New Motion Estimation Techniques and their SIMD Implementations for Video Coding

Chunjiang Duanmu

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy at
Concordia University
Montreal, Quebec, Canada

August 2005

Your file Votre référence
ISBN: 0-494-09971-2
Our file Notre référence
ISBN: 0-494-09971-2

ABSTRACT

New Motion Estimation Techniques and their SIMD Implementations for Video Coding

Chunjiang Duanmu, Ph. D.
Concordia University, 2005

Compression of video signals is of great importance to modern multi-media systems. In order to achieve efficient data compression, block motion estimation is generally employed to remove temporal redundancies inherent in video signals and thus, it is a crucial component of international video coding standards. This thesis aims at developing techniques to reduce the computational complexity of a given block motion estimation algorithm without sacrificing its accuracy, to utilize the single instruction multiple data (SIMD) technique to accelerate a block motion estimation process, and to develop a new fast block motion estimation algorithm suitable for implementation using SIMD architecture.

A method to detect blocks that are stationary between successive frames, is proposed. In this method, when a block is judged as stationary, the search process for such a block is skipped in the block motion estimation process. The statistical characteristics of the video sequence are utilized in deciding as to which blocks are stationary. Simulation studies are carried out showing that this method reduces the computational complexity of the various block motion estimation algorithms without sacrificing the accuracy of the original algorithm.

A vector-based fast block motion estimation algorithm, suitable for implementation on an SIMD architecture, is proposed. This algorithm maintains the

accuracy and coding efficiency of the full-search algorithm, but the complexity is only a very small fraction of that of the full-search algorithm. It is also shown that by implementing the proposed algorithm on an SIMD architecture, the execution time of the algorithm can be further reduced by about 74%.

The concept of an eight-bit partial sum is introduced so as to take advantage of the byte-type data parallelism in the existing SIMD architectures. A method of employing these partial sums to speedup a given block motion estimation process is proposed. The notion of the eight-bit partial sums is extended to the four-level case and it is shown that there are fifteen possible methods of utilizing these multi-level partial sums to accelerate block motion estimation algorithms. It is shown that any of these fifteen methods can accelerate a given block motion estimation algorithm without any loss of accuracy. The full-search algorithm is used to determine as to which one of these fifteen methods would provide the lowest computational complexity in order for it to be chosen to accelerate the various motion estimation algorithms. Simulation studies have been conducted and the results show that the proposed scheme is capable of providing a substantial speedup for the various existing motion estimation algorithms without any loss of accuracy.

ACKNOWLEDGEMENTS

This thesis would not have been written were it not for the help of numerous people throughout the past half-decade.

First and foremost, I would like to express my deepest gratitude to my supervisors, Dr. M. Omair Ahmad and Dr. M.N.S. Swamy, for their guidance, encouragement, and support during the course of this research. I very much appreciate the freedom they have given to explore new ideas and avenues to research. I also would like to record my sincere thanks to them for spending time with me and correcting the initial drat of this thesis which resulted in improvement of the presentation. The advice and comments of the committee members are gratefully acknowledged.

I am indebted to Dr. Wei-Ping Zhu of the Department of Electrical and Computer Engineering, Dr. A. Shatnawi who was a visiting scholar, and Dr. Yang-Li Wang who was a visiting scientist in the Center for Signal Processing and Communications for their fruitful discussions, suggestions and constant moral support.

I would like to thank Dr. H. L. Xiong and Dr. K. L. Du, who were post doctoral fellows at the Center for their friendship and encouragement. I would also like to acknowledge the support of Mr. Qinglin Zhang, Mr. Xiaojun Lu, and Mr. Wei Chu, the graduate students working with my supervisors.

Last but not the least, I am similarly grateful to my wife and my parents. Were it not for their support, inspiration and love, I would have never gotten to this point.

To my wife and my parents

TABLE OF CONTENTS

LIST OF FIGURES..... x

LIST OF TABLES..... xiii

LIST OF ABBREVIATIONS..... xv

LIST OF SYMBOLS..... xvii

Chapter 1 Introduction..... 1

1.1 Structure of a Video Coding System..... 2

1.2 Scope and Organization of the Thesis..... 4

Chapter 2 Block Motion Estimation Algorithms..... 7

2.1 Block Motion Estimation Process and the Full-Search Algorithm..... 8

2.2 Fast Block Motion Estimation Algorithms..... 10

2.3 Performance Comparison..... 18

2.4 Summary..... 24

Chapter 3 A Method for Fast Block Motion Estimation by Exclusion of Stationary Macroblocks from the Search Process..... 25

3.1 Proposed Method..... 26

3.2 Criterion for the Detection of Stationary Macroblocks..... 30

3.3 Adaptive Threshold Value..... 31

3.4 Simulation Results..... 33

3.5 Summary..... 47

Chapter 4 A Vector-Based Fast Block Motion Estimation Algorithm..... 48

4.1 The Vector-Based Motion Estimation Algorithm..... 49

4.1.1 Formation of Partial Sums..... 49
4.1.2 Lower Bounds for the MAD..... 52
4.1.3 Fast Method to Compute the Partial Sums..... 54
4.1.4 Algorithm..... 58
4.2 SIMD Implementation of the Proposed Algorithm..... 59
4.3 Computational Complexity of the Proposed Algorithm..... 62
4.3.1 Computational Complexity of $MAD^l(\vec{V})$ for a Gven l, $0 \le l \le 5$ 63
4.3.2 Computational Complexity to Calculate the Frame Partial Sums..... 65
4.3.3 Computational Complexity of the Algorithm..... 65
4.3.4 Theoretical Speedup..... 66
4.3.5 Practical Speedup..... 66
4.4 Simulation Results..... 67
4.5 Summary..... 73

Chapter 5 Fast Block Motion Estimation with Eight Bit Partial Sums using SIMD Architectures..... 74

5.1 Eight Bit Partial Sums of Sixteen Luminance Values..... 75
5.2 Multi-Level Eight-Bit Partial Sums..... 81
5.2.1 Formation of Multi-Level Eight-Bit Partial Sums..... 81
5.2.2 Upper Bounds for the SADs..... 84
5.2.3 The Methods of Using Multi-Level Eight-Bit Partial Sums..... 86
5.2.4 Optimal Method of Using Multi-Level Eight-Bit Partial Sums..... 88
5.2.5 SIMD Implementation for the Computation of SAD..... 91
5.3 Computational Complexity..... 92
5.3.1 Computational Complexity of a Block Motion Estimation Algorithm Incorporating the Proposed Scheme..... 92
5.3.2 Theoretical Speedup of a Block Motion Estimation Algorithm Incorporating Scheme 5.2..... 93

5.3.3 Practical Speedup of a Block Motion Estimation Algorithm Incorporating Scheme 5.2 94

5.4 Simulation Results 94

5.5 Summary 95

Chapter 6 Conclusion 108

References 112

Appendix A Block Motion Estimation Algorithms 120

A.1 Full-Search Algorithm 120

A.2 Two-Dimensional Logarithmic Search Algorithm 121

A.3 Orthogonal Search Algorithm 122

A.4 One at a Time Search Algorithm 123

A.5 Conjugate Direction Search Algorithm 124

A.6 Three-Step Search Algorithm 125

A.7 Four-Step Search Algorithm 126

A.8 Unrestrictive Center-Biased Diamond Search Algorithm 127

A.9 Predicative Search Area Algorithm 128

A.10 Block-Based Gradient Descent Search Algorithm 129

A.11 Selective Elimination Algorithm 130

Appendix B Computational Complexity of Block Motion Estimation Algorithms 131

B.1 Computational Complexity of the FSA and the Multi-Step Search Algorithms 131

B.2 Computational Complexity of the SEA 133

LIST OF FIGURES

Figure 1.1	Video encoder structure.....	3
Figure 2.1	Block motion estimation.....	9
Figure 2.2	Search pattern of the 2-D logarithmic search algorithm.....	12
Figure 2.3	Search patterns of the orthogonal search algorithm.....	12
Figure 2.4	Search pattern of the three-step search algorithm.....	14
Figure 2.5	Search patterns of the four-step search algorithm.....	15
Figure 2.6	Large diamond search pattern (LDSP).....	16
Figure 2.7	Small diamond search pattern (SDSP).....	16
Figure 2.8	Determination of the search area in the predicative search area algorithm.....	17
Figure 2.9	Rate-distortion performances of the various algorithms.....	23
Figure 3.1	Percentage of stationary macroblocks for the *Trevor* sequence....	27
Figure 3.2	Percentage of stationary macroblocks for the *Foreman* sequence.....	27
Figure 3.3	Percentage of stationary macroblocks for the *Miss America* sequence.....	28
Figure 3.4	Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Salesman* sequence.....	34
Figure 3.5	Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Hall Objects* sequence.....	34
Figure 3.6	Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Silent* sequence.....	35

Figure 3.7 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Akiyo* sequence..... 35

Figure 3.8 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *News* sequence..... 36

Figure 3.9 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Foreman* sequence..... 36

Figure 3.10 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Trevor* sequence..... 37

Figure 3.11 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Mother & Daughter* sequence..... 37

Figure 3.12 Percentage of *speedup* and *error in judgment* of proposed method combined with the 3SSA for the *Mother & Daughter* sequence..... 38

Figure 3.13 Percentage of *speedup* and *error in judgment* of proposed method combined with the 4SSA for the *Mother & Daughter* sequence..... 38

Figure 3.14 Percentage of *speedup* and *error in judgment* of proposed method combined with the UDSA for the *Mother & Daughter* sequence..... 39

Figure 3.15 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Salesman* sequence..... 41

Figure 3.16 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Hall Objects* sequence..... 42

Figure 3.17 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Silent* sequence..... 42

Figure 3.18 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Akiyo* sequence..... 43

Figure 3.19 Rate-distortion performances of FSA and proposed method combined with the FSA for the *News* sequence..... 43

Figure 3.20 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Foreman* sequence..... 44

Figure 3.21 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Trevor* sequence..... 44

Figure 3.22 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Mother & Daughter* sequence..... 45

Figure 3.23 Rate-distortion performances of 3SSA and proposed method combined with the 3SSA for *the Mother & Daughter* sequence..... 45

Figure 3.24 Rate-distortion performances of 4SSA and proposed method combined with the 4SSA for the *Mother & Daughter* sequence..... 46

Figure 3.25 Rate-distortion performances of UDSA and proposed method combined with the UDSA for the *Mother & Daughter* sequence..... 46

Figure 4.1 Packed word-type subtraction on SIMD registers..... 62

Figure 5.1 Packed byte-type addition on SIMD registers..... 91

Figure A.1 Current block B_c and its four neighboring blocks B_1, B_2, B_3, and B_4..... 128

LIST OF TABLES

Table 2.1	Computational complexity of the various algorithms.....	20
Table 4.1	Fast method to calculate the frame partial sums.....	57
Table 4.2	A vector-based fast block motion estimation algorithm.....	60
Table 4.3	Employment of SIMD technique for the calculations of frame partial sums.....	61
Table 4.4	Computational complexity of $MAD^l(\vec{V})$, $0 \le l \le 5$	64
Table 4.5	The average number of times per block that $MAD^l(\vec{V})$ needs to be calculated in the proposed VFA.....	69
Table 4.6	Theoretical speedup of the proposed VFA over the FSA and SEA without SIMD implementation.....	70
Table 4.7	Practical speedup of the proposed VFA over FSA and SEA without SIMD implementation.....	71
Table 4.8	Practical speedup of the proposed VFA using SIMD architecture.....	72
Table 5.1	Conditions used for the various methods.....	88
Table 5.2	Computational complexity of the various methods.....	89
Table 5.3	Computational complexity and average number of CPU cycles per block using the FSA.....	96
Table 5.4	Computational complexity and average number of CPU cycles per block using the 2DLSA.....	97
Table 5.5	Computational complexity and average number of CPU cycles per block using the OSA.....	98
Table 5.6	Computational complexity and average number of CPU cycles per block using the OATSA.....	99
Table 5.7	Computational complexity and average number of CPU cycles per block using the CDSA.....	100

Table 5.8 Computational complexity and average number of CPU cycles per block using the 3SSA..... 101

Table 5.9 Computational complexity and average number of CPU cycles per block using the 4SSA..... 102

Table 5.10 Computational complexity and average number of CPU cycles per block using the UDSA..... 103

Table 5.11 Computational complexity and average number of CPU cycles per block using the PSAA..... 104

Table 5.12 Computational complexity and average number of CPU cycles per block using the BGDSA..... 105

Table 5.13 Computational complexity and average number of CPU cycles per block using the SEA..... 106

Table A.1 Full-search algorithm..... 120

Table A.2 2-D logarithmic search algorithm..... 121

Table A.3 Orthogonal search algorithm..... 122

Table A.4 One at a time search algorithm..... 123

Table A.5 Conjugate direction search algorithm..... 124

Table A.6 Three-step search algorithm..... 125

Table A.7 Four-step search algorithm..... 126

Table A.8 Unrestricted center-biased diamond search algorithm..... 127

Table A.9 Block-based gradient descent search algorithm..... 129

Table A.10 Selective elimination algorithm..... 130

LIST OF ABBREVIATIONS

2DLSA	Two-dimensional logarithmic search algorithm
3SSA	Three-step search algorithm
4SSA	Four-step search algorithm
BGDSA	Block-based gradient descent search algorithm
CCITT	Consultative Committee of International Telegraph and Telephone
CDSA	Conjugate direction search algorithm
DCT	Discrete cosine transform
DWT	Discrete wavelet transform
FSA	Full-search algorithm
LDSP	Large diamond search pattern
MAD	Mean absolute difference
MMAD	Modified mean absolute difference
MPEG	Moving Picture Experts Group
OATSA	One at a time search algorithm
OSA	Orthogonal search algorithm
PS	Practical speedup
PSAA	Predictive search area algorithm
PSNR	Peak signal to noise ratio
PSTN	Public service telephone network
QCIF	Quarter common intermediate format

RCC	Reduction in the computational complexity
RUC	Region of uncertainty
SAD	Sum of the absolute differences
SDSP	Small diamond search pattern
SEA	Selective elimination algorithm
SIMD	Single instruction multiple data
TMN	Test model near-term: This was the name of the codec for the simulation model used during the development of the H.263 video compression standard.
UB	Upper bound
UDSA	Unrestricted center-biased diamond search algorithm
VFA	Vector-based fast block motion estimation algorithm
VLSI	Very large scale integration

LIST OF SYMBOLS

B_c	Current block
B_r	Reference block
$b \times b$	Size of a block
(x, y)	Index of a pixel in a frame
$I_c(x, y)$	Luminance value of pixel (x, y) in the current frame
$I_r(x, y)$	Luminance value of pixel (x, y) in the reference frame
(x_0, y_0)	Top-left corner of a block
$\vec{V} = (V_x, V_y)$	Motion vector
$MAD(\vec{V})$	Mean absolute difference corresponding to the motion vector $\vec{V} = (V_x, V_y)$
$B_c(m, n)$	Luminance value of pixel (m, n) in the current block
$B_r(m, n, \vec{V})$	Luminance value of pixel (m, n) in the reference block corresponding to the motion vector $\vec{V}$
$S \times S$	Size of search window
P	Set of all candidate motion vectors inside the search window
$\vec{V}_{op}$	Optimum motion vector
$\eta_{MAD}(x)$	The number of times $MAD(\vec{V})$ needs to be calculated per block of the block motion estimation algorithm x
$C(x)$	Computational complexity per pixel of the block motion estimation algorithm x
Q	Quantization step size
$R(Q)$	Total number of output bits for the encoder

$PSNR(Q)$	Peak signal to noise ratio
$MSE(Q)$	Mean square error between the original video sequence and the reconstructed video sequences
$f_o^k(m,n)$	Luminance value at the position (m,n) in the k-th frame of the original video sequence
$f_r^k(m,n,Q)$	Luminance value at the position (m,n) in the k-th frame of the reconstructed sequence
$MAD_{\min}$	Minimum value of MAD
W	Width of a video frame
H	Height of a video frame
T	Predetermined threshold
N_u	The number of the stationary macroblocks undetected in the previous frame
$MAD_l(0,0)$	The value of $MAD(0,0)$ of the l-th undetected stationary macroblock in the previous frame
T_1, and T_2	Values for the determination of the adaptive threshold
N_e	The number of non-stationary macroblocks for which the values of $MAD_{\min}$ lie between T and T_1 in the previous frame
$MAD_{\min}^k$	The value of $MAD_{\min}$ of the k-th non-stationary macroblocks for which the values of $MAD_{\min}$ lie between T and T_1 in the previous frame
P_0, P_1, P_2, and P_3	Parameter values used in the determination of the value of the adaptive threshold
$LB_{SEA}(\bar{V})$	Lower bound for the MAD in the selective elimination algorithm
(p,q)	Index of data sets or partial sums
$\psi^l(p,q)$	Data set at the l-th level $(1 \le l \le 5)$

Ψ^0	Set of all the pixels in a 16×16 block
$S_c^l(p,q)$	Partial sum at the l-th level ($1 \le l \le 5$) of the current block
$S_r^l(p,q,\vec{V})$	Partial sum at the l-th level ($1 \le l \le 5$) of the reference block
S_c^0	Partial sum at the 0-th level of the current block
$S_r^0(\vec{V})$	Partial sum at the 0-th level of the reference block
$MAD^l(\vec{V})$	Mean of the absolute differences of the partial sums at the l-th level ($0 \le l \le 5$)
$F_c^l(m,n)$	Frame partial sum at level l ($0 \le l \le 5$) at the location (m,n) in the current frame
$F_r^l(m,n)$	Frame partial sum at level l ($0 \le l \le 5$) at the location (m,n) in the reference frame
$CS(n)$	Set of all the candidate motion vectors checked so far in a motion estimation process
$\vec{V}_{op}(n)$	Optimum motion vector in the set $CS(n)$
$\vec{V}_{curr}$	Current candidate motion vector to be checked in a motion estimation process
$n_a(l)$	The number of additions in the computation of $MAD^l(\vec{V})$
$n_s(l)$	The number of subtractions in the computation of $MAD^l(\vec{V})$
$n_{ab}(l)$	The number of absolute value calculations in the computation of $MAD^l(\vec{V})$
$n_t(l)$	Computational complexity of $MAD^l(\vec{V})$ per pixel
$m_t(l)$	The average number of times per block that $MAD^l(\vec{V})$ needs to be computed

$C(VFA)$	Computational complexity of the proposed vector-based fast block motion estimation algorithm
η_{FSA}^{VFA}	Computational complexity of the proposed vector-based fast block motion estimation algorithm as a percentage of the computational complexity of the full-search algorithm
η_{SEA}^{VFA}	Computational complexity of the proposed vector-based fast block motion estimation algorithm as a percentage of the computational complexity of the selective elimination algorithm
PS_{FSA}^{VFA}	Execution time of the proposed vector-based fast block motion estimation algorithm as a percentage of that of the full-search algorithm
PS_{SEA}^{VFA}	Execution time of the proposed vector-based fast block motion estimation algorithm as a percentage of that of the selective elimination algorithm
$PS(SIMD)$	Execution time of the proposed vector-based fast block motion estimation algorithm using an SIMD architecture as a percentage of the execution time of this algorithm without an SIMD implementation
$BS_{16}^{c}(n)$	Partial sum of sixteen luminance values corresponding to the current block
$BS_{16}^{r}(n,\vec{V})$	Partial sum of sixteen luminance values corresponding to the reference block
$PS_{c}(n)$	Eight-bit partial sums of sixteen luminance values for the current block
$PS_{r}(n,\vec{V})$	Eight-bit partial sums of sixteen luminance values for the reference block
$[A] >> [B]$	Shift A to the right by B bits
$MMAD(\vec{V})$	Modified mean absolute difference (MMAD) corresponding to the eight-bit partial sums of sixteen luminance values

$LB(\vec{V})$	A lower bound for $MMAD(\vec{V})$
$MMAD_{\min}$	Minimum value of MMAD computed so far in a block motion estimation process
$MBS_c^l(m,n)$	Multi-level eight-bit partial sums for the current block
$MBS_r^l(m,n,\vec{V})$	Multi-level eight-bit partial sums for the reference block
$SAD^l(\vec{V})$	Sum of the absolute differences of all the eight-bit partial sums between the current and reference blocks at the l-th level
$UB^l(\vec{V})$	An upper bound for $SAD^l(\vec{V})$
$\eta'_{LB}(x)$	The number of times $SAD^1(\vec{V})$ needs to be calculated in the block motion estimation algorithm x incorporating the use of Scheme 5.2
$\eta'_{MAD}(x)$	The number of times $MAD(\vec{V})$ needs to be calculated in the block motion estimation algorithm x incorporating Scheme 5.2
$C'(x)$	Computational complexity per pixel in the block motion estimation algorithm x incorporating Scheme 5.2
$C'_{ps}(H)$	Computational complexity per pixel to calculate all the eight-bit partial sums in a frame
$RCC(x)$	The percentage of the reduction in the computational complexity of algorithm x incorporating Scheme 5.2
SA	Search area

Chapter 1

Introduction

Video signals usually consist of more than 15 pictures (video frames) per second. This requires a huge amount of data to be stored, transmitted, and processed. In order to overcome the problems involving large volume of data, video compression becomes a necessity. For example, for a low resolution quarter common intermediate format (QCIF) video sequence, a network bandwidth of 6 Mbits/s is required for transmission, if no data compression is performed. To transmit this QCIF format video sequence over a public service telephone network (PSTN) with a bandwidth of 64 kbits/s, the video signal needs to be compressed by a factor of more than 100. Even for a network with a wider bandwidth, video compression can make the network to offer more services or to accommodate more users.

As a result of the requirement of video compression in multimedia communications, several international standards in video coding have been proposed to pave the way for ubiquitous applications of multimedia products. Block motion estimation is an essential part of these standards. However, these standards do not specify the exact motion estimation algorithm to be employed and is left to the developers implementing the standards. For this reason, the development of fast block motion estimation algorithms with good compression efficiency has been a focus of recent research activities and is expected to continue to attract a great deal of research effort in the near future.

1.1 Structure of a Video Coding System

The structure of a typical video encoder, such as that of MPEG-1 [56, 57], MPEG-2 [31, 58-60, 74], MPEG-4 [2, 3, 32, 61-63, 70, 88, 90], H.261 [25, 33, 53], H.263 [8, 25, 34, 93, 98], H.263++ [8], or H.26L [35], is as shown in Figure 1.1. The main objective of an encoder is to use as few output bits as possible to represent the original video sequence, given a distortion requirement of the reconstructed video sequence. To achieve this objective, block motion estimation [9, 20, 23, 27, 64, 66, 83, 94] is employed to remove the temporal redundancies of a video sequence, while the discrete cosine transform (DCT) [11, 78, 80] or discrete wavelet transform (DWT) [6, 79, 84, 91] is utilized to remove the spatial redundancies.

The correlation between video frames is generally utilized to compress the digital video. For example, a large number of blocks in the current frame can be represented by their corresponding regions (reference blocks) in a previously reconstructed video frame. This reconstructed video frame is referred to as the reference frame in the literature. The basic idea of motion estimation and compensation is to use a reference block to represent the current block. On the encoder side, the process of finding a good reference block for the current block is called block motion estimation. The translational motion of a reference block from the reference to the current frame is indicated by a motion vector. The information of the motion vector is encoded and sent to the decoder, so that the decoder at the receiver end can retrieve the reference block. The process of retrieving a reference block is called block motion compensation. Since the number of bits required to represent the motion vectors are, in general, significantly less than those required to

represent the current frame directly, a large number of bits can be saved to represent the current frame by utilizing the technique of block motion estimation and compensation.

Most of the times, the prediction error, that is, the error between the current block and its reference block is large and in addition, there are spatial correlations between the prediction errors. For this reason, the prediction errors are transformed by utilizing the DCT or DWT to remove the spatial correlations between the prediction errors. The quantized DCT or DWT coefficients are encoded and sent to the decoder, so that the decoder at the receiver can use the encoded bits of the prediction errors to enhance the quality of the reconstructed video sequence.

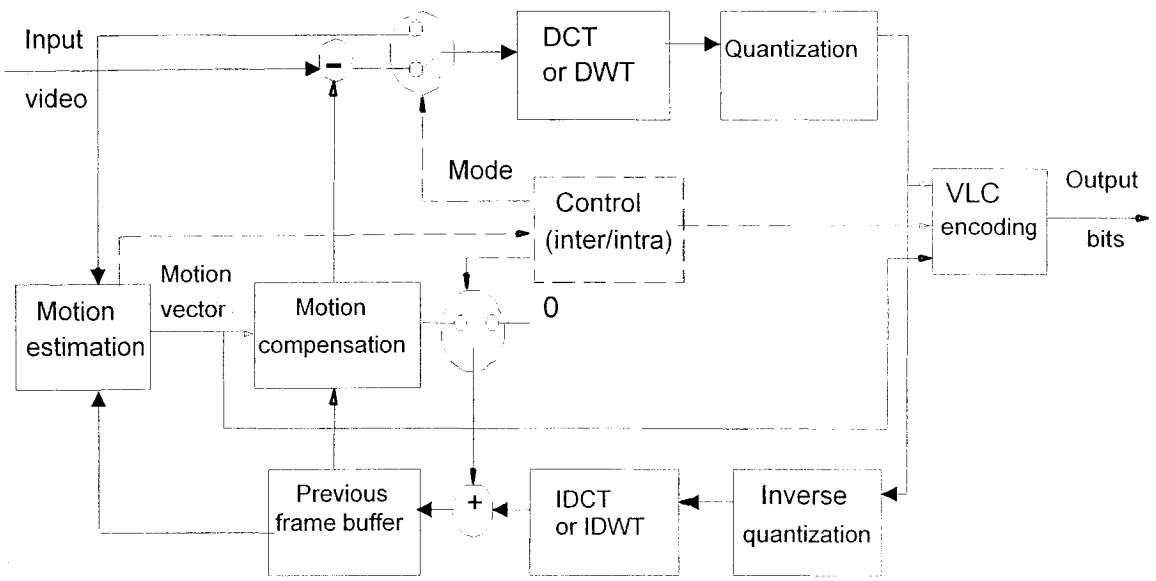


Figure 1.1 Video encoder structure

In Figure 1.1, there is also a mode decision block to decide the optimal operational modes and parameters of the encoder. In the inter mode of the current block, the process of motion estimation is first performed and the information of the resulting motion

vectors are encoded. Then, the prediction errors are transformed, quantized and encoded. In the intra mode of the current block, the process of motion estimation is not employed and the current block is directly transformed, quantized and encoded. The inter mode is suitable for all situations, except for that in which the scene changes. In the latter situation, the block motion estimation cannot give a good prediction of the current block, and the intra mode is more suitable. The information regarding these modes and parameters is sent to the decoder in addition to the information about the motion vectors and prediction errors, so that the received bits can be correctly decoded.

1.2 Scope and Organization of the Thesis

The international video coding standards, as discussed earlier, do not specify the algorithm for the block motion estimation, and is left to the individual developer implementing the standard. In this way, products from different vendors with varying performance can compete with one another. Thus, it is essential to develop fast block motion estimation algorithms providing good accuracy.

The objective of this thesis is to develop techniques for reducing the computational complexity of a given block motion estimation algorithm without sacrificing its accuracy, to utilize the single instruction multiple data (SIMD) technique [36-42, 86] to accelerate a block motion estimation process, and to develop a new fast block motion estimation algorithm suitable for implementation on SIMD architectures. The thesis is organized as follows.

In Chapter 2, we first briefly describe the block motion estimation process and the full-search algorithm. Then, some typical fast block motion estimation algorithms in the

literature are reviewed. In order to evaluate the compression efficiency of a block motion estimation algorithm, the concept of rate-distortion curve is discussed and used to asses the coding efficiency of a video codec. Simulations on the existing block motion estimation algorithms are carried out for the purpose of comparing their computational complexity as well as compression efficiencies.

Based on our observation that there are a large number of stationary blocks that do not experience motion between video frames, and the search process in block motion estimation is unnecessary for such a block, in Chapter 3 a method has been proposed to detect and skip altogether the search process for such stationary blocks in a motion estimation process. The proposed method is then applied to a number of existing block motion estimation algorithms to reduce the computational requirement of the algorithms.

As the SIMD technique provides an option to accelerate some of the execution processes by utilizing data-parallelism, a vector-based fast block motion estimation algorithm, suitable for SIMD implementation, is proposed in Chapter 4. It is proved that this algorithm has the same accuracy and coding efficiency as that of the full-search algorithm.

In order to take advantage of the byte-type data parallelism in the existing SIMD technique, we introduce the concept of eight-bit partial sums in Chapter 5. Since these partial sums are of only eight bits, eight of them can be processed concurrently in a single 64-bit SIMD register. A method of employing these partial sums to speedup a given block motion estimation algorithm is then described. The notion of the eight-bit partial sums is extended to the four-level case and it is shown that there are fifteen possible methods of utilizing these multi-level eight-bit partial sums to accelerate a block motion

estimation algorithm without any loss of accuracy. Each of these fifteen methods is used in the full-search algorithm to determine the one that provides the lowest computational complexity. The one with the lowest computational complexity is adopted to accelerate the various block motion estimation algorithms. Simulations are performed for this scheme on typical video test sequences.

Chapter 6 concludes the thesis by highlighting the contributions of this investigation and provides some suggestions for further work.

Chapter 2

Block Motion Estimation Algorithms

The objective of a block motion estimation algorithm is to obtain a good reference block for the current block under the constraint that the resulting motion vector is inside a search window in the reference frame (the previous reconstructed frame). The similarity between the reference and current blocks is usually measured by their mean absolute difference (MAD). The full-search algorithm [20] is a brute-force algorithm, since for every candidate motion vector inside the search window a computation of the MAD is carried out.

Since the full-search algorithm (FSA) for block motion estimation has a very high computational complexity, many fast block motion estimation algorithms have been proposed in the literature. These algorithms can generally be categorized into either multi-step or exhaustive search algorithms. For a multi-step search algorithm, only a subset of the candidate motion vectors inside the search window is selected and thus, the number of MAD computations is reduced. For an exhaustive search algorithm, the computation of a lower bound for MAD is executed to circumvent the calculation of MAD, wherever possible.

In this chapter, we first describe the block motion estimation process and the full-search algorithm in Section 2.1. Then, we briefly review some typical fast block motion

estimation algorithms in Section 2.2. Simulation studies are carried out in Section 2.3 in order to compare the various block motion estimation algorithms both in terms of their computational complexity as well as their rate-distortion performance.

2.1 Block Motion Estimation Process and the Full-Search Algorithm

In a block matching algorithm, the current frame is divided into a number of fixed-sized blocks and the motion vector for each current block, say B_c, with size $b \times b$ where b is a power of 2, is determined by finding the best possible matching reference block in the reference frame according to a defined matching criterion. In existing video coding standards [7, 8, 27, 30, 56, 65], b is chosen as 16 and the matching criterion used is the MAD. We define the luminance values of the current and reference frames as $I_c(x, y)$ and $I_r(x, y)$ respectively, where x and y are the row and column indices of a pixel in a frame. The location of the top-left corner of the current macroblock is denoted as (x_0, y_0). Then, the MAD corresponding to the candidate motion vector $\vec{V} = (V_x, V_y)$ can be expressed as

$$MAD(\vec{V}) = \sum_{n=0}^{15} \sum_{m=0}^{15} | B_c(m,n) - B_r(m,n,\vec{V}) | \tag{2.1}$$

where

$$\begin{cases} B_r(m,n,\vec{V}) = I_r(x_0 + V_x + m, y_0 + V_y + n) \\ B_c(m,n) = I_c(x_0 + m, y_0 + n) \end{cases} \tag{2.2}$$

In (2.1), without loss of generality, we have not included the division by the factor 256.

In general, a search window of size $S \times S = (2w+1) \times (2w+1)$ is used to find the best matching block and confine the search in the reference frame to a limited region.

Thus, the value of the motion vector gets restricted to $-w \le V_x,\ V_y \le w$ and the candidate motion vectors inside the search widow ($S \times S$) form the set

$$P = \{(V_x, V_y) \mid -w \le V_x,\ V_y \le w\} \tag{2.3}$$

The value of w commonly used in video coding standards is 15, making the search window size to be (31×31).

Figure 2.1 depicts the current macroblock, the reference block, the search window, and the motion vector involved in the block motion estimation process. As seen from Figure 2.1, the displacement from the location of the top-left corner of the current macroblock to the top-left corner of the reference block in the reference frame is the corresponding motion vector for the current macroblock.

