MPEG-4 Software Video Encoding

Hamosfakidis, Anastasios

QUEEN MARY
AND WESTFIELD COLLEGE
UNIVERSITY OF LONDON

Department of Computer Science

Research Report No. RR-01-05 ISSN 1470-5559 April 2001

MPEG-4 Software Video Encoding Using Parallel Processing

Anastasios Hamosfakidis

MPEG-4 SOFTWARE VIDEO ENCODING USING PARALLEL PROCESSING

Anastasios Hamosfakidis

A Thesis submitted in fulfilment of the requirements
of the degree of doctor of Philosophy
in the University of London

Department of Computer Science
Queen Mary and Westfield College
University of London.

2001

Abstract

This thesis presents a software model that allows a parallel decomposition of the MPEG-4 video encoder onto shared memory architectures, in order to reduce its total video encoding time.

Since a video sequence consists of video objects each of which is likely to have different encoding requirements, the model incorporates a scheduler which

(a) always selects the most appropriate video object for encoding and,

(b) employs a mechanism for dynamically allocating video objects allocation onto the system processors, based on video object size information.

Further spatial video object parallelism is exploited by applying the single program multiple data (SPMD) paradigm within the different modules of the MPEG-4 video encoder. Due to the fact that not all macroblocks have the same processing requirements, the model also introduces a data partition scheme that generates tiles with identical processing requirements. Since, macroblock data dependencies preclude data parallelism at the shape encoder the model also introduces a new mechanism that allows parallelism using a circular pipeline macroblock technique

The encoding time depends partly on an encoder's computational complexity. This thesis also addresses the problem of the motion estimation, as its complexity has a significant impact on the encoder's complexity. In particular, two fast motion estimation algorithms have been developed for the model which reduce the computational complexity significantly.

The thesis includes experimental results on a four processor shared memory platform, Origin200 that demonstrate the efficiency of the model in terms of parallel speedup.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor, Professor Yakup Paker for his guidance and encouragement. His support has greatly contributed to my research and to the preparation of this thesis. I have truly benefited and enjoyed working with him.

I would also like to express my sincere appreciation to my graduate committee members, Dr Alan Pearmain, and Dr. Sylvia Wilbur, for their support.
I wish to thank the IRISA INRIA Rennes, France, the CSAR centre of University of Manchester, and the MoMuSys European ACTS project for supporting this research. The TMR programme of the European Union provided me with a very generous research award, for which I am grateful.

I would also like to acknowledge to Hanin Naguib, David Hawes, and Dr William Ferreira for their support in preparing this thesis.

Finally, I owe a considerable debt of gratitude to my parents, who have provided their unconditional support for this adventure, in the way that only parents can.

Table of Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

1.1 Introduction to Video Compression

Moving images represented as a digital video have become an important commodity in our society, and are increasingly used in areas such as education, communication, entertainment and publishing [Smoliar 94]. Demand for digital video has been growing at a phenomenal rate in the past few years. One of the major obstacles to deploying digital video in many applications is the fact that huge amount of digital data is required to represent uncompressed video which easily overwhelm the available storage systems and communication channels. For example, a digital video sequence that conforms to the ITU-R 601 digital video recommendation (720x486 pixels per frame, 30 frames per second and 16 bits per pixel), which has a resolution comparable to the National Television System Committee (NTSC) analog video signal, has an uncompressed rate of 168 megabits per second (Mbps) [Rec. 601, 90] [Tekalp 95]. That means that a typical two hour movie would occupy approximately 150 giga-bytes of disk space. In order to transmit, access and store video, this colossal quantity of data obviously needs to be compressed.

In the traditional representation a video signal contains sample redundancy [Rabbani 91]. Redundancy can be classified as *spatial* due to the correlation between the neighbouring pixel values, *spectral* due to correlation between the

different colour planes /spectral bands, or *temporal* due to correlation between subsequent frames in a video stream.

The main objective of a video compression algorithm is to exploit both the spatial and temporal redundancy of a video sequence such that fewer bits needs to be used to represent a video sequence up to an acceptable visual distortion.

During the past decade various compression methods [Watkinson 95] have been developed to overcome the problem of compression leading to a number of encoding standards such as H.261 [CCITT 90], H.263 [ITU-T 96], MPEG-1 [11172-2, 93], MPEG-2 [13818-2, 94], [Mitchell 96], [Eckart 95], and MPEG-4.

The current MPEG-4 work item of ISO MPEG, first proposed in May 1991 and approved in July 1993, initially targeted audiovisual coding at a very low bit rate. The motivation for MPEG-4 resulted from the common belief that either MPEG-1 or MPEG-2 was inadequate for a variety of applications where either the channel has a low bit rate or the storage medium has a reduced capacity, but that there is a need to store long audiovisual sequences. Soon the scope of the MPEG-4 changed. It has become a coding standard that supports new ways for communication, access, and manipulation of digital audiovisual data. The focus of MPEG-4 [Ebrahimi 97], [Koenen N3156, 99], [Koenen N2723, 99] [Bhaskaran 97]. [IEEE CSVT 97] is the convergence of common applications of three major and largely interrelated industries: telecommunications, computing and TV/film. MPEG-4 supports standard functionalities such as synchronisation, low delay mode, virtual channel allocation, flexibility, security, the ability to handle various audiovisual contents and formats, and the ability to access quality of decoded audio or video. Additionally, eight key functionalities have been identified which support the MPEG-4 focus and are not thought to be well-supported by existing (MPEG-1/2) or other emerging standards. These are: content-based manipulation and bitstream

editing, content-based multimedia tools, content-based scalability, coding of multiple concurrent data streams, hybrid and synthetic coding, improved coding efficiency, improved temporal random access at very low-bit rates, and robustness in error prone environments.

Moreover, with a flexible toolbox approach, MPEG-4 is capable of supporting diverse new functionalities, and hence will cover a broad range of present and future multimedia applications such as, content-based storage and retrieval, surveillance, studio/television post production, and mobile multimedia applications.

1.2 Research Problem and Motivation

At present very few MPEG-4 video coding hardware solutions are available, as it was found that dedicated hardware is less flexible and could become obsolete. A hardware encoder is often optimised [Gove 94] [Hamosfakidis 97] for a particular coding algorithm and therefore could not be used for other coding algorithms. Thus software-based implementations have been found to be a more viable solution. A software encoder allows flexibility since coding algorithms such as motion estimation, and discrete cosine transformations (DCT), can be substituted by other more efficient algorithms, and permits the inclusion of new tools, which are desirable features for the MPEG-4 based multimedia applications. In addition, it offers portability and reusability as a software encoder could be ported easily from one system to another.

The main obstacle of software encoding is its poor encoding speed compared to hardware encoding. To overcome this, parallel processing can be used to which the structure of the video encoder happens to be very suitable. More specifically, a video sequence is a series of frames in time, and therefore a natural choice is to parallelise the encoder at the temporal dimension of the video signal. In temporal parallelism, processors are assigned groups of frames from a video sequence. The

sequence is split up into groups of frames and each processor works on its own group. There are no data locality problems since the whole frame is assigned to the processor and no upper limit on the number of processors that can be used- a typical two hour movie contains around 200,000 frames. Additionally, parallelism can be applied to the spatial domain of the video sequence. The spatial parallelism is accomplished by dividing the video frame into tiles where each tile contains a given number of blocks of pixels (macroblocks). The tiles are assigned to the system processors, [Tan 95], [Yu. 94], [Moulin 95], [Akramullah 97] [Huang 94], and the video sequence is encoded in a frame by frame fashion.

In past decades a lot of research effort was spent on speeding up the software video encoders by using parallel processing. In the literature, two well known implementations of the MPEG-1 encoder, that uses the temporal approach, are mentioned: (a) The Berkeley's MPEG-1 encoder [Gong 94] that distributes the frames to a cluster of workstations interconnected over the LAN/internet, and (b) a modified version of this [Gong 95] that encodes frames in real time. A portable and scalable parallel implementation of the MPEG-2 video encoder that achieves real time encoding is presented in [Akramullah 97]. This implementation is based on spatial parallelism, where the selection of the appropriate video data structure and its size plays a key role [Moulin 95] [Akramullah 97], since motion estimation needs to use data from a wide area of the frame that might not be covered by the frame tile.

Finally, a spatial-temporal parallel video encoder is presented in [Shen 96]. In this approach, the system processors are divided into groups and the frame sequence is divided into sections, with each section assigned to a group of processors. Spatial parallelism occurs inside the processors group by dividing frames into tiles.

However, the parallel processing of the MPEG-4 software video encoder is not a trivial task that could be accomplished by using straightforward data-partitioning methods or multitasking. Specifically, with the MPEG-4 approach a frame is more likely to be composed of distinct video objects, arbitrarily shaped, with sizes varying over time. Moreover, these video objects need to be tightly synchronised when are encoded. The orchestration and the allocation of the MPEG-4 video objects onto a parallel architecture for processing presents an interesting challenge.

Since the time required to compress a video sequence depends both on the underlying hardware processing power, and on the encoder's computational complexity, this thesis also addresses the problem of reducing the encoder's complexity, especially as motion estimation complexity dominates and determines the video encoder's complexity.

1.3 Contributions of the thesis

This thesis proposes a software model that (a) reduces the MPEG-4 video encoder's complexity and (b) permits parallel processing in an optimised way that can result (under circumstances: availability of hardware resources- number of processors) in real time encoding. The major contributions of this work are discussed in the following subsections

- **A Data Partition Shape Adaptive Scheme for Motion Estimation and Texture Coding**

Load balancing in the MPEG-1, and MPEG-2 standards is achieved by dividing frames into equally sized tiles, and assigning them to the processors. In MPEG-4, macroblock processing requirements vary significantly, depending exclusively on the macroblock location in the video object. Load balancing among the system

processors cannot be accomplished by dividing the object into equally sized tiles, since the tiles are likely to have different processing requirements. Therefore, a new data partition scheme is proposed for the MPEG-4 parallel video encoding that divides the frame into tiles based on the macroblocks real processing requirements.

- **A Fast Parallel Scheme for MPEG-4 Binary Shape Coding**

Data parallelism is not straightforward in the shape encoder, due to macroblock dependencies which restrict the degree of parallelism. The proposed scheme reduces the shape encoder's computational complexity by using a fast search algorithm in shape motion estimation, and introduces a circular pipeline technique that makes enables parallel shape encoding.

- **Two Block-based Fast Motion Estimation Algorithms**

Motion estimation plays an important role in video encoding due to the fact that it has a significant impact on the quality of the reconstructed video sequence and makes a considerable contribution to the encoder's complexity. Two fast motion estimation algorithms are introduced that reduce the computational complexity of the motion estimation. More specifically:

(i) "A Fast Motion Prediction Scheme for bi directional encoding". This uses a two level searching technique in order to locate the region of the search area where the motion occurs. This scheme can be combined with other categories of fast motion estimation algorithms that involve pixel subsampling or subsampled motion-field estimation (block sub sampling) or subblock motion-field estimation (smaller block size is used) for further speed improvement.

(ii) "A Hexagonal Search Algorithm for Fast Block Matching Motion Estimation". Based on real word image sequence characteristics of centre-biased motion vector distribution, this algorithm employs a two search pattern scheme that

outperforms some of the well known fast motion estimation algorithms, making it suitable for real time encoding

- **A Heuristic Scheduler Suitable for the Parallel MPEG-4 Video Encoding onto Shared Memory Architectures.**

This thesis addresses the problem of producing a schedule for the encoding of more than one video object, onto a platform of more than one processor. It has been proven in [El-Rewini 94] [Papadimitriou 79][Ulman 75] that finding an optimal schedule for a set of tasks is NP-complete in its general case, and in several restricted ones. Only for a few restrictive cases [Hu 61][Coffman 76][Gabow 82] can scheduling be found by optimal algorithms. These cases are far removed from practice involving massive parallel processing. To deal with these cases heuristic solutions have been introduced. Generally, heuristics depend upon several parameters of scheduling (tasks, target machine, etc.) and are characterised by their ability to produce answers in less than exponential time, but without guaranteeing an optimal solution. A heuristic scheduler has been proposed for shared memory architectures, that selects the video objects according to their deadlines, precedence constraints and frame rates, while ensuring maximum utilisation of the system processors by using dynamic video object allocation policies.

1.4 Organisation of the Thesis

The thesis is organised as follows. The research problem is stated, with an overview of the proposed model that will be used for the parallel MPEG-4 encoding. The chapter concludes by addressing issues related with the MPEG-4 software, and the hardware platform.

Chapter 3 deals with motion estimation complexity and its impact on the overall performance at the MPEG-4 video encoder. It presents two new fast block-based motion estimation algorithms that are used by the model.

Chapter 4 deals with scheduling issues arising from the parallel MPEG-4 video encoding. It proposes a heuristic scheduler that is used by the parallel model in order to prioritise and orchestrate the encoding of the MPEG-4 video objects onto the parallel architecture.

Chapter 5 discusses a data partition scheme that is applicable for the spatial parallelism of the MPEG-4 video objects onto a parallel architecture. Load balancing and system utilisation are also addressed.

Chapter 6 proposes the new parallel scheme that is suitable for the fast parallel MPEG-4 shape encoding.

Chapter 7 presents experimental results of the performance of the proposed MPEG-4 software parallel model, when it is used for the encoding of MPEG-4 segmented video sequences with different characteristics, onto a four processor shared memory architecture (SGI Origin200). The conclusions of this thesis, and future work are discussed in Chapter 8.

Chapter 2

2 The MPEG-4 Video Encoder Problem

MPEG-4 is designed to address a wide range of multimedia applications, which cover noninteractive video communications (e.g., multimedia broadcasting), digital storage media (e.g., optical disks), content-based image and video databases, interactive video communications (e.g, video conferencing and telephony), video surveillance, and interactive video games [Diepold 98]. Interactive video communications and video surveillance require real time encoding, with encoding frame rates not less than 25 frames per second.

We found that the encoding frame rate for the News QCIF (176*144 pixels) video sequence is 1,2 frames per second when it is encoded on a single processor (R10000) SGI Onyx workstation, using the final committee draft (FCD) of the MPEG-4 software codec[1] [Simulation 98]. Similarly, for the CIF resolution (352*288 pixels) Coastguard video sequence the encoding frame rate is 0.2 frames per second. Rates far removed from real time encoding.

In Chapter 1 we mentioned that the encoding time of a video compression standard depends both on the processing power of the hardware platform used, and on the computational complexity of various algorithms. The rest of this chapter gives a brief technical description of the MPEG-4 video object coding, states the research

[1] Similar to MPEG-1, and MPEG-2 MPEG-4 developed simulation models in order to have a reference video quality, where a simulation model contains a specific reference implementation of the MPEG-4 encoder and decoder, including all the details that are not specified in the standard.

problem of speeding up the software MPEG-4 encoder, and describes the approach that is adopted to implement a fast software parallel MPEG-4 video encoder onto shared memory architectures.

2.1 The Research problem

2.1.1 MPEG-4 video encoder computational complexity

The MEPG-4 video encoder encodes video objects, where a video object (VO) is an arbitrarily shaped video segment that has a semantic meaning. A 2-D snapshot of a video object at a particular time instant is called a video object plane (VOP). A VOP is defined by its texture (luminance and chrominance values) and its shape. MPEG-4 allows content based access to not only the video objects, but also temporal instances of the video objects, i.e, VOPs. In general, MPEG-4 coding of a VOP involves coding of motion, texture and shape information. However, when the VOP is a rectangularly shaped video frame, MPEG-4 video encoding becomes quite similar to that specified in MPEG-1/MPEG-2.

The MPEG-4 video object coding consists of shape coding (for arbitrarily shaped VOs), motion compensated prediction to reduce temporal redundancies, and DCT-based texture coding of the motion compensated prediction error data to reduce spatial redundancies. The video coding is performed at the macroblock level. VOPs are divided into macroblocks, such that they are represented with the minimum number of macroblocks within a bounding rectangle. Similar to MPEG-1 and MPEG-2, MPEG-4 supports intracoded (I), temporally predicted (P), and bidirectionally predicted (B) VOPs.

Figure 2.1 shows the basic VOP encoder structure. The encoder consists mainly of two parts: a hybrid of a motion compensated predictor and a DCT-based coder [Scafer 95] [Steinmentz 94], and a shape encoder.In the first part, motion

estimation and compensation are performed (except I-VOPs) on texture data, followed by DCT and quantisation. Then, the difference between the predicted data and the original texture data is coded by variable length coding (VLC). Motion information is also encoded by using VLC. Then, the VOP is reconstructed as in the decoder, that is, by applying inverse quantisation , applying inverse DCT (IDCT) and adding the resulting data to the motion compensated predicted data. The resulting VOP is then used for the prediction of future VOPs. The shape encoder encodes the binary shape information and the transparency information of the object. Since the shape of the VOP may not change significantly between consecutive VOPs, predictive coding is employed to reduce temporal redundancies. Thus motion estimation and compensation are also performed for the shape of the object. Finally, motion texture, and shape information is multiplexed with the headers to form the coded VOP bit stream

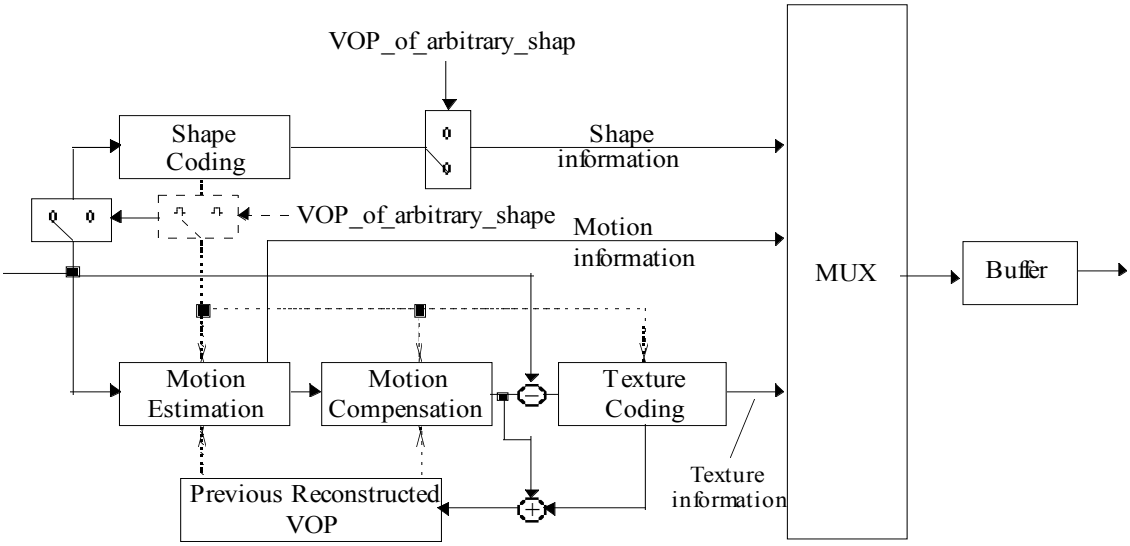


Figure 2.1: Basic MPEG-4 video encoder structure

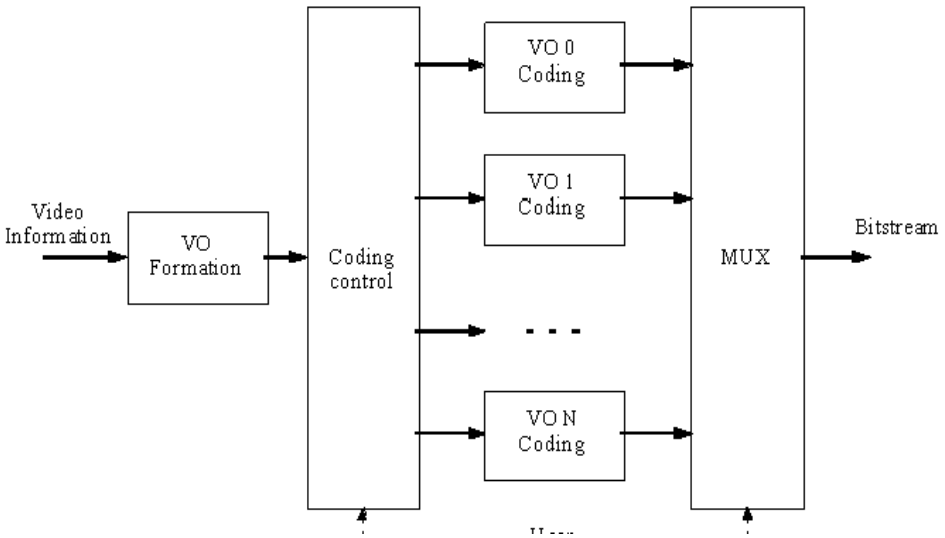
By performing benchmark measurements with the FCD MPEG-4 video encoder we found that motion estimation is the most computationally demanding part of the encoder, (60%-80% of total encoding time) with significant impact on the image quality and the overall encoding time. The (FCD) codec uses the full search (FS) block matching algorithm which is the simplest, and provides an optimal solution

by exhaustively evaluating all the possible candidates within the search range in the reference frame. However the computational complexity of the FS algorithm prevents its use in real time encoding. Since the MPEG-4 does not specifies the motion estimation scheme that is used in the encoder, this thesis considers the problem of motion estimation's computational complexity and in Chapter 3 proposes two fast motion estimation algorithms that reduce significantly the complexity of the motion estimation searching process, while accomplishing very small distortions of the image quality, compared to the full search algorithm.

2.1.2 Parallel processing of the MPEG-4 Video Encoder

2.1.2.1 MPEG-4 Video Encoder and Parallel Processing

The advantage of a software based MPEG-4 video encoder has already been mentioned. However the computational requirements of fast (real time) encoding that achieves encoding rates of 25-30 frames per second far exceed the processing power of today's workstations or a single processors PCs, even if fast motion estimation algorithms are used. Parallel processing is a natural choice to meet the large computational requirements of MPEG-4 video encoding. The MPEG-4 video encoder software structure happens to be very suitable for parallel computing, since it deals with the concurrent encoding of a frame's video objects, figure 2.2. Each frame consists of one or more arbitrarily shaped video objects, where their shape and number is provided to the MPEG-4 video encoder. The MPEG-4 encoding of a video frame is performed by decomposing the frame into its video



objects, and then for each video object independent encoding is applied.

Figure 2.2: MPEG-4 encoder software structure

Thus, parallelism can be achieved simply by creating multiple instances of the MPEG-4 video encoder that can concurrently encode the frame's video objects. With this straightforward approach a coarse spatial parallelism has been introduced by applying the single program multiple data (SPMD) programming paradigm [Williams], where every processor runs an instance of the MPEG-4 encoder.

A "producer-consumer software" model was introduced in [Hamosfakidis 98] that applies this coarse spatial parallelism using thread programming [Lewis 95][Kleiman 95][Stein 92], figure 2.3, for a shared memory platform. A thread is a different stream of control that executes each instruction independently allowing a multithread process to perform numerous tasks concurrently.

For each video object of the frame a thread is created that runs an instance of the MPEG-4 encoder. These threads are called consumers. Initially in the system only one thread exists: the producer thread. The producer is assigned the tasks of: (a) identifying the number of video objects that compose the frame, (b) creating the MPEG-4 video encoders the consumers, and (c) to distributing the video objects. A naive scheduler is implemented in producer which ensures the continuous flow of the video object data to the corresponding consumers. Between the producer and the consumers exist synchronisation buffers. When a consumer encodes a video object it sets the status of its corresponding synchronisation buffer. The producer continuously checks the status of these buffers. If a buffer is not set, the producer signals the consumer to proceed with the encoding of its next video object plane (VOP), otherwise if the status flag of the buffer is set the producer thread blocks waiting signal from the consumer. When a consumer has finished the encoding of its video object for the entire video session it notifies the producer. When all

consumers have notified the producer that their encoding has finished the producer terminates them.

The above scheme does not exploit efficiently the computational power of the parallel architecture, since concurrency is restricted to the number of video objects that make up the frame, and which is always constant. The degree of parallelism is rather limited and the system is not scalable. Moreover, the model lacks the load balancing aspect, as video objects vary in size over the time of the encoding session. It is more likely some system processors are heavily loaded since their assigned video objects are large, while other system processors are inactive for long periods of the encoding session, since their assigned video objects are small. Load balancing occurs only when the consumers encoding VOPs of equal size.

A solution to the load balancing problem is to use a video object's spatial parallelism [Hamosfakidis MPEG-4, 97]. The video object is divided into tiles and each tile is assigned to a processor (data-parallel model). However the spatial techniques that were discussed in the previous chapter, are not feasible for MPEG-4 video encoding, since they have been developed for fixed size frames whose data processing requirements are identical for any part of the frame. Additionally, these approaches lack the scheduling aspect that arises in MPEG-4 video encoding.

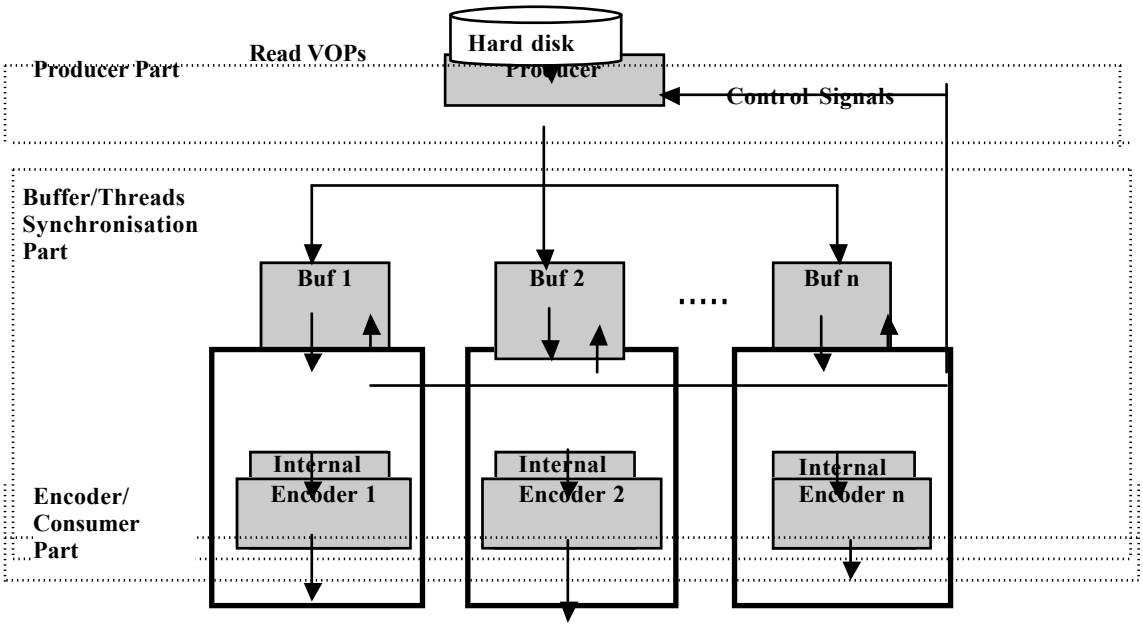


Figure 2.3: Producer-Consumers model for the parallel VOPs encoding

2.1.2.2 Issues arising from the parallel MPEG-4 Video Encoding

Besides the synchronisation and the allocation of the video objects onto a parallel architecture in order to achieve fast (real time) MPEG-4 parallel encoding the following issues need to be carefully addressed.

