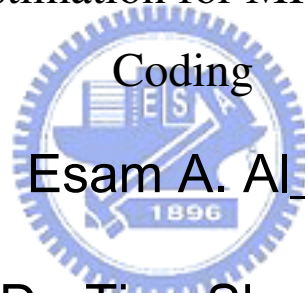


國立交通大學
電子工程學系電子研究所
博士論文

用於MPEG-4 與H.264視訊編碼的移動估測設計

Design of Motion Estimation for MPEG-4 and H.264 Video



研究生：Esam A. Al-Qaralleh

指導教授：Dr. Tian-Sheuan Chang

Dr. Chein-Wei Jen

中華民國九十 5 年 6 月

用於MPEG-4 與H.264視訊編碼的移動估測設計

Design of Motion Estimation for MPEG-4 and H.264 Video Coding

研 究 生 : Esam A. Al_Qaralleh

指導教授 : Dr. Tian-Sheuan Chang

Dr. Chein-Wei Jen

國 立 交 通 大 學

電子工程學系電子研究所

博 士 論 文

A Dissertation

Submitted to department of Electronic Engineering and Institute of Electronics

College of Electrical Engineering

National Chiao Tung University

In Partial Fulfillment of the requirements for the Degree of Doctor of

Philosophy

In

Electronic Engineering

July 2006

Hsinchu, Taiwan, Republic of China

中華民國九十 5 年 6 月

國立交通大學

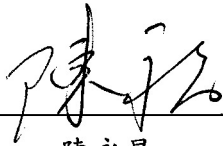
論文口試委員會審定書

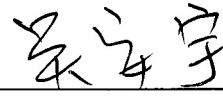
本校電子工程學系電子研究所博士班 Esam A. Al Qaralleh 君

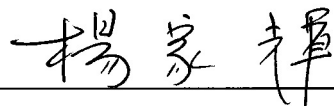
所提論文 Design of Motion Estimation for MPEG-4 and H.264 Video Coding


合於博士資格標準、業經本委員會評審認可。


口試委員：

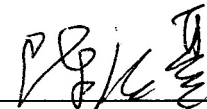

陳永昌 教授



吳安宇 教授


楊家輝 教授

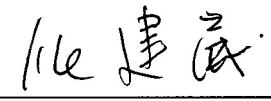

李鎮宜 教授



任建葳 教授


陳紹基 教授


張添烜 教授

指導教授：


任建葳 教授


張添烜 教授

所長：


陳紹基 教授

系主任：



李鎮宜 教授

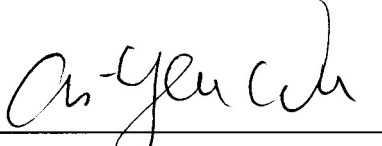
中華民國 95 年 7 月 19 日

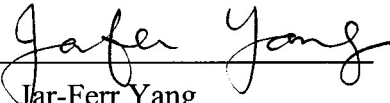
Department of Electronics Engineering
& Institute of Electronics
National Chiao Tung University
Hsinchu, Taiwan, R.O.C.

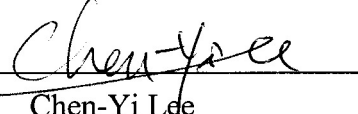
Date :19/7/2006

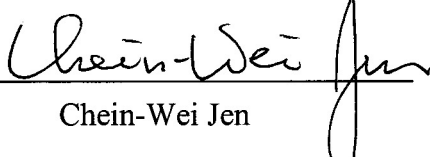
We have carefully read the dissertation entitled Design of Motion Estimation for MPEG-4 and H.264 Video Coding. submitted by Esam A. Al_Qaralleh in partial fulfillment of the requirements of the degree of DOCTOR OF PHILOSOPHY and recommend its acceptance.

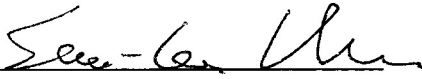

Yung-Chang Chen

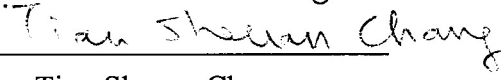

An-Yeu (Andy) Wu

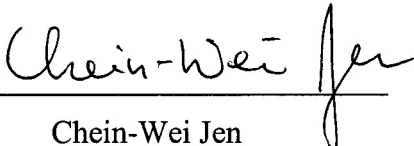
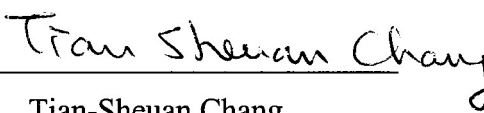

Jar-Ferr Yang

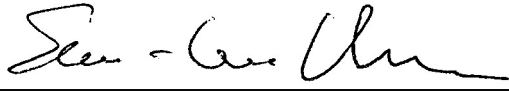

Chen-Yi Lee

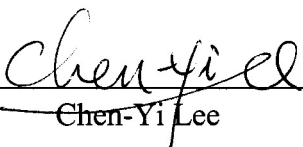

Chein-Wei Jen


Sau-Gee Chen


Tian-Sheuan Chang

Thesis Advisor : 
Chein-Wei Jen 
Tian-Sheuan Chang

Director : 
Sau-Gee Chen

Chairman : 
Chen-Yi Lee

ACKNOWLEDGE

I would like to thank professor Tian-Sheuan Chang, for his supervision and support during this work. He is so kind and generous. This work won't be possible without his fruitful and valuable comments and directions throughout this work. Thank you.

To my first teacher professor Chein-Wei Jen, my deep appreciation and thankful for his supervision in the first years of my stay in Taiwan.

To all my friends and lab members who were always helpful and supportive, their friendly behavior made me feel comfortable during my stay in Taiwan which is my second home beside Jordan.

Esam A. Al_Qaralleh

Table of Contents

Chapter 1 Introduction	1
1.1 Overview of Video Coding	1
1.2 Cores of Digital Video Technology	2
1.2.1 MPEG-4 Overview	3
1.2.2 H.264 Overview	6
1.3 Motivation and Contributions	8
1.4 Dissertation Organization	10
Chapter 2 A Fast Binary Motion Estimation Algorithm and its Architecture Design	11
2.1 Overview of the MPEG-4 Video System	11
2.1.1 Video Object and Video Object Plane	11
2.1.2 MPEG-4 Binary Shape Coding	12
2.2 Motion Estimation for MPEG-4 Video	16
2.2.1 Block-matching Algorithm	17
2.2.2 Search Range and Search Points	18
2.3 Analysis of Binary Motion Estimation	19
2.4 Previous Work on Binary Motion Estimation	20
2.5 Motivation for Fast Binary Motion Estimation.....	22
2.6 The Proposed Algorithm.....	24
2.6.1 Software simulation results and analysis.....	24
2.6.2 Block-matching Algorithm	25
2.6.3 Consideration for the Classification and Matching Methods	29
2.7 The Architecture Design.	30
2.7.1 Architecture Design	30
2.7.2 PE Design	31

2.7.3 Data reuse and data flow	32
2.7.4 Experiments results	34
2.7.5 Comparisons.....	36
2.8 Summary	37
<i>Chapter 3 Texture Motion Estimation for MPEG-4 and H.264.</i>	<i>39</i>
3.1 Introduction to Block Matching.....	39
3.2 Exploration of Algorithms.....	41
3.2.1 Proposed Algorithm-Software Approach	42
3.3 Exploration of Architectures	53
3.3.1 ME Architectures – an overview	54
3.3.2 Proposed Architecture	54
3.4 Summary	60
<i>Chapter 4 Data Reuse Exploration Between Vertical Adjacent Macro Blocks.....</i>	<i>62</i>
4.1 Introduction	62
4.2 Data reuse in the motion estimation	63
4.3 Data reuse and parallel processing of vertical blocks.	64
4.3.1 Proposed data reuse scheme	67
4.3.2 Architecture Design	70
4.3.3 Design comparisons	77
4.4 Summary	78
<i>Chapter 5 Future Work and Conclusion</i>	<i>79</i>
5.1 Contributions Summary	79
5.2 Future Work	82
5.2.1 Fractional Motion Estimation	82
5.2.2 Mode Selection and Hilbert Transform.....	87



5.2.3 Binary Motion Estimation.....	89
5.3 Summary	91
<i>References</i>	92



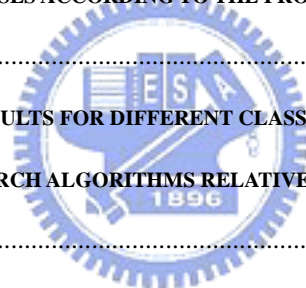
List of Figures

FIG. 1.1 GENERAL STRUCTURE OF MPEG-4 VOP ENCODER	4
FIG.1.2 THE BLOCK DIAGRAM OF H.264 ALGORITHM.....	7
FIG. 1.3 THE RELATIONS BETWEEN THE WORKS PRESENTED IN THIS THESIS.....	10
FIG. 1.4 CONTRIBUTIONS OF THE WORKS PRESENTED IN THIS THESIS.	10
FIG.2.1 VIDEO OBJECT PLANE (VOP)	13
FIG.2.2 GENERAL STRUCTURE OF MPEG-4 VOP ENCODER.....	13
FIG. 2.3 BLOCK MATCHING OF MOTION ESTIMATION FOR SHAPE	14
FIG.2.4 BLOCK DIAGRAM OF MPEG-4 BINARY SHAPE ENCODING SYSTEM	16
FIG. 2.5 FRACTIONAL-PEL SEARCH POINTS	19
FIG. 2.6 PSEUDO CODE FOR FULL-SEARCH BMA KERNEL FOR SHAPE MOTION ESTIMATION	20
FIG. 2.7 ARCHITECTURE DESIGN PRESENTED IN [11].	22
FIG. 2.8 ARCHITECTURE DESIGN PRESENTED IN [10].	22
FIG. 2.9 COMPUTATIONAL COMPLEXITY OF MPEG-4 SHAPE ENCODER.....	23
FIG. 2.10 FLOWCHART FOR THE PROPOSED ALGORITHM.....	26
FIG. 1.11 (A)CURRENT BAB CONTAINS CERTAIN NUMBER OF “1” BITS AND BELONGS TO CLASS 3. (B) SEARCH WINDOW SHOWING TWO BABS, EACH ONE FROM DIFFERENT CLASSES.	27
FIG. 2.13 THE IMPACT OF CLASSES OVERLAPPING ON BITS/SHAPE, TOTAL ENCODED BIT-STREAM AND THE NUMBER OF SEARCH POSITIONS.	28
FIG.2.13 THE PROBABILITY DISTRIBUTION FOR THE 256 CLASSES IN THE CONTAINER_2_OBJ.	30
FIG. 2.14 ARCHITECTURE BLOCK DIAGRAM OF THE PROPOSED ALGORITHM	31
FIG. 2.15 (A) THE SAD ARCHITECTURE, AND (B) THE TREE ADDER.	32
FIG. 2.16 TWO PASSES NEEDED TO COVER THE SEARCH WINDOW	33
FIG. 2.17 SLIDING WINDOW APPROACH: COUNTING THE “1” IN BABS.....	33
FIG. 2.18 ADJACENT BABS BELONG TO THE SAME CLASS, AND MATCH THE SAME CLASS OF CURRMB. SAD CALCULATION WILL BE HELD AT THE SAME TIME IN HARDWARE IMPLEMENTATION.....	35
FIG. 3.1 BLOCK MATCHING ALGORITHM.	40
FIG. 3.2 (A) ALGORITHM FLOW FOR THE PDS, AND (B) OUR PROPOSED ALGORITHM.	45
FIG. 3.3 (A) SEVEN MACROBLOCK MODES IN H.264/AVC, (B)VALUES FOR SAD _{TH} , ERROR, AND DEC FOR	

DIFFERENT BLOCK SIZES IN H.264 AND THAT FOR MPEG-4	46
FIG. 3.5 SCAN PATTERNS IN THE SEARCH WINDOW, (A) RASTER SCAN , AND (B) SPIRAL SCAN.....	50
FIG. 3.6 BLOCK DIAGRAM OF THE PROPOSED ARCHITECTURE.	56
FIG. 3.7 ME UNIT INTERNAL DESIGN.....	58
FIG. 3.8 SPATIAL OVERLAPPING BETWEEN DIFFERENT MODES.	58
FIG 4.1 FOUR DATA REUSE LEVELS (A) LEVEL A (B) LEVEL B (C) LEVEL C AND (D) LEVEL D.....	65
FIG. 4.2 MEMORY ACCESS DIAGRAM IN [64]. THE OVERLAPPED SEARCH REGION (GRAY PART) WILL BE ACCESSED TWICE. EACH BLOCK IN THE DIAGRAM DENOTES ONE MB.	66
FIG. 4.3 PROPOSED SCHEME WITH FIVE CURRENT BLOCKS	67
FIG. 4.4 (A) PERCENTAGE OF NUMBER OF MEMORY WHEN NORMALIZED TO ONE CURRENT BLOCK, (B) THE PERCENTAGE INCREASE IN BUFFER MEMORY SIZE NORMALIZED TO ONE CURRENT BLOCK.	71
FIG. 4.5 BLOCK DIAGRAM FOR THE PROPOSED ARCHITECTURE WITH 4-PE'S.....	72
FIG. 4.6 (A) DETAILED BLOCK DIAGRAM OF PE.(B) BLOCK DIAGRAM OF ABSOLUTE DIFFERENT BLOCK (C) BLOCK DIAGRAM OF SAD ACCUMULATION BLOCK, WHERE AC DENOTES THE ACCUMULATOR AND R DENOTES THE REGISTERS.....	74
FIG. 4.7 PE OPERATION FOR THE (A) FIRST CANDIDATE POSITION AND (B) SECOND CANDIDATE POSITION SHOWING THE DATA MULTIPLEXING BETWEEN BOTH PE'S.	75
FIG. 4.8 MAPPING OF MEMORY MODULES TO THE SEARCH RANGE FOR EVERY PE.....	75
FIG. 5.1 NINE SEARCH POSITIONS NUMBERED ACCORDING TO THEIR ACCESSING ORDER AS IN JM9.0 ..	83
FIG. 5.2 TESTING PATTERNS FOR THE PROPOSED ALGORITHM.	85
FIG. 5.3 SHOWS THE NINE SEARCHING POSITIONS, AND THE OVERLAPPED REGION BETWEEN THEM. ...	86
FIG. 5.4 CODING MODES IN H.264	88
FIG. 5.5 HILBERT SCAN EXAMPLE FOR 16X16 BLOCK.	89
FIG. 5.6 ILLUSTRATION OF ABME SEARCH STRATEGY WITH XOR BLOCK-MATCHING CRITERION.	91

List of Tables

TABLE 2.1 BAB CODING MODES.....	16
TABLE 2.2 COMPUTATIONAL COMPLEXITY FOR MPEG-4 VIDEO ENCODING.....	17
TABLE 2.3 COMPLEXITY ANALYSIS OF BINARY MOTION ESTIMATION	20
TABLE 2.4 16-CLASSES CLASSIFICATION, SHOWING THE NUMBER OF “1” IN EACH CLASS, AND THE RANGE OF “1” IN EVERY BAB THAT MATCHES WITH EACH CLASS.....	26
TABLE 2.5 PERFORMANCE OF THE PROPOSED ALGORITHM AND THE FULL SEARCH METHOD WHEN SEARCH WINDOW IS ± 16 AND 255 CLASSES WITHOUT OVERLAPPING.	27
TABLE 2.6 PERFORMANCE OF THE PROPOSED ALGORITHM AND THE FULL SEARCH METHOD WHEN SEARCH WINDOW IS ± 16 AND 255 CLASSES WITH 6 CLASSES OVERLAPPING.	28
TABLE 2.7 COMPARISON BETWEEN DIFFERENT CLASSES.....	28
TABLE 2.8 RESULTS FOR DIVIDING CLASSES ACCORDING TO THE PROBABILITY OF CONTAINER_2 TEST SEQUENCE.	29
TABLE 2.9 HARDWARE SIMULATION RESULTS FOR DIFFERENT CLASSES OVERLAPPING.....	36
TABLE 2.10 CHG_SP FOR VARIOUS SEARCH ALGORITHMS RELATIVE TO THE FULL SEARCH ALGORITHM.	37
TABLE 2.11 AVERAGE BIT-RATE FOR VARIOUS SEARCH ALGORITHMS. ALL ARE RELATIVE TO THE FULL SEARCH ALGORITHM.....	37
TABLE 3.1 EXPERIMENTAL RESULTS FOR THE PROPOSED METHOD WITH $Q_p = 16$	50
TABLE 3.2 EXPERIMENTAL RESULTS FOR MPEG-4 WITH SPIRAL SCAN PATTERN, $Q_p = 16$, AND 4MV ENABLED.....	52
TABLE 3.3 EXPERIMENTAL RESULTS FOR H.264 WITH SPIRAL SCAN PATTERN, $Q_p = 28$, ± 32 SR, AND RDO DISABLED.	52
TABLE 3.4 COMPARISON FOR DIFFERENT ALGORITHMS	53
TABLE 3.5 THE RESULTS OF APPLYING THE HARDWARE (H/W) CONDITIONS TO JM9.0, $Q_p = 28$, ± 32 SR, AND RDO DISABLED	55
TABLE 3.6 THE EFFECT OF DISABLING TERMINATION AFTER CERTAIN NUMBER OF LINES COMPARED TO THE HARDWARE SIMULATION RESULTS.	55
TABLE 3.7 MODES GENERATED EVERY 4 CLOCK CYCLES	59



* THE LABELS ARE AS INDICATED IN FIG. 3.3 (A)	59
TABLE 3.8 CLOCK CYCLES CONSUMED TO FINISH 16x16 SEARCH WINDOW FOR 50 FRAMES.	59
TABLE 3.9 COMPARISON OF SOME VBSME CORE	60
TABLE 4.1 REDUNDANCY ACCESS COUNT AND BUFFER SIZE NEEDED BY EACH REUSE LEVEL.	65
TABLE 4.1 NUMBER OF ACCESS AND SEARCH RANGE BUFFER SIZE FOR DIFFERENT NUMBER OF CURRENT BLOCKS.	68
TABLE 4.2 COMPARISONS FOR CIF FORMAT, SEARCH RANGE ± 16 AND USING 4 PE. PE IS ASSUMED TO BE 1-D ARRAY.	69
TABLE 4.3 MEMORY ACCESS CYCLES FOR DIFFERENT NUMBER OF CURRENT BLOCKS IN THE 2-D ARRAY CASE.	69
TABLE 4.4 TIMING AND DATA ACCESSED FROM DIFFERENT MEMORY MODULES FOR EVERY PE.	76
TABLE 4.5 COMPARISON BETWEEN OUR SCHEME AND OTHERS FOR 352x288 CIF FORMAT, SEARCH RANGE ± 16 AND USING 4 CURRENT BLOCKS, WHERE N IS ASSUMED TO BE 16.	77
TABLE 5.1 PSNR, ENCODED BIT STREAM AND COMPLEXITY REDUCTION FOR THE PROPOSED ALGORITHM.	84



Abstract

Motion estimation is one of key part in modern video standards like MPEG-4 and H.264 to remove the temporal redundancy between video frames. However, it is also computational intensive and memory intensive. Thus, in this dissertation, we propose two designs, binary motion estimation and variable block size motion estimation, to reduce the computational load, and one vertical data reuse scheme to minimize the memory access.

The first work supports the binary motion estimation for shape coding adopted by MPEG-4. In binary motion estimation, its processing is at the bit level and thus is not suitable for general purpose processors due to their word-level processing capability. Thus, we propose a fast algorithm and its architecture that takes advantages of this bit level (binary level). With the count of bits in a block, the proposed algorithm classifies and tests every candidate search position and then skips those unlikely to be a match. The proposed algorithm can adaptively overlap matching between different classes to get more accurate motion vector or more skipping ratio. The proposed algorithm achieves a saving in computational complexity ranging from 96.69% to 99.71% comes with the expense of increasing the shape encoded bits by 0.7% to 12.8%. Due to the simplicity and the regularity of the algorithm, the proposed hardware is also regular and needs only 11582 gate count.

The second work supports the variable block size motion estimation. Variable block size limits the efficiency of early termination, but the algorithm shows good performance in this field. This design uses the early termination that adaptively changes its threshold to fit the variable block size and achieve early skipping. Different variables can be tuned by the algorithm to compromise between the high skipping ratio and the accurate motion vector. The proposed algorithm outperforms other similar algorithms with a complexity reduction of 78% and 51% for MPEG-4 and H.264 respectively. The hardware implementation of the algorithm can process one MB in 16 clock cycles, and completes a 16x16 search window in 4096 clock cycle without any termination process and an average 1032 clock cycles with termination process. The hardware uses only 16 registers and 31 adders and gate count of 16k.

Finally, the third work reduces the huge memory access by vertical processing adjacent current macroblocks. Vertical processing can achieve the same speed up of the horizontal processing but lower memory access especially for large search window. A design is introduced to demonstrate the efficiency of the vertical processing compared to horizontal processing using the same number of processing elements. This simple and regular design can be easily extended to any number of PE without extra cost to the control circuit or any change in the data flow. The required data bandwidth is reduced by 60.9% with four processing elements and 61k gate count when compared to the previous designs.



Introduction

1.1 Overview of Video Coding

The rapid development of digital technology has brought the video to a new era. In the past, people used tapes and televisions, but today the digital video can be found everywhere in our daily life. The VCD and DVD have replaced traditional tapes owing to the ease of preservation. As computer networks become more and more popular, video over network tends to be more and more important. Handheld devices probably will be one of the terminals for video broadcast in the near future.

The increasing demand to incorporate video data into telecommunications services, the corporate environment, the entertainment industry, and even at home has made digital video technology a necessity. A problem, however, is that still image and digital video data rates are very large, typically in the range of 150Mbits/sec. Data rates of this magnitude would consume a lot of the bandwidth, storage and computing resources in the typical personal computer. Therefore, Video Compression standards have been developed to eliminate picture redundancy, allowing video information to be transmitted and stored in a compact and efficient manner. Although the applications on video change so rapidly, the bases of the video technology are still the same:

1. Insensitivity to high frequency of human eye.

The human eye is lazy to the high frequency part of the signal. Thus, both compression of still images and video improves its compression ratio by largely discarding the high-frequency components.

2. Spatial redundancy.

In a small area of the current coded frame, the pattern is usually lack of large variation. That is the pattern of the neighbors can be good prediction of currently coded area.

3. Temporal redundancy.

The characteristics of video include the temporal similarity, which means almost no great changes exist between two close frames. By eliminating the temporal redundancy from the current coded frame, the data rate can be largely lowered. In fact, the applying of video coding tools to exploits the temporal redundancy is the major improvement in compression ratio.

Thus, most of the video coding standards, including IYU H.261, H.263, H.264, MPEG, MPEG-2, and MPEG-4, are established on similar fundamentals, such as block-based coding, DCT transform, motion estimation/compensation, and prediction of data and so on.

1.2 Cores of Digital Video Technology


Compression is the most important core technique of video applications at present. Video standards evolution starts from H.261, MPEG-1, MPEG-2/H.262, H.263, H.263+, H.263++, MPEG-4, H.26L, to the latest H.264/AVC. H.264/AVC outperforms its previous standards in compression performance due to many new prediction and entropy coding tools, such as, multiple reference frames, variable block sizes, intra prediction, context-based adaptive variable length coding, context-based adaptive binary arithmetic coding, deblocking and rate distortion optimized mode decision. This high performance comes with the expense of high computational complexity by one order.

The toughest challenge of video compression is real-time processing. For HDTV applications, several tera-operations/instructions per second (TOPS) of computing power and several tera-bytes per seconds (TB/s) of memory access are demanded. The required resources of real-time coding are far beyond the capabilities of today's general purpose processors. For mobile applications with much smaller image sizes, power

consumption is the most critical issue. If a processor-based software implementation is adopted, the operating frequency of the processor should be as high as hundreds of MHz, which will violate the strict power constraints. Although the complexity of mobile applications is much lower than that of HDTV applications, real-time compression is still too heavy burden for processors. Hence, application specific integrated circuits (ASICs) play the key role of video applications. Hardware-oriented algorithms and efficient VLSI architectures are urgently needed. Among all the video coding standards, block matching estimation is always the processing bottleneck. More than half, sometimes even up to 90% of computing power and memory access are dominated by motion estimation. Therefore we should pay attention to the motion estimation when developing a video coding system.

In the following two subsections, a brief introduction of two compression standards, MPEG-4 and H.264, showing their main features, basic functional blocks and the distinguished tools implemented in each of them.

1.2.1 MPEG-4 Overview



MPEG-4 is an ISO standard (ISO/IEC international standard 14496) developed by the Moving Picture Experts Group (MPEG). It defines the deployment of non-proprietary multimedia content independently of platform or transmission medium. It has relied on and taken from a number of existing technologies while at the same time adding a number of innovative tools and concepts.

The MPEG-4 visual standard consists of a set of tools (as shown in Fig. 1.1) that enable applications by supporting several classes of functionalities. The most important features covered by MPEG-4 standard can be clustered in three categories and summarized as follows:

1. *Compression efficiency*: Compression efficiency has been the leading principle for MPEG-1 and MPEG-2 and in itself has enabled applications such as Digital TV and DVD. Improved coding efficiency and coding of multiple concurrent data streams will increase acceptance of applications based on the MPEG-4 standard.

2. *Content-based interactivity*: Coding and representing video objects rather than video frames enables content-based applications. It is one of the most important novelties offered by MPEG-4. Based on efficient representation of objects, object manipulation, bit stream editing, and object-based scalability allow new levels of content interactivity.

3. *Universal access*: Robustness in error-prone environments allows MPEG-4 encoded content to be accessible over a wide range of media, such as mobile networks as well as wired connections. In addition, object-based temporal and spatial scalability allow the user to decide where to use sparse resources, which can be the available bandwidth, but also the computing capacity or power consumption.

To support some of these functionalities, MPEG-4 should provide the capability to represent arbitrarily shaped video objects. Each object can be encoded with deferent parameters, and at deferent qualities. The shape of a video object can be represented in MPEG-4 by a binary or a gray level (alpha) plane. The texture is coded separately from its shape. In the following, two main parts of MPEG-4 will be discussed which are related to the work presented later in the thesis: shape coding and texture motion estimation.

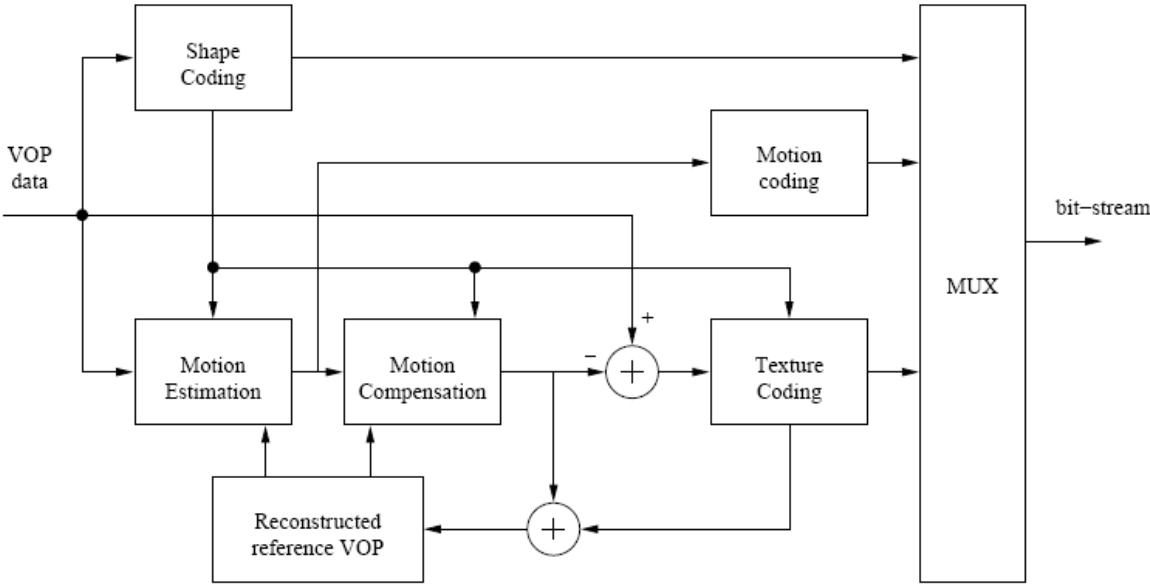
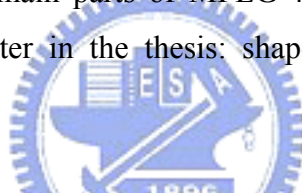


Fig. 1.1 General structure of MPEG-4 VOP encoder

1.2.1.1 Shape coding

The shape coding scheme also relies on motion estimation to compress the shape information even further. In MPEG-4 visual standard, two kinds of shape information are considered as inherent characteristics of a video object. These are referred to as binary and gray scale shape information. By binary shape information, one means label information that defines which portions (pixels) of the support of the object belong to the video object at a given time. The binary shape information is most commonly represented as a matrix with the same size as that of the bounding box of a Video Object Plan (VOP). Every element of the matrix can take one of the two possible values depending on whether the pixel is inside or outside the video object. Gray scale shape is a generalization of the concept of binary shape providing a possibility to represent transparent objects, and reduce aliasing effects. In gray scale, the shape information is represented by 8 bits, instead of a binary value.

In the past, the problem of shape representation and coding has been thoroughly investigated in the fields of computer vision, image understanding, image compression and computer graphics. However, this is the first time that a video standardization effort has adopted a shape representation and coding technique within its scope. In its canonical form, a binary shape is represented as a matrix of binary values called a bitmap. Since its beginning, MPEG adopted a bitmap based compression technique for the shape information. This is mainly due to the relative simplicity and higher maturity of such techniques. Experiments have shown that bitmap-based techniques offer good compression efficiency with relatively low computational complexity.

Binary shape information is encoded by a motion compensated block based technique allowing both lossless and lossy coding of such data. In MPEG-4 video compression algorithm, the shape of every VOP is coded along with its other properties (texture and motion). The next chapter will discuss the process of shape coding in more detail.

1.2.1.2 Texture Motion estimation and compensation

Motion estimation and compensation are commonly used to compress video sequences by exploiting temporal redundancies between frames. The approaches for motion compensation in the MPEG-4 standard are similar to those used in other video

coding standards. The main difference is that the block-based techniques used in the other standards have been adapted to the VOP structure used in MPEG-4. MPEG-4 provides three modes for encoding an input VOP namely:

1. A VOP may be encoded independently of any other VOP. In this case the encoded VOP is called an *Intra VOP* (I-VOP).
2. A VOP may be predicted (using motion compensation) based on another previously decoded VOP. Such VOPs are called *Predicted VOPs* (P-VOP).
3. A VOP may be predicted based on past as well as future VOPs. Such VOPs are called *Bidirectional Interpolated VOPs* (B-VOP). B-VOPs may only be interpolated based on I-VOPs or P-VOPs.

Obviously, motion estimation is necessary only for coding P-VOPs and B-VOPs. Motion estimation (ME) is performed only for macroblocks in the bounding box of the VOP in question. If a macroblock lies entirely within a VOP, motion estimation is performed in the usual way, based on block matching of 16x16 macroblocks as well as 8x8 blocks (in advanced prediction mode).

1.2.2 H.264 Overview

H.264 video coding standard has the same basic functional elements as previous standards (MPEG-1, MPEG-2, MPEG-4 part 2, H.261, and H.263), i.e., transform for reduction of spatial correlation, quantization for bitrate control, motion compensated prediction for reduction of temporal correlation, and entropy encoding for reduction of statistical correlation. Moreover, to fulfill better coding performance, the important changes in H.264 occur in the details of each functional element by including intra-picture prediction, a new 4x4 integer transform, multiple reference pictures, variable block sizes and a quarter pel precision for motion compensation, a deblocking filter, and improved entropy coding. Improved coding efficiency comes at the expense of added complexity to the coder/ decoder. H.264 utilizes some methods to reduce the implementation complexity. Multiplier-free integer transform is introduced.

Multiplication operation for the exact transform is combined with the multiplication of quantization.