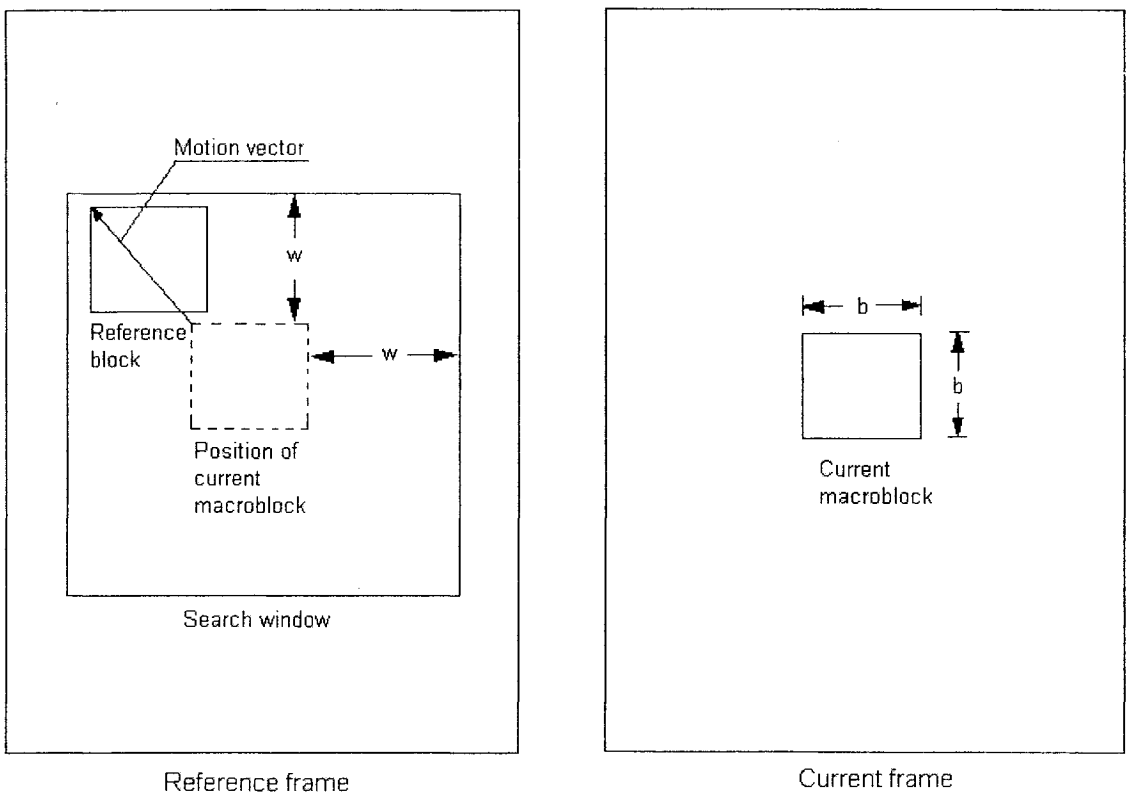


Figure 2.1 Block motion estimation

In the full-search algorithm (FSA) for block motion estimation, for every $\vec{V} \in P$, the block matching computation of (2.1) is executed . The candidate motion vector $\vec{V}$ with the minimum value of MAD is selected as the optimum motion vector $\vec{V}_{op}$ for the current macroblock , i.e.,

$$\vec{V}_{op} = \arg\{\min_{\vec{V} \in P}[MAD(\vec{V})]\} \tag{2.4}$$

The FSA is given in Section A.1 of Appendix A.

2.2 Fast Block Motion Estimation Algorithms

The block matching computation of (2.1) is very time consuming; it needs 256 subtractions, 256 operations to calculate absolute values, and 255 additions. For a search window of size 31×31, 961 block matching computations are required for the current macroblock. For the QCIF format video sequence, there are 11×9 macroblocks in a video frame. Thus, for a video sequence in this format, the FSA requires, for each video frame, $256 \times 31 \times 31 \times 11 \times 9 = 24{,}355{,}584$ subtractions, 24,355,584 operations to calculate the absolute values, and $255 \times 31 \times 31 \times 11 \times 9 = 24{,}260{,}445$ additions. Moreover, there are at least 10 video frames per second in real time applications. This huge amount of computational requirement hinders the usage of the FSA in real time applications. Hence, a number of fast block motion estimation algorithms have been proposed in the literature. Most of these can be grouped into multi-step and exhaustive search algorithms.

1) Multi-step search algorithms

The 2-D logarithmic search algorithm [45], orthogonal search algorithm [75], one at a time search algorithm [82], conjugate direction search algorithm [82], three-step search

algorithm [49], four-step search algorithm [73], unrestricted center-biased diamond search algorithm [89, 100], predictive-area search algorithm [10], and the block-based gradient-descent search algorithm [54] are fast block motion estimation algorithms belonging to this category. In this category of fast algorithms, the motion estimation process consists of several search steps. At each search step, a number of locations are selected, where the values of the MAD are calculated.

The motion estimation process in the 2-D logarithmic search algorithm (2DLSA) [45] is executed by successively reducing the search area. Each step in the algorithm has five search locations, as shown in Figure 2.2. At each location, a block matching computation of MAD is carried out. At the first step, the search center is located at the center of the search window. At the next step, the search center is moved to the location corresponding to the minimum value of MAD at the previous step. The search length of the initial step is $\left\lfloor \frac{S-1}{4} \right\rfloor$, where S is the length of the search window. Whenever the search center has the minimum value of MAD, the search length is reduced to one-half of its previous value. This procedure continues until the search length is unity. In the final step, all the neighboring eight locations around the search center are checked and the optimum motion vector corresponds to the location with the minimum value of MAD among all the eight searched locations. This algorithm is given in Section A.2 of Appendix A.

The goal of the orthogonal search algorithm (OSA) is to minimize the total number of search locations in the worst case [75]. In this algorithm, vertical and horizontal search patterns are utilized alternatively, as shown in Figure 2.3. At every search step excluding the first one, the center of the search pattern is the location with the minimum value of MAD computed so far in the previous step. After both vertical and horizontal search

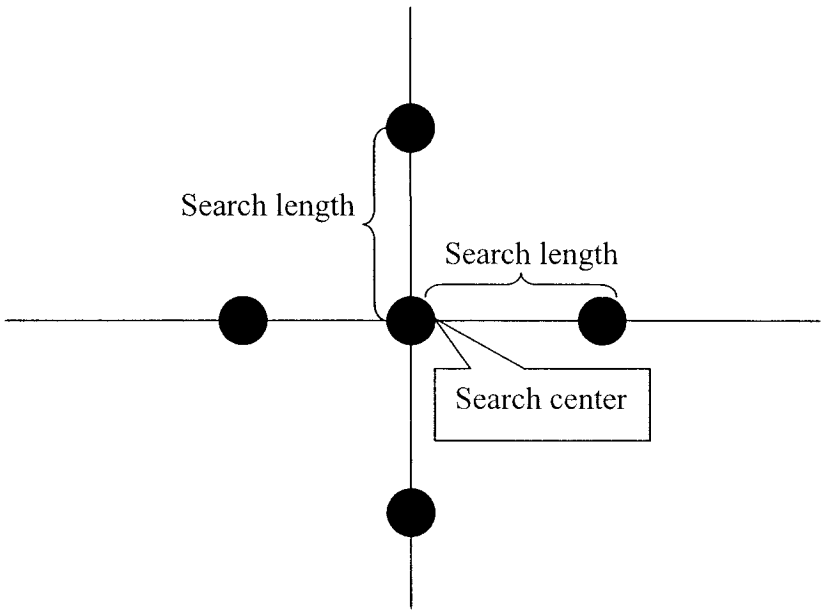
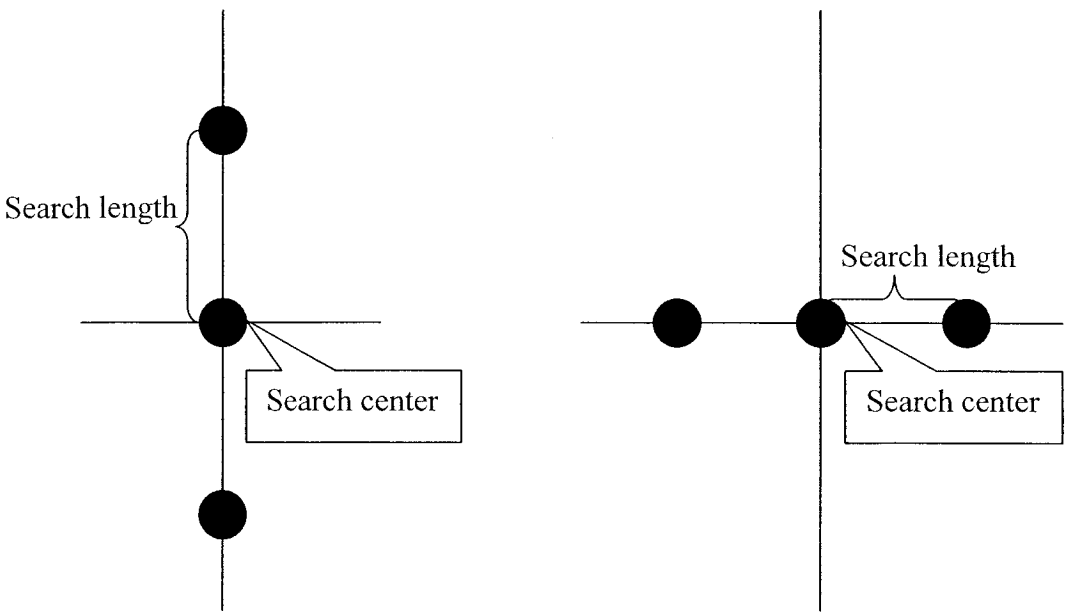


Figure 2.2 Search pattern of the 2-D logarithmic search algorithm



(a) Vertical search pattern (b) Horizontal search pattern

Figure 2.3 Search patterns of the orthogonal search algorithm

patterns at a given search length are utilized, the search length is reduced by one-half of its previous value and the search process continues until the search length is unity. This algorithm is given in Section A.3 of Appendix A.

The one at a time search algorithm (OATSA) has two search steps [82]. The direction of the search in each step is parallel to one of the coordinate axes, i.e., one component of the candidate motion vector is adjusted, the other being fixed. In the first step, the search begins along the horizontal search direction. When the local minimum value of MAD is found along the horizontal search direction, the search direction is changed to the vertical one. In the second step, the search starts from the location of minimum value of MAD in the previous step and ends at the local minimum value of MAD along the vertical direction. This algorithm is given in Section A.4 of Appendix A.

In the conjugate direction search algorithm (CDSA) [82], OATSA is first carried out. Then, at the location of the optimum motion vector in the OATSA, a third step of the search is carried out in the direction connecting the location (0,0) and the location of the optimum motion vector in the OATSA. This algorithm is given in Section A.5 of Appendix A.

In the three-step search algorithm (3SSA) [49], the search locations of the FSA are greatly reduced by tracking the direction of a locally optimum motion vector. For a search window of size $S \times S$, the initial search length is $\lceil S/4 \rceil$, where $\lceil x \rceil$ is the smallest integer equal to or larger than x. The shape of the search pattern of the 3SSA is shown in Figure 2.4. At every chosen search location, a computation of the MAD is carried out. In every search step excluding the initial one, the search length is one half of that of the previous step and the search center is moved to the search location with the minimum

value of the MAD computed so far. From the above discussion, we see that the 3SSA for block motion estimation needs to carry out only $\lceil \log_2 S - 1 \rceil$ search steps. For a search window of size 31×31, the total number of chosen search locations in the 3SSA is fixed at 33. Thus, in the 3SSA, the computational complexity of the FSA is reduced by a factor of about 30 for a 31×31 search window. This algorithm is given in Section A.6 of Appendix A.

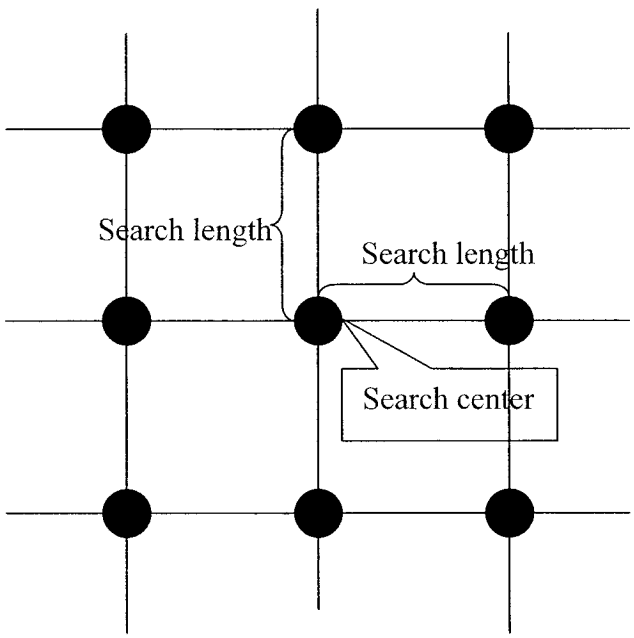
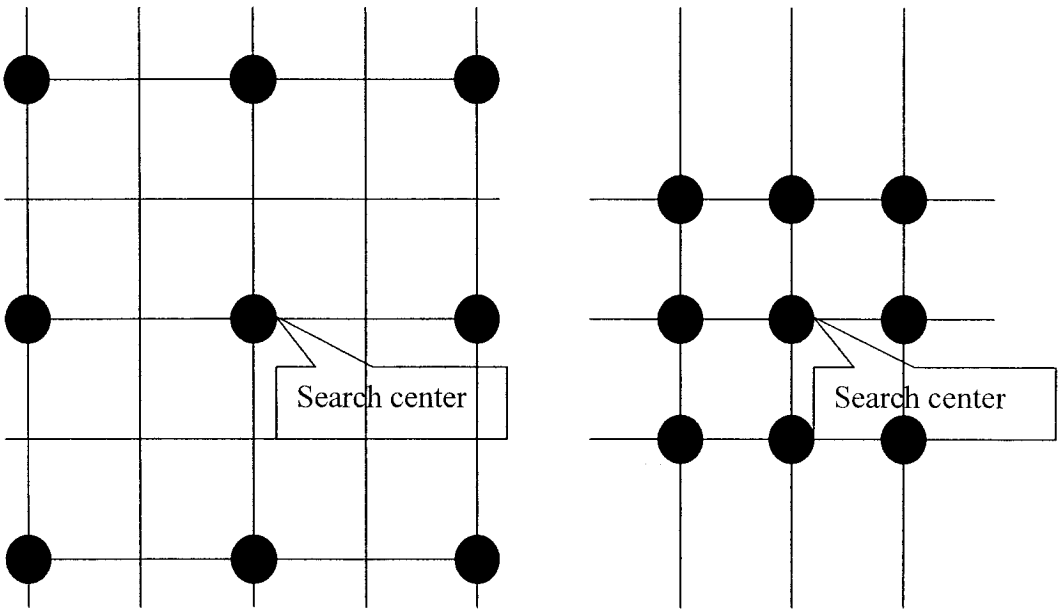


Figure 2.4 Search pattern of the three-step search algorithm

In the four-step search algorithm (4SSA) [73], two search patterns are utilized. In the fixed search pattern, the search length is fixed at 2 as shown in Figure 2.5(a). In the initial step of the 4SSA, the search center is located at the center of the search window. At each step of this algorithm, this fixed search pattern is recursively employed until the search center has the minimum value of MAD or the search process reaches the boundary of the search window. Similar to the 3SSA, the search center of every step is the location

with the minimum value of MAD in the previous step. When the search center has the minimum value of MAD or the search process reaches the boundary of the search window, the final search pattern of Figure 2.5(b) is employed to find the optimum motion vector. This algorithm is given in Section A.7 of Appendix A.



(a) Fixed search pattern (b) Final search pattern

Figure 2.5 Search patterns of the four-step search algorithm

In the unrestricted center-biased diamond search algorithm (UDSA) [89, 100], two search patterns are employed. In the large diamond search pattern (LDSP) shown in Figure 2.6, eight search locations surrounding the center one form the shape of a large diamond. In the small diamond search pattern (SDSP) shown in Figure 2.7, four search locations surrounding the center one form the shape of a small diamond. In this

algorithm, the LDSP is employed repeatedly until the location of the minimum value of the MAD is at the center of the LDSP. Then, the SDSP is utilized to find the optimum motion vector with the minimum value of the MAD among the five searched locations. This algorithm is given in Section A.8 of Appendix A.

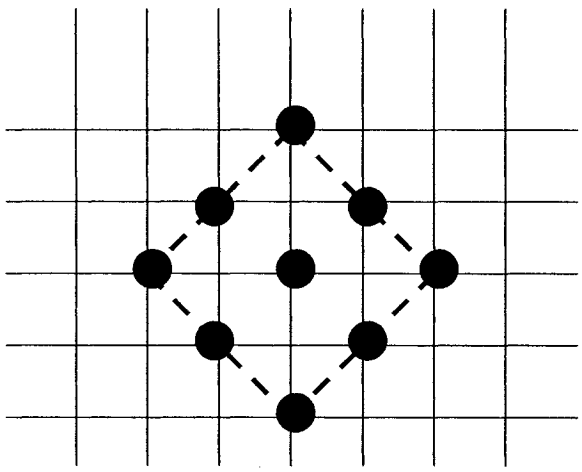


Figure 2.6 Large diamond search pattern (LDSP)

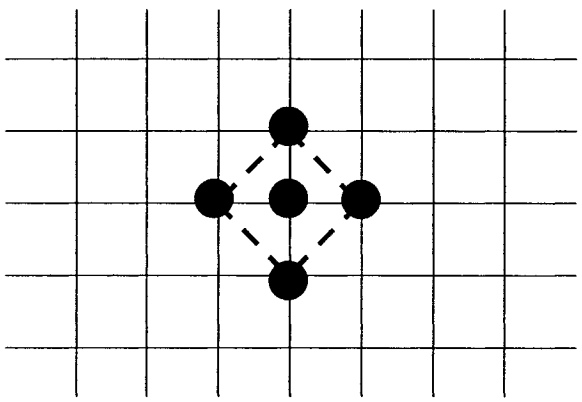


Figure 2.7 Small diamond search pattern (SDSP)

In some instances, the motion vectors between the neighboring blocks might be similar due to the reason that a large moving object may cover several neighboring blocks with similar translational motions between video frames. The predictive search area algorithm (PSAA) [10] tries to utilize the correlations between the neighboring motion vectors in such cases. The set of the candidate motion vectors in this algorithm is the union of the four sub-areas, where each sub-area lies in the region indicated by the neighboring motion vector, as shown in Figure 2.8. Thus, the PSAA reduces the computational complexity of the FSA by restricting the search area. This algorithm is given in Section A.9 of Appendix A.

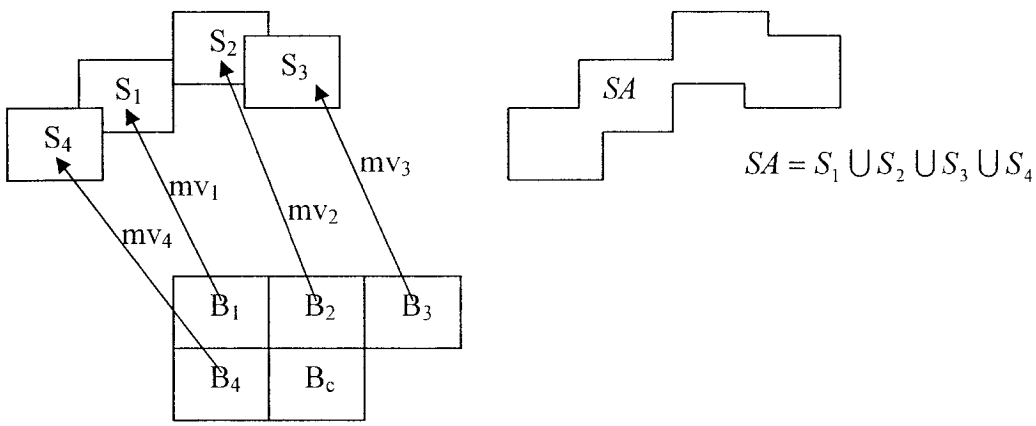


Figure 2.8 Determination of the search area in the predicative search area algorithm

In the block-based gradient descent search algorithm (BGDSA) [54], the search pattern is a 3×3 square. Initially, the center of the search pattern is at the location (0,0) in the search window. If the location with the minimum value of the MAD lies at the center of the search pattern of the 3×3 square, the search stops; otherwise, the center of the search pattern of the 3×3 square is moved to the location with the minimum value of

the MAD in the previous iteration. This algorithm is given in Section A.10 of Appendix A.

2) Exhaustive search algorithms

Since the values of the MAD inside a search window have several local minima instead of only one, these multi-step search algorithms are often trapped in a local minimum. For this reason, the multi-step search algorithms lose the accuracy of the FSA, and exhaustive search algorithms have been proposed.

The selective elimination algorithm (SEA) [52] is a fast block motion estimation algorithm belonging to the category of exhaustive search algorithms. It exhaustively searches every location inside the search window as the FSA does. However, it avoids the computation of the MAD whenever possible by calculating some lower bounds for the MAD. Since every search location is searched and the corresponding optimum motion vector is not excluded during the search process, the SEA can reduce the computational complexity and maintain the accuracy of the FSA. The SEA is given in Section A.11 of Appendix A.

2.3 Performance Comparison

(a) Compleixty

Simulation study of various block motion estimation algorithms is carried out for the CCITT test video sequences in the QCIF format using the TMN20 framework of the H.263 video codec. The encoding process is carried out for frames 1, 4, 7, ..., 97, 100, ..., that is, two frames are skipped in between the frames that are encoded.

In our simulation, eight video sequences, *Salesman, Car Phone, Silent, Akiyo, News, Foreman, Trevor, and Mother & Daughter*, are selected to evaluate the performance of the various block motion estimation algorithms. Each video sequence consists of various kinds of motion activities such as zooming, panning, and fast or slow motion. *Salesman, Silent,* and *Akiyo* are sequences having slow translational motion, whereas the remaining sequences have high translational motion. All the test video sequences consist of 300 frames, except for the sequences *Trevor* and *Mother & Daughter*, which have 150 and 800 frames respectively.

In order to find the computational complexity per pixel, we assign one unit for each of the operations of addition, subtraction, and right shift, and two units for each of the operations of comparison and taking the absolute value. This is a reasonable assumption on commonly used CPUs.

Let x denote a block motion estimation algorithm utilized in the block motion estimation process. Let $\eta_{MAD}(x)$ and $C(x)$ denote, respectively, the number of times $MAD(\vec{V})$ needs to be calculated per block and the computational complexity per pixel of the block motion estimation algorithm x in order to complete the estimation process. The computational complexity of algorithm x is derived in Appendix B, where x represents the algorithm FSA, 2DLSA, OSA, OATSA, CDSA, 3SSA, 4SSA, UDSA, PSAA, BGDSA or SEA.

Table 2.1 gives the simulation results concerning the computational complexity of the various algorithms for the eight video sequences mentioned earlier. In addition, the computational complexity of a given algorithm for a particular video sequence as a percentage of the computational complexity of the FSA is given in the parenthesis. From

Table 2.1 Computational complexity of the various algorithms

	Salesman	Car Phone	Silent	Akiyo	News	Foreman	Trevor	Mother & Daughter
C(FSA)	3847.75	3847.75	3847.75	3847.75	3847.75	3847.75	3847.75	3847.75
C(2DLSA)	68.39 (1.78%)	72.91 (1.89%)	70.27 (1.83%)	68.15 (1.77%)	69.39 (1.80%)	77.56 (2.02%)	71.67 (1.86%)	69.11 (1.80%)
C(OSA)	68.07 (1.77%)	68.07 (1.77%)	68.07 (1.77%)	68.07 (1.77%)	68.07 (1.77%)	68.03 (1.77%)	68.07 (1.77%)	68.07 (1.77%)
C(OATSA)	20.54 (0.53%)	24.70 (0.64%)	22.70 (0.59%)	20.26 (0.53%)	21.42 (0.56%)	28.07 (0.73%)	24.74 (0.64%)	21.46 (0.56%)
C(CDSA)	28.55 (0.74%)	32.87 (0.85%)	30.83 (0.80%)	28.27 (0.73%)	29.43 (0.76%)	39.36 (1.02%)	32.95 (0.86%)	29.51 (0.77%)
C(3SSA)	132.13 (3.43%)	132.13 (3.43%)	132.13 (3.43%)	132.13 (3.43%)	132.13 (3.43%)	132.13 (3.43%)	132.13 (3.43%)	132.13 (3.43%)
C(4SSA)	68.71 (1.79%)	77.08 (2.00%)	72.83 (1.89%)	68.31 (1.78%)	70.31 (1.83%)	89.09 (2.32%)	75.35 (1.96%)	70.07 (1.82%)
C(UDSA)	53.09 (1.38%)	65.10 (1.69%)	59.10 (1.54%)	52.49 (1.36%)	55.53 (1.44%)	81.36 (2.11%)	63.30 (1.65%)	55.25 (1.44%)
C(PSAA)	148.95 (3.87%)	183.30 (4.76%)	162.04 (4.21%)	145.38 (3.78%)	158.96 (4.13%)	206.00 (5.35%)	175.37 (4.56%)	157.71 (4.10%)
C(BGDSA)	37.96 (0.99%)	53.45 (1.39%)	46.61 (1.21%)	36.88 (0.96%)	40.56 (1.05%)	74.91 (1.95%)	52.05 (1.35%)	40.92 (1.06%)
C(SEA)	978.89 (25.44%)	823.14 (21.39%)	742.66 (19.30%)	543.67 (14.13%)	634.95 (16.50%)	1034.54 (26.89%)	932.84 (24.24%)	753.07 (19.57%)

this table, it can be seen that a multi-step search algorithm has a computational complexity that is substantially lower than that of the FSA. Among the 9 multi-step

search algorithms, the PSAA has the highest computational complexity and is about 4% of that of the FSA. The computational complexity per pixel of this algorithm ranges from 145 to 206 for the different test sequences. For the fast moving *Foreman* sequence, the computational complexity per pixel of the PSAA is 206. Thus, among the multi-step search algorithms, the PSAA has a relatively high computational requirement, especially for a fast moving sequence. It can also be seen that the computational complexity of 3SSA is the same for all the test sequences, showing the regularity of this algorithm. Due to this regularity, the 3SSA is sometimes selected in VLSI implementation. We also see that the 4SSA or UDSA has less computational complexity than that of the 3SSA. This is due to the fact that when the search center has the minimum value for the MAD, a final search step is executed in these two algorithms, in contract to the 3SSA where the search process continues. We see that the BGDSA has less computational complexity than that of the UDSA, since the search process is stopped in the BGDSA when the search center has the minimum value for the MAD. Since in the CDSA, the OATSA is first executed and then the search process is continued in a conjugate direction, we see that the CDSA has a little higher computational complexity than that of the OATSA. We also see that the SEA has one-forth to one-fifth the computational complexity of that of the FSA, and hence is also not suitable for real-time applications.

(b) Rate-distortion

In a video coding system, the bit-rate is measured by $R(Q)$, which is the total number of output bits for the encoder with a quantization step size of Q. The distortion of the system for the quantization step Q is measured by its peak signal to noise ratio (PSNR), which is defined as

$$PSNR(Q) = 10 * \log_{10}(\frac{255^2}{MSE(Q)}) \tag{2.5}$$

where $MSE(Q)$ is mean square error between the original video sequence and the reconstructed video sequences. The $MSE(Q)$ can be expressed as

$$MSE(Q) = \frac{1}{N}\sum_{k}\sum_{m,n}(f_r^k(m,n,Q) - f_o^k(m,n))^2 \tag{2.6}$$

where $f_o^k(m,n)$ and $f_r^k(m,n,Q)$ represent the luminance values at the position (m,n) in the k-th encoding frame of the original video sequence and the reconstructed sequence, respectively and N is the total number of pixels. Higher the value of the PSNR, lower the distortion.

For a fixed value of the quantization step size, Q, a rate-distortion pair $(R(Q), PSNR(Q))$ can be evaluated for a given video coding system. The rate-distortion curve is used to asses the compression efficiency of a video coding system.

The *Foreman* sequence in the QCIF format is selected for evaluating the rate-distortion performance of the multi-step search algorithms. Since the SEA has the same rate-distortion performance as the FSA, simulation of the rate-distortion performance for the SEA is not carried out. Figure 2.9 shows the rate-distortion performance of the multi-step search algorithms 2DLSA, OSA, OATSA, CDSA, 3SSA, 4SSA, UDSA, PSAA, and BGDSA. The rate-distortion performance of the FSA is also shown in this figure in order to make a comparison of a multi-step search algorithm with the FSA. From this figure, it can be seen that at the same level of distortion, a multi-step search algorithm needs 30 to 80 percent more bits than the FSA to encode the same sequence. Thus, a 30 to 80 percent

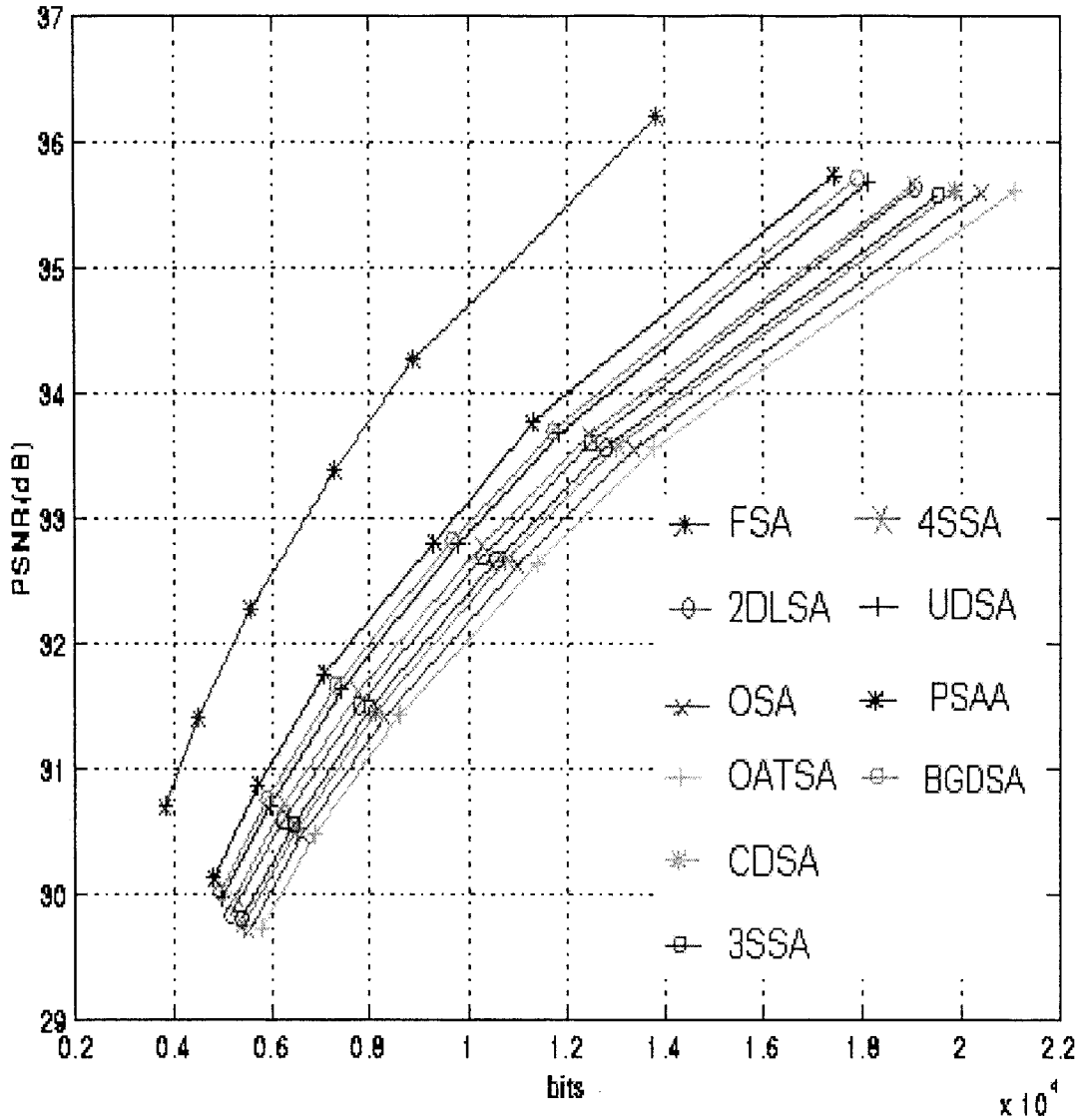


Figure 2.9 Rate-distortion performances of the various algorithms

of coding efficiency of the FSA is lost in a multi-step search algorithm. It can also be seen that among the 9 multi-step search algorithms, the PSAA has the best rate-distortion performance and the OATSA the worst. Since in the CDSA, the OATSA is first executed, it is seen that the CDSA has a better rate-distortion performance than the OATSA. We

also see that the CDSA has about the same rate-distortion performance as the 3SSA and that the 2DLSA has a better rate-distortion performance than the CDSA or 3SSA.

2.4 Summary

In this chapter, we have first carried out a review of the existing block motion estimation algorithms. Simulation studies of these block motion estimation algorithms have been conducted on the H.263 video codec using eight selected video test sequences. The simulation results have shown that the computational complexity of any of the multi-step search algorithms is substantially smaller than that of the FSA. This is achieved at the expense of reduced coding efficiency. Simulation results have also shown that the selective elimination algorithm has a high computational complexity as the full-search algorithm, and hence not suitable for real time applications.