1. *Coding constraints imposed by the coding pattern*:

I-VOPs have the highest data rates, the lowest motion artefacts, and are used to provide functionality such as random access, and fast forward/backward. On the other hand B-VOPs have the lowest data rates and the highest motion artefacts. Moreover, I and P VOPs are used as references. I VOPs are encoded independently, while P-VOPs are encoded using previously encoded I or P VOPs. B-VOPs are encoded using a previously encoded I or P, VOP and a future prediction of an I or P VOP. The set of all acceptable MPEG coding patterns (SAMCP) is given by:

[Hamosfakidis IEEE MS 99][2]

where the analogy of I- P - and B- VOPs in an "intra period" depends on a number of factors such as quantisation level, quality of the picture source, and type of encoder (open-loop or closed loop) [Wilkinson 96]. The most commonly used MPEG coding patterns are:

(i) $I_1P_2P_3P_4P_5P_6P_7P_8P_9$

(ii) $I_1P_2B_3P_4B_5P_6B_6P_7B_8$.

[2] ($P + I$) means either P or I VOP and

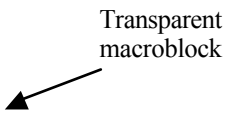
Thus, from this discussion it is obvious that VOP encoding needs to be done according to coding pattern constraints; for example the P_3 VOP cannot be encoded before the P_2 VOP in the (i), or the B_3 VOP before the P_2 VOP in (ii).

2. *Encoding deadlines:*

In real time encoding the frame has to be encoded before the expiration of its assigned deadline. In the simplest case the encoding deadline calculated using the sequence frame rate. For instance, if the frame rate of a video sequence is 25 frames per second then the time that the frame needs to complete its encoding is 1/25 sec.

3. *Video object formation:*

In order to describe an arbitrarily shaped VOP, MPEG-4 defines a VOP by means of a bounding rectangle. This bounding box surrounds the video object with the minimum number of macroblocks, see figure 2.4. As is shown in this figure, there are three different types of macroblocks depending on whether their pixels are inside or outside the VOP (transparent, opaque, and contour). Transparent macroblocks lie completely outside the video object while opaque macroblocks lie inside the video object and contour macroblocks lie partly inside the object. Each macroblock type has different processing requirements for each part of the MPEG-4 video encoder. Thus, spatial VOP parallelism based on the current data partitioning techniques would soon lead to an unbalanced processing system.



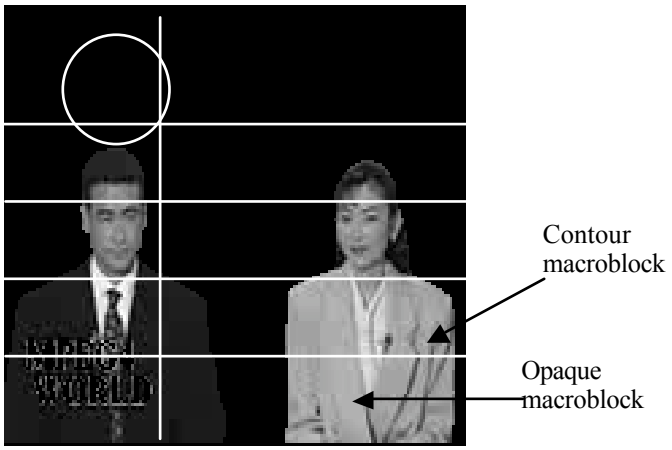


Figure 2.4: The different types of macroblocks in the VOP (News 2)

From the above discussion it can be concluded that the implementation of a software encoder on a parallel platform is not a trivial task, and previous parallel video encoding solutions cannot be applied in a straightforward way. A "sophisticated" software model is required, that incorporates a scheduler [Hamosfakidis IEEE MS 99] which calculates video object deadlines from the sequence, and distributes the VOPs according to their precedence constraints and their calculated deadlines to the parallel platform. In addition, the scheduler should dynamically allocate the VOPs to the group of processors in order to achieve load balancing.

The next section describes the framework of the proposed model, while analytical descriptions of each of the techniques, schemes, and algorithms that the model incorporates will be provided in the following chapters

2.2 Proposed framework for real time MPEG-4 parallel encoding

The target architecture plays an important role in the selection of the heuristic scheduler, and the parallel programming technique that will be adopted to implement the proposed model. This thesis deals with the parallel processing of the MPEG-4 video encoder on a shared memory platform, where parallelism is provided by processes and threads. More specific, the operating system assigns threads and processes to the system processors, therefore by increasing or decreasing the number of threads/processes we increase or decrease the degree of the parallelism employed in the system. Maximum system parallelism is achieved when the number of threads/processes that run on the system equals to, or is greater than the number of processors.

Initially the producer in our proposed model runs as a process that implements the scheduler. In order to satisfy real time encoding requirements the scheduler needs to sort the VOPs according to their deadlines in a list where the VOP with the earliest deadline is selected first. When the list of the VOPs of all the video objects is formed the scheduler (producer) creates the consumers and distributes the VOPs to them. For each video object in the sequence, the producer creates a consumer process that runs an instance of the MPEG-4 video encoder. There are synchronisation buffers between the producer and the consumers that control the flow of a VOP's data to the model. More specifically, there is a flag for each synchronisation buffer that is set when its encoder is encoding a VOP. The producer selects the VOP with the earliest deadline from the list and checks the flag status of the synchronisation buffer of the corresponding consumer for this video object. If the flag is off the consumer sets it and starts the VOP encoding, while the producer proceeds with the next VOP in the list. If all the flags of all the synchronisation buffers are set, then the producer blocks until a consumer unblocks it by signalling for new VOP data.

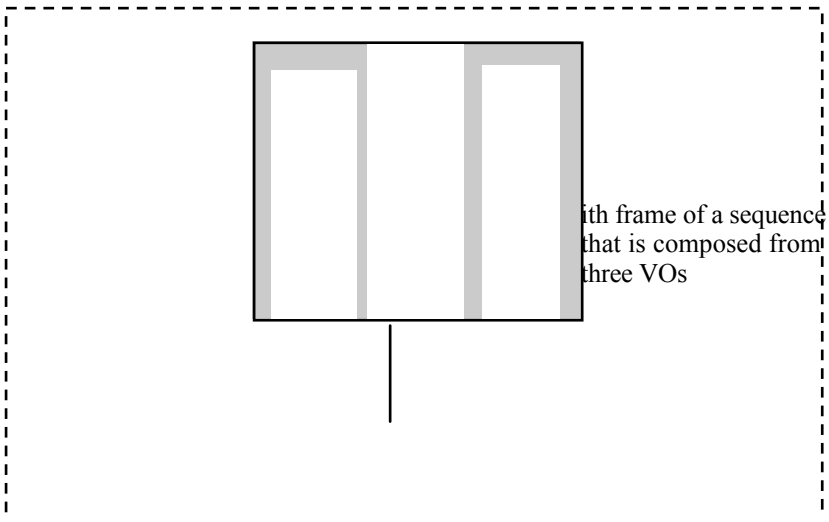
In order to facilitate VOP assignments to dynamically formed group of processors, the producer provides the consumers with the VOP size, which is calculated by the producer as a percent of the whole frame size. Each consumer in order to better exploit the system parallelism, applies the SPMD paradigm for every part of the MPEG-4 encoding procedure. Specifically, every time that the consumer enters either the motion estimation/compensation, the texture, or the shape coding module, it create threads that run multiple instances of this module on different data areas of the VOP. Therefore, by increasing or decreasing the number of threads created, the number of system processors that are allocated to the VOP is increased or decreased. Since maximum system parallelism is exploited when the number of threads equals to or is greater than the number of system processors, the number of threads that a consumer can create is determined by multiplying the VOP size (given as a percentage of the frame size) with the number of threads that exploit the maximum system parallelism.

Figure 2.5 demonstrates the proposed dynamic allocation mechanism when three video objects are encoded on a shared memory platform which consists of seven processors, P_i, where $1 \leq i \leq 7$. Specifically, the sequence is composed from three video objects (VO_1, VO_2, and VO_3) whose VOP computational requirements vary. It is assumed that the scheduler has already created the VOP list, and that all the VOPs that belong to the same frame have the same encoding deadlines. Initially, there are three processes that simultaneously run three instances of the MPEG-4 encoder, one for each video object. The calculated VOP size for each video object as a percent of the frame size is for VO_1 0.30%, for VO_2 0.30% and for VO_3 0.40%, while the number of threads that exploit the maximum system parallelism is 7, one thread per processor. Therefore, at this example for VOP_3 $0.4*7 \approx 3$ threads are created, that run on a group of three system processors (For instance P_5, P_6, and P_7). For VOP_1 and VOP_2 which have same sizes $0.3*7 \approx 2$ threads are created, that run on the other two groups of processors, group 1 (P_1, P_2), and group 2 (P_3, P_4).

An analytical description of the proposed scheduler and the way that it allocates the VOPs to the system processors is given in Chapter 4.

When the SPMD paradigm is applied at the consumer (processes) level it is more likely to create threads that require more processing than other threads that handle a different VOP area, since different macroblock types require different amounts of processing, This leads to processor assignments with unbalanced threads in terms of processing power. Therefore, the model incorporates a data partitioning solution that results in equal workload distribution among the processors by creating threads that encode equal number of macroblocks with similar processing requirements.

Measurements demonstrate [Kuhn 98] that shape encoding is the most computationally demanding part after motion estimation, where the consumer cannot apply SPMD parallelism, since the macroblock shape status, which indicates the amount of processing power that a macroblock needs, is dynamically derived for each macroblock when its shape is encoded. To reduce the shape encoding complexity a new solution is incorporated to the proposed parallel model that speeds up shape encoding, and exploits system parallelism by introducing a circular pipeline distribution algorithm, explained in Chapter 6.



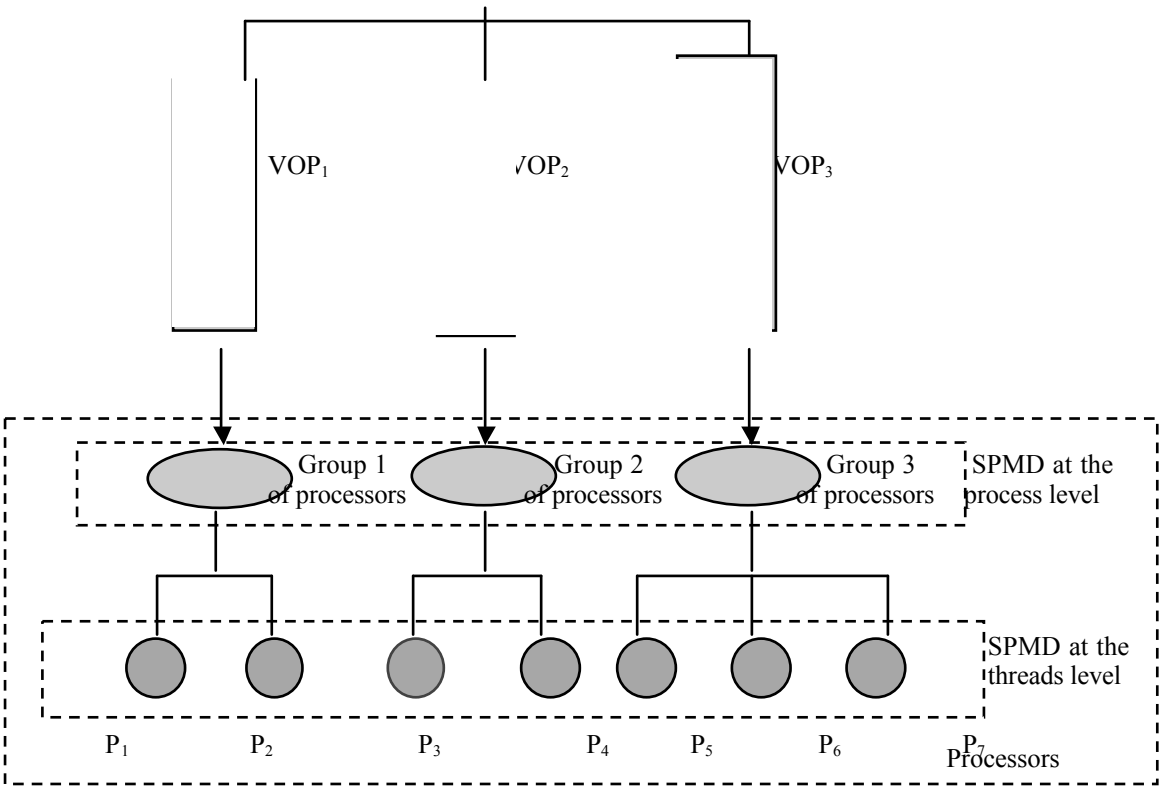


Figure 2.5: Proposed dynamic VOPs allocation mechanism on MPEG-4 parallel encoding

2.3 Implementation issues

The rest of this chapter addresses issues related to (i) the hardware platform that is used to demonstrate the efficiency of the proposed parallel model in MPEG-4 video encoding, and to the (ii) the software that is used.

2.3.1 Hardware architecture

The implementation and experimentation was done on an Origin200, which is a shared memory Silicon Graphics platform that consists of four processors R12000 with a 270 MHz clock speed.

2.3.2 Other MPEG-4 video encoder characteristics useful for the experiments

2.3.2.1 Profile

Profiles and levels in MPEG-4 [14469-2, 99] are standardised in order to give users a number of well-defined and well-chosen conformance points. These mainly serve two purposes: 1) ensuring interoperability between MPEG-4 implementations, and 2) allowing conformance to the standard to be tested. For this thesis the core profile, Table 2.1, is used for experiments.

Object Type Tools	Core
Basic Tools • I-VOP, P-VOP • AC/DC prediction • 4MV, Unrestricted MV	X
Error resilience • Slice Resynchronisation • Data Partitioning • Reversible VLC	X
B-VOP	X
Two quantisation methods	X
P-VOP based temporal scalability • Rectangular • Arbitrary shape	X
Binary shape	X

Table 2.1 MPEG-4 CORE PROFILE

2.3.2.2 Simulation software and sequences used on experiments

The final committee draft (FCD) MPEG-4 simulation software codec [Simulation 98] is the reference software, obtained by the MoMuSys project[3]. Most of the experiments described in this thesis make use of the two well-known MPEG-4 video sequences: “News” and “Coastguard”, figures 2.6 (a) and (b). These sequences are used widely from the researchers as benchmarks in order to evaluate the performance of different MPEG coding algorithms and tools. The News is a sequence of QCIF resolution (176*144 pixels), which is composed from four fixed size video objects (news0, news1, news2, news3) with different motion coding requirements (fast motion -news1, slow motion- news0, news2). The Coastguard sequence has CIF resolution, (352*288 pixels) and is composed of four video objects whose sizes vary over the sequence display time. Both sequences use binary segmentation masks for the shape information (alpha planes).

(a)

[3] European ACTS multimedia project for Mobile Multimedia Systems. This project has provided very significant input to the MPEG-4 standardisation process.

(b)

Figure 2.6: QCIF video sequence News, (b) CIF sequence Coastguard

Chapter 3

3 Fast Motion Estimation Block-based Algorithms

3.1 Overview

The MPEG-4 standard, like its predecessors MPEG-1 and MPEG-2, makes extensive use of the interframe block-based predictive coding. Interframe coding uses motion estimation (ME) which has been proven to be an effective way to exploit the temporal redundancy of video sequences. Motion estimation algorithms have attracted attention within the research community and industry due to the following facts

- they are the most computational intensive part of coding process (about 60-80% of the total computation time) which limits the overall performance/ speed of the encoder.
- they have a high degree of impact on the visual quality of final image for a given bit-rate
- the method to extract the motion vectors (MVs) from video material is not standardised, thus it is open to competitive approaches.

This chapter starts by analysing the motion estimation/compensation mechanism of MPEG-4 in order to identify where full search motion estimation is employed. Then it presents two new block based fast motion estimation algorithms that

proposed to replace the full motion estimation search algorithm of the MPEG-4 video encoder.

3.2 MPEG-4 motion estimation

3.2.1 Motion estimation of P-VOPs

The motion vectors of P-VOP macroblocks are derived as follows in the MPEG-4 standard:

Initially, both 8x8 and 16x16 vectors are obtained from the full search algorithm. The search is made with integer pixel displacement for the Y component of the YUV video sequence. The comparisons are made between the incoming block and the displaced block in the previous reconstructed VOP. For the 8x8 integer vectors only a small amount of additional computation is needed since the 8x8 search is centered around the 16x16 vector with a search window of ± 2 pixels. After integer pixel motion estimation the coder makes a decision on whether to use intra or inter prediction in the coding. If intra mode is chosen, no further operations are necessary for the motion search. Otherwise, if inter mode is chosen the motion search continues with half sample search around the vector position performed for both 16x16 and 8x8 vectors. Finally the coder makes a comparison between the sum of absolute differences (SAD) of the best half sample 16x16 vector and the SAD of the best half sample 8x8 vectors in order to adopt the most appropriate vector.

By measuring the complexity of the different steps that are used in P-VOP motion vector derivation, it was observed that the integer pixel search overrules and dominates the amount of time that is required for deriving P-VOP motion vectors- 90% of the computational complexity. Therefore, the full search algorithm that is used in integer pixel searching needs to be replaced by a fast search solution.

3.2.2 B-VOPs motion estimation

In MPEG-4 a B-VOP's macroblock is encoded in one of the following modes:

- Direct coding: Bi-directional motion compensation is used, derived by extending the H.263 approach of employing P-picture macroblock motion vectors and scaling them to derive forward and backward motion vectors for B-VOPs macroblocks.
- Forward & Backward Coding: Forward and backward motion compensation used in the same manner as in MPEG-1 and MPEG-2 with the difference that VOP is used for prediction instead of a picture.
- Bi-directional Coding: The bi-directional coding in MPEG-4 uses interpolated motion compensation in the same manner as in MPEG-1 and MPEG-2 except that VOP is used for prediction instead of a picture/frame.

The MPEG-4 encoder calculates the macroblock's SAD value using the above modes, where the mode that results in the smallest SAD is chosen.

Again, by measuring the complexity of the different modes that are used in B-VOP motion vector derivation, it was observed that the most computationally intensive modes are the forward and backward coding modes. Therefore for these modes we propose the replacement of the full search algorithm by a fast search solution proposed in section 3.4.

3.3 Motion estimation algorithms

The block matching technique has been widely used for motion vector estimation due to its simplicity. The straightforward approach is the full search algorithm that performs full search block matching by searching all locations in a given search area, and then selecting the position where the residual error is minimized. However, this procedure requires an extremely large amount of computation. Motion vector estimation is known to be one of the main bottlenecks in real-time encoding applications, and the search for an effective motion vector estimation algorithm has been a challenging problem for years. Over the past decades fast block matching algorithms have been developed to reduce the computational cost. Generally these can be categorised into the four groups detailed bellow:

A. Block Matching with Pixel Subsampling

One technique to reduce the complexity of motion vector estimation is block matching with pixel sub sampling [B.Liu 93]. Instead of limiting the number of search locations, the number of pixels used in matching error computation is reduced. This technique is called "alternative 4:1 pixel". Two other techniques were also presented in [B.Liu 93] to enhance performance: the sub-sampled motion field estimation which exploits the idea of block sub sampling, and the sub-block motion-field estimation which uses a smaller block size in motion estimation. For this category two fast algorithms are proposed, based on the combination of the first (alternating 4:1 pixel sub sampling) and the second (sub-sampled motion-field) techniques and the combination of the first and the third (sub-block motion field) techniques.

B. Block Matching with Spatial/Temporal Correlations

This group of algorithms exploits the information from adjacent blocks by using spatial and temporal correlations of motion vectors [Xie 92],[Zafar 91],[Zhang 91],[Chalidabhongse 97]. The main idea is to select a set of initial motion vector candidates from spatially and/or temporally neighbouring blocks and choose the

best one (according to a certain rule) as the initial estimate for further refinement. In theory the initial estimate can be obtained by using an auto regressive model [Zafar 91], [Zhang 91]. For such a simple model, only one candidate is chosen and used as the initial estimate in the experiments. The refinement process makes use of full search within a reduced search area, which still involves a considerable amount of computation. A hybrid algorithm that uses both block-recursive and block matching methods was proposed in [Xie 92]. Although its original motivation did not aim at the use of spatial and temporal correlations, it did provide an interesting way to use both correlations effectively. In this algorithm, the motion vector candidates are selected from two spatially and one temporally neighbouring blocks. In the refinement process, a recursive method is used to explore the gradient direction for motion vector updates. However, this gradient approach does not work well for sequences with fast motion since an oscillation in the search direction can happen in refinement. Finally, a hybrid algorithm that performs fast motion vector estimation using multi resolution-spatio-temporal correlation (MRST) is introduced in [Chalidabhongse 97]. The frame is decomposed into different resolutions (this group of algorithms will be discussed in the next paragraph). Motion estimation is first performed on the coarsest resolution, and motion vectors of finer resolutions are refined based on the motion information obtained by the spatio-temporal correlation and the previous level motion vector.

C. Hierarchical and Multiresolution Fast block Matching

This category of fast motion estimations algorithms initially predict an approximate large-scale motion vector and then refine the prediction in a multi resolution fashion to derive the motion vector of finer resolution. They are called multiresolution [J.Li 93], [Uz 91], [Zafar 93] or hierarchical algorithms [Bierling 88], [Dufaux 92]. The hierarchical algorithms use the same image size but different block sizes at each level. The assumption is that the motion vector obtained from a larger block size provides a good initial estimate for motion vectors associated with

smaller blocks which are contained in larger blocks. But this assumption is often not true and the estimation can be very poor. Furthermore larger block sizes imply higher computational costs for block matching.

The multiresolution algorithms [J.Li 93], [Uz 91], [Zafar 93] use different image resolutions with a smaller image size at a coarser level (pyramid form). They are divided into two groups: constant block size and variable block size. In [J.Li 93] and [Uz 91] the same block size is used at each level, where a block at the coarser level represents a larger area than that of a finer level, so that a smaller search area can be used at coarser levels. In [Zafar 93] different block sizes are employed at each level to maintain a one-to-one correspondence between blocks in different levels and the motion vector of each block can be used directly as an initial estimate for the corresponding block at the finer level. In [X.Lee 96] a fast hierarchical motion compensation scheme is reported, which uses, instead of the original block pixel values a sign truncated feature (STF). Methods in this category work relatively well, however for the motion vector refinement they use only the information from coarser levels in finer levels without considering other useful information such as temporal and /or spatial correlations among motion vectors of the same level. In addition, the refinement process is performed by using full search in a reduced search area, which, nevertheless, requires a considerable amount of computation.

D. Fast block Matching Algorithms that reduce the number of search locations

Another interesting category of algorithms that reduce the complexity of motion vector estimation is block matching with subsambling of the search range. With this technique the number of search points used in matching error computation is reduced. Several fast algorithms, such as the new three-step search [R.Li 94], the

one at a time search [Srinivasan 85], the cross-search [Ghanbari 90], the four-step search [Po 96], the BBGDS [L.Liu 96], and the diamond search [Zhu 97] are reported. Based on the property of center-biased motion vector distribution, these algorithms employ various checking point patterns that derive MV in a limited number of steps. In [Z.He 97] the search range is partitioned into nested zones. A search starts from the first zone and continues in the second, third and up to z_{max}th search zone, until a block is found to satisfy one of the following: 1) the matching error is less than a threshold value T 2) the overall best matching is found, or 3) the minimum point is in the centre (first zone search). Finally, a hybrid method that exploits the zone searching and the spatial macroblock relations is proposed in [Touparis 99].

3.4 Proposed Fast Motion Estimation Methods

As stated in Chapter 2, this thesis aims not only the identification of the most computational intensive parts of the motion estimation mechanism, but the development of motion estimation solutions that are simple, fast and suitable for the MPEG standard, yet work effectively in the sense of producing small residual errors. Moreover, taking into account the facts, that (i) hierarchical algorithms are not suitable at the MPEG coding, due to the extended use of different block sizes, and (ii) that the most frequently used algorithms in MPEG are based on techniques that explore correlations or search range sub-sampling, this thesis focuses on the development of motion estimation algorithms that exploit correlations, or range sub sampling. The rest of this chapter discusses two new algorithms, which exploit range sub-sampling/correlations, including overall performance comparisons with other well-known algorithms in this field.

3.4.1 Fast motion prediction scheme for bi directional coding.

3.4.1.1 Design Motivation

In predictive coding the size of the search area is proportional to the temporal distance of the current frame[4] from its reference. The insertion of frames increases the search area of the reference frame which in turn increases complexity. The proposed algorithm [Hamosfakidis WIAMIS 99] exploits the temporal correlation of motion fields between consecutive frames. The macroblock displacement is used for tracking the area of motion, and excluding search points that are not in the direction of macroblock displacement. The original search area is partitioned into two zones. The first zone (inner) is used to track the motion displacement, and the second zone (outer) is used to derive the motion vector through a refinement procedure, figure 3.1.

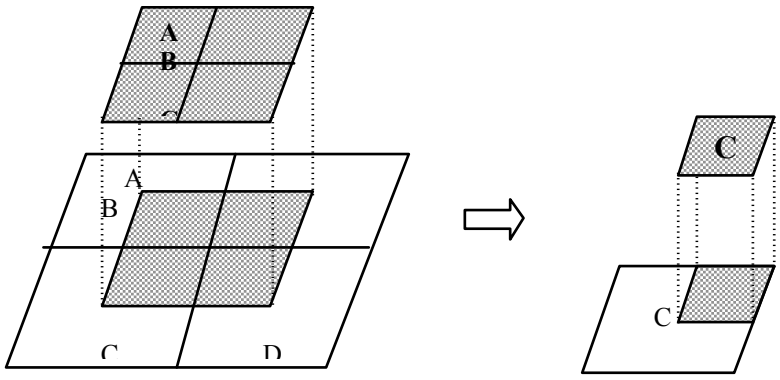


Figure 3.7: The inner zone is shaded grey. In Step1 track the motion in the inner zone, and in Step 2 limit the search. If motion is tracked at C area of inner zone, then the search in step 2 is limited to zone C in outer zone.