The block diagram for H.264 coding is shown in Fig. 1.2. Encoder may select between intra- and inter-coding for block-shaped regions of each picture. Intra-coding uses various spatial prediction modes to reduce spatial redundancy in the source signal for a single picture. Inter-coding (predictive or bi-predictive) is more efficient using inter-prediction of each block of sample values from some previously decoded pictures. Inter-coding uses motion vectors for block-based inter-prediction to reduce temporal redundancy among different pictures.

Inter-prediction is to reduce the temporal correlation with help of motion estimation and compensation. In H.264, the current picture can be partitioned into the macroblocks or the smaller blocks. A macroblock of 16x16 luma samples can be partitioned into smaller block sizes up to 4 x 4. For 16x16 macroblock mode, there are 4 cases: 16x16, 16x8, 8x16 or 8x8, also four cases: 8x8, 8x4, 4x8 or 4x4 for 8x8 mode.

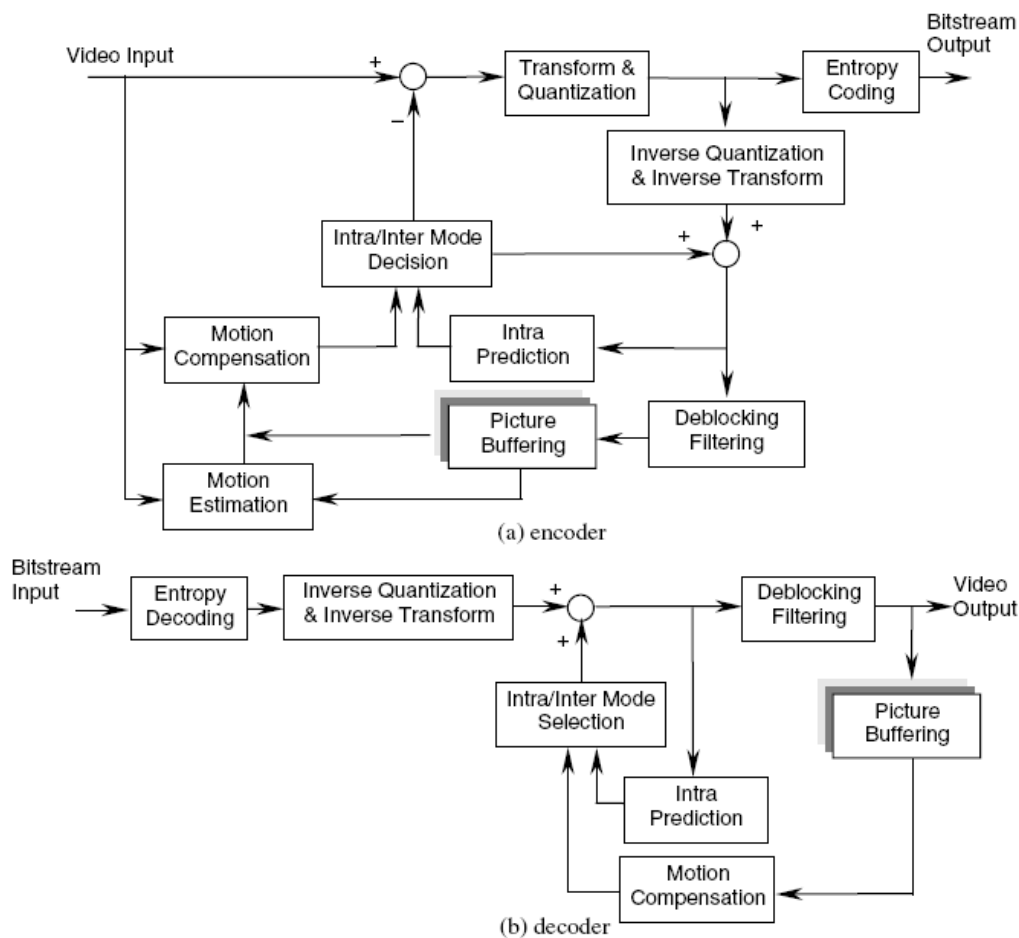


Fig.1.2 The block diagram of H.264 algorithm.

The smaller block size requires larger number of bits to signal the motion vectors and extra data of the type of partition, however the motion compensated residual data can be reduced. Therefore, the choice of partition size depends on the input video characteristics. In general, a large partition size is appropriate for homogeneous areas of the frame and a small partition size may be beneficial for detailed areas. The inter-prediction process can form segmentations for motion representation as small as 4x4 luma samples in size, using motion vector accuracy of one-quarter of the luma sample. Sub-pel motion compensation can provide significantly better compression performance than integer-pel compensation, at the expense of increased complexity. Quarter-pel accuracy outperforms half-pel accuracy. Especially, sub-pel accuracy would increase the coding efficiency at high bitrates and high video resolutions.

1.3 Motivation and Contributions

From the above brief introduction, one major block is common with MPEG-4 and H.263, the inter-prediction part. The core of that part is the motion estimation. Motion estimation consumes 50% to 90% from the system computing power and memory access. Bear in mind that, the shape coding adopted in MPEG-4 is a bit-level processing algorithm, and thus it deals with bits not with words. The last issue will add more burden to the general purpose processors. Moreover, special data scheduling and placement should be taken into considerations. Motion estimation using variable block size that is adopted in H.264 is another burden we need to take into consideration.

Real time processing and the restricted power consumption in handheld devices, created a big challenge for designers to implement those coders to meet such requirements. This leads to adopt fast algorithms to meet the real time restriction, with the expense of quality degradation and increased encoded bitstream size. Thus, a dedicated hardware is suitable to implement the motion estimation part, especially when implementing the shape coding.

We proposed a fast Binary Motion Estimation (BME) algorithm and architecture to serve the shape coding part of MPEG-4. The dedicated hardware implemented will solve the burden of bit-level processing of the bit mask generated for the VOP. Also the

fast algorithm speeds up the matching process to save the time and enhance the system performance to meet the real time requirements.

Integer motion estimation is the kernel of the motion estimation process. It is the first and basic stage and after it, comes the fractional motion estimation, and then the mode decision adopted in H.264. We propose a fast integer motion estimation algorithm and architecture to serve this part of the motion estimation process. The hardware implementation should take into consideration the variable block size adopted in H.264. The fast algorithm by early termination scheme, based on testing the macroblock line by line and comparing the generated sum of the absolute difference (SAD) to an adaptive threshold value.

Finally, to reduce the memory traffic from the system main memory to the dedicated hardware, a data reuse strategy used to serve this issue. Vertical processing of macroblocks will reduce significantly the memory bandwidth needed. This approach could be combined with the former mentioned algorithms to serve fast motion estimation algorithms and reduce the system memory bandwidth. This contribution will reduce the over all system power by minimizing the main system memory access; also minimum the gap to real time processing for high quality video formats.

Fig 1.3 shows the relation between the works presented in this thesis. Binary motion estimation for shape coding is part of the Motion Estimation kernel. BME serves MPEG-4 only. The work extended to serve the motion estimation which is adopted by both MPEG-4 and H.264. The motion estimation part of this work applies an early termination scheme to reduce the complexity of the motion estimation. To reduce the burden of huge memory access, a new approach proposed to reduce the main memory access. Vertical processing for motion estimation of adjacent macro blocks reduced significantly the main memory access. Fig. 1.4 shows the problems of the motion estimation process and the contributions made to solve these problems.

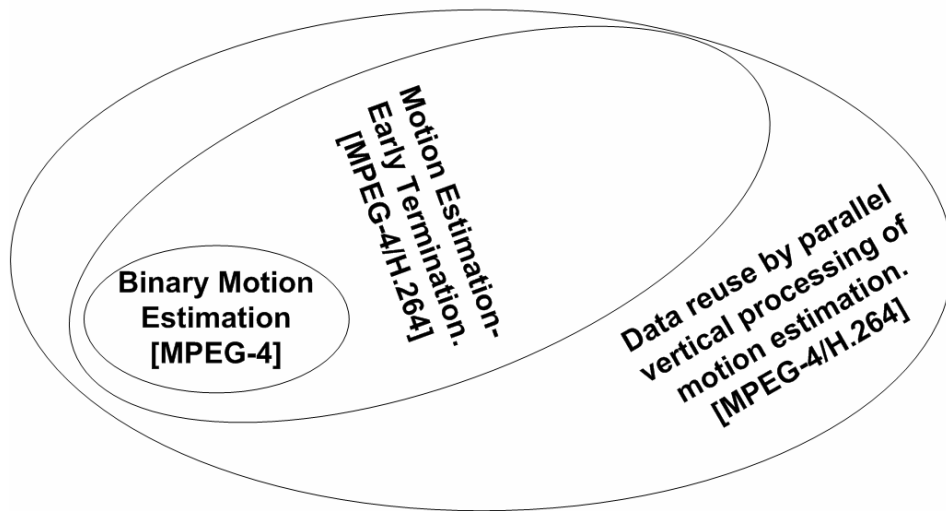


Fig. 1.3 The relations between the works presented in this thesis

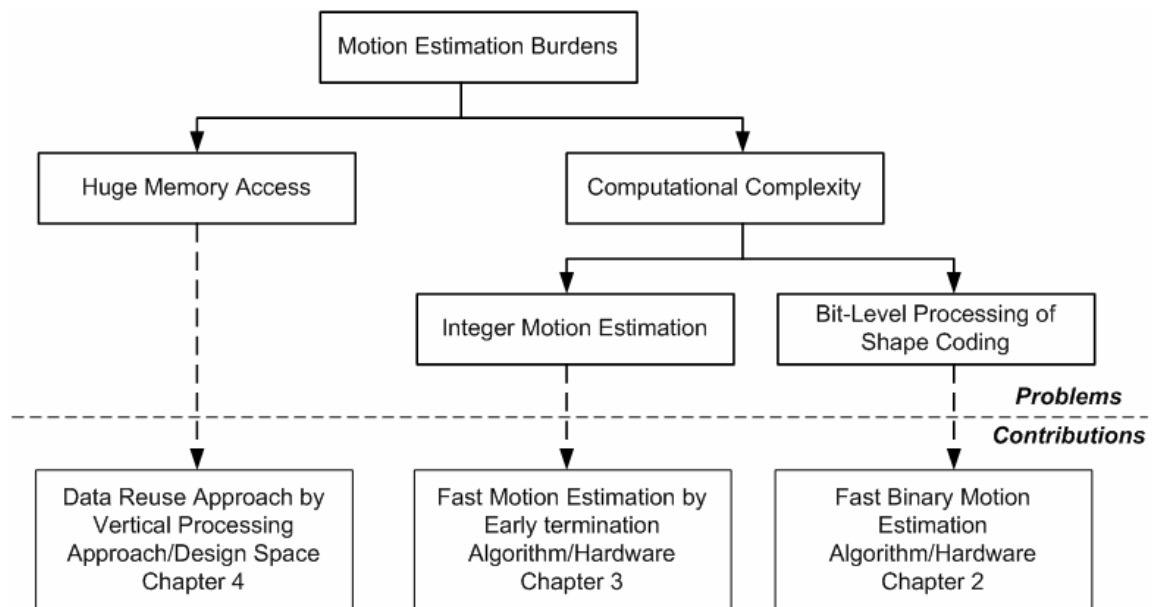


Fig. 1.4 Contributions of the works presented in this thesis.

1.4 Dissertation Organization

This dissertation is organized as follows. Chapter 2 will present the Binary Motion Estimation for shape coding adopted in MPEG-4. The proposed algorithm and hardware will be presented in detail. Integer motion estimation with early termination will be presented in chapter 3. Detailed description of the algorithm, and the hardware implemented will be given. Data reuse and the reduction in the main system memory access will be presented in chapter 4. Finally, the conclusion and future work will be presented in chapter 5.

Chapter 2 A Fast Binary Motion Estimation Algorithm and its Architecture Design

In this chapter, we first present an introduction to MPEG-4 system with the focus on the shape encoding. Details will be presented to explain the function of binary motion estimation to show its role in MPEG-4 coding system and the computational burden it introduces to the system. Then, we will show the previous work on MPEG-4 shape coding and its contribution. Finally, we present our algorithm for binary motion estimation and the associated hardware implementation.

2.1 Overview of the MPEG-4 Video System

MPEG-4 standard is one of the popular international standards for video coding. MPEG-4 includes the following tools for natural video coding: *I-VOP* coding, *P/B-VOP* prediction coding, temporal scalability for rectangular or arbitrary-shaped objects, rectangular spatial scalability, error resilience for rectangular or arbitrary-shaped objects, binary/grayscale shape coding, and rectangular or object-based texture coding.

One of the MPEG-4's special features is the object-based scene description. It allows the transmission of arbitrarily shaped video objects. The purpose of using shape coding is to promote better subjective picture quality, higher coding efficiency as well as more user interactions. For more detailed information please refer to [76].

2.1.1 Video Object and Video Object Plane

MPEG-4 international standard treats moving pictures as an organized collection of visual objects and provides several advanced techniques to access and represent the moving arbitrarily shaped natural and synthetic objects in video scenes. The MPEG-4 visual specifications support several types of visual objects, among which is the *video object*. The video object may be thought of as a sequence of two-dimensional images. Each image can be associated with shape information to define its shape. As shown in

Fig. 2.1, a *video object plane* (VOP), which is the instance of a video object at a given time, is composed of a bitmapped alpha component to define the object's shape as in Fig. 2.1(a) and three color components (YCbCr) to render the object's texture as in Fig. 2.1(b). Basically, the VOP is defined as a minimum rectangle that encloses the whole object. In binary shape coding, bi-level alpha components can be either opaque or transparent. An opaque alpha component means the corresponding pixel resides in the video object and the texture information should be displayed. A transparent alpha component means the corresponding pixel is not part of the video object, where background should be displayed. Blocks that are completely outside the shape are transparent blocks since it consists of all transparent pixels. Blocks that are inside the shape are opaque blocks, because it consists of all opaque pixels. The block residing on the border of shape is the boundary block. In Fig. 2.1(c), transparent block are labeled "T", opaque block are labeled "O" and boundary block are labeled "B". A gray-scaled shape coding is similar to binary shape coding except that 256-level alpha components is used to identify the shape of video object. Gray-scaled shape coding will not be discussed.

It is noteworthy that the size of a VOP at a given time depends on the shape of the video object at that time. That is, the size of the VOP for a video object is time-variant. MPEG-4 video encoding is based on the VOP encoder shown in Fig. 2.2. The alpha component of a VOP is encoded using a binary shape encoder while the color components are encoded using motion estimation and compensation followed by DCT-based texture coding. The input data of a VOP undergoes shape coding, texture coding and motion coding. A multiplexer interleaves the bit-stream from the three coding units and output encoded bit-stream.

2.1.2 MPEG-4 Binary Shape Coding

Coding the binary shape information of a VOP takes a few major steps. First, a bounding rectangle is created and extended to multiples of 16x16 pixels with extended alpha samples set to transparency. Next, binary alpha data are grouped with what are called binary alpha blocks (BABs) to have the same dimensions as a macroblock. Finally, shape coding is initiated on a BAB-by-BAB basis.

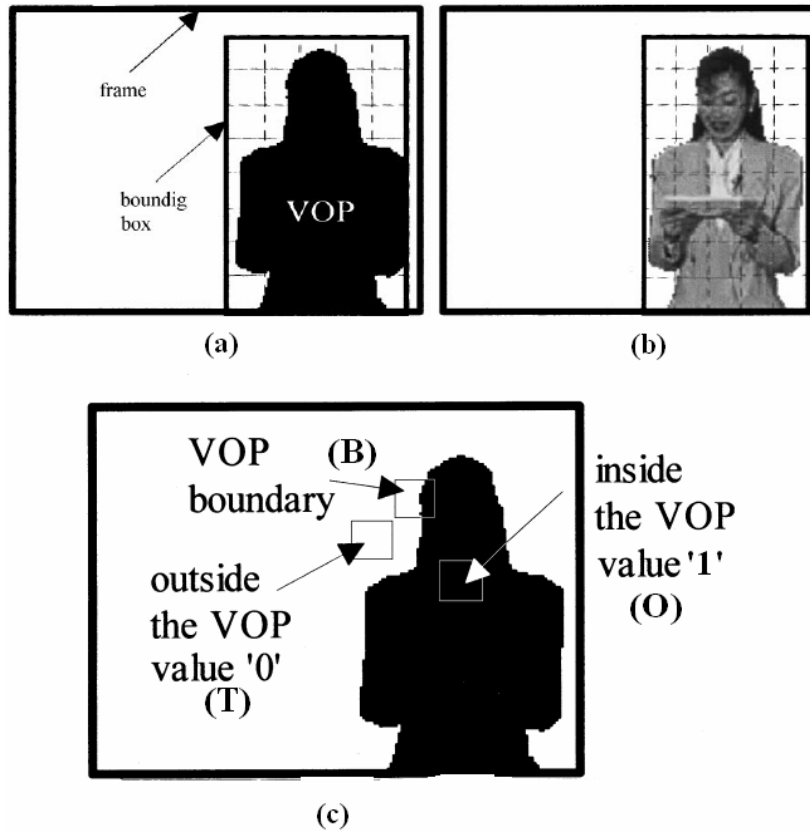


Fig.2.1 Video object plane (VOP)

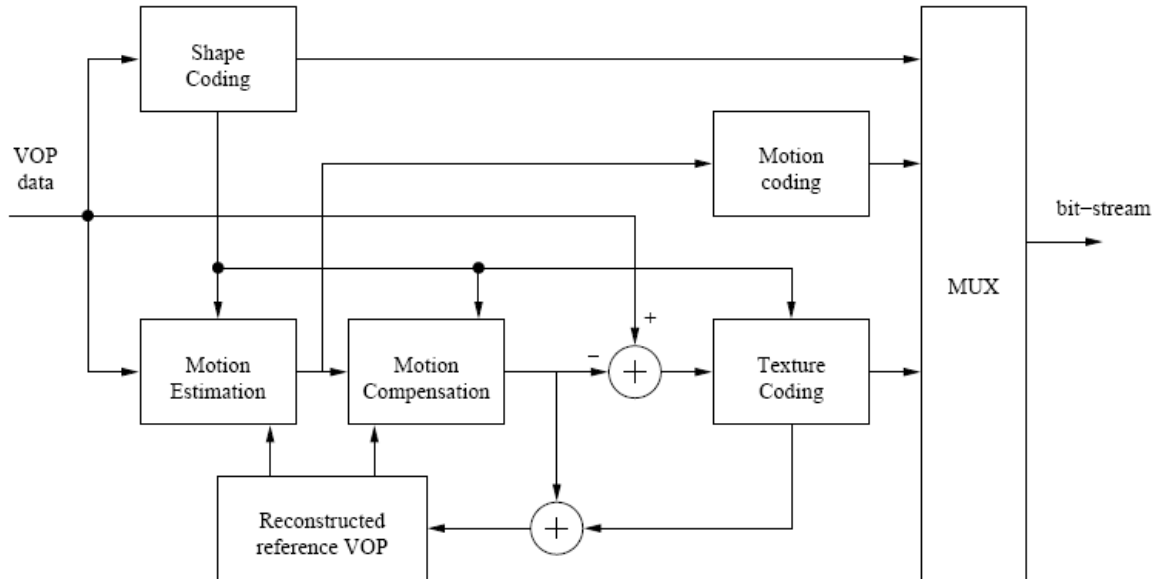


Fig.2.2 General structure of MPEG-4 VOP encoder

The encoding of a BAB can be further divided into the following scenarios. First of all, if all the pixels in a BAB are either opaque or transparent, this BAB is a non-

boundary BAB and only the coding mode is encoded by means of a variable-length coder (VLC). Second, in inter-frame, a boundary BAB can be coded with reference to a suitable prediction BAB from the previous coded frame to remove temporal redundancy. This procedure is called binary motion estimation (BME). Based on the assumption that the movement of an object is homogeneous, motion vector of neighboring BAB or texture block is used as the motion vector predictor for shape (MVPs). If the motion compensation (MC) error between the block indicated by the MVPs and current BAB is less than or equal to a predefined threshold, the MVPs is directly employed as motion vector of shape (MVs), and the procedure of BME terminates. Otherwise, full-search block matching is performed around the MVPs within a predefined search range. Each candidate block residing in the search range indicated by a motion vector is compared to the current BAB by computing 16x16 sum of absolute differences (SADs). The motion vector that minimizes the SADs is taken as MVs. The MVs is further interpreted as motion vector difference for shape (MVDs), i.e., $MVDs = MVs - MVPs$ as shown in Fig. 2.3. In this case, coding mode and MVDs are encoded by shape encoder using VLC.

At last, apart from those cases mentioned above, it is generally necessary to employ context-based arithmetic encoding (CAE) to the pixels within a BAB. There are two CAE operation modes, one is intra-mode and the other is inter-mode. Intra-mode CAE exploits the spatial redundancy by estimating the probability of the current pixel from its neighboring pixels. As for inter-mode, temporal redundancy is exploited by estimating the probability of the current pixel from its neighboring pixels and motion compensated neighboring pixels of the previous coded frame.

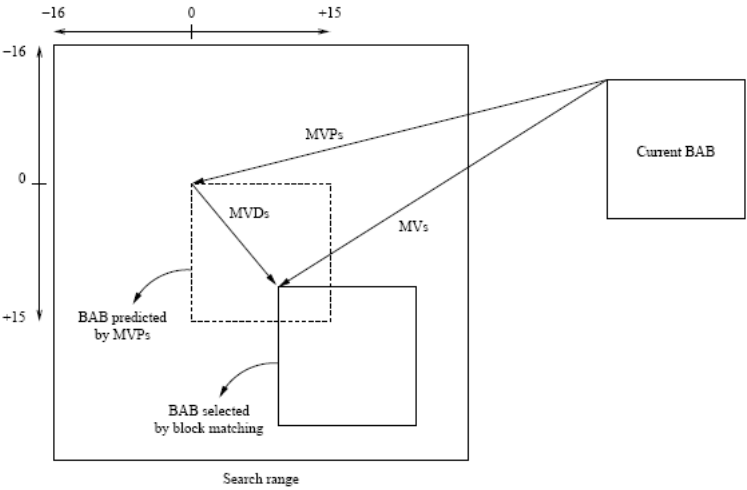


Fig. 2.3 Block matching of motion estimation for shape

In intra-frame coding, only intra-mode CAE is performed. In inter-frame coding, both intra- and inter-mode CAE are performed. In both modes, CAE operation is performed pixel by pixel in both horizontal and vertical raster scan order. Therefore, there are total four different coding processes: $CAE_{(intra_h)}$, $CAE_{(intra_v)}$, $CAE_{(inter_h)}$, and $CAE_{(inter_v)}$. At each pixel, a template is formed to extract a context number used to access a probability table. The accessed probability and the pixel value are then used to drive an arithmetic encoder. In intra-frame coding, the encoded bitstream with minimum size between $CAE_{(intra_h)}$ and $CAE_{(intra_v)}$ is chosen as the final output. Similarly, the encoded bitstream with minimum size among $CAE_{(intra_h)}$, $CAE_{(intra_v)}$, $CAE_{(inter_h)}$, and $CAE_{(inter_v)}$ is chosen in inter-frame coding. In addition, the shape encoder may decide to encode sub-sampled versions (i.e., lossy coding) of BABs to save encoded bits. If this is the case, the sub-sampling factor, also known as conversion ratio (CR), is also encoded into the bitstream. The BAB (sub-sampled or not) then undergoes CAE. Note that CAE itself introduces no additional loss.

In summary, there are seven coding modes for MPEG-4 shape coding in total, as listed in Table 2.1. The syntax of the coding mode is represented by `bab_type` field in the bitstream.

Fig. 2.4 illustrates the block diagram of MPEG-4 binary shape coding system. To improve the performance of the shape encoder, parallel processing of partial or all operations in each step can be implemented. The general coding procedure can be as follows [76]:

- Step 1. Perform mode decision to determine if the current BAB is a boundary or nonboundary BAB. For a non-boundary BAB, go to Step 5 with `bab_type` = 2 or 3. For a boundary BAB, go to Step 2 or 3 for inter-frame or intra-frame coding respectively.
- Step 2. Perform BME. If a qualified motion vector is found, go to step 5 with `bab_type`= 0 or 1. Otherwise go to Step 3.
- Step 3. Perform size conversion to obtain a sub-sampled version of current BAB. The quality of the sub-sampled BAB must satisfy the predefined threshold. Go to Step 4.
- Step 4. Perform intra-mode CAE for intra-frame coding. Perform intra- and inter-mode CAE for inter-frame coding. Go to Step 5 with `bab_type` = 4, 5, or 6.
- Step 5. Perform VLC for BAB coding mode and other related information.

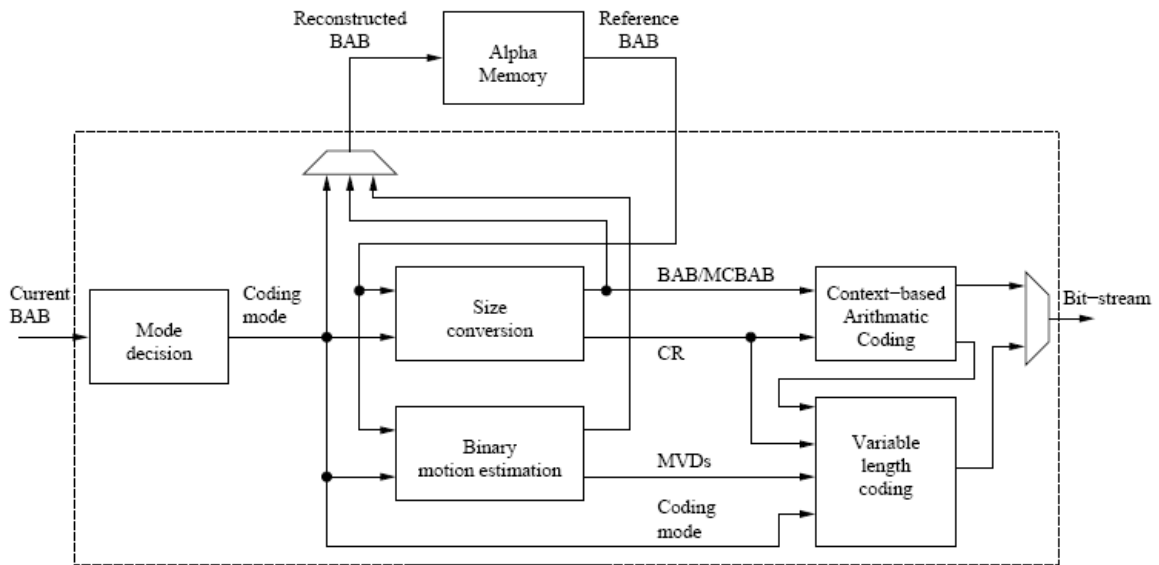


Fig.2.4 Block diagram of MPEG-4 Binary shape encoding system

Table 2.1 BAB coding modes [76]

Coding mode (bab_type)	Semantic	Used in VOP
0	no update && MVDs=0	P,B
1	no update && MVDs≠0	P,B
2	transparent	I,P,B
3	opaque	I,P,B
4	Intra CAE	I,P,B
5	Inter CAE && MVDs=0	P,B
6	Inter CAE && MVDs≠0	P,B

2.2 Motion Estimation for MPEG-4 Video

Motion estimation is an effective algorithm to exploit the temporal redundancy of video sequences. The macro-block in the current VOP can be represented by a block in the reference VOP and a motion vector indicating the displacement of the blocks. Thus, the generated encoded bits when encoding the motion vector is much smaller than that for encoding the macro-block contents. So using the encoded motion vector bits to replace the macro-block contents in encoded bit-stream will achieve high compression ratio. In video decoding phase, the macro-block is restored by copying the block contents in reference VOP back to the current VOP.

Table 2.2 Computational complexity for MPEG-4 video encoding [76]

	MIPS	Percentage
Motion Estimation	1002	66%
CAE-Shape coding	201	13%
MB-padding	57	4%
Motion compensation	15	1%
IDCT	14	0.9%
H.263 quantization	10	0.7%
DCT	9	0.6%
Others	188	12%

The motion estimation has attracted much research interests for the following reasons. First, the workload of motion estimation is most demanding in the video encoding process. In Table 2.2, profiling results showed that motion estimation takes about 66% of total computation time. It is the bottleneck to the performance of an encoder. Second, the encoded video quality is highly impacted by motion estimation at a given bitrate. Finally, the MPEG-4 video coding standard does not specify a standardized way to find the motion vector. Hence, the estimation procedure is open for competition and many algorithms for motion estimation have been proposed. The block-matching algorithm is most popular in hardware implementations of motion estimation. The block-based nature leads to regularity and parallelism which are suitable for hardware-based design.

2.2.1 Block-matching Algorithm

The block-matching algorithm is one of the most popular techniques for motion estimation. The block matching algorithm applies almost equally to both binary shape coding and texture coding of MPEG-4. The only difference is that binary shape coding deals with bi-level alpha plane data, whereas the texture coding deals with multi-level luminance components of the texture. The basic concept of block matching is to represent the block in current VOP using a block in the reference VOP and a motion vector indicating their displacements. The block in reference VOP that is a best match is the one that minimize the matching cost. A search range is set to confine the search procedure within an area that is more probably to have a good match. Various search strategies are proposed to further lower the searching time or searching cost.

The motion vector is the displacement vector of the current macro-block and the block that minimize the matching cost. Various matching cost has been proposed. These cost function vary in terms of implementation complexity and efficiency. The most popular

cost function is the Sum of Absolute Difference (SAD) which is more suitable for hardware implementation and its computation burden is the lowest among the others.

2.2.2 Search Range and Search Points

The search range is a region in the reference VOP that are likely to contain a similar block to the current block. This region is probably around the corresponding location of the current block, or around a predicted location relative to the current block. Instead of using all possible blocks within the reference VOP, only the blocks within the search range is inspected to save matching effort.

The search point is the location of a block in the search range that the matching cost of the block is to be calculated for block-matching. The search point is located on every integer-pel within the search range. In full-search block matching, the current block is shifted to each search point and the matching cost is computed. The number of search points on integer-pel is $(M-m+1) \times (N-n+1)$, where $(M \times N)$ is the dimension of search range while $(m \times n)$ is the size of current block. If the size of search range is 47x47 pixels, and the size of current block is set to 16x16 pixels. Then, there are totally $32 \times 32 = 1024$ search points to be explored by full-search block matching algorithm. For fast motion estimation techniques, matching cost are inspected on only some of the search points.

The fractional-pel motion estimation extends the search points from integer-pel to half-pel or even quarter-pel, which could generates better matching results. As shown in Fig. 2.5, the pixel values of the fractional pixels are interpolated, the matching cost calculated on the fractional search points using these interpolated pixels. Since the computation is too expensive to explore all fractional search points, the fractional-pel search is done hierarchically. The fractional-pel motion estimation usually carried out after the motion vector is found by integer-pel motion estimation. Then, the fractional-pel motion estimation is performed on the fractional search points around the integer search point just found.

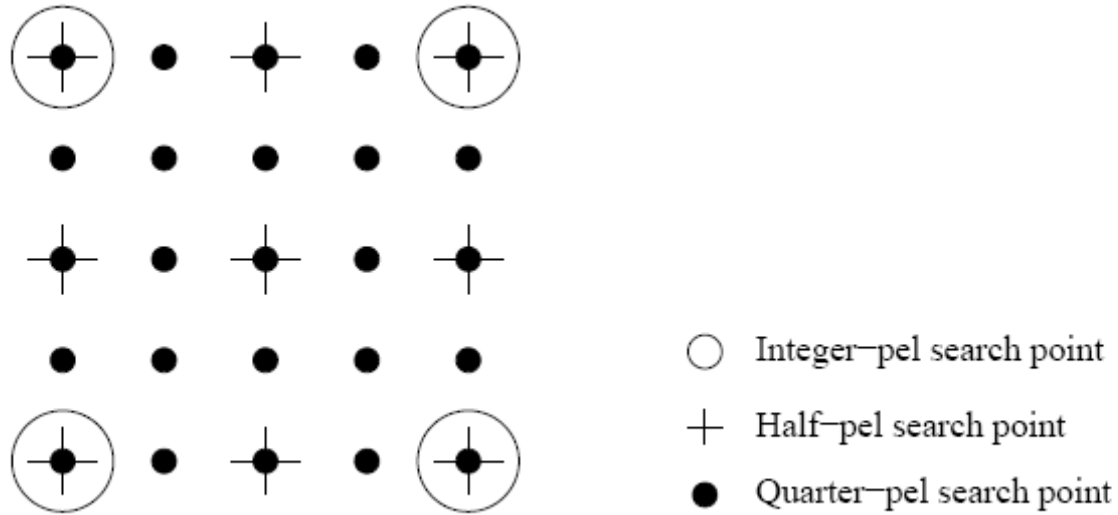


Fig. 2.5 Fractional-pel search points

2.3 Analysis of Binary Motion Estimation

The computational complexity and memory bandwidth requirements of binary motion estimation are analyzed based on the pseudo code of full-search BMA algorithm. Fig. 2.6 shows the kernel of full-search BMA algorithm for shape coding. The search range is from $-VSR$ to VSR in vertical direction, and from $-HSR$ to HSR in horizontal direction. The current BAB size is 16×16 . $I_c(m, n)$ denotes the pixel value of current BAB, and $I_r(i, j, m, n)$ denotes the pixel value of reference block at search point (i, j) .