Chapter 3

A Method for Fast Block Motion Estimation by Exclusion of Stationary Macroblocks from the Search Process

A typical video frame contains many macroblocks that experience little motion with respect to the reference frame. We shall refer to these macroblocks as stationary macroblocks. In this chapter, we aim at accelerating the block motion estimation process by excluding these stationary macroblocks from the search process, and thus reduce the computational requirement for the block motion estimation by avoiding the search process altogether for stationary macroblocks.

Details of the proposed method to reduce the computational complexity of a given block motion estimation algorithm are given in Section 3.1. The criterion to detect these stationary macroblocks is discussed in Section 3.2. In order to make the proposed method work efficiently for different video sequences, an adaptive threshold, described in Section 3.3, is employed. In Section 3.4, simulation results to demonstrate the effectiveness of the proposed method to accelerate a given block motion estimation algorithm are given. The rate-distortion performance of the H.263 video codec, with and without the proposed method, is discussed.

3.1 Proposed Method

As mentioned earlier, there are a number of stationary macroblocks in a typical video sequence and these are usually located in the background area or in the still objects of the video frames. The percentage of such stationary macroblocks in a frame is usually very high. In order to find the number of stationary macroblocks for a given frame, a simulation study is carried out. In this investigation, a macroblock is deemed as stationary when the optimum motion vector for the macroblock is (0,0) in the full-search algorithm. Figures 3.1, 3.2, and 3.3 depict the percentage of the stationary macroblocks for *Trevor, Foreman, and Miss America* test sequences, respectively. From these figures, it can be seen that the stationary macroblocks comprise more than 50, 20 and 40 percent of all the macroblocks for the three sequences, respectively. From Figure 3.1, it can be seen that for the *Trevor* sequence, the percentage of stationary macroblocks goes to almost zero in the neighbourhood of the 60th frame, and this is due to the fact that there are some abrupt changes in the scene in the frames around the 60th frame. Figure 3.2 shows that since the *Foreman* sequence is a fast moving sequence, the percentage of stationary macroblocks is low throughout the sequence, whereas Figure 3.3 shows that since the *Miss America* sequence is a slow moving sequence, the percentage of stationary macroblocks is high throughout the sequence.

Since the stationary macroblocks do not move between the frames, the optimum motion vector for a stationary macroblock is (0,0) . Thus, the motion vectors of the stationary macroblocks can be determined beforehand without carrying out the computationally intensive search process in the block motion estimation. Consequently,

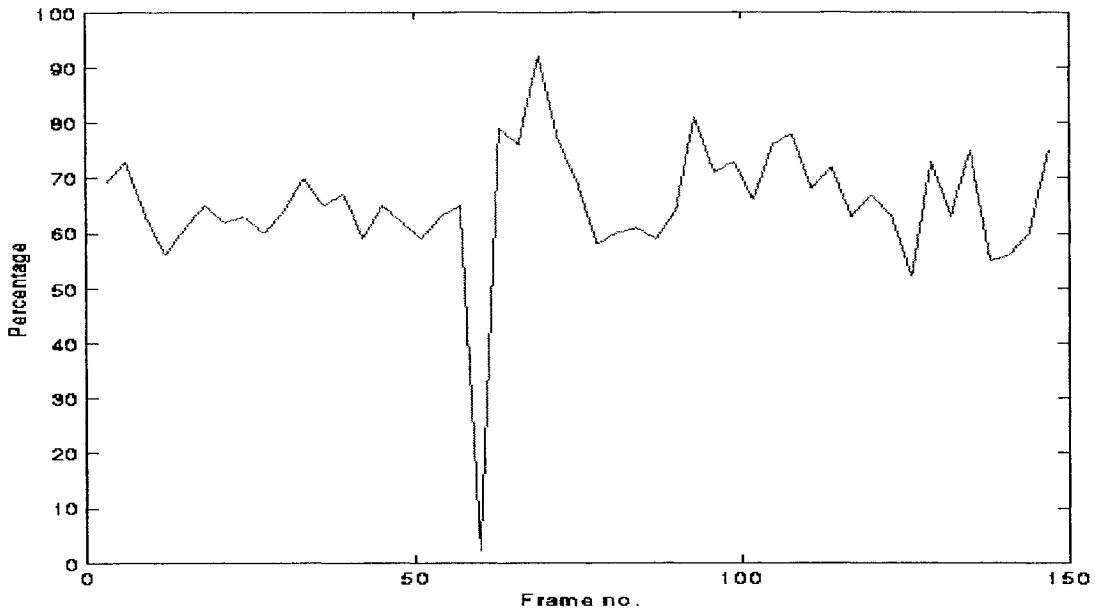


Figure 3.1 Percentage of stationary macroblocks for the *Trevor* sequence

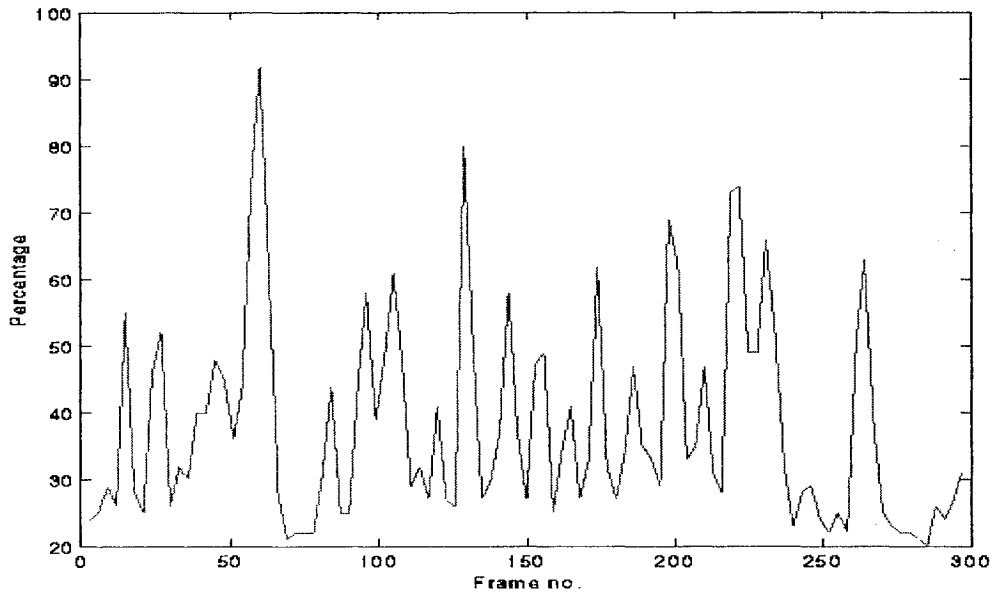


Figure 3.2 Percentage of stationary macroblocks for the *Foreman* sequence

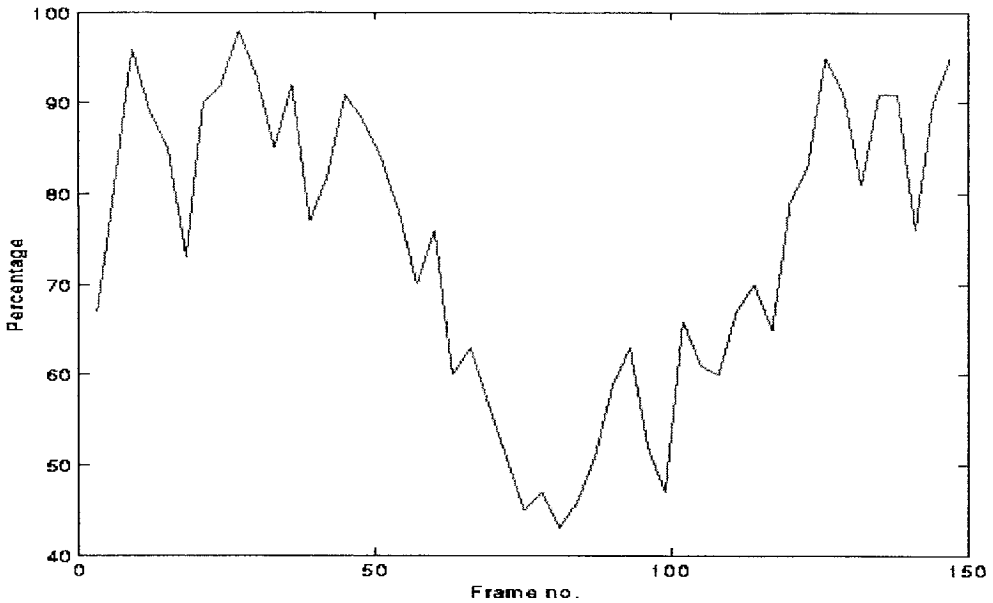


Figure 3.3 Percentage of stationary macroblocks for the *Miss America* sequence

the search process for the determination of the optimum motion vector of a stationary macroblock is unnecessary. Based on these observations, we now propose a new fast block motion estimation method by carrying out the search process only for the non-stationary macroblocks. The proposed method carries out the block motion estimation using the following two steps.

1) Check if the current macroblock is a stationary macroblock.
2) If the current macroblock is a stationary one, set the resulting motion vector for the current macroblock as (0,0) and skip the search process for this macroblock. If the current macroblock is not a stationary one, do the normal block motion estimation algorithm and find the motion vector with the minimum value of MAD.

By detecting the stationary macroblocks in the first step, the proposed method can save a great deal of computations by not carrying out the search process for such macroblocks in the block motion estimation.

Let $\vec{V}_{op}$ and $MAD_{\min}$ denote the optimum motion vector and the corresponding value of MAD, respectively. The proposed method is summarized in Algorithm 1. As previously discussed, when the current macroblock is judged as a stationary macroblock, the search process for the corresponding macroblock in the block motion estimation algorithm is skipped, and the $\vec{V}_{op}$ and $MAD_{\min}$ for such a macroblock are set as $(0,0)$ and $MAD(0,0)$, respectively.

Algorithm 1: Proposed Block Motion Estimation Method

1 Initialization
$\vec{V}_0 = (0,0)$
$MAD = MAD(\vec{V}_0)$
2 Stationary block judgement
If (current macroblock is stationary)
Go to (4);
else
Go to (3);
3 Search Process
For (each search location of $\vec{V}$ in a motion estimation algorithm)
{
Calculate $MAD(\vec{V})$
If ($MAD(\vec{V}) < MAD$)
{
$MAD = MAD(\vec{V})$
$\vec{V}_0 = \vec{V}$
}
}
4 Output
$\vec{V}_{op} = \vec{V}_0$
$MAD_{\min} = MAD$

3.2 Criterion for the Detection of Stationary Macroblocks

In order for the proposed method to be efficient, it is necessary to have a suitable criterion for the detection of the stationary macroblocks. The value of $MAD(0,0)$ is chosen as the criterion to judge whether or not a current macroblock is stationary. More formally,

(i) If $MAD(0,0) \leq T$, the current macroblock is judged as stationary, and

(ii) If $MAD(0,0) > T$, the current macroblock is judged as non-stationary,

where T stands for a predetermined threshold.

There are two advantages of the choice of $MAD(0,0)$ as the criterion.

1. $MAD(0,0)$ is the MAD corresponding to the motion vector $(0,0)$. Since $MAD(0,0)$ should also be computed in the conventional block motion estimation process, there is no computational overhead in using this value for the determination of stationary macroblocks.

2. Since a stationary macroblock does not move between the reference and current frames, $MAD(0,0)$ for a stationary macroblock has a small value, while $MAD(0,0)$ for a non-stationary macroblock has a relatively large one. Thus, the value of $MAD(0,0)$ can very well distinguish the stationary macroblocks from the non-stationary ones. In the next section, we discuss as to how to choose the value of T for different video sequences.

3.3 Adaptive Threshold Value

If a large number of stationary macroblocks remain undetected in a previous frame due to the large values of $MAD(0,0)$, the level of the threshold T should be made larger in order to detect more stationary macroblocks in the current frame. On the other hand, if there are some non-stationary macroblocks judged as stationary ones in the previous frame due to a large value of T, the level of the threshold should be made smaller. Thus, the proposed algorithm should perform better if the threshold T can be adapted to the statistical characteristics of the video sequence.

Define N_u as the number of the stationary macroblocks undetected in the previous frame, and $MAD_l(0,0)$ as the value of $MAD(0,0)$ of the l th undetected stationary macroblock in the previous frame. The method for making the threshold level T to be adaptive is now given as Algorithm 2 by using pseudo code. In Algorithm 2, the first frame refers to the first one that utilizes the block motion estimation technique. Let N_e denote the number of non-stationary macroblocks for which the values of $MAD_{\min}$ lie between T and T_1 in the previous frame and k the index of such a macroblock. In our method, the parameter values of P_0, P_1, P_2, and P_3 are set as 400, 100, 30 and 1200, respectively. These parameter values have been chosen after a simulation study with a large number of video test sequences. Initially, the threshold value is set as P_0 for the first frame. This value of P_0 should be relatively small in order not to erroneously judge a large number of non-stationary macroblocks as stationary ones in the first frame. Subsequently, this threshold value is adaptively varied from frame to frame according to the statistics of the video sequence. If there are a large number of undetected stationary

macroblocks in the previous frame, the threshold value should be made larger. Thus, T_1 is larger than T and the difference between T_1 and T depends on the number of undetected stationary macroblocks in the previous frame. On the other hand, if there are a lot of non-stationary macroblocks for which the values of $MAD_{\min}$ lie between T and T_1 in the previous frame, the threshold value should be made smaller in order not to erroneously judge a lot of non-stationary macroblocks as stationary ones in the current frame. Thus, T_2 is smaller than T_1 and the difference between them depends on the number of non-stationary macroblocks for which the values of $MAD_{\min}$ lie between T and T_1 in the previous frame. In order not to let the threshold value to be unreasonably high, P_3 is set as the upper limit of the threshold value.

Algorithm 2: Determination of the Adaptive Threshold T

If (first frame)

$T = P_0$;

else

{

$diff_1 = \sum_{l=1}^{N_u} (MAD_l(0,0) - T)$;

$T_1 = T + diff_1 / P_1$;

$diff_2 = \sum_{k=1}^{N_u} (T_1 - MAD_{\min}^k)$;

$T_2 = T_1 - diff_2 / P_2$;

$T = T_2$

If ($T > P_3$)

$T = P_3$;

}

3.4 Simulation Results

A simulation study on the proposed method for fast block motion estimation is carried out for the CCITT test video sequences in the QCIF format using the TMN20 framework of the H.263 video codec. The encoding process for a frame is carried out after skipping two frames in between, i.e., the encoding is done for the frames 1, 4, 7, and so on.

The percentage of detected stationary macroblocks out of all the macroblocks in a frame can be considered as the *speedup* offered by the proposed method for a given block motion estimation algorithm. The *error in judgment* is the percentage of the non-stationary macroblocks judged as stationary ones out of the total number of macroblocks in a frame. Figures 3.4-3.11 give the percentage of *speedup* and *error in judgment* of the proposed method in conjunction with the FSA for the *Salesman*, *Hall Objects*, *Silent*, *Akiyo*, *News*, *Foreman*, *Trevor*, and *Mother & Daughter* test video sequences, respectively. From these figures, it can be seen that the proposed method can provide a speedup for the FSA. For all the test sequences excluding the *Foreman*, this method offers a speedup of more than 50% for the FSA. Only for the very fast moving *Foreman* sequence, the percentage of *speedup*, which our method can offer for the FSA is relatively small. This is due to the fact that there is only a small number of stationary macroblocks in this sequence as shown in Figure 3.2.

In Figures 3.12-3.14, we illustrate the percentage *speedup* and *error in judgment* of the proposed method in conjunction with the 3SSA, 4SSA and UDSA, respectively, for the *Mother & Daughter* sequence. From these figures and Figure 3.11, it can be seen that

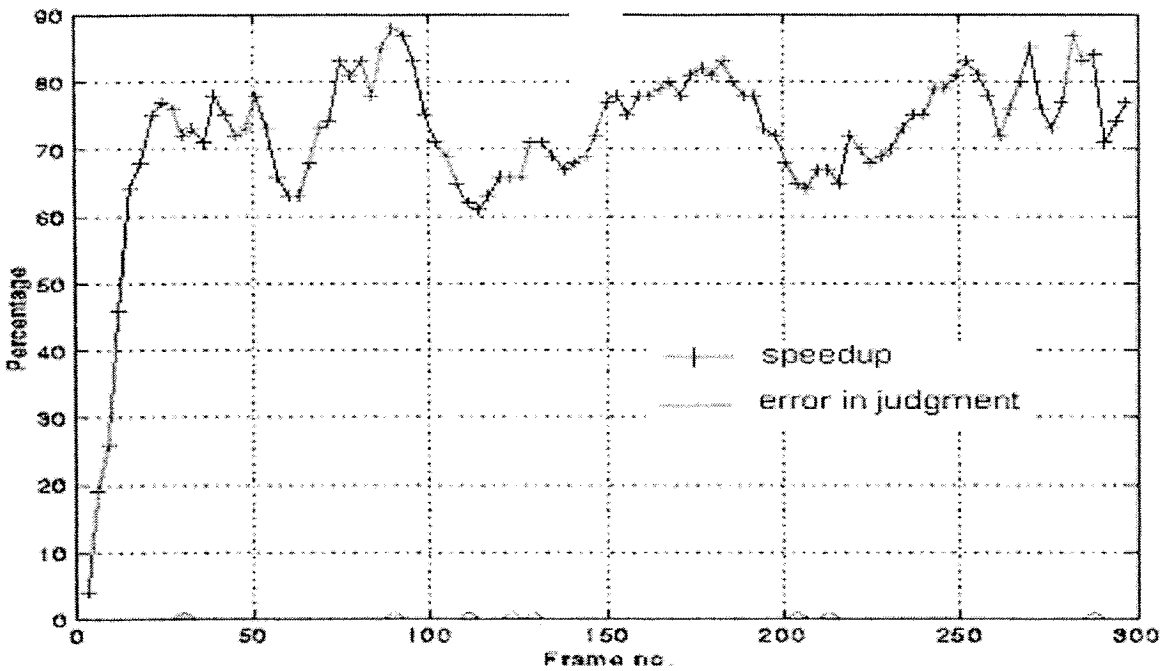


Figure 3.4 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Salesman* sequence

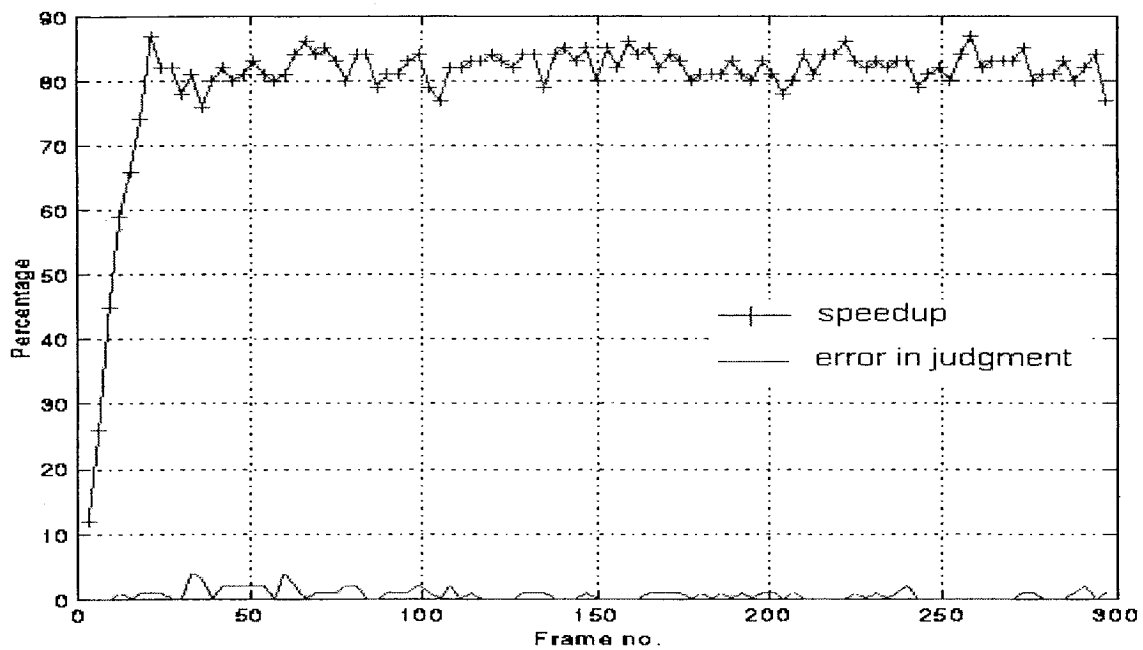


Figure 3.5 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Hall Objects* sequence

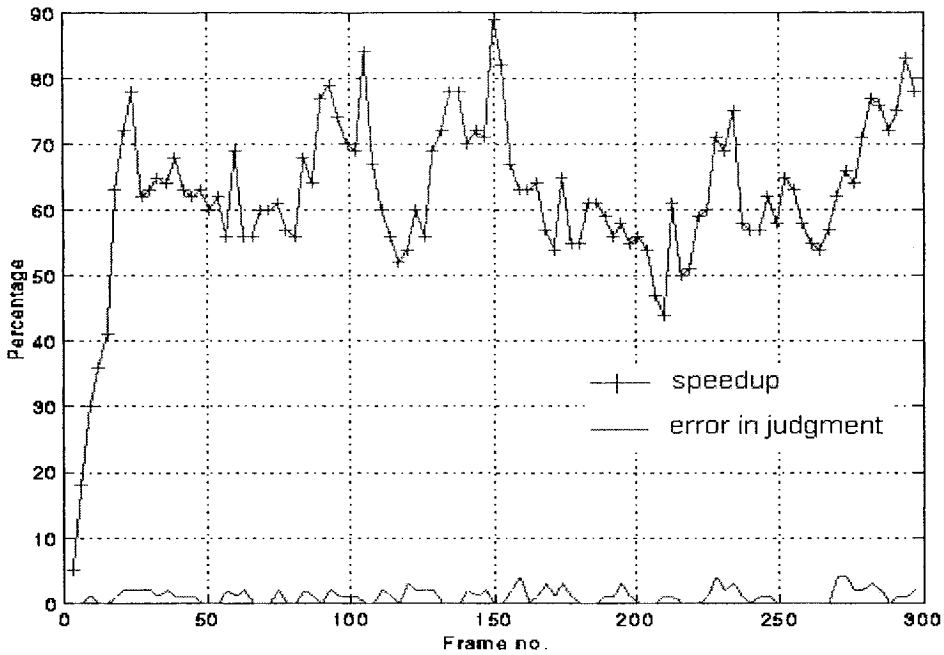


Figure 3.6 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Silent* sequence

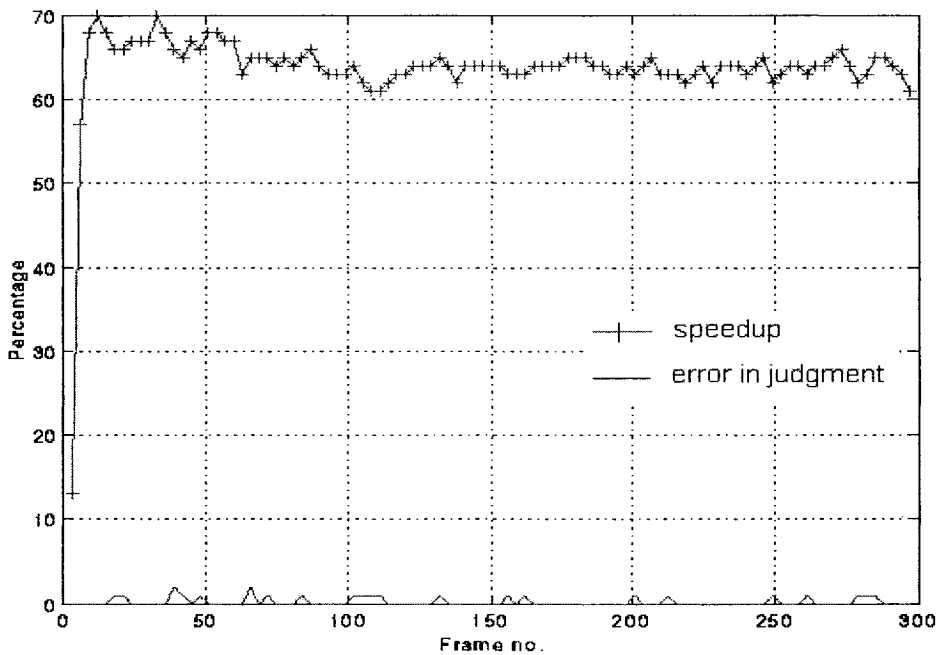


Figure 3.7 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Akiyo* sequence

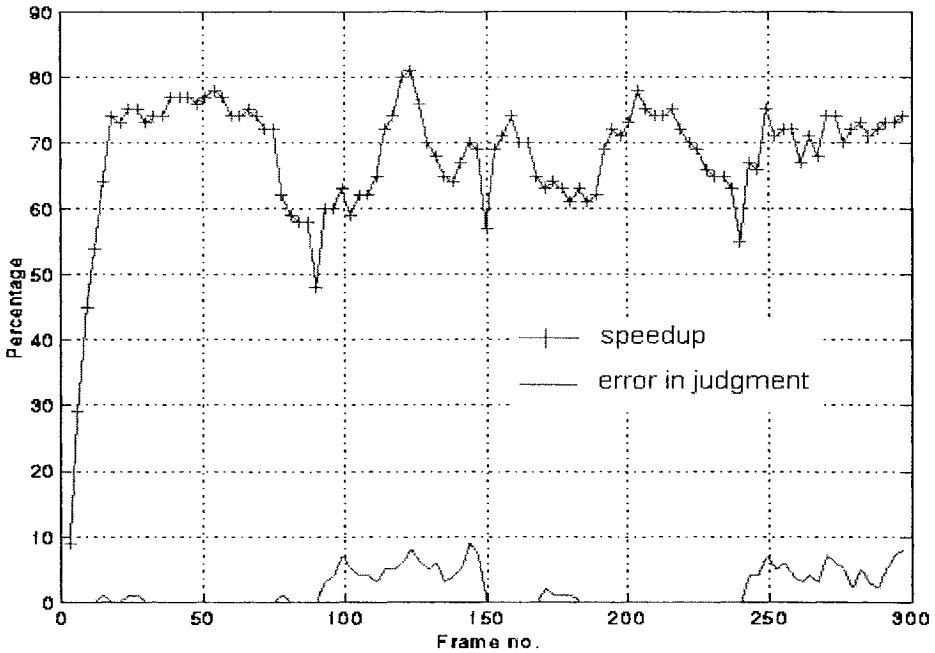


Figure 3.8 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *News* sequence

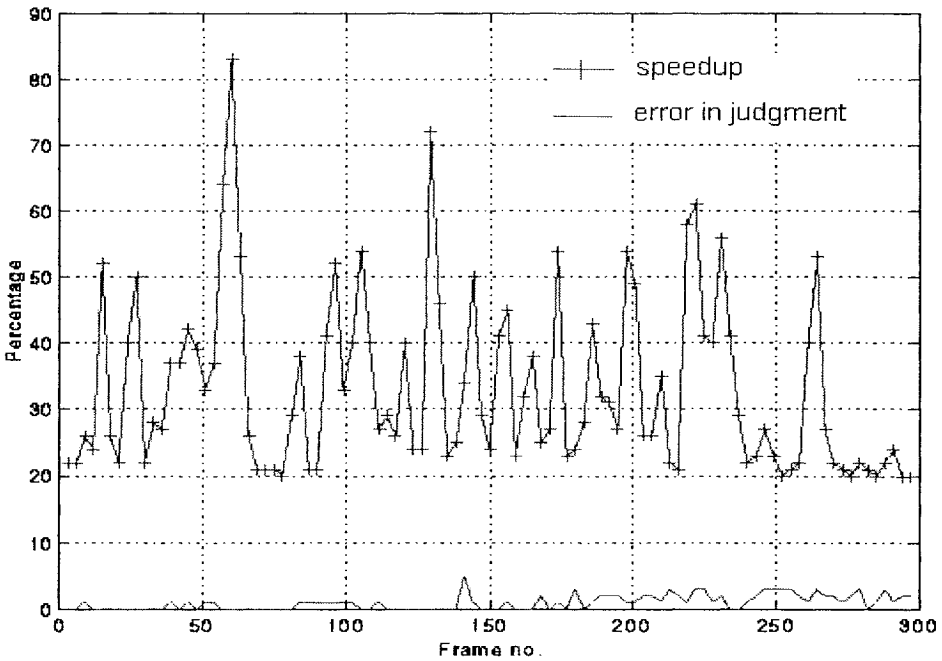


Figure 3.9 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Foreman* sequence

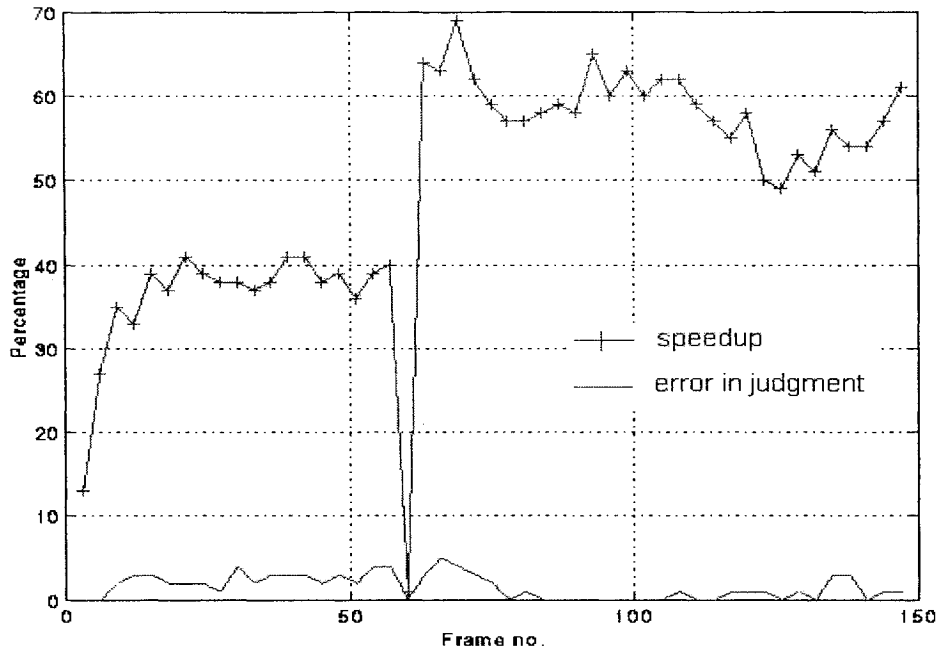


Figure 3.10 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Trevor* sequence

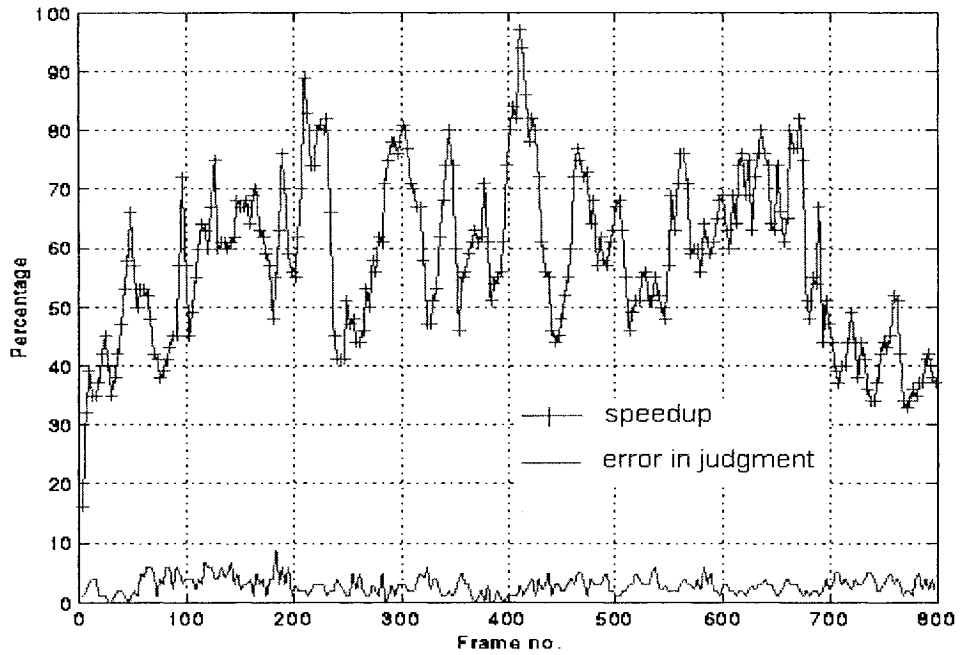


Figure 3.11 Percentage of *speedup* and *error in judgment* of proposed method combined with the FSA for the *Mother & Daughter* sequence

