching algorithm and thus power reduction is achieved. Proposed architecture resulted in 56% throughput improvement and 23% reduction in number of operations compared to the existing Global elimination architecture. Proposed architecture has been designed for two different search ranges. Architecture with search range $[-16, +15]$ supports CIF@48.6fps, SD@16fps, HD@5.34fps whereas architecture with search range $[-8, +7]$ supports CIF@145.39 fps, SD@48fps, HD@16fps frame rates. Thus by using two similar engines having search range $[-8, +7]$ in parallel HD frames can be processed at 30fps frame rate.

References

- [1] Jong-Nam Kim and Tae-Sun Choi 1998 A fast three-step search algorithm with minimum checking points using unimodal error surface assumption. *IEEE Trans. Consumer Electron.* 44(3): 638–648
- [2] Lai-Man Po and Wing-Chung Ma 1996 A novel four-step search algorithm for fast block motion estimation. *IEEE Trans. Circuits Syst. Video Technol.* 6(3): 313–317
- [3] Zhu C, Xiao Lin and Lap-Pui Chau 2002 Hexagon-based search pattern for fast block motion estimation. *IEEE Trans. Circuits Syst. Video Technol.* 12(5): 349–355
- [4] Chan Y L and Siu W C 1996 New adaptive pixel decimation for block motion vector estimation. *IEEE Trans. Circuits Syst. Video Technol.* 6(1): 113–118
- [5] Li W and Salari E 1995 Successive elimination algorithm for motion estimation. *IEEE Trans. Image Process.* 4(1): 105–107
- [6] Gao X Q, Duanmu C J and Zou C R 2000 A multilevel successive elimination algorithm for block matching motion estimation. *IEEE Trans. Image Process.* 9(3): 501–504
- [7] Yu-Wen Huang, Shao-Yi Chien, Bing-Yu Hsieh and Liang-Gee Chen 2004a Global elimination algorithm and architecture design for fast block matching motion estimation. *IEEE Trans. Circuits Syst. Video Technol.* 14(6): 898–907
- [8] Fan C-P and Lin S-W 2007 fast global elimination algorithm and lowcost VLSI design for motion estimation. *TENCON 2007–2007 IEEE Region 10 Conference*, November, pp 1–4
- [9] Huang Y-W, Tsai C-H and Chen L-G 2004b Parallel global elimination algorithm and architecture design for fast block matching motion estimation. *IEEE International Conference on Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04)* vol 5, pp. V-153–156
- [10] Griselda Saldaha and Miguel Arias-Estrada 2007 Compact fpga based systolic array architecture for architecture motion estimation using full search block matching algorithm. *Programmable logic, 2007. SPL'07.2007 3rd Southern conference*
- [11] Li Man Ho 2006 Variable block size motion estimation hardware for video encoders. (Master of Philosophy thesis) the Chinese University of Hong Kong
- [12] Sinan Yalcin, Hasan F Ates and Ilker HAMzoglu 2005 A high performance hardware architecture for a sad reuse based heirarchical motion estimation algorithm for H.264 video coding. *Field programmable logic and applications 2005*