To calculate a SAD value, first, the pixels in the current BAB and the reference block are compared, as shown in line 9. This requires two memory access and one bitwise XOR operation. Next, the difference is accumulated in SAD, as shown in line 10. The above two operations are repeated for 256 times to get a SAD value of a search point. Then, one compare operation is required to update the minimum SAD. Finally, a total number of $2VSR \times 2HSR$ search points need to be processed to find a motion vector of the current block. By using this method, the complexity of the BME kernel is analyzed. Let V denotes the total number of search points, i.e., $V \equiv 2VSR \times 2HSR$, then the analysis results are listed in Table 2.3. For a core profile level 2 application1 with $VSR = 16$; $HSR = 16$, the complexity of each operation in terms of mega-operation-per-second (MOPS) is listed in the last column of Table 2.3.

```

1  MIN_SAD=MAX_VALUE
2  for(j=-VSR; j<VSR; j++)
3  for(i=-HSR; i<HSR; i++)
4  {
5      SAD=0;
6      for(n=0; n<16; n++)
7      for(m=0; m<16; m++)
8      {
9          DIFF = Ir(i,j,m,n) XOR Ic(m,n)
10         SAD += DIFF
11     }
12     UPDATE_MIN_SAD(i, j, SAD)
13 }

```

Fig. 2.6 Pseudo code for full-search BMA kernel for shape motion estimation [76]

Table 2.3 Complexity analysis of binary motion estimation [76]

	Operation	Complexity	MPOS (CP Lv2)
Data Processing			
Pixel Compare	XOR	$V \times 16 \times 16$	3114.4
SAD accumulation	ADD	$V \times 16 \times 16$	3114.4
Update min SAD	COMP	V	12.1
Memory bandwidth			
Load current pixel	LOAD	$V \times 16 \times 16$	3113.4
Load reference pixel	LOAD	$V \times 16 \times 16$	3114.4

The static analysis of full-search BMA showed that MPEG-4 binary motion estimation needs giga-operations per second, and hundred-mega byte scale memory access per second. To meet the stringent requirements on low cost and low power in VLSI design, an optimized algorithm is essential.

2.4 Previous Work on Binary Motion

Estimation

Shape coding is the basic block of coding an arbitrarily shaped objects newly adopted by MPEG-4. This approach gives the ability of controlling one object or even implementing clickable multimedia objects, which is in the near future will make the user interact with the multimedia more actively. Many researchers worked to enhance the implementation of the shape coding part in both software and hardware implementation. Following a brief survey of previous work on

binary motion estimation, which is the core for the shape encoding.

Many fast algorithms concerned about the software implementation of binary motion estimation process. The fast algorithm approaches use various skipping techniques to speed up the BME. In [11] and [12], they skipped those search positions which are depart from the contour line of the object. They first define and locate the boundary pixels (search positions), then perform Boundary Search (BS) to locate the target MVs location. Weighted SAD and diamond search pattern for furthest improvement. In [13] it skips computations of Boundary Alpha Blocks (BAB) by testing if the motion compensation error for that BAB is less than a predefined threshold value. Also it explores the coloration between the neighboring BAB's and the current BAB. In [14] it generated a mask for the points close enough to the object boundary, and limited the search process only to those points. However, software implementation of BME on processors is not efficient since processors are more efficient at the word level instead of the bit-level operation as in BME.

Considering hardware implementation, the data in BME is just a one-bit binary value (0/1), which easily can be represented by hardware and achieve computation speedup by bit parallelism. The hardware design in [11] implements the algorithm presented earlier in the same paper. The hardware consists mainly from two parts, the first part (Fig. 2.7 (a)) searches for the boundary pixels, when it finds them it precedes to the second part. The second part (Fig. 2.7 (b)) is the processing elements array which process and computes the SADs for that position. Each processing element calculates the weighted SAD of a search position in a sub-search-range. The weighted data is calculated by subtracting sum of MVDs from 7'b1100000. The hardware design in [10] presented BME architecture by employing bit parallelism technique using 1-D systolic array to perform a full search BME (Fig. 2.8). It deploys the data dispatch technique to reduce the bit addressing operation.

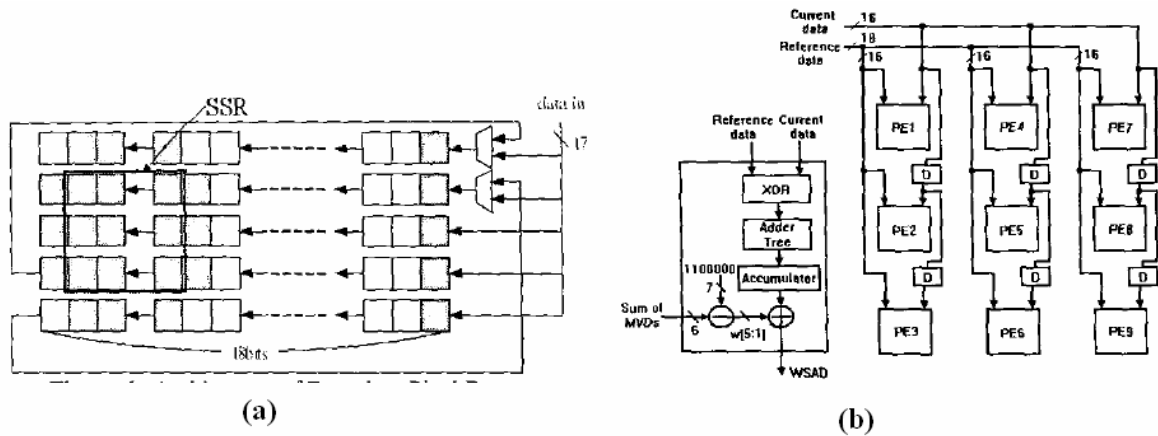


Fig. 2.7 Architecture design presented in [11].

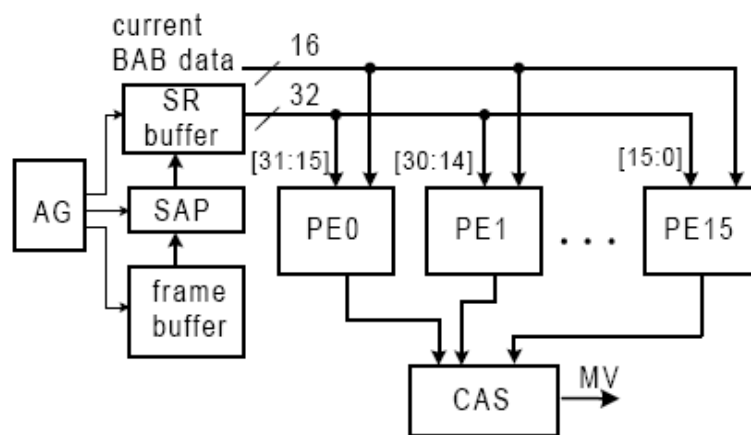


Fig. 2.8 Architecture design presented in [10].

2.5 Motivation for Fast Binary Motion

Estimation

From the above discussion, shape coding plays an important role in MPEG-4 coding system, which is one of the novelty techniques adopted by MPEG-4 to enhance the coding efficiency and introduce more interactivity between the user and the multimedia. Shape coding makes it easy to deal with single or multiple objects at the same time, adding some control and interactivity with the user such as, creating a clickable multimedia objects. Binary motion estimation is the bulk of shape encoding. Its binary nature is a burden to general purpose processors; since, these processors are designed to work efficiently with word-level operations, not with bit-level operations.

Optimized software solutions may reduce this burden when running on general purpose processors. Optimization needs careful data scheduling, extra operations, such as, shift and pack and it takes extra memory access to access individual bits.

The computational complexity of the MPEG-4 shape encoder is shown in Fig. 2.9 [12]. It can be seen that BME takes nearly 90% of total computational complexity of the MPEG-4 shape encoder [12]. Therefore, optimization on BME is essential to remove the bottleneck in achieving real time shape coding.

Moreover, the binary nature of every BAB, it contains certain number of "1". Exploring this feature, we can propose a fast algorithm, simply, by testing those search points containing almost same number of "1" as the current BAB and skip the other. The algorithm can be applied as software optimized solution, or easily mapped to a hardware implementation. No extra hardware circuits needed to test the number of "1" in every BAB.

The motivation of our approach is that, BME only deals with binary values (1 or 0) instead of the 8-bit pixel values in the texture ME. Hence, the block matching of BME can be regarded as a comparison of number of "1" contained in each candidate block with that of the current block. Thus, the proposed algorithm classifies each candidate block according to the number of "1" it contains, and only performs the block matching between those blocks belonging to the same class. Furthermore, we also present a hardware design for the proposed algorithm. Hardware design can make the binary bit-level processing much easier and faster than the software approach since traditional processor only deals with word-level processing. The simplicity and regularity of the proposed algorithm leads to a simple and regular hardware design.

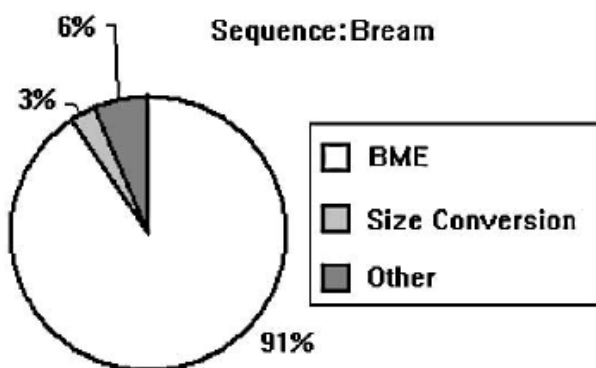


Fig. 2.9 Computational complexity of MPEG-4 shape encoder [12].

2.6 The Proposed Algorithm

Fig. 2.10 shows the proposed algorithm flowchart. The algorithm starts by classifying the current BAB according to the number of “1” it contains. Then, for each search position in the search window, we also classify the candidate BAB using the same method (counting “1” contained in the BAB). If both the current BAB and the candidate BAB are in the same class (called a match later on), we start computing SAD for that position. Otherwise, we skip that position and start testing the next search position. The main concept behind this algorithm is that, BME only deals with binary values. Thus, we can use the number of “1” contained by the BAB to approximate the BAB’s data and classify it into different classes. Hence, we can quickly skip SAD computation between different classes and only compute the SAD for the same class.

Table 2.4 represents an example for classifying each BAB according to the number of “1” it contains, i.e. if a BAB contains 20 pixels marked as “1”, it will be classified as class 2. Fig. 2.11(a) shows a BAB, the shadowed pixels represent “1” in the BAB. This BAB is classified as class 3 according to the classification in Table 2.4 (contains 34 pixels representing “1”). Fig. 2.11(b) represents the search window that shows two BABs: one with almost the same number of “1”, classified as class 3 (contains 35 pixels representing “1”), while the other classified as class 12 (contains 189 pixels representing “1”). It is clear that the BAB with almost the same number of “1” is more likely to be a match to the current BAB rather than the other one with larger number of “1”.

Classification and matching rule will severely affect the quality of searching results. The matching rule can be generalized from the same-class matching to the adjacent-classes matching. Thus, a matching could be hold for those belonging to the same class or adjacent classes. This feature (overlapping between one or more adjacent classes) gives us the ability to refine the MVS to be more accurate, which will be presented later.

2.6.1 Software simulation results and analysis

In the following two subsections, we will show the efficiency of the proposed algorithm by integrating it into the MPEG-4 verification model V18.0 [6]. All the

following test sequences are in CIF format with 300 VOP and one Video Object (VO). Then we will show how to explore the flexibility of our algorithm (the classification and the matching rule) to control both the search positions and the bit rate to get more refined MVs.

2.6.2 Block-matching Algorithm

Table 2.5 and Table 2.6 summarize the results compared with the full search algorithm, where CHG_BIT denotes the change of bits in percentage, and CHG_SP denotes the change of search position. Table 2.5 assumes ± 16 search window with non-overlapping 255 classes (every two adjacent classes differ only by one pixel value). Due to the strict non-overlapping class partitioning, the search positions saving (CHG_SP) is -99.58% (the negative sign indicates saving, in other words the percentage of search positions to that of the full search is 0.42%) of that in the full search algorithm. Such reduction comes with the cost of average 8.52% increase in the encoded shape bit rate (bits/shape).

Table 2.6 shows the effect of overlapping classes with the same ± 16 search window. With class overlapping, the increase in the bits/shape is significantly reduced to 0.65%, but it also reduces the CHG_SP to -95.68% compared with the non-overlapping case. This is because class overlapping will enable more class matching for BABs with slight difference in number of "1". Thus, the increase in the encoded bit stream will be smaller than the non-overlapping case at the cost of more search positions. Without class overlapping, we may skip possible search positions due to small difference. Fig. 2.12 shows the effect of overlapping on bit rate and search positions, it shows that, the extra bit rate increase reduced significantly with just two or three pixel overlapping. On the other hand, the required search positions are linearly increased as the number of classes overlapped is increased. Table 2.7 shows the effect of class partitioning. As the number of classes decreases (that is, count number of "1" in each class increases), the increase in encoded bit stream will be lower with the cost of increased search positions. This is because more BABs in the search window will be classified to be a match. This increases the required search positions but also reduces the bit rate due to more accurate motion matching as shown in Fig. 2.12.

Table 2.4. 16-classes classification, showing the number of “1” in each class, and the range of “1” in every BAB that matches with each class.

classes number	# of “1” included in each class	# of “1” included in the matched BAB
class 1	16	1~16
class 2	32	17~32
...
class 15	240	225~240
class 16	256	241~255

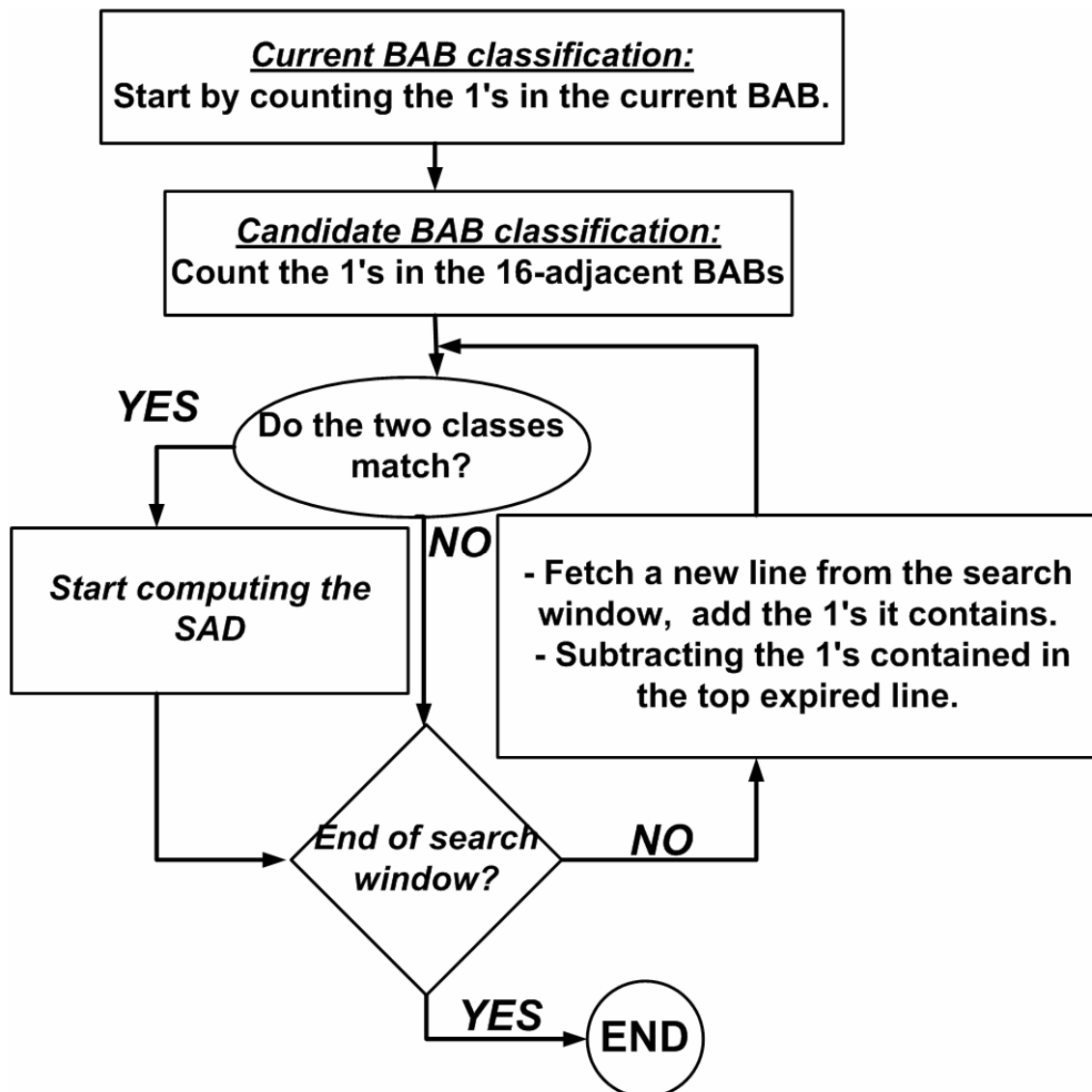


Fig. 2.10 Flowchart for the proposed algorithm.

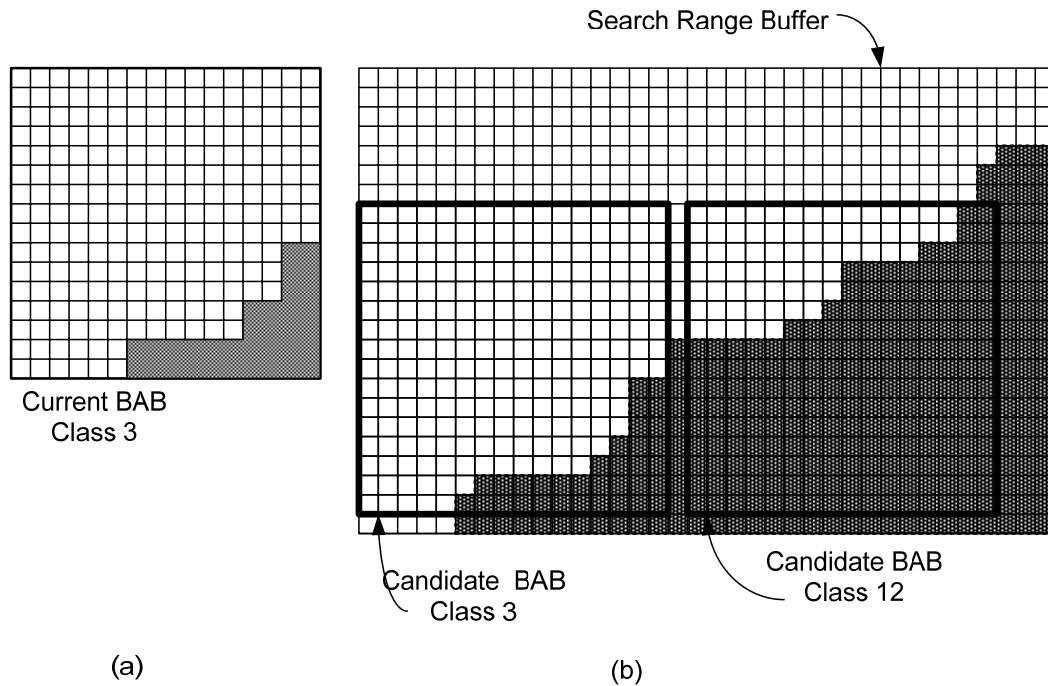


Fig. 1.11 (a) Current BAB contains certain number of “1” bits and belongs to class 3. (b) Search window showing two BABs, each one from different classes.

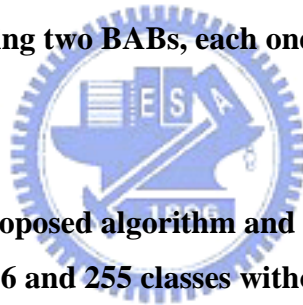


Table 2.5 Performance of the proposed algorithm and the full search method when search window is ± 16 and 255 classes without overlapping.

Test Sequences	Bits/shape			# of MB processed (search positions)		
	<i>FS</i>	<i>Proposed</i>	<i>CHG BIT</i>	<i>FS</i>	<i>Proposed</i>	<i>Saving</i>
<i>Foreman</i>	311720	328329	5.33%	7810280	25918	99.67%
<i>Stefan</i>	236964	248268	4.77%	6141676	21812	99.64%
<i>singer-247</i>	239246	275191	15.02%	6125008	20941	99.66%
<i>News</i>	298696	336933	12.80%	5863564	16956	99.71%
<i>dancer-247</i>	440828	481913	9.32%	9623385	36769	99.62%
<i>coastguard</i>	176666	188283	6.58%	2062352	11885	99.42%
<i>coastguard_obj_0</i>	401875	433491	7.87%	6066320	41951	99.31%
<i>coastguard_obj_1</i>	256330	283021	10.41%	4134816	19069	99.54%
<i>coastguard_obj_2</i>	119925	122755	2.36%	2036772	11714	99.42%
<i>coastguard_obj_3</i>	171030	181109	5.89%	1949582	11382	99.42%
Total	2653280	2879293	8.52%	51813755	218397	99.58%

Table 2.6 Performance of the proposed algorithm and the full search method when search window IS ± 16 and 255 classes with 6 classes overlapping.

Test Sequences	Bits/shape			# of MB processed (search positions)		
	<i>FS</i>	<i>Proposed</i>	<i>CHG_BIT</i>	<i>FS</i>	<i>Proposed</i>	<i>Saving</i>
<i>Foreman</i>	311720	314979	1.05%	7810280	272905	96.51%
<i>Stefan</i>	236964	238255	0.54%	6141676	259932	95.77%
<i>singer-247</i>	239246	239865	0.26%	6125008	250416	95.91%
<i>News</i>	298696	298912	0.07%	5863564	185674	96.83%
<i>dancer-247</i>	440828	444623	0.86%	9623385	431812	95.51%
<i>coastguard</i>	176666	177752	0.61%	2062352	94341	95.43%
<i>coastguard_obj_0</i>	401875	405486	0.90%	6066320	364255	94.00%
<i>coastguard_obj_1</i>	256330	258102	0.69%	4134816	207529	94.98%
<i>coastguard_obj_2</i>	119925	120353	0.36%	2036772	80475	96.05%
<i>coastguard_obj_3</i>	171030	172147	0.65%	1949582	92980	95.23%
Total	2653280	2670474	0.65%	51813755	2240319	95.68%

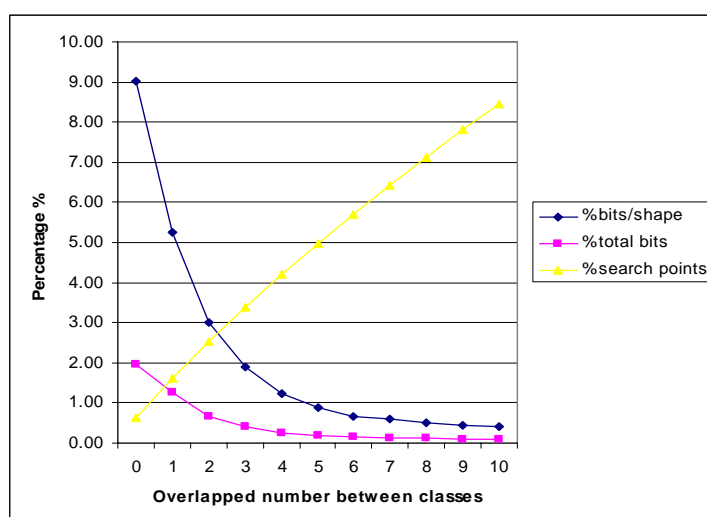


Fig. 2.13 The impact of classes overlapping on bits/shape, total encoded bit-stream and the number of search positions.

Table 2.7. Comparison between different classes.

Class type	Bits in each class	CHG_BIT for shape	CHG_BIT for total bits	Saving in Search positions
<i>16 class</i>	16	1.17	0.24	93.18
<i>32 class</i>	8	2.52	0.53	96.25
<i>64 class</i>	4	4.57	0.99	98.04
<i>256 class</i>	1	9.03	1.96	99.38

2.6.3 Consideration for the Classification and Matching Methods

The optimum classification of classes and class-overlapping are highly content dependent. For some test sequences, the probability of classes are not uniformly distributed (BABs belonging to a certain class are more probable than others). Fig. 2.13 shows the probability distribution for the 255-classes of the test sequence “container_2_obj”. It is clear that, BABs in which the number of “1” in the range 1-83 are more probable, among which the range 57-81 has higher occurrence. Thus, we can divide the intervals for each class according to the probability density such that, 1-56 to 8-classes (each class differs by 7 bits of “1” from the neighbor classes), 57-81 to 25-classes (each class differs by one bit “1” from the neighbor classes), and the remaining into 4-classes. We ran three tests for the same test sequence, as shown in Table 2.8. For the case-1 without overlapping between classes, we got a smaller number of search positions but larger bits/shape. In the case-2, we ran the same test with uniform overlapping between classes, and as expected this resulted in a higher number of search positions with fewer bits/shape. The case-3 compromises between reduction in search positions and bits/shape by overlapping only in the range of high probability, in the range of 57-81. We got lower search positions than the case-2, and lower bits/shape compared to case-1. By applying overlapping to those classes with more probable ones will refine the MVs with a little increase in search positions.

Table 2.8 Results for dividing classes according to the probability of container_2 test sequence.

Test	CHG_BIT for shape	Saving in SP
Case-1	16.45%	95.40%
Case-2	1.80%	81.70%
Case-3	3.14%	90.30%

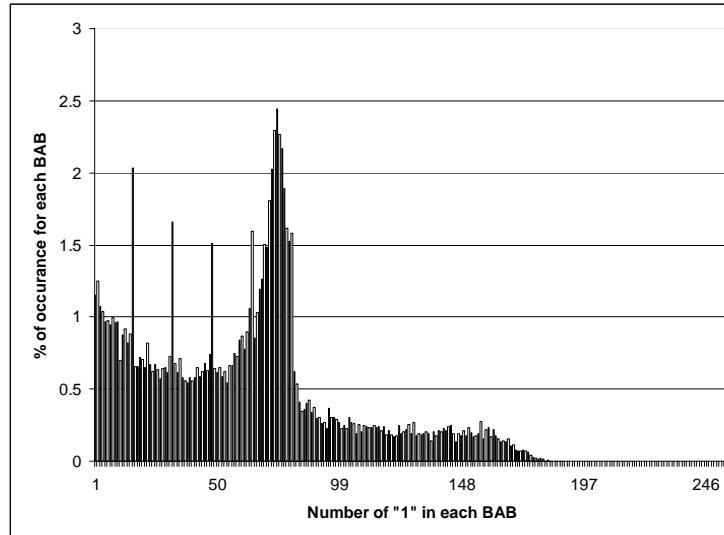


Fig.2.13 The probability distribution for the 256 classes in the container_2_obj.

Statistics in Fig. 2.13 can be calculated at the run time, by accumulating the occurrence of every class, overlapping those of high probability, and joining more than one class for those with less probability. The statistics can be made according to a “frame window,” such that, for a predefined number of frames (e.g., 10 frames window) we count the statistics and consider the results for the coming frames. This will be explored in a future work by dynamic class assignment and overlapping.

2.7 The Architecture Design.

2.7.1 Architecture Design

Fig. 2.14 shows the block diagram of BME architecture. Due to the simplicity and regularity of the proposed fast algorithm, the whole architecture is similar to the full search architecture presented in [14]. However, instead of full search, we adopt the fast algorithm but maintain the regularity of full search. The extra hardware needed is the modification to the accumulator to support addition and subtraction, and also extra registers to save the accumulated count for “1” in every BAB. The addressing and control unit is simple due to regular data flow. In Fig. 2.14 the search window buffer (SR buffer) stores partial search window data that can be reused by PE array to reduce data transfer from off-chip frame memory. The PE array contains 16 processing elements, and each can compute the SAD of one candidate BAB. Another function of the PE is to count the “1” within every candidate BAB.

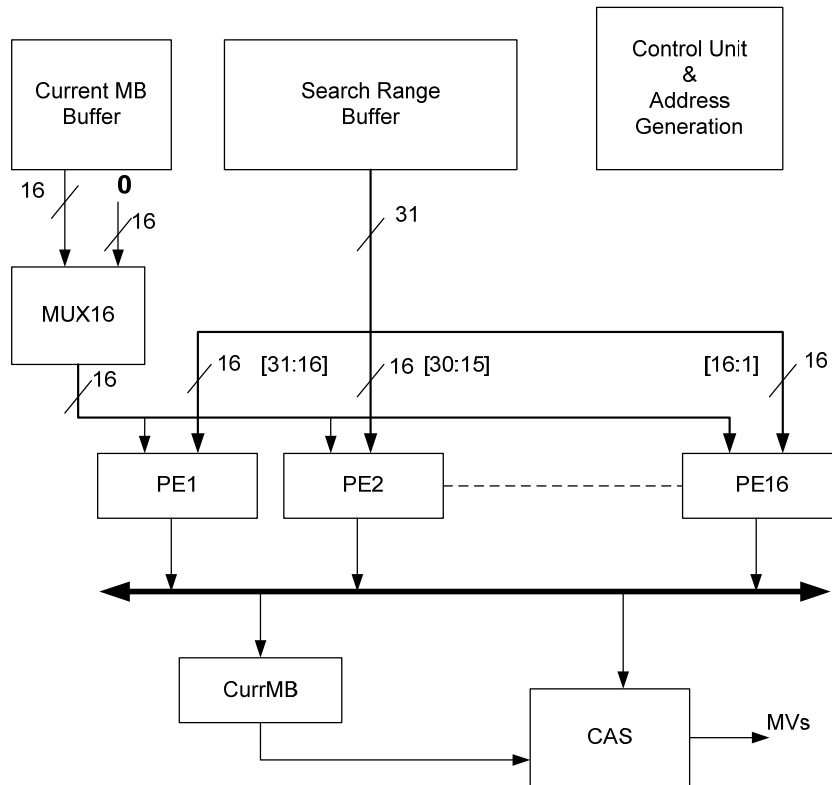


Fig. 2.14 Architecture block diagram of the proposed algorithm

Thus, a MUX (Multiplexer) will be used to select the operation for the PE between counting “1” and computing the SAD. Compare and select (CAS) module compares results of PE and selects the motion vector of minimal SAD. Control/Address generation (AG) module generates address for accessing SR buffer and control signals to other modules. Registers are used to store the count of “1” for each BAB in the SR, in which each register is 8-bit (enough to count up to 255).

2.7.2 PE Design

Fig. 2.15 shows the architecture of a single PE, it consists of an XOR circuit followed by an adder tree, and ended with an accumulator. The accumulator supports both addition and subtraction. For SAD computation, one row of current BAB and one row of candidate BAB are compared by bit-wise XOR. The resulted row of binary data represents the difference values between pixels of these two rows. The adder tree will sum up those binary data as partial SAD. The accumulator sums up 16 rows of partial SAD to obtain the SAD of one candidate position in SR (the SAD of one candidate BAB is produced every 16 cycles). In each PE there are two registers, one to save the partial SAD for later use as final SAD for that search position (`sad_reg`). The other register will hold the count of “1” pixels of candidate BAB (`count_reg`).