[4] The terms frame and VOP will be used interchangeably.

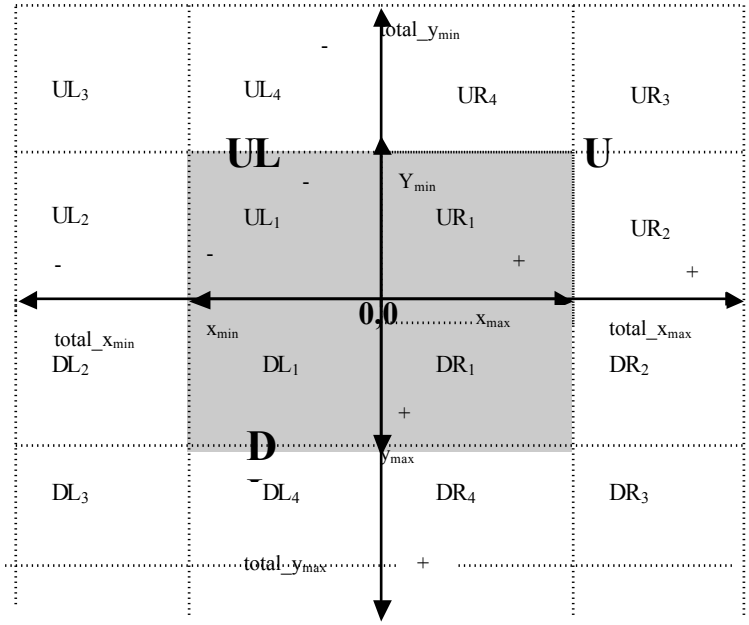


Figure 3.8: Search area-inner/outer zones and sub area partitions

3.4.1.2 Search area partition

The search area is divided into two nested zones (inner zone size is _ of original search area) for the following reasons:

- it suits many of the coding patterns that the encoder uses. Most commonly used patterns are the IPP…, or the IBPBP… [Wilkinson 96]. These patterns generate a worthwhile gain in terms of (a) efficient compression rates (bits per pixel and bit-rates), and (b) required computational complexity compared with other patterns using more than one B VOP between two reference VOPs
- the size of the inner area is either not too small that it could lead to a less effective prediction of the macroblock displacement, or too large that it could increase the computational complexity.

Each zone is divided into four sub regions: the upper left (UL), the upper right (UR), the down left (DL), and the downright (DR). UL and DR sub regions derive motion vectors with negative and positive co-ordinates (x,y) respectively. Similarly, UR and DL derive motion vectors with (positive x, negative y) co-

ordinates, and (negative x, positive y) co-ordinates respectively, figure 3.2. The sign of the motion is defined as follows: On the Y-axis macroblock downwards/upwards displacements are assigned positive/negative values. On the X-axis macroblock left/right displacements are assigned negative/positive values, figure 3.3.

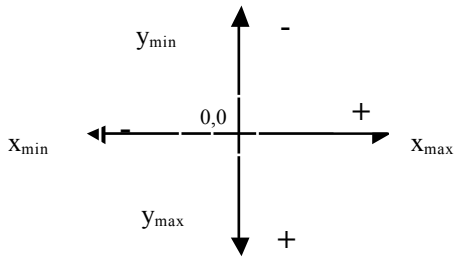


Figure 3.9: Sign of the macroblock displacement at the search area

3.4.1.3 Core of the fast motion prediction scheme

The motion vector is derived from a two step procedure. In the first step, a coarse motion vector is derived from the inner search zone using full search. Then at the second step the algorithm derives a refined motion vector using full search at the part of the outer zone that includes the inner zone where the motion was tracked. There are three cases for each motion vector derived in the first step. Each of these cases determines the area of the outer zone where the search will be continued from the scheme in the second step:

*Case 1:*There is no motion tracking. The derived motion vector (x,y) co-ordinates have zero values and therefore for this case the algorithm uses the following assumption: if no macroblock displacement is tracked in the inner zone then the probabilities of tracking a displacement in the outer zone are almost zero, and therefore there is no need to search in the outer zone in order to derive a motion

vector. The search is terminated and the derived motion vector is the one with zero co-ordinates.

Case 2: The algorithm tracks motion along the X or Y axis by searching the inner zone. A motion vector is derived, where one of its co-ordinates has no zero value. In this case, the search area of the outer zone is expanded up to its total size along the axis where motion occurred. The search is continued in the outer zone, and a new motion vector is searched by the algorithm (second step). The algorithm compares the SAD value that the new vector generates with the SAD value of the vector that has been derived from the first step. The new motion vector is chosen if it contributes to a smaller SAD.

Case 3: The algorithm locates the motion in one of the four sub regions (SR) of the inner zone, where SR is either the UL, or the UR, or the DL, or the DR region. Motion vector searching continues in the outer zone as follows: Based on the assumption that the motion is smooth and varies slowly, the initial sub- region (SR) of the outer zone where the search will be continued is the SR_2, see figure 3.4. There is more likelihood in this area of finding a refinement motion vector (Step A). If the motion vector that is found in Step A generates a smaller SAD value than the motion vector that was found in SR1 from the inner's zone search, then the search continues at the SR3 sub-region (Step C). If the motion vector that is found in SR_3 sub-region generates even smaller SAD than the motion vector found in SR_2 then the search is stopped and the SR_3 motion vector is used by the scheme. Otherwise, the motion vector derived from the SR_2 sub-region is used and the search is stopped.

On the other hand, if the motion vector that was found in Step A generates bigger SAD than the motion vector that was found in SR_1 (inner zone) the search continues in the SR_4 (Step C). If the derived motion vector in SR_4 sub-region leads

to a smaller SAD than the motion vector of the SR_1 sub-region, the search stops and the scheme uses the motion vector of the SR_4 sub- region. Otherwise, the motion vector that was derived from SR_1 is used and the search is stopped.

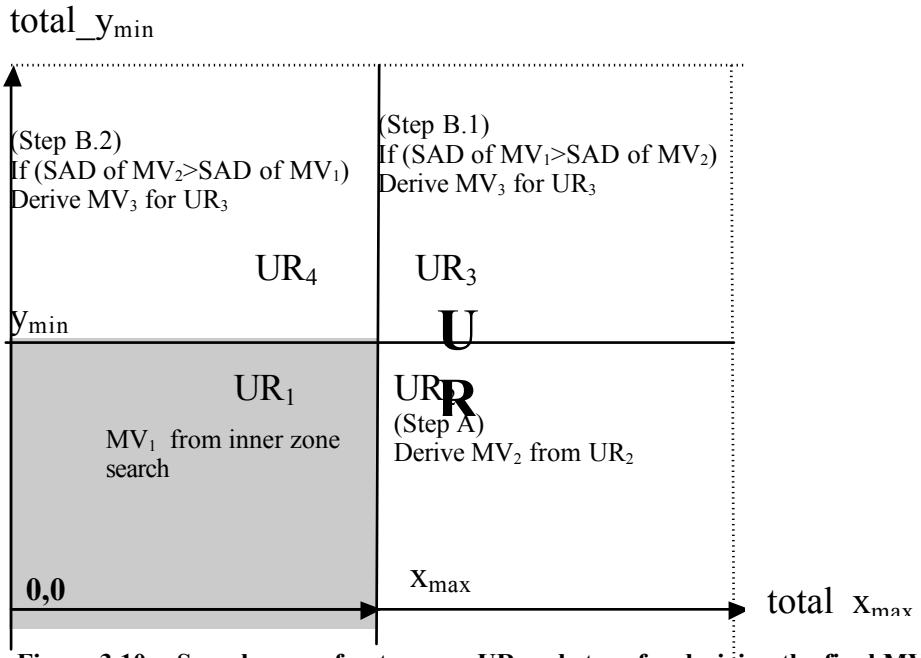


Figure 3.10: Search area of outer zone UR and steps for deriving the final MV

The proposed scheme as pseudo code is given in tables 3.1 and 3.2. Table 3.1 describes how the scheme makes decisions regarding the direction of the motion using the inner zone. Full search performed in the inner zone and a motion vector (MV) is derived. By examining the motion vector coordinates (x,y) the scheme determines the location of the motion vector in the inner zone and the search area of the outer zone where search will be continued. Table 3.2 describes how search is performed in the outer zone of the scheme's search area.

```
TWO_LEVEL_MOTION_PREDICTION_SCHEME () {
      Divide the search area into two zones;
       WHILE (macroblocks) {
            FOR each macroblock {
                  Search  inner zone and find MV that minimises the SAD;
                    IF (MV.x <0 && MV.y <0)
```

```
                    Search_Area (UL, SAD, MV);
                IF (MV.x <0 && MV.y >0)
                    Search_Area (DL, SAD, MV);
                IF (MV.x >0 && MV.y >0)
                    Search_Area (DR, SAD, MV);
                IF (MV.x >0 && MV.y <0)
                    Search_Area (UR, SAD, MV);
                IF (MV.x ==0 && MV.y ==0)
                    Return MV;
                IF (MV.x >0 && MV.y ==0)
                    Search_Area (R, SAD, MV);
                IF (MV.x ==0 && MV.y >0)
                    Search_Area (D, SAD, MV);
                IF (MV.x ==0 && MV.y <0)
                    Search_Area (U, SAD, MV);
                IF (MV.x <0 && MV.y ==0)
                    Search_Area (L, SAD, MV);
            } /* end FOR */
        } /* END WHILE */
}
```

Table 3.2 PSEUDOCODE OF THE PROPOSED MOTION PREDICTION SCHEME FOR TRACKING THE DIRECTION OF MOTION

```
Search_Area (direction, min_error, motion_vector) {
    Store_MV=motion_vector;  store_SAD=min_error;
    IF (direction==UL | | direction==DL | | direction==UR | | direction==DR) {
        Use expanded sub-area indicated by the direction;
        Divide sub area into four sub regions {1,2,3,4};
        Derive the MV for the 2nd sub region, which is next to 1st sub region;
        IF ( store_SAD > SAD) {
            Store_SAD=SAD;  store_MV=MV;
            Derive the MV for the 3rd sub region that lays directly above 2nd sub
region;
            IF (store_SAD > SAD)
                Return MV;
            ELSE
                Return store_MV;
        } ELSE {
            Derive the MV for the 4th sub region that lays directly above 1st sub
region;
            IF (store_SAD < SAD)
                Return store_MV;
            ELSE
                Return MV;
        }  /* end if store_SAD */
    } /* end if direction */
    IF (direction==U | | direction==D | | direction==R | | direction==L) {
```

```
                Expand sub area only at the indicated direction;
                Divide sub area into two sub regions {1,2};
                 Derive the MV for the  2nd sub region ;
                IF (store_SAD < SAD)
                                Return_store_MV;
                ELSE
                                Return MV;
    } /* end IF direction */
} /* end Search  Area */
```

Table 3.3 PSEUDOCODE FOR DERIVING THE MV IN THE OUTER ZONE

3.4.1.4 Complexity analysis

The evaluation of the computational complexity of the proposed motion estimation scheme will be based on the number of searching points that it uses to derive the motion vectors. If NxN (N^2) is the size of the search area then the full search algorithm computational complexity is N^2 - where the computational complexity is defined as the number of the searching points that the algorithm has to check. Because the proposed scheme employs a two-zone search its complexity is given as the sum of the complexities produced at each zone search, figure 3.4. Initially the scheme searches only the inner zone using the full search algorithm, where its size is by definition N/2xN/2 ($N^2/4$), and therefore the complexity is given as $N^2/4$. The calculation of the computational complexity at the next step is not so simple as before since it depends on the following scenarios/cases:

Case 1: No motion at all is tracked in the inner zone. According to the scheme the search is terminated at the first step, and therefore the total area used for searching is 1/4 of the original one. In this scenario the proposed scheme is four times faster than the full search algorithm.

Case 2: Motion tracked along the X or Y axis at the inner zone. The additional area that is used in the next search has size N/2xN/4 ($N^2/8$). Therefore the total search area is ($N^2/4$+ $N^2/8$) =3/8 N^2. The scheme in this case is 8/3 times faster than the full search algorithm.

Case 3: Motion tracked in one of the four sub regions (UL, DL, UR, DR) of the inner zone. According to the scheme an additional search has to be performed on two sub- regions (SR) of the outer zone where the size of each sub region is (N/4xN/4); thus the additional area that has to be searched has size $2x(N/4xN/4)=N^2/8$. The total used search area is $N^2/4$ (first step)+ $N^2/8$ (second step) $=3/8\ N^2$, and the scheme in this case is 8/3 times faster than the full search algorithm.

From the above analysis is clearly that the proposed scheme performs between 2.66 and 4 (best and worst case scenarios) times faster than the full search algorithm.

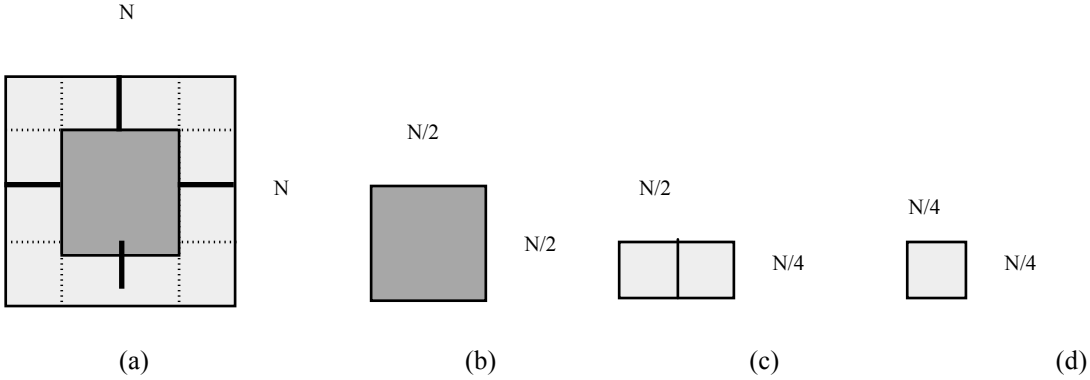


Figure 3.11: Size of search area: (a) original search area, (b) inner search area- Step 1, (c) additional search area for case 2, (d) the additional search areas for case 3

3.4.1.5 Experimental results

The overall performance of the proposed scheme is compared against the overall performance of the full search algorithm. As overall performance, this thesis considers measurements related to the computational complexity and to the image distortion. For the evaluation of the computational complexity the number of average search points used to derive the macroblock's motion vector is selected as measure, whereas for the image distortion-quality the peak signal to noise ratio (PSNR) is selected as a measure. PSNR is an image quality metric defined as

$PSNR=10log_{10}$ (MSE is the mean squared error), where larger values of it indicate higher quality.

For both schemes/algorithms the same set of video sequences is used with the following characteristics

- Slow motion video sequences- News 0. It is a background object from the News sequence with very slow motion
- Medium-High motion requirements video sequences- News 1, Rallycross. News 1 is a video object with high motion requirements in News sequence (dancer), and Rallycross is a video sequence with medium motion requirements.
- Intra coding period of ten VOPs for the IBP pattern, and intra coding period of nine VOPs for the IBBP pattern.

Tables 3.3 and 3.4 present the simulation results when the IBP pattern is chosen and tables 3.5 and 3.6 the simulation results when the IBBP pattern is chosen, since these patterns are the most often ones in predictive coding with B VOPs. In terms of results related to image quality, the proposed scheme performs similarly to the full search algorithm for both patterns and for every sequence. The PSNR of the proposed scheme is identical with the PSNR of the full search algorithm.

For experimental results related to computational complexity we can observe the following:

1. As expected for sequences with slow motion requirements, the proposed scheme reaches its best performance, which is 4 times faster than the full search algorithm. For News 0, where most of the search occurred within the

inner zone, the complexity of the proposed scheme is reduced by a factor of 3.42 and by a factor of 3.48 for the IBP and IBBP patterns

2. The average speed up in the computations of the proposed scheme is 3.12 times for the IBP pattern and 2.87 for the IBBP pattern.

Algorithms	News 0			News 1		
	Av. SP per MV	Complexity	Average PSNR	Av. SP per MV	Complexity	Average PSNR
FS	225	100%	36.08	225	100%	32.03
Proposed Scheme	65.8	29.23%	36.08	86.8	38.57%	32.03

Table 3.4 AVERAGE SEARCH POINTS PER MOTION VECTOR ESTIMATION AND PSNR FOR THE FIRST 100 FRAMES OF NEWS 0 AND NEWS 1-IBP PATTERN

Algorithms	Rallycross			Average Complexity for the Three video sequences
	Av. SP per MV	Complexity	Average PSNR	
FS	225	100%	34.24	100%
Proposed Scheme	63.55	28.24%	34.24	32.01%

Table 3.5 AVERAGE SEARCH POINTS PER MOTION VECTOR ESTIMATION AND PSNR FOR THE FIRST 100 FRAMES OF RALLYCROSS- AVERAGE COMPLEXITY FOR THE THREE VIDEO SEQUENCES

Algorithms	News 0			News 1		
	Av. SP per MV	Complexity	Average PSNR	Av. SP per MV	Complexity	Average PSNR
FS	225	100%	36.18	225	100%	32.04
Proposed Scheme	64.81	28.81%	36.18	87.89	39.06%	32.04

Table 3.6 AVERAGE SEARCH POINTS PER MOTION VECTOR ESTIMATION AND PSNR FOR THE FIRST 100 FRAMES OF NEWS 0 AND NEWS 1-IBBP PATTERN

Algorithms	Rallycross			Average Complexity for the Three video sequences
	Av. SP per MV	Complexity	Average PSNR	
FS	225	100%	34.27	100%
Proposed Scheme	82.11	36.49%	34.27	34.78%

Table 3.7 AVERAGE SEARCH POINTS PER MOTION VECTOR ESTIMATION AND PSNR FOR THE FIRST 100 FRAMES OF RALLYCROSS- AVERAGE COMPLEXITY FOR THE THREE VIDEO SEQUENCES

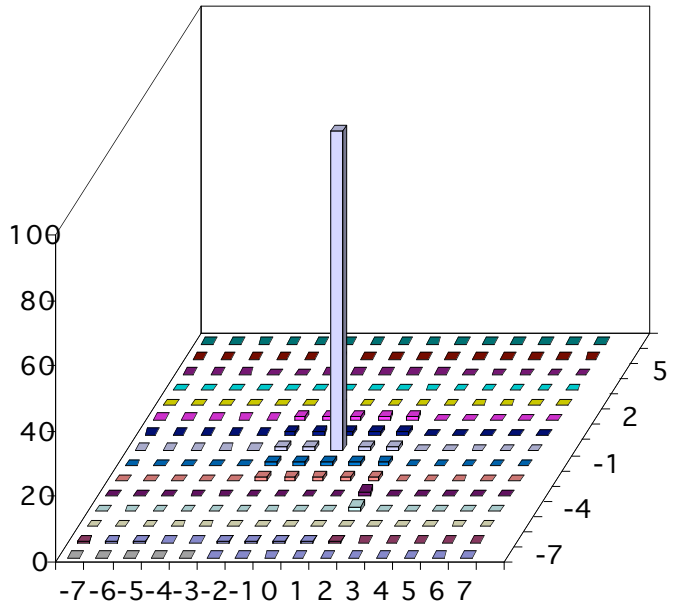
Overall, the proposed scheme performs better than the full search algorithm by a factor of three. This number can be improved further, since the proposed scheme does not deal directly with the searching technique that is used in the search area for the motion vector derivation. The most suitable candidates for the scheme are the block motion estimation algorithms that involve either pixel sub-sampling, or sub-sampled motion-field estimation (block sub-sampling), or sub block motion-field estimation (a smaller block size is used).

3.4.2 A Hexagonal (HS) algorithm for fast block matching motion estimation

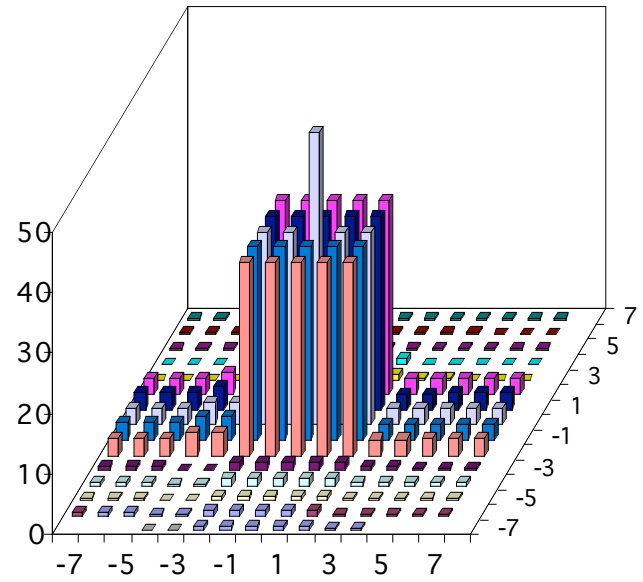
3.4.2.1 Design Motivation

Experimental results [Musmann 85] have shown that the block motion field of real world video sequences is usually smooth, and varies slowly. This leads to a centre-biased global minimum motion vector distribution instead of a uniform distribution. This can be observed based on the full search algorithm for the well-known News, and the Rallycross video sequences, figure 3.6. For the News 0 (background object) sequence, nearly all the blocks (97.55%) can be considered stationary, figure 3.6 (a). For the News 1 (dancer) sequence of faster motion and camera zooming, the motion vector distribution is still highly central-biased: 48.31% of motion vectors found at the centre of the search area, and 80% of these are enclosed in a central 5x5

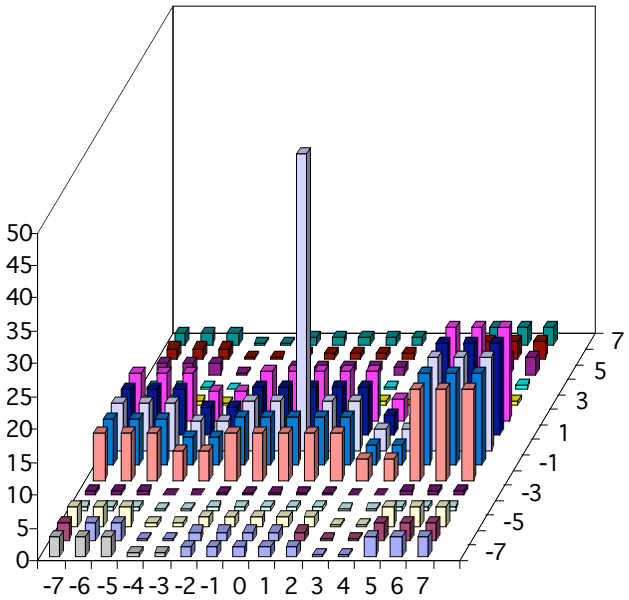
area, figure 3.6 (b). For the non segmented medium motion sequence Rallycross 52.76 % of its motion vectors are enclosed in a central 5x5 area, and 61.16% are located in a central search 9x9 area, figure 3.6 (c).



(a)



(b)



(c)

Figure 3.12: Motion vector distribution for (a) News 0, (b) News 1, and (c) Rallycross sequences. X and Y axis specify the range of the search area while Z axis gives the probability in percentage of the motion vector distribution within the search area.

Based on the characteristic of central-biased motion vector distribution fast block matching algorithms have been developed, as mentioned earlier, that reduce the number of search locations by searching only the area in which the distribution is more likely to occur. A large search pattern exploited by the TSS [Koga 81] with size 9x9 and nine checking points is quite likely to mislead the search path in the wrong direction and hence miss the optimal point. A centre-biased TSS algorithm, called NTSS [R.Li 94] tends to achieve better performance because it has a higher probability of finding the global optimum point. The average number of search points in NTSS is less than that in TSS but NTSS loses the simplicity of the TSS, to some extent. Using a moderate search pattern with a fixed size of 5x5 searching points, the 4SS [Po 96] obtains a performance that is similar to NTSS. However, 4SS still requires testing of 17 checking points for a stationery block. The DS [58] employs two diamond search patterns of sizes 9x9 and 3x3 respectively, is faster than the NTSS and the 4SSS, but does not cover edge points of the search area. From the above discussion it is clear that the shape and the size of the search patterns jointly determine not only the image quality (PSNR) but also the computational complexity of this category of fast block matching motion estimation algorithms. The proposed hexagonal (HS) algorithm employs a new search pattern that reduces further the computational complexity without causing distortion to the image quality.

3.4.2.2 Hexagonal Search (HS) algorithm

Since motion vectors are not evenly distributed in the search area in fact most of them are located inside a centre-biased window of size 9x9, the HS patterns are designed to take into account the following:

(i) *reduced computational complexity*: the point where the minimum block distortion (MBD) occurs should be tracked using a small number of

checking points, which cover a significant portion of the centre-biased search window, and

(ii) *search patterns shape:* when the MBD point is located, the search pattern has to be shaped in such a way that allows a refined search which covers all searching points around the MBD point in order to derive the MBD point for the best matching block.

As shown in figure 3.7, the HS algorithm utilises a centre-biased search pattern of seven checking points, out of which six points surround the centre one to compose a hexagon (Step 1). The hexagon points are checked and the centre of the hexagonal search window is then shifted to the point with minimum block distortion. The search pattern and its size, for the next two steps of the HS, depend on the location of the MBD points. If the MBD point is found at the center of the hexagonal pattern, the search proceeds to the final step (Step 3), with a smaller search pattern for a refinement search. Otherwise, the hexagonal search pattern is applied repeatedly until the MBD point is found at the center of the hexagon (Step 2). When the final step (Step 3) is reached, the search pattern is changed from hexagonal to a star, figure 3.7 (b) with a variable number of search points, best case 4 and worst case 6. For edge points of the search area the hexagonal search pattern (step 2) is modified, figure 3.8. The HS algorithm is summarised as follows:

Step 1: The initial hexagonal pattern is centered at the origin of the search window and the seven checking points (_) of the hexagon are tested, figure 3.7 (a). If the MBD point is found at the center position then go to Step 3, otherwise go to Step 2.