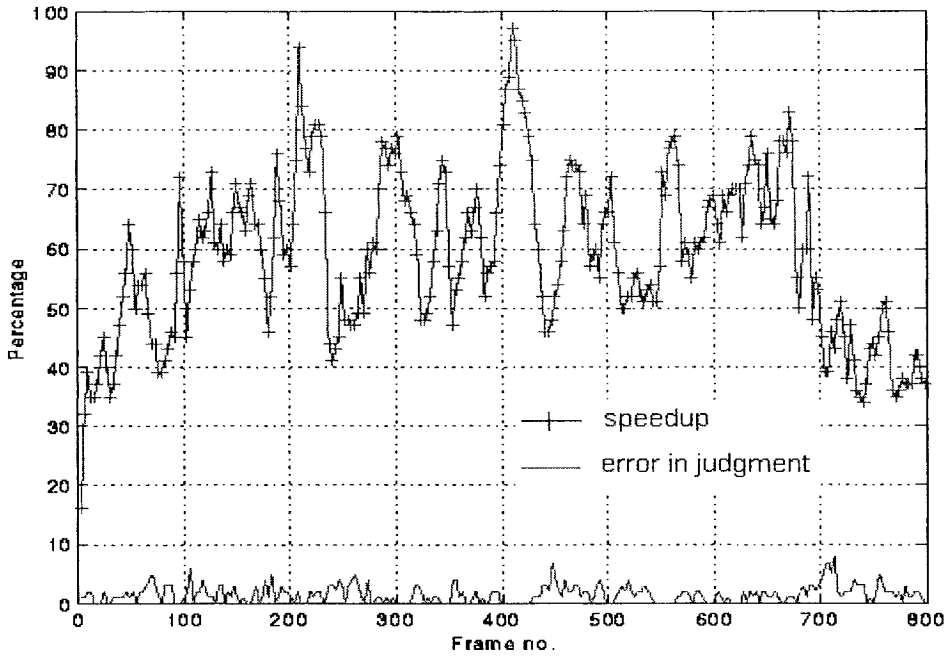


Figure 3.12 Percentage of *speedup* and *error in judgment* of proposed method combined with the 3SSA for the *Mother & Daughter* sequence

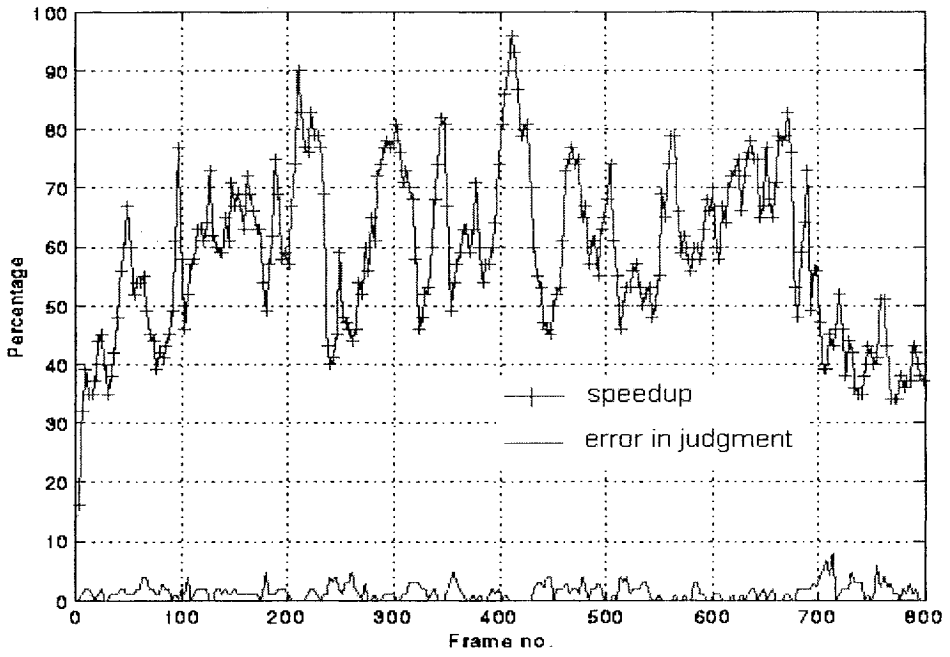


Figure 3.13 Percentage of *speedup* and *error in judgment* of proposed method combined with the 4SSA for the *Mother & Daughter* sequence

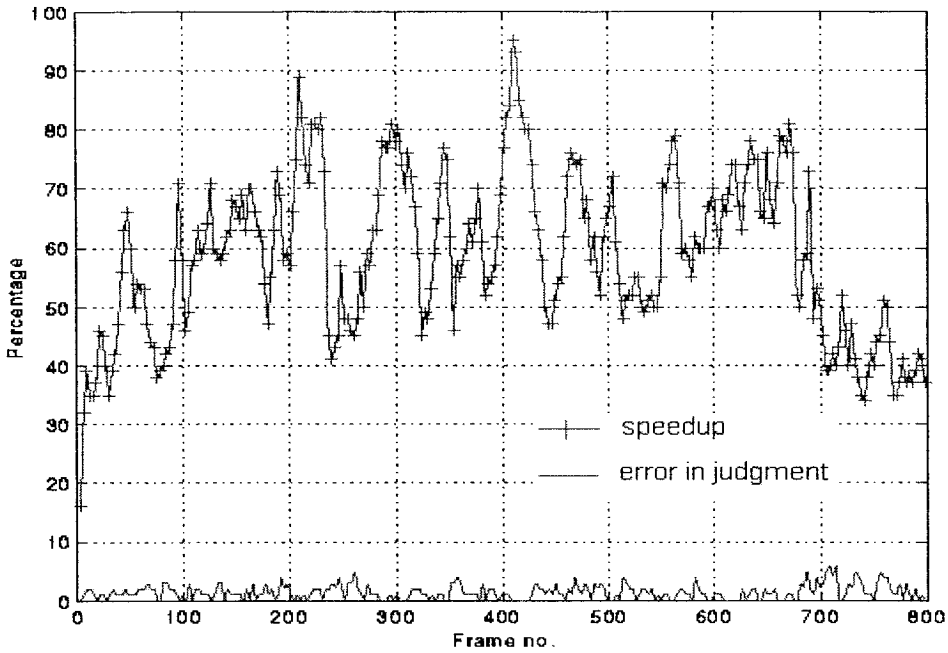


Figure 3.14 Percentage of *speedup* and *error in judgment* of proposed method combined with the UDSA for the *Mother & Daughter* sequence

the percentage *speedups* are approximately the same, irrespective of the block motion estimation algorithm chosen. Thus, the *speedup*, provided by our method, remains about the same for the various block motion estimation algorithms, showing that this *speedup* is determined by the characteristics of the video sequence.

The percentage of *error in judgment* of the proposed method in conjunction with a block motion estimation algorithm for various test sequences is generally below 5, as shown in Figures 3.4-3.14. This includes the test sequence *Trevor*, wherein there is a sudden scene-change around the 60th frame. Thus, this method has a relatively high accuracy in judging the stationary macroblocks. Even if there are some non-stationary macroblocks judged as stationary ones, their $MAD(0,0)$ values should be smaller than the threshold value of T. Thus, the block matching error of such macroblocks requires just a

few more bits for encoding. However, the requirement of these extra bits can sometimes be offset by the bit savings of the motion vectors of value $(0,0)$, since the motion vector of $(0,0)$ needs the least number of bits for its representation.

From Figures 3.4-3.14, it can also be seen that for the initial frame of the video sequences, the efficiency of proposed method is not very high, since the number of detected stationary macroblocks is initially small. This is due to the small initial value of the threshold T. According to our simulation study, the initial value of the threshold T cannot be set to a large value, since in that case a large number of non-stationary macroblocks will be judged as stationary ones. However, the advantage of the adaptive threshold value is obvious from these figures. After a few frames, the efficiency of the proposed method becomes quite high. This is in view of the fact that the value of the adaptive threshold T becomes larger based on the statistical characteristics of the video sequence.

Figures 3.15-3.22 depict the rate-distortion performance of the FSA and the proposed method combined with the FSA for the *Salesman*, *Hall Objects*, *Silent*, *Akiyo*, *News*, *Foreman*, *Trevor*, and *Mother & Daughter* test video sequences, respectively. From these figures, we can see that the rate-distortion performance of the proposed method combined with the FSA is approximately the same as that of the original FSA for the various test sequences. Thus, these figures demonstrate that the proposed method does not result in much of a loss of the coding efficiency of the FSA. Figures 3.23-3.25 illustrate similar rate-distortion performance curves for the proposed method in the case of the 3SSA, 4SSA and UDSA, respectively, for the *Mother & Daughter* sequence. From these figures, it can be seen that when the proposed method is utilized for a block motion estimation

algorithm, the coding efficiency is about the same as that of the original algorithm. Thus, the proposed method does not decrease much of the coding efficiency of a video coding system, while offering speedup for the block motion estimation process.

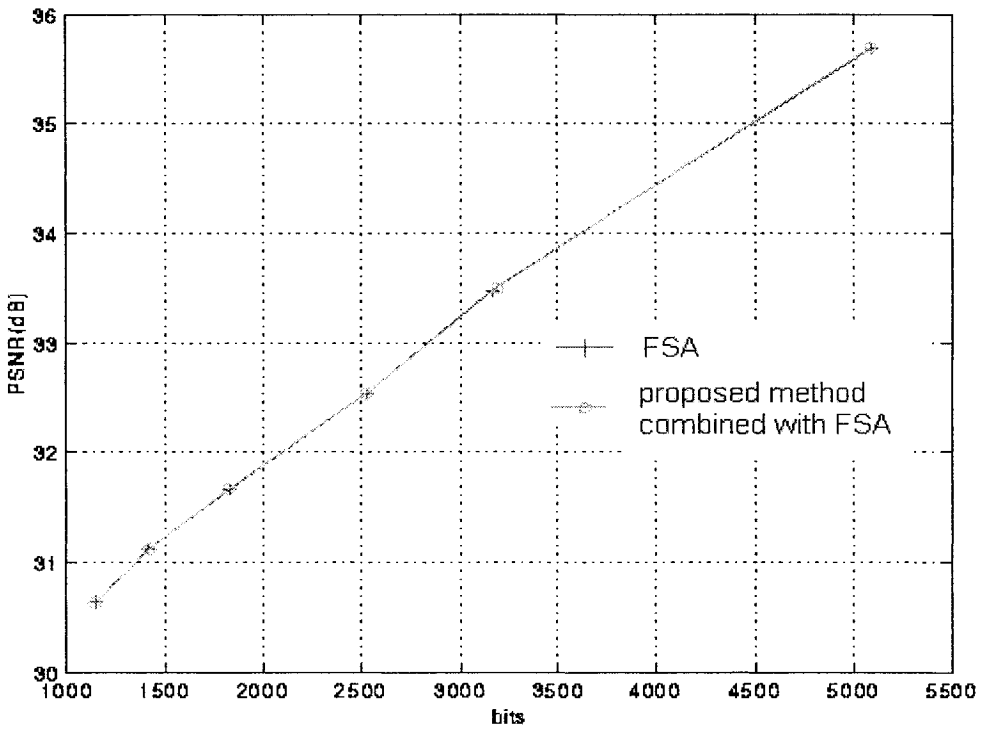


Figure 3.15 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Salesman* sequence

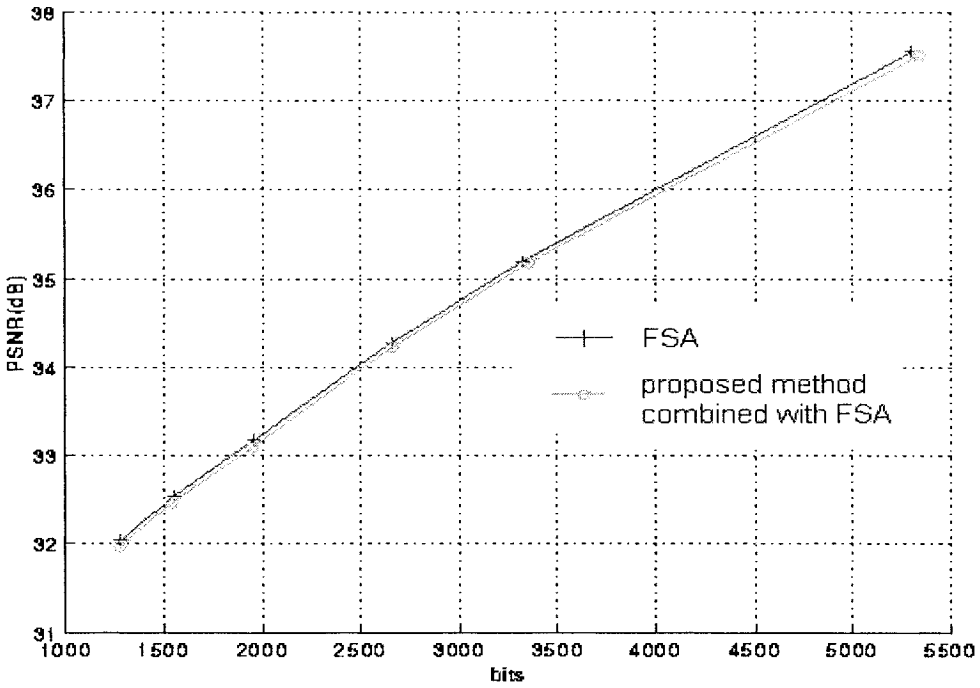


Figure 3.16 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Hall Objects* sequence

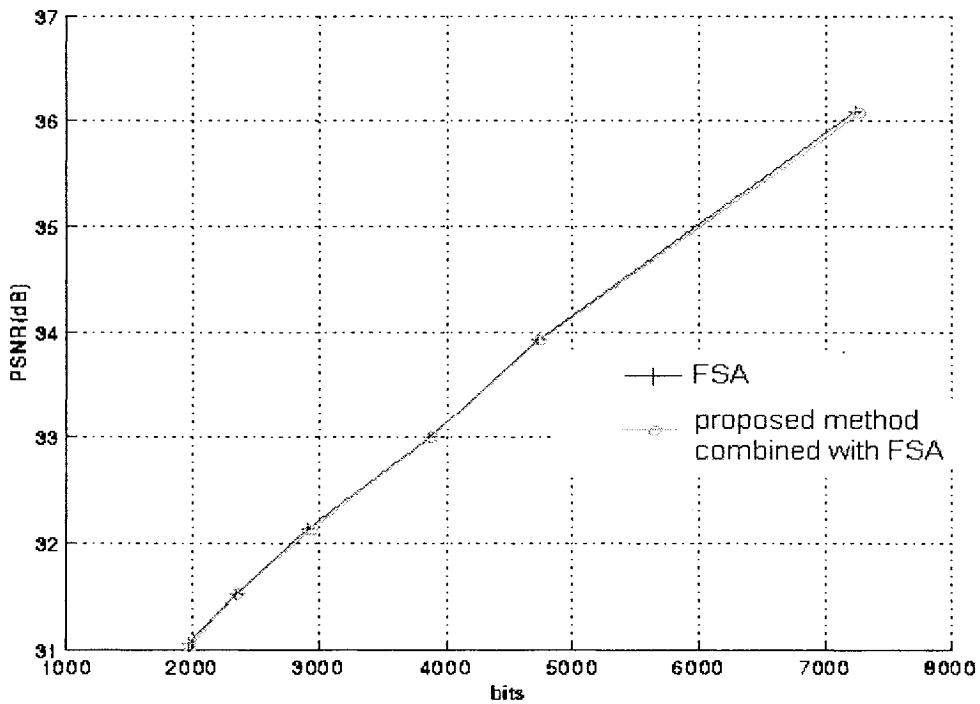


Figure 3.17 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Silent* sequence

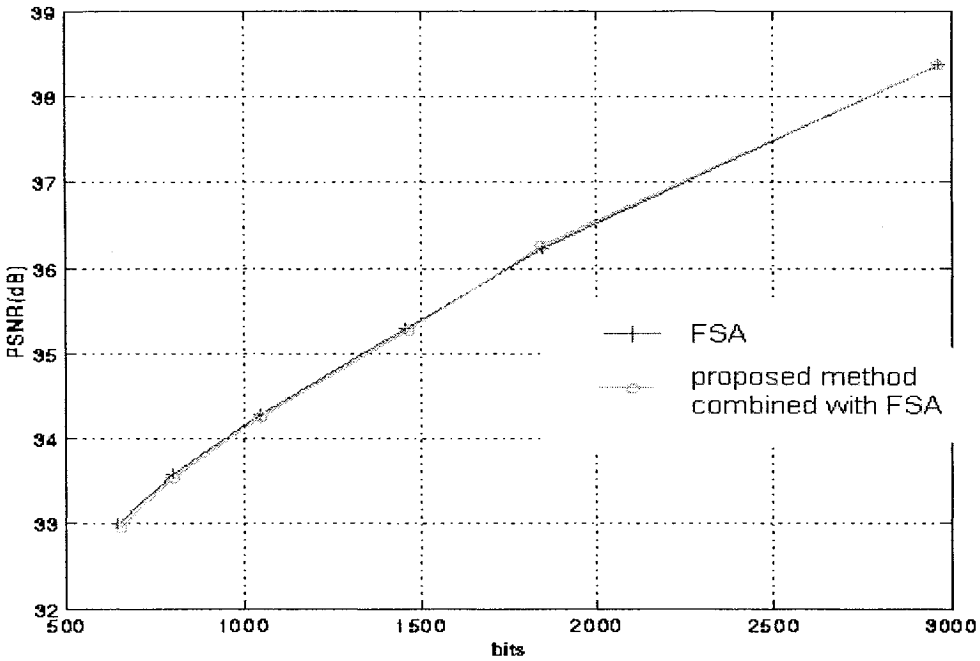


Figure 3.18 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Akiyo* sequence

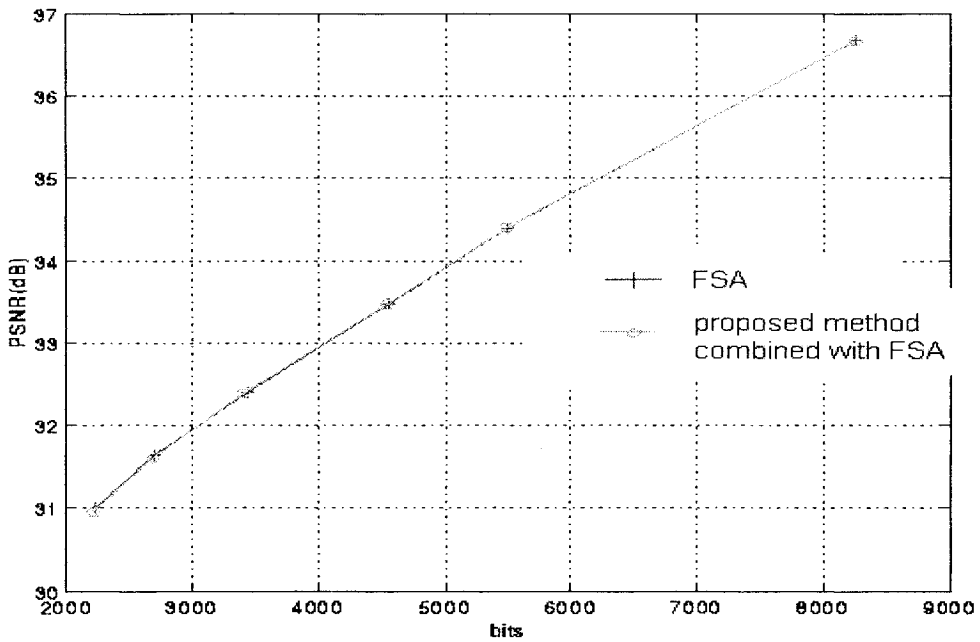


Figure 3.19 Rate-distortion performances of FSA and proposed method combined with the FSA for the *News* sequence

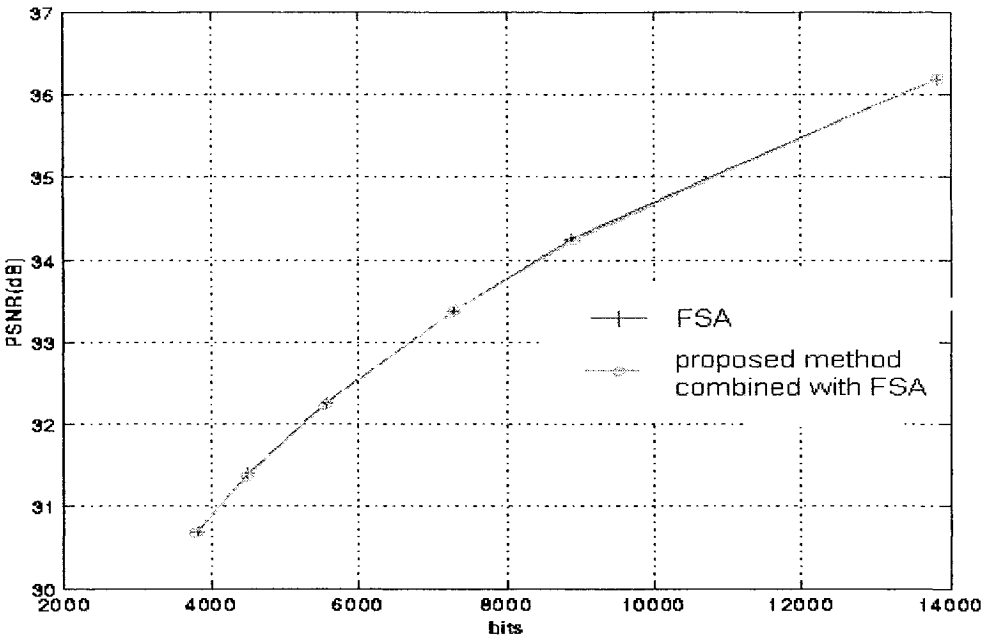


Figure 3.20 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Foreman* sequence

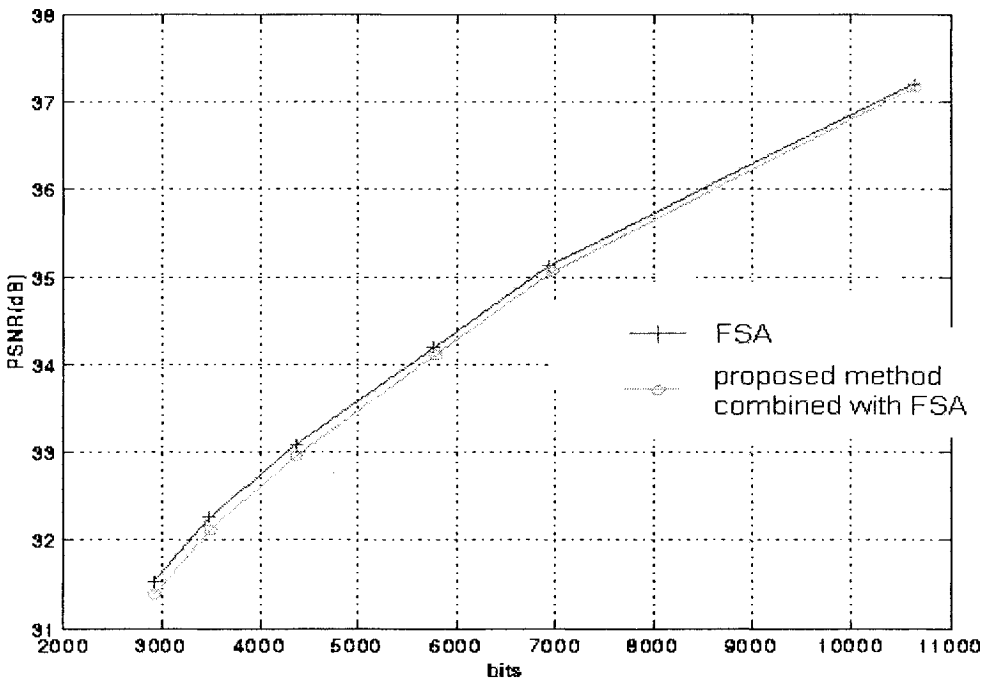


Figure 3.21 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Trevor* sequence

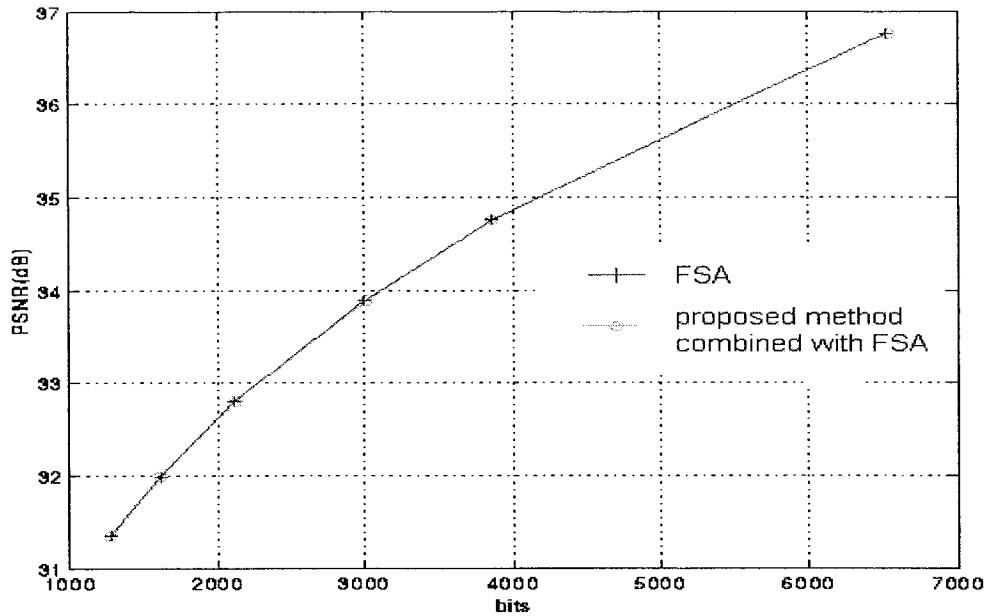


Figure 3.22 Rate-distortion performances of FSA and proposed method combined with the FSA for the *Mother & Daughter* sequence

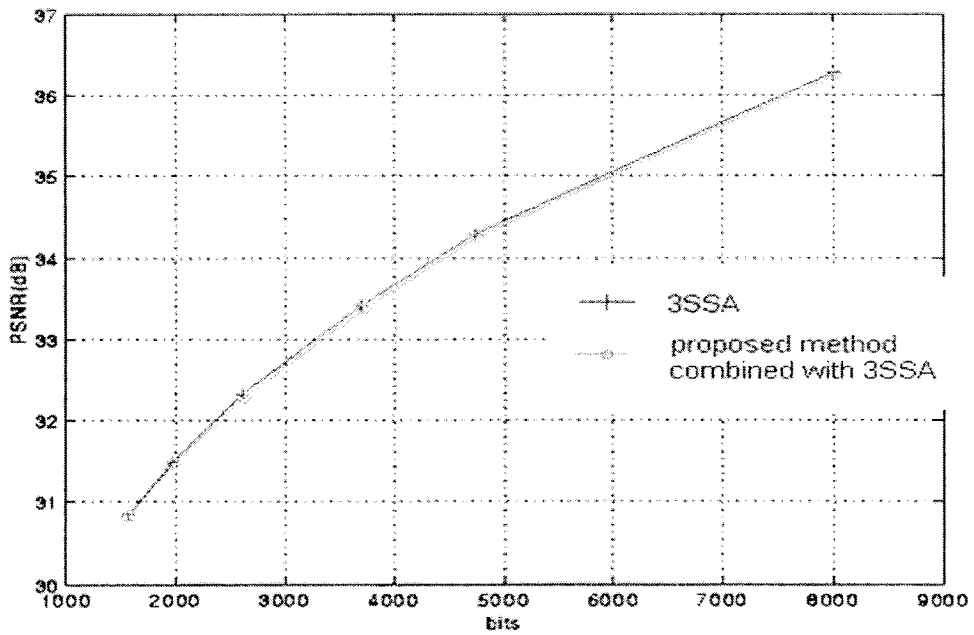


Figure 3.23 Rate-distortion performances of 3SSA and proposed method combined with the 3SSA for the *Mother & Daughter* sequence

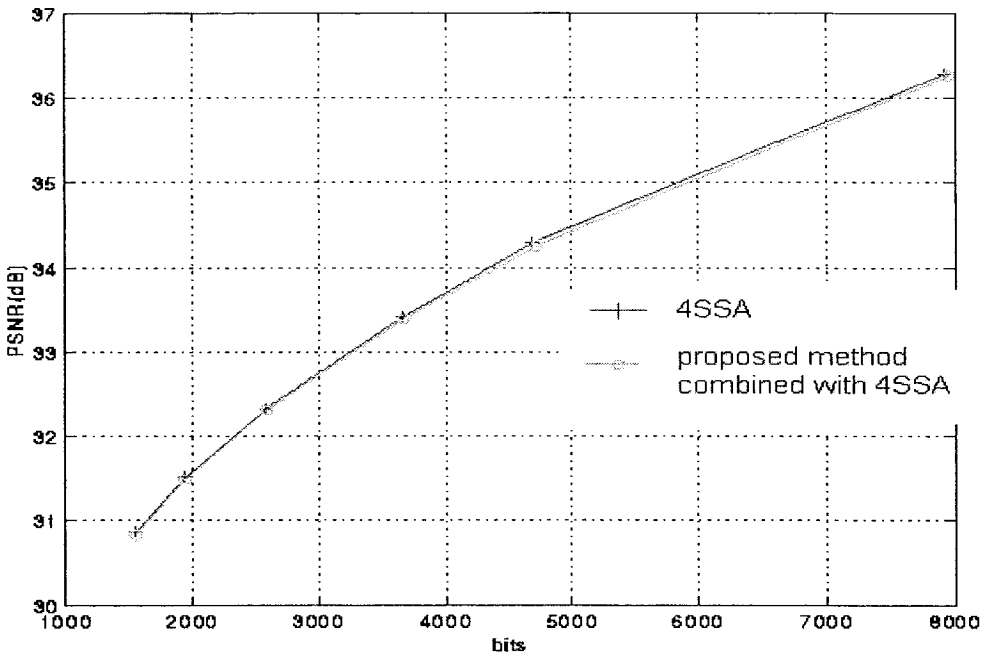


Figure 3.24 Rate-distortion performances of 4SSA and proposed method combined with the 4SSA for the *Mother & Daughter* sequence

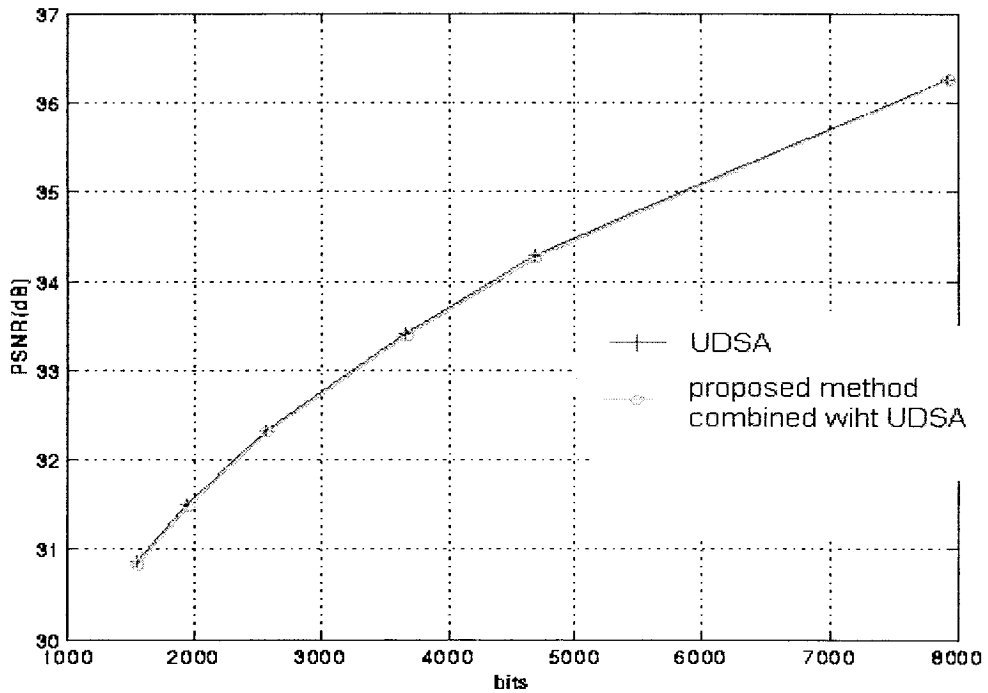


Figure 3.25 Rate-distortion performances of UDSA and proposed method combined with the UDSA for the *Mother & Daughter* sequence

3.5 Summary

Through a simulation study in this chapter, it has been established that the positions of a large number of macroblocks generally do not change from one frame to the next. Such macroblocks have been defined in this chapter as stationary macroblocks. Since the value of the motion vector for a stationary macroblock is already known to be $(0,0)$, the search process for such a stationary macroblock in a given block motion estimation algorithm is unnecessary. Based on this observation, a fast block motion estimation method has been proposed, in this chapter, by skipping the search process for the stationary macroblocks. In this method, a predetermined threshold is employed to detect the stationary macroblocks. An algorithm to adapt the threshold value based on the statistical characteristics of a given video sequence has been proposed. Simulation studies have shown that the proposed method can provide a speedup for a given block motion estimation algorithm, with about the same coding efficiency as that of the original algorithm. These studies have also shown that the amount of the *speedup* resulting from the application of the proposed scheme to a block motion estimation algorithm is approximately the same, irrespective of the algorithm chosen for the application. This result is due to the fact that the *speedup* is mainly governed by the characteristics of the video sequence rather than the chosen algorithm.