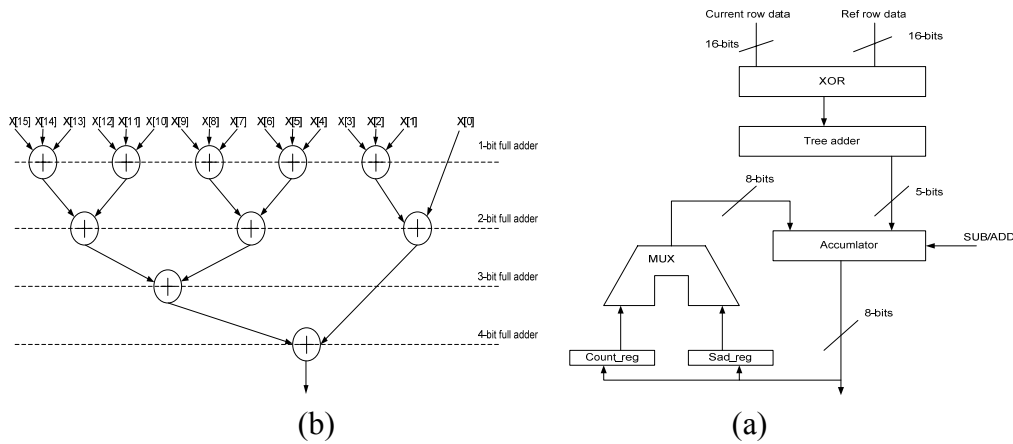


Fig. 2.15 (a) The SAD architecture, and (b) the tree adder.

2.7.3 Data reuse and data flow

Data reuse concept should be explored while reading from the search window buffer. Data redundancy exists in both directions, horizontally and vertically. The horizontal data redundancy is due to computing SAD for more than one adjacent candidate BABs in the search window buffer. The vertical data redundancy is due to counting “1” of adjacent candidate BABs. Since PE's in the PE array take responsibility for adjacent candidate BABs, the input data from search window for every PE have large redundancy. With data dispatching, we can achieve better maximal data reuse utilization. As shown in Fig. 2.14 data dispatch is implemented by hardwiring the desired reference data into each PE. Data [31:16] are dispatched to PE1, and data [30:15] are dispatched to PE2 and so on. The search window width is 48 pixels (for search range (-16, +15)), while the data fed into 16 PE are 31bits, which leaves 17 bits to be scanned. Thus, only two passes will be needed to cover the entire search window, as shown in Fig. 2.16. Each pass will cover part of the search window, supplying the 16 PE with proper data to do the match. To facilitate the counting of “1” and SAD computation, the proposed design adopts the sliding window approach to read the pixel from the search window by sliding vertically as shown in Fig. 2.17.

Sliding down in the search window will keep tracking of the “1” counting for every adjacent candidate BAB by adding a new row, and subtracting the top expired row. As we slide down by one row, we still make use of the remaining 15 rows (the area marked by crossed bars).

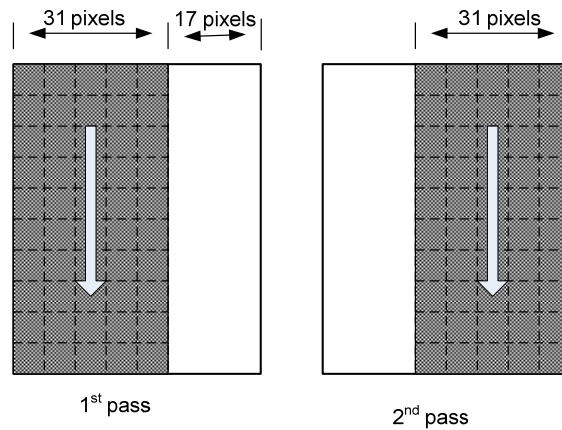


Fig. 2.16 Two passes needed to cover the search window

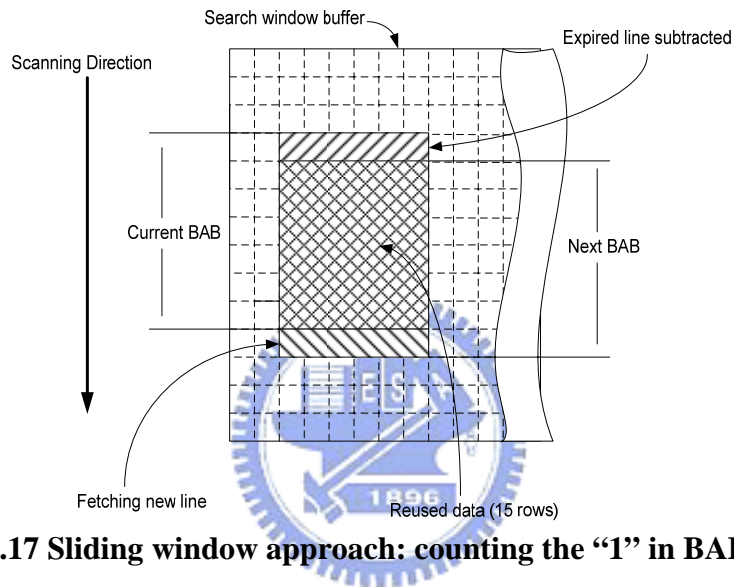


Fig. 2.17 Sliding window approach: counting the “1” in BABs.

This will just add an overhead of 2-clock cycles to keep tracking of the “1” count for every BAB (2-clock cycles rather than 16-clock cycles to perform full SAD computation). We can summarize the procedure of the architecture operation as follows:

- Step 1. Current BAB classification: count the number of “1” in the current BAB, and store the result into “CurrMB” register (this step needs 16-clock cycles).
- Step 2. Candidate BAB classification: start counting the number of “1” for 16 adjacent BABs within the search window, and store the results into each specified register (Reg1~Reg16). Each register located inside the PE as shown in Fig. 2.15(a) (this step needs 16 clock cycles).
- Step 3. Class match and SAD computation: The comparison circuit will classify the results stored in the registers and determine which one matches the current BAB class. If a match occurs, start calculating the SAD for that position only (16-clock cycles when a match occurs to compute the SAD).

- Step 4. Proceeding to new data: If there is no match, we proceed to the next row by the sliding window approach (2- clock cycles overhead to count the number of “1” in this way).
- Step 5. Repeat steps (4) and (3) to the end of the search window.

2.7.4 Experiments results

Since the architecture consists of 16-PEs working simultaneously, it is highly probable that more than one match could occur (two or more adjacent BAB belong to the same class), and hence performs the SAD computation for more than one match at the same time, as shown in Fig. 2.18. This will save the processing time since more than one match to be processed in one time slot needed to do one match. The whole design has been implemented in Verilog code and synthesized by Synopsis Design Compiler. The synthesized gate count for the architecture is 11582 for the total design, using 0.18 um cell library.

The required cycle count is quite low due to our simple scheme to skip unlikely search positions. The cycle count to perform one full search can be calculated as follows; as an initial step we need to count “1” for the current BAB (16 clock cycles) and for the first 16 rows in the search window buffer (16 clock cycles). Then proceeding by adding new row and subtracting the top expired row (Fig. 2.17), this will consume 2 clock cycles. Since we can scan the search window in two passes, so:

$$(2 \text{ clocks}) \times (2 \text{ passes}) \times (32 \text{ rows for each pass}) = 128 \text{ clock cycles}$$

So, 128 clock cycles are needed to scan the search window. When a match occurs, 16 clock cycles are needed to compute SAD for that match (i.e. we express the number of matches or search positions by #SP). The total clock cycles will be $(16+16+128+ (\# \text{ SP}) \times 16)$ cycles. These cycles are needed to find one BAB MV.

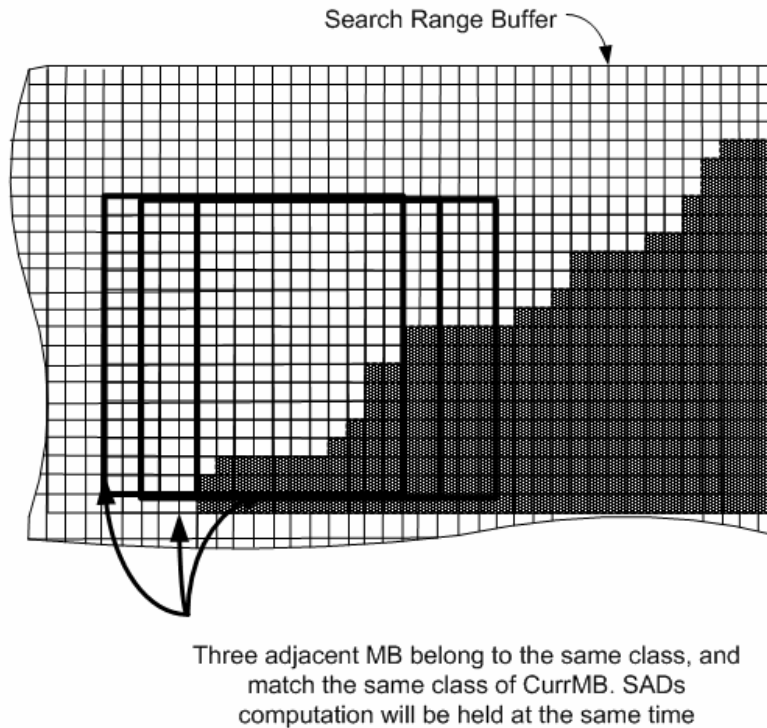


Fig. 2.18 Adjacent BABs belong to the same class, and match the same class of CurrMB. SAD calculation will be held at the same time in hardware implementation.

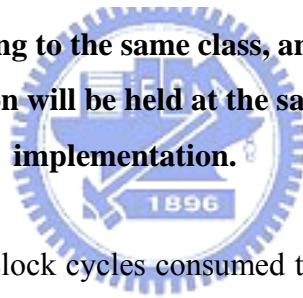


Table 2.9 gives the average clock cycles consumed to scan one search window for different classes overlapping. From Table 2.9, the average clock cycles to complete one frame in case of 32 classes overlapping (worst case) is 563. From which we can calculate the total clock cycles to complete one frame. Assuming the percentage of Boundary Alpha Blocks be 50% of the total alpha blocks (e.g., for CIF 352×288 , the boundary alpha blocks will be 198), we need $563 \times 198 = 111,474$ clock cycle to complete one frame. From the above calculations, the overhead of our algorithm will be as follows, 16 clock cycles to count the “1” of the current BAB, another 16 clock cycles to count “1” for the first search position in the search window, the later will be repeated twice, since we scan the search window twice. 2 clock cycles for every search position to add and subtract one row of pixels. Theoretically, the clock cycles needed to perform one full search window using one PE would be $(31 \times 31 \times 16 = 15376)$, and for 16-PE would be $(2 \times 31 \times 16 = 992)$ clock cycles, while the worst case in our design is 563 clock cycles.

Table 2.9 Hardware simulation results for different classes overlapping.

	32 classes overlapping	16 classes overlapping	no overlapping
test sequences	<i>AVE clk *</i>	<i>AVE clk *</i>	<i>AVE clk *</i>
<i>foreman</i>	506	443	276
<i>Stefan</i>	635	536	302
<i>singer-247</i>	626	497	258
<i>news</i>	543	474	269
<i>dancer-247</i>	595	506	276
<i>coastguard_obj_0</i>	410	354	250
<i>coastguard_obj_1</i>	555	483	289
<i>container_obj_0</i>	540	452	284
<i>container_obj_1</i>	553	503	331
<i>container_obj_2</i>	662	578	302
<i>container_obj_4</i>	570	475	245
<i>Average</i>	563	482	282

2.7.5 Comparisons

Table 2.10 and Table 2.11 show the comparison results between our proposed algorithm, and other fast algorithms. The proposed algorithm can achieve lower search positions and still has lower bit rate increase. Moreover, the flexibility of our proposed algorithm which lies in the classification of classes, and overlapping between classes, according to run time statistics, will benefit in tradeoff between reduction in search points and bits/shape. The comparisons are based on bits/shape and reduction in search positions. Bits/shape presented by [12] is based on WSAD (weighted SAD) which gives lower bit rate and different values than normal SAD implemented by MPEG-4 VM, even for full search algorithm. Besides, they employed the diamond search algorithm to minimize the number of search positions that is not regular and is not suitable for hardware design. The average reduction in search positions achieved by our proposed algorithm is larger compared to others. The average search position reduction in [13] is -90.95%, and that for [12] is -96.78%, while it varies from -96.59% to -99.69% for our proposed algorithm. The minor increase in bits/shape produced by the proposed algorithm is not much deviating apart from other fast algorithms. The software implementation of the proposed algorithm is complete bale to the algorithm presented in [13] which is a software approach, so our proposed algorithm is suitable for software and hardware implementation.

Table 2.10. CHG_SP for various search algorithms relative to the full search algorithm.

Sequence	Chen [14]	Yu [13]	Tsai [12]	Proposed	
				with 6 classes overlapping	without overlapping
<i>news</i>	46.04%	99.12%	96.74%	96.69%	99.71%
<i>foreman</i>	43.94%	82.78%	96.85%	96.49%	99.67%

Table 2.11. Average bit-rate for various search algorithms. All are relative to the full search algorithm.

Sequence	Chen [14]	Yu [13]	Tsai [12]*	Proposed	
				with 6 classes overlapping	without overlapping
<i>news</i>	0.00%	-0.74%	0.19%	0.07%	12.80%
<i>foreman</i>	0.00%	0.47%	-0.35%	1.05%	5.33%

* Weighted SAD

For hardware design comparison, the proposed algorithm is simple to be implemented in hardware and similar to the full search scheme. No special computation circuitry needed, which can make it switched to a full search without disabling any extra hardware. Control circuit and address generator is simple. BME architecture presented in [10] employs a full search algorithm, and needs a gate count of 9666 while operated at 7.29 MHz for core profile at level two. In comparison, our implementation needs slightly larger gate count of 11582 but needs fewer cycle count and lower frequency, only 3.34 MHz.

2.8 Summary

Shape coding introduced by MPEG-4 gives the ability to deal with arbitrary shape objects and adds more interactivity with the user. In which, binary motion estimation is the heart of the shape coding. The binary nature of the shape coding represents a burden on general purpose processors. These processors were designed to work efficiently with word level processing but not with bit level processing. Thus, we propose a fast algorithm to speed up the binary motion estimation process and its architecture.

The proposed algorithm eliminates unlikely candidate positions by counting the number of "1" contained in the BAB boundary. If the number of "1" counted doesn't

belong to the same class of the current BAB, it will be rejected; otherwise, it will be processed and matched. The saving in complexity ranges from 96.69% to 99.71% comes with the expense of increasing the shape encoded bits by 0.70 %to 12.80%.

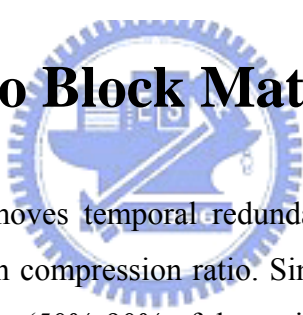
Due to the binary nature of the binary motion estimation, a dedicated hardware is proposed to serve this part of the shape coding module. With the simple and regularity of the algorithm, the proposed hardware is also regular and needs only 11582 gate count.



Chapter 3 Texture Motion Estimation for MPEG-4 and H.264.

The previous chapter explored the binary motion estimation, which is adopted by MPEG-4 as a novel approach to process individual objects. Texture motion estimation still the heart of video coding systems. During the last two decades, hundreds of fast algorithms and VLSI architectures have been proposed. As for hardware implementations, many architectures were derived to support either full search, fast search or both. In this chapter, a brief survey for the fast algorithms and architectures will be introduced. Then, the proposed algorithm and architecture will be introduced accordingly to show its performance compared to candidates in the same category.

3.1 Introduction to Block Matching



Motion estimation (ME) removes temporal redundancy within frames and thus provides coding systems with high compression ratio. Since ME module is usually the most computationally intensive part (50%-90% of the entire system) in a video encoder, efficient implementation of ME is a must. Block matching approach is mostly selected as the ME module in video codecs and is also adopted in all existing video coding standards because of its simplicity and good performance. The block matching algorithm (BMA) is shown in Fig. 3.1 and described as follows. Each luma frame is divided into blocks of size $N \times N$, and each block in the current frame is matched with candidate blocks of size $N \times N$ within the search area in the reference frame. The best matched block has the lowest distortion among all of the candidate blocks (the distortion is mostly evaluated by the sum of absolute differences (SAD)). The displacement of the best matched block, or namely the motion vector (MV) of the current block, will be transmitted with prediction residues to the decoder.

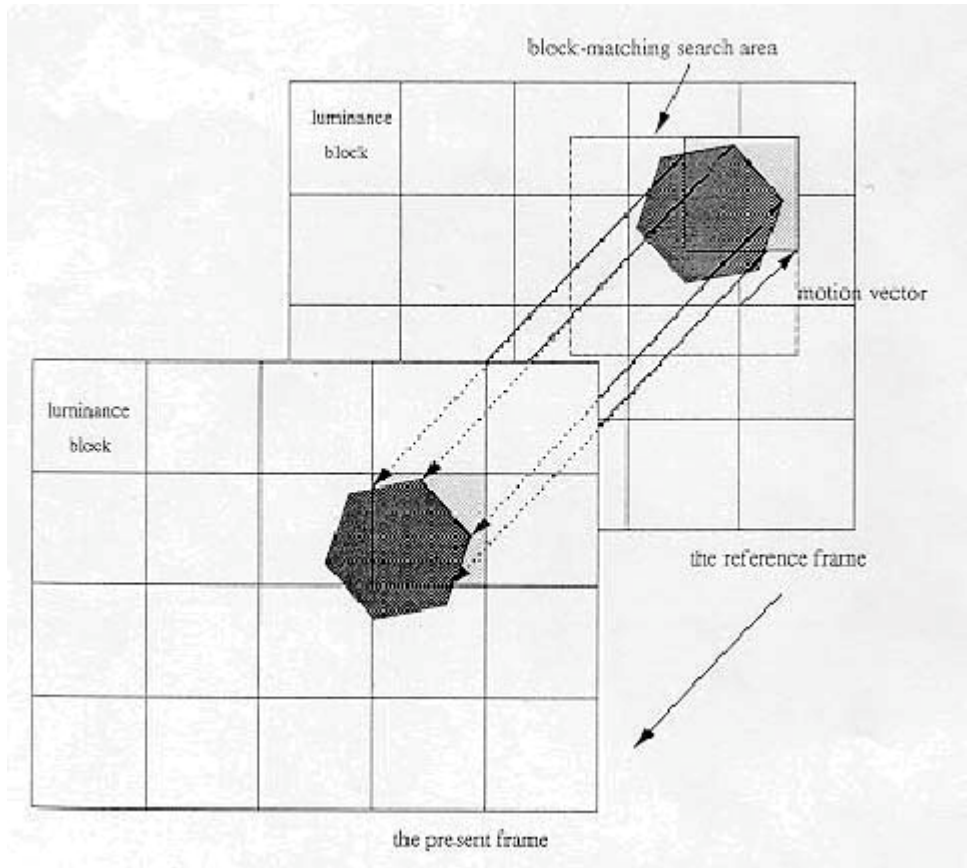


Fig. 3.1 Block Matching Algorithm.

Among all the BMA's, full search block matching algorithm (FSBMA) is the most popular. FSBMA can be described as:

$$SAD(m,n) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |c(i,j) - s(i+m, j-n)| \quad (3.1)$$

where $SAD(m,n)$ represents the distortion of the candidate block at search position (m,n) , $c(x,y)$ is the current block pixels, $s(x,y)$ stands for search area pixels, the search range is $[-p, p-1]$, the block size is $N \times N$, and MV expresses the motion vector of current block with minimum SAD among $(2p)^2$ search positions.

FSBMA demands a lot of computation. For example, real-time ME for CIF (352x288) 30 frames per second (fps) videos with search range as $[-16, +15]$ requires 9.3 giga-operations per second (GOPs). Clearly, such huge computational complexity is far beyond the processing capabilities of today's general purpose processors. Therefore, many fast algorithms and hardware architectures had been proposed.

This Chapter organized into two main parts; the first part explores the motion estimation algorithms, presenting the previous work and introducing the proposed algorithm, showing its performance compared to other related works. The other part

explores the architecture design for the motion estimation. Surveying previous works in this field, and proposing architecture to serve the proposed algorithm in the first part. The reasons for choosing this architecture, the implementation and performance will be shown accordingly. Finally, a summary will be made at the end of the chapter.

3.2 Exploration of Algorithms

As shown earlier, full search algorithm represents a heavy burden on the system. The need for reducing the computation time and complexity leads to many fast algorithms. Those fast algorithms adopted techniques to reduce the number of search points, or to reduce the computation complexity. This will come with the expense of reduction in picture quality, and lower coding efficiency. Many researches contributed to reduce the computational burden caused by the full search algorithm. Following are a brief survey for the previous works. For more information of the algorithms, please refer to [15].

Many algorithms explore the reduction in search positions, for example, two dimensional logarithmic search [16], three step search [17], conjugate direction search [18], modified logarithmic search [19], cross search [20], parallel hierarchical one dimensional search [21], one dimensional full search [22], new three step search [23], four step search [24], block-based gradient descent search [25], center-biased diamond search [26] [27], advanced diamond zonal search [28] [29], minimum bounded area search [30], one-dimensional gradient descent search [31], cross diamond search [32], predictive line search [33], kite cross diamond search [34], and many others, have been proposed.

To simplify the matching process many algorithms proposed different ways and techniques. For example, sub-sampling [35], pixel difference classification (PDC) [36] and integral projection was introduced in [37] and [38]. On the other hand, some algorithms reduced the matching process complexity by reducing the number of bits needed to represent single pixel. In [39] and [40], their algorithms involve transforming each pixel to one-bit representation and then applying conventional ME search strategies. In [41], they directly truncate the bit-width of pixels.

Apart from reducing the number of search positions or simplifying the complexity, an early-termination techniques applied to do full search, but skipping those unlikely matched candidate position early and save the computations. Successive elimination algorithm (SEA) [42], partial distortion elimination (PDE) [43] is simple and effective. In [44], an adaptive scanning order of pixels in a candidate block was proposed for distortion calculation to further speed up the PDE.

In fact, the results of fast full search are not exactly the same as FSBMA. Sometimes, minor differences occur. For example, when two or more search positions have the same minimum SAD, the result will be dependent on scan order. However, these minor differences do not cause noticeable effect on quality. Recently, in the new video coding standard, H.264/AVC, multiple reference frames and variable block sizes make the BMA much more complex, which becomes the hottest new topic of fast ME [75].

The proposed algorithm is a modification for the PDS, by applying an adaptive threshold rather than fixed one. The quality almost the same, but the computational burden greatly reduced with a little increase in encoded bit stream. Also, due to the regularity of data flow and simplicity of estimating the threshold value, make it easy to implement the algorithm into hardware without extra cost compared to full search architectures.

3.2.1 Proposed Algorithm-Software Approach

3.2.1.1 Introduction and Motivation

Early termination scheme is one way to reduce the complexity of the block distortion measure, which is our concern here. Partial distortion search algorithm (PDS) is recommended to be used in MPEG-4 and H.264 reference software to reduce the computational complexity of the SAD without introducing any loss in PSNR quality. In the PDS, the accumulated partial SAD (PSAD) is used to eliminate the impossible candidates of motion vector (MV) before the completion of calculating the SAD in a matching block. Thus, if the PSAD is greater than the current minimum SAD at anytime, this candidate is rejected and the remaining SAD computation is skipped. The PSAD is computed and accumulated by calculating the SAD for one line of the block at

a time. Though this scheme can skip some unnecessary SAD computations, there is still room to be improved. For example, this scheme cannot efficiently skip the SAD computations when the candidate SAD is similar to the current minimum SAD. This situation will happen especially at the search points near to the predicted or the final MV location.

Adjustable partial distortion search (APDS) [48] is a normalized partial distortion comparison method capable of adjusting the prediction accuracy against searching speed by a quality factor. It uses the halfway-stop technique with progressive partial distortions (PPD) to increase early rejection rate of impossible candidate MVs at very early stages. APDS divides each matching block into 16 equal sized groups, each group is sub-sampled in difference patterns, and each pattern has its own impact on the speed and the quality. Matching process starts by accumulating the distortion measure for one group after another. So by comparisons of the normalized partial distortion against normalized minimum block distortion, it can save more computational complexity.

Hilbert-grouped partial distortion search (HGPDs) [47] first employs the Hilbert scan to extract the representative pixels according to the edge information in the 1-D Hilbert sequence, groups them into 16 16-pixel groups, sorts these groups in descending order, and finally computes the partial distortion according to the order of the groups. It uses a new search strategy by scanning the search window twice. In the first scan phase it computes the distortion for the first group of the 16-groups established above. Then h-search points with the lowest PSAD are located and tested later to choose the best search center among them for the spiral scan which is the second scan phase. The overhead of the HGPDs is the grouping of pixels according to their activities in the Hilbert scan which consists of extra calculations and sorting also two scan phases for the new search strategy. So by doing the comparisons using the representative pixels first, this algorithm can save more computational complexity.

Both APDS and HGPDs exhibits irregular data flow due to extracting pixels in irregular pattern, which hinders their use in the hardware design. The PDS still consumes extra SAD computations especially for search points which are closer to the final MV location. H.264 introduces a variable block size ME, which reduces the efficiency of the early termination algorithms mentioned above. Small size blocks (i.e. 4x4) contain small number of pixels, in which the coherency between those pixels are high. So when employing the early termination algorithm, each block will consume more SAD computations till the termination happens. Repeating this process for all

small size blocks mean more SAD computations when compared with normal 16x16 MB.

To solve above problems, we propose an early termination algorithm by adaptive threshold instead of using nearly constant minimum SAD value during the SAD accumulation. The results show that it can reduce the SAD computation significantly with negligible quality degradation. The proposed algorithm shows good efficiency with variable block size. Also it exhibits regular data flow, since it tests the macroblock line by line.

3.2.1.2 Early Termination Algorithm

Fig. 3.2 (a) shows the flowchart of computing the PDS. It assumes a minimum SAD (SAD_{min}), by computing the SAD_{pred} at the predicted location in the search window, and sets that value to be SAD_{min} for later comparison. Then it starts the SAD computation for each search point. During the SAD computation, if the partial SAD is equal to or greater than SAD_{min} , it terminates the remaining partial SAD computation and jumps to the next search position. For such early termination purpose, the partial SAD is computed one line at a time and accumulated to the total SAD for that MB. This method can quickly skip the unnecessary SAD computation and preserve the search quality. However, if the SAD of the candidate MB is similar to the current minimum SAD, there is little room to save more computations.

Fig. 3.2 (b) shows the flow of the proposed algorithm. Our proposed algorithm adopts the similar approach as the PDS. We still compute the partial SAD one line at a time and add it to the total SAD. However, during the accumulation of SAD, we use a threshold value SAD_{TH} instead of the minimum SAD (SAD_{min}) as an early termination condition. If the accumulated SAD value is larger than the threshold, we skip the remaining SAD computation and proceed to the next search point. Otherwise, we continue the SAD computation/accumulation, and update the SAD_{TH} accordingly. When a match occurs, which means lower SAD value than the current minimum SAD, SAD_{TH} is modified according to the new value of minimum SAD.

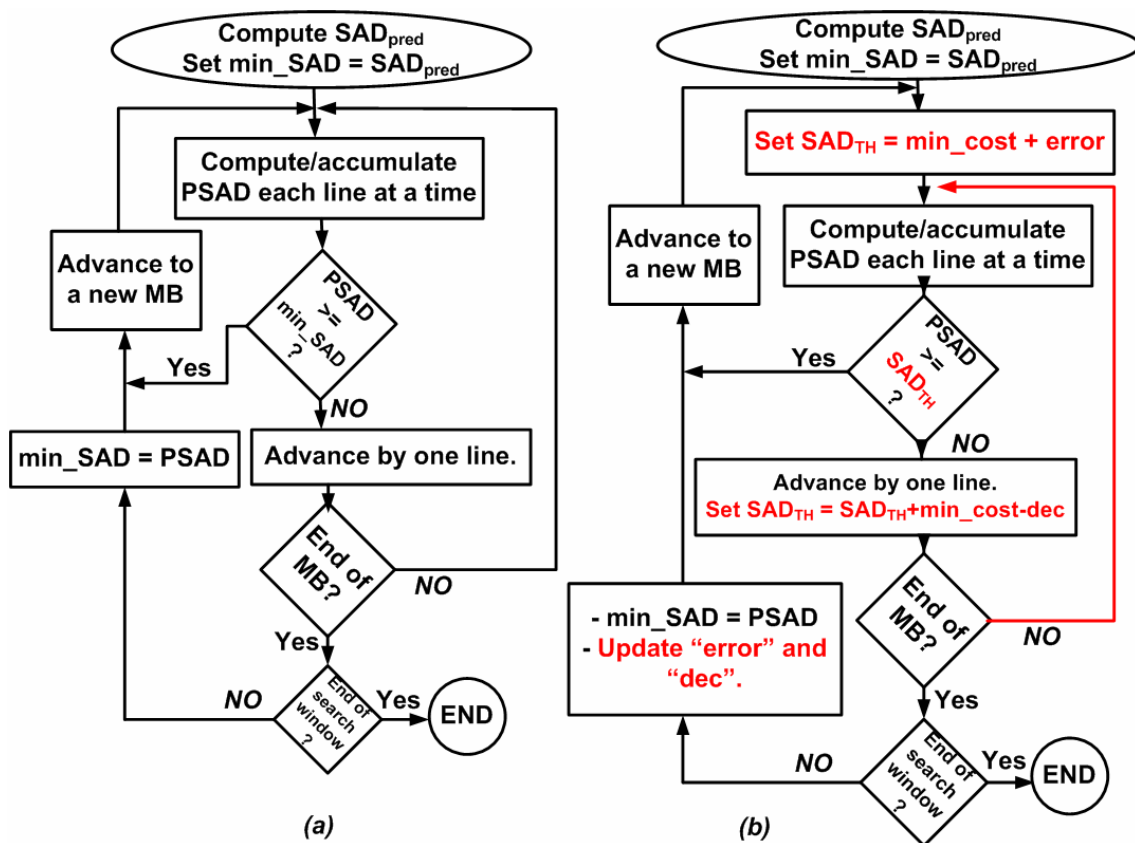
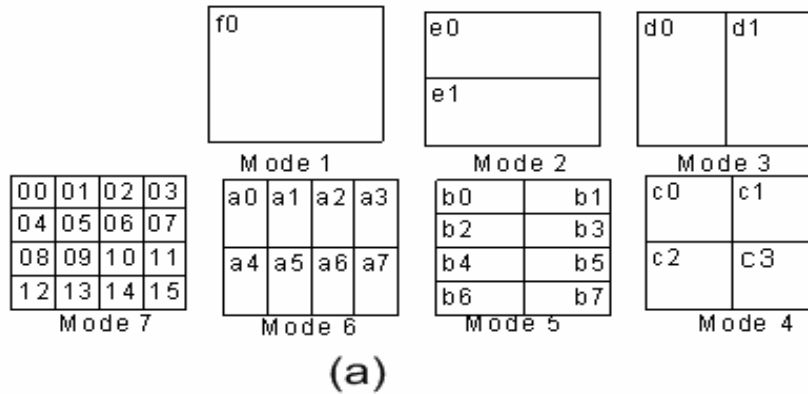


Fig. 3.2 (a) algorithm flow for the PDS, and (b) our proposed algorithm.