Step 2: The MBD point found in the previous search step is re-positioned as the center point to form a new hexagon. If the new MBD point obtained is located at the centre position, go to Step 3; otherwise, recursively repeat Step 2. The

hexagonal pattern is modified on the borders of the search area in order to cover the edge points. Figure 3.8 (a) presents all the possible shapes of the hexagonal pattern when it reaches the left/right or the up/down borders of the search area. More precisely, there are two different scenarios when the pattern reaches the top or down borders. The first scenario is the centre point of the pattern, when it is shifted towards the up or down borders, to be on the border. In this case the new hexagonal pattern employs 4 checking points. The other scenario is when the checking points of the shifted hexagonal pattern are out of the borders of the search area, in this case the shifted pattern has 6 checking points. Similarly there are two cases when the shifted hexagonal pattern reaches the right or left borders of the search area. One case is when the centre of the shifted pattern is on the border of the search area, and the other case is where checking points of the shifted pattern lie outside the borders. In both cases the modified pattern has 5 checking points.

Step 3: Switch the search pattern from hexagon to star (_), figure 3.7 (a).

There are two different star patterns that are employed for different locations of the MBD point in the search area

- The MBD point is not an edge point. The initial star pattern, figure 3.7 (a) is centered to the MBD point of the hexagonal pattern and its four checking points are tested. If the new MBD point calculated for the star pattern is located at the centre then this point is the final solution for the motion vector and the search stops. Otherwise, if the new MBD point is one of the other points of the initial star then its neighbouring points, excluding the central star point, are checked figure 3.6 (b). The new derived MBD point is the final solution of the MV since it generates the smallest MBD in the pattern.
- The MBD point is an edge point. The initial star pattern, adjusted to three checking points, is centered on the MBD of the hexagonal pattern, and its three checking points are tested, figure 3.8 (b). If the MBD point is at the centre of the modified star the search stops, otherwise the neighbouring points of the

MBD, excluding the central point, are examined. The new derived MBD point is the final solution of the motion vector that points to the best matching block.

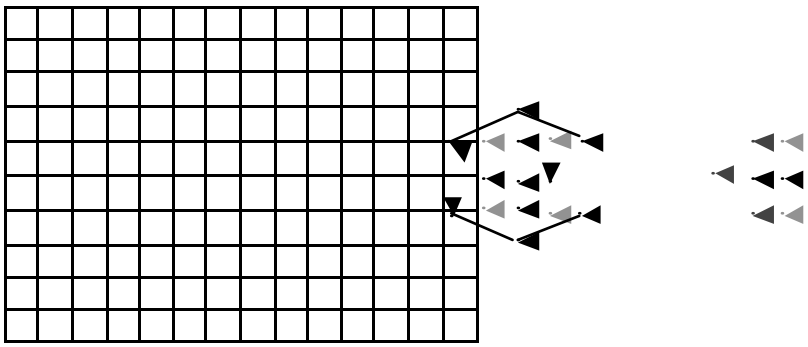


Figure 3.13: (a) Hexagon and initial star patterns, neighbouring points of the initial star are shown by grey colour, (b) Expanded star pattern for no edge points.

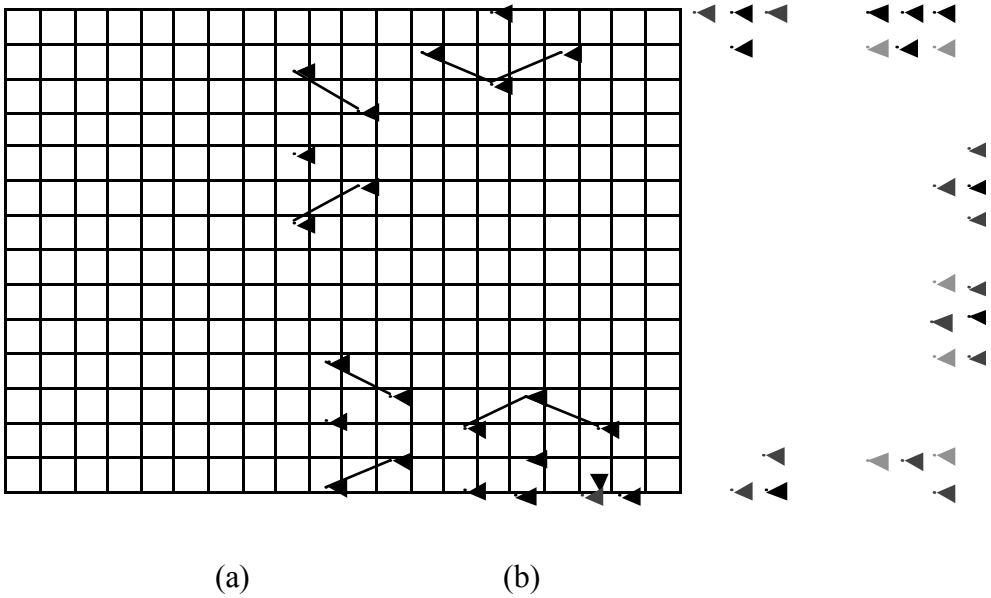


Figure 3.14: (a) all possible shapes of the hexagonal pattern when it reaches the left/right or the up/down limits of the search area, and (b) possible star shapes (initial and expanded) for a down/up or right edge MBD points.

Note that the checking points of the hexagon search pattern are partially overlapping when Step 2 is repeated. Only three checking points need to be calculated in the new pattern. In addition, at Step 3 when the search pattern changes from hexagon to star, three, four, or six points of the star need to be calculated, depended on star's MBD point location. An example of a possible search path using the proposed HS within a search 15x15 window is illustrated in figure 3.9, to demonstrate the checking points overlapping along the search path, MV (3, -4). Figure 3.10 also presents an example of how the HS derives a motion vector from the borders of the search area, MV (-7,7).

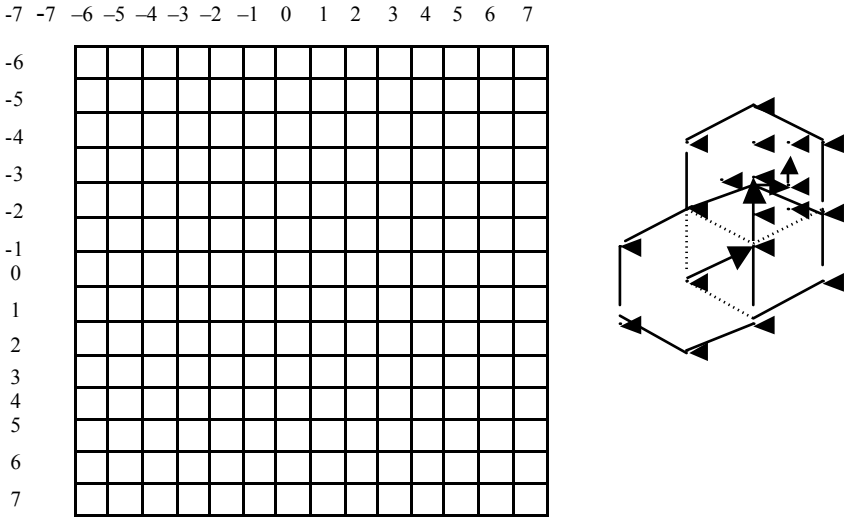
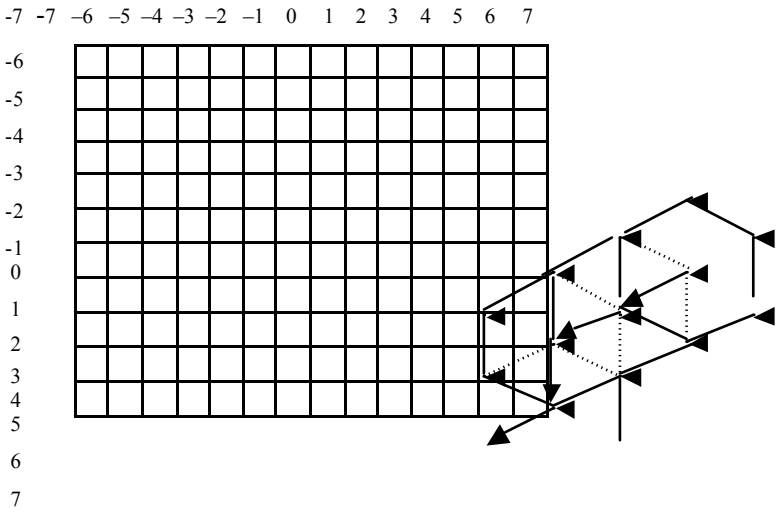


Figure 3.15: HS search path for MV (3, -4)



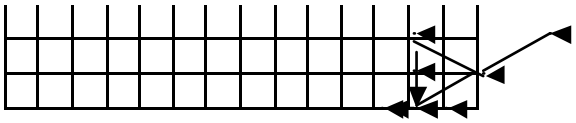


Figure 3.16: HS search path for MV (-7,7)

3.4.2.3 Experimental results

The overall performance of the Hexagonal search (HS) algorithm is compared against the overall performance of the full search (FS) algorithm and three other well-known algorithms from the category of search range sub sampling. These are: the new three step (NTSS), the diamond search (DS), and the four step search (4SS) algorithms. Similar to the case of the fast prediction scheme for the bi directional coding, the overall performance takes into account the average search points per MV of the algorithm, and the generated PSNR. The same set of video sequences is used (News 0, News 1, and Rallycross). The block size fixed at 16x16 pixels, and the maximum motion in row and column is assumed to be ±7 for simplicity. The first 100 frames of the News and Rallycross video sequences are used. The simulation results are shown in Table 3.7 and 3.8 and figure 3.11.

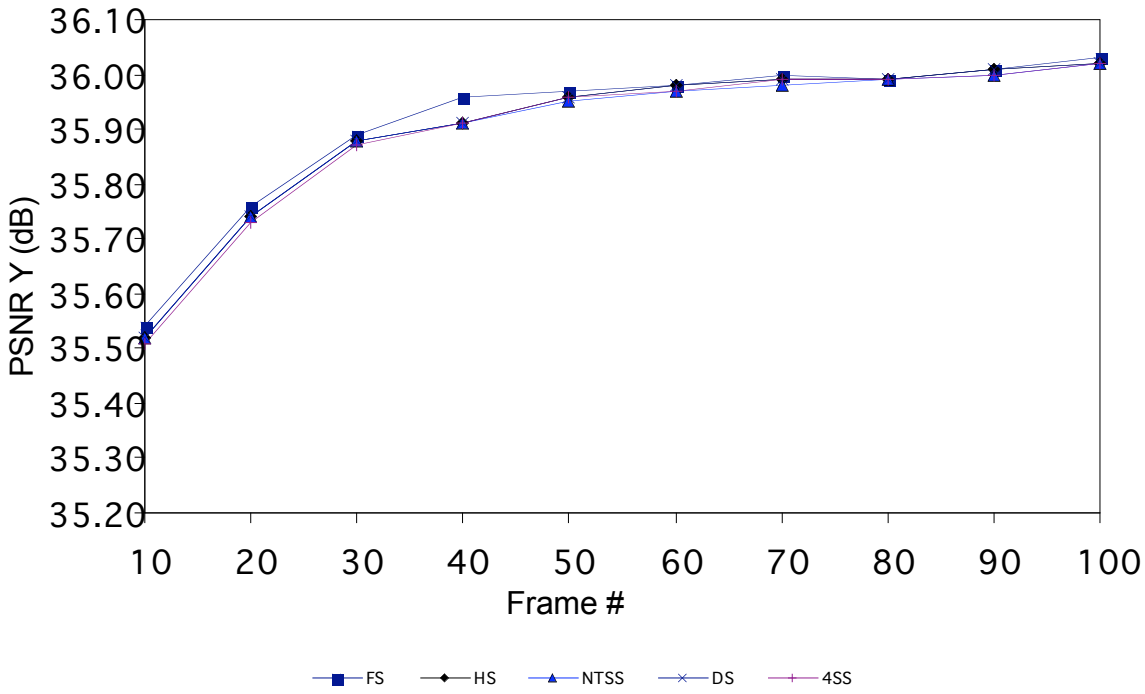
Algorithms	News 0			News 1		
	Av. SP per MV	Complexity	Average PSNR	Av. SP per MV	Complexity	Average PSNR
FS	225	100%	35.91	225	100%	32.04
4SS	17.03	7.57%	35.89	18.37	8.16%	32.01
NTSS	17.05	7.58%	35.89	18.96	8.42%	32.00
DS	13.05	6%	35.9	15.25	6.77%	32.00
HS	11.06	4.91%	35.9	12.67	5.63%	32.02

Table 3.8 AVERAGE SEARCH POINTS PER MOTION VECTOR ESTIMATION AND PSNR FOR THE FIRST 100 FRAMES OF NEWS 0 AND NEWS 1

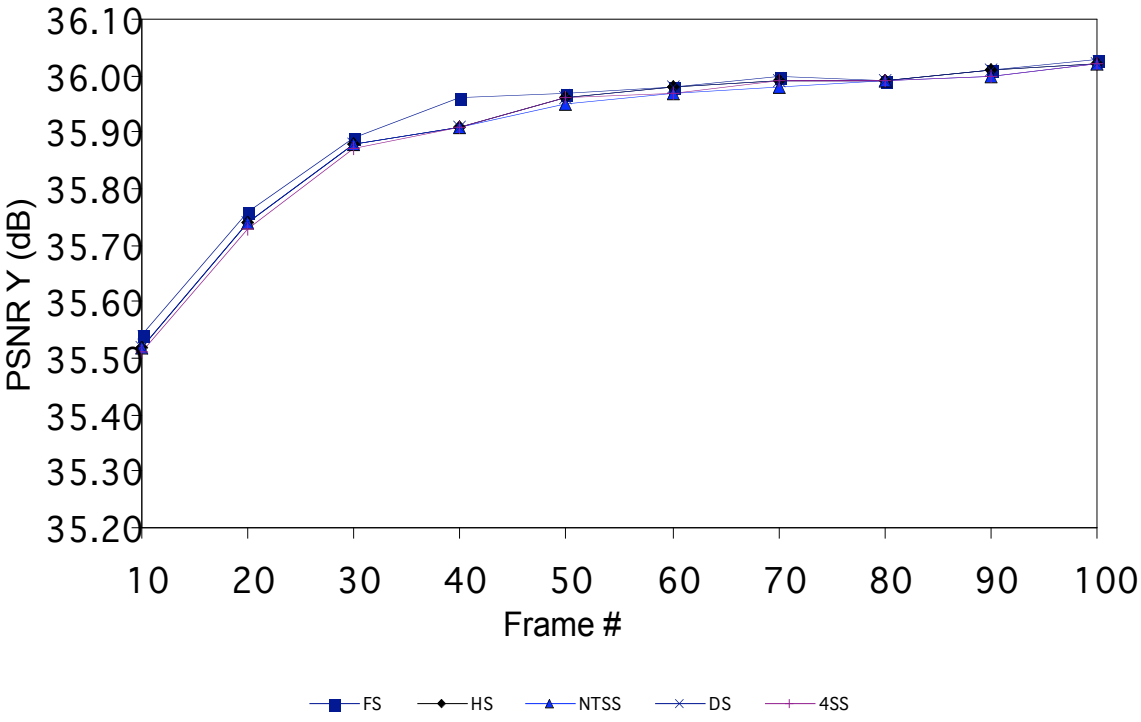
Algorithms	Rallycross	Average

	Av. SP per MV	Complexity	Average PSNR	
FS	225	100%	34.15	100%
4SS	20.67	9.18%	34.13	8.30%
NTSS	21.01	9.34%	34.12	8.45%
DS	17.99	7.99%	34.14	6.92%
HS	14.44	6.41%	34.14	5.65%

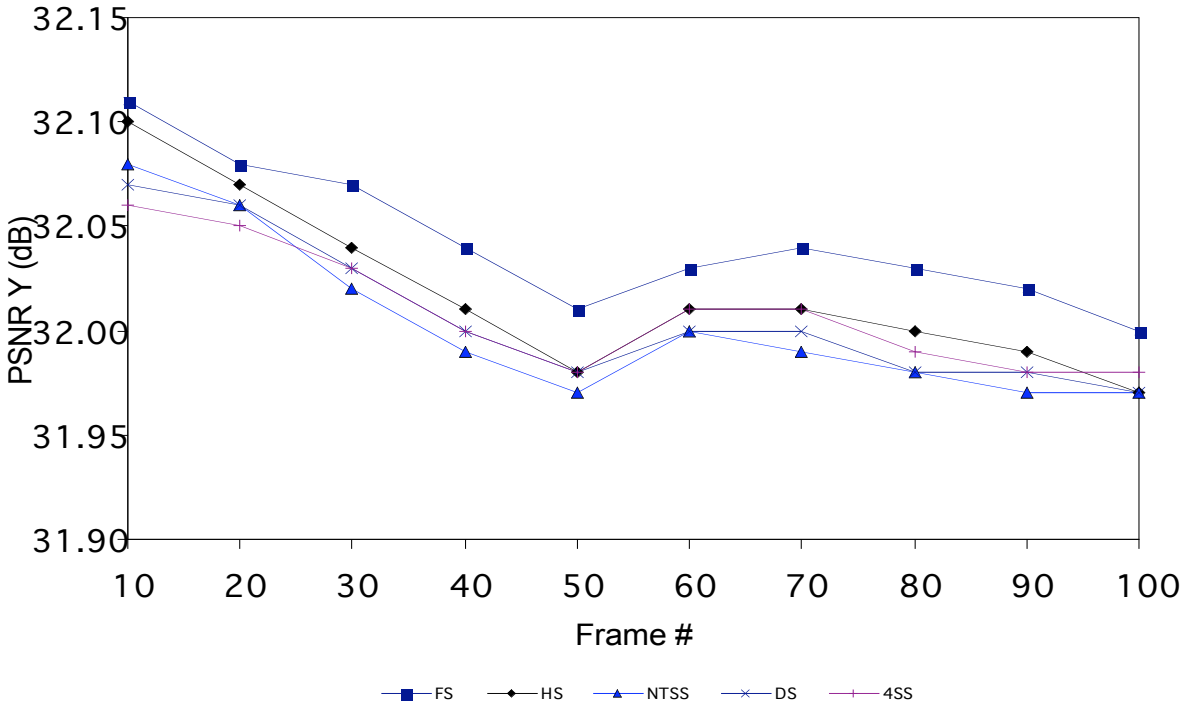
Table 3.9 AVERAGE SEARCH POINTS PER MOTION VECTOR ESTIMATION AND PSNR FOR THE FIRST 100 FRAMES OF RALLYCROSS- AVERAGE COMPLEXITY FOR THE THREE VIDEO SEQUENCES



(a) Rallycross


36.10
36.00
35.90
35.80
35.70
35.60
35.50
35.40
35.30
35.20
PSNR Y (dB)
10
20
30
40
50
60
70
80
90
100
Frame #
FS
HS
NTSS
DS
4SS

(b) News 0



(c) News 1

Figure 3.17: PSNR comparisons of HS, DS, NTSS, 4SS, and FS for (a) "Rallycross, (b) News 0, and (c) News 1.

Examing the results that are related to the image quality, the proposed scheme performs very close to the PSNR performance of the full search algorithm, and generates better PSNR results, for every sequence, than the other fast search algorithms. In particular for the News 1 sequence, which has fast motion requirements, the HS performs better than the other fast search algorithm, figure 3.11 (c). In terms of the performance related to computational complexity we observe the following

1. The simulation shows that the average HS computational complexity is less then the average computational complexity of all the other fast motion

algorithms. Specifically it is faster than the NTSS by 49.5%, faster than the DS by 22.47 %, and faster than the 4SS by 47%.

2. For sequences with medium to large motion vector distribution, HS outperforms all the other fast search algorithms. For instance for the Rallycross video trailer the HS computational complexity is 6.41% while DS, 4SS, NTSS complexities are 7.99%, 9.18%, and 9.34% respectively with similar performance to FS in terms of PSNR, figures 3.11, 3.12 and tables VI, VII.

(a)

(b)

(c)

(d) (e)

Figure 3.18: The 55th estimated frame for the Rallycross sequence using different searching algorithms. Estimated frames using (a) FS, (b) DS, (c) 4SS, (d) NTSS, and (e) HS.

3.5 Conclusions

Two block motion estimation algorithms were discussed in this chapter. Both of them are fast and suitable for the MPEG standard yet work effectively in the sense of producing small residual errors.

The proposed fast motion prediction scheme for bi directional coding exploits the temporal correlation of motion fields between consecutive frames. The macroblock displacement is used to track the area of motion and to exclude the search points that are not in the direction of the macroblock's displacement. Experimental results using with the most common used B coding patterns showed that the proposed scheme derives the motion vectors in average case three times faster than the full search algorithm. The speed up can be improved further, by involving in the scheme search process either pixel sub-sampling, or sub-sampled motion-field estimation, or sub- block motion-field estimation.

Based on the real world image sequence's characteristics of center-biased motion

vector distribution, a new Hexagonal Search (HS) algorithm with a center-biased pattern is proposed. Since the shape and the size of the pattern determine not only the image quality but also the number of operations that are performed for the MV derivation, the HS employs patterns that (a) use less checking points to derive the MVs compared with other well known algorithms, and (b) cover all checking points of the search area. Experimental results performed on sequences with different motion requirements have demonstrated that the HS uses in average case 12.9 search points to derive a motion vector while the DS uses 15.43 search points to derive a motion vector, the NTSS 19 search points, the 4SS 18.69, and the full search 225 search points. Moreover, the HS performs closer to the PSNR performance of the FS algorithm compared with the other algorithms that are used in our experiments.

Chapter 4

4 A scheduler for real-time MPEG-4 video encoding

It was mentioned in the introduction that the selection, and distribution of the MPEG-4 video objects onto a parallel architecture for encoding is a challenging problem. This chapter proposes and describes a heuristic scheduler suitable for MPEG-4 real time encoding on shared memory platforms, when one or more video objects exist in a scene.

4.1 MPEG-4 scheduling and heuristic schedulers

In Chapter 2 was shown that the proposed producer-consumer model for the parallel processing of the MPEG-4 video coding standard requires a scheduler which selects VOPs according to their calculated deadlines. But in MPEG-4 video encoding VOP deadline calculation is not a simple task, since a frame may comprise of more than one video object, each with different coding patterns, frame rates, and more likely, VOPs that appear and disappear over the encoding session. Therefore, the proposed scheduler should take into account in the VOP's deadline calculation formula, not only communication costs among the system processors, and the VOPs estimated encoding times, but also (from the MPEG-4 configuration files) the number of video objects that exist in the encoding session, with their starting and ending display times, their frame rates and the precedence constraints imposed

by their coding patterns. Additionally, as VOPs vary in size, the scheduler needs to adopt a dynamic VOP allocation policy to the system processors since a static VOP allocation would lead to an unbalanced system.

The problem of scheduling has received a lot of attention in the literature, where the design of scheduling heuristics depends both on the tasks characteristics, and on the target platform [Sarkar 89]. Since the platforms are either shared memory (centralised) or distributed, and the tasks are either independent or dependent with real or non real time requirements the scheduling heuristics are generally placed into the categories, shown in figure 4.1.

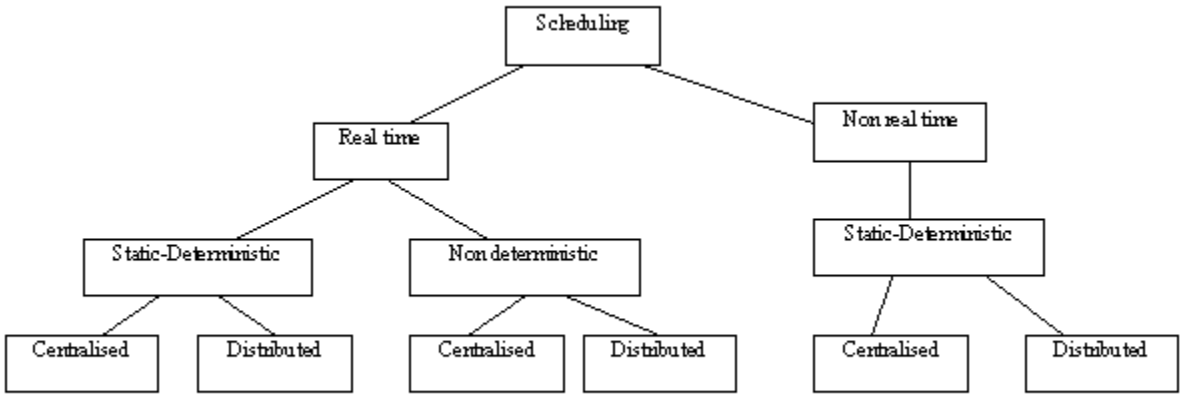


Figure 4.19: Taxonomy of scheduling heuristics

In real time scheduling, tasks have to be performed not only correctly but also within a defined time (or deadlines). Typically, real time scheduling decisions are based on task characteristics such as timing, precedence constraints and resource requirements [Andre 91]. On the other hand, no time constraints are considered for tasks scheduled by non real-time schedulers, where the main objective is the minimisation of the total execution time of the parallel program. Moreover

- Non real- time static scheduling heuristics require previous knowledge of

the information in the task graph. They are classified as centralised [Adam 74] or distributed. Distributed heuristics apply to Multiple Instructions Multiple Data (MIMD) architectures and take into consideration the interprocessor communication costs. They use either list scheduling [Wu 90][Hwang 89][Sih 93][Kwok 96], or clustering [Kim 88][Gerasoulis 92][Sarkar 89][Palis 96]. Centralised heuristics apply to shared memory architectures where communication costs are low. Their objective is either the equalisation of the load among the system processors, or the minimisation of the schedule length. The schedule length is defined to be the longest path in a task graph from the source node to its terminal node including all communications and computation costs. For homogeneous processor platforms the minimisation of the schedule length is actually the minimisation of the communication costs among the processors.

- In real time scheduling there are two categories of heuristics: dynamic and static. Real time dynamic scheduling heuristics, both for distributed and centralised systems [Ramamritham 90][Hamidzadeh 98][Manimaran 98][Zhao 87][Ramamritham 95][Hou 97] [Chetto 89] [Lehoczky 89], produce feasible schedules that take into account deadlines and resource constraints. They are suited for independent non- periodic tasks that arrive sporadically. The real time static heuristics produce optimal schedules for real-time tasks when their characteristics are known a-priori. There are heuristics for periodic tasks and non-periodic tasks that take into account parameters such as communication costs, deadline constraints and resource requirements.