Chapter 4

A Vector-Based Fast Block Motion Estimation Algorithm

In modern CPUs, the single instruction multiple data (SIMD) technique is commonly used to provide execution speedup by employing data parallelism. In view of this, a vector-based fast block motion estimation algorithm, suitable for implementation on SIMD architectures, is proposed and described in this chapter. The proposed algorithm can accelerate the execution of the FSA for block motion estimation in two aspects. First, the computational complexity of the FSA can be reduced by employing the proposed algorithm. Second, the implementation on SIMD architectures can bring a further speedup by utilizing data parallelism.

In Section 4.1, a number of partial sums of the luminance values are defined and expression derived for the same. These partial sums are utilized to calculate some lower bounds for the MAD. A fast method to calculate these partial sums is developed. Then, an algorithm using these partial sums for reducing the computational complexity of the FSA is proposed. It is proved that this algorithm maintains the accuracy of the FSA. Section 4.2 gives the implementation of this algorithm on SIMD architecture. In Section 4.3, the computational complexity of this algorithm is discussed. Section 4.4 gives simulation results to demonstrate the effectiveness of the algorithm with respect to the computational complexity.

4.1 The Vector-Based Motion Estimation Algorithm

As in the case of the selective elimination algorithm (SEA) [52], the objective of our algorithm is to find some lower bounds for the MAD. In order to use the SIMD technique to accelerate the motion estimate process efficiently, these lower bounds must fulfill the following two requirements.

1. The computation of the lower bounds must be computationally much less intensive than the computation of the MAD itself. Otherwise, there are no savings in the computation.
2. The manipulations of these lower bounds can be easily and efficiently performed using the SIMD technique. In order to satisfy this requirement, elements involved in the computation of these lower bounds should be stored in a contiguous memory space so that the SIMD technique can be fully utilized to carry out the calculations corresponding to several data elements in parallel.

It is shown in this section that the vector-based fast block motion estimation algorithm not only satisfies the above two requirements, but also maintains the accuracy of the full-search algorithm.

4.1.1 Formation of Partial Sums

For a 16×16 block in a video coding standard, we first partition every column of this block into different data sets. Then, the addition of all the luminance values in a data set is defined as a partial sum.

Each column of a 16×16 block is partitioned into upper and lower half sections recursively until it cannot be further partitioned. Each partition can be considered as a representation of the block at level l, such that $1 \le l \le 5$. The sections in the final level, i.e., level 5, can be represented by the data sets given by

$$\Psi^5(p,q) = \{(m,n) \mid m = p \ \ and \ \ n = q\} \tag{4.1}$$

where (p,q) and (m,n) are, respectively, the indices of the data set and the pixel in a block, $0 \le m,n \le 15$, and the superscript denotes the level number. The sections of the partition at level l, $(1 \le l \le 4)$, can be represented by the sets given by

$$\Psi^l(p,q) = \Psi^{l+1}(2p,q) \bigcup \Psi^{l+1}(2p+1,q) \tag{4.2}$$

where $0 \le p < 2^{l-1}$, and $0 \le q \le 15$. Thus,

$$\Psi^l(p,q) = \{(m,n) \mid 2^{5-l} p \le m \le 2^{5-l} p + 2^{5-l} - 1 \ \ and \ \ n = q\} \tag{4.3}$$

We extend the definition of level l to include the 0th level as

$$\Psi^0 = \bigcup_{q=0}^{15} \Psi^1(0,q) \tag{4.4}$$

Thus,

$$\Psi^0 = \{(m,n) \mid 0 \le m,n \le 15\} \tag{4.5}$$

From (4.5), we see that Ψ^0 is the set of all the pixels in a 16×16 block.

The partial sums, as defined above, are the sums of all the luminance values in a data set. Thus, the partial sums at the level l $(1 \le l \le 5)$ of the current and reference block can be, respectively, expressed as

$$S_c^l(p,q) = \sum_{(m,n)\in\Psi^l(p,q)} B_c(m,n) \tag{4.6}$$

and

$$S_r^l(p,q,\vec{V}) = \sum_{(m,n)\in\Psi^l(p,q)} B_r(m,n,\vec{V}) \tag{4.7}$$

where (p,q) is the index of a partial sum in a 16×16 block, $\vec{V}=(V_x,V_y)$ is a candidate motion vector, and the subscript r and c represent the reference and current blocks, respectively. When $l=0$, the partial sums at the 0-th level are defined as

$$S_c^0 = \sum_{(m,n)\in\Psi^0} B_c(m,n) \tag{4.8}$$

and

$$S_r^0(\vec{V}) = \sum_{(m,n)\in\Psi^0} B_r(m,n,\vec{V}) \tag{4.9}$$

where Ψ^0 is given by (4.4).

Using (4.2) in (4.6), it can be seen that, when $1\le l\le 4$, the following holds.

$$S_c^l(p,q) = \sum_{(m,n)\in\Psi^{l+1}(2p,q)} B_c(m,n) + \sum_{(m,n)\in\Psi^{l+1}(2p+1,q)} B_c(m,n) \tag{4.10}$$

Using (4.6), (4.10) can be expressed as

$$S_c^l(p,q) = S_c^{l+1}(2p,q) + S_c^{l+1}(2p+1,q) \tag{4.11}$$

Similarly, using (4.2) in (4.7), when $1\le l\le 4$, the following equation can be derived.

$$S_r^l(p,q,\vec{V}) = S_r^{l+1}(2p,q,\vec{V}) + S_r^{l+1}(2p+1,q,\vec{V}) \tag{4.12}$$

Combining (4.11) and (4.12), when $1\le l\le 4$, we have

$$\begin{cases} S_c^l(p,q) = S_c^{l+1}(2p,q) + S_c^{l+1}(2p+1,q) \\ S_r^l(p,q,\vec{V}) = S_r^{l+1}(2p,q,\vec{V}) + S_r^{l+1}(2p+1,q,\vec{V}) \end{cases} \tag{4.13}$$

From (4.1), (4.6) and (4.7), when $l=5$, we have

$$\begin{cases} S_c^5(p,q) = B_c(p,q) \\ S_r^5(p,q,\vec{V}) = B_r(p,q,\vec{V}) \end{cases} \tag{4.14}$$

Using (4.4) and (4.5) in (4.8) and (4.9), when $l=0$, we have

$$\begin{cases} S_c^0 = \sum_{m=0}^{15}\sum_{n=0}^{15} B_c(m,n) = \sum_{q=0}^{15} S_c^1(0,q) \\ S_r^0(\vec{V}) = \sum_{m=0}^{15}\sum_{n=0}^{15} B_r(m,n,\vec{V}) = \sum_{q=0}^{15} S_r^1(0,q,\vec{V}) \end{cases} \tag{4.15}$$

4.1.2 Lower Bounds for the MAD

The mean of the absolute differences of the partial sums at level l $(1 \le l \le 5)$ is defined as

$$MAD^l(\vec{V}) = \sum_{q=0}^{15}\sum_{p=0}^{2^{l-1}-1} | S_c^l(p,q) - S_r^l(p,q,\vec{V}) | \tag{4.16}$$

where, without loss of generality, we have not included the division by the factor of 2^{l+3}.

The absolute difference of the partial sums at level 0 is given by

$$MAD^0(\vec{V}) = | S_c^0 - S_r^0(\vec{V}) | \tag{4.17}$$

Using (4.13) in (4.16), for $1 \le l \le 4$, we have

$$MAD^l(\vec{V}) = \sum_{q=0}^{15}\sum_{p=0}^{2^{l-1}-1} | S_c^{l+1}(2p,q) + S_c^{l+1}(2p+1,q) - S_r^{l+1}(2p,q,\vec{V}) - S_r^{l+1}(2p+1,q,\vec{V}) | \tag{4.18}$$

The above equation can be expressed as

$$MAD^l(\vec{V}) = \sum_{q=0}^{15}\sum_{p=0}^{2^{l-1}-1} | (S_c^{l+1}(2p,q) - S_r^{l+1}(2p,q,\vec{V})) + (S_c^{l+1}(2p+1,q) - S_r^{l+1}(2p+1,q,\vec{V})) | \tag{4.19}$$

Using the inequality

$$| a+b | \le | a | + | b | \tag{4.20}$$

$MAD^l(\vec{V})$ can be expressed as

$$MAD^{l}(\vec{V}) \le \sum_{q=0}^{15} \sum_{p=0}^{2^{l-1}-1} (|\, S_c^{l+1}(2p,q) - S_r^{l+1}(2p,q,\vec{V})\,| + |\, S_c^{l+1}(2p+1,q) - S_r^{l+1}(2p+1,q,\vec{V})\,|) \tag{4.21}$$

The above inequality can be rearranged as

$$MAD^{l}(\vec{V}) \le \sum_{q=0}^{15} \sum_{p=0}^{2^{(l+1)-1}-1} |\, S_c^{l+1}(p,q) - S_r^{l+1}(p,q,\vec{V})\,| \tag{4.22}$$

From (4.16), the right side of (4.22) is just $MAD^{l+1}(\vec{V})$. Thus, for $1 \le l \le 4$, the following inequality holds.

$$MAD^{l}(\vec{V}) \le MAD^{l+1}(\vec{V}) \tag{4.23}$$

When $l = 5$, $MAD^{5}(\vec{V})$ can be expressed using (4.14) in (4.16) as

$$MAD^{5}(\vec{V}) = \sum_{q=0}^{15} \sum_{p=0}^{15} |\, B_c(p,q) - B_r(p,q,\vec{V})\,| \tag{4.24}$$

The right side of (4.24) is just $MAD(\vec{V})$. Thus, for $l = 5$, we have

$$MAD^{5}(\vec{V}) = MAD(\vec{V}) \tag{4.25}$$

From (4.15), when $l = 0$, (4.17) can be expressed as

$$MAD^{0}(\vec{V}) = |\sum_{m=0}^{15} \sum_{n=0}^{15} B_c(m,n) - \sum_{m=0}^{15} \sum_{n=0}^{15} B_r(m,n,\vec{V})\,| \tag{4.26}$$

The right side of (4.26) is the lower bound for the $MAD(\vec{V})$ in the selective elimination algorithm, i.e.,

$$LB_{SEA}(\vec{V}) = MAD^{0}(\vec{V}) \tag{4.27}$$

where $LB_{SEA}(\vec{V})$ is as defined in (A.8). Using (4.15), (4.17) can be rewritten as

$$MAD^{0}(\vec{V}) = |\sum_{q=0}^{15} (S_c^{1}(0,q) - S_r^{1}(0,q,\vec{V})\,| \tag{4.28}$$

Using the inequality

$$|\sum_{q=0}^{15} c(q)| \leq \sum_{q=0}^{15} |c(q)| \tag{4.29}$$

$MAD^0(\vec{V})$ can be expressed as

$$MAD^0(\vec{V}) \leq \sum_{q=0}^{15} |S_c^1(0,q) - S_r^1(0,q,\vec{V})| \tag{4.30}$$

The right side of (4.30) is $MAD^1(\vec{V})$ from (4.16). Thus, we have

$$MAD^0(\vec{V}) \leq MAD^1(\vec{V}) \tag{4.31}$$

Combining (4.23), (4.25), (4.27), and (4.31), we have

$$LB_{SEA}(\vec{V}) \leq MAD^1(\vec{V}) \leq MAD^2(\vec{V}) \leq MAD^3(\vec{V}) \leq MAD^4(\vec{V}) \leq MAD(\vec{V}) \tag{4.32}$$

From the inequalities given by (4.32), we see that there are four lower bounds for the $MAD(\vec{V})$ and they are $MAD^l(\vec{V})$, $1 \leq l \leq 4$. These lower bounds are tighter than the lower bound in the selective elimination algorithm. Due to this fact, the number of times the calculation of $MAD(\vec{V})$ that can be circumvented in the vector-based fast block motion estimation algorithm is much more than that in the selective elimination algorithm.

In passing, it is noted that the term "multi-level partial sums" was introduced in [21, 22]. However, the partial sums used therein do not satisfy the Requirement 2 given on page 49.

4.1.3 Fast Method to Compute the Partial Sums

From the above discussion, it is obvious that in order to calculate $MAD^l(\vec{V})$, one needs to compute $S_c^l(p,q)$ and $S_r^l(p,q,\vec{V})$, which are the partial sums of the luminance values. A fast method to calculate these partial sums is now given.

The frame partial sums at level l $(1 \le l \le 5)$ at the location (m,n) in the current and the reference frames, respectively, are defined as

$$\begin{cases} F_c^l(m,n) = \sum_{i=0}^{2^{5-l}-1} I_c(m+i,n) \\ F_r^l(m,n) = \sum_{i=0}^{2^{5-l}-1} I_r(m+i,n) \end{cases} \tag{4.33}$$

where $I_c(x,y)$ and $I_r(x,y)$ are the luminance values at the location (x,y) in the current and reference frames, respectively. The frame partial sums at 0 th level at the location (m,n) in the current and reference frames, respectively, are defined as

$$\begin{cases} F_c^0(m,n) = \sum_{i=0}^{15}\sum_{j=0}^{15} I_c(m+i,n+j) \\ F_r^0(m,n) = \sum_{i=0}^{15}\sum_{j=0}^{15} I_r(m+i,n+j) \end{cases} \tag{4.34}$$

The frame partial sums are calculated only once and saved in the memory for future use. The corresponding block partial sums at the l th $(1 \le l \le 5)$ level can be expressed as

$$\begin{cases} S_c^l(p,q) = F_c^l(x_0 + p \bullet 2^{5-l}, y_0 + q) \\ S_r^l(p,q,\vec{V}) = F_r^l(x_0 + V_x + p \bullet 2^{5-l}, y_0 + V_y + q) \end{cases} \tag{4.35}$$

where (x_0, y_0) is the upper-left corner of the block, $\vec{V} = (V_x, V_y)$ is the candidate motion vector, and (p,q) is the index of the partial sum in a block. The corresponding block partial sums at the 0 th level can be expressed as

$$\begin{cases} S_c^0 = F_c^0(x_0, y_0) \\ S_r^0(\vec{V}) = F_r^0(x_0 + V_x, y_0 + V_y) \end{cases} \tag{4.36}$$

From (4.33), we see that, when $l = 5$, we have

$$\begin{cases} F_c^5(m,n) = I_c(m,n) \\ F_r^5(m,n) = I_r(m,n) \end{cases} \tag{4.37}$$

and when $1 \le l \le 4$, we have

$$F_c^l(m,n) = \sum_{i=0}^{2^{5-l}-1} I_c(m+i,n) = \sum_{i=0}^{2^{5-l-1}-1} I_c(m+i,n) + \sum_{i=0}^{2^{5-l-1}-1} I_c(m+2^{4-l}+i,n) \qquad (4.38)$$

The right side of (4.38) is $F_c^{l+1}(m,n) + F_c^{l+1}(m+2^{4-l},n)$, using (4.33). Thus, for $1 \le l \le 4$, we have

$$F_c^l(m,n) = F_c^{l+1}(m,n) + F_c^{l+1}(m+2^{4-l},n) \qquad (4.39)$$

For a faster computation of $F_c^l(m,n)$, we could use (4.39) instead of (4.33). We initially obtain $F_c^5(m,n)$ using (4.37). Then, we employ (4.39) to compute $F_c^l(m,n)$ from $F_c^{l+1}(m,n)$ recursively, for $1 \le l \le 4$.

From (4.33) and (4.34), we have

$$F_c^0(m,n) = \sum_{j=0}^{15}[\sum_{i=0}^{15} I_c(m+i,n+j)] = \sum_{j=0}^{15} F_c^1(m,n+j) \qquad (4.40)$$

The use of (4.40) is not a good method to compute $F_c^0(m,n)$, since it needs 15 additions to compute $F_c^0(m,n)$ per pixel. Now, we give a fast method to compute $F_c^0(m,n)$ that needs only two additions per pixel. Equation (4.40) may be rewritten as

$$F_c^0(m,n) = \sum_{j=0}^{15} F_c^1(m,n-1+j) + F_c^1(m,n+15) - F_c^1(m,n-1) \qquad (4.41)$$

The first term on the right side of (4.41) is $F_c^0(m,n-1)$ from (4.40). Hence,

$$F_c^0(m,n) = F_c^0(m,n-1) + F_c^1(m,n+15) - F_c^1(m,n-1) \qquad (4.42)$$

From (4.42), we see that only one addition and one subtraction are needed to compute $F_c^0(m,n)$ for $n \ge 1$. Thus, we can use (4.42) to compute $F_c^0(m,n)$ (for $n \ge 1$) efficiently. In order to use (4.42) to calculate $F_c^0(m,n)$, we have to initially calculate $F_c^0(m,0)$ using

$$F_c^0(m,0) = \sum_{j=0}^{15} F_c^1(m,j) \tag{4.43}$$

Once $F_c^0(m,0)$ is found, we can use (4.42) to calculate $F_c^0(m,n)$ from $F_c^0(m,n-1)$ recursively. Table 4.1 summarizes the method outlined above for a fast calculation of the frame partial sum.

Table 4.1 Fast method to calculate the frame partial sums

1) If (not the first encoding frame)
$F_r^l(m,n) = F_c^l(m,n)$

2) For ($m = 0$; $m < H$; $m = m+1$)
{
For ($n = 0$; $n < W$; $n = n+1$)
$F_c^5(m,n) = I_c(m,n)$
}
For ($l = 4$; $l \geq 1$; $l = l-1$)
{
For ($m = 0$; $m < H - 2^{5-l} + 1$; $m = m+1$)
{
For ($n = 0$; $n < W$; $n = n+1$)
{
$F_c^l(m,n) = F_c^{l+1}(m,n) + F_c^{l+1}(m + 2^{4-l}, n)$
}
}
}

3) For ($m = 0$; $m < H - 15$; $m = m+1$)
{
$F_c^0(m,0) = \sum_{j=0}^{15} F_c^1(m,j)$
}
For ($n = 0$; $n < W - 15$; $n = n+1$)
{
For ($m = 1$; $m < H - 15$; $m = m+1$)
{
$F_c^0(m,n) = F_c^0(m,n-1) + F_c^1(m,n+15) - F_c^1(m,n-1)$
}
}

Note: W and H represent the width and height of a video frame.

4.1.4 Algorithm

The inequalities given by (4.32) form the basis of the vector-based motion estimation algorithm. In this section, we discuss the use of the inequalities given by (4.32) for the reduction of the computational complexity of the FSA.

Let $CS(n)$ be the set of all the candidate motion vectors checked so far in a motion estimation process and $\vec{V}_{op}(n)$ the optimum motion vector in the set $CS(n)$, i.e.,

$$\vec{V}_{op}(n) = \arg\{\min_{\vec{V}\in CS(n)}[MAD(\vec{V})]\} \tag{4.44}$$

where n is the number of candidate motion vectors checked. Denoting the current candidate motion vector to be checked in a motion estimation process by $\vec{V}_{curr}$, we have

$$CS(n+1) = CS(n) \bigcup \{\vec{V}_{curr}\} \tag{4.45}$$

and

$$\vec{V}_{op}(n+1) = \arg\{\min_{\vec{V}\in CS(n+1)}[MAD(\vec{V})]\} \tag{4.46}$$

Since we have $CS(n) \subset CS(n+1)$, the following inequality can be derived

$$MAD(\vec{V}_{op}(n+1)) \leq MAD(\vec{V}_{op}(n)) \tag{4.47}$$

If at any level of l, $0 \leq l \leq 4$,

$$MAD^{l}(\vec{V}_{curr}) > MAD(\vec{V}_{op}(n)) \tag{4.48}$$

then, we have from (4.32)

$$MAD(\vec{V}_{curr}) \geq MAD^{l}(\vec{V}_{curr}) > MAD(\vec{V}_{op}(n)) \tag{4.49}$$

Comparing (4.47) and (4.49), the current candidate motion vector $\vec{V}_{curr}$ is not a better motion vector than $\vec{V}_{op}(n)$ whenever (4.48) holds, that is,

$$\vec{V}_{curr} \neq \vec{V}_{op}(n+1) \quad \text{whenever } MAD^{l}(\vec{V}_{curr}) > MAD(\vec{V}_{op}(n)) \tag{4.50}$$

Thus, if (4.48) is satisfied for any level l, $0 \le l \le 4$, the computation of $MAD(\vec{V}_{curr})$ can be avoided without the exclusion of the optimum motion vector. In this way, a large number of computations of $MAD(\vec{V})$, which are very computationally intensive, can be circumvented while keeping the accuracy of the full-search algorithm. In order to utilize (4.32) more efficiently, the computations of $MAD^l(\vec{V})$ are carried out successively from level 0 to level 5, and stopped whenever (4.48) is satisfied. The vector-based fast block motion estimation algorithm is summarized in Table 4.2. In this table, $\vec{V}_{op}$ and $MAD_{\min}$ are, respectively, the optimum motion vector checked so far and its corresponding value of MAD.

4.2 SIMD Implementation of the Proposed Algorithm

In this section, we give an implementation of the algorithm using an SIMD technique to further accelerate the process of motion estimation.

Usually, the luminance values of a video frame are saved in the memory in a row-by-row raster scan manner. According to (4.39), the frame partial sums at level l can be obtained by the summation of the two frame partial sums at level l+1 in the same column. In order to use the data parallelism offered by the SIMD technique, the frame partial sums at every level are also saved in a row-by- row raster scan manner. The process of using the SIMD technique in these calculations is given in Table 4.3.

Table 4.2 A vector-based fast block motion estimation algorithm

Step 1) **Initialization**

a) Compute all the partial sums for the current frame and save them in a continuous memory space.

b) Retrieve all the partial sums for the reference frame in a saved continuous memory space.

Step 2. For every current block, execute the block motion estimation process.

Step 2.1 Initialization

$\vec{V}_{op} = (0,0)$

$MAD_{\min} = MAD(\vec{V}_{op})$

Step 2.2 Search

(*) For (each search location of V_{curr} in the full-search algorithm)

{

If ($MAD^0(\vec{V}_{curr}) > MAD_{\min}$)

Go to (*) and select next search location;

If ($MAD^1(\vec{V}_{curr}) > MAD_{\min}$)

Go to (*) and select next search location;

If ($MAD^2(\vec{V}_{curr}) > MAD_{\min}$)

Go to (*) and select next search location;

If ($MAD^3(\vec{V}_{curr}) > MAD_{\min}$)

Go to (*) and select next search location;

If ($MAD^4(\vec{V}_{curr}) > MAD_{\min}$)

Go to (*) and select next search location;

Calculate the $MAD(\vec{V}_{curr})$;

If ($MAD(\vec{V}_{curr}) < MAD_{\min}$)

{

$MAD_{\min} = MAD(\vec{V}_{curr})$

$\vec{V}_{op} = \vec{V}_{curr}$

}

}

Table 4.3 Employment of SIMD technique for the calculations of frame partial sums

1) Load four $(l+1)$-th level frame partial sums into the first SIMD register.
2) Load the four $(l+1)$-th level frame partial sums at the next 2^{4-l}-th row and the same column as in (1) into the second SIMD register.
3) If $l=4$, unpack the loaded data from the byte type to the word type.
4) Do the four-word summations of the data of the first two SIMD registers in only one SIMD instruction.
5) Store the data and go to (1) for the next four partial sums.

Since the partial sums of a macroblock are also saved in a row-by-row raster scan manner, the proposed algorithm is very suitable for an SIMD implementation. Equation (4.16) can be rewritten as

$$MAD^l(\vec{V}) = \sum_{p=0}^{2^{l-1}-1} r(p) \tag{4.51}$$

where

$$r(p) = \sum_{q=0}^{15} | S_c^l(p,q) - S_r^l(p,q,\vec{V}) | \tag{4.52}$$

As seen from (4.51), the elements involved in the computation of $MAD^l(V_x,V_y)$ consist of only $r(p)$'s, which form a vector. This is the reason why the proposed algorithm is called the vector-based fast block motion estimation algorithm. The significant feature of the proposed algorithm is that $S_c^l(p,q)$ and $S_r^l(p,q,\vec{V})$ in (4.52) are, respectively, saved

in a contiguous memory space. In this way, four subtractions in (4.52) can be carried out simultaneously in an SIMD instruction utilizing two 64-bit SIMD registers, as shown in Figure 4.1.

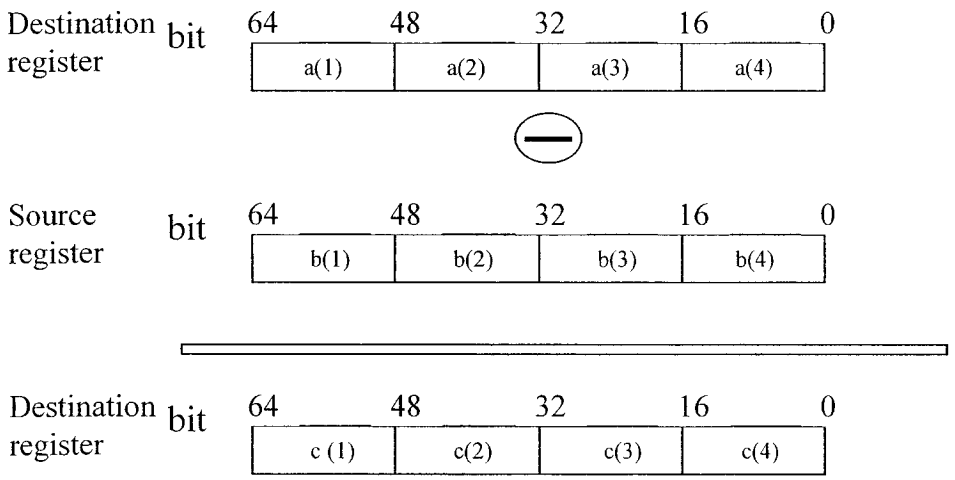


Figure 4.1 Packed word-type subtraction on SIMD registers

4.3 Computational Complexity of the Proposed Algorithm

In Section 4.1, a vector-based fast block motion estimation algorithm has been proposed. This algorithm maintains the accuracy of the full-search algorithm as mentioned in Section 4.1.4. In this section, we first analyze the computational complexity to compute the l th level lower bounds of $MAD^l(\vec{V})$ for $0 \le l \le 5$, and then discuss the computational complexity to compute the frame partial sums. Based on these, theoretical

and practical speedups of the proposed vector-based fast block motion estimation algorithm over the full-search and selective elimination algorithms are determined.

4.3.1 Computational Complexity of $MAD^l(\vec{V})$ for a Given l, $0 \le l \le 5$

Equations (4.16) and (4.17) are used to calculate the value of $MAD^l(\vec{V})$. When $l = 0$,

$$MAD^0(\vec{V}) = | S_c^0 - S_r^0(\vec{V}) | \tag{4.53}$$

and when $1 \le l \le 5$,

$$MAD^l(\vec{V}) = \sum_{p=0}^{15} \sum_{q=0}^{2^{l-1}-1} | S_c^l(p,q) - S_r^l(p,q,\vec{V}) | \tag{4.54}$$

From (4.53), we see that the computation of $MAD^0(\vec{V})$ needs only one subtraction and one absolute value calculation (the operation to calculate the absolute value of x). From (4.54), we see that the computation of $MAD^l(\vec{V})$ (for $1 \le l \le 5$) needs $16 \times 2^{l-1}$ subtractions, $16 \times 2^{l-1}$ absolute value calculations, and $16 \times 2^{l-1} - 1$ additions. Let $n_a(l)$, $n_s(l)$, and $n_{ab}(l)$ denote, respectively, the number of additions, subtractions, and absolute value calculations in the computation of $MAD^l(\vec{V})$. Then, from the above discussion, we have

$$\begin{cases} n_a(0) = 0 \\ n_s(0) = 1 \\ n_{ab}(0) = 1 \end{cases} \tag{4.55}$$

and

$$\begin{cases} n_a(l) = 16 \times 2^{l-1} - 1 \\ n_s(l) = 16 \times 2^{l-1} \\ n_{ab}(l) = 16 \times 2^{l-1} \end{cases} \qquad 0 \le l \le 5 \tag{4.56}$$

As discussed in Chapter 2, in order to find the computational complexity per pixel, we assign one unit for each of the operations of addition and subtraction, and two units for each of the operations of comparison and taking the absolute value, a reasonable assumption on commonly used CPUs. Let $n_t(l)$ denote the computational complexity of $MAD^l(\vec{V})$ per pixel. Then, we have

$$n_t(l) = (n_a(l) + n_s(l) + 2n_{ab}(l))/256 \tag{4.57}$$

since one block consists of 256 pixels. The computational complexity of $MAD^l(\vec{V})$ for a given l, $0 \le l \le 5$, is provided in Table 4.4.

Table 4.4 Computational complexity of $MAD^l(\vec{V})$, $0 \le l \le 5$

l	Additions ($n_a(l)$)	Subtractions ($n_s(l)$)	Absolute value calculations ($n_{ab}(l)$)	Computational complexity ($n_t(l)$)
0	0	1	1	3/256
1	15	16	16	63/256
2	31	32	32	127/256
3	63	64	64	255/256
4	127	128	128	511/256
5	255	256	256	1023/256

4.3.2 Computational Complexity to Calculate the Frame Partial Sums

In Section 4.1.3, we discussed a fast method to calculate the frame partial sums. When $1 \le l \le 4$, we see from (4.39) that for the calculation of $F_c^l(m,n)$ from $F_c^{l+1}(m,n)$, only one addition is needed. For a video frame of size $W \times H$, we need to use (4.39) at most

$$n_f(1) = 4 \times W \times H$$

times to calculate all the frame partial sums of $F_c^l(m,n)$, for $1 \le l \le 4$, $1 \le m \le H$, and $1 \le n \le W$.

When $l = 0$, we use (4.42) and (4.43) to calculate $F_c^0(m,n)$. The number of additions $n_f(2)$ and the number of subtractions $n_f(3)$ needed in calculating all the $F_c^0(m,n)$ values are given by

$$n_f(2) = 15(H-15) + (W-15)(H-16)$$

and

$$n_f(3) = (W-15)(H-16).$$

Let n_f denote the total computational complexity per pixel required to calculate all the frame partial sums. Then

$$n_f < (n_f(1) + n_f(2) + n_f(3))/(WH) < 6 \qquad (4.58)$$

4.3.3 Computational Complexity of the Algorithm

Denoting by $m_t(l)$ $(0 \le l \le 5)$) the average number of times per block that $MAD^l(\vec{V})$ needs to be computed, the computational complexity of the proposed algorithm (VFA) is given by

$$C(VFA) = \sum_{l=0}^{5} m_t(l)\{n_t(l) + \tfrac{2}{256}\} + n_f \tag{4.59}$$

where the factor $\frac{2}{256}$ arises as a consequence of the operation of comparison between $MAD^l(\vec{V}_{curr})$ and $MAD_{\min}$ per block (see Table 4.2).

4.3.4 Theoretical Speedup

The computational complexity of the proposed algorithm as a percentage of the computational complexity of the FSA is

$$\eta_{FSA}^{VFA} = \frac{C(VFA)}{C(FSA)} \times 100\% \tag{4.60}$$

As discussed previously, the proposed algorithm can be implemented using an SIMD architecture, where a four-fold data parallelism can be employed. As a result of this, the execution time of the proposed algorithm using an SIMD architecture as a percentage of the execution time of the algorithm without SIMD implementation is theoretically

$$TS(SIMD) = \frac{1/4}{1} \times 100\% = 25\% \tag{4.61}$$

The computational complexity of the proposed algorithm as a percentage of the computational complexity of the SEA is

$$\eta_{SEA}^{VFA} = \frac{C(VFA)}{C(SEA)} \times 100\% \tag{4.62}$$

4.3.5 Practical Speedup

Let P_{FSA}, P_{SEA}, and P_{VFA} denote the average number of CPU cycles needed per block to carry out the FSA, SEA, and VFA, respectively, without any SIMD implementation. Let

$P_{VFA}(SIMD)$ be the average number of CPU cycles needed per block to implement the proposed VFA using an SIMD architecture. Thus, without any SIMD implementation, the execution time of the VFA as a percentage of that of the FSA and SEA, respectively, are given by

$$PS_{FSA}^{VFA} = \frac{P_{VFA}}{P_{FSA}} \times 100\% \tag{4.63}$$

and

$$PS_{SEA}^{VFA} = \frac{P_{VFA}}{P_{SEA}} \times 100\% \tag{4.64}$$

The execution time of the proposed VFA using an SIMD architecture as a percentage of the execution time of the VFA without an SIMD implementation is

$$PS(SIMD) = \frac{P_{VFA}(SIMD)}{P_{VFA}} \times 100\% \tag{4.65}$$

4.4 Simulation Results

Simulation studies on the proposed algorithm are carried out for the CCITT test video sequences in the QCIF format using the TMN20 framework of the H.263 video codec. The encoding process for a frame is carried out after skipping two frames in between, i.e., the encoding is done for frames 1, 4, 7, ..., 97, 100, ...and so on. The search window is of size 31×31. The SIMD technique employed is the Intel's MMX instruction set.

In our research work, ten video sequences, including the *Salesman, Car Phone, Silent, Akiyo, News, Foreman, Trevor, Mother & Daughter, Miss America,* and *Claire* are selected for the performance evaluation of the proposed algorithm.