To help adaptively changing the threshold value, three parameters are introduced in the flow diagram, “ SAD_{TH} ”, “error”, and “dec”. The values for SAD_{TH} , “error” and “dec” for H.264 and MPEG-4 are depicted in Fig.3.3 (b). These parameters have considered the effect of variable block size and thus have different values according to their block size. The parameter “ SAD_{TH} ” represents the base threshold value used in early termination process, which is a summation of current minimum SAD and extra error margin (error). The reason to introduce the “error” parameter is explained as follows. Since we process every candidate MB line by line; each line consists of 16 pixels in the case of 16x16 block size. Some lines may generate large PSAD, and others may generate low PSAD. The accumulated PSAD for the 16-lines will represent the SAD value for that candidate position. While comparing the PSAD for every line with the normalized SAD_{TH} , it could be the case of $PSAD > SAD_{TH}$ for these lines and terminates the SAD process.



```

H.264
if(mode== 1)      {min_cost= SADmin /16; error = 64; dec=4; }
if(mode== 2)      {min_cost= SADmin /8;  error = 32; dec=4; }
if(mode== 3)      {min_cost= SADmin /16; error = 32; dec=2; }
if(mode== 4)      {min_cost= SADmin /8;  error = 16; dec=2; }
if(mode== 5)      {min_cost= SADmin /4;  error= 8;  dec=2; }
if(mode== 6)      {min_cost= SADmin /8;  error= 8;  dec=1; }
if(mode== 7)      {min_cost= SADmin /4;  error= 4;  dec=1; }
SADTH = min_cost + error;

MPEG-4
min_cost = min_SAD/16; error = 64; dec=4;
SADTH = min_cost + error;

```

(b)

Fig. 3.3 (a) Seven macroblock modes in H.264/AVC, (b) values for SAD_{TH}, error, and dec for different block sizes in H.264 and that for MPEG-4

Meanwhile, if we proceeded to compute the PSAD for the remaining lines (those lines may generate low PSAD), the total SAD for that candidate position is lower than the global min_SAD. To give the chance for those candidate positions with inhomogeneous PSAD distribution among the MB lines, we introduce an error margin. If the generated PSAD for those lines are less than “error” we give them more chance to proceed in the SAD process. Otherwise terminate the process if the generated PSAD is larger than “error”.

The “error” parameter represents a margin of allowance for the candidate macroblock to pass the test in case that macroblock is similar or near to the target macroblock. The “error” parameter gives it the chance to continue the SAD test unless it really generates PSAD much greater than SAD_{TH}. This “error” parameter should be decreased line by line by using “dec” parameter (decrement), every time we process a new line by

subtracting “dec” from SAD_{TH} . Thus, when processing the last line in the candidate MB, we will compare the true min_cost with the accumulated SAD for that position.

The value for “error” is set to the difference of 8-pixels, if this value increases, more accurate MV will be generated on the expense of increasing the SAD computations. On the other hand, lower “error” value will generate more error in the generated MV, but less SAD computations. Fig 3.4 shows the effect of changing the “error” parameter on the generated bit rate, the PSNR, and the reduction in complexity, applied on three test sequences Stefan, Coastguard, and Mobile, which exhibit high and low motion. In Fig 3.4 (a), it is clear that as the “error” value increases, the change in bit rate will decrease, which means more accurate generated MV. In Fig 3.4 (b), when the “error” parameter increases the quality will be better and closer to that of the full search without termination.

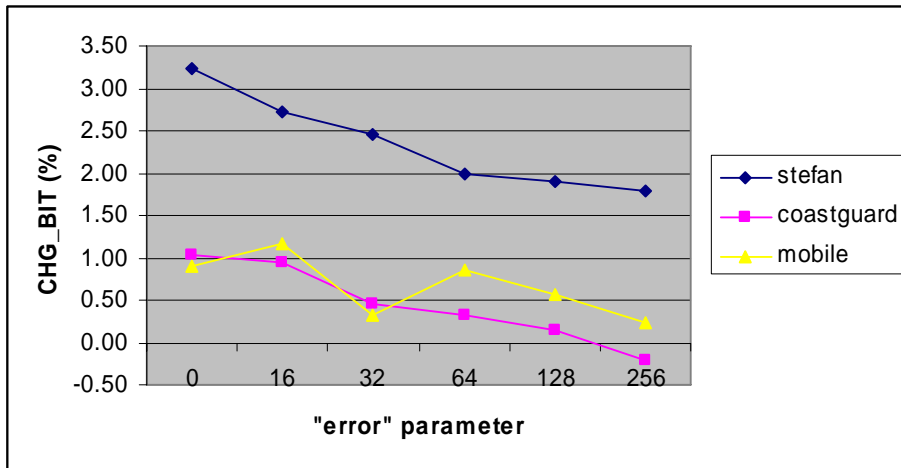
Finally the complexity reduction variation due to the “error” parameter is shown in Fig 3.4 (c), in which more computations could be saved when the “error” parameter is smaller. The “dec” parameter is simply the result of dividing the “error” by the number of lines in each sub-macroblock according to each mode we are working on.

In summary, the advantage of our algorithm is that with the adaptive threshold value instead of the constant one allows us to skip more unnecessary SAD computation, as shown later in the simulation results.

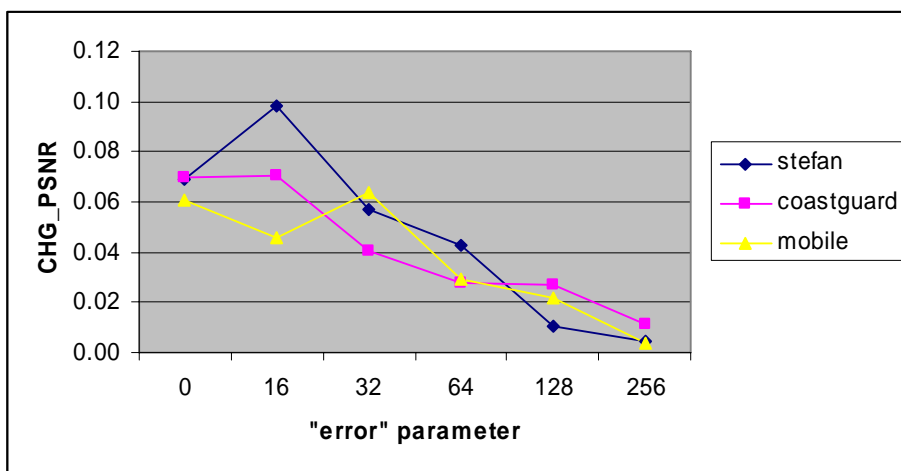
3.2.1.3 Simulation Results

In the following, we will show the simulation results of the proposed algorithm by integrating it into the MPEG-4 verification model V18.0, and H.264 JM 9.0. All the following test sequences are in CIF format with 300 frames and ± 32 search range, unless otherwise specified.

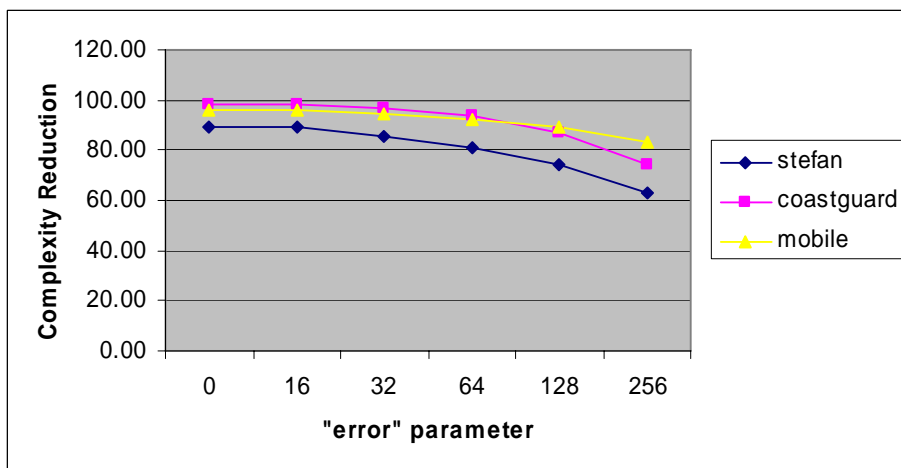
Like other early termination algorithms, the efficiency of the proposed algorithm depends on the scan pattern in the search window, and the initial SAD_{TH} value that affects how fast we can reach the minimum SAD point and save more computations. In the following test, we test two scan patterns as depicted in Fig. 3.5, the spiral scan adopted by MPEG-4 VM18.0 and the normal raster scan used for most of the hardware implementations. The complexity saving is due to the adaptive threshold mechanism during the SAD accumulation. SAD_{TH} should be set to a value that ensures fast early termination and gives accurate results as well.



(a)



(b)



(c)

Fig. 3.4 The effect of "error" parameter variation on (a) generated bit rate (b)PSNR, and (c) complexity reduction.

When SAD_{TH} adapted to a lower value, more line skipping will occur. Only those MBs near to the lowest SAD point will consume more SAD computations because the error between those MBs and the current MB will be smaller, and thus many lines will pass the threshold detection. Setting SAD_{TH} to large value will slow down the termination process, especially for the raster scan method. In the following testing results, the first SAD_{TH} is set to be SAD_{pred} , which is the SAD calculated at the predicated MV position.

Table 3.1 shows the results for both scanning methods, relative to the search algorithm implemented in VM18.0 without any early termination. The comparison results are produced and tabulated according to the parameters as below:

CHG_BIT	Change of bits used for the whole sequence.
CHG_PSNR	Change of PSNR.
CHG_COMPLEXITY	Change of SAD computations for the whole sequence.

It is clear that the spiral search pattern achieves better results in terms of total bits and PSNR, and also saves more SAD computations than the raster scan method. This is due to the raster scan pattern will pass by many local minimum, changing SAD_{TH} accordingly and may skip the targeted MB. This effect is shown clearly in the low and moderate motion test sequences, such as M&D and Mobile, where the MV is more likely to be near to the predicated position.

The proposed early termination method is also applied to the 8x8 sub-block motion estimation. Table 3.2 shows the performance results of the proposed algorithm compared with MPEG-4 VM 18.0 enabling the 4MV option. Our method can save 77.91% of complexity with negligible quality loss. The last column shows the time consumed by the ME subroutine to find the MV. This is computed with the *time* function implemented by Visual C++ 6.0, on a P4 2GHz, 768 MB memory and windows XP system.

The same tests are applied to H.264. Table 3.3 shows the results for the proposed method compared to the early termination method employed in JM 9.0. The proposed method can save 50.6% of complexity with similar quality. The difference in percentage between the time saving and the complexity saving is due to the accuracy of the *time* function, since many search positions will be skipped after few lines tests, and the *time* function will not capture this time difference. The difference appears more in H.264 since it works with small block sizes, and more error in capturing the time.

The result for H.264 shows less complexity reduction and more PSNR degradation but less bit rate increase compared to MPEG-4. This is due the variable block size ME of H.264. The small block size tends to consume more SAD computations compared to that of large block size. For example when testing 4x4 block, at least we need to go through one line test, which in total equivalent to 4 lines test in a 16x16 block. Meanwhile, if the tests only applied on 16x16 block, we can stop SAD computations after the first line test. The PSNR degradation is due to reducing the chances to select modes with smaller size blocks, which gives better quality when selected. When dealing with small size blocks, the “error” parameter also will be small, hence early termination will occur frequently, and will result in ignoring more small size blocks from the mode selection later by H.264.

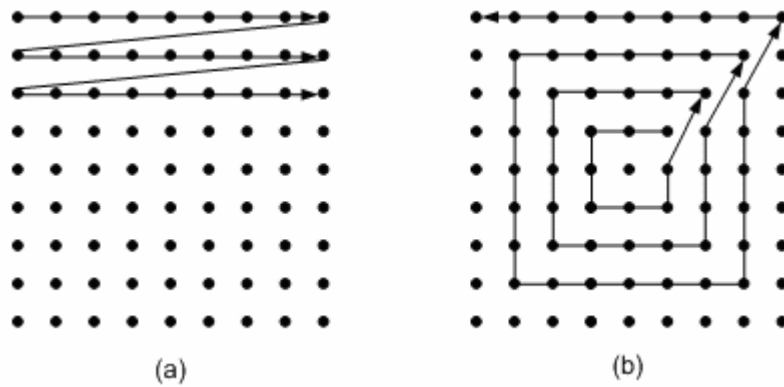


Fig. 3.5 Scan patterns in the search window, (a) raster scan , and (b) spiral scan.

Table 3.1 Experimental results for the proposed method with $Q_p = 16$.

Test sequence	CHG_BIT (%)		CHG_PSNR		CHG_COMPLEXITY (%)	
	raster scan	spiral scan	raster scan	spiral scan	raster scan	spiral scan
Foreman	1.77	1.04	-0.01	-0.01	-75.30	-86.10
Stefan	2.53	2.35	0.00	0.00	-76.20	-87.40
News	1.30	1.02	0.00	0.02	-74.90	-91.90
Coastguard	0.96	0.51	-0.01	-0.01	-78.40	-91.10
M&D	1.28	0.07	-0.04	0.01	-70.80	-86.60
Mobile	2.21	0.90	-0.01	-0.01	-81.80	-91.80
<i>Total/Average</i>	1.68	0.98	-0.01	0.00	-76.23	-89.15

3.2.1.4 Performance comparisons

The comparison hereafter is based on the SAD computational complexity and the PSNR value. When compared with other early termination algorithms such as [47] and [48]. The algorithm presented in [47] is considered as lossless due to no degradation in PSNR with complexity reduction of 16.6% compared to PDS. However, when it employs the early jump out technique it will result in -0.033dB PSNR reduction in average and 64.5% complexity reduction. While the algorithm presented in [48] can achieve a speedup of 18.7 times compared to PDS by testing one pixel at a time, which result in a high PSNR degradation of -0.885dB. The presented algorithm is superior to the above two algorithms in terms of negligible PSNR loss (in average -0.0018dB), and also in higher complexity reduction (77.914% compared to PDS). Comparing to [48], our algorithm can achieve a theoretical speedup of 64 times by testing 4-pixels at a time rather than 16-pixels (skipping the SAD computations after testing 4-pixel only).

For other non-early termination algorithms, Table 3.4 presents a comparison between our proposed algorithm with recently proposed GEA and QME. It is clear that the proposed method can save more computations with negligible quality loss.

The simplicity and regularity of the proposed algorithms makes it suitable for hardware implementation, with low control circuitry, regular data flow, and no special circuitry to read pixels in certain form (as Hilbert Scan or different block patterns employed in[48]). Also it can be integrated with other fast algorithms such as QME to achieve more computation reduction since its PSNR degradation is very low.

Table 3.2 Experimental results for MPEG-4 with Spiral scan pattern, Qp = 16, and 4mv enabled.

Test sequence	Bit rate			PSNR		
	VM 18.0	Proposed	CHG_BIT (%)	VM 18.0	Proposed	CHG_PSNR
Foreman	1806074	1818842	0.71	31.8644	31.8641	-0.0003
Stefan	8023412	8157877	1.68	28.4128	28.4085	-0.0043
News	1109574	1113001	0.31	31.9215	31.9281	0.0066
Coastguard	3522424	3534403	0.34	29.1339	29.1293	-0.0046
M&D	668805	669789	0.15	34.0386	34.0355	-0.0031
Mobile	7941210	7970559	0.37	26.3518	26.3463	-0.0055
<i>Total/Average</i>	27349747	27521063	0.63			-0.0018
	Complexity			Timing (ME time)		
	VM 18.0	Proposed	CHG_COMPL-EXITY (%)	VM 18.0	Proposed	CHG_Time (%)
Foreman	7.19E+08	2.02E+08	-71.919	114	58	-49.1
Stefan	8.02E+08	2.66E+08	-66.853	173	74	-57.2
News	1.48E+09	1.65E+08	-88.882	156	49	-68.6
Coastguard	8.37E+08	2.12E+08	-74.716	116	36	-69.0
M&D	1.38E+09	2.75E+08	-80.062	192	94	-51.0
Mobile	6.06E+08	1.39E+08	-77.081	117	32	-72.6
<i>Total/Average</i>	6.43E+09	1.42E+09	-77.914	868	343	-60.5

Table 3.3 Experimental results for H.264 with spiral scan pattern, Qp = 28, ±32 SR, and RDO disabled.

Test sequence	Bit rate			PSNR		
	JM	Proposed	CHG_BIT (%)	JM	Proposed	CHG_PSNR
Foreman	3223016	3240832	0.55	37.01	36.97	-0.04
Stefan	13694704	13710984	0.12	35.4	35.37	-0.03
News	2228896	2244720	0.71	38.09	38.07	-0.02
Coastguard	11279688	11231760	-0.42	34.52	34.51	-0.01
M&D	1382624	1382808	0.01	38.93	38.92	-0.01
Mobile	17495736	17458936	-0.21	33.76	33.74	-0.02
<i>Total/Average</i>	49304664	49270040	-0.07			-0.02
	Complexity			Timing (ME time)		
	JM	Proposed	CHG_COMPL-EXITY (%)	JM	Proposed	CHG_Time (%)
Foreman	3.07E+10	1.66E+10	-45.89	1841	1111	-39.7
Stefan	5.43E+10	2.74E+10	-49.47	3686	1986	-46.1
News	2.83E+10	1.52E+10	-46.35	1694	1118	-34.0
Coastguard	6.78E+10	3.08E+10	-54.49	4165	1981	-52.4
M&D	2.86E+10	1.40E+10	-50.92	1697	1099	-35.3
Mobile	5.37E+10	2.60E+10	-51.59	3835	2044	-46.7
<i>Total/Average</i>	2.63E+11	1.30E+11	-50.60	16918	9339	-44.8

Table 3.4 Comparison for different algorithms

Test sequence	CHG BIT		
	<i>GEA</i> ($\Delta\%$) [45]	<i>QME</i> ($\Delta\%$) [46]	<i>Proposed</i> (%)
Foreman	0.77	0.41	1.20
Stefan	0.85	0.15	1.87
News	0.45	0.34	0.67
Coastguard	0.95	0.09	0.31
M&D	0.67	0.48	0.61
Mobile	0.29	0.12	0.48
	CHG PSNR		
	<i>GEA</i> (ΔdB) [45]	<i>QME</i> (ΔdB) [46]	<i>Proposed</i> (ΔdB)
Foreman	-0.0222	-0.0059	-0.0046
Stefan	-0.0317	-0.0123	-0.0077
News	-0.0384	-0.0176	-0.0062
Coastguard	-0.0018	-0.0003	-0.0058
M&D	-0.0665	-0.0234	0.0112
Mobile	-0.0027	0	-0.0078
	CHG COMPLEXITY		
	<i>GEA</i> ($\Delta\%$) [45]	<i>QME</i> ($\Delta\%$) [46]	<i>Proposed</i> ($\Delta\%$)
Foreman	-66.41	-60.49	-83.88
Stefan	-69.40	-67.41	-85.37
News	-39.09	-66.86	-91.31
Coastguard	-72.92	-65.21	-87.92
M&D	-58.99	-39.86	-85.56
Mobile	-73.24	-59.36	-90.17

3.3 Exploration of Architectures

The motion estimation process represents a heavy burden on any general purpose processor. One feasible solution to reduce this burden is to map the motion estimation algorithm to a dedicate hardware, which works in parallel with the general purpose processor. This dedicate hardware may support one or more algorithms. Many researchers worked on this issue, and many hardware implementations for different algorithms were presented to support full search and fast search algorithms. In this subsection, a brief survey of different architectures will be presented followed by the architecture implementation for the proposed algorithm mentioned above. Comprehensive survey for the architectures can be found in [15] and [75].

3.3.1 ME Architectures – an overview

Many FSBMA architectures were developed due to the regularity of data flow. Most of them belong to systolic arrays [49] composed of locally connected processing elements (PEs). In [50], Komarek and Pirsch contributed a detailed systolic mapping procedure to derive FSBMA architectures. In [51], Vos and Stegherr proposed a 2-D semi-systolic array with an adder tree. In [52], Yang, Sun, and Wu implemented the first VLSI motion estimator in the world. In [53], a powerful FSBMA chip with 1024 PEs to provide computational capability of 165GOPS was designed.

Fast ME algorithms can reduce the heavy computation burden of FSBMA with acceptable video quality loss. The challenges of architecture design for fast ME algorithms include unpredictable data flow, irregular memory access, difficult mapping to systolic arrays, low hardware utilization, and sequential procedures with data dependence that cannot be parallelized.

In [54], Jong, Chen, and Chiueh developed a fully pipelined parallel architecture for the three step search BMA. In [55], Dutta and Wolf modified the data flow of the 1-D linear array in [52] to support FSBMA, three step search, and conjugate direction search on the same architecture. In [56], Lin, Anesko, and Petryna proposed a joint algorithm-architecture design of a programmable motion estimator chip. Various algorithms are implemented through a search strategy with macro-commands that can be executed efficiently on the chip. Many designed can be found in [57], [58], [59], [60].

3.3.2 Proposed Architecture

3.3.2.1 Hardware oriented modification

The hardware design should consider three issues: supporting MPEG-4 and H.264 simultaneously, supporting variable block size ME simultaneously, and motion vector predictor. The first issue is easily to be solved by supporting all modes of H.264 since block size of MPEG-4 is only a subset of H.264. The second issue can be solved by accumulating the SAD of the smaller block size to generate the SAD of larger block size, as adopted in many designs [61] [62]. By adopting this approach, the motion vector predictor within the macro block will depend on each other since predicted MV

is the medium value of the MVs of the top left, top, and top right, as addressed in [61] [62]. Thus, in our architecture the exact MV predictor for all kind of sub-blocks modes is replaced by that of mode 1 (16x16). This constraint will result in a slightly degrade in PSNR and increase in encoded bit stream. Table 3.5 shows the results when emulating the hardware by setting the MV predictor to mode 1(16x16), also accumulating the SAD of the smaller block size to generate the SAD of larger block size.

The early termination will be held whenever the partial SAD for mode 1 is equal to or greater than SAD_{TH} . The termination will also hold for other modes, which will result in error in determining the correct MV for each mode. This can be compensated by disabling the termination process if the tested lines of the MB are more than predefined count. Table 3.6 shows the effect of disabling the termination at different number of lines. The results are compared to the hardware results for PSNR, and encoded bit stream, not to the original JM9.0. Our proposed hardware implementation will terminate whenever the PSAD is greater than their accumulated threshold SAD_{TH} and will start at the next search point.

Table3.5 The results of applying the hardware (H/W) conditions to JM9.0, $Q_p = 28$, ± 32 SR, and RDO disabled

Test sequence	PSNR			Bit Rate		
	<i>JM9.0</i>	<i>H/W emulation</i>	<i>CHG_PSNR</i>	<i>JM9.0</i>	<i>H/W emulation</i>	<i>CHG_BIT (%)</i>
foreman	36.24	36.21	-0.03	4996192	5043360	0.9
Stefan	35.39	35.38	-0.01	13831760	14094144	1.9
news	38.1	38.09	-0.01	2244480	2302520	2.6
coastguard	34.53	34.5	-0.03	11297480	11178392	-1.1
M&D	38.95	38.95	0	1384408	1389800	0.4
mobile	33.76	33.72	-0.04	17524048	17659752	0.8
Total			-0.02	51278368	51667968	0.8

Table 3.6 The effect of disabling termination after certain number of lines compared to the hardware simulation results.

# of lines, after which no termination	<i>CHG_PSNR</i>	<i>CHG_BIT (%)</i>	<i>CHG_COMPLEXITY (%)</i>
10 lines	0.01	-1.17	-61.14
11 lines	0.00	-0.94	-62.45
12 lines	0.01	-0.98	-63.41
13 lines	0.01	-0.75	-64.03
14 lines	0.01	-0.60	-64.44

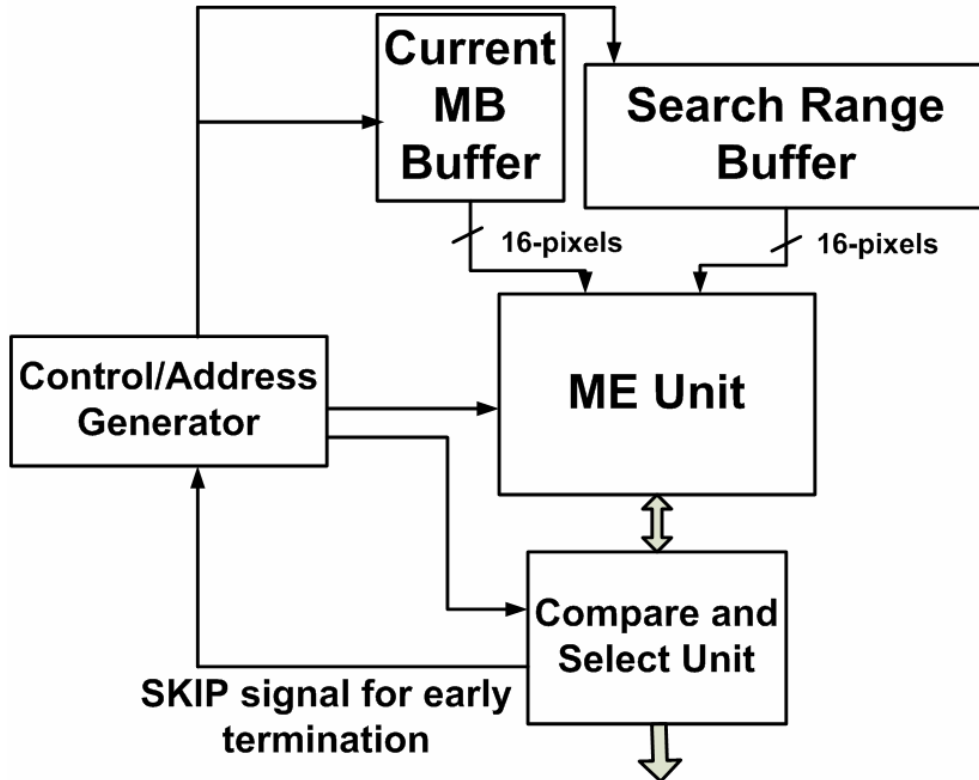


Fig. 3.6 Block diagram of the proposed architecture.

3.3.2.2 Architecture design

The simplicity of the algorithm leads to a simple and an efficient hardware implementation which employ the full search scan. Since the early termination scheme relies on testing one line of the MB at a time, the proposed architecture adopts the 1-D array architecture for SAD computation. The block diagram of the proposed architecture is shown in Fig. 3.6. The current MB buffer and search range buffer store the data to reduce external memory communication. The ME unit will do the SAD computation for all sub-blocks at the same time, and sends the results to the compare and select unit (CS). When a termination occurs, a ‘SKIP’ signal will be generated and sent back to the control unit, which will send a signal to ME unit to flush all registers, and for the search range buffer to advance by one MB.

Fig. 3.7 shows the circuit diagram of the ME unit, it consists of 16-absolute difference (AD), an optimized shared adder tree, accumulators to accumulate the SAD for each sub-micro block (AC0 to AC8), a total of 16 SAD registers to save the accumulated SAD for each mode as will be shown later. As shown in Fig. 3.8, one line of SAD is part of 4-sub block of mode 7 (4x4), 4-sub blocks of mode 6 (4x8), 2-sub blocks of mode 5 (8x4), 2-sub blocks of mode 4 (8x8), 2 sub blocks of mode 3 (8x16), 1-sub

block of mode 2 (16x8) and finally one block of mode 1 (16x16). So the architecture serves the number of registers accordingly. The registers R70 ~ R73 will accumulate the SAD for mode 7 (sub blocks 00 ~ 15 as shown in Fig. 3.3 (a)), while the registers R60 ~ R63 will accumulate the SAD for mode 6, registers R50 and R51 will accumulate the SAD for mode 5, and so on for the other registers. The size of every register depends on the sub-block it serves by taking into account the maximum SAD could be resulted. Registers R70 ~ R73 are 12-bits, R60 ~ R63, R50 and R51 are 13-bits, R40 and R41 are 14-bits, while R30, R31, and R20 are 15-bits, and finally R10 is 16-bits register.

As shown in Fig. 3.7, the current and reference data are 16-pixeles inputted in parallel at a time to the AD unit. The resulted partial SAD from each AD will be added by the optimized shared tree adder and end up with SAD of one line. Accumulators AC0 to AC3 will accumulate the SAD for mode 7 and mode 6, while AC4 and AC5 will accumulate the SAD for mode 5. AC6 and AC7 will accumulate the SAD for modes 4 and 3, while AC8 accumulates for modes 2 and 1.

The resulted SAD for every sub-block of every mode will be submitted to the compare and select (CS) circuit to generate the final motion vector for each sub-block of every mode. The sub-blocks of mode 7 and mode 5 will be generated every 4 clock cycles, while that of mode 6, 4 and 2 will be generated every 8 clock cycles, and finally mode 3 and 1 will be generated every 16 clock cycles. Accordingly the data stored in registers R70~R73, R50 and R51 should be submitted to CS unit and reset the registers to zero in order to start accumulate SAD for the new sub-blocks every 4 clock cycles. Registers R60~R63, R40, R41 and R20 should be submitted to CS unit, and reset the registers to zero every 8 clock cycles. Registers R30, R31 and R10 will submit the SAD accumulated to CS unit and reset to zero every 16 clock cycles. Table 3.7 shows the modes generated every 4 clock cycles.

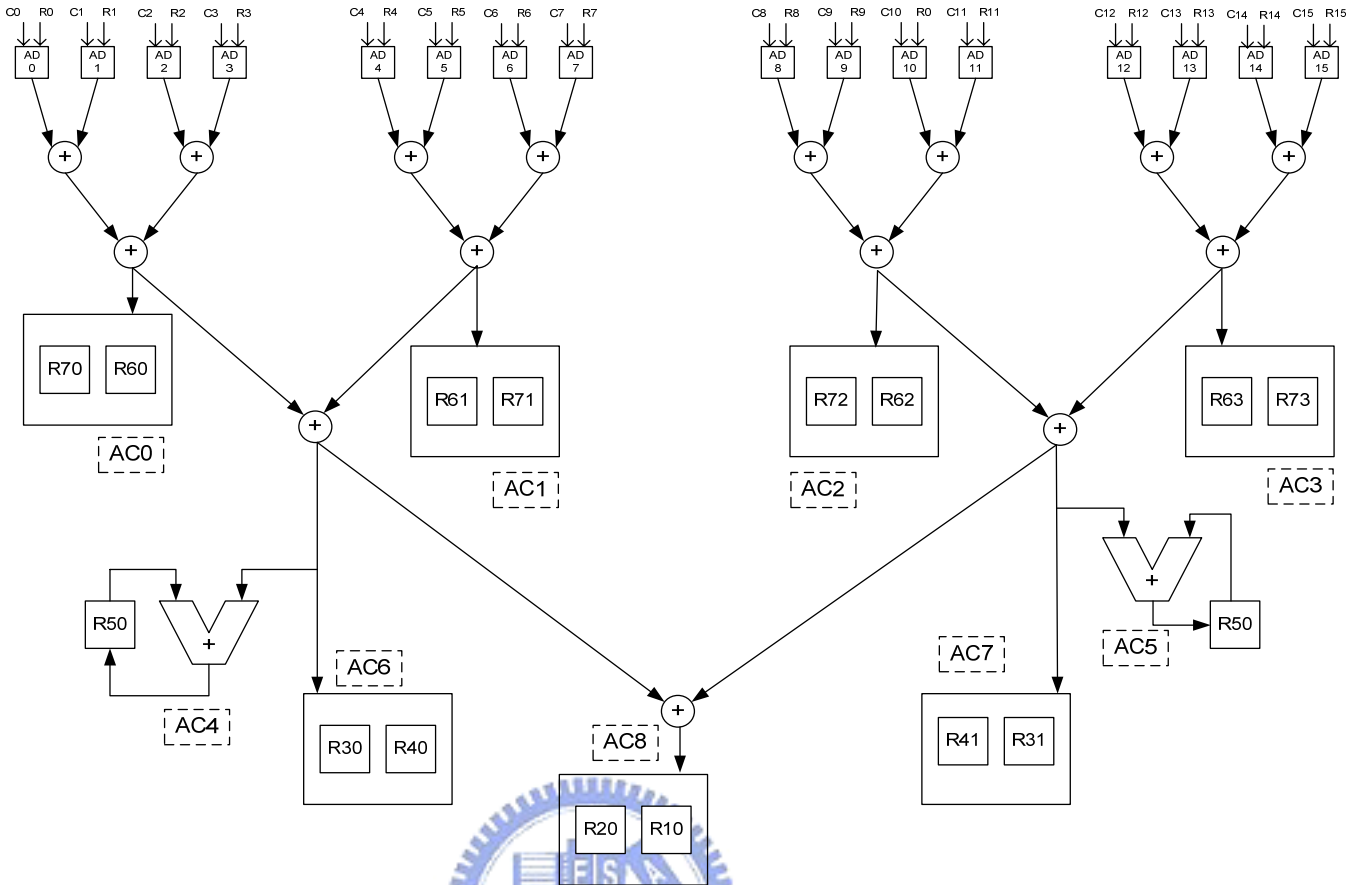


Fig. 3.7 ME unit internal design.