From the above discussion it can be concluded that none of the existing scheduling heuristics can be employed directly to the MPEG-4 video object scheduling [Hamosfakidis IEEE MS 98] [Hamosfakidis IEEE MS 99] [Y. He 98] [Y. He 97].

Since this thesis deals with the parallel MPEG-4 video encoding onto shared memory platforms, none of the distributed scheduling heuristics from the above taxonomy are suitable. These schedulers analyse the problem of optimally assigning the modules of a program over the processors. Their objective is to assign modules, wherever possible, to the processors on which they execute most rapidly while taking into account the overhead of interprocessor communication. While an MPEG-4 scheduler has to select the most appropriate VOP for encoding using its calculated deadline, and then to distribute it on a group of processors where its size is related to VOP size.

The non real time scheduling heuristics cannot be applied to real time MPEG-4 scheduling, even if they select the VOPs according to their precedence constraints, since they omit the aspect of VOP deadline calculation.

Whilst real time dynamic scheduling heuristics schedule tasks according to their time constraints they are not applicable in the case of the MPEG-4 scheduling, since they do not consider precedence constraints that are imposed by the video objects coding patterns; most of them deal with independent non-periodic tasks that arrive sporadically. In addition, the heuristics of this category are used more in order to produce an answer in polynomial time about the feasibility of tasks to meet their real time requirements when they arrive in the system.

4.2 Proposed real time MPEG-4 heuristic scheduler

4.2.1 VOP selection mechanism

One interesting category from the taxonomy in the previous section heuristic is List Scheduling. List schedules [Manacher 67] are a class of implementable schedules in

which tasks are assigned priorities and placed in a list ordered by decreasing magnitude of priority. Whenever executable tasks contend for processors, the selection of the tasks to be immediately processed is done on the basis of priority, with the higher priority tasks being the first ones assigned to processors. This section proposes a scheduler whose selection mechanism is based on the core of list scheduling. VOP deadlines are calculated for all video objects of the sequence. The VOP with the earliest deadline is assigned the highest priority. By using this information a list is created that contains all the VOPs of the video sequence sorted in an ascending order of their deadlines.

Before the VOP deadline calculation, the scheduler needs to know the order in which the VOPs are encoded. Therefore for each video object, the scheduler creates its VOP coding sequence using the information of the video object's coding pattern. More specific, the VOP coding sequence contains the VOPs of the coding pattern ordered according to their actual encoding requirements. B VOPs are always placed after their references in the VOP coding sequence. For instance, if a video object has the $I_1B_2P_3B_4P_5B_6P_7$ coding pattern its VOP coding sequence is the $I_1P_3B_2P_5B_4P_7B_6$.

Next, for each VOP of the VOP coding sequence, the scheduler calculates its deadline using the video object frame rate (FR), and its starting display time (ST). A VOP with real time requirements must complete its encoding before the arrival of the next VOP, where the time period between two arrivals is given as 1/FR. For instance, the time period between two VOP arrivals for a video object with FR of 25 frames per second is 1/25 = 40 milliseconds. Therefore, the formula that the proposed scheduler uses to calculate the VOP deadlines on a shared memory platform, where no communication costs exist among the processors, is the following one

Dij is the calculated deadline of the j'th VOP in the ordered VOP coding sequence

of the i'th video object of the video sequence that has FRi frame rate, and its first VOP is displayed for first time in the sequence at STi.

When the scheduler completes the VOP deadline calculation for each VOP in the sequence, it starts placing VOPs into the list, ordered in decreasing magnitude of priority, by employing the earliest deadline first (EDF) [Chetto 89] algorithm. While VOPs exist, the scheduler searches all the VOP coding sequences to find the VOP with the next earliest deadline that needs to be placed in the priority list. The VOP with the earliest deadline is assigned the highest priority. In case there are VOP deadline conflicts, the scheduler resolves them by introducing a priority rule that assigns priorities based on VOP coding types. Since the I-VOPs are always reference VOPs, they are assigned the highest priority, B-VOPs have the smallest priority, while P-VOPs have higher priority than B-VOPs and less than I-VOPs. Thus, if an I and a B VOP have the same deadline the scheduler puts the I VOP and then the B VOP in the list. Table 4.1 presents the pseudo code for the VOPs calculation deadline

---

Variables
num_vos: denotes the number of video objects existing in video session;

Temporal_VOPList = NULL; /* For each video object a list is constructed based on video object's VOPs deadlines and video object's coding pattern constraints*/

VOPList = NULL; /* Final VOP list that contains in an ascending order the deadlines of all VOPs of all video objects that exist in the video session */

Assigned Priorities for VOP coding types:
I_VOP_priority=1 –highest priority,
P_VOP_priority=2;
B_VOP_priority=3-lowest priority;

```
VOPListConstructor () {
    Continue=1;
    FOR each video object construct its Temporal_VOPList;
    WHILE (Continue) {
        Find the VOP with the earliest deadline among the Temporal_VOPLists;
        IF more than one VOPs have identical deadlines check their coding types;
            IF both VOPs have same coding types
```

```
                    Put them in the VOPList;
                ELSE
                    Put first in the VOPList the VOP with the highest priority;
            ELSE
                    Put VOP in the VOPList;

            IF all Temporal_VOPLists empty;
                Continue=0;
        } /* end WHILE */

} /* end of VOP LIST CONSTRUCTOR*/
```

Table 4.10 PSEUDOCODE FOR THE PROPOSED VOP LIST CONSTRUCTOR

Figures 4.2 and 4.3 demonstrate an example of how the scheduler generates the VOP list from the VOPs playout chart. Figure 4.2 shows the play out time chart of a general MPEG-4 sequence. The sequence has three video objects; VO_0, VO_1, and VO_2. VO_0 and VO_1 start at time unit 0 and VO2 at time unit 60. Their frame rates are 10, 25, and 50 frames per second respectively. VO_0 has two instances and is encoded as a sequence of one I and one P- VOP. VO_1 has five instances and is encoded as a sequence of one I, two B, and two P VOPs. VO_2 has seven instances and is encoded as a sequence of one I, two P, and four B VOPs. Moreover, VO_0 and VO_1 are both synchronized at the start of the session whilst all of them end at the same time. The maximum VOP encoding time for each of the VO_0, VO_1, and VO_2 is 100, 40 and 20 time units respectively.

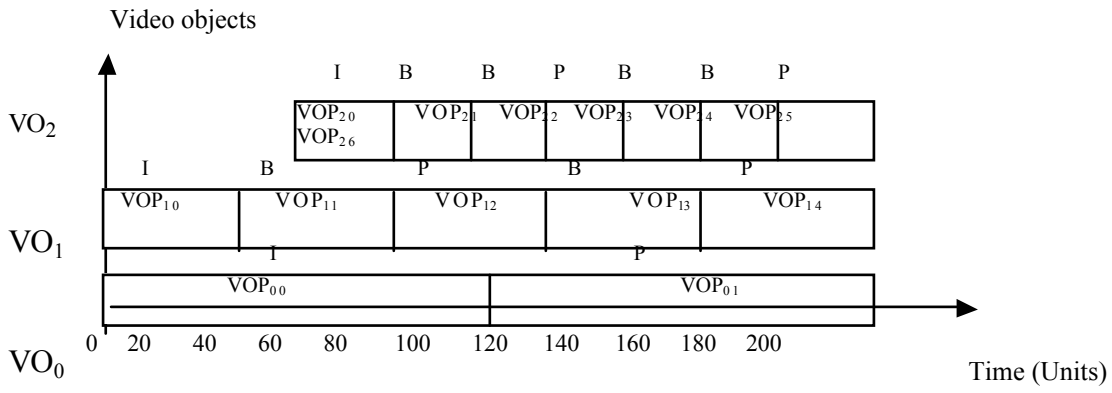


Figure 4.20: VOPs playout chart

Using the proposed selection policy the resulting list is shown in figure 4.3, with the VOP with the earliest deadline shown by the pointer at the top of the list. In case of deadline conflicts the VOP with the highest priority type is placed first in the list (I and P-VOPs-VOP_{00} and $VOP_{23,}$ P and B-VOPs-VOP_{26} and $VOP_{14,}$ P and B-VOPs-VOP_{01} and $VOP_{13,}$ and $VOP_{25,}$).

VOP_{10}, Type I, Deadline 40 ◄
VOP_{20}, Type I, Deadline 80
VOP_{12}, Type P, Deadline 80
VOP_{00}, Type I, Deadline 100
VOP_{23}, Type P, Deadline 100
VOP_{11}, Type B, Deadline 120
VOP_{21}, Type B, Deadline 120
VOP_{22}, Type B, Deadline 140
VOP_{26}, Type P, Deadline 160
VOP_{14}, Type B, Deadline 160
VOP_{24}, Type B, Deadline 180
VOP_{01}, Type P, Deadline 200
VOP_{13}, Type B, Deadline 200
VOP_{25}, Type B, Deadline 200

Figure 4.21: Deadline VOP list sorted in an ascending order

From this example it can be observed that the VOP selection mechanism of the proposed scheduler always selects the VOP with (i) the earliest deadline that can be encoded, as far it concerns its coding pattern constraints, and (ii) the most important coding type. I and P-VOP types are more important for the encoding procedure, since they are used as references for the encoding of other P and B-VOPs.

4.2.2 VOP distribution policy

As has been mentioned earlier a static allocation scheme leads to an unbalanced system since video object sizes vary over time. The scheduler needs to employ a dynamic VOP allocation scheme to allocate the system processors, that is related to VOP size. The scheduler for the producer-consumer model proposed in Chapter 2 creates, for each video object, a process that runs an identical MPEG-4 video encoder. So far, without the VOP size calculation the scheduler distributes the VOPs in a static fashion, with each video object assigned only one process that also restricts the degree of system parallelism to the fixed number of the video objects. Maximum system parallelism is achieved in the encoding of an MPEG-4 frame when all system processors are assigned an MPEG-4 thread/process. By calculating the VOP's size as a percent of the frame size, the scheduler determines the percentage of the maximum number of threads that could be created for this VOP. This information is passed to the consumer (MPEG-4 encoding process) of the corresponding video object, and thus dynamic VOP allocation occurs, since the consumer always creates a number of encoding threads that varies as the VOP size varies.

The scheduler allocates the VOPs to the consumers by employing two different

policies: the One VOP Scheduling (OVS) policy, and the Multiple VOP Scheduling (MVS) policy. With the OVS policy only one VOP at a time is distributed and encoded onto the system. The VOP size as a percent of the frame size is set to 100%. Thus, the corresponding MPEG-4 video encoder (consumer) encodes the VOP by creating for each MPEG-4 encoding module the maximum number of threads that exploit the system parallelism.

The MVS policy distributes more than one VOP concurrently to the processors. Each VOP is assigned to its corresponding consumer (MPEG-4 video encoding process). Further parallelism occurs at the process level (consumer) by creating threads for each module of the MPEG-4 encoding process. The number of threads that are created for each VOP in the consumer process is related to VOP size which is given as percentage of the frame size.

Scheduling decisions about the most appropriate VOP allocation policy occurs as follows:

The scheduler checks the number of video objects in the video sequence. If no more than one video object exists in the sequence, then only one consumer is created by the model, and the scheduler employs the OVS allocation policy. More specifically, as has been described in Chapter 2 the communication between the scheduler and the consumer is accomplished through a synchronisation buffer. When the consumer performs VOP encoding, it sets the flag of its synchronisation buffer. Every time the scheduler selects a VOP from the VOP list, it checks the flag status of the consumer's synchronisation buffer. If it is set, the scheduler blocks till the consumer finishes the encoding of its current VOP.

If more than one video object exists in the sequence, the scheduler employs the MVS allocation policy. Specifically, the scheduler selects the VOP with the earliest

deadline from the list, calculates its size as a percentage of the frame size, and checks the flag status at the synchronisation buffer of the corresponding consumer (encoder) for this video object. If the flag is not set the encoder sets it, and starts the VOP encoding, while the producer proceeds with the next VOP from the list. Otherwise the producer blocks till the encoder unblocks it by signaling for new VOP data.

Because video objects might appear and disappear from the video sequence, a possible scenario for the scheduler is to deal with the encoding of only one video object for a number of frames. In that case, the scheduler's allocation policy should be changed to the OVS. Therefore, when the MVS policy is employed, the scheduler checks if the first m VOPs in the sorted list belong to the same video object with the VOP that is encoded now by the system, m is the number of video objects that exist in the sequence. If so, that means that the scheduler cannot distribute concurrently any of the next m VOPs to the consumers, since all of them belong to the same video object and coding pattern constraints apply. Therefore, the OVS policy is employed for the next m VOPs of the list.

Similarly when the OVS distribution policy is employed, the scheduler checks if the first two VOPs of the list belong to the same video object. If not, then these VOPs can be encoded concurrently and thus the VOP allocation policy is changed for them from OVS to MVS.

4.3 Conclusions

This chapter presented a scheduler suitable for real time encoding that selects and distributes the MPEG-4 video objects onto a parallel architecture for encoding. The VOP deadline calculation is not a simple task since a frame consists of more than one video objects that might have different coding patterns, frame rates, and

furthermore, VOPs are likely to change over the encoding session, some enter and some leaving the scene. By using list scheduling, where the VOP deadlines are calculated according to the formula that takes into account the MPEG-4 encoding video object characteristics, the scheduler creates a list, in which the VOPs are placed in an ascending order of their deadlines. The selection of the appropriate VOP is achieved by using the earliest deadline first (EDF) policy.

Since VOP size varies over the encoding period the scheduler introduces two dynamic VOP allocation policies to the system processors, the OVS which is employed when one VOP per frame is encoded, and the MVS which is employed when multiple VOPs per frame are encoded. Both of them allocate the VOP to a group of processors that its size equals to the number of threads that will be created in the MPEG-4 encoding process. The number of the generated threads is related to the VOP size which is calculated by the scheduler.

Chapter 5

5 A Shape Adaptive Data Partitioning Scheme for MPEG-4 Video Encoding

5.1 Dynamic shape adaptive load balancing scheme for MPEG-4

The previous chapter addressed the issues that are related to the appropriate video object selection, and allocation onto the parallel architecture. Each VOP is dynamically assigned to a group of processors where the number of group processors is equal to the number of threads that will be used for the VOP

encoding. Further parallelism (spatial) is achieved inside the consumer process by dividing the video object into tiles that are processed by different processors. In MPEG-4 unlike MPEG-1 and MPEG-2, some computationally intensive algorithms of the encoder are data dependent, which makes their execution times different for different data areas of the video object. Transparent macroblocks do not need encoding. While contour macroblocks are processed by all algorithms (motion estimation, texture and shape coding) of the encoder, and opaque macroblocks do not need shape encoding. Table 5.1 depicts the operations performed on different types of macroblock.

TOOLS	TRANSPARENT MB	CONTOUR MB	OPAQUE MB
Motion estimation		X	X
CAE-Shape coding		X	
DCT/IDCT		X	X

Table 5.11 MPEG-4 TOOLS AND THEIR OPERATIONS ON MACROBLOCKS

Thus, if the video object data is partitioned into equally sized tiles that are encoded by different processors, it will eventually lead to unbalanced workloads among the processors. Data partitioning should be addressed carefully in the parallel processing in order to achieve real time MPEG-4 video encoding.

In its simplest form a data partitioning method restricts the data sub region to a rectangular shape in order to avoid the use of complex data structures.

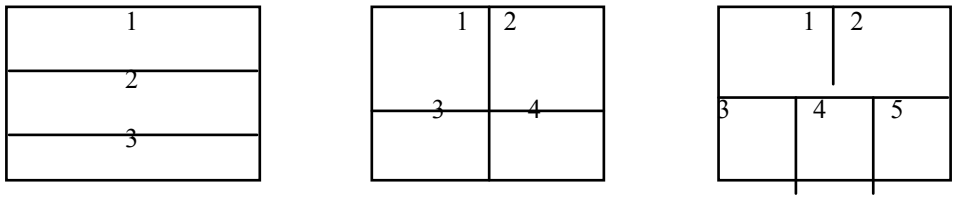


Figure 5.22: Partitioning methods: (a) Stripwise, (b) Blockwise, and (c) Recursive partitioning

Figure 5.1 shows several commonly used partitioning methods. Stripwise partitioning divides the data area horizontally, or vertically into n sub areas for n processors. Blockwise partitioning divides the data area evenly along both the horizontal and vertical dimensions. Finally, the recursive method [Berger 87] divides the data area recursively in a binary fashion. Although this method contributes to an optimally distributed load it is not widely used due to the relatively high cost of the recursive operations executed during the decomposition.

If the MPEG-4 video objects are large enough and their bounding box is filled entirely, the above mentioned partitioning methods may achieve load balancing, as the opaque and contour macroblocks are likely to be distributed uniformly on the system processors. But in the general case, there are VOPs sub regions composed of transparent macroblocks, and VOP sub regions composed of opaque and contour macroblocks, which makes the straightforward application of the above partitioning methods on the MPEG-4 video data inefficient. Moreover as the VOP size may be irregular, it may be even not possible to employ the strip-wise and blockwise data partitioning methods. Figure 5.2 presents a data partitioning example that uses both the stripwise and the blockwise techniques for the News 1 video object of the News segmented video sequence when it is sent for encoding on a four-processor platform. A bounding box surrounds the News 1 video object. All the macroblocks that are reside within the white area of the bounding box are either opaque, or contour (on borders) macroblocks, while the rest of them are transparent

Figure 5.23: Data partition example on "News". (a) Strip-wise partitioning and (b) blockwise partitioning

From the above example it is apparent that both partitioning techniques result in a distribution where some processors are assigned all the transparent macroblocks, which do not need processing, while others are assigned with computationally intensive opaque/contour macroblocks. In order to achieve load balancing, a shape adaptive data partition scheme is proposed [Hamosfakidis IWSNHC 99] that partitions video object data according to the actual macroblock processing requirements.

5.1.1 A Dynamic Shape Adaptive Data Partition Scheme

To accomplish load balancing among the system processors, the construction of video object tiles should based on the type of video object macroblocks. The type of a macroblock can be extracted from the video object's alpha plane. Specifically, by using the video object's alpha plane we construct a binary matrix that contains the type information of all the video object macroblocks. Each cell of the matrix corresponds to the co-located video object macroblock. If the macroblock is inside or on the video object's border the matrix cell is set to one, otherwise is set to zero, see figure 5.3. In this example the dancer video object from News sequence is used (size 96x80 pixels). A binary matrix of five rows (80/16=5) and six columns (96/16=6) is created from the information that is extracted from the MPEG-4 alpha plane. The transparent macroblocks are found in the black area of the bounding box are presented by the cells with the zero value in the binary matrix.

1	1	1	1	1	1
1	1	1	1	1	1
0	1	1	1	1	1
0	1	1	1	1	1
0	1	1	1	1	1

Figure 5.24: The dancer video object for the News sequence and its macroblock shape info table

Because the transparent maroblocks are not processed by the encoder, the shape adaptive scheme does not use them in the construction of video object tiles. The scheme knows the number of contour, and opaque macroblocks that exist in the video object by counting the number of non- zero valued cells in the generated matrix. The calculated sum is divided by the number of the system processors in order to derive the number of macroblocks per tile. The scheme creates the tiles as follows: tiles are formed by using all the video object's macroblocks, so that their corresponding matrix cell values are non-zero. The first tile is formed when the counting number of no transparent macroblocks reaches the appropriate value of macroblocks per tile. Then the scheme proceeds with the formulation of the next tile by starting to count the next non-transparent macroblock that follows the first tile. The same procedure is repeated till the formation of last tile.

For the above example, if the dancer VOP has to be distributed onto a two-processor platform, its tiles (A, B) are formed using the matrix information of figure 5.3. Tiles A, and B processing requirements depends exclusively on the macroblocks that require actual processing. Load balancing is "guaranteed" since

each processor is assigned equal workloads. Slice A contains 13 “active” macroblocks and slice B contains 14 “active” macroblocks, figure 5.4.

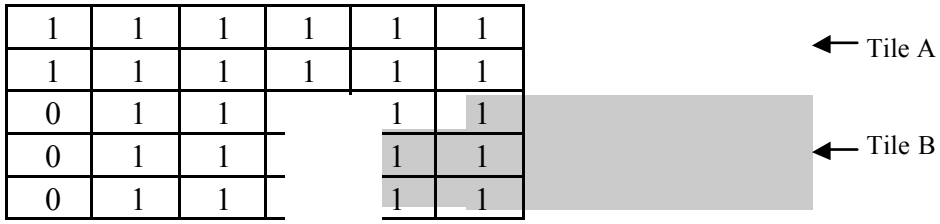


Figure 5.25: VOP generated tiles of dancer video object when is encoded onto a two-processor platform

A description of the proposed dynamic shape adaptive scheme in terms of pseudo code is given in table 5.2, where Num_Resources refers to the number of available resources, threads/processors, etc.

```
SHAPE_ADAPTIVE_SCHEME(Num_Resources) {

        FOR each instance of the Video Object  {

            WHILE(macroblocks_exist) {
                Find macroblock _type from video object's alpha plane;
                IF (macroblock.type!= TRANSPARENT) /*outside VOP */
                        Set corresponding matrix cell to one;
                ELSE
                        Set corresponding matrix cell to zero;
            } /* end WHILE */
```

```
                Divide the number of the non zero macroblocks by the Num_Resources;
                Form Video Object tiles using the number from the above division;
                Distribute Video Object to system processors for further processing;
            } /* end FOR */
} /* end SHAPE_ADAPTIVE_SCHEME */
```

Table 5.12 PSEUDOCODE FOR THE PROPOSED SHAPE ADAPTIVE SCHEME

Since typical MPEG-4 video encoding involves motion estimation/compensation, and texture and shape coding, the rest of this chapter will investigate the applicability of the proposed scheme at the MPEG-4 video encoder modules. By adopting the SPMD model at consumers level, each processor in a VOP's group will run an identical instance of the same module (motion estimation, or texture coding, or shape coding module) on different areas of the VOP. The proposed scheme is applicable, if and only if there are no macroblock dependencies. Therefore for each module of the video encoder an overview of its structure is given in order to identify parts where the proposed data partition scheme can be applied. Experimental results also will be given about the speed up that is accomplished by the scheme

5.2 Texture coding

5.2.1 Overview of texture coding

The MPEG-4 texture coding involves discrete cosine transformations (DCT), followed by DC/AC predictions, and variable length coding (VLC). The next paragraph overviews the texture coding mechanism in order to identify parts where the proposed scheme can be applied.

Transform coding is a very popular compression method for still image and video coding [Cho 91]. The most successful transformation is the DCT which is applied to data areas that form blocks of 8x8 pixels size. After the DCT transformation, the precision of the DCT coefficients is further reduced by quantising them on a block basis. The last step before the entropy coding is the lossless prediction of the quantised DC and AC coefficients. This procedure involves dependencies between different blocks of data since for the selection of the quantised DC (QDC), the values of neighbouring blocks are used [Puri97]. If X refers to the current block, A, B, and C correspondingly to the left block, to the block above and to the left, and to the block immediately above the current block X respectively, figure 5.5, then the QDC value of block X, QDCx, is predicted by the next formula.

If (|QDCA -QDCB| < |QDCB –QDCC|) QDCP = QDCC Else QDCP = QDCA ;

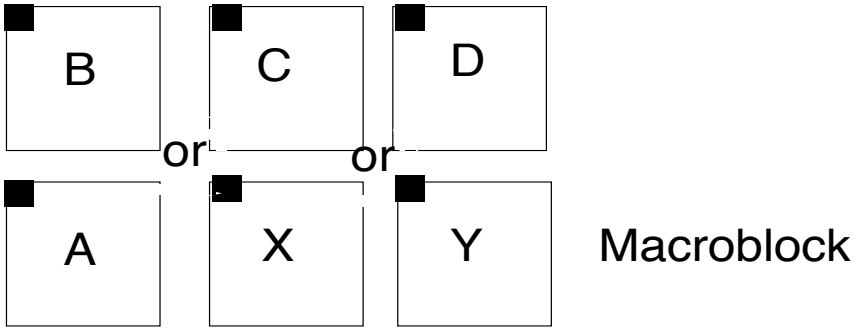


Figure 5.26: Previous neighboring blocks used in DC prediction

Similarly to DC prediction, the AC coefficients from either the first row or the first column of the previous coded block are used to predict the corresponding coefficients of the current block.

From this discussion it is clear that the proposed data partition scheme can be

applied only for the DCT transformation and the quantisation procedure, since no data dependencies exist between different blocks that might exist on different tiles which are encoded concurrently by different processors.

5.2.2 Experimental results of the proposed dynamic shape adaptive scheme on texture coding

By measuring the encoding time of the different parts of the texture encoder it was observed that the most computationally intensive part is the DCT and inverse DCT (IDCT) macroblock transformations. Due to the fact that these operations like the coefficient quantisation are performed on a block basis, it is possible to run all of them in a parallel fashion using the proposed data partition scheme.

As mentioned earlier, for our experiments a parallel implementation of the texture encoder has been developed to the Origin 200. In order to exploit the parallelism of Origin200 the thread model is used to implement the SPMD parallel paradigm. Each instance of the texture encoder runs on the system as a thread which performs DCT/IDCT and quantisation on a different area of the video object. The operating system schedules the threads to the system processors, and as the number of threads increases the utilisation of the system processors increases as well. Maximum system parallelism is achieved when the number of threads, num_threads, equals to or is greater than the number of processors. The num_threads is given to the system as a parameter before the MPEG-4 encoder starts. Initially, the MPEG-4 encoder runs as one thread, the “master” thread. When it starts the texture coding, the “master” thread checks the number of threads that has been requested to create. If num_threads=0, then texture coding is performed on the master thread. Otherwise, “slaves” threads are created. Each “slave” thread is assigned a VOP tile that is generated by the proposed scheme.

Experiments have been performed for the texture coding part of the News 0, News 1 (QCIF), and Coastguard 0 (CIF) video objects to demonstrate the efficiency of the proposed data partition scheme on the parallel processing of the texture coding. An efficient data partitioning mechanism leads to a load balanced system and in theory, if there are no data dependencies, to a speed up that equals the number of system processors. Thus, by drawing the graphs of speed up versus the number of threads, conclusions can be made about the efficiency of the data partition scheme.

Figure 5.6 shows the speed up curves for News 0, News 1, and Coastguard 0 video objects, when one, two, three, and four slave threads are used for the parallel processing of the DCT/IDCT and quantisation modules, when the proposed data partition scheme is used.