Table 4.5 lists the values of $m_t(l)$ $(0 \le l \le 5)$ for various test sequences in the proposed VFA. The values given in Table 4.5 are used in obtaining the values of $C(FSA)$, $C(SEA)$, and $C(VFA)$ as well as those of η_{FSA}^{VFA} and η_{SEA}^{VFA} for the various test sequences and these are given in Table 4.6. From Table 4.6, it can be seen that the computational complexity of the VFA is about 2 to 8 percent of that of the FSA and 11 to 27 of that of the SEA. It should be noted, as mentioned earlier, that a further reduction of 75% can be achieved for the complexity of the proposed VFA by implementing it using an SIMD architecture.

Table 4.7 gives the values of P_{FSA}, P_{SEA}, and P_{VFA}, the total CPU cycles per block of the FSA, SEA, and the proposed VFA without an SIMD implementation for the various test sequences. It also gives the values of PS_{FSA}^{VFA} and PS_{SEA}^{VFA}, the execution time of the VFA as a percentage of the execution time of the FSA and that of the SEA, respectively. It can be seen from this table that the execution time of the proposed VFA is about 2 to 11 percent of that of the FSA and 16 to 40 percent of that of the SEA. Table 4.8 gives the values of P_{VFA} and $P_{VFA}(SIMD)$, the CPU cycles per block for the proposed VFA with and without an SIMD implementation. It is seen from this table that the execution time of the proposed VFA using an SIMD architecture is about 26 percent of that of the VFA with no SIMD implementations. It is noted that this practical speedup of 74% is close to the theoretical speedup of 75%.

Table 4.5 The average number of times per block that $MAD^{l}(\vec{V})$ needs to be calculated in the proposed VFA

Test sequence	$m_t(0)$	$m_t(1)$	$m_t(2)$	$m_t(3)$	$m_t(4)$	$m_t(5)$
Salesman	960	238.86	29.03	6.46	1.85	1.48
Car Phone	960	200.03	66.51	29.78	15.21	8.02
Silent	960	179.85	46.62	17.29	7.11	3.96
Akiyo	960	130.22	17.73	3.59	1.79	1.18
News	960	152.97	36.23	19.08	8.03	2.7
Foreman	960	252.78	107.18	43.58	21.16	11.32
Trevor	960	227.38	32.21	13.7	6.17	4.14
Mother & Daughter	960	182.53	39.01	11	3.59	2.31
Miss America	960	228.83	84.4	34.86	9.69	3.11
Claire	960	102.39	35.51	12.08	2.57	1.59

Table 4.6 Theoretical speedup of the proposed VFA over the FSA and SEA without SIMD implementation

Test sequence	$C(FSA)$ from (B.5)	$C(SEA)$ from (B.24)	$C(VFA)$ from (4.59)	η_{FSA}^{VFA}	η_{SEA}^{VFA}
Salesman	3847.75	978.89	116.14	3.02%	11.86%
Car Phone	3847.75	823.14	201.54	5.24%	24.48%
Silent	3847.75	742.66	141.37	3.67%	19.04%
Akiyo	3847.75	543.67	78.66	2.04%	14.47%
News	3847.75	634.95	127.90	3.32%	20.14%
Foreman	3847.75	1034.54	274.42	7.13%	26.53%
Trevor	3847.75	932.84	141.41	3.68%	15.16%
Mother & Daughter	3847.75	753.07	118.24	3.07%	15.70%
Miss America	3847.75	938.55	192.25	5.00%	20.48%
Claire	3847.75	432.30	92.28	2.40%	21.35%

Table 4.7 Practical speedup of the proposed VFA over the FSA and SEA without SIMD implementation

Test sequence	P_{FSA}	P_{SEA}	P_{VFA}	PS_{FSA}^{VFA}	PS_{SEA}^{VFA}
Salesman	2717760	701748.66	118862.02	4.37%	16.94%
Car Phone	2717760	591821.12	211339.7	7.78%	35.71%
Silent	2717760	534691.54	147934.00	5.44%	27.67%
Akiyo	2717760	394188.82	80619.70	2.97%	20.45%
News	2717760	458594.26	135718.18	4.99%	29.59%
Foreman	2717760	741156.18	289796.36	10.66%	39.10%
Trevor	2717760	669248.78	144709.24	5.32%	21.62%
Mother & Daughter	2717760	542278.62	123129.27	4.53%	22.71%
Miss America	2717760	673353.92	208302.85	7.66%	30.94%
Claire	2717760	315402.09	97969.85	3.60%	31.06%

Table 4.8 Practical speedup of the proposed VFA using SIMD architecture

Test sequence	P_{VFA}	$P_{VFA}(SIMD)$	$PS(SIMD)$
Salesman	118862.02	31279.48	26.32%
Car Phone	211339.70	55615.71	26.32%
Silent	147934.00	38930	26.32%
Akiyo	80619.70	21215.71	26.32%
News	135718.18	35715.31	26.32%
Foreman	289796.36	76262.2	26.32%
Trevor	144709.24	38081.38	26.32%
Mother & Daughter	123129.27	32402.44	26.32%
Miss America	208302.85	54816.54	26.32%
Claire	97969.85	25781.54	26.32%

4.5 Summary

In this chapter, a vector-based fast block motion estimation algorithm, suitable for implementation using SIMD architecture, has been proposed. In this algorithm, certain partial sums of the luminance values have been defined and a fast method to calculate these partial sums developed. These partial sums have been used to calculate some lower bounds for the MAD. These bounds have then been utilized in the algorithm to reduce significantly the number of times $MAD(\bar{V})$ needs to be computed. It has been shown that this algorithm maintains the accuracy and coding efficiency of the full-search algorithm. Simulations have been carried out on a number of test sequences, and the simulation results show that the computational complexity of the algorithm is about 2 to 11 percent of that of the full-search algorithm and 11 to 27 percent of that of the selective elimination algorithm. These results also show that the execution time of the algorithm can be reduced by about 74% by implementing it using an SIMD architecture.

Chapter 5

Fast Block Motion Estimation with Eight Bit Partial Sums Using SIMD Architectures

As discussed in Chapter 2, since the values of the MAD inside a search window have several local minima instead of only one, multi-step search algorithms are often trapped in a local minimum, and hence, these multi-step search algorithms lose the accuracy of the full-search algorithm. The selective elimination algorithm (SEA) described in Chapter 2, and the vector-based algorithm (VFA) proposed in Chapter 4 are fast block motion estimation algorithms belonging to the category of exhaustive search algorithms. This category of fast block motion estimation algorithms exhaustively searches every search location inside the search window as the full-search algorithm does. However, they avoid the computation of the MAD whenever possible by calculating some of the lower bounds of the MAD. Since every search location is searched and the corresponding optimum motion vector is not excluded during the search process, these algorithms can reduce the computational complexity and maintain the accuracy of the full-search algorithm.

As discussed in Chapter 4, the SIMD technique offers a good mechanism to accelerate the implementation of block motion estimation through data parallelism. However, exhaustive search block motion estimation algorithms, such as the SEA, and VFA, cannot take advantage of the byte-type data-parallelism in this technique, since the partial sums of these algorithms are of more than nine bits.

In this chapter, new partial sums of only eight bits, instead of more than nine bits in the literature, are derived to discard as many of the MAD computations as possible, without excluding the optimal motion vector. The presented partial sums can not only be utilized in the full-search as well as in some of the fast block motion estimation algorithms with no loss of accuracy, but also be implemented on SIMD architectures to take advantage of byte-type data-parallelism.

In Section 5.1, we define the eight-bit partial sums of sixteen luminance values. In Section 5.2, the notion of the eight-bit partial sums is extended to the four-level case. It is shown that there are fifteen possible methods of utilizing these multi-level eight-bit partial sums to accelerate a block motion estimation algorithm without any loss of accuracy of the algorithm. Each of these fifteen methods is used in the full-search algorithm to determine the one that provides the lowest computational complexity. This method is adopted as the chosen scheme to accelerate various block motion estimation algorithms. In Section 5.3, computational complexity of the scheme incorporated with various block motion estimation algorithms is discussed. Simulations are carried out in Section 5.4 to demonstrate the effectiveness of proposed scheme to accelerate block motion estimation algorithms.

5.1 Eight Bit Partial Sums of Sixteen Luminance Values

Our objective here is to use eight-bit partial sums to accelerate the block motion estimation process. To achieve this objective, there are two problems that must be solved: the first is a method to obtain eight-bit partial sums, and the second a method to utilize these partial sums in a block motion estimation process.

The partial sums of sixteen luminance values corresponding to the current and reference blocks are defined, respectively, as

$$\begin{cases} BS_{16}^{c}(n) = \sum_{m=0}^{15} B_c(m,n) \\ BS_{16}^{r}(n,\vec{V}) = \sum_{m=0}^{15} B_r(m,n,\vec{V}) \end{cases} \tag{5.1}$$

where m and n are, respectively, the row and column indices in a block. It is seen that the above are of twelve bits. We now define the eight-bit partial sums of sixteen luminance values for the current and reference blocks, respectively, to be the quantity obtained by a four-bit right-shift operation of the 12-bit partial sums $BS_{16}^{c}(n)$ and $BS_{16}^{r}(n,\vec{V})$, i.e.,

$$\begin{cases} PS_c(n) = [BS_{16}^{c}(n)] >> [4] \\ PS_r(n,\vec{V}) = [BS_{16}^{r}(n,\vec{V})] >> [4] \end{cases} \tag{5.2}$$

where the notation of "$[A] >> [B]$" stands for shifting A to the right by B bits [15].

The MAD criterion should be slightly modified in order to use these eight-bit partial sums to reduce the computational complexity of a block motion estimation process. The modified mean absolute difference (MMAD) corresponding to the eight-bit partial sums of sixteen luminance values is defined as

$$MMAD(\vec{V}) = [MAD(\vec{V})] >> [4] + 16 \tag{5.3}$$

It can be seen that the computation of the MMAD requires only two operations, one right shift operation and one addition, after the computation of the MAD. The following theorem gives a lower bound for the MMAD.

Theorem 5.1: A lower bound for $MMAD(\vec{V})$ is $\sum_{n=0}^{15} | PS_c(n) - PS_r(n,\vec{V}) |$.

Proof: The eight-bit partial sums defined in (5.2) can be expressed as

$$\begin{cases} PS_c(n) = BS_{16}^c(n)/16 - \delta_1(n) \\ PS_r(n,\vec{V}) = BS_{16}^r(n,\vec{V})/16 - \delta_2(n) \end{cases} \tag{5.4}$$

where

$$\delta_1(n), \delta_2(n) \in \{i/16 \mid 0 \le i \le 15\} \tag{5.5}$$

Then,

$$S = \sum_{n=0}^{15} | PS_c(n) - PS_r(n,\vec{V}) | \tag{5.6}$$

can be written as

$$S = \sum_{n=0}^{15} | \frac{BS_{16}^c(n)}{16} - \delta_1(n) - \frac{BS_{16}^r(n,\vec{V})}{16} + \delta_2(n) |$$

or
$$S \le \sum_{n=0}^{15} | \frac{BS_{16}^c(n) - BS_{16}^r(n,\vec{V})}{16} | + \sum_{n=0}^{15} | \delta_1(n) - \delta_2(n) | \tag{5.7}$$

In view of (5.5), we have

$$\sum_{n=0}^{15} | \delta_1(n) - \delta_2(n) | \quad \le \quad 15 \tag{5.8}$$

We now consider the first summation

$$\begin{aligned} \sum_{n=0}^{15} | BS_{16}^c(n) - BS_{16}^r(n,\vec{V}) | &= \sum_{n=0}^{15} \left| \sum_{m=0}^{15} B_c(m,n) - \sum_{m=0}^{15} B_r(m,n,\vec{V}) \right| \\ &= \sum_{n=0}^{15} \left| \sum_{m=0}^{15} (B_c(m,n) - B_r(m,n,\vec{V}) \right| \\ &\le \sum_{n=0}^{15} \sum_{m=0}^{15} \left| B_c(m,n) - B_r(m,n,\vec{V}) \right| \\ &= MAD(\vec{V}), \text{ using (2.1).} \end{aligned} \tag{5.9}$$

Hence, from (5.7), (5.8) and (5.9) we get

$$S \le \frac{MAD(\vec{V})}{16} + 15 \tag{5.10}$$

Since

$$\frac{MAD(\vec{V})}{16} \quad \leq \quad \{[MAD(\vec{V})] >> [4]\} + 1$$

we get

$$\begin{aligned} S \leq \quad & \{[MAD(\vec{V})] >> [4]\} + 16 \\ = \quad & MMAD(\vec{V}), \text{ using (5.3).} \end{aligned} \tag{5.11}$$

Hence, the theorem.

We shall denote this lower bound given by Theorem 5.1 as $LB(\vec{V})$.

In an exhaustive or multi-step search algorithm, the computation of $MAD(\vec{V})$ is carried out at every chosen search location of $\vec{V}$. In the following theorem, we show that some of the computations of $MAD(\vec{V})$ can be skipped without loss of accuracy.

Theorem 5.2: If $LB(\vec{V})$ is larger than the minimum value of $MMAD$ computed so far, the computation of $MAD(\vec{V})$ can be skipped without any loss of accuracy in the exhaustive as well as multi-step search algorithms.

Proof: Define $CS(j)$ as the set of all the candidate motion vectors checked so far in a motion estimation process and $\vec{V}_{op}(j)$ as the optimum motion vector in the set $CS(j)$, i.e.,

$$\vec{V}_{op}(j) = \arg\{\min_{\overrightarrow{mv} \in CS(j)} [MAD(\overrightarrow{mv})]\} \tag{5.12}$$

where j is the number of the checked candidate motion vectors. Define $\vec{V}$ as the current candidate motion vector to be checked in a motion estimation process and $CS(j+1)$ as the set containing the current candidate motion vector as well as all the candidate motion vectors checked so far in a motion estimation process, i.e.,

$$CS(j+1) = CS(j) \bigcup \{\vec{V}\} \tag{5.13}$$

Since $CS(j) \subset CS(j+1)$, we can show that

$$MAD(\vec{V}_{op}(j+1)) \leq MAD(\vec{V}_{op}(j)) \tag{5.14}$$

From (5.14) and (5.3), we have

$$MMAD(\vec{V}_{op}(j+1)) \leq MMAD(\vec{V}_{op}(j)) \tag{5.15}$$

From Theorem 5.1

$$LB(\vec{V}_{op}(j+1)) \leq MMAD(\vec{V}_{op}(j+1)) \tag{5.16}$$

From (5.15) and (5.16), we have

$$LB(\vec{V}_{op}(j+1)) \leq MMAD(\vec{V}_{op}(j)) \tag{5.17}$$

Thus, if

$$LB(\vec{V}) > MMAD(\vec{V}_{op}(j)) \tag{5.18}$$

where $MMAD(\vec{V}_{op}(j))$ is the minimum value of $MMAD$ computed so far, then we have

$$\vec{V} \neq \vec{V}_{op}(j+1) \tag{5.19}$$

and the current candidate motion vector of $\vec{V}$ is not a better estimate than $\vec{V}_{op}(j)$. Thus, the computation of $MAD(\vec{V})$ can be skipped if (5.18) holds. Hence, the theorem.

Based on Theorem 5.2, an algorithm can be formulated using the eight-bit partial sums to accelerate a block motion estimation process and is given in Scheme 5.1. In this Scheme, $MMAD_{\min}$, $MAD_{\min}$, and $\vec{V}_o$, respectively, stand for the minimum value of MMAD, the minimum value of MAD, and the optimum motion vector, computed so far.

Scheme 5.1

Step 1) **Initialization**

a) Compute all the eight bit partial sums of sixteen luminance values for the current frame and save them in a continuous memory space.

b) Retrieve all the eight bit partial sums of sixteen luminance values for the reference frame in a saved continuous memory space.

Step 2. For every current block, execute the block motion estimation process.

Step 2.1 Initialization

$\vec{V}_0 = (0,0)$

$MAD_{\min} = MAD(\vec{V}_0)$

$MMAD_{\min} = [MAD_{\min}] >> [4] + 16$

Step 2.2 Search

(*) For (each search location of $\vec{V}$ in a motion estimation algorithm)

{

If ($LB(\vec{V}) > MMAD_{\min}$)

Go to (*) and select next search location

Calculate the $MAD(\vec{V})$

If ($MAD(\vec{V}) < MAD_{\min}$)

{

$MAD_{\min} = MAD(\vec{V})$

$MMAD_{\min} = [MAD_{\min}] >> [4] + 16$

$\vec{V}_0 = \vec{V}$

}

}

5.2 Multi-Level Eight-Bit Partial Sums

In this section, we extend the notion of the eight-bit partial sums discussed in Section 5.1 to the multi-level case. It is shown that the eight-bit partial sums of sixteen luminance values are just the fourth-level eight-bit partial sums. From these multi-level eight-bit partial sums, the multi-level sum of the absolute differences (SAD) of all the eight-bit partial sums between the current and reference blocks is described. An upper bound (UB) for the SAD of a particular level is then established. It is shown that the MMAD described in the previous section is just the UB of the fourth level. Finally, an SIMD implementation for computing the SADs is described.

5.2.1 Formation of Multi-Level Eight-Bit Partial Sums

Consider a column $C(n)$, $0 \le n \le 15$, in a (16×16) block. It has sixteen elements, $C(n) = \{(m,n) \mid 0 \le m \le 15\}$. We first partition this set of $C(n)$ into eight subsets of $C^1(m,n)$, where each subset $C^1(m,n)$ is defined as,

$$C^1(m,n) = \{(k,n) \mid 2m \le k \le 2m+1\} \tag{5.20}$$

where $0 \le m \le 7$ and $0 \le n \le 15$. Thus, each of the subsets consists of two neighboring elements. We say that these eight subsets of $C^1(m,n)$ constitute the first level. We now define every subset in the $(l+1)$-th level partition to be the union of two neighboring subsets in the l-th level, i.e.,

$$C^{l+1}(m,n) = C^l(2m,n) \bigcup C^l(2m+1,n) \tag{5.21}$$

where $0 \le m \le 2^{4-l} - 1$ and $0 \le n \le 15$. Thus, we have partitioned the set $C(n)$ into eight subsets of $C^1(m,n)$ corresponding to level 1, four subsets of $C^2(m,n)$ corresponding to

level 2, two subsets of $C^3(m,n)$ corresponding to level 3, and finally the set $C^4(0,n)$ equals to $C(n)$ itself.

The l-th level partial sums corresponding to the current and reference blocks are defined, respectively, as the summation of all the luminance values in the set $C^l(m,n)$

$$\begin{cases} MBS_c^l(m,n) = \sum\limits_{(k,n)\in C^l(m,n)} B_c(k,n) = \sum\limits_{k=m2^l}^{m2^l+2^l-1} B_c(k,n) \\ MBS_r^l(m,n,\vec{V}) = \sum\limits_{(k,n)\in C^l(m,n)} B_r(k,n,\vec{V}) = \sum\limits_{k=m2^l}^{m2^l+2^l-1} B_r(k,n,\vec{V}) \end{cases} \tag{5.22}$$

where (m,n) is the index of a partial sum and $(m,n) \in \Omega^l = \{(m,n) \mid 0 \le m \le 2^{4-l}-1, 0 \le n \le 15\}$. For a 16×16 block, all the elements in the block consist of a set of $S_B = \{(k,n) \mid 0 \le k \le 15, 0 \le n \le 15\}$. Thus, we have

$$\bigcup_{(m,n)\in\Omega^l} C^l(m,n) = S_B \tag{5.23}$$

From (5.20) and (5.22), a first-level partial sum can be expressed as

$$\begin{cases} MBS_c^1(m,n) = B_c(2m,n) + B_c(2m+1,n) \\ MBS_r^1(m,n,\vec{V}) = B_r(2m,n,\vec{V}) + B_r(2m+1,n,\vec{V}) \end{cases} \tag{5.24}$$

From (5.21) and (5.24), the $(l+1)$-th level partial sum for l=1, 2, 3 can be expressed in terms of the two l-th level partial sums as

$$\begin{cases} MBS_c^{l+1}(m,n) = MBS_c^l(2m,n) + MBS_c^l(2m+1,n) \\ MBS_r^{l+1}(m,n,\vec{V}) = MBS_r^l(2m,n,\vec{V}) + MBS_r^l(2m+1,n,\vec{V}) \end{cases} \tag{5.25}$$

From (5.24) and (5.25), we obtain a fast method to compute the partial sums. In this method, we first use (5.24) to get the partial sums of the first level. Then, we use (5.25) recursively to compute the $(l+1)$-th level partial sums from the l-th level partial sums. From (5.24) and (5.25), we can also see that the l-th level partial sums $MBS_c^l(m,n)$ and

$MBS_r^l(m,n,\vec{V})$ consist of $(8+l)$ bits, since $B_c(m,n)$ and $B_r(m,n,\vec{V})$ are 8-bit quantities. Then, the proposed multi-level eight-bit partial sums for the current and reference blocks can be, respectively, obtained by an l-bit right-shift operation of $MBS_c^l(m,n)$ and $MBS_r^l(m,n,\vec{V})$, i.e.,

$$\begin{cases} MPS_c^l(m,n) = [MBS_c^l(m,n)] >> [l] \\ MPS_r^l(m,n,\vec{V}) = [MBS_r^l(m,n,\vec{V})] >> [l] \end{cases} \tag{5.26}$$

From (5.26), we see that

$$\begin{cases} MPS_c^4(0,n) = PS_c(n) \\ MPS_r^4(0,n,\vec{V}) = PS_r(n,\vec{V}) \end{cases} \tag{5.27}$$

Thus, the eight-bit partial sums of sixteen luminance values are just the fourth-level eight- bit partial sums. From (5.26), we can also see that

$$\begin{cases} MPS_c^l(m,n) = \lfloor MBS_c^l(m,n)/2^l \rfloor \\ MPS_r^l(m,n,\vec{V}) = \lfloor MBS_r^l(m,n,\vec{V})/2^l \rfloor \end{cases} \tag{5.28}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x.

We now define the l-th level sum of the absolute differences of all the eight-bit partial sums between the current and reference blocks as

$$SAD^l(\vec{V}) = \sum_{(m,n)\in\Omega^l} | MPS_c^l(m,n) - MPS_r^l(m,n,\vec{V}) | \tag{5.29}$$

From (5.29), it can be seen that $LB(\vec{V}) = SAD^4(\vec{V})$.

5.2.2 Upper Bounds for the SADs

In order to use the multi-level eight-bit partial sums to accelerate a block motion estimation process, we must find an upper bound (UB) for $SAD^l(\vec{V})$. The process to find this UB is described below.

Using (5.28) in (5.29), we have

$$SAD^l(\vec{V}) = \sum_{(m,n)\in\Omega^l} \left| D(m,n,\vec{V}) \right| \tag{5.30}$$

where

$$D(m,n,\vec{V}) = \left\lfloor MBS_c^l(m,n)/2^l \right\rfloor - \left\lfloor MBS_r^l(m,n,\vec{V})/2^l \right\rfloor \tag{5.31}$$

At the same time, we have

$$\left\lfloor A/2^l \right\rfloor = A/2^l - \delta \tag{5.32}$$

where $\delta \in \Gamma^l = \{i/2^l \mid 0 \le i \le 2^l - 1\}$, and A and i are integers.

Using (5.22) and (5.32) in (5.30), we have

$$SAD^l(\vec{V}) = \sum_{(m,n)\in\Omega^l} | G(m,n,\vec{V})/2^l - \delta(m,n,\vec{V}) | \tag{5.33}$$

where

$$G(m,n,\vec{V}) = \sum_{(k,n)\in C^l(m,n)} (B_c(k,n) - B_r(k,n,\vec{V})) \tag{5.34}$$

$$\delta(m,n,\vec{V}) = \delta_1(m,n) - \delta_2(m,n,\vec{V}), \quad \delta_1, \delta_2 \in \Gamma^l \tag{5.35}$$

From (5.33), we have the inequality

$$SAD^l(\vec{V}) \le \sum_{(m,n)\in\Omega^l} (| G(m,n,\vec{V}) | / 2^l + | \delta(m,n,\vec{V}) |) \tag{5.36}$$

From (2.1), (5.23) and (5.34), we have

$$\sum_{(m,n)\in\Omega^l} |G(m,n,\vec{V})| \le \sum_{(m,n)\in\Omega^l} \left(\sum_{(k,n)\in C^l(m,n)} |B_c(k,n) - B_r(k,n,\vec{V})| \right)$$
$$= \sum_{(k,n)\in S_B} |B_c(k,n) - B_r(k,n,\vec{V})|$$
$$= MAD(\vec{V}) \tag{5.37}$$

From (5.35), we have

$$|\delta(m,n,\vec{V})| = |\delta_1(m,n) - \delta_2(m,n,\vec{V})| \tag{5.38}$$

Since $\delta_1(m,n)$ and $\delta_2(m,n,\vec{V}) \in \Gamma^l$

$$|\delta(m,n,\vec{V})| \le |\delta_{\max} - \delta_{\min}| \tag{5.39}$$

where $\delta_{\max}$ and $\delta_{\min}$ are the maximum and minimum values of $\delta \in \Gamma^l$. Hence

$$|\delta(m,n,\vec{V})| \le |(2^l - 1)/2^l - 0| = (2^l - 1)/2^l \tag{5.40}$$

From (5.36), (5.37), and (5.40) we have

$$SAD^l(\vec{V}) \le MAD(\vec{V})/2^l + 2^{8-l} - 2^{8-2l} \tag{5.41}$$

From (5.41),

$$SAD^l(\vec{V}) \le \{E(l) + [MAD(\vec{V})] >> [l]\} \tag{5.42}$$

where $E(l) = 2^{8-l} - 2^{8-2l} + 1$.

Thus, an upper bound for $SAD^l(\vec{V})$ is found, namely,

$$UB^l(\vec{V}) = E(l) + [MAD(\vec{V})] >> [l] \tag{5.43}$$

and we have

$$SAD^l(\vec{V}) \le UB^l(\vec{V}) \tag{5.44}$$

It can be seen that the computation of $UB^l(\vec{V})$ requires only two operations after the computation of $MAD(\vec{V})$, namely, one right shift operation of $MAD(\vec{V})$ by l bits and one addition. From (5.3) and (5.43), we also see that $MMAD(\vec{V}) = UB^4(\vec{V})$, i.e., MMAD is just the fourth-level UB.

5.2.3 The Methods of Using Multi-Level Eight-Bit Partial Sums

In this section, we first describe a method of using the eight-bit partial sums at a particular level to accelerate the block motion estimation process. Then, we give all the possible methods of using the eight-bit partial sums at the four levels to accelerate the block motion estimation process.

Theorem 5.3: If $SAD^l(\vec{V})$ is larger than the minimum value of UB^l computed so far, the corresponding computation of $MAD(\vec{V})$ can be skipped without any loss of accuracy of the exhaustive as well as multi-step search algorithms.

Proof: From (5.14), we have

$$MAD(\vec{V}_{op}(j+1)) \le MAD(\vec{V}_{op}(j)) \tag{5.45}$$

where $\vec{V}_{op}(j)$ and $\vec{V}_{op}(j+1)$ are, respectively, the optimal motion vectors before and after checking the current motion vector.

From (5.43) and (5.45), we have

$$UB^l(\vec{V}_{op}(j+1)) \le UB^l(\vec{V}_{op}(j)) \tag{5.46}$$

From (5.44), we have

$$SAD^l(\vec{V}_{op}(j+1)) \le UB^l(\vec{V}_{op}(j+1)) \tag{5.47}$$

From (5.46) and (5.47), we get the inequality

$$SAD^l(\vec{V}_{op}(j+1)) \le UB^l(\vec{V}_{op}(j)) \tag{5.48}$$

Thus, if

$$SAD^l(\vec{V}) > UB^l(\vec{V}_{op}(j)) \tag{5.49}$$

where $UB^l(\vec{V}_{op}(j))$ is the minimum value of UB^l computed so far, then

$$\vec{V} \neq \vec{V}_{op}(j+1) \tag{5.50}$$

and the current candidate motion vector of $\vec{V}$ is not a better estimate than $\vec{V}_{op}(j)$. Thus, the computation of $MAD(\vec{V})$ can be skipped if (5.49) holds, and hence the Theorem.

From Theorem 5.3, it is seen that there are four conditions that can be used to reduce the computational complexity of a block motion estimation process. These are

Condition 1: $SAD^1(\vec{V}) > UB^1(\vec{V}_{op}(j))$

Condition 2: $SAD^2(\vec{V}) > UB^2(\vec{V}_{op}(j))$

Condition 3: $SAD^3(\vec{V}) > UB^3(\vec{V}_{op}(j))$

Condition 4: $SAD^4(\vec{V}) > UB^4(\vec{V}_{op}(j))$

Each of the above conditions correspond to a particular level of l $(1 \leq l \leq 4)$. There are fifteen possible combinations in which these conditions can be used and each one gives rise to a method of using the conditions to accelerate the block motion estimation process. These methods are listed in Table 5.1. For a particular method in this table, the symbol 'X' denotes that this condition is employed and the symbol 'O' denotes that this condition is not employed. For example, for Method 8, only Conditions 2 and 3 are used. For each of these fifteen methods, the accuracy of the block motion estimation process is maintained, since from Theorem 5.3, we know that if any of the conditions used in these methods is satisfied, the computation of $MAD(\vec{V})$ can be skipped without loss of accuracy.

Table 5.1 Conditions used for the various methods

Conditions \ Methods	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Condition 1	X	O	O	O	X	X	X	O	O	O	X	X	X	O	X
Condition 2	O	X	O	O	X	O	O	X	X	O	X	X	O	X	X
Condition 3	O	O	X	O	O	X	O	X	O	X	X	O	X	X	X
Condition 4	O	O	O	X	O	O	X	O	X	X	O	X	X	X	X

5.2.4 Optimal Method of Using Multi-Level Eight-Bit Partial Sums

In Section 5.2.3, fifteen possible methods of using the multi-level eight-bit partial sums were given. In this section, each of these methods is simulated and evaluated in order to find the optimal one among them. This optimal method is finally illustrated as our scheme to accelerate the block motion estimation process.

Table 5.2 lists the computational complexity corresponding to each of the fifteen methods to accelerate the full-search algorithm for various video test sequences. From this table, we see that Method 7, which uses conditions 1 and 4, has the least average computational complexity. Thus, the method using the eight- bit partial sums of levels 1 and 4 has the least average computational complexity among all the possible methods. This method is formalized as Scheme 5.2 and is used to accelerate the block motion estimation process.

Table 5.2 Computational complexity of the various methods

Method / Sequence	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Salesman	486	682	1120	1938	471	458	374	687	593	1051	476	382	390	619	408
Hall objects	562	759	1173	1996	558	530	470	760	689	1136	558	487	493	722	521
Silent	531	702	1151	1951	497	501	411	723	616	1080	518	411	430	653	447
Akiyo	481	671	1111	1962	460	448	396	676	611	1071	464	400	408	636	424
News	577	769	1172	1998	569	533	481	761	696	1138	562	497	499	727	527
Foreman	854	951	1313	2023	787	743	643	948	812	1233	783	647	663	867	702
Trevor	510	708	1129	1970	500	473	419	702	638	1088	494	430	432	662	454
Mother & Daughter	800	953	1344	2084	781	760	679	979	874	1310	807	703	727	945	773
Average	600	774	1189	1990	578	556	484	779	691	1138	583	494	505	729	532

Scheme 5.2

***Step 1)* Initialization**

a) Compute all the eight bit partial sums of levels one and four for the current frame and save them in a continuous memory space.

b) Retrieve all the eight bit partial sums of levels one and four for the reference frame in a saved continuous memory space.

Step 2. For every current block, execute the block motion estimation process.