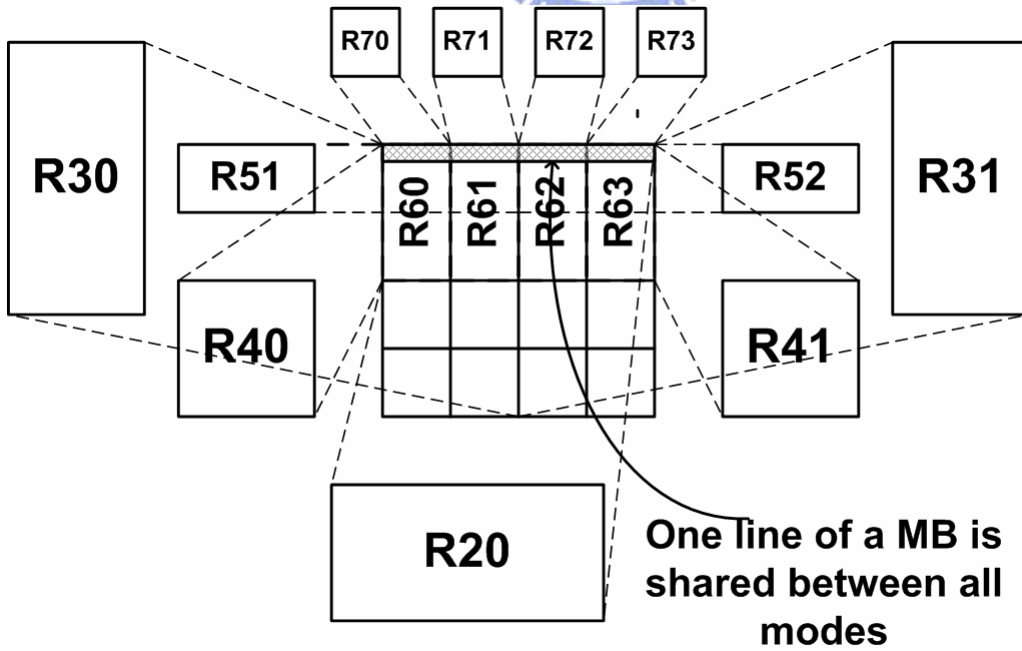


Fig. 3.8 Spatial overlapping between different modes.

Table 3.7 Modes generated every 4 clock cycles*

Clock cycle number	Modes generated and submitted to CS unit.
4	Mode7(00,01,02,03), Mode5(b0, b1)
8	Mode7(04,05,06,07), Mode6(a1,a2,a3,a4), Mode5(b2,b3), Mode4(c0,c1) and Mode2(e0).
12	Mode7(08,09,10,11), Mode5(b4,b5)
16	Mode7(12,13,14,15), Mode6(a6,a7,a8,a9), Mode5(b6,b7), Mode4(c2,c3), Mode2(e1), Mode3(d0,d1) and Mode1

*The labels are as indicated in Fig. 3.3 (a)

Table 3.8 Clock cycles consumed to finish 16x16 search window for 50 frames.

Test sequence	No termination	Proposed	<i>CHG_ClockCycle (%)</i>	<i>Average clock cycles/SR</i>
foreman	14586922	4083958	-72.00%	1147
stefan	14586922	3929368	-73.06%	1103
news	14586922	3188245	-78.14%	895
caostguard	14586922	4845678	-66.78%	1360
M&D	14586922	2766542	-81.03%	777
mobile	14586922	3244230	-77.76%	911
Average			-74.78%	1032

Compare and select (CS) unit will compare the SAD of every sub-block, and generate 41 motion vector in case of H.264 or one motion vector in case of MPEG-4. Also the CS unit will monitor R10 every clock cycle and test it with the threshold value derived from the minimum SAD for mode 1. If the partial SAD stored in R10 is equal to or larger than the threshold value, a 'SKIP' signal will be generated to the control unit, which in turn will reset all registers and advance to the next MB in search window and start doing the SAD computation for the new searching point.

3.3.2.3 Architecture Implementation

The architecture described has been designed by using VERILOG and synthesized by SYNOPSIS design Compiler with UMC 0.18um CMOS standard cell library. The gate count is 16 K and the operated frequency is 86.8 MHz. this architecture can process 720x480 @15 fps video size without applying any termination process. If we assume the termination process will save only 50% of complexity as a worst case, then we can process the same video size @ 30fps. The design can be pipelined by one

stage to raise the operating frequency if needed to process large video sizes. Table 3.8 shows the clock cycles needed to finish one search range.

3.3.2.4 Performance Comparisons

An exact comparison is complicated by the fact that these have been implemented with different technologies and exhibits variations in their specifications and capabilities. The architecture in [61] using 1-D array with 16-PE, each PE can process one MB candidate in 261 clock cycle. The 16-PE will process the 16x16 search window in approximately 4176 clock cycle with the cost of 208 register and 32 adders. While the architecture in [62] is a 2-D array, to process one MB in one clock cycle, followed by adder tree to generate other mode's SAD. The proposed architecture process one MB in 16 clock cycle, and scan 16x16 search window in 4096 clock cycle (without any termination process and an average 1032 clock cycles with termination process) using only 16 registers and 31 adders. Table 3.9 shows the comparison between the proposed architecture and other works.

Table 3.9 Comparison of some VBSME core

	[61]	[62]	This work
PE numbers	1D array with 16 PE	2D array with 256 PE	One PE
Process	0.13 um	0.35 um	0.18 um
Voltage	1.2	N/A	1.8
Frequency	294 MHz	66.67 MHz	86.8 MHz
Gate count	61k	91k without the internal memory	16k
Average Cycles/SR	4176	1583	*4096

* This value in case of no termination process, when applying termination it will be in average 1032

3.4 Summary

To reduce the motion estimation burden represented by the extensive computations needed to find the target motion vector, a fast algorithm and its architecture were presented to reduce this burden by reducing the amount of the computations required to process one search window.

The proposed algorithm adopted PDE to early terminate the candidate position by applying a test based on threshold value. If the candidate position generated a distortion larger than threshold at any instant of time, the process will be terminated and the

remaining computations will be skipped. The key point is that the threshold value is adaptively updated and modified during the matching process. This adaptively changing threshold can give more chances to some candidate positions to be tested and gets more accurate motion vector. Moreover, the algorithm showed a good performance when applied to variable block size motion estimation, which degrades any early termination algorithm. The proposed algorithm outperformed other similar algorithms, achieving a complexity reduction of 78% and 51% for MPEG-4 and H.264 respectively.

The corresponding architecture adopted tree-adder architecture to implement the proposed algorithm. The architecture supports the variable block size, and the early termination scheme. The hardware design will degrades a little of the performance due to some limitations of the hardware considerations.

The architecture described has been designed by using VERILOG and synthesized by SYNOPSIS design Compiler with UMC 0.18um CMOS standard cell library. The gate count is 16 K and the operated frequency is 86.8 MHz. this architecture can process 720×480 @15 fps video size without applying any termination process. If we assume the termination process will save only 50% of complexity as a worst case, then we can process the same video size @ 30fps. The design can be pipelined by one stage to raise the operating frequency if needed to process large video sizes. Table 3.8 shows the clock cycles needed to finish one search range.

Chapter 4 Data Reuse Exploration

Between Vertical Adjacent Macro Blocks.

4.1 Introduction

Power consumption of multimedia applications executing on embedded systems is heavily dependent on data transfers between system memory and processing units. Efficient exploitation of temporal locality in the memory accesses on array signals can have a very large impact on the power consumption in embedded data dominated applications, especially for motion estimation.

In motion estimation (ME), in order to find the best matched candidate and its corresponding motion vector (MV), a search window within one reference frame has to be searched. The traffic between frame buffer and ME core is very heavy (in the order of TB/s for SDTV videos). The instruction profiling shows that 2.76 tera-operations/s (TOPS) of computational loading and 4.25 tera-bytes/s (TB/s) of memory access are required for real-time encoding SDTV (YUV420, 720x480, 30fps) videos (JM8.5, baseline options, full search, four reference frames, search range [-32, +31]). It consumes too much power and is not achievable in today's VLSI technology. The common solution is to design local buffers to store reusable data. By means of local memory access, the external memory bandwidth can be greatly reduced. Four data reuse strategies have been proposed with different tradeoffs between local memory size and system bus bandwidth, and are indexed from level-A to D [64], [74]. Data reuse concept can be explored to reduce this amount of data transfer. Minimizing accessing the main system memory will result in reducing the power consumed by such a huge memory system, and also will avoid the latency to acquire data from the main memory.

Moreover, reducing the memory access will result in reducing the interconnection power consumption.

Recently, to reduce the number of memory accesses in one search block, a block matching method within a search area to reuse the search data is provided using systolic process arrays. To further reduce the data access and computation time during the block matching, the data shared between search range windows explored. Only 1/3 of the search window data need to be accessed after processing the first macroblock [62]. More data reuse can be explored through the reuse of the previously-search data in two dimensions as addressed in [63]. Parallel processing in the horizontal direction will speed up the ME process, also will reduce the memory access by reusing shared data between the adjacent search windows. This approach to reduce the memory access demands more buffer memory, while we can achieve the same memory access reduction by implementing Level C of data reuse scheme with less buffer memory. The proposed approach address the vertical processing for more than two processing elements to achieve higher processing speed for high end applications, also to further reduce the memory access compared to the horizontal parallel processing.

This Chapter is organized as follows. First, we briefly introduce the data reuse within the motion estimation algorithm, and explore the different levels of data reuse. Then we propose an algorithm to explore more data reuse with less system resources, , Its associated hardware implementation is also presented. Finally, a summary will be given in section 4.4.

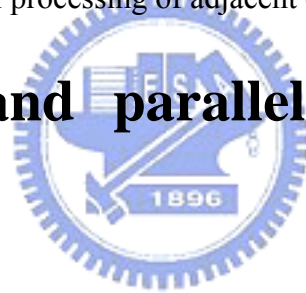
4.2 Data reuse in the motion estimation

Motion estimation searches the reference frame for a candidate block to match the current block with minimum distortion error. This process will access both the current and the reference frame for pixels more than once. These pixels can be reused by saving them on a local memory, rather than accessing them from the main memory system. Four levels of data reuse are introduced and discussed in [64], [74] in details. Level A reuses the shared pixels between adjacent candidate blocks, while Level B achieves more data reuse by reusing shared pixels among adjacent strips of candidate blocks within one search window. Pixels shared between adjacent search windows will save

more memory access by reusing them. Level C of data reuse explores the shared data between adjacent search windows, while Level D explores those shared pixels between adjacent strips of search windows. Unfortunately, saving more memory access will come with the expense of implementing larger buffer memory on chip. This amount of buffer memory will differ from one architecture design to another. Fig. 4.1 shows the geometry of the four reuse levels.

Table 4.1 summarizes for each reuse scheme the Redundancy Access Count R_a and the required memory size needed to apply each Level scheme in case of using single PE to process one candidate block. R_a represents the average access count per pixel in FSBM processing, with a smaller value indicating greater reduction of memory bandwidth. It is clear that, to reduce the data redundancy, more buffer memory is needed. From Table 4.1, Level D achieves the minimum memory access by one-access for every pixel in the reference frame. However, the amount of the buffer memory needed is high. In our approach, we want to get as close as to Level D in low memory access, with minimum buffer memory by vertical parallel processing of adjacent candidate macro blocks.

4.3 Data reuse and parallel processing of vertical blocks.



In the previous sections we review four levels of data reuse, namely Level A, B, C and D. Among the four levels, Level C scheme is often used because it is easy to reuse the horizontally overlapped region between two adjacent search regions. The required local memory size for the reference frame in Level C scheme depends on the detailed implementation of ME architecture. In general, the buffer size in this level is equal to one search range size. As for the Level D scheme, it can minimize the memory access by fully reusing the horizontally and vertically overlapped search regions but with a huge local memory size, $(W+SR_H-1) \times (SR_V-1) + (SR_H-1) \times N$ where W is the width of the frame, and SR_H and SR_V are the horizontal and vertical size of the search window. The required buffer size could be as large as one frame size.

Table 4.1 Redundancy access count and buffer size needed by each reuse level.

	Redundancy Access Count	Buffer memory size in bytes
Level A.	$SR_V \times \left(1 + \frac{SR_H}{N}\right)$	$N \times (N - 1)$
Level B.	$\left(1 + \frac{SR_H}{N}\right) \times \left(1 + \frac{SR_V}{N}\right)$	$(N + SR_H) \times (N - 1)$
Level C.	$\left(1 + \frac{SR_V}{N}\right)$	$(N + SR_V - 1) \times (SR_H - 1)$
Level D.	1	$(W + SR_H - 1) \times (SR_V - 1) + (SR_H - 1) \times N$

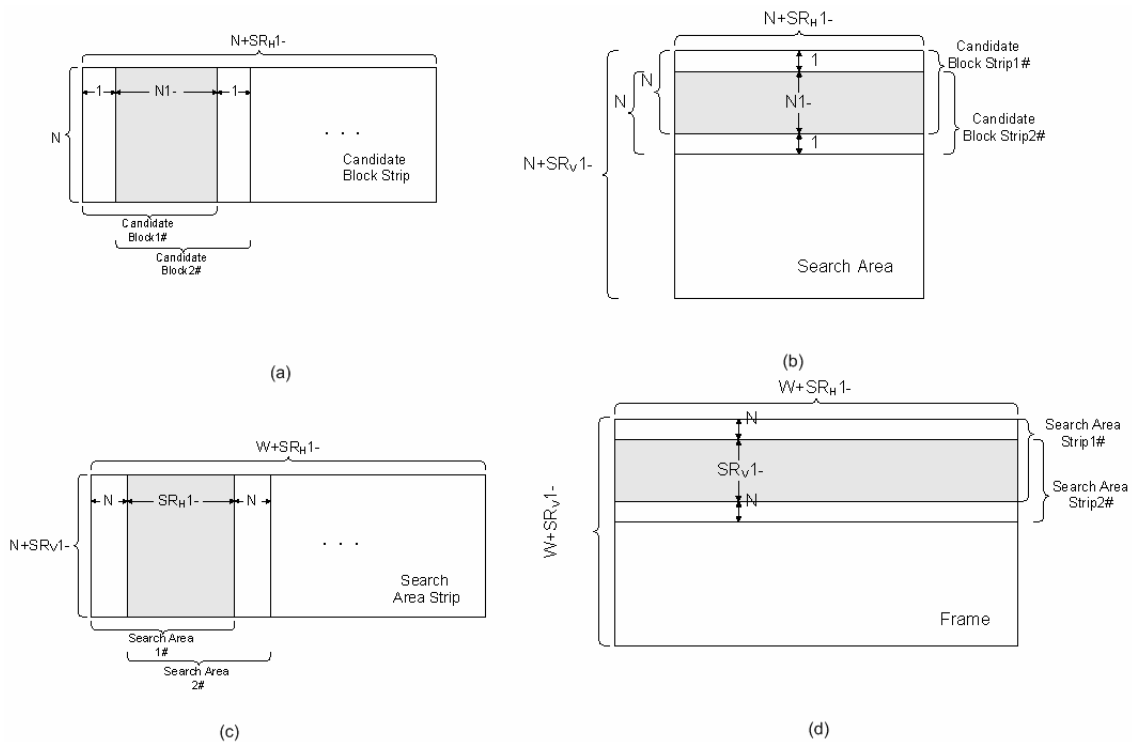


Fig 4.1 Four data reuse levels (a) Level A (b) Level B (c) Level C and (d) Level D.

For today's high quality video sequence, it's impossible to execute motion estimation with only one PE to meet the real time constraints. Parallel processing is the solution for this issue. Using more than one PE will also affect the amount of memory required in every reuse Level scheme. For example, 2-D systolic array with 256 PE's will eliminate the need for the current block buffer ($N \times N$). Parallel processing may not only applied to the same current block, but also to process more than one adjacent current block at the same time. Processing adjacent blocks in parallel will explore two levels of data reuse, namely Level C and D. To improve the data reuse to be close to the Level D without too large buffer cost, the design in [63] explores the data reuse in both directions, which is classified in [65] to be Level C+ scheme. Their design processes four current blocks at

the same time, two at the horizontal direction and two at the vertical direction. Thus, the search range data can be reused not only at the horizontal direction as that in Level C but also at the vertical direction. However, their scheme just reduces the number of memory access by one. Every overlapped search range stripe will still be accessed twice as shown in Fig. 4.2 (one access for each search range). Thus, it is still far away from the Level D reuse scheme: one time access. Besides, this data reuse scheme will not be beneficial when dealing with large search regions for large video sizes like HDTV. For example, for ± 16 search range, each stripe needs to be accessed twice for the design like [63] (i.e. Fig. 4.2). While for ± 32 search range, each line will be accessed three times due to larger overlapped regions. Thus, for larger search range, the scheme in [63] still needs large number of data access.

To further improve the data reuse to approach the Level D reuse scheme, we propose a general approach that adopts multiple vertically adjacent current blocks for parallel searching and share their overlapped search range. When the parallelism is increased, the required memory access will approach that in the Level D scheme. Therefore, this approach can be regarded as a general method to implement the Level D like reuse scheme.

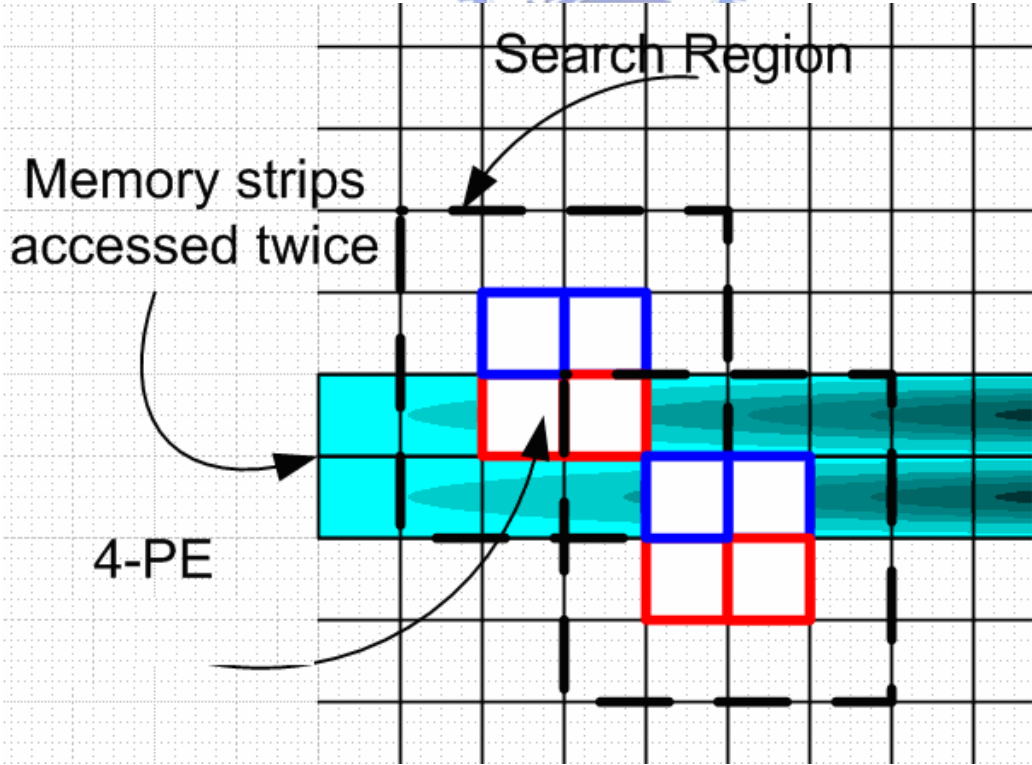


Fig. 4.2 Memory access diagram in [63]. The overlapped search region (gray part) will be accessed twice. Each block in the diagram denotes one MB.

To demonstrate its effectiveness, we also design a motion estimation architecture that can efficiently implement the approach. The resulting design can easily achieve real time SDTV processing and needs lower bandwidth than previous designs with small buffer size increase.

4.3.1 Proposed data reuse scheme

As mentioned in the previous section, the level D scheme can effectively reduce the memory access to only once. However, the main problem with this level is the huge buffer memory required. Thus, our proposed scheme is to be midway between saving more memory bandwidth and using less buffer memory through the use of multiple current blocks arranged vertically and working in parallel in horizontal direction. With this, Level D scheme can be applied partially to those overlapped search range.

Fig. 4.3 shows an example with five current blocks. In this example, since five current blocks are processed simultaneously, the central overlapped search range of these current blocks will only be accessed once for the ME processing. Only the top and bottom boundaries of the search range overlapped with vertical adjacent ME processing will be accessed twice. The one-time access region can be increased with more parallel processed blocks to save more memory bandwidth, but this will also increase the required buffer cost.

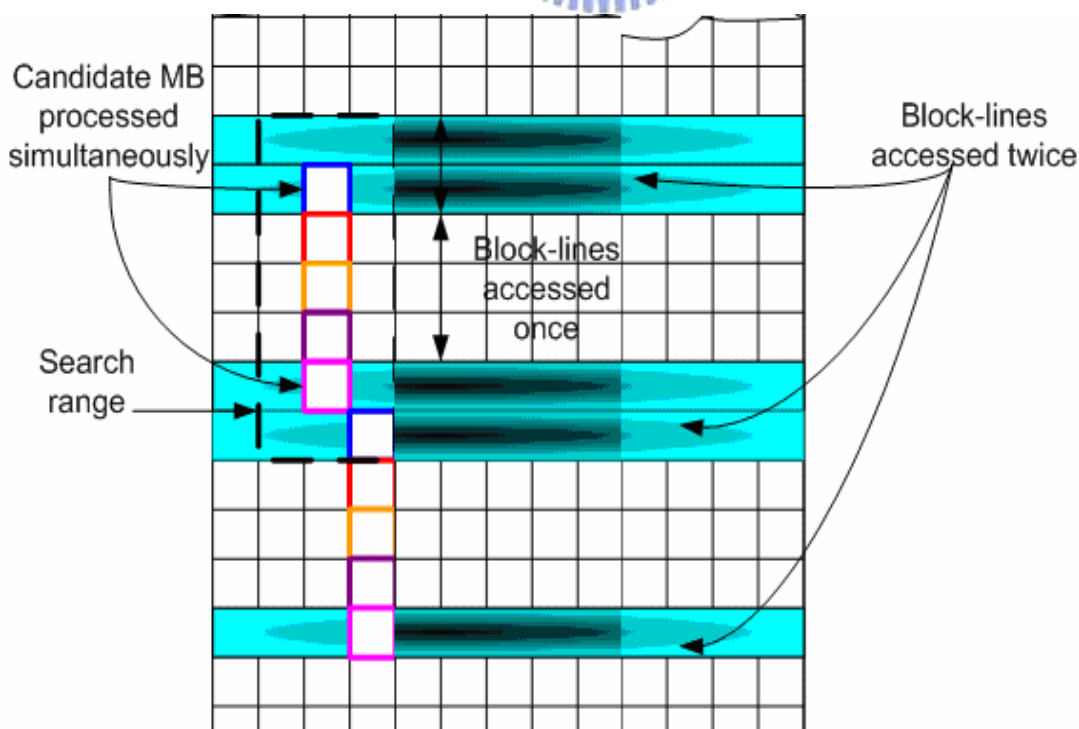


Fig. 4.3 Proposed scheme with five current blocks

Table 4.1 shows the search range buffer cost and the average number of memory access for different number of current blocks. The required buffer cost increases linearly with the number of current blocks. Each additional current block will increase the memory size by $(16+2P) \times 16$ bytes (where the size of the MB is 16×16 and P is half of the search range), but this is still much smaller than that in Level D scheme. The average number of memory access is also higher for larger search range but it is reduced significantly with more parallel processed current blocks.

These numbers of memory access can be derived as followings. When using single current block and ± 16 search range, the number of memory access per MB is $3H'-2$ ($H' = H/16$, and H is the height of the image frame). For two current blocks and above the access times will be according to the following simple equation: $[(H'(Nc+2)/Nc) - 2Nc+2]$ (where Nc is the number of current blocks used in the design). Meanwhile For a single current block and ± 32 search range, the average number of memory per MB is $5H'-6$. For two current blocks this will be $3H'-4$, and for three current blocks this will be $7H'/3-4$, and for four current blocks and above this will be $[(H'(Nc+4)/Nc) - 2Nc+4]$.

Table 4.1 Number of access and search range buffer size for different number of current blocks.

no. of current blocks	Average no. of accesses of one MB stripe*		SR buffer size in bytes		Current MB buffer size
	± 16 SR	± 32 SR	± 16 SR	± 32 SR	
1	3	5	$3 \times 3 \times N \times N$	$5 \times 5 \times N \times N$	256
2	2.00	3.00	$4 \times 3 \times N \times N$	$6 \times 5 \times N \times N$	512
3	1.67	2.33	$5 \times 3 \times N \times N$	$7 \times 5 \times N \times N$	768
4	1.50	2.00	$6 \times 3 \times N \times N$	$8 \times 5 \times N \times N$	1024
5	1.40	1.80	$7 \times 3 \times N \times N$	$9 \times 5 \times N \times N$	1280
6	1.33	1.67	$8 \times 3 \times N \times N$	$10 \times 5 \times N \times N$	1536
7	1.29	1.57	$9 \times 3 \times N \times N$	$11 \times 5 \times N \times N$	1792
8	1.25	1.50	$10 \times 3 \times N \times N$	$12 \times 5 \times N \times N$	2048
9	1.22	1.44	$11 \times 3 \times N \times N$	$13 \times 5 \times N \times N$	2304
10	1.20	1.40	$12 \times 3 \times N \times N$	$14 \times 5 \times N \times N$	2560

* Assuming the frame height is much larger than block height (16 pixels)

Fig. 4.4(a) shows the comparison of number of memory access with the single current blocks. As the number of current blocks is increased, the number of memory access decreases quickly and we can achieve partial Level D scheme of data reuse. This comes with an extra buffer size for every added MB, which is the usual cost for the data

reuse. As shown in Fig. 4.4(b) the percentage increase in buffer size compared to single block is increasing, but the percentage increase for ± 32 search range is less than that for ± 16 search range. We can draw from above; our proposed architecture can achieve lower memory access with lower percentage increase in search range buffer size for larger search ranges, which is more suitable for large video sizes with better quality.

When compared with previous design, the proposed scheme can save a lot of memory bandwidth, as shown in Table 4.2. In [63], it explores the processing of more than one current block in both horizontal and vertical direction to make use of the shared data between overlapped search regions. In our approach, we expand the parallelism only in the vertical direction that works in parallel in horizontal direction. As an example, for CIF size (352x288) and ± 16 search range, using the architecture of [63], the number of reference data pixels accessed from the reference main memory for one frame will be almost $(2 \times 288 - 2 \times 16) \times 352 = 191488$. By applying our scheme to the same conditions as above but with the four-current blocks in the vertical direction, the number of reference data pixels accessed from the reference main memory for one frame will be $(6 \times 288 / 4 - 6 \times 16) \times 352 = 118272$, which are 38% lower. The extra SR buffer memory is only 12.5% larger than that in [63].

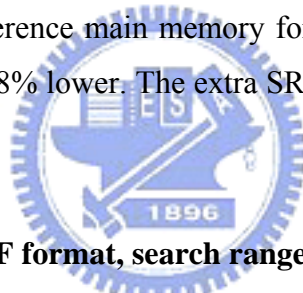


Table 4.2 Comparisons for CIF format, search range ± 16 and using 4 PE. PE is assumed to be 1-D array.

Design	pixels accessed/frame	SR buffer (bytes)
Single current block	304128	$3 \times 3 \times N \times N$
[63]	191488	$4 \times 4 \times N \times N$
proposed	118272	$6 \times 3 \times N \times N$

Table 4.3 Memory access cycles for different number of current blocks in the 2-D array case.

Number of current blocks	1	2	3	4	5	6	7	8	9	10
Clock cycles for 32-bits bus	576	760	960	1152	1366	1536	1728	1920	2112	2304

4.3.2 Architecture Design

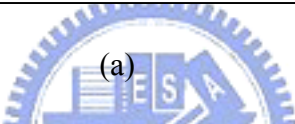
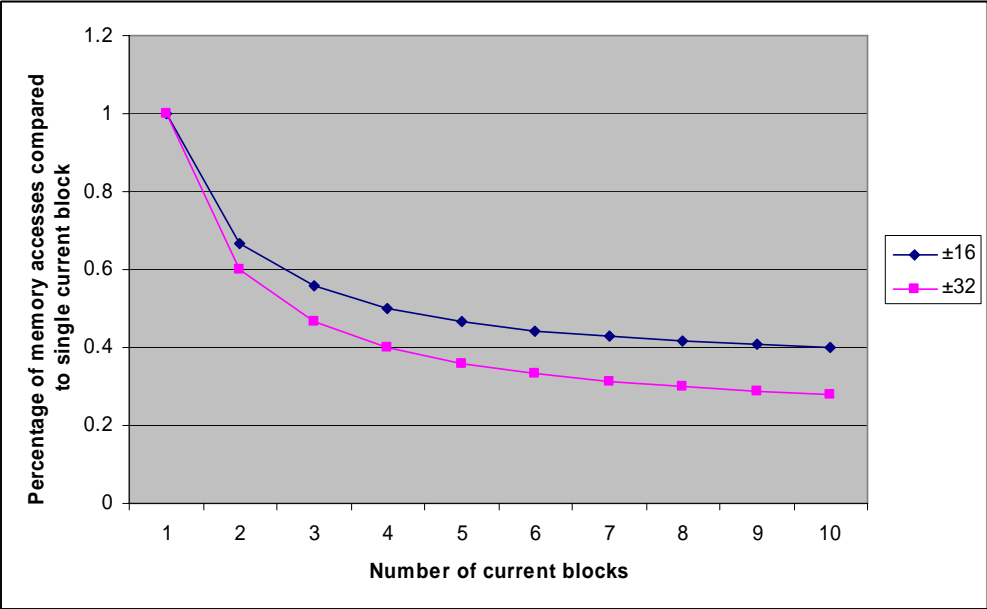
4.3.2.1 Consideration for Parallelism.

The limitation of number of current blocks is the data availability for such parallelism, which could be determined by system bandwidth. A trade off between the SAD processing elements (PE) and system data width should be made carefully to insure smooth and un-interrupted processing, which strongly depends on the ME architectures. In ME architectures, two styles of designs are often used, 1-D and 2-D systolic arrays. A 2-D systolic array presented in [62] will process every candidate position in one clock cycle, with a latency of 16- clock cycles for the first candidate position. It can complete the ME process of one search range in 1,536 clock cycles. Now, if the main system data bus width is 32-bits, and we use one 2-D array for one current block, the clock cycles needed to fill the search range data for different number of current blocks can be tabulated in Table 4.3 (as an extreme case we consider filling the full search range buffer with data, rather than considering filling only 1/3 of the search range buffer). It is clear that for more than six current blocks the available data rate will be less than that needed by the 2-D array and results in idle hardware utilization. So for 2-D array designs, we can limit the design to no more than six parallel current blocks. For the 1-D array, it processes every candidate position in 16 clock cycles, and a total of 16,384 clock cycle to scan all the search range. Similar limitation can be derived to determine the maximum available parallelism.