Figure 5.27: Speed up curves for the News 0, News 1, and Coastguard 0 video objects

By observing the above graphs the following can be concluded

(i) The speed up increases almost linearly as the number of threads increases in the four-processor Origin200, indicating the scalable behavior of the scheme. The speedup reaches the maximum value when the number of threads reaches the number of system processors.

(ii) The slope of the curves is related to VOP size and has a value of less than one. This is caused mainly by an overhead associated with "slave" threads scheduling mechanism. More specifically, thread's lifetime is related to texture coding time of video object's tile. Since texture coding performance is very fast, threads are created, scheduled, and destroyed very fast on the system. The kernel has to schedule with a high frequency new generated texture coding threads that last for a few microseconds or milliseconds. This leads to a non- optimal system parallelisation, since the time that is required by the kernel to start the threads on the system processors is significant, compared to the time that is required for the threads to execute their tasks on the system processors.

5.3 Motion Estimation and dynamic shape adaptive scheme

Earlier in Chapter 3 two fast motion estimation algorithms were proposed in order to reduce encoder's computational complexity. Although these fast motion estimation algorithms contribute significantly to the reduction of the motion estimation complexity, the amount of computation is still large for large size VOPs. If real time requirements need to be met, motion estimation can be applied on different data areas of the VOP to achieve parallel processing. The rest of this section investigates the application of the proposed scheme on the motion estimation part. It starts by dealing with issues regarding the conditions that need

to be satisfied for parallel motion estimation, and concludes by presenting speed up results for the proposed data partition scheme when different motion estimation algorithms and coding patterns are used.

5.3.1 Motion estimation/Motion compensation

As discussed in section 3.2, the motion estimation procedure derives the motion vectors by searching around the original macroblock position in the reference VOP. The reference is either a past VOP (P-VOPs) or a future VOP (B-VOPs prediction). The size of the search area at the reference VOP is dependent on a range variable (f_code), which is given to the encoder as a parameter. In the general case, the MPEG-4 search range around the original macroblock forms a window with dimensions 32x32 pixels for P-VOPs with unary temporal distance, while for other temporal distances the size of this window increases almost linearly. Access problems in the search area at the reference VOP arise when the VOP is divided into tiles that are processed by different processors, since each processor needs to have access to an area at the reference VOP with a size corresponding to the search window size.

When the proposed scheme is applied to cluster systems, the above problem is solved by cashing redundant data in the local memory of the processors [Akramullah 97]. Even simpler is the solution in the case of shared memory systems, where each processor has access to the same address space. The reference VOP is kept in the shared memory to which each processor has access. More precisely, each processor works with an area of the reference VOP where its size corresponds to its assigned tile size expanded by a number of macroblocks rows that used to form the complete search window area.

5.3.2 Experimental results

The parallel motion estimation search that uses the proposed data partition scheme is implemented on the four- processor SGI Origin 200. As in the case of parallel DCT encoding, in order to exploit Origin200 processing power the thread model is adopted again. Each instance of the motion estimation algorithm is running on the system as a thread. The number of motion estimation threads, num_threads, is given as a parameter before the MPEG-4 encoder starts running. Initially, the MPEG-4 encoder runs as one thread, "master" thread. When it enters the motion estimation part, the "master" thread checks the number of threads, num_threads, that it has been requested to create. If num_threads =0, then the motion estimation is performed on the master thread. Otherwise, "slave" threads are created and each "slave" thread is assigned a VOP tile that is generated by the proposed data partition scheme. The thread library provides a system call that allows multiple threads to read simultaneously the same area of data. This call is used widely in our implementation each time that more than one processor tries to access the same search area in the reference VOP. In this way, no memory access contention occurs when motion estimation is performed at the borders of two neighbouring tiles.

The efficiency of the proposed data partition scheme, in parallel motion estimation searching, is tested for different motion estimation algorithms, and different temporal distances. More specific, the experiments are performed for the full search algorithm, and the HS fast estimation algorithm using two different temporal distances. Unary one, for P VOPs coding patterns, where search area window size is 32x32 pixels, and another one for the IBPBP coding pattern where search area size is 48x48 pixels. Similar to the case of texture coding, by drawing the graphs of speed up versus the number of threads conclusions can be made about the data partition scheme at the parallel motion estimation search.

The speedup curves from the application of the data partitioning scheme for the

News 0, News 1 and Coastguard 0 video objects, when the HS motion estimation is applied for temporal distance one are given in figure 5.7, and for temporal distance two are given in figure 5.8.

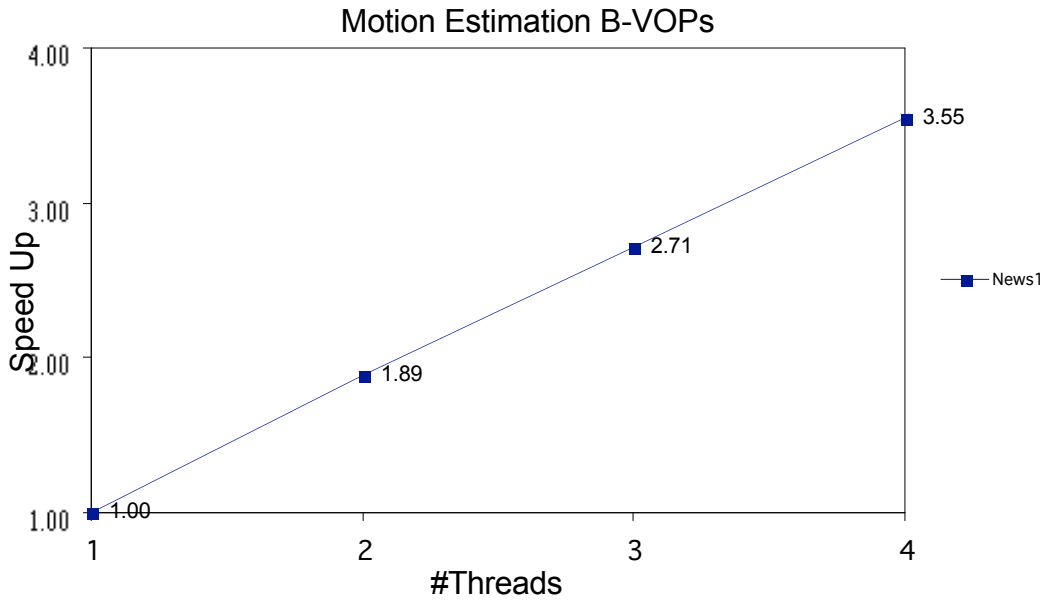
(a)

(b)

(c)

Figure 5.28: Speed up curves based on P-VOPs measurements for (a) the news 0 , (b) news 1, and (c) coastguard 0 video objects when the HS fast algorithm is applied.

(a)


Motion Estimation B-VOPs
4.00
3.00
2.00
1.00
Speed Up
1.00
1.89
2.71
3.55
1
2
3
4
#Threads
News1

(b)

(c)

Figure 5.29: Speed up curves based on B-VOPs measurements for (a) news 0, (b) news 1, and (c) coastguard 0 video objects when the HS fast algorithm is applied.

The speedup curves from the application of the data partitioning scheme for News 0, News 1, and Coastguard 0 sequences, when the full search motion estimation algorithm is applied for temporal distance one are given in figure 5.9, and for temporal distance two are given in figure 5.10.

(a)

(b)

(c)

Figure 5.30: Speed up curves based on P-VOPs measurements for (a) news 0, (b) news 1, and (c) coastguard 0 video objects when the full search algorithm is applied.

(a)

(b)

(c)

Figure 5.31: Speed up curves based on B-VOPs measurements for the News 0, News 1, and Coastguard 0 video objects when the full search algorithm is applied.

By observing the above graphs the following can be concluded.

(i) Similar to DCT, the speed up increases linearly increasing as the number of threads increases, indicating the scalable behavior of the scheme. The speed up reaches the maximum value when the number of threads reaches the number of system processors.

(ii) The scheme behaviour is independent of the size of search area. For both types of VOPs (P and B) the scheme has demonstrated the same behaviour in terms of speed up results.

(iii) The slope of the curves is related to VOP size and the type of the search algorithm that is employed for motion estimation. For larger VOPs, like Coastguard 0, the speed up curves for both P and B-VOPs

are slightly better from the corresponding speed up curves for the P and B-VOPs of the smaller size VOPs, News 0 and News 1. There is an overhead associated with the "slave" threads scheduling mechanism. The time that a VOP tile needs for motion estimation is related to the thread lifetime. If small size VOP tiles are used in motion estimation then their threads are created and destroyed more often than the threads that do motion estimation in bigger size VOP tiles. Thus the kernel has to schedule more often the threads on the system processors and if every few milliseconds or microseconds, threads are created and destroyed, the optimum system parallelisation cannot be achieved since it takes some time for the kernel to set up the threads on the processors. The same observations apply for the speed up curves that use different motion estimation search algorithms. The speed up curves (for all the video objects and all the VOP types) generated by the full search motion estimation algorithm are slightly better than the speed up curves generated by HS. Again this can be explained by the frequency of thread scheduling on the parallel system. When a fast motion estimation algorithm is applied, the time that a tile needs for the derivation of its motion vectors is significantly less than the time that this tile needs when the full search algorithm is used. Therefore, in the full search algorithm, the thread scheduling on the system processors happens less frequently than in HS.

5.4 Shape coding and proposed scheme

As it has been shown in table 5.1 only the contour macroblocks are processed by the context arithmetic encoder. At first glance, the shape adaptive data partition scheme seems very suitable for the MPEG-4 shape encoder. However, it is not applicable to the shape encoder, since the binary matrices that are used for the data

partition cannot be used in the case of shape encoding. In shape encoding the decision of whether or not to encode a contour macroblock is made dynamically. Furthermore there are a lot of other inter relations between macroblock data that affect the shape coding parallelism. The next chapter addresses the main issues related to the implementation of a parallel MPEG-4 shape encoding scheme.

5.5 Conclusions

Video object spatial parallelism is achieved by dividing the video object into tiles that are processed by different processors. In MPEG-4 unlike MPEG-1 and MPEG-2, some computationally intensive algorithms of the encoder are data dependent, which makes their execution times different for different data areas of the video object. This chapter presented a data partition scheme that generates tiles with identical processing requirements based on macroblock's actual processing requirements. Since typical MPEG-4 video encoding involves motion estimation, and texture and shape coding, this chapter investigated the applicability of the data partition scheme at the MPEG-4 video encoder modules. Experimental results on the four-processor Origin200 showed that:

1. For motion estimation, and texture coding the scheme speed up is almost linear, indication of the scalable behavior of the scheme
2. In the motion estimation the scheme behavior is independent of the size of the search area
3. The scheme performance is related to VOP size and in the case of the motion estimation is also related to the type of the search algorithm that is used.
4. Optimum parallelism is not achievable when the scheme runs on a thread environment, in which the overhead associated with the thread scheduling is significant compared to the thread lifetime.

Chapter 6

6 A New Fast Parallel Implementation of MPEG-4 Binary Shape Coding

6.1 Outline

This chapter investigates the complexity of the MPEG-4 shape encoder, by analyzing its parts, and proposes a scheme that reduces complexity suitable for real time parallel shape processing if a fast motion estimation algorithm is used. The most computational intensive task of the shape encoder is shape motion estimation (ME). This is shown by replacing the full search algorithm at the shape motion estimation part of the shape encoder with the HS and measuring the shape encoding time for the first 100 frames of the News 0 and Coastguard 0 video sequences. Table 6.1 shows clearly that motion estimation complexity dominates and determines the complexity of the shape encoder.

We developed a pipeline technique that allows video object data parallelism at the shape encoder.

	News 0	Coastguard 0
Total Shape Coding Time	3498 (msec)	11952 (msec)
Motion Estimation Time using the full search algorithm	1780 (msec)	7541 (msec)
CAE Time	594 (msec)	1734 (msec)
Coding Shape Decisions Time	65 (msec)	221 (msec)
VOP reconstruction and bit stream operations	260 (msec)	850 (msec)

Table 6.13 SHAPE CODING TIMES FOR THE FIRST 100 FRAMES OF NEWS AND COASTGUARD SEQUENCES

It should also be mentioned that the selection of the video objects, used in this Chapter, has been done in such a way that their alpha planes cover all the possible cases that the shape encoder encounters. There are two types of alpha planes in MPEG-4 video encoder: (i) size varies over time- Coastguard 0, or (ii) fixed size- News 0.

6.2 Binary Shape Encoder

As mentioned earlier, compared to other coding standards, MPEG-4 has the important capability to represent arbitrary shapes. If the video sequence is encoded as one video object (rectangular frame) no extra information about shape needs to be transmitted from the encoder. Otherwise, for arbitrary shaped objects the shape information must be encoded and transmitted to the decoder. The object shape information is given by the object's alpha plane which is formed from a rectangular bounding box, and surrounds the object in such a way that its horizintal and vertical dimensions are multiples of 16 pixels (macroblock size). The pixels of the bounding box that are on the boundaries, or inside the video object are called "white", and have a value of 255, while the pixels that are outside the video object are called "black", have a value of 0.

Each shape macroblock can be encoded in no less than five modes, Table 6.2

Mode	Description
"transparent"	All pixels within the macroblock are "black"
"opaque"	All pixels within the macroblock are white
"intra"	Macroblock is encoded using the intra template.
"inter"	Macroblock is encoded using the inter template
"no update"	No correction is applied within the macroblock after the motion compensation

Table 6.14 MACROBLOCK SHAPE MODE DECISIONS

“Intra” and “inter” shape coding is similar to “intra”, and “inter” texture coding. A transparent or opaque macroblock that belongs to an I-VOP is assigned either the “transparent” or the “opaque” shape mode. Contour macroblocks are assigned the “intra” mode. For “opaque” and “transparent” shape modes no further action is taken. For “intra” modes the macroblock is processed by the context arithmetic encoder (CAE) [Brady 97]. Macroblock shape decisions are encoded as well, and sent to the decoder.

For predictive coding the appropriate macroblock shape mode decision is chosen by driving a motion estimation/compensation scheme. In particular, if the error produced by the comparison of the original shape macroblock with the motion compensated macroblock is less than a threshold value the macroblock is assigned the “no update” mode. In the “no update” mode, the shape mode is encoded and sent to the decoder with additional information about the macroblock’s shape motion vector. Otherwise, if the error exceeds the threshold value the shape encoder checks the macroblock’s pixel values by dividing it into four sub blocks (4x4 size), and examining the sub block values. If all sub blocks are transparent or opaque the macroblock is assigned the “no update” mode. Otherwise it is sent to the context arithmetic encoder, where it is encoded both in “intra” and “inter” modes in order to select the mode that results in the best compression. Finally, the macroblock shape mode is encoded and multiplexed to the shape bitstream, with the macroblock’s compressed data produced by the arithmetic encoder.

6.3 Fast Search Shape Motion Estimation

6.3.1 Description of shape motion

The motion estimation procedure of the MPEG-4 shape encoder completes in two

steps: (i) an appropriate prediction of the shape motion vector (MVs) is found, and (ii) based on the prediction found in i) a shape motion vector is computed by the full search motion estimation algorithm. The prediction of the shape motion vector is estimated by selecting the suitable initial candidate among the motion vectors of the three previously decoded surrounding texture macroblocks, and the motion vectors of the three previously decoded surrounding shape binary macroblocks. The candidates are located and denoted as shown in figure 6.1.

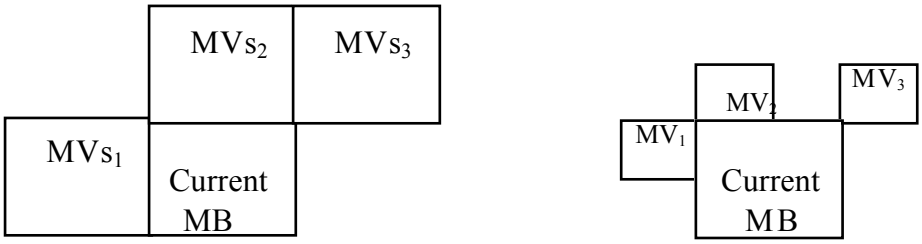


Figure 6.32: Candidates for the shape motion vector prediction

The initial shape motion vector prediction is determined by taking the first encountered motion vector that exists from the MVs_1, MVs_2, MVs_3, MV_1, MV_2, and MV_3 in this order.

In the next step, the initial candidate is either accepted as the shape motion vector, or is used as the basis for a new motion vector search, which depends on the resulting prediction errors. The motion compensated error (MC) is computed by comparing the macroblock indicated by the predicted shape motion vector with the current macroblock. If the computed motion compensated error is less than or equal to a predefined threshold value, the predicted shape motion vector is directly employed as the shape motion vector, and the procedure terminates. Otherwise, the shape motion vector is searched around the predicted shape motion vector. A

full search (FS) algorithm is employed using a search range of ±N pixels around the predicted shape motion vector. The sum of absolute differences (SAD) is calculated between the original macroblock and the macroblock indicated by the shape motion vector. The shape motion vector that minimises the SAD is chosen, and this is further interpreted as motion vector difference of shape (MVDs), where *MVDs = MVs - MVPs* is coded differentially.

In case of conflicts, when more than one shape motion vectors minimise the SAD by an identical value, the MVDs that minimises the code length (Q) is chosen. Q is given as:

Q=2 AbsoluteValue(MVDs in horizontal dimension) + 2 * AbsoluteValue(MVDs in vertical dimension) + (2-x)*,

where x=1 if horizontal element of MVDs is 0 and x=1 in any other case.

If more than one shape motion vector minimises the SAD by an identical value and by an identical Q, then the MVD with the smallest vertical element is chosen. If the vertical elements are identical, then the MVD with the smallest horizontal element is chosen.

6.3.2 Fast motion estimations algorithms and shape search

As shown in Table 6.1 motion estimation, especially the full search (FS) algorithm complexity dominates the overall shape encoder complexity. Therefore, it needs to be substituted by another fast motion estimation search algorithm. In Chapter 3 it was shown that there are different categories of fast motion estimation algorithms that can be used to speed up the texture motion estimation procedure. Are there any similarities between the texture and the shape motion field? If so, then it could be convenient to use the fast motion estimation algorithms developed for the texture field to the shape field. By examining the Coastguard 0, and News 0 alpha planes motion field was revealed that the shape motion, similar to the texture

motion, is smooth and varies slowly. That also leads to a center-biased, global minimum shape motion vector distribution instead of a uniform one. Figure 6.2 (a) shows that for the Coastguard 0 alpha plane, 58% of its shape motion vectors are found in a central square area of size 5x5. While for the News 0 alpha plane, nearly all its shape motion vectors considered stationary, 97.55% are found in a central area of size 5x5, figure 6.2 (b).

(b)

(a)

(b)

Figure 6.33: Shape MV distribution for (a) News 0, and (b) Coastguard

In Chapter 3 it was shown that the HS, and the diamond search (DS) algorithms are two fast block matching algorithms that outperform the full search algorithm. We use both of them in the shape motion estimation search, in order to observe their computational complexity performance, with the following enhancements.

- The HS that was shown in Chapter 3 employs two search patterns: a search pattern whose checking points form a hexagon, and a star pattern (deployed at the final step) with a variable number of searching points. When the search pattern is changed from hexagon to star, the original HS is modified in order to be compliant with the compression requirements of the shape encoder. Specifically, when the pattern changes to star for each pattern point as well as its SAD calculation the code length Q is also calculated. Therefore, if more than one searching point of the star pattern minimises the SAD by an identical value, the Q code length criterion is applied, to select the most appropriate point.
- The diamond search (DS) algorithm employs two search patterns, figure 6.3. The first pattern is called the "large diamond search pattern" (LDSP) and comprises of nine checking points that compose a diamond shape. The second pattern consists of five checking points forming a smaller diamond shape, called the "small diamond search pattern" (SDSP) [58]. When the search pattern changes from LDSP to SDSP, the code length Q is calculated

as well. Therefore, if more than one searching point of the SDSP pattern minimises the SAD by an identical value the Q code length criterion is applied to select the most appropriate point.

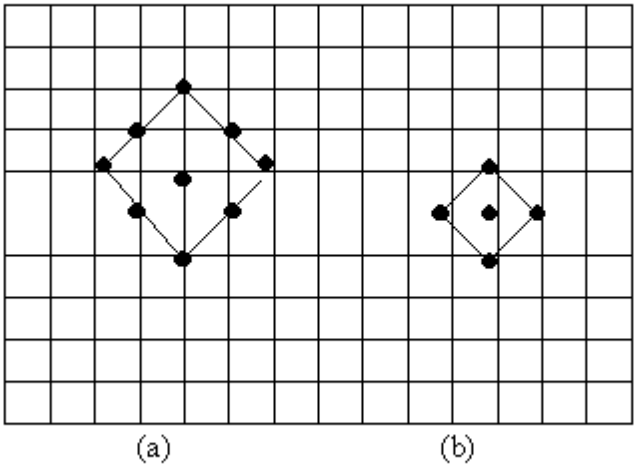


Figure 6.34: LDSP and SDSP patterns

For the following experiments the required average number of search points for each shape macroblock is used to measure the computational complexity, and the average number of bits per VOP is used to measure the compression efficiency. The maximum search motion range in each row and column is ±8 pixels. The results are shown in Tables 6.3

Algorithms	News 0			Coastguard 0		
	Av. SP per MV	Complexity	Bits per VOP	Av. SP per MV	Complexity	Bits per VOP
FS	256	100%	1001.86	256	100%	17618.10

DS	13.10	5.12%	1006.96	14.13	5.52%	17924.52
HS	10.06	3.92%	1006.93	10.97	4.28%	17854.54

Table 6.15 AVERAGE COMPLEXITY AND BITS PER VOP FOR THE FIRST 100 FRAMES OF NEWS 0 AND COASTGUARD 0

Table 6.3 shows that modified HS and DS outperform the FS, for both cases of alpha plane shape encoding (news 0 and coastguard 0), by reducing significantly the complexity of the motion estimation, and thus the overall shape encoding time. The trade off for using these fast search algorithms is a slight increase in the average number of bits per VOP, as is shown in the above table, due to the fact that fewer macroblocks are assigned the "no update" shape mode which results in better compression results.

6.4 A Parallel Shape Motion Estimation Scheme

As shown earlier in Chapter 5 not all macroblocks require the same processing power. By generating matrices that contain the macroblock type information, it is possible to create a data partition that leads to load balancing for motion estimation and texture coding before the actual video object processing. In shape coding, the macroblocks are coded not by their type information, but on their shape mode decisions that are decided for P and B VOPs on the fly. For P and B VOPs their shape mode decisions are based on their alpha plane values and the generated motion compensation error. Furthermore, more restrictions are applied to the degree of parallelism at the shape encoder for context arithmetic encoding, and I-VOP shape mode coding. The following paragraphs present the dependencies that prevent a straightforward application of the previous Chapter's data partition scheme.

6.4.1 Motion estimation dependencies

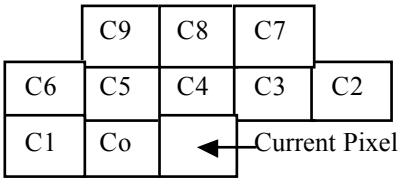
To derive current macroblock shape motion vector the shape motion vectors of previously encoded and decoded neighbouring macroblocks need to be used. The prediction of the (i,j) macroblock shape motion vector depends on the shape motion vectors of the ((i-1), j), (i, (j-1)), and ((i+1), (j-1)) macroblocks.

6.4.2 Dependencies in the encoding of shape mode decisions

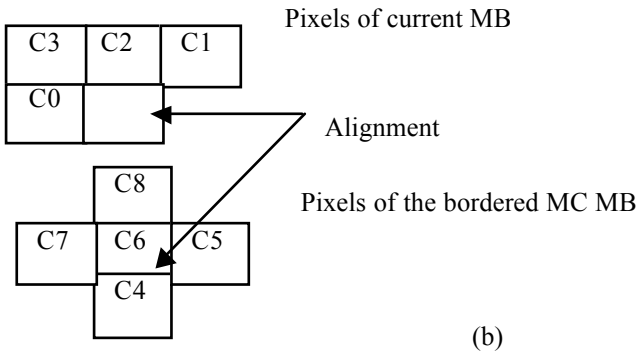
The shape encoder encodes the macroblock shape mode decisions of an I-VOP by determining a cord word for each macroblock. The cord word of (i,j) macroblock is generated by calculating number l, which indexes a table of cord words. To calculate the l the shape modes of the neighbouring blocks are needed. Index l is given as l =27*(f(i-1, j-1)-3)+9*(f(i,j-1)-3)+3*(f(i+1,j-1)-3)+(f(i-1,j)-3), where f(x,y) is one of the seven shape mode of the (x,y) macroblock [14469-2, 99].

6.4.3 Dependencies imposed by the Context Arithmetic Encoder (CAE)

In the context arithmetic encoding of MPEG-4 alpha planes, it is assumed that a high degree of local correlation exists between the alpha plane pixel values. Therefore for a given pixel the probability distribution is conditional upon the values of the pixels in the local neighbourhood. When the CAE is applied to a macroblock, pixels from neighbouring macroblocks are used to make up the context. MPEG-4 uses two different contexts, one for “intra” CAE macroblocks, figure 6.4 (a), and another for “inter” CAE macroblocks, figure 6.4 (b).