Step 2.1 Initialization

$\vec{V}_0 = (0,0)$

$MAD_{\min} = MAD_{\min\_so\_far} = MAD(\vec{V}_0)$

$UB^4_{\min} = [MAD_{\min}] >> [4] + 16$

$UB^1_{\min} = [MAD_{\min}] >> [1] + 65$

Step 2.2 Search

(*) For (each search location of $\vec{V}$ in a motion estimation algorithm)

{

If ($SAD^4(\vec{V}) > UB^4_{\min}$)

Go to (*) and select next search location

If ($SAD^1(\vec{V}) > UB^1_{\min}$)

Go to (*) and select next search location

Calculate the $MAD(\vec{V})$

If ($MAD(\vec{V}) < MAD_{\min\_so\_far}$)

{

$MAD_{\min} = MAD_{\min\_so\_far} = MAD(\vec{V})$

$UB^4_{\min} = [MAD_{\min}] >> [4] + 16$

$UB^1_{\min} = [MAD_{\min}] >> [1] + 65$

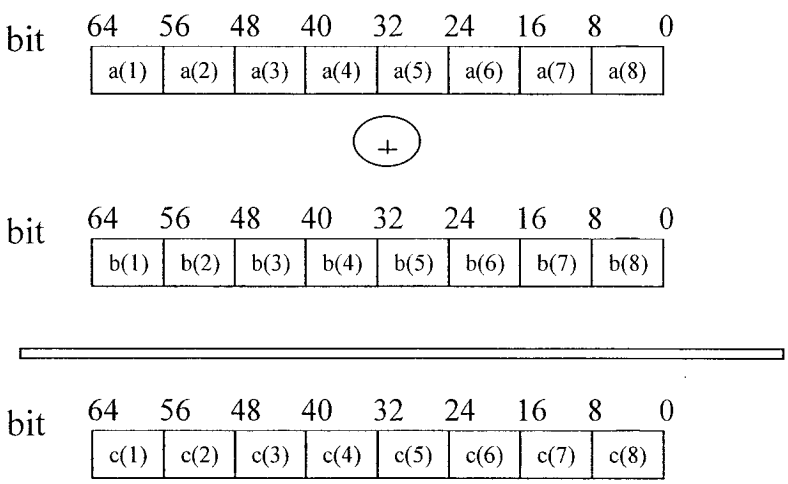
$\vec{V}_0 = \vec{V}$

}

}

5.2.5 SIMD Implementation for the Computation of SAD

Since the partial sums in the SEA, and VFA are of more than eight-bits, they cannot be in the packed byte format on SIMD architectures. Now, let us look carefully at the computation of the SADs. Since the partial sums in (5.26) are of only eight bits, they can be in packed byte format on SIMD architectures. In the SIMD architecture of Intel MMX, eight of these partial sums can be put into a 64-bit SIMD register, so that eight of them can be manipulated in one SIMD instruction as shown in Figure 5.1. Thus, an eight-fold data parallelism can be realized on such an architecture.



Note: In the above figure, c(i)=a(i)+b(i)

Figure 5.1 Packed byte-type addition on SIMD registers

5.3 Computational Complexity

In this section, we first evaluate the computational complexity of a block motion estimation algorithm incorporating the proposed Scheme 5.2. Then, we discuss the theoretical and practical speedups that can be achieved by incorporating Scheme 5.2 in a given block motion estimation algorithm.

5.3.1 Computational Complexity of a Block Motion Estimation Algorithm Incorporating the Proposed Scheme

Let $\eta'_{LB}(x)$ and $\eta'_{MAD}(x)$ respectively denote the number of times $SAD^1(\vec{V})$ and $MAD(\vec{V})$ need to be calculated in the block motion estimation algorithm x incorporating Scheme 5.2. Let $C'(x)$ denote the corresponding computational complexity per pixel. Then, the computational complexity of the FSA, with Scheme 5.2 incorporated, can be expressed as

$$\begin{aligned} C'(FSA) &= (16+15+16\times 2+2)\times \tfrac{S^2}{256} + \eta'_{LB}(FSA)\times \tfrac{128+128\times 2+127+2}{256} \\ &\quad + \eta'_{MAD}(FSA)\times \tfrac{256\times 2+256+255+2+4}{256} + C'_{ps}(H) \\ &= \tfrac{65}{256}S^2 + \tfrac{513}{256}\eta'_{LB}(FSA) + \tfrac{1029}{256}\eta'_{MAD}(FSA) + C'_{ps}(H) \end{aligned} \tag{5.51}$$

where $C'_{ps}(H)$ is the computational complexity per pixel to calculate all the eight-bit partial sums in a frame. In Scheme 5.2, all the eight-bit partial sums in a frame are calculated using a fast computational mechanism similar to that in the SEA. These partial sums are saved in a contiguous memory space before the motion estimation process begins. Thus, $C'_{ps}(H)$ can be expressed as

$$C'_{ps}(H) = \tfrac{2W(H-1)+16W+3(H-16)W}{WH} = 5 - \tfrac{34}{H} \tag{5.52}$$

The computational complexity per pixel of the 2DLSA, OSA, OATSA, CDSA, 3SSA, 4SSA, UDSA, PSAA, BGDSA, and SEA, incorporating Scheme 5.2, can be expressed as

$$C'(2DLSA) = \tfrac{65}{256}\eta_{MAD}(2DLSA) + \tfrac{513}{256}\eta'_{LB}(2DLSA) + \tfrac{1029}{256}\eta'_{MAD}(2DLSA) + C'_{ps}(H) \quad (5.53)$$

$$C'(OSA) = \tfrac{65}{256}\eta_{MAD}(OSA) + \tfrac{513}{256}\eta'_{LB}(OSA) + \tfrac{1029}{256}\eta'_{MAD}(OSA) + C'_{ps}(H) \quad (5.54)$$

$$C'(OATSA) = \tfrac{65}{256}\eta_{MAD}(OATSA) + \tfrac{513}{256}\eta'_{LB}(OATSA) + \tfrac{1029}{256}\eta'_{MAD}(OATSA) + C'_{ps}(H) \quad (5.55)$$

$$C'(CDSA) = \tfrac{65}{256}\eta_{MAD}(CDSA) + \tfrac{513}{256}\eta'_{LB}(CDSA) + \tfrac{1029}{256}\eta'_{MAD}(CDSA) + C'_{ps}(H) \quad (5.56)$$

$$C'(3SSA) = \tfrac{65}{256}\eta_{MAD}(3SSA) + \tfrac{513}{256}\eta'_{LB}(3SSA) + \tfrac{1029}{256}\eta'_{MAD}(3SSA) + C'_{ps}(H) \quad (5.57)$$

$$C'(4SSA) = \tfrac{65}{256}\eta_{MAD}(4SSA) + \tfrac{513}{256}\eta'_{LB}(4SSA) + \tfrac{1029}{256}\eta'_{MAD}(4SSA) + C'_{ps}(H) \quad (5.58)$$

$$C'(UDSA) = \tfrac{65}{256}\eta_{MAD}(UDSA) + \tfrac{513}{256}\eta'_{LB}(UDSA) + \tfrac{1029}{256}\eta'_{MAD}(UDSA) + C'_{ps}(H) \quad (5.59)$$

$$C'(PSAA) = \tfrac{65}{256}\eta_{MAD}(PSAA) + \tfrac{513}{256}\eta'_{LB}(PSAA) + \tfrac{1029}{256}\eta'_{MAD}(PSAA) + C'_{ps}(H) \quad (5.60)$$

$$C'(BGDSA) = \tfrac{65}{256}\eta_{MAD}(BGDSA) + \tfrac{513}{256}\eta'_{LB}(BGDSA) + \tfrac{1029}{256}\eta'_{MAD}(BGDSA) + C'_{ps}(H) \quad (5.61)$$

$$C'(SEA) = 4 - \tfrac{17}{W} - \tfrac{47}{H} + \tfrac{255}{WH} + \tfrac{5}{256}S^2 + \tfrac{65}{256}\eta_{MAD}(SEA) + \tfrac{513}{256}\eta'_{LB}(SEA) + \tfrac{1029}{256}\eta'_{MAD}(SEA) + C'_{ps}(H) \quad (5.62)$$

where $\eta_{MAD}(x)$ is defined in Section 2.4 and $C'_{ps}(H)$ is given by (5.52).

5.3.2 Theoretical Speedup of a Block Motion Estimation Algorithm Incorporating Scheme 5.2

The proposed scheme of using eight-bit partial sums for a block motion estimation algorithm can greatly reduce the computational complexity of algorithm x, since the

value of $\eta'_{MAD}(x)$ is much less than that of $\eta_{MAD}(x)$. The percentage of the reduction in the computational complexity is given by

$$RCC(x) = \frac{C(x) - C'(x)}{C(x)} \times 100 \tag{5.64}$$

This reduction $RCC(x)$ may be considered as a measure of the theoretical speedup that results as a consequence of incorporating Scheme 5.2.

5.3.3 Practical Speedup of a Block Motion Estimation Algorithm Incorporating Scheme 5.2

Let $P(x)$ be the average number of CPU cycles needed per block to carry out the block motion estimation using algorithm x. Let $P'(x)$ denote the average number of CPU cycles required per block to carry out the estimation using algorithm x with Scheme 5.2 incorporated. Then, the percentage of practical speedup that results as a consequence of incorporating Scheme 5.2, is expressed as

$$PS(x) = \frac{P(x) - P'(x)}{P(x)} \times 100 \tag{5.65}$$

5.4 Simulation Results

A simulation study on the proposed method for fast block motion estimation is carried out for the CCITT test video sequences in the QCIF format using the TMN20 framework of the H.263 video codec. The encoding process for a frame is carried out after skipping two frames in between, i.e. the encoding is done for the frames 1, 4, 7,...and so on.

In our research work, seven video sequences, including the *Salesman, Car Phone, Akiyo, News, Foreman, Trevor, and Mother & Daughter*, are selected for the performance evaluation of the proposed method. For the seven video sequences considered, Table 5.3 gives the computational complexity per pixel as well as the average number of CPU cycles needed per block to carry out the block motion estimation on SIMD architecture of Intel's MMX using FSA, with and without Scheme 5.2 being incorporated. It also includes the reduction in the computational complexity (RCC) and the practical speedup (PS) for these sequences. Similar results are given in Tables 5.4-5.13 for 2DLSA, OSA, OATSA, CDSA, 3SSA, 4SSA, UDSA, PSAA, BGDSA, and SEA respectively. From these tables, we see that Scheme 5.2 reduces the computational complexity of FSA, 2DLSA, OSA, OATSA, CDSA, 3SSA, 4SSA, UDSA, PSAA, BGDSA, and SEA by more than 80%, 30%, 30%, 20%, 18%, 35%, 30%, 28%, 45%, 27%, and 50%, respectively. It is also seen that the proposed scheme accelerates the execution of these algorithms on SIMD architectures by more than 70%, 25%, 20%, 14%, 12%, 30%, 20%, 20%, 38%, 27%, and 50%, respectively.

5.5 Summary

In this chapter, a new concept of an eight-bit partial sum, that is obtained by carrying out a four-bit right-shift operation on the sum of the 16 luminance values of a column of a 16×16 block of a video frame, has been introduced. These partial sums have been defined so as to take advantage of the byte-type data parallelism in the existing single-instruction multiple-data (SIMD) technique for an improved speed of a given motion estimation algorithm. Since these partial sums have the characteristic of having only eight

Table 5.3 Computational complexity and average number of CPU cycles per block using the FSA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(FSA)$	60.78	130.70	62.51	83.3	152.77	67.08	141.57
$\eta'_{MAD}(FSA)$	3.24	31.32	9.29	18.41	24.38	11.27	39.88
$C(FSA)$	3847.8	3847.8	3847.8	3847.8	3847.8	3847.8	3847.8
$C'(FSA)$	383.33	635.87	411.02	489.19	652.31	428.10	691.93
$P(FSA)$	715200	715200	715200	715200	715200	715200	715200
$P'(FSA)$	88259	140642	93545	109695	145404	97076	151911
$RCC(FSA)$	90.04%	83.47%	89.32%	87.29%	83.05%	88.87%	82.02%
$PS(FSA)$	87.66%	80.34%	86.92%	84.66%	79.67%	86.43%	78.76%

Table 5.4 Computational complexity and average number of CPU cycles per block using the 2DLSA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(2DLSA)$	7. 24	9. 68	6. 44	6. 69	10. 09	8. 43	9. 12
$\eta'_{MAD}(2DLSA)$	1. 72	4. 04	2. 53	3. 55	4. 8	4. 12	4. 39
$C(2DLSA)$	68. 39	72. 91	68. 15	69. 39	77. 56	71. 67	69. 11
$C'(2DLSA)$	30. 28	44. 75	31. 91	36. 57	48. 91	42. 49	44. 79
$P(2DLSA)$	12725	13567	12680	12911	14431	13336	12859
$P'(2DLSA)$	6398	9292	6638	7529	10113	8771	9244
$RCC(2DLSA)$	55. 7%	38. 6%	53. 2%	47. 3%	37. 0%	40. 7%	35. 2%
$PS(2DLSA)$	49. 7%	31. 5%	47. 6%	41. 7%	29. 9%	34. 2%	28. 1%

Table 5.5 Computational complexity and average number of CPU cycles per block using the OSA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(OSA)$	6.57	8.56	6.29	6.59	10.4	8.04	8.37
$\eta'_{MAD}(OSA)$	1.75	4.65	2.64	3.6	4.14	4.31	4.09
$C(OSA)$	68.07	68.07	68.07	68.07	68.03	68.07	68.07
$C'(OSA)$	29.04	44.64	32.04	36.49	46.28	42.24	42.02
$P(OSA)$	12665	12665	12665	12665	12658	12665	12665
$P'(OSA)$	6114	9170	6651	7502	9618	8683	8668
$RCC(OSA)$	57.3%	34.4%	52.9%	46.4%	32.0%	38.0%	38.3%
$PS(OSA)$	51.7%	27.6%	47.5%	40.8%	24.0%	31.4%	31.6%

Table 5.6 Computational complexity and average number of CPU cycles per block using the OATSA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(OATSA)$	2. 17	3. 34	2. 18	2. 21	3. 15	2. 1	2. 61
$\eta'_{MAD}(OATSA)$	1. 51	1. 60	1. 21	1. 55	2. 42	1. 85	1. 42
$C(OATSA)$	20. 54	24. 70	20. 26	21. 42	28. 07	24. 74	21. 46
$C'(OATSA)$	16. 25	19. 22	15. 05	16. 55	22. 33	17. 74	16. 83
$P(OATSA)$	3822	4597	3770	3986	5223	4604	3993
$P'(OATSA)$	3243	3899	3020	3304	4475	3528	3388
$RCC(OATSA)$	20. 9%	22. 2%	25. 7%	22. 8%	20. 4%	28. 3%	21. 6%
$PS(OATSA)$	15. 1%	15. 2%	19. 9%	17. 1%	14. 3%	23. 4%	15. 2%

Table 5.7 Computational complexity and average number of CPU cycles per block using the CDSA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(CDSA)$	3.13	4.12	3.13	3.23	5.23	3.06	3.52
$\eta'_{MAD}(CDSA)$	1.64	2.87	2.19	2.12	3.51	2.45	2.41
$C(CDSA)$	28.55	32.87	28.27	29.43	39.36	32.95	29.51
$C'(CDSA)$	19.20	26.38	21.39	21.38	31.58	22.58	23.13
$P(CDSA)$	5312	6117	5260	5476	7323	6131	5491
$P'(CDSA)$	3892	5319	4298	4308	6392	4530	4656
$RCC(CDSA)$	32.7%	19.7%	24.3%	27.4%	19.8%	31.5%	21.6%
$PS(CDSA)$	26.7%	13.0%	18.3%	21.3%	12.7%	26.1%	15.2%

Table 5.8 Computational complexity and average number of CPU cycles per block using the 3SSA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(3SSA)$	10.00	14.63	8.94	9.88	18.59	11.7	14.38
$\eta'_{MAD}(3SSA)$	1.92	6.60	3.01	4.52	7.65	4.91	6.14
$C(3SSA)$	132.13	132.13	132.13	132.13	132.13	132.13	132.13
$C'(3SSA)$	40.66	68.67	42.90	50.83	80.81	56.04	66.33
$P(3SSA)$	24585	24585	24585	24585	24585	24585	24585
$P'(3SSA)$	8744	14315	9080	10627	16879	11737	13859
$RCC(3SSA)$	69.2%	48.0%	67.5%	61.5%	38.8%	57.6%	49.8%
$PS(3SSA)$	64.4%	41.8%	63.1%	56.8%	31.4%	52.3%	43.6%

Table 5.9 Computational complexity and average number of CPU cycles per block using the 4SSA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(4SSA)$	8.08	11.64	6.71	7.25	14.46	9.22	10.53
$\eta'_{MAD}(4SSA)$	1.83	4.98	2.68	3.7	5.61	4.61	4.02
$C(4SSA)$	68.71	77.08	68.31	70.31	89.09	75.35	70.07
$C'(4SSA)$	32.43	52.71	33.06	38.35	61.64	46.27	46.19
$P(4SSA)$	12784	14341	12710	13082	16576	14021	13038
$P'(4SSA)$	6863	10937	6874	7907	12856	9547	9617
$RCC(4SSA)$	52.8%	31.6%	51.6%	45.5%	30.8%	38.6%	34.1%
$PS(4SSA)$	46.3%	23.7%	45.9%	39.6%	22.5%	31.9%	26.2%

Table 5.10 Computational complexity and average number of CPU cycles per block using the UDSA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(UDSA)$	7.36	10.69	6.17	6.61	13.1	9.02	8.06
$\eta'_{MAD}(UDSA)$	1.83	3.90	2.6	3.56	5.04	3.84	3.49
$C(UDSA)$	53.09	65.10	52.49	55.53	81.36	63.30	55.25
$C'(UDSA)$	30.00	45.72	30.66	35.57	56.14	42.02	38.18
$P(UDSA)$	9879	12114	9767	10333	15138	11779	10281
$P'(UDSA)$	6305	9526	6334	7293	11703	8702	7889
$RCC(UDSA)$	43.5%	29.8%	41.6%	35.9%	31.0%	33.6%	30.9%
$PS(UDSA)$	36.2%	21.4%	35.2%	29.4%	22.7%	26.1%	23.3%

Table 5.11 Computational complexity and average number of CPU cycles per block using the PSAA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(PSAA)$	13.82	19.66	9.91	12.05	25.19	14.54	20.03
$\eta'_{MAD}(PSAA)$	2.88	8.35	3.91	5.4	9.57	5.78	8.00
$C(PSAA)$	148.95	183.30	145.38	158.96	206.00	175.37	157.71
$C'(PSAA)$	53.22	89.01	49.29	60.40	106.41	67.95	86.72
$P(PSAA)$	27714	34106	27051	29577	38330	32631	29346
$P'(PSAA)$	11431	18649	10385	12662	22386	14311	18171
$RCC(PSAA)$	64.3%	51.4%	66.1%	62.0%	48.3%	61.3%	45.0%
$PS(PSAA)$	58.8%	45.3%	61.6%	57.2%	41.6%	56.1%	38.1%

Table 5.12 Computational complexity and average number of CPU cycles per block using the BGDSA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(BGDSA)$	5.64	7.03	4.39	5.05	10.66	5.92	4.96
$\eta'_{MAD}(BGDSA)$	1.69	3.66	2.21	3.08	5.81	3.69	2.79
$C(BGDSA)$	37.96	53.45	36.88	40.56	74.91	52.05	40.92
$C'(BGDSA)$	25.03	36.68	24.54	29.58	53.93	34.49	28.26
$P(BGDSA)$	7063	9946	6862	7547	13939	9685	7614
$P'(BGDSA)$	5200	7525	5009	6009	11082	7027	5758
$RCC(BGDSA)$	34.1%	31.4%	33.5%	27.1%	28.0%	33.7%	30.9%
$PS(BGDSA)$	26.4%	24.3%	27.0%	20.4%	20.5%	27.4%	24.4%

Table 5.13 Computational complexity and average number of CPU cycles per block using the SEA

	Salesman	Car phone	Akiyo	News	Foreman	Trevor	Mother & Daughter
$\eta'_{LB}(SEA)$	51.46	101.51	38.79	56.37	132.10	59.28	72.00
$\eta'_{MAD}(SEA)$	3.08	27.48	4.80	13.03	21.86	9.66	20.08
$C(SEA)$	978.89	823.14	543.67	634.95	1034.54	932.84	753.07
$C'(SEA)$	203.24	391.38	157.16	231.12	443.57	242.35	298.17
$P(SEA)$	184671	155742	103734	120683	195041	176118	142705
$P'(SEA)$	48142	86513	37204	52611	99257	55875	66670
$RCC(SEA)$	79.2%	52.5%	71.1%	63.6%	57.1%	74.0%	60.4%
$PS(SEA)$	73.9%	44.5%	64.1%	56.4%	49.1%	68.3%	53.3%

bits, eight of them can be processed concurrently in a single 64-bit SIMD register. A method of employing these partial sums to speedup a block motion estimation process has been proposed. The notion of the eight-bit partial sums has then been extended to the four-level case and shown that there are fifteen possible methods of utilizing these multi-level partial sums to accelerate the block motion estimation algorithms without any loss of accuracy. The full-search algorithm has then been used to determine as to which one of these fifteen methods would provide the lowest computational complexity in order for it to be chosen to accelerate various motion estimation algorithms. Simulations have been carried out to find the average number of CPU cycles needed per block for various algorithms incorporating the chosen method. These simulations have shown that the proposed scheme is capable of providing a substantial speedup for the various existing motion estimation algorithms.

Chapter 6

Conclusion

6.1 Concluding Remarks

Recent advances in communications, digital signal processing, and computer vision have led to a surge of multimedia applications where video signals play an important role. In order to overcome the problems arising from large volume of data of the video signals, video compression becomes a necessity of multimedia systems. As a result of this, several international standards in video coding have been proposed to pave the way for ubiquitous applications of multimedia products. In these standards, block motion estimation is an essential part to remove the temporal redundancies in video signals. However, these standards do not specify the exact motion estimation algorithm to be employed and this task is left to the developers implementing the standards. For this reason, the development of fast block motion estimation algorithms with good compression efficiency has been a focus of recent research activities and is expected to continue to attract a great deal of research effort in the near future. In view of this, this thesis has been concerned with developing techniques to reduce the computational complexity of a given block motion estimation algorithm without sacrificing its accuracy, to utilize the single instruction multiple data (SIMD) technique to accelerate a block motion estimation process, and to develop a new fast block motion estimation algorithm suitable for implementation on SIMD architectures.

The full-search algorithm as well as the existing fast motion estimation algorithms have been reviewed. These fast algorithms have been categorized into multi-step and exhaustive search algorithms. Simulation results have shown that the computational complexity of any of the multi-step search algorithms is substantially smaller than that of the full-search algorithm, but with a reduced coding efficiency. Simulation results have also shown that the selective elimination algorithm has as a high computational complexity as the full-search algorithm, and hence is not suitable for real- time applications.

Through a simulation study, it has been established that a large number of macroblocks generally do not move from one frame to the next. Such macroblocks have been referred to as stationary macroblocks. Since the value of the motion vector for a stationary macroblock is already known to be $(0,0)$, the search process for such a macroblock in a given block motion estimation algorithm is unnecessary. Based on this observation, a fast block motion estimation method has been proposed by skipping the search process for the stationary macroblocks. Simulation studies have shown that the proposed method can speedup a given block motion estimation algorithm, with about the same coding efficiency as that of the original algorithm. These studies have also shown that the amount of speedup resulting from the application of the proposed scheme to a given block motion estimation algorithm is approximately the same, irrespective of the algorithm chosen for the application. This result is due to the fact that the speedup is mainly governed by the characteristics of the video sequence rather than the chosen algorithm.

A vector-based fast block motion estimation algorithm, suitable for implementation on an SIMD architecture, has been proposed. In this algorithm, certain partial sums of the luminance values have been defined and a fast method to calculate these partial sums developed. These partial sums have been used to calculate some lower bounds for the MAD. These bounds have then been utilized in the algorithm to reduce significantly the number of times $MAD(\vec{V})$ needs to be computed. It has been shown that this algorithm maintains the accuracy and coding efficiency of the full-search algorithm. Simulation results have shown that the computational complexity of this algorithm is about 2 to 11 percent of that of the full-search algorithm and 11 to 27 percent of that of the selective elimination algorithm. These results have also shown that the execution time of the algorithm can be reduced by about 74% by implementing it on an SIMD architecture.

The concept of an eight-bit partial sum has been introduced in this thesis. These partial sums have been formed so as to take advantage of the byte-type data parallelism in the existing SIMD technique for an improved speed of a given motion estimation algorithm. Since these partial sums have the characteristic of having only eight bits, eight of them can be processed concurrently in a single 64-bit SIMD register. A method of employing these partial sums to speedup a block motion estimation process has been proposed. The notion of the eight-bit partial sums has then been extended to the four-level case and shown that there are fifteen possible methods of utilizing these multi-level partial sums to accelerate a given block motion estimation algorithms without any loss of accuracy. The full-search algorithm has then been used to determine as to which one of these fifteen methods would provide the lowest computational complexity in order for it to be chosen to accelerate the various motion estimation algorithms. Simulations have

been conducted to determine the average number of CPU cycles needed per block for the various algorithms incorporating the chosen method. These simulations have shown that the proposed scheme is capable of providing a substantial speedup for the various existing motion estimation algorithms.

6.2 Scope for Further Research

In the proposed vector-based fast block motion estimation algorithm, a method to utilize the correlation of neighbouring motion vectors to predict the current motion vector, such as the median prediction, can be developed. Such a study might provide a more accurate initial value for the motion vector in the vector-based fast block motion estimation algorithm, thus leading to a lower computational complexity.

The scheme of using long-term memory has been adopted in the video coding standard H.26L for the purpose of motion estimation, providing a coding gain of about 0.5-1 dB. It is worth conducting further research in the utilization of the proposed eight-bit partial sums in conjunction with long-term memory for motion estimation.

References

[1] J. K. Aggarwal, and N. Nandhahumar, "On the computation of motion from sequences of images—a review," *Proceedings of the IEEE*, pp. 917-935, 1988.

[2] O. Avaro, A. Elftheriadis, C. Herpel, G. Rajan, and L. Ward, "MPEG-4 systems: overview," *Multimedia Systems, Standards, and Networks*, New York: Marcel Dekker, pp. 331-365, 2000.

[3] A. Basso, M. R. Civanlar, and V. Balabanian, "Delivery and control of MPEG-4 content over IP networks," *Multimedia Systems, Standards, and Networks*, New York: Marcel Dekker, pp. 501-523, 2000.

[4] C.S. Beightler, D.T. Phillips and D.J. Wilde, *Foundations of Optimization*, 2ed. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[5] T. Berger, *Rate Distortion Theory*. Englewood Cliffs, NJ: Prentice Hall, 1971.

[6] Y. T. Chan, *Wavelet Basics*. Kluwer Academic Publishers, Norwell, MA, 1995.

[7] T. Chen, "Emerging standards for multimedia applications," *Multimedia Image and Video Processing*, CRC Press, pp. 1-18, 2000.

[8] T. Chen, G. J. Sullivan, and A. Puri, "H.263 (including H.263++) and other ITU-T video coding standards," *Multimedia Systems, Standards, and Networks*, New York: Marcel Dekker, pp. 55-85, 2000.

[9] M. C. Chen, and A. N. Willson Jr., "Rate-distortion optimal motion estimation algorithms for motion-compensated transform coding," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 147-158, April 1998.

[10] Kuo-Liang Chung,and Lung-Chun Chang, "A new predictive search area approach for fast block motion estimation," *IEEE Transactions on Image Processing*, vol. 12, no. 6, pp. 648-652, June 2003.

[11] R. J. Clark, *Transform Coding of Images*. London: Academic Press, 1985.

[12] C. J. Duanmu, M. O. Ahmad, and M. N. S. Swamy, "A fast three-step search algorithm by the utilization of multi-level vector partial sums," in *Proceedings of the 2003 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2003)*, Montreal, Canada, vol. 3, pp. 1981–1984, , May 2003.

[13] C. J. Duanmu, M. O. Ahmad, and M. N. S. Swamy, "A new lower bound for fast block motion estimation algorithms," in *Proceedings of the 2003 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE 2003)*, Montreal, Canada, vol. 3, pp. 1975-1980, May 2003.

[14] C. J. Duanmu, M. O. Ahmad, and M. N. S. Swamy, "A continuous tracking algorithm for long-term memory motion estimation," in *Proceedings of the 2003 IEEE International Symposium on Circuits and Systems (ISCAS 2003)*, Bangkok, Thailand, vol. 2, pp. II-356-II-359, May 2003.

[15] C. J. Duanmu, M. O. Ahmad, and M. N. S. Swamy, "8-bit partial sums of 16 luminance values for fast block motion estimation," in *Proceedings of the 2003 IEEE International Conference on Multimedia and Expo. (ICME 2003)*, Baltimore, U.S.A., vol. 1, pp. 689 – 692, July 2003.

[16] C. J. Duanmu, M. O. Ahmad, and M. N. S. Swamy, "Fast block motion estimation with eight-bit partial sums using SIMD architectures," *Submitted to IEEE Transactions on Circuits and Systems for Video Technology*, 2005.

[17] C. J. Duanmu, M. O. Ahmad, M. N. S. Swamy, and A. Shatnawi, "A vector based fast block motion estimation algorithm for implementation on SIMD architectures," in *Proceedings of the 2002 IEEE International Symposium on Circuits and Systems (ISCAS 2002)*, Phoenix, U.S.A, vol. 4 , pp. IV-337 - IV-340, May 2002.

[18] C. J. Duanmu, M. O. Ahmad, M. N. S. Swamy, and A. Shatnawi, "Optimization of the three-step search algorithm by exclusion of stationary macroblocks from the search process," in *Proceedings of the 9th International Conference on Electronics, Circuits and Systems (ICECS 2002)*, Dubrovnik, Croatia, vol. 3, pp. 1035 - 1038, Sept. 2002.

[19] D. E. Dudgeon, and R. M. Mersereau, *Multidimensional Digital Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1984.

[20] F. Dufaux, and F. Moscheni, "Motion estimation techniques for digital TV: A review and a new contribution," *Proceedings of the IEEE*, pp. 858-876, June 1995.

[21] X.Q. Gao, C.J. Duanmu, C.R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, pp. 501 - 504, March 2000.

[22] X.Q. Gao, C.J. Duanmu, C.R. Zou, and Z.Y. He, "Multi-level successive elimination algorithm for motion estimation in video coding," in *Proceedings of the 1999 IEEE International Symposium on Circuits and Systems (ISCAS 1999)*, Orlando, U.S.A., vol. 4, pp. 227 - 230, June 1999.

[23] B. Girod, "Motion compensation: Visual aspects, accuracy, and fundamental limits," *Motion Analysis and Image Sequence Processing*, Boston: Kluwer Academic Pusblishers, pp. 126-152, 1993.

[24] B. Girod, "Motion-compensating predication with fractional-pel accuracy," *IEEE Transactions on Communications*, pp. 604-612, April 1993.

[25] B. Girod, E. Steinbach, and N. Farber, "Comparison of the H.263 and H.261 video compression standards," *SPIE Standards and Common Interfaces for Video*, pp. 233-251, Oct. 1995.

[26] B. Grob, and C. E. Herndon, *Basic Television and Video Systems*, 6th ed. New York: McGraw Hill, 1999.

[27] H.-M. Hang, Y.-M. Chou, and S.-C. Cheng, "Motion estimation for video coding standards," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, pp. 113-136, Nov. 1997.

[28] Y. Hashimoto, M. Yamamoto, and T. Asaida, "Cameras and display systems," *Proceedings of the IEEE*, vol. 7, pp. 1032-1043, July 1995.

[29] B. G. Haskell, "Frame replenishment coding of television," *Image Transmission Techniques*, New York: Academic Press, 1979.

[30] B. G. Haskell, "Image and video coding: Emerging standards and beyond," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 814-817, Nov. 1998.

[31] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2*. New York: Chapman & Hall, 1997.

[32] C. Herpel, A. Elftheriadis, and G. Franceschini, "MPEG-4 systems: Elementary stream management and delivery," *Multimedia Systems, Standards, and Networks*, NewYork: Marcel Dekker, pp. 367-405, 2000.

[33] ITU-T Recommendation H.261, "Video codec for audiovisual services at px64 k bits/s," Mar. 1993.

[34] ITU-T Recommendation H.263, "Video coding for low bit rate communication," 1998.

[35] Joint Video Team, "Draft Text of Final Draft International Standard for Advanced Video Coding (ITU-T Rec. H.26L | ISO/IEC 14496-10 AVC)," ISO/IEC JTC1/SC 29/ WG 11 N5555, Pattaya, March 2003.

[36] Intel Co., IA-32 Intel Architecture Optimization Reference Manual, 2005 [Online]. Available: http:// developer.intel.com / design/ pentium4/ manuals/ index_new.htm#aorm

[37] Intel Co., IA-32 Intel Architecture Software Developer's Manual Volume 1: Basic Architecture, 2005 [Online]. Available: http:// developer.intel.com / design / pentium4 / manuals / index_new.htm # sdm_vol1, 2005

[38] Intel Co., IA-32 Intel Architecture Software Developer's Manual Volume 2A: Instruction Set Reference, A-M, 2005 [Online]. Available: http:// developer.intel.com / design / pentium4 / manuals / index_new.htm#sdm_vol2a

[39] Intel Co, IA-32 Intel Architecture Software Developer's Manual Volume 2B: Instruction Set Reference, L-Z, 2005 [Online]. Available: http:// developer.intel.com / design / pentium4 / manuals / index_new.htm#sdm_vol2b

[40] Intel Co., IA-32 Intel Architecture Software Developer's Manual Volume 3: System Programming Guide, 2005 [Online]. Available: http:// developer.intel.com / design / pentium4 / manuals / index_new.htm#sdm_vol3

[41] Intel Co., Introduction to Streaming SIMD Extensions, 2005 [Online]. Available: http:// developer.intel.com / software / products / college/ia32 / strmsimd / clikngo.htm

[42] Intel Co, Pentium Processor with MMX Technology, 2003 [Online]. Available: http:// support.intel.com/ design/archives/processors/mmx, June 2003

[43] A. K. Jain, *Fundamentals of Digital Image Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[44] A. K. Jain, "Image data compression: A review," *Proceedings of the IEEE*, vol. 69, pp. 345-389, Mar. 1981.

[45] J. R. Jain, and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Transactions on Communications*, pp. 1799-1808, Dec. 1981.