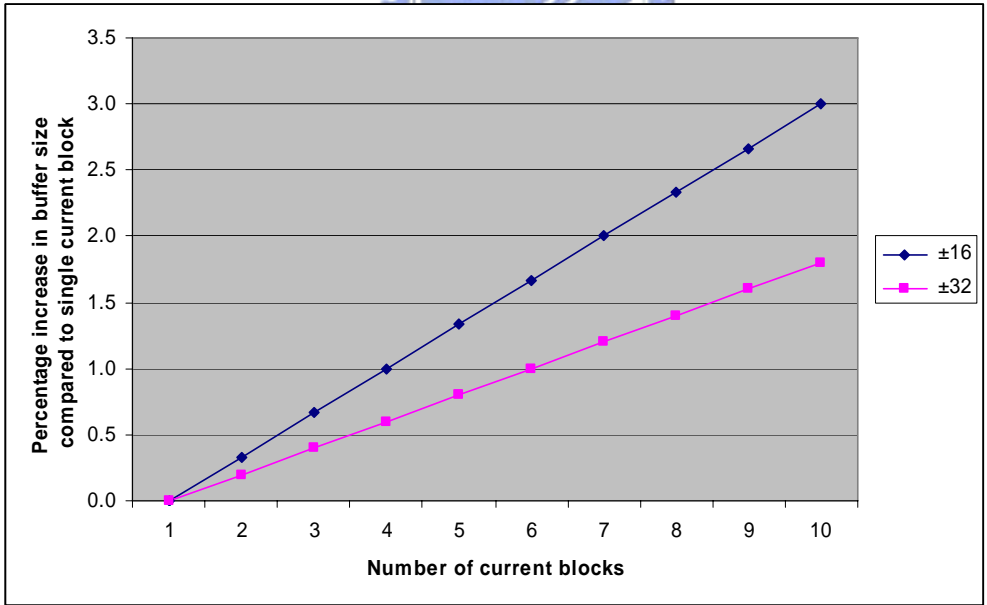
4.3.2.2 Architecture Design.

In the architecture design of the proposed scheme, the most critical issue is how to distribute the search range data without too much overhead. For easy data distribution, we divide the search range in the unit of MB height (i.e. 16 pixels height) since each ME processing is overlapped by one MB height. Thus, for the case of ± 16 search range and four parallel current block processing, the total data can be divided into six smaller search range buffers, as shown in Fig. 4.5. Each buffer is 16-pixels height and 48-pixels width (which is 1/3 of the search range data for ± 16 search range) and the total search range buffer size is 48×96 words. Since each overlapped search range region is allocated into single buffer, the data can be easily shared between different processing units. Thus, each PE will connect to three search range buffers and share these data. This configuration can minimize the wiring connection between different PE's and the

memory modules and will also simplify the addressing and control circuit. Besides, this also avoids data conflict by reading from two different positions from the same memory module by two PE's and thus eliminate the use of multi port memories.



(a)



(b)

Fig. 4.4 (a) Percentage of number of memory when normalized to one current block, (b) the percentage increase in buffer memory size normalized to one current block.

Fig. 4.5 shows the block diagram of the architecture for four parallel current blocks processing that contains four PE's, six search range buffers (M0 to M5), four current block buffers (CB0 to CB3) and finally four 128-bits multiplexers to exchange data between two memory modules. The proposed design can be easily scaled for higher processing rate by directly add one more PE, current block buffer and 1/3 of the associated search range buffer. Fig. 4.6(a) shows the block diagram of each PE. It consists of two small data exchanged PE's, PE_a and PE_b, which include two absolute difference blocks (AD_a and AD_b), and two SAD accumulation blocks.

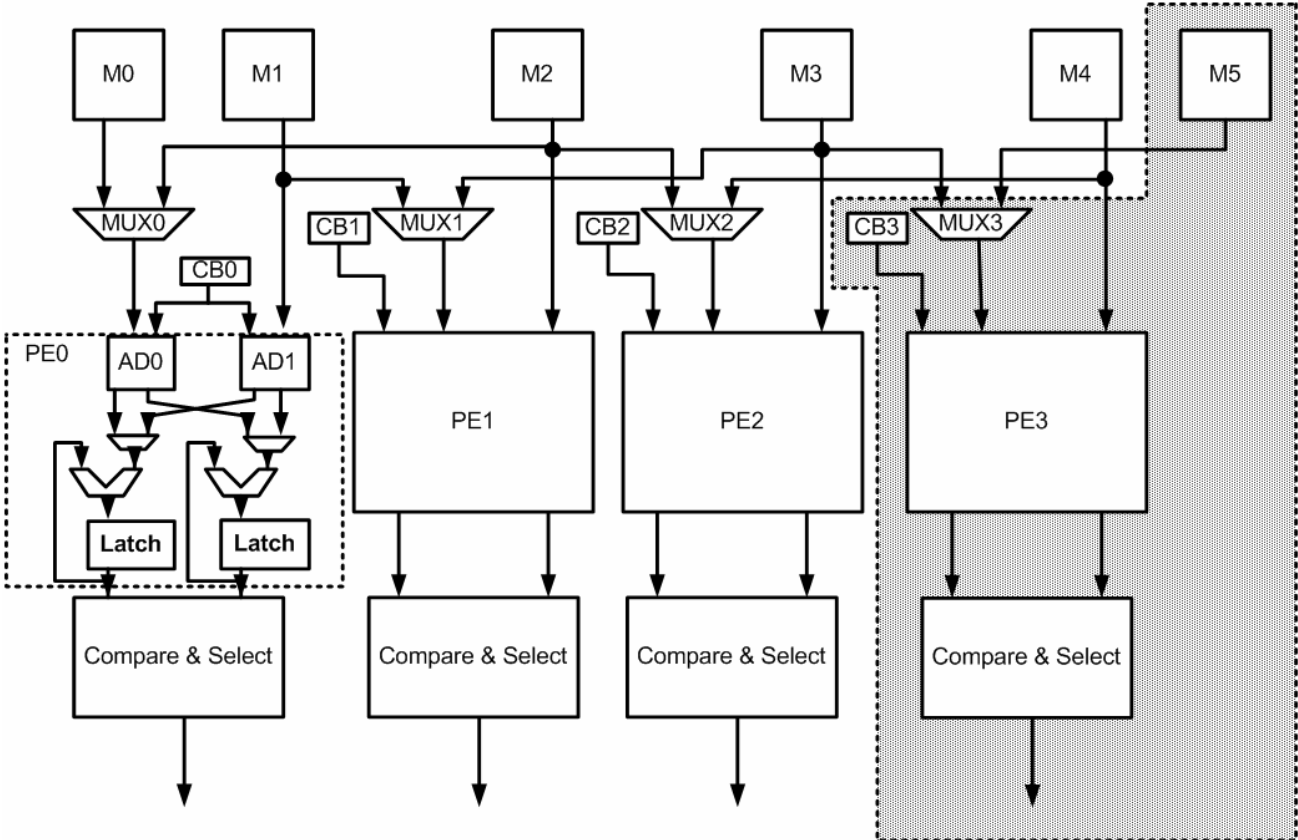
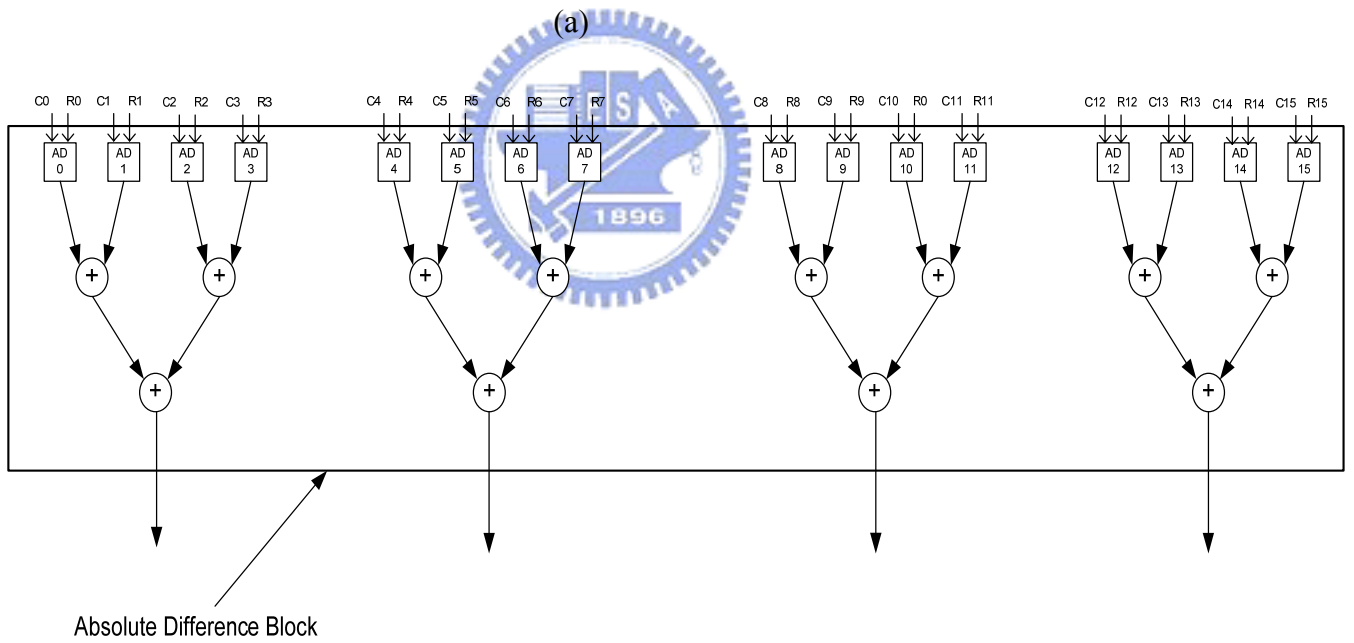
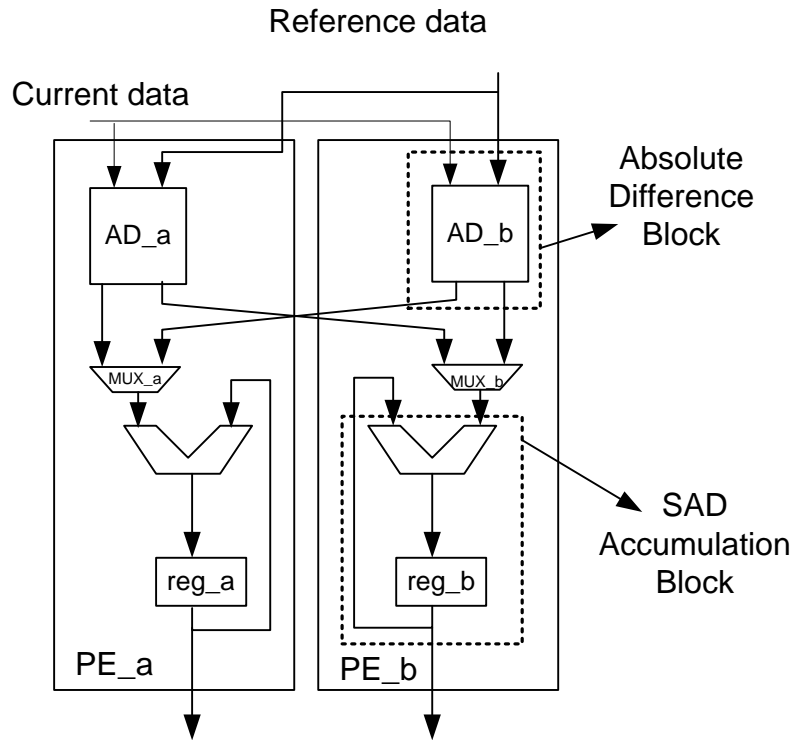


Fig. 4.5 Block diagram for the proposed architecture with 4-PE's.



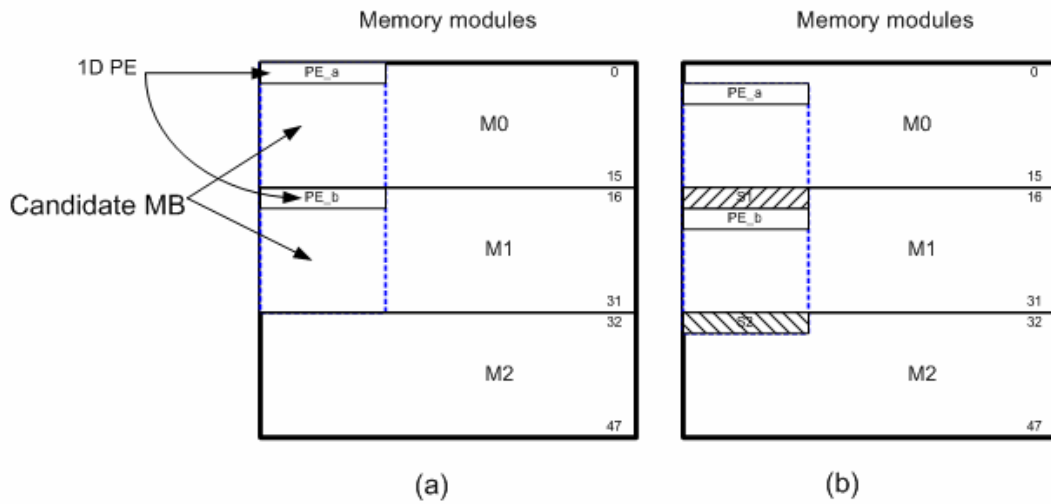


Fig. 4.7 PE operation for the (a) first candidate position and (b) second candidate position showing the data multiplexing between both PE's.

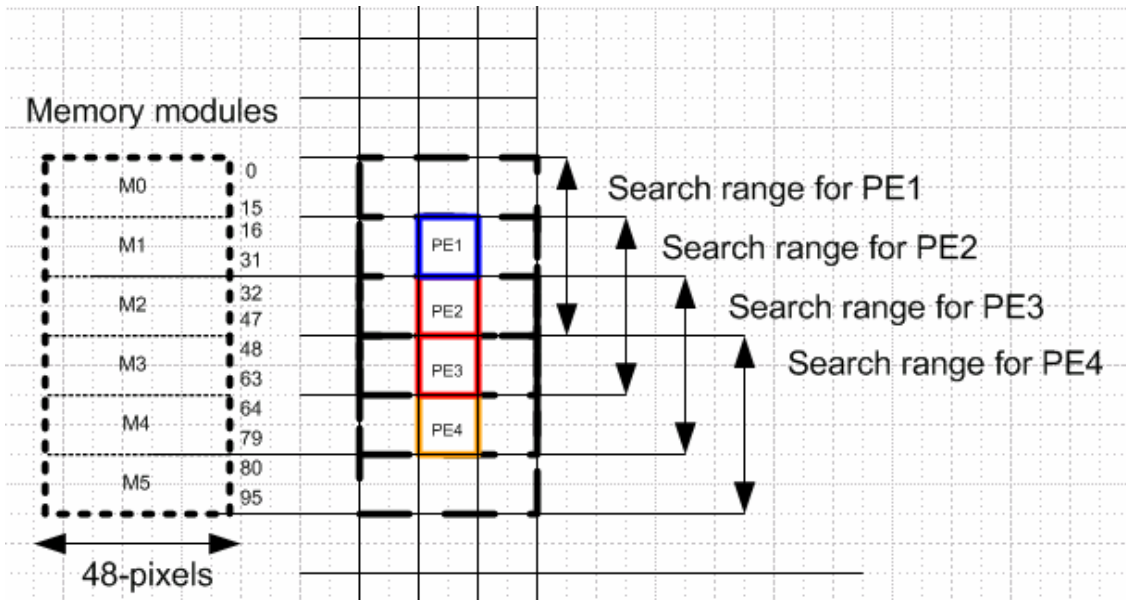


Fig. 4.8 Mapping of memory modules to the search range for every PE.

The operation of the proposed architecture is described below, as illustrated in Fig. 4.7 and Table 4.7. The numbers shown beside every memory modules in Table 4.7 represent the pixel lines as numbered in Fig. 4.8 (i.e. 2-15/M0 means processing the lines 2 to 15 from the search range which is located in memory module M0, and 32-33/M2 processing lines 32 to 33 from the search range which is located in memory module M2). Below we will describe the process of PE0 as an example, which is the same for all PE's in the architecture. First, for processing the first candidate position, PE_a will get the reference data from M0, and PE_b will get its data from M1 as shown

in Fig. 4.7(a) and Table 4.4. When proceeding to the next candidate position (we use the vertical direction first scan order), PE_a will process the first 15-lines from M0, while the last line is located in M1 (the shaded strip S1 as shown in Fig. 4.7) and the same condition is also applied to PE_b (the shaded strip S2). Thus, PE_a can't process the last line which is located in M1 because it is not connected to that memory module, but PE_b can process it. Similar condition also happens to PE_b. Therefore PE_b will process the last line associated for PE_a which is located in M1 and the result will be directed to reg_a through MUX_a as shown in Fig. 4.6(a). A similar operation is also done in PE_a. Finally, reg_a will hold the absolute difference for the top candidate position, and reg_b will hold the absolute difference for the lower candidate position. During the process, though we switch the process between two PE's and different memory modules, the addressing for the current block buffer and search range buffer are not changed, which simplifies the addressing and control circuit.

With above scheduling, the proposed architecture can complete the ME process in $(16 \times 16 \times 16) / 2$ (two small PE's) = 2048 clock cycles for four parallel current blocks with ± 16 search range for each block. Thus, we can generate one MV for 512 cycles in average, which is more than the performance of the 2-D array case.

Table 4.4 Timing and data accessed from different memory modules for every PE.

Time slot	0	1	2	3	...	14	15	
PE0	PE_a	0-15/M0	1-15/M0 32/M2	2-15/M0 32-33/M2	3-15/M0 32-34/M2	...	14-15/M0 32-46/M2	15/M0 32-47/M2
	PE_b	16-31/M1	17-31/M1 16/M1	18-31/M1 16-17/M1	19-31/M1 16-18/M1	...	30-31/M1 16-30/M1	31/M1 16-31/M1
PE1	PE_a	16/31/M1	17-31/M1 48/M3	18-31/M1 48-49/M3	19-31/M1 48-50/M3	...	30-31/M1 48-62/M3	31/M1 48-63/M3
	PE_b	32-47/M2	33-47/M2 32/M2	34-47/M2 32-33/M2	35-47/M2 32-34/M2	...	46-47/M2 32-46/M2	47/M2 32-47/M2
PE2	PE_a	32-47/M2	33-47/M2 64/M4	34-47/M2 64-65/M4	35-47/M2 64-66/M4	...	46-47/M2 64-78/M4	47/M2 64-79/M4
	PE_b	48-63/M3	49-63/M3 48/M3	50-63/M3 48-49/M3	51-63/M3 48-50/M3	...	62-63/M3 48-62/M3	63/M3 48-63/M3
PE3	PE_a	48-63/M3	49-63/M3 80/M5	50-63/M3 80-81/M5	51-63/M3 80-82/M5	...	62-63/M3 80-94/M5	63/M3 80-95/M5
	PE_b	64-79/M4	65-79/M4 64/M4	66-79/M4 64-65/M4	67-79/M4 64-66/M4	...	78-79/M4 64-78/M4	79/M4 64-79/M4

4.3.2.3 Implementation

The proposed architecture has been designed by using VERILOG and synthesized by SYNOPSIS design Compiler with UMC 0.18um CMOS standard cell library. The gate count is 61 K and the operated frequency is 154 MHz in the worst case. In

which, the gate count for single PE is almost 16 K. The critical path for the architecture is 6.49 ns. Thus, with 512 cycles per MV processing rate, this architecture can process 300,781 MB/sec which is able to process 1280×1024 @30 fps video size in real time.

4.3.3 Design comparisons

The system presented in [63] uses four current blocks processing to reuse the data shared between adjacent search memories in two columns and two rows. The speedup for this architecture will be four times faster than single PE. Though data in the horizontal directions can be reused well, data reuse between adjacent vertical search ranges is still poor. Such access redundancy can be efficiently removed in our design. As we showed above, our proposed system reduces the memory access times by 38% compared to [63] using the same number of PE's. Table 4.5 shows the architectural comparison with other ME designs in [62] and [61]. All these designs can support the variable block size ME in H.264. The gate count of our design is similar to that in the 1-D array [61], and much less than that in 2-D array [62]. Meanwhile, with the new data reuse scheme, the required data bandwidth is reduced by 61%, which is much lower than that in both designs.

Table 4.5 Comparison between our scheme and others for 352×288 CIF format, search range ±16 and using 4 current blocks, where N is assumed to be 16.

PE #	[62]	[61]	proposed
process technology	0.35um	0.13um	0.18um
Gate count	106K	61K	61K
Max. Frequency	66.67MHz	294 MHz	154 MHz
Architecture	2-D array	1-D array	Four 1-D array
Memory access/frame* (32-bits data bus width)	76,032	76,032	29,568 (-61%)
Cycles to process one frame	608,256	3,244,032 (+433.3%)	811,008 (+33.3%)
Search range buffer size*	3N×3N	3N×3N	3N×6N
Current block buffer size*	N×N	N×N	4N×N

* These numbers based on the assumption that Level C scheme of data reuse is already implemented and used.

4.4 Summary

A new approach to save memory access through vertical processing had been presented. The goal of the approach is to achieve more data reuse as close to Level D as possible, while the buffer memory size maintained within reasonable limits compared to the original Level D buffer memory required. The proposed approach outperforms the approach in [63] in terms of reducing the memory access, specially for larger search window which is used in high quality video applications. The more processing elements working in parallel vertically, the more reduction in memory access will be achieved and higher processing speed. Design space for the proposed approach was explored, showing the limitations of implementing it. Finally, design architecture was proposed with four processing elements. The design is easily extendable to any number of processing elements to serve the needs of different applications. The reduction in memory access was almost 61%.

The proposed architecture has been designed by using VERILOG and synthesized by SYNOPSIS design Compiler with UMC 0.18um CMOS standard cell library. The gate count is 61 K and the operated frequency is 154 MHz in the worst case. In which, the gate count for single PE is almost 16 K. The critical path for the architecture is 6.49 ns. Thus, with 512 cycles per MV processing rate, this architecture can process 300,781 MB/sec which is able to process 1280×1024 @30 fps video size in real time.

Chapter 5 Future Work and Conclusion

The contributions made through the past chapters will be summarized. Future works related to motion estimation process and mode selection will be addressed. In the first sub-section, fractional motion estimation will be addressed, later we will present the idea of applying the Hilbert Transform to the mode selection. Finally, Binary Motion estimation for texture encoding will be presented in brief.

5.1 Contributions Summary

All the works presented in the thesis were about motion estimation. The heavy burden of the motion estimation represented by two main parts: huge memory access from the main memory, and the extensive computations required through out the searching process. In MPEG-4, shape coding introduced another burden through its working on the bit-level rather than word-level. While H.264 introduced the variable block size and mode decision which increased this burden a lot. The presented works contributed to reduce the burden on both sides; the memory access and the computations.

In Chapter two, shape coding in MPEG-4 were addressed. Binary motion estimation was the real burden in that part, due to its bit-level nature and complexity represented by distortion measurement. The complexity reduction achieved by applying an algorithm, which classifying and testing every candidate BAB. If a candidate BAB belongs to the same class as the current BAB, the distortion measure will be held, else skip that position and advanced to a new one. The algorithm fully scans the search window, but skips the unlikely positions after a simple test. Also, the adaptivity of the algorithm, which represented by overlapping more than one class, gives the ability to tune the performance with the increase in encoded shape-bits. The saving in complexity ranges from 96.69% to 99.71% comes with the expense of increasing the shape encoded bits by 0.7 %to 12.8%.

Hardware implementation for the BME algorithm was presented. Due to the regularity of data flow, the control and address generator unit was simple, and no data

scheduling. The hardware explored the data reused between the horizontal adjacent BAB's and eliminated the need for any shift and pack operations through using the dispatching technique. Due to the possibilities that more than one adjacent BAB's belongs to the same class, they can be processed at the same time, which will increase the speedup of the algorithm. Due to the simplicity and the regularity of the algorithm, the proposed hardware is also regular and needs only 11582 gate count.

Integer motion estimation which is the real burden of any video coding system addressed in Chapter 3. The computational complexity reduced significantly with the proposed algorithm. The video quality almost remained the same with a little increase in encoded bit-stream. Moreover, the proposed algorithm showed a good performance with the variable block size motion estimation which is adopted by H.264. Variable block size reduces the efficiency of any early termination algorithm, but the proposed algorithm showed high reduction in computational complexity. The threshold value used to terminate the matching process is adaptive. It can be set to increase the skipping ratio or getting more accurate motion vector. Also, the threshold value modified regularly during the matching process and not constant as that in the normal PDE algorithms. The proposed algorithm outperformed other similar algorithms, achieving a complexity reduction of 78% and 51% for MPEG-4 and H.264 respectively.

To further assist reducing the motion estimation on the system, a dedicated hardware implementation for the proposed algorithm was presented. The dedicated hardware can work in parallel with the general purpose processor to generate the motion vector. The design generates motion vectors for all block sizes included in the macroblock. Early termination scheme easily implemented due the simple idea of the early termination scheme proposed. It is more close to 2-D array in performance, while keeping low silicon area as that in 1-D array. The proposed architecture process one MB in 16 clock cycle, and scan 16x16 search window in 4096 clock cycle (without any termination process and an average 1032 clock cycles with termination process) using only 16 registers and 31 adders.

The computational burden reduced with the proposed algorithms mentioned above. Moreover, the huge memory access can be reduced also. The next part will deal with this problem.

Motion estimation module generates a huge amount of memory requests to get pixel data from the current and reference main memory. Data reuse can reduce this amount of requests by saving data temporarily on buffers located on chip. Moreover, the real time constraints of video application hardly can be met by using one processing element. Multiple processing elements can be integrated either to process one search window or to process more than search window. More than one processing element processing one search window, will speed up the motion estimation process, however, the memory access will not reduced. Processing more than one search window will speed up the motion estimation process. The way this scheme will affect memory access depends on the way these processing elements are arranged.

In the proposed approach, the parallel processing elements are working in vertical manner. Vertically adjacent search windows will be processed simultaneously, making use of the overlapped pixels between them to reduce the external memory access. This approach makes it feasible to get closer to Level D of data reuse with a small on chip buffer memory. Design space for this approach explored to show the limitations of using it. Also, a suggested design introduced, characterized with regularity and ease to be extended to any number of PE without extra cost to the control circuit or any change in the data flow. Meanwhile, with the new data reuse scheme, the required data bandwidth is reduced by 61%, which is much lower than other presented designs.

To integrate all the works presented into one working system, we may face some problems and limitations. In the case of the Binary Motion Estimation, it can be easily integrated with the Integer Motion Estimation part for MPEG-4 without any problems. However, applying the early termination scheme to the approach of vertical processing has limitations. This will introduce complications for data flow between different search windows. One search window may skip one candidate position, while adjacent search window may need to process that position. In this case we can disable other processing elements while processing that position by the potential processing elements. This will reduce the hardware utilization factor.

In the following subsections, more work to enhance and extend the presented works. Fractional motion estimation contributed a lot in the coding efficiency and worthy to be addressed as an extend work to the Integer motion estimation. Fractional motion estimation introduced a new burden to the system, that is, filters need to be

implemented to generate fractional pixels values. Moreover, computational complexity should be reduced too. Mode decision is the next block after the motion estimation. By estimating which modes to process will reduce the burden of motion estimation too much in case of H.264. Finally, Binary motion estimation is an effective way to reduce the complexity of the Integer motion estimation by working on the bit-level. The possibilities of apply the algorithm in Chapter 2 to the binary motion estimation for texture is open. Careful study for the nature of the generated bit map is needed.

5.2 Future Work

5.2.1 Fractional Motion Estimation

Generally motion estimation is conducted into two steps: the first step is integer pixel motion vector estimation; and the second is fractional pixel motion vector estimation. For fractional pixel motion estimation, 1/2-pel accuracy is frequently used (H.263, MPEG-1, MPEG-2, MPEG-4), higher resolution motion vector are adopted recently in MPEG-4 (1/4-pel accuracy) and H.26L (1/4, 1/8-pel accuracy).

Algorithms on fast motion estimation are always hot research spot. Especially fast integer pixel motion estimation has achieved much more attention because traditional fractional pixel motion estimation (such as 1/2-pel) only takes a very little proportion in the computation load of motion estimation. However, with the development of integer fast motion estimation algorithm and the decreasing of integer motion search points, computation load of fractional pixel motion estimation become more and more comparable to that of integer case. For instance, some center-biased fast integer motion vector estimation algorithms have reduced the checking pixel positions averagely down to 10 [65][66], while 8 1/2-pel positions are needed to be checked around the best integer pixel matching position in full 1/2-pel motion estimation, and more positions should be searched if higher resolution motion estimation is adopted

As a complete work for the motion estimation, Fractional Motion Estimation (FME) should be addressed. As mentioned above, Fast Integer Motion Estimation (FIME) reduced the search points dramatically, and achieved a high speed up, also the

number of search points of the fractional motion estimation become comparable to that of the fast integer motion estimation; the need for applying fast fractional motion estimation is a must.

In the following two sub-sections, we will introduce two methods to minimize the fractional pixel search positions and the filtration process. The main concept of the first method is to minimize the number of search positions (maximum are 9 positions for half and quarter pel ME). The other method will reduce the computational complexity of the filtration process used to compute the $\frac{1}{2}$ and $\frac{1}{4}$ fractional pixels.

5.2.1.1 Proposed Fast Motion Estimation.

The proposed algorithm depends on the assumption that, the error surface model is homogenous and uni-modal. The error surface will follow one direction and there is no many minimum. Fig. 5.1 shows 9-search positions numbered according to the order the JM9.0 test these positions, it starts with position 0 then up to position one and so on.

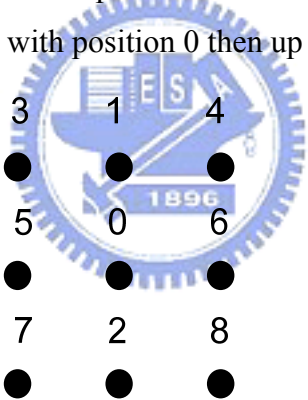


Fig. 5.1 Nine search positions numbered according to their accessing order as in JM9.0

The proposed Algorithm

Depending on the uni-modal error surface, we assume that the minimum SAD can be traced by testing the search points one by one, then make the choice about the next search position rather than testing all the 9 points in case of $\frac{1}{2}$ fractional pixel resolution. The algorithm can be summarized as follow:

1. Start by testing points 5, 0, and 6 respectively, and then determine the position with the lowest SAD among them.

2. After locating the position with the lowest SAD above, test the top position according to the one located in step 1. If this position has lower SAD, then terminate the process and assign this position as the target MV.
3. Else test the bottom position, if the generated SAD for this position is less than the position in step 1, then assign this position as the target MV and terminate the process, else the target position is the middle position.

For example, we start by testing 5, 0 and 6, assume position 6 has the lowest SAD value, then we proceed by testing position 4. If position 4 has lower SAD, then terminate the process and assign position 4 as the target MV. Else we test position 8, if the generated SAD is lower than position 6, then assign this position to be the target MV. If neither positions 4 or 8 has lower SAD value, then assign position 6 to be the target MV. Fig 5.2 shows three testing patterns belonging to the same algorithm. The numbers shown on the arrows representing the order in which every position will be tested. Fig.5.2(a) represents the above mentioned algorithm, while (b) and(c) shows another possible testing pattern. The saving in search positions of (a) and (b) is about 4~5 positions (44% to 55% reduction in search points), and for that of (c) 2~4 positions (22% to 44%) Table 5.1 shows the PSNR, encoded bit stream and complexity reduction when applying the pattern in Fig. 5.2(a).