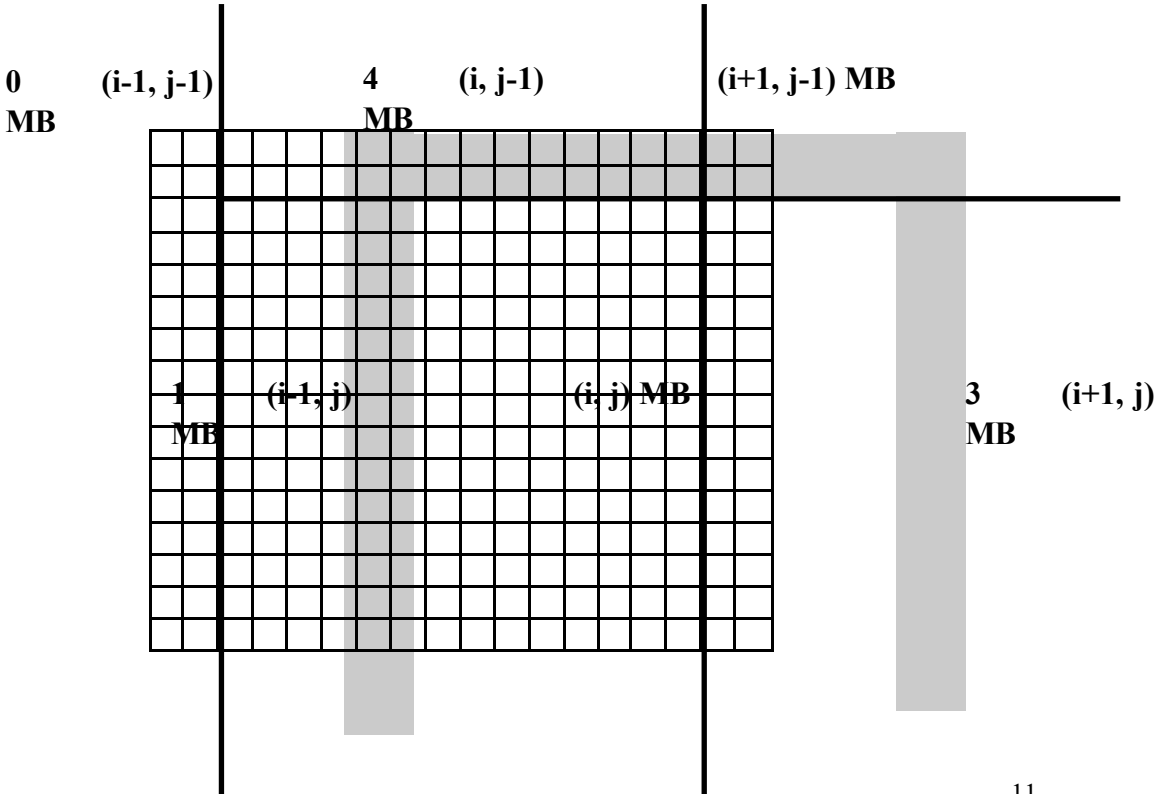
(a)



(b)

Figure 6.35: (a) "intra" template and context construction, (b) "inter" template and context construction

Therefore, for construction of "intra" or "inter" MPEG-4 templates, a pixel border of size equals to two, or one pixels has to be built around the macroblock. Figure 6.5 displays a bordered macroblock.



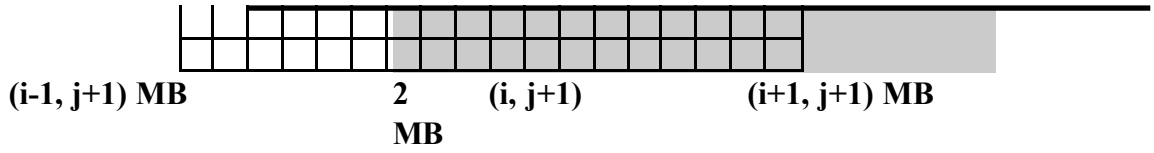


Figure 6.36: Current bordered Macroblock

The above figure clearly shows the data dependencies among the macroblocks when CAE is applied. Every macroblock of P and B VOPs is encoded both in "intra" and "inter" modes. In "intra" mode, for the context construction of (i,j) macroblock's top left and top right corner pixels, the pixel values of the (i-1, j-1), (i,j-1), and (i+1,j-1) previously decoded macroblocks are needed. Similarly when the "inter" template is built, the context construction of the top left and top right pixels of the (i,j) macroblock needs the pixel values of the previously decoded (i-1, j-1), and (i,j-1) macroblocks.

6.5 The Proposed Parallel Binary Shape Coding Scheme

To accomplish parallelism in a video object's shape encoding a pipeline scheme is introduced, where the macroblock dependencies are handled according to the following rule: "The macroblock (i,j) is processed by the processor P_n, where $1 \leq n \leq$ number of system processors, if the (i, j-1), (i+1, j-1) macroblocks have been processed by the processor P_{n-1}, and the (i-1,j) macroblock by the processor P_n"

As it was observed in the previous section, for the shape motion estimation, for the shape mode decision coding, and for the context arithmetic coding of the (i,j) macroblock, the (i-1, j),(i, j-1), (i+1, j-1) macroblocks need to be encoded first. The above rule makes sure that the macroblock is encoded only when this information is available. The macroblocks of the VOP alpha planes are encoded row by row from the proposed scheme. Every processor is assigned a row. If there are

more processors than rows then the processors that have not been assigned a row will remain inactive; if there are more rows than processors then it is likely more than one processors will be assigned more than one row. Macroblock processing happens as follows:

Initially the first row starts to be encoded, since there are no dependencies from above neighbouring macroblocks. The rest of the processors remain inactive for their (i,j) macroblocks till the (i, j-1),(i+1, j-1) macroblocks of the previous row have been processed. In particular, when the processor that encodes the first row has finished with the encoding of the first two macroblocks of its row, it signals to the processor that it has been assigned the second row to start its encoding. Similarly, when that processor has finished the encoding of the first two macroblocks of its row, it signals to the processor that has been assigned the third row. This procedure continues for every processor that has been assigned a row, figure 6.6.

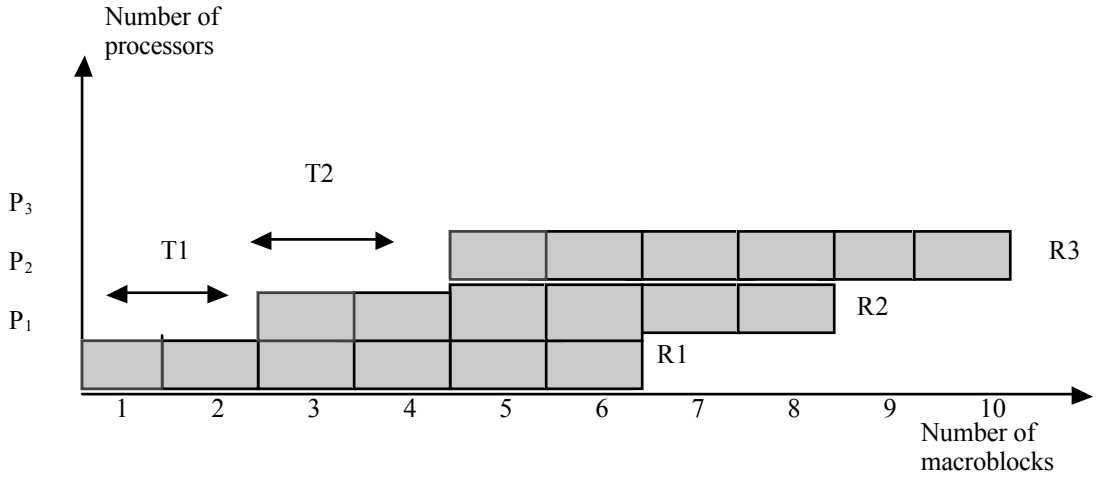


Figure 6.37: Macroblock pipeline mechanism

This figure demonstrates an example of three processors (P_1, P_2, P_3) that have to encode three rows (R_1, R_2, R_3) with six macroblocks each. In the time period T_1 only P_1 is active. P_2 cannot start its processing before the start of T_2. P_3 starts at the end of T_2 when the first two macroblocks of R_2 have been encoded.

Every time a processor needs to proceed with its macroblock shape coding, it checks if it can do that, if not it is waiting for notification from the processor that encodes the above row. When a processor has finished its row encoding, it checks if it has been assigned another row. If so the above rule is applied again.

By applying this rule for the shape encoding on a multiprocessor platform of n processors the maximum degree of parallelism, for a VOP of height m with m>n, will be accomplished after the encoding of 2*(n-1) macroblocks, which happens after n-1 steps, assuming that in each step two macroblocks are encoded. Figure 6.7 demonstrates this by an example where a VOP of seven columns and six rows is sent for shape encoding onto a multiprocessor platform consisting of three processors (P_1, P_2, and P_3). The VOP size is given as a table where the (i, j) cell corresponds to the (i, j) macroblock of the VOP, $0 \leq i \leq 6$, and $0 (j (5$. The value inside the cell denotes the time unit that the macroblock will be encoded. It is assumed that the macroblock's shape coding is completed in one time unit.

At time unit one and two, the (0,0), and (0,1) macroblocks are processed by the P_1 processor while at time unit three (0,2) and (1,0) macroblocks are processed simultaneously by P_1 and P_2. Maximum parallelism is achieved at the fifth time unit and lasts till time unit fourteen. From the fifteenth up to the sixteenth units only P_2 and P_3 processors are active, and at time unit seventeen only P_3 remains active. The shape encoding ends after two units.

	0	1	2	3	4	5	6	
0	1	2	3	4	5	6	7	P_1
1	3	4	5	6	7	8	9	P_2
2								P_3
3								P_1
4								P_1

5	6	7	8	9	10	11
8	9	10	11	12	13	14
10	11	12	13	14	15	16
12	13	14	15	16	17	18

Figure 6.38: Shape coding VOP parallelism for tile length equal to one row

An interesting observation about the proposed scheme is that it achieves and ensures the longest duration of maximum shape coding parallelism in comparison with any other VOP pipeline splitting mechanism. If the VOP is split into tiles of height greater than one row then maximum parallelism will occur significantly later. For the above example, if the VOP tiles had size two, maximum parallelism would have happened only after the eleventh time unit, figure 6.8.

	0	1	2	3	4	5	6	
0	1	2	4	6	8	10	12	P_1
1	3	5	7	9	11			
2	6	8	10	12				P_2
3	9	11						
4	12							
5								P_3

Figure 6.39: Shape coding VOP parallelism for tile length greater than one row

6.6 Experiments

The proposed scheme is implemented using the SPMD paradigm. The hardware platform where the experiments were performed is a four processor Origin 200. In

order to exploit its processing power the thread model is adopted again. Each instance of the shape encoder is running on the system as a thread. The kernel schedules the threads on the system processors. By increasing or decreasing the number of threads, we increase or decrease the degree of parallelism that we would like to exploit in the system. Maximum parallelism is achieved when the number of threads equals (or is greater than) the number of processors. The number of threads, num_threads, is given as a parameter before the encoding starts.

Initially, the MPEG-4 encoder runs as one thread, the "master" thread. When it enters the binary shape module the "master" thread checks the number of threads that it is requested to create. If no additional threads are requested, i.e. num_threads =0, then the shape encoding is continued on the master thread. Otherwise, the master thread creates "slave" threads. Each "slave" thread is assigned a VOP's row as follows: first thread first row, second thread second row etc. If more rows exist than threads then the remaining rows are assigned to the threads using the same assignment order from the first round of assignments. The thread proceeds with the encoding of its row macroblocks by applying the rule of the proposed scheme. If the rule allows the macroblock shape encoding, the thread continues, otherwise is blocked till the thread in the above row's macroblocks unblocks it. Tables 6.4 and 6.5 present the total shape encoding times for the first 100 frames of the news 0 and coastguard 0 video sequences, when the scheme is using one, two three, or four "slave" threads respectively on the Origin200

0 Number of "slave" threads	**Total Shape Encoding Time for news 0 object**
1	1740 (msec)
2	1150 (msec)
3	861 (msec)
4	700 (msec)

Table 6.16 TOTAL SHAPE ENCODING TIME FOR NEWS 0

1 Number of "slave" threads	Total Shape Encoding Time for coastguard 0 object
1	6540 (msec)
2	4224 (msec)
3	3190 (msec)
4	2546 (msec)

Table 6.17 TOTAL SHAPE ENCODING TIME FOR COASTGUARD 0

If no parallelism is used, the total shape encoding times for News 0, and Coastguard 0 video sequences are 1740 and 6450 milliseconds respectively. It is shown from the tables that as the number of threads increases the total shape encoding time decreases. If maximum parallelism is exploited- four "slave"threads, then the total encoding time for News 0, and Coastguard 0 is reduced to 700 and 2546 milliseconds respectively.

(a)

(b)

Figure 6.40:Speed up curves for (a) news 0 and (b) coastguard 0 alpha planes

From the above results, if speed up curves versus the number of threads are drawn, for the shape coding of Coastguard 0, and News 0 video sequences, figure 6.9, the following are observed.

(i) The speed up curves are almost linearly increasing as the number of threads increases, indicating of the scalable behaviour of the scheme. The speedup reaches the maximum value when the number of threads reaches the number of system processors.

(ii) The slope B of the speeding up line Y, where $Y = B*X$, has a value of less than one. This is expected since the pipeline mechanism does not provide full data parallelism. In addition, delays are added by the thread

synchronisation/signalling mechanism.

In conclusion the proposed scheme, even if does not provide maximum data parallelism of the shape encoding onto a parallel platform, reduces significantly the length of the encoding time. Figure 6.10 presents the total shape encoding times (in milliseconds) for the proposed scheme and the original MPEG-4 shape encoder on Origin200. The shape encoding has been performed for the first 100 frames of News 0, and Coastguard 0 alpha planes, and the scheme employs the fast motion estimation HS algorithm. Without the proposed scheme the shape encoding times for the News 0 and Coastguard 0 video sequences are 3498 and 11952 milliseconds respectively. With our scheme the total encoding times for the News 0 and the Coastguard 0 sequences are 700, and 2546 milliseconds. That shows a five times speed up in the shape encoding times of our video sequences.

Coastguard 0 **News 0**

Figure 6.41: Total shape encoding times for the original and the recommended fast parallel scheme.

6.7 Conclusions

In shape encoder the decision of encoding or not a contour macroblock is made dynamically. Furthermore macroblock dependencies in the (i) encoding of shape mode decisions, in the (ii) shape motion estimation, and at the (ii) arithmetic encoder prevent the application of Chapter 5 data partition scheme in shape encoder parallelisation. A circular pipeline algorithm that controls the VOP's macroblock flow on the system processors is introduced in this chapter. Experimental results showed that the algorithm demonstrates a scalable behavior and exploits the parallelism in the shape encoder when the macroblock dependencies allow that.

By examining the shape motion field we found that there are similarities between the texture and the motion estimation field. Therefore, we replace the full search motion estimation algorithm by the HS fast motion estimation algorithm. Experimental results on the four-processor Origin200 showed that the proposed scheme when it uses the HS algorithm achieves a five times speed up in the shape encoding time of the original shape encoder.

Chapter 7

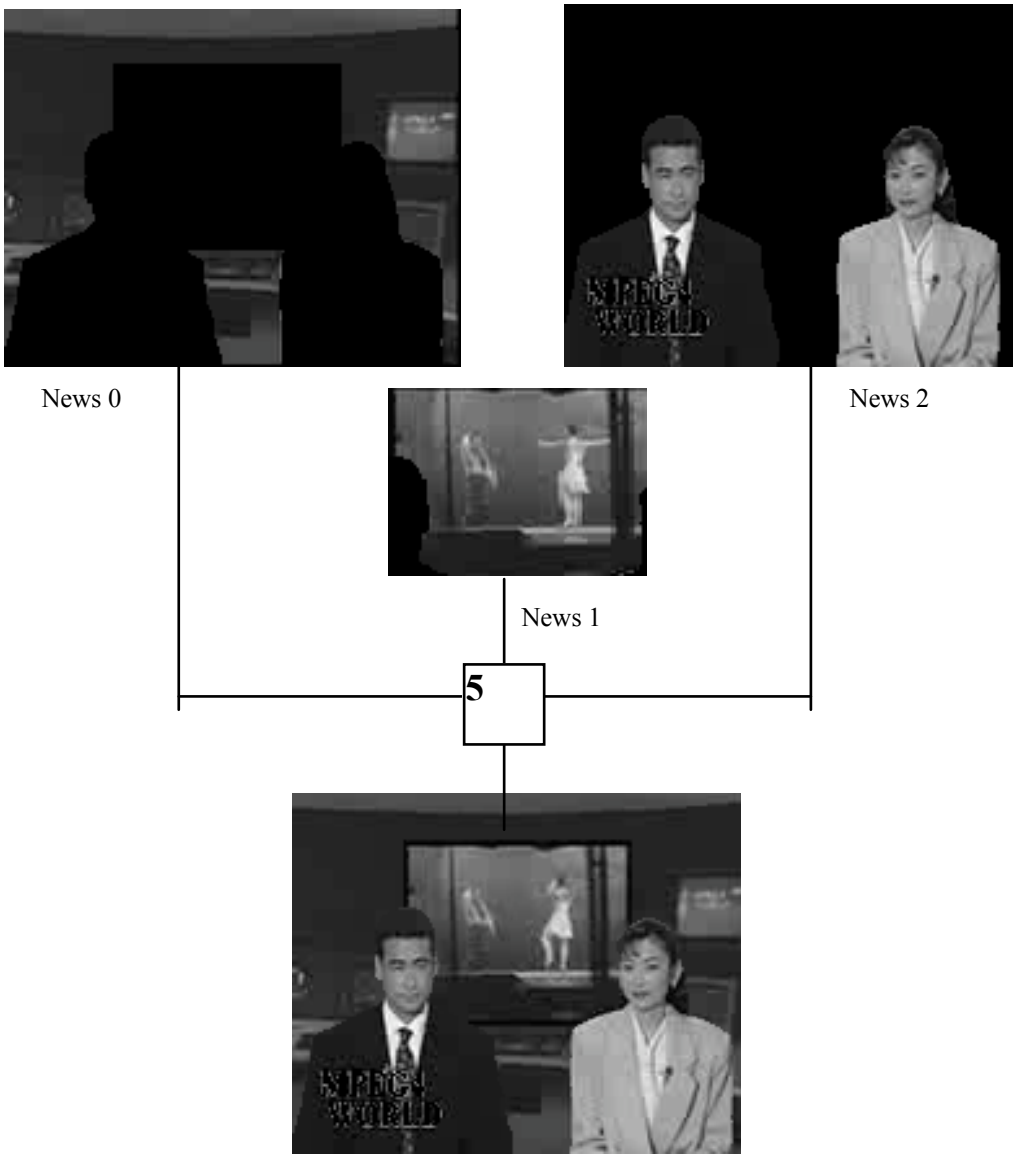
7 Experimental Results for the "Producer-Consumer" Model.

7.1 Outline of the experimental results

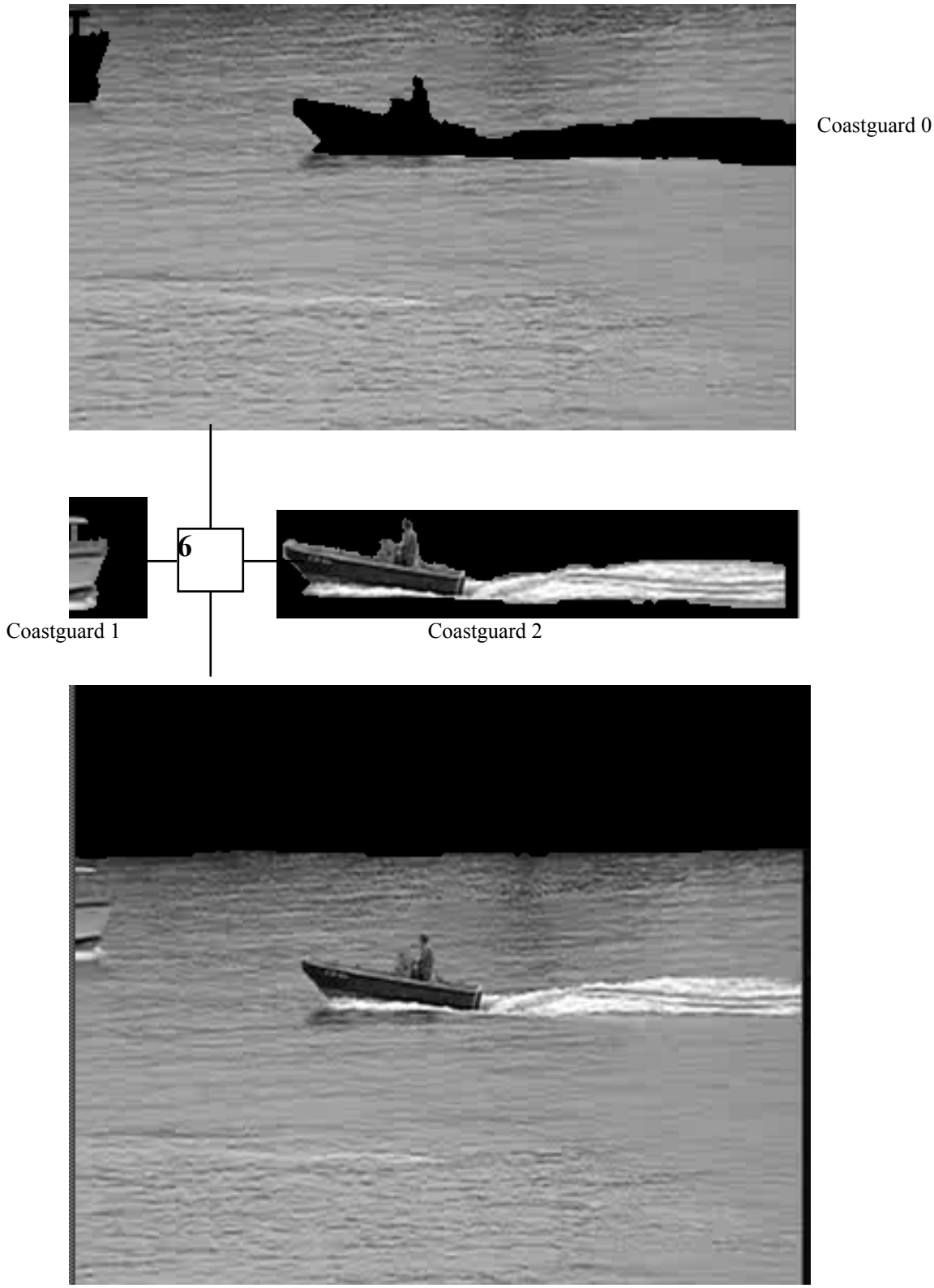
In Chapters 3, 5, and 6 experimental results were presented for the proposed solutions that used from the proposed model for the parallel MPEG-4 video encoding. Specifically, the results in Chapter 3 demonstrated the performance of the proposed fast motion estimations algorithm in terms of their computational complexity and image quality in the motion estimation part of the video encoder. In Chapter 5, results were presented for the proposed data partition scheme when VOP spatial parallelism is applied to motion estimation and texture coding modules of the MPEG-4 video encoder. Finally, the results in Chapter 6 have shown that the proposed shape coding scheme allows parallelism and speeding up the MPEG-4 shape encoding part.

The results of this Chapter plan to demonstrate the overall performance of the proposed model when MPEG-4 video encoding is applied on shared memory platforms. For these experiments the "producer consumer" model is used with the proposed scheduler of Chapter 4. For VOP spatial parallelism in motion estimation and texture coding parts of the consumer process, the recommended data partitioning scheme is employed, and for shape encoding the proposed shape

encoding scheme is applied. Additionally, in order to reduce the MPEG-4 video encoder computational complexity, the full search algorithm used in motion estimation is replaced by the HS fast motion estimation algorithm, described in Chapter 3. Once again, the shared memory platform for the experiments is a four processor Origin200, where maximum processor utilisation is achieved when each of its processors is scheduled to run one thread/process.



(a)



(b)

Figure 7.42: (a) Example of segmented News sequence with three objects (news 0, news 1 and news 2), (b) Example of segmented Coastguard sequence with three variable size objects (coastguard 0, coastguard 1 and coastguard 2

7.2 Single VOP per frame experiments

We have designed two different sets of experiments. The first set is performed on individual MPEG-4 video objects from the News and Coastguard video sequences, figure 7.1, and is intended to demonstrate the scalability of the proposed parallel MPEG-4 video encoding model. For this category of experiments the proposed "producer-consumer" model will proceed with the encoding of the MPEG-4 objects as follows.

Initially the scheduler of the producer process creates a sorted list according to VOP deadlines for all VOPs of the MPEG-4 video object sequence. Since only one video object is encoded by the system, the scheduler adopts the OVS allocation policy. Specifically, the scheduler sends the consumer (MPEG-4 video encoding process) the VOP with its size information, which in this case is 100% (as a percent of the frame size). The consumer process, in order to exploit the Origin200 processing power creates, for each module of the MPEG-4 encoder (motion estimation, texture coding, and shape coding), the maximum number of threads that exploit the maximum degree of system parallelism (in this case four). Each thread runs on a different part of the VOP, where the VOP data partition is performed by the scheme proposed in Chapter 5 scheme. For the binary shape coding, the consumer exploits system parallelism by creating threads that run the shape encoder according to the proposed scheme of Chapter 6. Finally, when the consumer process has finished with its VOP encoding, it notifies the producer to send the next VOP from the sorted VOP list, see figure 7.2.

Figure 7.2 shows how the "producer consumer model" encodes a single VOP per frame when it runs on the four-processor, Origin200.