[46] N. S. Jayant, and P. Noll, *Digital Coding of Waveforms*. Englewood Cliffs, NJ: Prentice Hall, 1984.

[47] ISO/IEC, "IS 10918-1: Information technology—digital compression and coding of continuous-tone still images: Requirements and guidelines," 1990. (JPEG)

[48] S. Kappagantula and K. R. Rao, "Motion predictive interframe coding," *IEEE Transactions on Communications*, vol. 33, pp. 1011-1015, Sept. 1985.

[49] T. Koga, K. Iinuma, A. Hirano, and T. Ishi-guro, "Motion-compensated interframe coding for video conferencing," in *Proceedings of National Telecommunication Conference*, pp. C9.6.1-C9.6.5, New Orleans, USA, Nov. 1981.

[50] T. Komarek, and P. Pirsch, "Array architecture for block matching algorithms," *IEEE Transactions on Circuits and Systems*, vol. 36, pp. 269-277, Oct. 1989.

[51] O. Lee, and Y. Wang, "Motion compensated prediction using nodal based deformable block matching," *Journal of Visual Communications and Image Representation*, pp. 6: 26-34, March 1995.

[52] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing*, vol. 4, pp. 105-107, Jan. 1995.

[53] M. L. Liou, "Overview of the *kx*64 kbps video coding standard," *Communications of the ACM*, vol. 34, pp. 47-58, Apr. 1991.

[54] Lurng-Kuo Liu, and Ephraim Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 4, pp.419-422, August 1996.

[55] H. Lohscheller, "A subjectively adapted image communication system," *IEEE Transactions on Communications*, pp. 1316-1322, Dec. 1984.

[56] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, MPEG Video Compression Standard. Bonn, Germany: Chapman and Hall, 1996.

[57] ISO/IEC, "IS 11172: Information technology—coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s," 1993. (MPEG-1)

[58] ISO/IEC, "IS 13818-1: Information technology—generic coding of moving pictures and associated audio information: Systems," 1995. (MPEG-2 Systems)

[59] ISO/IEC, "IS 13818-2: Information technology—generic coding of moving pictures and associated audio information: Video," 1995. (MPEG-2 Video)

[60] ISO/IEC, "IS 13818-3: Information technology—generic coding of moving pictures and associated audio information: Audio," 1995. (MPEG-2 Audio)

[61] ISO/IEC, "IS 14496-1: Information technology—coding of audio-visual objects—part 1: Systems," 1999. (MPEG-4 Systems)

[62] ISO/IEC, "IS 14496-2: Information technology—coding of audio-visual objects—part 2: Visual," 1999. (MPEG-4 Video)

[63] ISO/IEC, "IS 14496-3: Information technology—coding of audio-visual objects—part 3: Audio," 2000. (MPEG-4 Audio)

[64] H. G. Musmann, P. Pirsch, and H.-J. Grallert, "Advances in picture coding," *Proceedings of the IEEE*, pp. 523-548, April 1985.

[65] A. N. Netravali, and B. G. Haskell, *Digital Pictures—Representation, Compression and Standards, 2nd ed.* New York: Plenum Press, 1995.

[66] Y. Ninomiya, and Y. Ohtsuka, "A motion-compensated interframe coding scheme for television pictures," *IEEE Transactions on Communications*, vol. 30, pp. 201-211, Jan. 1982.

[67] S. Okubo, "Reference model methodology—a tool for the collaborative creation of video coding standards," *Proceedings of the IEEE*, pp. 139-150, Feb. 1995.

[68] A. V. Oppenheim, and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs: Prentice Hall, 1989.

[69] M. T. Orchard and G. J. Sullivan, "Overlapped block motion compensation: An estimation-theoretic approach," *IEEE Transactions on Image Processing*, pp. 693-699, 1994.

[70] J. Ostermann and A. Puri, "Natural and synthetic video in MPEG-4," in *Proceedings of 1998 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 1998)*, Seattle, USA, pp. 3805-3809, Nov. 1998.

[71] W.B. Pennebaker and J.L. Mitchell, *JPEG: Still Image Data Compression Standard*. Van Nostrand Reinhold, New York, USA, 1993.

[72] P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI architecture for video compression—a survery," *Proceedings of the IEEE*, vol. 83, pp.220-246, Feb. 1995.

[73] L. M. Po and W. C. Ma, "A novel four-step algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, pp. 313-317, June 1996.

[74] A. Puri, "Video coding using the MPEG-2 compression standard," *SPIE Visual Communications and Image Processing*, pp. 1701-1712, Nov. 1993.

[75] A. Puri, H.-M. Hang, and D. Schilling, "An efficient block-matching algorithm for motion compensated coding," in *Proceedings of 1987 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 1987)*, vol. 2, pp. 1063-1066, April 1987.

[76] A. Puri, and A. Wong, "Spatial domain resolution scalable video coding," *SPIE Visual Communications and Image Processing*, pp. 718-729, Nov. 1993.

[77] A. Puri, L. Yan, and B.G. Haskell, "Temporal resolution scalable video coding," in *Proceedings of 1994 IEEE International Conference on Image Processing (ICIP 1994)*, Austin, USA, pp. 947-951, Nov. 1994.

[78] K. R. Rao and P. Yip, *Discrete Cosine Transform—Algorithms, Advantages, Applications*. Academic Press Inc., London, 1990.

[79] O. Rioul and M. Vetterli, "Wavelet and signal processing," *IEEE Signal Processing Magazine*, vol. 4, pp.14-38, Oct. 1991.

[80] K. Sayood, *Introduction to Data Compression*. San Francisco: Morgan Kaufmann, 1996.

[81] G. M. Schuster, and A. K. Katsaggelos, *Rate-distortion Based Video Compression*. Boston: Kluwer Academic Publishers, 1997.

[82] R. Srinivasan and K. R. Rao, "Predictive coding based on efficient motion estimation," *IEEE Transactions on Communications*, vol. 33, pp. 888-896, Aug. 1985.

[83] C. Stiller, and J. Konrad, "Estimating motion in image sequences," *IEEE Signal Processing Magazine*, pp. 70-91, July 1999.

[84] G. Strang and T. Nguyen, *Wavelets and Filter Banks*. Wellesley-Cambridge Press, Wellesley, MA, 1996 [Online]. Available: http://www-math.mit.edu/~gs/books/wfb.html

[85] G. J. Sullivan, and T. Wiegand, "Rate-distgortion optimization for video compression," *IEEE Signal Processing Magazine*, vol. 15, pp. 74-90, Nov. 1998.

[86] Sun Co, VIS Instruction Set, June 2003 [Online]. Available: http:// www. Sun.com/ processors/ vis

[87] A. M. Tekalp, *Digital Video Processing*. Upper Saddle River, NJ: Prentice Hall, 1995.

[88] A. M. Tekalp, and J. Ostermann, "Face and 2-D mesh animation in MPEG-4," *Signal Processing : Image Communications,* pp. 387-421, Jan. 2000.

[89] J. Y. Tham, S. Ranganath, M. Ranganath, and A. Al. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, pp. 369-377, Aug. 1998.

[90] A. Vetro, H. Sun, and Y. Wang, "MPEG-4 rate control for multiple video objects," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, pp. 186-199, Feb. 1999.

[91] M. Vetterli and J. Kovacevic, *Wavelets and Subband Coding*. Englewood Cliffs, NJ, Prentice Hall, 1995.

[92] Y. Wang, and O. Lee, "Active mesh—a feature seeking and tracking image sequence representation scheme," *IEEE Transactions on Image Processing*, pp. 610-624, Sept. 1994.

[93] Y. Wang and J. Ostermann, "Evaluation of mesh-based motion estimation in H.263 like coders," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 243-252, June 1998.

[94] Y. Wang, J. Ostermann, and Y. Q. Zhang, *Video Processing and Communications*. Englewood Cliffs, NJ: Prentice Hall, 2003.

[95] J. Watkinson, *The Art of Digital Video, 2nd ed.* Oxford: Focal Press, 1994.

[96] J. C. Whitaker, *DTV Handbook: The Revolution in Digital Video, 3rd ed.* New York: McGraw Hill, 2001.

[97] J. C. Whitaker, and K. B. Benson, *Standard Handbook of Video and Television Engineering, 3rd ed.* New York: McGraw-Hill, 2000.

[98] T. Wiegand, M. Lightstone, D. Mukherjee, T. Campell, and S. K. Mitra, "Rate-distortion optimized mode selection for very low bit rate video coding and the emerging H.263 standard," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 182-190, April 1996.

[99] T. Wiegand, X. Zhang, and B. Girod, "Long-term memory motion-compensated prediction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 9, pp. 70-84, no. 1, February, 1999.

[100] S. Zhu and K. K. Ma, "A new diamond search algorithm for fast block matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, pp. 287–290, Feb. 2000.

Appendix A

Block Motion Estimation Algorithms

A.1 Full-Search Algorithm (FSA) [20]

The FSA is summarized in Table A.1. In this table, $CS(l)$, $\vec{V}_{op}(l)$, and $Min(l)$ are, respectively, the set of candidate motion vectors, the optimum motion vector and its corresponding MAD value, computed so far in the FSA. The current motion vector is denoted by $\vec{V}_{curr}$ and P is as defined in (2.3).

Table A.1 Full-search algorithm

1) *Initialization* $l = 1$ $CS(l) = \{(0,0)\}$ $\vec{V}_{op}(l) = (0,0)$ $Min(l) = MAD(\vec{V}_{op}(l))$

2) *Search process*
for (every $\vec{V}_{curr} \in P$)
{
$CS(l+1) = CS(l) \bigcup \{\vec{V}_{curr}\}$
$Min(l+1) = Min(l)$
$\vec{V}_{op}(l+1) = \vec{V}_{op}(l)$
if ($MAD(\vec{V}_{curr}) < Min(l+1)$)
{
$Min(l+1) = MAD(\vec{V}_{curr})$
$\vec{V}_{op}(l+1) = \vec{V}_{curr}$
}
$l = l+1$
}

3) *Output* $\vec{V}_{op}(S \times S)$ is the selected motion vector in the full-search algorithm

A.2 Two-Dimensional Logarithmic Search Algorithm (2DLSA) [45]

For the search length $m > 0$, we define

$$N(m) = \{(i, j) \mid -m \le i \le m, \quad -m \le j \le m\} \tag{A.1}$$

and

$$M(m) = \{(0,0), \ (m,0), \ (0,m), \ (-m,0), \ (0,-m)\} \tag{A.2}$$

The search pattern given by (A.2) is employed until the search length is unity, after which the search pattern given by (A.1) is employed. The algorithm is summarized in Table A.2.

Table A.2 2-D logarithmic search algorithm

1) *Initialization* $MAD(i, j) = \infty$ for $(i, j) \notin N(w)$, where w is defined in Section 2.1. $n' = \lfloor \log_2 w \rfloor$, where $\lfloor x \rfloor$ stands for the largest integer less than x $n = \max(2, 2^{n'} - 1)$ $q = l = 0$
2) $M'(n) \longleftarrow M(n)$
3) Find $(i, j) \in M'(n)$ such that $MAD(i + q, j + l)$ is minimum. If $i = 0$ and $j = 0$, go to 5); otherwise go to 4)
4) $q \leftarrow q + i$, $l \leftarrow l + j$; go to 3)
5) $n \leftarrow \lfloor n/2 \rfloor$; if $n = 1$, go to 6); otherwise, go to 2)
6) find $(i, j) \in N(1)$ such that $MAD(i + q, j + l)$ is minimum. $q \leftarrow q + i$; $l \leftarrow l + j$. (q, l) is the optimum motion vector in the algorithm.

A.3 Orthogonal Search Algorithm (OSA) [75]

This algorithm recursively reduces the region of uncertainty (RUC), where it assumes the optimum motion vector lies. The search patterns shown in Figure 2.3 are employed in this algorithm. The algorithm is summarized in Table A.3.

Table A.3 Orthogonal search algorithm

1) *Initialization* Search region, $SR = (2w+1) \times (2w+1)$. Step number, $i = 1$ Initial step size $l = \lceil w/2 \rceil$, where $\lceil x \rceil$ is the least integer larger than or equal to x
2) *Step 1* Three search locations are placed horizontally in the center of SR. The distance between every two neighboring search locations equals to the step size as shown in Figure 2.3(b). The location with the minimum value of MAD is selected as the center for the next step. Step number $i \leftarrow (i+1)$
3) *Vertical step* Two more search locations are placed vertically around the location with the minimum value of MAD from the previous step. The distance between every two neighboring search locations equals to the step size as shown in Figure 2.3(a). The location with the minimum value of MAD is taken as the center for the next step.
4) *Stopping rule* The remaining region of uncertainty (RUC) now has an area of $4(l-1) \times (l-1)$. If $l = 1$, stop; Otherwise, $l \leftarrow \lceil l/2 \rceil$, $i \leftarrow (i+1)$, and continue.
5) *Horizontal step* Two more search locations are placed horizontally around the location with the minimum value of MAD from the vertical step. The location with the minimum value of MAD is taken as the center for the next step. $i \leftarrow (i+1)$ and go back to 3).

A.4 One at a Time Search Algorithm (OATSA) [82]

The search process of this algorithm is first carried out in the horizontal (j-th) direction to find a local minimum of MAD in this direction. Then the search process is carried out in the vertical (i-th) direction, starting from the location of local minimum value of MAD in the j-th direction. This algorithm is summarized in Table A.4.

Table A.4 One at a time search algorithm

1) *Initialization* $MAD(i,j) = \infty$ for $(i,j) \notin N(w)$, where $N(w)$ is defined in (A.1).
2) *Step 1* Compute $MAD(i,j)$, $MAD(i,j+1)$, and $MAD(i,j-1)$, a local minimum value of MAD is found. If $MAD(i,j+1)$ turns out to be the minimum, $MAD(i,j+2)$ is also computed and the minimum of $MAD(i,j)$, $MAD(i,j+1)$, and $MAD(i,j+2)$ is found. Proceed in this manner until the minimum is closeted between two higher values, i.e. $MAD(i,j+l)$ $(-w \le l \le w)$ is the minimum among $MAD(i,j+l)$, $MAD(i,j+l-1)$, and $MAD(i,j+l+1)$. The j-direction search stops and a minimum in this direction is obtained.
3) *Step 2* The search continues now in the i-direction, similar to step 1. Computing $MAD(i,j+l)$, $MAD(i-1,j+l)$, and $MAD(i+1,j+l)$, a minimum is found. If $MAD(i+1,j+l)$ turns out to be the minimum, $MAD(i+2,j+l)$ is also computed and the minimum of $MAD(i,j+l)$, $MAD(i+1,j+l)$, and $MAD(i+2,j+l)$ is found. Proceed in this manner until the minimum is closeted between two higher values. i.e. $MAD(i+q,j+l)$ $(-w \le q \le w)$ is the minimum among $MAD(i+q,j+l)$, $MAD(i+q-1,j+l)$, and $MAD(i+q+1,j+l)$. The i-direction search stops and we obtain a minimum in this direction.

A.5 Conjugate Direction Search Algorithm (CDSA) [82]

In CDSA, OATSA is first carried out to find the location of the local minimum of MAD. Then, the search process is continued in the conjugate direction, which connects the location (0,0) and the position of the local minimum of OATSA. The position of the local minimum along this conjugate direction is selected as the location of the optimum motion vector in CDSA. This algorithm is summarized in Table A.5.

Table A.5 Conjugate direction search algorithm

1) Step 1 One at a time search algorithm is first carried out and (q,l), the location of the minimum value of the MAD in the one at a time search algorithm, is found.
2) Step 2 The direction of search now is the vector connecting the location (0,0) and (q,l). *Case 1:* If $MAD(3,3)$ is the minimum from 1), compute $MAD(2,2)$ and $MAD(4,4)$. Proceed in the direction connecting (0,0) and (3,3) until the minimum is closeted between two higher values. *Case 2:* A problem occurs if the i-direction component q is not equal to the j-direction component l. This could be like $MAD(3,2)$ obtained as a minimum from 1). In such a case, the nearest grid locations on the direction joining (0,0) and (3,2) are chosen. These are (2,1) and (4,3). Proceed in this direction until a local minimum is closeted between two higher values.

A.6 Three-Step Search Algorithm (3SSA) [49]

For the search length $m > 0$, we define

$$T(m) = M(m) \cup \{(-m,-m),(-m,m),(m,m),(m,-m)\} \tag{A.3}$$

where $M(m)$ is given by (A.2). The search pattern of $T(m)$ is employed in the 3SSA. The initial search length in the 3SSA is approximately one-fourth of the length of the search window. Then, the search length in each step of 3SSA is one-half of the search length in the previous step until the search length is unity. The initial search center is at the center of the search window. The center of the selected search locations in every other step is the location corresponding to the minimum value of the MAD in the previous step. This algorithm is summarized in Table A.6.

Table A.6 Three-step search algorithm

1) Initialization $MAD(i,j) = \infty$, for $(i,j) \notin N(w)$, where $N(w)$ is defined in (A.1). $q = l = 0$ $n = \lceil w/2 \rceil$
2) $T'(n) \leftarrow T(n)$
3) Find $(i,j) \in T'(n)$ such that $MAD(i+q, j+l)$ is minimum
4) $q \leftarrow q+i$, $l \leftarrow l+j$, $n \leftarrow \lceil n/2 \rceil$ if $n = 1$ go to 5); otherwise go to 2)
5) Find $(i,j) \in T(1)$ such that $MAD(i+q, j+l)$ is minimum
6) $q \leftarrow q+i$, $l \leftarrow l+j$. (q,l) is the optimum motion vector of the three-step search algorithm.

A.7 Four-Step Search Algorithm (4SSA) [73]

The search pattern of $T(2)$ given by (A.3) is recursively utilized until the location of the minimum value of MAD lies at the center of the search pattern or the search process reaches the boundary of the search window. The initial search center is at the center of the search window. The center of the selected search locations in every other step is the location with the minimum value of MAD in the previous step. In the final search step, the search pattern of $T(1)$ given by (A.3) is employed. This algorithm is summarized in Table A.7.

Table A.7 Four-step search algorithm

1) *Initialization* $MAD(i, j) = \infty$, for $(i, j) \notin N(w)$, where $N(m)$ is defined in (A.1). $q = l = 0$
2) Find $(i, j) \in T(2)$ such that $MAD(i+q, j+l)$ is minimum, where $T(m)$ is defined in (A.3).
3) $q \leftarrow q+i$, $l \leftarrow l+j$ if $i = 0$ and $j = 0$ go to 4); otherwise go to 2)
4) Find $(i, j) \in T(1)$ such that $MAD(i+q, j+l)$ is minimum
5) $q \leftarrow q+i$, $l \leftarrow l+j$. (q,l) is the optimum motion vector of the four-step search algorithm.

A.8 Unrestrictive Center-Biased Diamond Search Algorithm (UDSA) [89, 100]

A large diamond search pattern (LDSP) as shown in Figure 2.6 is recursively utilized in the UDSA until the location of the minimum value of the MAD lies at the center of the LDSP, or the search process reaches the boundary of the search window. In the final search step, a small diamond search pattern (SDSP) as shown in Figure 2.7 is employed. This algorithm is summarized in Table A.8.

Table A.8 Unrestricted center-biased diamond search algorithm

1) The initial LDSP is located at the center of the search window. At each of the nine search locations of the LDSP, block matching computation of the MAD is carried out to find the location with the minimum block matching distortion among these locations. If the location with the minimum block matching distortion occurs at the center of the LDSP, go to 3); otherwise, go to 2).
2) The search location with the minimum block matching distortion is repositioned as the center search location to form a new LDSP. If the minimum block matching distortion is at the center position, go to 3); otherwise, recursively repeat this step.
3) Switch the search pattern from LDSP to SDSP. The search location with the minimum block matching distortion corresponds to the final solution of the motion vector of the diamond search algorithm.

A.9 Predicative Search Area Algorithm (PSAA) [10]

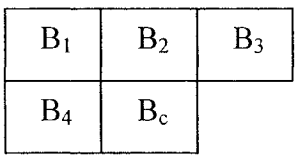


Figure A.1 Current block B_c and its four neighboring blocks B_1, B_2, B_3, and B_4

There are four neighboring blocks, B_1, B_2, B_3, and B_4, of the current block B_c, as shown in Figure A.1. We define the motion vector corresponding to the block B_i as $MV_i = (MVX_i, MVY_i)$, where $1 \le i \le 4$, as shown in Figure 2.8. Each sub-area S_i can be expressed as

$$S_i = \{(x, y) \mid MVX_i - L \le x \le MVX_i + L,\ MVY_i - L \le y \le MVY_i + L\} \qquad \text{(A.4)}$$

where L is selected as 2 in the PSAA. The search area SA in the predicative search area algorithm is the union of the four sub-areas, i.e.,

$$SA = S_1 \cup S_2 \cup S_3 \cup S_4 \qquad \text{(A.5)}$$

as shown in Figure 2.8. All the candidate motion vectors in the SA are exhaustively searched in the PSAA.

From (A.4) and (A.5), it can be seen that in the worst case, MAD needs to be computed 100 times in order to find the optimum motion vector in the PSAA. In the best case, however, only 25 such computations need to be carried out.

A.10 Block-Based Gradient Descent Search Algorithm (BGDSA) [54]

The search pattern of $T(1)$ given by (A.3) is recursively utilized in the BGDSA. When the location of the minimum value of the MAD lies at the center of the search pattern or the search process reaches the boundary of the search window, the search process in the BGDSA is stopped. The initial search center is at the center of the search window. The center of the selected search locations in every other step is the location with the minimum value of the MAD in the previous step. This algorithm is summarized in Table A.9.

Table A.9 Block-based gradient descent search algorithm

1) *Initialization* $MAD(i, j) = \infty$, for $(i, j) \notin N(w)$ $q = l = 0$
2) Find $(i, j) \in T(1)$ such that $MAD(i + q, j + l)$ is minimum, where $T(m)$ is defined in (A.3).
3) $q \leftarrow q + i$, $l \leftarrow l + j$ if $i = 0$ and $j = 0$ go to 4); otherwise go to 2)
4) $q \leftarrow q + i$, $l \leftarrow l + j$. (q,l) is the optimum motion vector of the BGDSA.

A.11 Selective Elimination Algorithm (SEA) [52]

The partial sums in the SEA are defined as the summation of the luminance values in a 16×16 block and can be expressed as

$$\begin{cases} PS_c(x,y) = \sum_{n=0}^{15}\sum_{m=0}^{15} I_c(x+m, y+n) \\ PS_r(x,y) = \sum_{n=0}^{15}\sum_{m=0}^{15} I_r(x+m, y+n) \end{cases} \tag{A.6}$$

where $I_c(x,y)$ and $I_r(x,y)$ are as defined in Section 2.1. It has been shown that for the SEA [52],

$$MAD(\vec{V}) \geq LB_{SEA}(\vec{V}) \tag{A.7}$$

where $$LB_{SEA}(\vec{V}) = \left| PS_c(x_0, y_0) - PS_r(x_0 + V_x, y_0 + V_y) \right| \tag{A.8}$$

This algorithm is summarized in Table A.10.

Table A.10 Selective elimination algorithm

1) *Initialization* $l = 1$ $CS(l) = \{(0,0)\}$; $\vec{V}_{op}(l) = (0,0)$; $Min(l) = MAD(\vec{V}_{op}(l))$

2) *Search process*
for (every $\vec{V}_{curr} \in P$)
{
$CS(l+1) = CS(l) \cup \{\vec{V}_{curr}\}$; $Min(l+1) = Min(l)$; $\vec{V}_{op}(l+1) = \vec{V}_{op}(l)$
if ($LB_{SEA}(\vec{V}_{curr}) < Min(l+1)$)
{
if ($MAD(\vec{V}_{curr}) < Min(l+1)$)
{
$Min(l+1) = MAD(\vec{V}_{curr})$; $\vec{V}_{op}(l+1) = \vec{V}_{curr}$
}
}
$l = l+1$
}

3) *Output* $\vec{V}_{op}(S \times S)$ is the optimum motion vector of the SEA

Appendix B

Computational Complexity of Block Motion Estimation Algorithms

B.1 Computational Complexity of the FSA and the Multi-Step Search Algorithms

In the case of the FSA or a multi-step search algorithm, $\vec{V}_{curr}$ and $Min(l)$ represent the current candidate motion vector and the minimum value of the MAD computed so far, respectively. At every search location in the algorithm, a calculation of $MAD(\vec{V}_{curr})$ and a comparison between $MAD(\vec{V}_{curr})$ and $Min(l)$ are carried out, where $MAD(\vec{V}_{curr})$ is given by (2.1). From (2.1), it can be seen that the computation of the $MAD(\vec{V}_{curr})$ for a macroblock involves 256 subtractions, 256 operations to calculate the absolute values and 255 additions. Thus, the computational complexity per pixel for the calculation of the $MAD(\vec{V}_{curr})$ is

$$C_{MAD} = (256 + 256 \times 2 + 255)/256 \tag{B.1}$$

where the division by 256 is due to the fact that one macroblock contains 256 pixels. The computational complexity per pixel for the comparison between $MAD(\vec{V}_{curr})$ and $Min(l)$ is

$$C_{C1} = 2/256 \tag{B.2}$$

Hence, the computational complexity of the FSA is

$$C(FSA) = (C_{MAD} + C_{C1})\eta_{MAD}(FSA) \tag{B.3}$$

where $\eta_{MAD}(x)$ and $C(x)$ are, respectively, the number of times $MAD(\vec{V})$ needs to be calculated per macroblock and the computational complexity per pixel of the block

motion estimation algorithm x in order to complete the estimation process. Now, for the FSA

$$\eta_{MAD}(FSA) = S^2 \tag{B.4}$$

where $(S \times S)$ is the size of the search window. From (B.1), (B.2), (B.3), and (B.4), $C(FSA)$ can be expressed as

$$C(FSA) = \tfrac{1025}{256} S^2 \tag{B.5}$$

The computational complexity of a multi-step search algorithm x is given by

$$C(x) = (C_{MAD} + C_{C1})\eta_{MAD}(x) \tag{B.6}$$

Specifically, we have

$$C(2DLSA) = (C_{MAD} + C_{C1})\eta_{MAD}(2DLSA) = \tfrac{1025}{256}\eta_{MAD}(2DLSA) \tag{B.7}$$

$$C(OSA) = (C_{MAD} + C_{C1})\eta_{MAD}(OSA) = \tfrac{1025}{256}\eta_{MAD}(OSA) \tag{B.8}$$

$$C(OATSA) = (C_{MAD} + C_{C1})\eta_{MAD}(OATSA) = \tfrac{1025}{256}\eta_{MAD}(OATSA) \tag{B.9}$$

$$C(CDSA) = (C_{MAD} + C_{C1})\eta_{MAD}(CDSA) = \tfrac{1025}{256}\eta_{MAD}(CDSA) \tag{B.10}$$

$$C(3SSA) = (C_{MAD} + C_{C1})\eta_{MAD}(3SSA) = \tfrac{1025}{256}\eta_{MAD}(3SSA) \tag{B.11}$$

$$C(4SSA) = (C_{MAD} + C_{C1})\eta_{MAD}(4SSA) = \tfrac{1025}{256}\eta_{MAD}(4SSA) \tag{B.12}$$

$$C(UDSA) = (C_{MAD} + C_{C1})\eta_{MAD}(UDSA) = \tfrac{1025}{256}\eta_{MAD}(UDSA) \tag{B.13}$$

$$C(PSAA) = (C_{MAD} + C_{C1})\eta_{MAD}(PASA) = \tfrac{1025}{256}\eta_{MAD}(PSAA) \tag{B.14}$$

$$C(BGDSA) = (C_{MAD} + C_{C1})\eta_{MAD}(BGDSA) = \tfrac{1025}{256}\eta_{MAD}(BGDSA) \tag{B.15}$$

where, as mentioned earlier, 2DLSA, OSA, OATSA, CDSA, 3SSA, 4SSA, UDSA, PSAA, and BGDSA respectively denote the 2-D logarithmic search, one at a time search, conjugate direction search, three-step search, four-step search, unrestricted center-biased

diamond search, predictive area search, and block-based gradient descent search algorithms.

B.2 Computational Complexity of the SEA

For the SEA (Section A.11), at every search location inside the search window, a calculation of $LB_{SEA}(\vec{V}_{curr})$ and a comparison between $LB_{SEA}(\vec{V}_{curr})$ and $Min(l+1)$ are carried out, where $LB_{SEA}(\vec{V}_{curr})$ is given by (A.8). From (A.8), it can be seen that the computation of $LB_{SEA}(\vec{V}_{curr})$ for a macroblock involves one subtraction and one operation to calculate the absolute value. Thus, the computational complexity per pixel of $LB_{SEA}(\vec{V}_{curr})$ is

$$C_{LB} = (1+2\times 1)/256 \tag{B.16}$$

The computational complexity per pixel for the comparison between $LB_{SEA}(\vec{V}_{curr})$ and $Min(l+1)$ is

$$C_{C2} = \tfrac{2\times 1}{256} \tag{B.17}$$

Since there are $S\times S$ locations inside the search window, the total computational complexity per pixel required to calculate $LB_{SEA}(\vec{V}_{curr})$ and to compare it with $Min(l+1)$ in the SEA is

$$C_{SEA1} = (C_{LB} + C_{C2})S^2 = \tfrac{5}{256}\times 961 \tag{B.18}$$

Define

$$SC_c(x,y) = \sum_{m=0}^{15} I_c(x+m,y) \tag{B.19}$$

For the current frame of size $W \times H$, the partial sums $PS_c(x,y)$ ($0 \le x \le H-16$, $0 \le y \le W-16$) given by (A.6) are computed as follows.

Step 1) For the 0-th row, calculate all the $SC_c(0,j) = \sum_{m=0}^{15} I_c(m,j)$, $0 \le j \le W-1$. These computations require $15W$ operations, since each computation of $SC_c(0,j)$ ($0 \le j \le W-1$) needs 15 operations.

Step 2) For every i-th row ($1 \le i \le H-16$), calculate all the $SC_c(i,j) = SC_c(i-1,j) - I_c(i-1,j) + I_c(i+15,j)$, $0 \le j \le W-1$. For each i-th row ($1 \le i \le H-16$), these computations require $2W$ operations, since each computation of $SC_c(i,j)$ ($0 \le j \le W-1$) needs 2 operations. Since there are $H-16$ such rows, the total number of operations required for this step is $2W(H-16)$.

Step 3) For the 0-th column, calculate all the $PS_c(i,0) = \sum_{j=0}^{15} SC_c(i,j)$, $0 \le i \le H-16$. These computations require $15(H-15)$ operations, since each computation of $PS_c(i,0)$ ($0 \le i \le H-16$) requires 15 operations.

Step 4) For every j-th column ($1 \le j \le W-16$), calculate all the $PS_c(i,j) = PS_c(i,j-1) - SC_c(i,j-1) + SC_c(i,j+15)$, $0 \le i \le H-16$. For each j-th column ($1 \le j \le W-16$), these computations require $2(H-15)$ operations, since each computation of $PS_c(i,j)$ ($0 \le i \le H-16$) needs 2 operations. Since there are $W-16$ such columns, the total number of operations required for this step is $2(H-15)(W-16)$.

From these four steps, we see that the total number of operations required to calculate the partial sums $PS_c(x,y)$ is

$$T_c = 15W + 2W(H-16) + 15(H-15) + 2(H-15)(W-16) \tag{B.20}$$

Thus, the total computational complexity per pixel required to compute all the partial sums $PS_c(x,y)$ in the SEA is

$$C_{SEA2} = \frac{T_c}{WH} = 4 - \frac{17}{W} - \frac{47}{H} + \frac{255}{WH} \tag{B.21}$$

since a video frame consists of $W \times H$ pixels. Once $PS_c(x,y)$ is computed for all (x,y), they are saved in the memory. Hence, $PS_r(x,y)$ for any value of (x,y) can be retrieved from the memory with no computation required.

When $LB_{SEA}(\vec{V}_{curr}) < Min(l+1)$, $MAD(\vec{V}_{curr})$ is computed and a comparison between $MAD(\vec{V}_{curr})$ and $Min(l+1)$ is carried out (see Table A. 10). The computational complexity per pixel for the calculation of $MAD(\vec{V}_{curr})$ and its comparison with $Min(l+1)$ is

$$C_{MADC} = C_{MAD} + C_{C1} \tag{B.22}$$

Since the average number of times required to calculate $MAD(\vec{V}_{curr})$ and compare it with $Min(l+1)$ is $\eta_{MAD}(SEA)$, the total computational complexity per pixel for the computation and comparison is given by

$$C_{SEA3} = (C_{MAD} + C_{C1})\eta_{MAD}(SEA) \tag{B.23}$$

From the above discussion, we see that the computational complexity of the SEA consists of C_{SEA1}, C_{SEA2}, and C_{SEA3}. Hence, using (B.18), (B.21), and (B.23), the total computational complexity per pixel of the SEA is obtained as

$$C(SEA) = \tfrac{5}{256} \times 961 + \tfrac{1025}{256}\eta_{MAD}(SEA) + (4 - \tfrac{17}{W} - \tfrac{47}{H} + \tfrac{255}{WH}) \tag{B.24}$$