Table 5.1 PSNR, encoded bit stream and complexity reduction for the proposed algorithm.

	foreman	Stefan	news	coastguard	M&D	mobile
PSNR Y	0.01	-0.03	-0.02	-0.02	-0.06	-0.04
Total bit increase (%)	2.53	0.45	0.98	0.11	1.03	1.02
Skipped ratio(%)	50.80	49.57	53.58	47.97	52.89	48.29

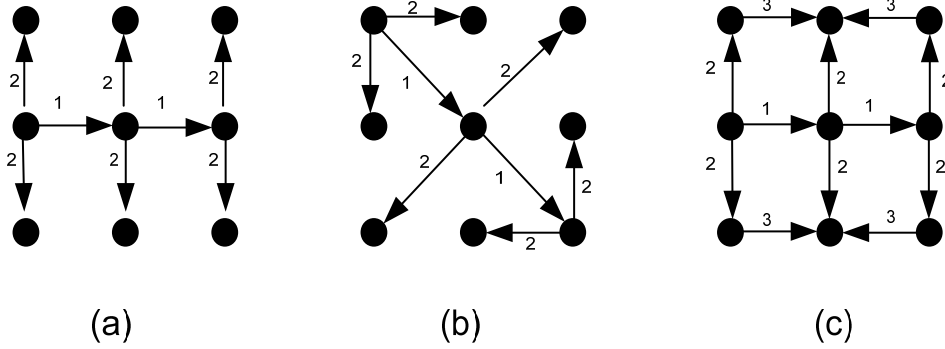


Fig. 5.2 Testing patterns for the proposed algorithm.

5.2.1.2 Reduce the Interpolation Cost by SAD Estimation

The fractional pixel values are generated by interpolation. In the case of $\frac{1}{2}$ pixel resolution, 6-tap filter is used to generate one pixel, while in the case of $\frac{1}{4}$ pixel resolution, 8-tap interpolation filter is used to generate one pixel. This process is power consuming also the filters occupy significant silicon area. In order to reduce the power consumption and save the time needed to compute all the fractional pixel values, we can compute those fractional pixels which are related to position zero (as shown in Fig. 5.1) and then use the normalized-SAD to generate the estimated SAD for the other positions.

Fig.5.3 shows the nine positions and the shared areas between those positions. The MB boundary of position zero contains the most shared pixels with the other positions. So we can make use of the overlapped area by estimating the SAD of other $\frac{1}{2}$ or $\frac{1}{4}$ pixels without calculating their values by interpolation. The idea is simple, first generate all the fractional-pixel values for position zero, then compute the SAD value for this position, and finally, normalize this SAD value by dividing it by 256 (shift to the right by 8-positions). To compute the SAD for the other positions, firstly we compute the SAD for the pixels in the overlapped area (i.e. they are already generated for position 0). Then for those pixels which are not generated, we add the normalized SAD in according to every pixel.

For example, to find the SAD for position one, we compute the SAD value for the overlapped region as usual, because these fractional pixel values are already calculated and ready to be used. The pixels in the top row in the MB boundary in position one are not generated, so we add the sum of the normalized SAD 16-times to replace the real

SAD value of the top row of position one. The formula below can summarize this process as follows:

$$SAD_{total} = SAD_{overlapped} + SAD_{normalized} \times (\text{number of pixels not generated}).$$

The total estimated SAD value for every pixel not belongs to the overlapped area can be summarized as follows:

- Positions 1, 2, 5 and 6 each contains 16 pixels (simply shift the normalized-SAD to the right 4 times and the result to the real computed SAD of the other pixels).
- Other positions each contains 31 pixels (this can be done by shifting the normalized SAD to the left 5 times and subtracting one normalized SAD then adding the result to the computed SAD of the other pixels).

Applying this scheme will result in saving the interpolation process for about 68 fractional pixels (almost 21% saving). Also, simplify the SAD computation process, such that rather than process every pixel to generate the SAD, we can simply shift the normalized SAD to the left 4 or 5 times and adding the result to the computed SAD for other pixels located in the overlapped area (almost 10% reduction in SAD process).

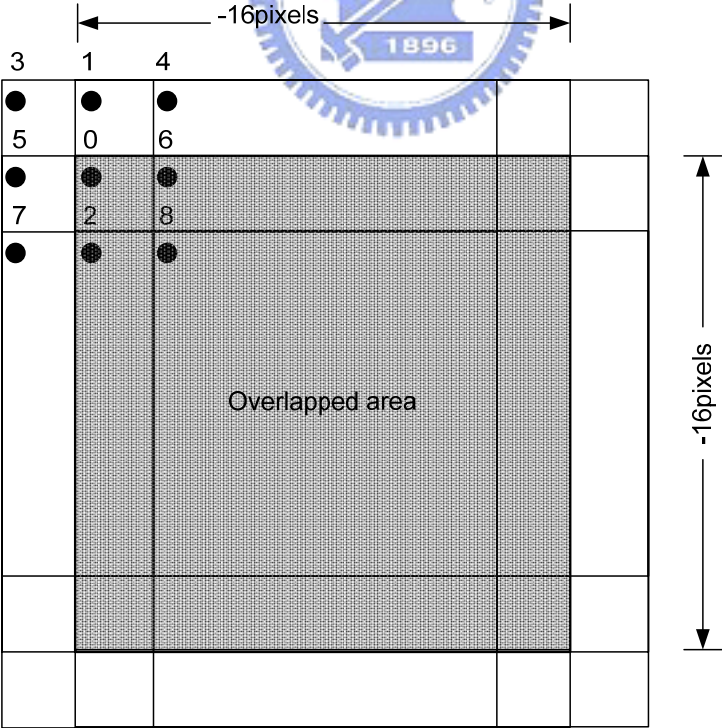


Fig. 5.3 Shows the nine searching positions, and the overlapped region between them.

In conclusion, the real burden of the fractional motion estimation is in the interpolation phase, i.e. the hardware cost, power consumption and the time required to generate the fractional pixels. The focus should address this part of the motion estimation, trying to reduce the cost while keeping the efficiency of the fractional motion estimation process.

5.2.2 Mode Selection and Hilbert Scan.

The newest video compression standard H.264/AVC introduces various coding modes. All modes at macroblock (MB) level for luma components are illustrated in Fig. 5.4(a). There are two intra prediction modes which are denoted as Intra 16x16 and Intra 4x4. The Intra 16x16 does spatial predictions of 16x16 luma block and the Intra 4x4 consists of sixteen 4x4 luma blocks that are separately predicted. For inter-frame prediction, each MB mode corresponds to a specific partition of the MB. For 8x8 inter prediction mode which is denoted as Inter 8x8, each of the four 8x8 blocks is split further in four ways. Fig. 5.4(b) shows five candidate modes for a 8x8 block in B-type frames. In general, selecting a mode with a large partition size means that a small number of bits for motion information are required; however, motion estimation may not be accurate resulting in generating a large number of bits for sending transform coefficients. On the other hand, selecting a mode with a small partition size may require a small number of bits needed to signal residual information but produce a large number of bits for motion vectors and side information. Therefore, the choice of coding mode has a significant impact on compression efficiency. In order to select an optimal mode in Rate-Distortion (R-D) sense, Lagrangian minimization is successfully applied to mode selection problem by Wiegand [67]. A general form of cost function used in Lagrangian R-D optimized (RDO) mode selection method is

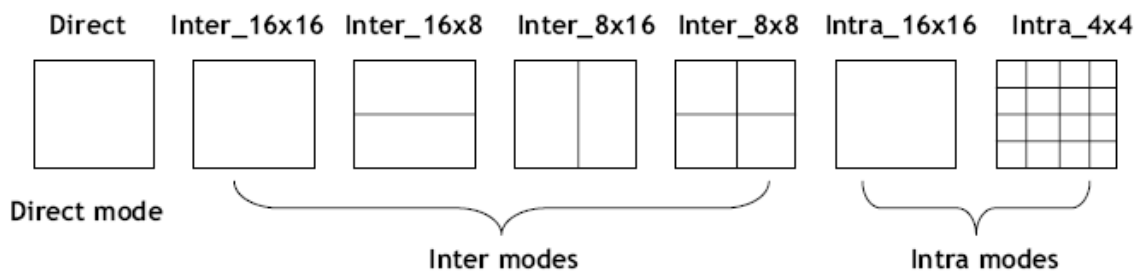
$$J = D + \lambda R \quad (5.1)$$

where J , D and R are R-D cost, distortion and rates of a mode, respectively and λ is a Lagrangian multiplier. In H.264/AVC, calculation of D is much easier than other standards because of integer transform and quantization that require only integer operations [67]. Calculation of R also can be implemented in a efficient way using a table lookups [68]. However, although Lagrangian R-D cost calculation for a single mode may have low-complexity, computation for RD costs of all modes becomes huge because a large number of candidate modes are provided in H.264/AVC [69].

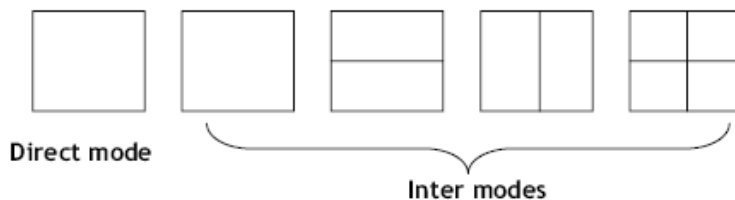
In *JVT* reference software, motion estimation is performed mode by mode with full search scheme, that is, we need to perform motion estimation for each coding mode in

every previous reference frame. The allowed modes are inter16x16, inter16x8, inter8x16, inter8x8, intra4x4 and intral6x16. Note that the inter8x8 block can be further partitioned into smaller blocks.

In addition, all kinds of coding modes *are* not averagely 'distributed' in RD optimization. Therefore we should analyze scenes that are suitable for different coding modes. For encoding macroblock, either we can accurately select one/few coding modes or eliminate some redundant coding modes which is not suitable for this macroblock. Based on analyzing the candidate macroblock for texture complexity; a lot of unnecessary computation can be saved [70].



(a) Coding modes at Macroblock level



(b) Coding modes for a 8x8 block in Inter_8x8

Fig. 5.4 Coding modes in H.264

Some macroblocks may belong to one object because they don't contain edge information. If more than one object are contained in a macroblock and moving in different directions, it is suitable if this macroblock split into multiple sub-blocks for motion estimation and compensation. Reversely if this macroblock contains edge information (its texture is complex); it is not always split into smaller sub-blocks. For example, in test sequence Paris, there are complex textures in its background, but they

mostly select larger block size coding mode because they belong to still background region and there is no motion [71].

In 1891, Hilbert curve is one of Peano curves which is called a space-filling curve. Fig. 5.5 shows an example of Hilbert curves. The Hilbert scan is the result of scanning a 2-D image through one of its Hilbert curves, as depicted in Fig. 5.5. The Hilbert scan extracts clusters in an image easier than other scanning methods (e.g. raster scan) and it preserves 2-D coherence. The edge information in a 2-D image is preserved in its 1-D Hilbert-scan sequence more effectively than the raster scan, which may miss edges due to its scanning direction.

The proposed idea is to use one of the Hilbert curves to scan the candidate macroblock prior to the mode selection process, and depending on the generated edge information and coherency between sub-blocks we may be able to estimate the targeted modes.

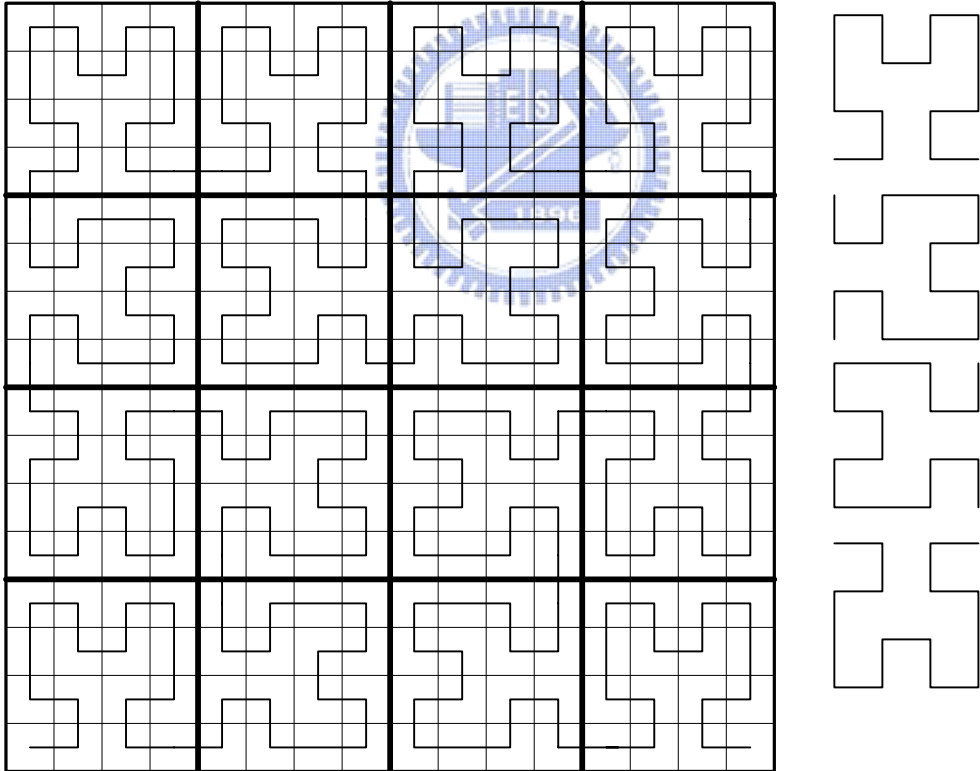


Fig. 5.5 Hilbert scan example for 16x16 block.

5.2.3 Binary Motion Estimation.

Many fast motion estimation algorithms were proposed to reduce the complexity of the motion estimation. One of the important headlines in the field of fast algorithms is the Binary Motion Estimation. The integer motion estimation represents a very

heavy burden on the video encoding system. Every pixel represented by 8-bits, and the architectures designed to serve this process will occupy large silicon area and consumes large power. In a system with a highly restrictions on the power consumption, we need to change the strategy, rather than processing the integer values, we may process pixels represented with lower bit resolution down to 1-bit. This will come with expense of using filters, degrading in the PSNR and increasing the generated bit stream. Also, H.264 represents other challenges when dealing with small block sizes such 4x4 in which it maybe represented by all zeros or ones, and in this case hardly can be find the target MV.

Many binary motion estimation schemes were proposed to significantly decrease both the computational complexity and bus bandwidth by reducing the bit depth. Based on a binary pyramid structure, a fast binary ME algorithm was proposed in [72], namely fast binary pyramid motion estimation (FBPME). The pyramidal structure of FBPME contains one integer layer at the lowest resolution (smallest picture size) and three binary layers that contain detail information. FBPME performs the tiled motion search with exclusive OR (XOR) Boolean block-matching criterion on binary layers and MAD on the integer layer. The block matching uses XOR operations that are much simpler and faster to implement than MAD operations. The problem of the integer layer mentioned above solved by the design proposed in [73], the proposed algorithm is shown in Fig. 5.6. The different algorithms and schemes based on converting the 8-bit pixels to 1-bit by different methods such as using filters or averaging with sounding pixels. The method used to generate the binary image affects a lot the resulted PSNR and encoded bit stream. H.264 introduced another problem by employing motion estimation for small block sizes (4x4).

A new methods and strategies are needed to convert the images into binary form taking into consideration higher compression performance, low area cost and compatible with smaller block sizes. The proposed idea depends on two parts:

- Software approach: implementing a new simple algorithm to generate the binary image based on the Hilbert Scan mentioned in the pervious sub-section. Based on the edge information of the small blocks to generate the binary image.
- Hardware approach: to be able to process both binary and 8-bit pixels, so the hardware should be flexible without high extra hardware cost.

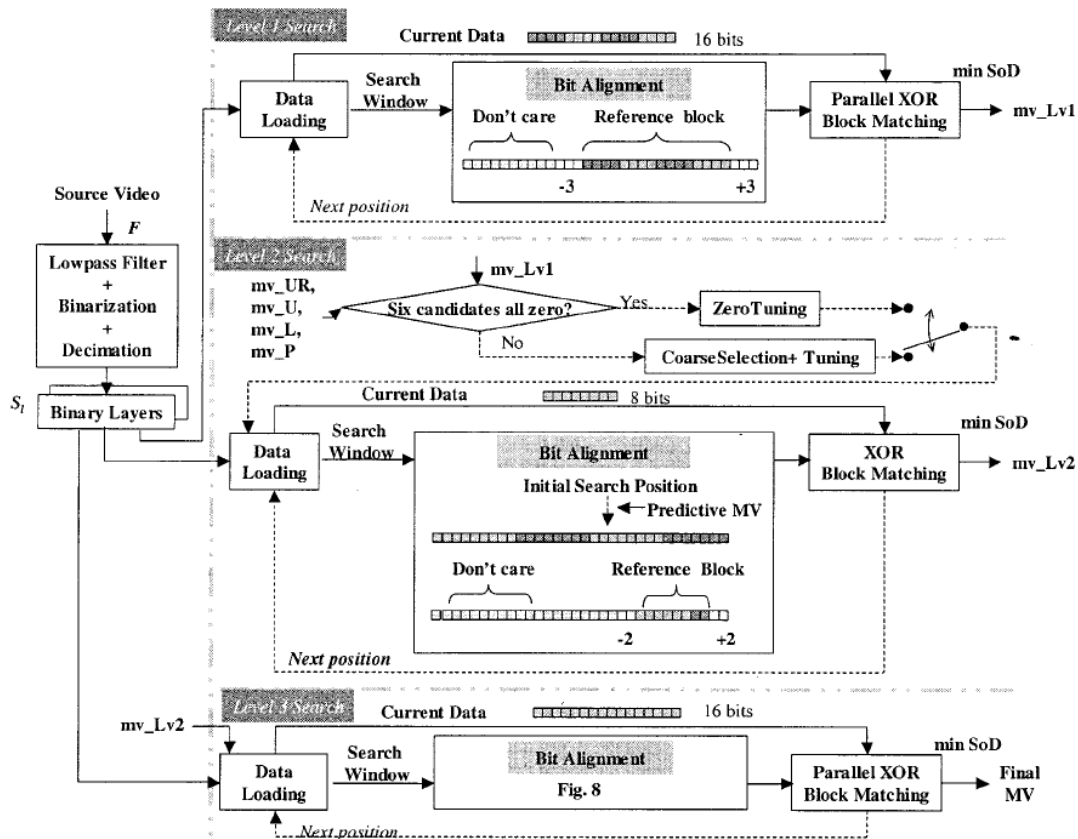


Fig. 5.6 Illustration of ABME search strategy with XOR block-matching criterion.

5.3 Summary

Motion estimation still a hot spot to be addressed. Contributions to reduce its complexity and enhancing the fast algorithms performance are arising. As continues work for the Integer motion estimation, fractional motion estimation should be added. Two ways to reduce the algorithm complexity were presented. One way is to reduce the number of processed search positions, and the other is by reducing the filtration burden. Mode decision can reduce a lot the motion estimation burden by limiting the modes needed to be processed rather than process the seven modes. Finally, Binary motion estimation for texture can achieve a lot of complexity reduction by transforming the pixels representation from eight bits into one bit. Moreover, the work presented in Chapter 2 can be integrated to achieve more computation reduction.



References

- [1] ISO/IEC JTC1/SC29, Coding of moving pictures and associated audio for digital storage media up to about 1.5 Mbit/s, ISO/IEC 1172-2, International Standard, November 1992.
- [2] ISO/IEC JTC1/SC29, Generic coding of moving pictures and associated audio, ISO/IEC 13818-2, Draft International Standard, November 1994.
- [3] ISO AEC JTC1/SC29/WG11, N2502a, Generic coding of Audio-Visual Objects: Visual 14496-2, Final Draft IS, Atlantic City, Dec. 1998.
- [4] ISO/IEC JTC1/SC29, Coding of audio-visual objects, ISO/IEC 14496-2, International Standard: 1999/Amd1:2000, January 2000.
- [5] Joint Video Team (JVT), "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification," 7th Meeting, Pattaya, Thailand, March 7-14, 2003.
- [6] ISO/IEC JTC1/SC29/WG11 N3908, "MPEG-4 Video Verification Model Version 18.0," Jan. 2001.
- [7] JVT reference software version 9.0 <http://bs.hhi.de/~suehring/tml/download/>.
- [8] N. Brady, "MPEG-4 standardized methods for the compression of arbitrarily shaped video objects," IEEE Transactions on Circuits and Systems for Video Technology, vol.9, no. 8, pp:1170-1189, Dec. 1999.
- [9] A. Puri, X. Chen and A. Luthra, "Video coding using the H.264/MPEG-4 AVC compression standard," Signal processing: Image Communication, vol.19, no. 9, pp: 793-849, Oct. 2004.
- [10] Y. C. Wang, H. C. Chang, W. M. Chao and L. G. Chen, "An Efficient Architecture of Binary Motion Estimation for MPEG-4 Shape Coding," Visual Communication and Image Processing, San Jose, CA, pp: 959-967, Jan. 2001.
- [11] T. H. Tsai and C. P. Chen, "An efficient binary motion estimation algorithm and its architecture for MPEG-4 shape coding," in proceeding of IEEE International Symposium on Circuit and System, vol. 2, pp: 496-499, May 2003.

- [12] T. H. Tsai; C. P. Chen;" A Fast Binary Motion Estimation algorithm for MPEG-4 shape coding," IEEE Transactions on Circuits and System for Video Technology, vol. 14, no. 6, pp: 908 – 913, June 2004.
- [13] D. Yu, S. K. Jang and J. B. Ra,"Fast Motion Estimation for Shape Coding in MPEG-4," IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no.4, pp: 358 – 363, April 2003.
- [14] K. Panusopone and X. Chen," A fast motion estimation method for MPEG-4 arbitrarily shaped objects," in Proceeding of IEE International Conference on Image Processing, vol. 3, pp: 624-627, Sept. 2000.
- [15] Peter Kuhn,"Algorithms, Complexity Analysis and VLSI Architectures for MPEG-4 Motion Estimation," Kluwer Academic Publishers, 1999.
- [16] J. Jain and A. Jain, "Displacement measurement and its application in internal image coding," IEEE Transactions on Communications, vol. COM-29, no. 12, pp: 1799-1808, Dec. 1981.
- [17] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in Proceedings of National Telecommunication Conference, pp: C9.6.1-C9.6.5, 1981.
- [18] R. Srinivasan and K. R. Rao,"Predictive coding based on efficient motion estimation," IEEE Transactions on Communications, vol. COM-33, no. 8, pp: 888-896, Aug. 1985.
- [19] S. Kappagantula and K. R. Rao, "Motion compensated interframe image prediction," IEEE Transactions on Communications, vol. COM-33, no. 9, pp: 1011—1015, Sept. 1985.
- [20] M. Ghanbari, "The cross search algorithm for motion estimation," IEEE Transactions on Communications, vol. 38, no. 7, pp: 950-953, July 1990.
- [21] L. G. Chen, W. T. Chen, Y. S. Jehng, and T. D. Chiueh, "An efficient parallel motion estimation algorithm for digital image processing," IEEE Transactions on Circuits and Systems for Video Technology, vol. 1, no. 4, pp: 378-385, Dec. 1991.
- [22] M. J. Chen, L. G. Chen, and T. D. Chiueh, "One-dimensional full search motion estimation algorithm for video coding," IEEE Transactions on Circuits and Systems for Video Technology, vol. 4, no. 5, pp: 504-509, June 1994.

- [23] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 4, pp: 438-442, Aug. 1994.
- [24] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp: 313-317, June 1996.
- [25] L. K. Liu and E. 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, Aug. 1996.
- [26] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, no. 4, pp: 369-377, Aug. 1998.
- [27] S. Zhu and K. K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no. 2, pp:287-290, Feb. 2000.
- [28] A. M. Tourapis, O. C. Au, M. L. Liou, G. Shen, and I. Ahmad, "Optimizing the MPEG-4 encoder - advanced diamond zonal search," in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp: 674-677, 2000
- [29] A. M. Tourapis, O. C. Au, and M. L. Liu, "Highly efficient predictive zonal algorithms for fast block-matching motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 10, pp: 934-947, Oct. 2002.
- [30] V. Christopoulos and J. Comelis, "A center-biased adaptive search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 3, pp: 423-426, Apr. 2000.
- [31] O. T. C. Chen, "Motion estimation using a one-dimensional gradient descent search," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 4, pp: 608-616, June 2000.
- [32] C. H. Cheung and L. M. Po, "A novel cross diamond search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 12, pp: 1168-1177, Dec. 2002.

- [33] Y. W. Huang, S. Y. Ma, C. F. Shen, and L. G. Chen, "Predictive line search: an efficient motion estimation algorithm for mpeg-4 encoding systems on multimedia processors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 1, pp: 111-117, Jan. 2003.
- [34] C. W. Lam, L. M. Po, and C. H. Cheung, "A novel kite-cross-diamond search algorithm for fast video coding and videoconferencing applications," in *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp: 365-368, 2004.
- [35] M. Bierling, "Displacement estimation by hierarchical block matching," in *Proceedings of SPIE Visual Communication and Image Processing*, pp: 942-951, 1988.
- [36] H. Gharavi and M. Mills, "Block matching motion estimation algorithms - new results," *IEEE Transactions on Circuits and Systems*, vol. 37, no. 5, pp: 649-651, May 1990.
- [37] J. S. Kim and R. H. Park, "A fast feature-based block matching algorithm using integral projections," *IEEE Journal on Selected Areas in Communications*, vol. 10, no. 5, pp: 968-979, June 1992.
- [38] K. Sauer and B. Schwartz, "Efficient block motion estimation using integral projections," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 5, pp: 513-518, Oct. 1996.
- [39] B. Natarajan and V. Bhaskaran, "Low-complexity block-based motion estimation via one-bit transforms," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 4, pp: 702-706, Aug. 1997.
- [40] J. H. Luo, C. N. Wang, and T. Chiang, "A novel all-binary motion estimation (ABME) with optimized hardware architectures," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 8, pp: 700-712, Aug. 2002.
- [41] Z. L. He, C. Y. Tsui, K. K. Chan, and M. L. Liou, "Low-power VLSI design for motion estimation using adaptive pixel truncation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 5, pp: 669-678, Aug. 2000.
- [42] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing*, vol. 4, no. 1, pp: 105-107, Jan. 1995.
- [43] Digital Video Coding Group, ITU-T recommendation H.263 software implementation, Telenor R&D, 1995.

- [44] J. N. Kim and T. S. Choi, "A fast full-search motion-estimation algorithm using representative pixels and adaptive matching scan," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 7, pp: 1040-1048, Oct. 2000.
- [45] Y. W. Huang, S. Y. Chien, B. Y. Hsieh, L. G. Chen, "An efficient and low power architecture design for motion estimation using global elimination algorithm," in *Proceedings of Acoustics, Speech, and Signal Processing*, vol. 3, pp: 3120-3123, May 2002,.
- [46] K. B. Lee, H. Y. Chin, H. C. Hsu, C. W. Jen, "QME: an efficient subsampling-based block matching algorithm for motion estimation," *IEEE International Symposium on Circuits and Systems*, vol. 2, pp:II -305 – II-308, May 2004
- [47] Y. L. Chan, W. C. Siu, "An adaptive partial distortion search for block motion estimation," *IEEE International Conference on Acoustics, Speech and Signal Processing*, vol.3, pp:153-156, April 2003.
- [48] C. H. Cheung, L. M. Po, "Adjustable partial distortion search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol.13, no. 1, pp: 100-110, Jan. 2003.
- [49] S. Y. Kung, *VLSI Array Processors*, Englewood Cliffs, NJ: Prentice Hall, 1988.
- [50] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 2, pp: 1301-1308, Oct. 1989.
- [51] L. D. Vos and M. Stegherr, "Parameterizable VLSI architectures for the full-search block-matching algorithm," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 2, pp: 1309-1316, Oct. 1989.
- [52] K. M. Yang, M. T. Sun, and L. Wu, "A family of VLSI designs for the motion compensation block-matching algorithm," *IEEE Transactions on Circuits and Systems*, vol. 36, no. 2, pp: 1317-1325, Oct. 1989.
- [53] A. Hanami, S. Scotzniovsky, K. Ishihara, T. Matsumura, S. I. Takeuchi, H. Ohkuma, K. Nishigaki, H. Suzuki, M. Kazayama, T. Yoshida, and K. Tsuchi-hashii, "A 165-GOPS motion estimation processor with adaptive dual-array architecture for high quality video-encoding applications," in *Proceedings of IEEE Custom Integrated Circuits Conference*, pp: 169-172, 1998

- [54] H. M. Jong, L. G. Chen, and T. D. Chiueh, "Parallel architectures for 3-step hierarchical search block-matching algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no. 4, pp: 407-416, Aug. 1994.
- [55] S. Dutta and W. Wolf, "A flexible parallel architecture adopted to block-matching motion estimation algorithms," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 1, pp: 74—86, Feb. 1996.
- [56] H. D. Lin, A. Anesko, and B. Petryna, "A 14-GOPS programmable motion estimator for H.26X video coding," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 11, pp: 1742-1750, Nov. 1996.
- [57] V. G. Moshnyaga, "A new computationally adaptive formulation of block-matching motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 1, pp: 118-124, Jan. 2001.
- [58] S. C. Hsia, "VLSI implementation for low-complexity full-search motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 7, pp: 613-619, July 2002.
- [59] C. D. Vleeschouwer, T. Nilsson, K. Denolf, and J. Bormans, "Algorithmic and architectural co-design of a motion-estimation engine for low-power video devices," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 12, pp: 1093-1105, Dec. 2002.
- [60] W. M. Chao, C. W. Hsu, Y. C. Chang, and L. G. Chen, "A novel motion estimator supporting diamond search and fast full search," *Proceedings of IEEE International Symposium on Circuits and Systems*, pp: 492-495, 2002.
- [61] S. Y. Yap, J. V. McCanny, "A VLSI architecture for variable block size video motion estimation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol 51, no. 7, pp: 384 – 389, July 2004.
- [62] Y. W. Huang, T. C. Wang, B. Y. Hsieh, L. G. Chen, "Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264," *Proceedings of the International Symposium on Circuits and Systems*, vol 2, pp: II-796 - II-799, May 2003.
- [63] B. S. Kim, J. D. Cho, "Maximizing memory data reuse for lower power motion estimation," *Proceedings of the 10th Great Lakes Symposium on VLSI*, pp.133-138, March 2000.

- [64] J. C. Tuan, T. S. Chang, and C. W. Jen, "On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, pp: 61-72, Jan. 2002.
- [65] P.I.Hosur and K.K.Ma, "Motion Vector Field Adaptive Fast Motion Estimation", *Second International Conference on Information Communications and Signal Processing*. Singapore. Dec. 1999.
- [66] A. M. Tourapis, O. C. Au, M. L.Liou, "New results on zonal based motion estimation algorithms-advanced predictive diamond zonal search," *International Symposium on Circuits and Systems*, vol 5 , pp: 183 -186, 2001
- [67] T. Wiegand, M. Lightstone, T. G. Campbell, 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*, vol. 6, no. 2, pp: 182-190, Apr. 1996.
- [68] H. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp: 598-603, July 2003.
- [69] D. Marpe, H. Schwarz, and T. Wiegand, "Context based adaptive binary arithmetic coding in the H.264/AVC video coding compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp:620-636, July 2003.
- [70] H. Kim and Y. Altunbasak, "Low-Complexity Macroblock Mode Selection for H.264/AVC. Encoders," *IEEE International Conference for Image Processing*, Oct. 2004.
- [71] Yanfei Shen, Dongming Zhang, Chao Huang, Jintao Li, "Fast mode selection based on texture analysis and local motion activity in H.264/JVT," *International Conference on Communications, Circuits and Systems*, vol. 1, pp:539 – 542, June 2004.
- [72] X. Song, T. Chiang, X. Lee, and Y. Q. Zhang, "New fast binary pyramid motion estimation for MPEG2 and HDTV encoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, pp: 1015–1028, Oct. 2000.
- [73] J. H. Luo, C. N. Wang, and T. Chiang, "A novel all binary motion estimation (ABME)," *IEEE International Symposium on Circuits and Systems*, vol. 2, pp:II-480 - II-483. May 2002

- [74] "Memory Organization and Its Interface to PEs for Motion Estimation," Jen-Chien Tuan, Master thesis. NCTU
- [75] "Algorithm and Architectuer Design for Motion Estimation, H.264/AVC Standard, and Intelligent Video Signal processing" Yu-Wen Huang, Ph.D thesis, NTU.
- [76] "Motion Estimation Engine for MPEG-4 Video," Hao-Yun Chin, Master thesis, NCTU.