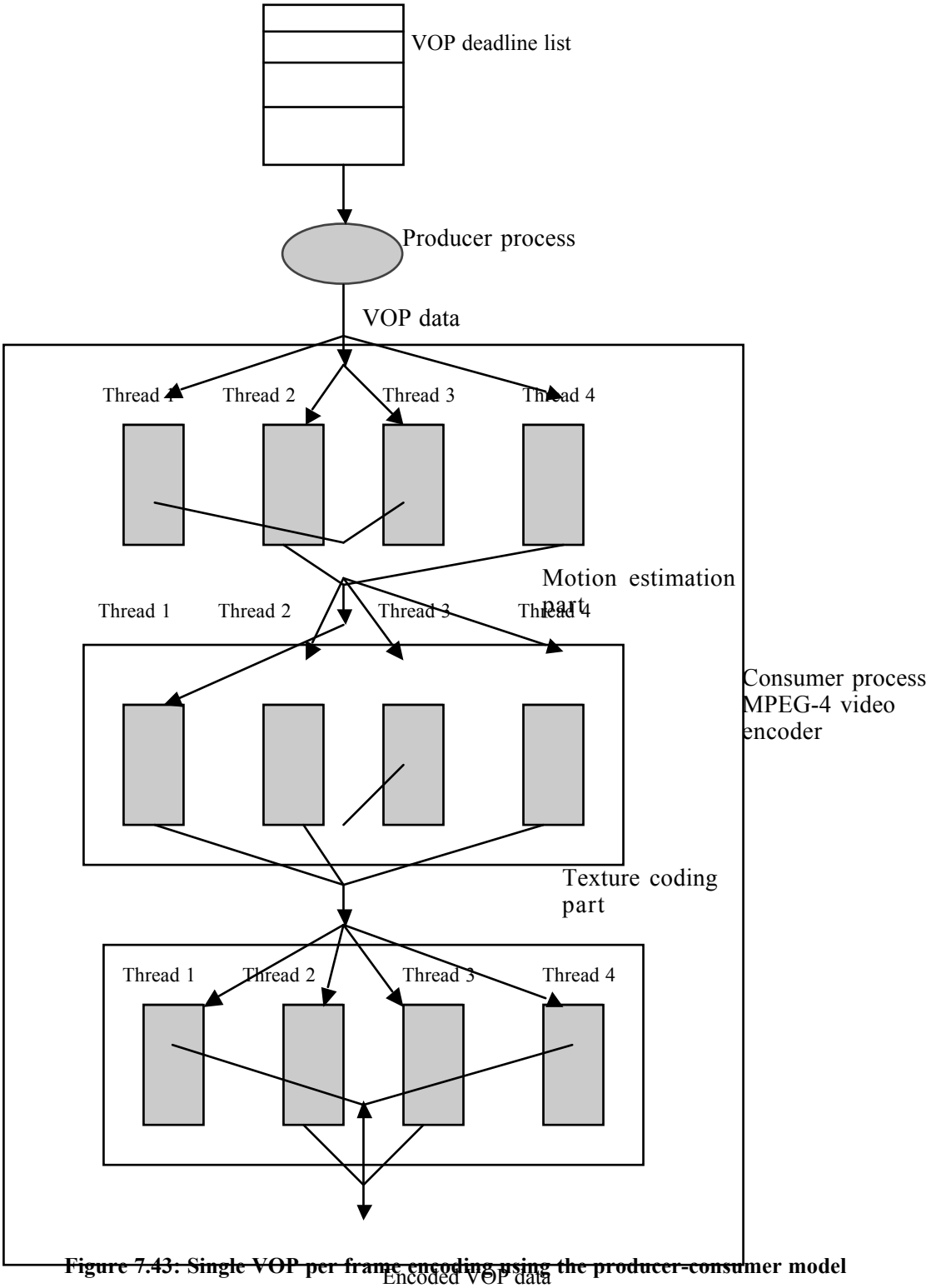


Figure 7.43: Single VOP per frame encoding using the producer-consumer model

The framework scalability is estimated by drawing the graph of the number of encoded frames per second, versus the number of system threads, since parallelism is exploited by the number of threads/processes that run concurrently to the system. More processors allow more threads for the MPEG-4 video encoding modules to run concurrently, and that results in faster encoding and faster encoding frame rates per second. The number of threads that the consumer creates is given as a parameter before the encoding. If we want to exploit the maximum system parallelism, the parameter that determines the number of threads is set to –1. In this case, the consumer retrieves from the system the number of processors, and creates equal number of threads. In any other case, the consumer gets the value of the threads parameter, and creates equal numbers of threads. The proposed model runs for different scenarios that involve no parallelism up to the maximum, figure 7.3.

(a)

(b)

(c)

Figure 7.44: Encoding frame rate for news 1, coastguard 1, and news 2 video objects

Figure 7.3 presents the graphs for the News 1, News2, and Coastguard 1 video objects, when different numbers of threads are used on the Origin200. News 1, and News 2 are fixed size video objects, whilst Coastguard 1 is a video object whose size varies over the encoding time. By observing the above graphs the following can be concluded

The encoding frame rates increase almost linearly as the number of threads increases, indicating the scalable behaviour of the scheme. Maximum frame rates are reached when the number of threads is equal to (or greater than) the number of system processors.

The slope of the graph curve, which indicates the resulting speedup, has a value of less than one. In theory, when maximum parallelism is employed, four threads in case of the four processor Origin200, the speedup should be four times in terms of encoding frame rates per second. When no thread parallelism is applied to the proposed model for News 1, News 2, and Coastguard 1 video objects their encoding frame rates are 16.95, 10.41 and 11.18 respectively, while when maximum parallelism applied their encoding frame rates are 29.12, 17.19, and 23.20 respectively. This could be explained by:

(i) Threads lifetime. More specifically, as it was shown in Chapters 3 and 5 the thread's lifetime is related to VOP's size. If the VOP size is small then the threads that created for the motion estimation and texture coding modules are executing very fast. The kernel has to schedule very frequently new texture coding or motion estimation threads that last only for few microseconds or milliseconds. This leads to non optimal

system parallelism, since the time that is required from the kernel to start the threads on the system processors is significant compared to the time that is required for the threads to execute their coding MPEG-4 module tasks on the system processors.

(ii) Binary shape coding. -The MPEG-4 video encoder performs binary shape encoding, which was shown in Chapter 6 can not be parallelisable in the same way that texture and motion estimation is parallelised, due to data dependencies between VOP macroblocks. The proposed MPEG-4 binary shape scheme provides thread parallelism, but through a pipeline mechanism with a lot of synchronisation control among threads that adds additional overhead.

7.3 Multiple VOPs per frame experiments

The second set of experiments is performed on the MPEG-4 News and Coastguard video sequences, and is intended to demonstrate the speed improvement in terms of total encoding times, or encoding frame rates per second, when the proposed "producer-consumer" model is used into a shared memory platform compared with the original software MPEG-4 encoder. All the segmented video objects of the News (News 0, News 1, News 2, and News 3) and Coastguard (Coastguard 0, Coastguard 1, Coastguard 2, and Coastguard 3) sequence are used at the encoding of each sequence.

More specifically, first we encode the sequences using the original MPEG-4 software video encoder on the Origin200. The original video encoder handles the sequences as follows:

It finds, for each frame, its video objects and for each video object performs the encoding using at motion estimation, the full search algorithm. When the encoder

has finished with the encoding of all the frame's video objects, it continues with the video objects of the next frame. The total encoding times for the first one hundred of the News and Coastguard video sequences are 82190 and 370215 milliseconds respectively, and their encoding frame rates per second are 1.2 and 0.2.

Then the same sequences are encoded using the proposed model. For this set of experiments the "producer-consumer" model will proceed with MPEG-4 video sequence encoding as follows:

Initially the scheduler of the producer process creates a sorted list according to VOP deadlines for all VOPs of all the video objects of the MPEG-4 video sequence (News and Coastguard), as has been shown in Chapter 4. Since more than one video objects exist in the scene (their number is four), and can be encoded by the system concurrently, the scheduler employs the MVS VOP allocation policy, section 4.2.2. The scheduler sends the consumers (MPEG-4 video encoding process) the VOP with its size information as a percent of the frame size. For each consumer the scheduler checks its flag status at the synchronisation buffer. If it is off, it selects the next VOP from the sorted list, and proceeds in the same way. Otherwise if the flag is set, then the scheduler blocks until the consumer unblocks it. The consumer process, as described previously, exploits the Origin200 processing power by applying VOP spatial parallelism, using the proposed data partition scheme at VOP's motion estimation and texture coding parts. For the shape encoding, the proposed parallel shape scheme is used.

The total encoding times for the first one hundred frames of the News and Coastguard video sequences for the "producer-consumer" model are 9830, and 30512 milliseconds respectively, and their encoding frame rates per second are 10.2 and 3.33 frames.

From the above results, expressed either in terms of total encoding times, or encoding frame rates per second, it is clearly shown that the proposed model for the parallel MPEG-4 video encoding outperforms significantly the original MPEG-4 video encoder. More specifically, on the four-processor platform, Origin200, the proposed MPEG-4 scheme achieves at least ten times faster encoding frame rates than the original encoder for News and Coastguard sequences, or in terms of total encoding times the encoding of the segmented News and Coastguard video sequences takes approximately 1/10th of the total encoding time that the original MPEG-4 encoder needs, figure 7.4.

News QCIF **Coastguard CIF**

Figure 7.45: Total encoding times for original MPEG-4 encoder and for proposed framework.

7.4 Conclusions

This chapter presented experimental results for the "producer-.consumer model". Two well-known benchmark video sequences are used in our experiments the News and the Coastguard. Both of them cover a wide range of MPEG-4 encoding requirements. The News is a QCIF video sequence and its frame is composed of

video objects with different motion requirements (slow, medium and high). The Coastguard has CIF resolution and its frame consists of video objects that their size varies over time. The first set of experiments showed that the scheme demonstrates a scalable behaviour which is independent of specific VOP characteristics such as:

1. motion requirements (News 0 is a background object with slow motion, and News 1 with high motion requirements),
2. video sequence resolutions (Coastguard 0 has CIF resolution, and News 0 and 1 have QCIF), and
3. MPEG-4 video object's size (the size of coastguard video object varies over time).

When the maximum system parallelism is exploited on the Origin200 the encoding frame rates reach the maximum value which for the case of the News 1 video sequence is real time encoding (almost 30 frames per second). The same observations are applicable in multiple VOPs (per frame) encoding where instead of one encoding process and multiple threads we have the scenario of multiple encoding processes using multiple encoding threads.

The second set of experiments evaluated the encoding performance of the proposed model using as benchmark the encoding performance of the original MPEG-4 encoder. It was shown that on the four-processor Origin200 the proposed scheme performs ten times faster than the original encoder for sequences that have different encoding characteristics in terms of resolutions, video object sizes, and motion requirements.

Chapter 8

8 Conclusions and Future Work

8.1 Conclusions

This thesis has presented a software model, the "producer-consumer" model that allows parallel processing for the MPEG-4 video encoder on shared memory architectures, in order to reduce its total video encoding time.

The emerging MPEG-4 standard has introduced and supported the concept of video objects. Each frame is segmented into objects where their coding requirements depend on a variety of parameters. Before the MPEG-4 standard, video encoder speedup (H.263, MPEG-1 and MPEG-2) was accomplished by fast motion estimation algorithms, and the use of parallel processing where the exploitation of the spatial, or the temporal parallelism of the video sequence provided a reliable framework for the parallel video encoding. For instance, using the spatial approach maximum parallelism was achieved by dividing the frame into equal size tiles, where their number is equal to the number of system processors.

Due to the object-oriented approach of the MPEG-4 video encoder, its video encoding onto a parallel architecture becomes a research challenge with no straightforward solutions.

Since a video sequence consists of video objects that are more likely to have different frame rates and sizes that vary over the video session, this framework incorporates a scheduler that (a) selects the VOPs taking into account precedence constraints, frame rates, and deadlines using priority list scheduling with an EDF policy, and (b) allocates VOP's onto the system processors by using a dynamic allocation mechanism based on the VOP's size information. This allocation mechanism employs two different policies, the OVS which applied when one video object exists in the sequence, and the MVS policy which applied when more than one video objects exists in the sequence.

When a VOP is dynamically allocated to a group of processors, the maximum parallelism inside the group is achieved by applying the SPMD paradigm onto the different modules of the MPEG-4 video encoder. More specifically, for each module of the MPEG-4 video encoder (texture coding, motion estimation) threads are created, where their number is equal to the number of group processor. Each thread runs the same module on a different VOP data area (spatial parallelism). Due to the fact that all macroblocks do not have the same processing requirements the model also has introduced a data partition scheme that generates VOP tiles with identical processing requirements. Moreover, since the parallelism of the shape encoder is not applicable, due to the macroblock dependencies at shape encoding, a scheme has been developed and used by the model that allows parallelism using a circular pipeline macroblock mechanism.

Moreover, since the encoding time depends not only on the processing power of the underlying hardware platform but also the encoder's computational complexity, this thesis deals with motion estimation algorithms, because their complexity has a significant impact on the complexity of the encoder. Particularly, two fast motion estimation algorithms have been developed for the model which reduce significantly

the computational complexity.

The results in Chapter 3 present the impact of the proposed fast motion estimation algorithms in motion estimation complexity. Specifically, it is clearly shown that the proposed HS fast motion estimation algorithm outperforms not only the full search algorithm, but and other well known motion estimation algorithms from the category of the algorithms that reduce the number of search locations such as DS, NTSS, and 4SS.

The experiments in Chapter 5 have demonstrated that the proposed data partition scheme for the texture and the motion estimation parts, results in a speed up that almost linearly increases as the number of threads increases, indicating the scalable behaviour of the scheme. By increasing the number of system processors we can increase the number of threads that run on the different VOP areas, and thus speed up further the encoding process. Similarly, the results in Chapter 6 demonstrate scalable behaviour for the proposed shape coding scheme, where the speed up improved significantly by replacing the full search shape motion estimation algorithm by the HS.

Finally Chapter 7 experiments with the "producer-consumer" model, that incorporates all the parts that have been discussed so far in this thesis, show a scalable framework for shared memory architectures where scalability is provided by the number of threads that are assigned to system processors. The model is also eligible for real time encoding, since it sorts the VOPs according to their encoding deadlines, taking into account coding pattern constraints, frame rates, and start session times.

8.2 Further work

In this thesis the proposed framework incorporates a scheduler that selects the VOPs according to their encoding deadlines and then distributes them to system processors using dynamic VOP allocation policies (OVS, MVS), where the number of processors in the group depends on VOP's size and numbers of video objects of the video sequence.

While the scheduler selects the appropriate VOPs for real time encoding does not have any policy that treats the case when a VOP cannot meet its deadline due to insufficient processing power. More specifically, the scheduler is developed in such a way that it exploits all the available processing power of the shared memory platform, but cannot make decisions about the selected coding pattern of a video object that its deadline cannot be met, or about its encoding frame rate. Since the selected coding pattern has an impact on the total video object's encoding time, further work should include a scheme that roughly estimates the encoding times of different coding patterns for each video object, based on current encoding times. For instance, if the video object's coding pattern for an intra period consists of more than one P VOP, the scheme by making the assumption that the VOP size does not change dramatically in the intra period, it will assigns to the P-VOPs of the intra period the encoding time that was calculated for the first P VOP of the period. Then it will check for each VOP of the period if its estimated deadline can be met. If not, it will replace the video object's coding pattern for this period with another pattern consisting of VOPs that their estimated deadlines can be met (maybe a coding pattern with less B-VOPs or P-VOP). The length of the intra period where the VOP size does not change significantly needs investigation.

Similarly, if the scheduler has also been provided with information regarding video

objects visibility importance when displayed on the video sequence, then further work would involve the development of a scheme that exploits this information in terms of video object frame rates. If a VOP cannot meet its deadline using the current frame rates, the scheme based on the visibility importance of this object would be able to adopt another frame rate that meets VOPs encoding deadlines for this video object.

8.3 Publications

Part of the work presented in this thesis has been published (or submitted) in the following papers:

A.Hamosfakidis, and Y. Paker, "Survey of Multimedia Architectures", Technical Report, ACTS project AMPA, Copyright of AMPA Consortium 1997 [17].

A.Hamosfakidis, and Y. Paker, "Concurrency analysis for real-time MPEG-4 video encoding", ", In Proceedings IEEE Multimedia Systems'99, vol II, pp. 862-866, Florence June 1999 [29].

A.Hamosfakidis, Y. Paker, and J. Cosmas, "A study of concurrency in MPEG-4 video encoder", In Proceedings IEEE Multimedia Systems'98, Austin, pp. 204-208, June 1998 [32].

A.Hamosfakidis, J. Cosmas, Y. Paker, and A. Pearmain, "Parallelisation of MPEG-4 video verification model encoder in inter/intra separate mode", Doc ISO/MPEG 1889, MPEG Bristol meeting, April 1997 [34].

A.Hamosfakidis and Y. Paker, “A fast motion prediction scheme for the MPEG-4 video encoder”, WIAMIS’99, pp. 125-129, May-June 1999 [61].

A.Hamosfakidis, and Y. Paker, “An MPEG-4 parallel shape adaptive motion estimation scheme”, Intern. Work. Synthet.-Natural. Hybrid Cod. and Three Dimen. Imag., IWSNHC3DI’99, pp. 41-45, September 1999 [65].

A.Hamosfakidis, and Y.Paker, “A new parallel MPEG-4 scheme for fast binary shape coding”, submitted in ACM Multimedia 2000, November 2000, Los Angeles, USA.

A.Hamosfakidis, and Y. Paker, “A novel Hexagonal Search algorithm for fast block matching motion estimation”, submitted in IEEE Signal Processing Society 2001 International Conference on Image Processing, Thessaloniki, Greece

9 REFERENCES

[Adam 74] T.Adam, K.M.Chandy, J.R.Dickson "A comparison of list schedules for parallel processing systems", Commun. ACM, 17, pp. 685-690, 1974.

[Ahmad 95] I. Ahmad, "Resource management on parallel and distributed systems with static scheduling: Challenges, solutions and new problems", Concurrency: Pract. Exp., vol. 7, pp. 339-348, August 1995.

[Akramullah 97] S. M. Akramullah, I. Ahmad, and M. L. Liou " Performance of software based MPEG-2 video encoder on parallel and distributed systems", IEEE Transactions on Circuits and Systems for Video Tech., vol. 7, no. 4, pp. 687-696, August 1997.

[Andre 91] M.Andre and G. Koob, Foundations of real-time computing: scheduling and resource management, Kluwer academic publishers, 1991.

[Berger 87] M.J. Berger and S.H. Bokhari, " A partitioning strategy for non uniform problems on multiprocessors", IEEE Trans. Comput., vol. C-36, pp. 570-580, May 1987.

[Bhaskaran 97] V. Bhaskaran, and K. Konstantinides, Image and Video Compression Standards, Kluwer Academic Publisher, Massachusetts, 1997.

[Bierling 88] M.Bierling, “Displacement estimation by hierarchical block matching”, In Proceedings SPIE Visual Commun., and Image Processing”, vol. 1001, pp. 942-951, 1988.

[Brady 97] N. Brady, F. Bossen, and N. Murphy, “Context based arithmetic encoding of 2D shape sequences”, International Conf. on Image Processing, vol II, Santa –Barbara, pp. 29-32, October 1997.

[CCITT 90] CCITT, Recommendation H.261: Video codec for audiovisual services at p*64 kbit/s. The International Telegraph and Telephone Consultative Committee, 1990.

[Chalidabhongse 97] J. Chalidabhongse, and C. Kuo, “ Fast motion vector estimation using multiresolution-spatio-temporal correlations”, IEEE Trans. Circuits Syst. Video Technol., vol. 7, no. 3, pp. 477-488, June 1997.

[Chetto 89] H. Chetto and M. Chetto “Some results of the earliest deadline scheduling algorithm” IEEE Trans. on Software. Engineering., vol 15, no. 10, pp. 1261-1269, Oct. 1989.

[Cho 91] N.I.Cho and S.Lee, “Fast algorithm and implementation of 2-D Discrete Cosine Transform”, IEEE Trans. on Circuits and Syst,, vol. 38, no.3, pp. 297-305, March 1991.

[Coffman 76] E.G. Coffman, Computer and Job-Shop scheduling theory, John Wiley & Sons, Somerset, N.J., 1976.

[Diepold 98] Klaus Diepold, “MPEG-4 Applications Document”, ISO/IEC JTC1/SC29/WG11/ N2322, MPEG 98, Dublin, July 1998.

[Dufaux 92] F. Dufaux, and M. Kunt, “Multigrid block matching motion estimation with an adaptive local mesh refinement”, In Proceedings SPIE Visual Commun., and Image Processing”, vol 1818, pp. 97-109, 1992.

[Ebrahimi 97] Touradj Ebrahimi, “MPEG-4 Video Verification Model Version 8”, no. N1796, MPEG Stockholm meeting, July 1997.

[Eckart 95] S. Eckart and C. Fogg “ISO/IEC MPEG-2 Software Video Codec”, In Proc. SPIE, vol 2419, pp. 100-109, February 1995.

[El-Rewini 94] H. El-Rewini, T. Lewis, and H. Ali, Task scheduling in parallel and distributed systems, Prentice Hall, Englewood Cliffs, N.J., 1994.

[Gabow 82] H. Gabow, “ An almost linear algorithm for two processor scheduling”, Journ. ACM, vol. 29, no. 3, pp. 766-780, July 1982.

[Gerasoulis 92] A.Gerasoulis and T.Yang, “ A comparison of clustering heuristics for scheduling DAGs on Multiprocessors”, J. Parallel and Distributed Computing, pp. 276-291, December 1992.

[Ghanbari 90] M. Ghanbari, “The cross-search algorithm for motion estimation”, IEEE Trans. Commun., vol 38, pp. 950-953, July 1990.

[Gong 94] K. Gong, and L. Rowe, “Parallel MPEG-1 Video Encoding”, September 1994 (http://bmrc.berkeley.edu/research/publications/1994/120/msreport-fm.html).

[Gong 95] K. Gong and L. Rowe, "A parallel implementation of an MPEG encoder: Faster than real time!", Proceedings of the SPIE Conference on Digital Video Compression: Algorithms and Technologies, pp. 407-418, San Jose, February 1995.

[Gove 94] R Gove, "The MVP: a highly integrated video compression chip", Proceedings of IEEE Data Compr. Conf. , pp. 215-224, Utah, March 1994

[Hamidzadeh 98] B. Hamidzadeh, and Y. Atif, "Dynamic scheduling of real time tasks by assignment", IEEE Concurrency, pp. 15-25, October-December 1998

[Hamosfakidis MPEG-4 97] A. Hamosfakidis, J. Cosmas, Y. Paker, and A. Pearmain, "Parallelisation of MPEG-4 video verification model encoder in inter/intra separate mode", Doc ISO/MPEG 1889, MPEG Bristol meeting, April 1997.

[Hamosfakidis 97] A. Hamosfakidis, and Y. Paker, "Survey of Multimedia Architectures", Technical Report, ACTS project AMPA, Copyright of AMPA Consortium 1997.

[Hamosfakidis 98] A. Hamosfakidis, Y. Paker, and J. Cosmas, "A study of concurrency in MPEG-4 video encoder", In Proceedings IEEE Multimedia Systems'98, Austin, pp. 204-208, June 1998.

[Hamosfakidis IEEE MS 99] A. Hamosfakidis, and Y. Paker, "Concurrency analysis for real-time MPEG-4 video encoding", ", In Proceedings IEEE Multimedia Systems'99, vol II, pp. 862-866, Florence June 1999.

[Hamosfakidis WIAMIS 99] A. Hamosfakidis and Y. Paker, “A fast motion prediction scheme for the MPEG-4 video encoder”, WIAMIS’99, pp. 125-129, May-June 1999.

[Hamosfakidis IWSNHC 99] A. Hamosfakidis, Y. Paker, “An MPEG-4 parallel shape adaptive motion estimation scheme”, Intern. Work. Synthet.-Natural. Hybrid Cod. and Three Dimen. Imag., IWSNHC3DI’99, pp. 41-45, September 1999.

[Y. He 98] Y.He, I. Ahmad and M. L. Liou, ``Real-time distributed and parallel processing for MPEG-4", Proc. of IEEE International Symposium on Circuits and Systems, Monterey, pp. 603-606, May 31 - June 3, 1998.

[Y. He 97] Y.He, I. Ahmad and M. L. Liou, ``An implementation of MPEG-4 video verification model encoder using parallel processing", Proc. of 3rd Asia-Pacific Conference on Communications (APCC '97), Sydney, December 1997.

[Z. He 97] Z. He and M.L.Liou, “A high performance fast search algorithm for block matching motion estimation”, IEEE Trans. Circuits Syst. Video Technol., vol. 7, no. 5, pp. 826-828, October 1997.

[Hou 97] C.Hou and K.Shin “Allocation of periodic task modules with precedence and deadlines constraints in distributed real-time systems” IEEE Trans. on Computers, vol. 46, no. 12, pp. 1338- 1355, December 1997.

[Hu 61] T. Hu, “ Parallel sequencing and assembly line problems”, Operations Research, vo.9, no.6, pp. 841-848, 1961.

[Huang 94] Z. Huang, Y. Takeuchi and H. Kunieda “Distributed load balancing schemes for parallel video encoding system”, IEICE Trans. Fundamental Electron. Commun. Comput. Science, vol. E77-A, no. 5, pp. 923-930, May 1994.

[Hwang 89] J. Hwang, Y. Chow, F. Anger, and C. Lee, “Scheduling precedence graphs in systems with interprocessor communication times”, SIAM Journal. Computing, vol. 18, no. 2, pp. 244-257, April 1989.

[IEEE CSVT 97] “Special Issue on MPEG-4”, IEEE Transactions on Circuits for Video Technology, vol. 7, no. 1, February 1997.

[ITUT-T 96] ITU-T Recommendation H.263: Video coding for low bitrate communication. The International Telecommunication Union, 1996.

[Kim 88] S.Kim, and J.Browne. “A general approach to mapping of parallel computations upon multiprocessor architectures”, Int’l Conferen. Parallel Proc., vol, 3, pp. 1-8, 1988.

[Koenen N3156, 99] Rob Koenen, “Overview of the MPEG-4 Standard”, ISO/IEC JTC1/SC29/WG11 N3156, December 1999.

[Koenen N2723, 99] Rob Koenen, “MPEG-4 Requirement Document”, Doc ISO/MPEG N2723, MPEG Seoul Meeting, March 1999.

[Koga 81] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, “Motion-compensated interframe coding for video conferencing”, in Proc. NTC81, New Orleans, LA, pp. C9.6.1-9.6.5, Nov 1981.

[Kleiman 95] S. Kleiman, D. Shah, and B. Smaalders, Programming with Threads, SunSoft Press A Prentice Hall Title, 1995.

[Kuhn 98] P. Kuhn, and W. Stechele, "Complexity analysis of the emerging MPEG-4 standard as a basis for VLSI implementation", Proceedings in SPIE Visual Communications and Image Processing, pp. 498-509, San Jose, 1998.

[Kwok 96] Y.Kwok and I.Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors", IEEE Trans. Parall. and Distrib. Sys., vol 7, no. 5, pp. 506-521, May 1996.

[X. Lee 96] X. Lee and Y.Q. Zhang, "A fast hierarchical motion-compensation scheme for video coding using block feature matching", IEEE Trans. Circuits Syst. Video Technol., vol. 6, no. 6, pp. 627-635, December 1996.

[Lehoczky 89] Lehoczky, J., L. Sha, and Y. Ding, " The rate-monotonic scheduling algorithm: exact characterisation and average case behaviour", IEEE Real-Time Systems Symposium, pp. 166-171, 1989.

[Lewis 95] B. Lewis, D. Berg, A Guide to Multithreading Programming, SunSoft Press A Prentice Hall Title, 1995.

[J. Li 93] J.Li, X. Lin, and Y. Wu, "Multiresolution tree architecture with its application in video sequence coding: A new result", In Proceedings SPIE Visual Commun., and Image Processing", vol. 2094, pp. 730-741, 1993.

[R. Li 94] R. Li, B. Zeng, and M. Liou, “A new three step search algorithm for block motion estimation", IEEE Trans. Circuits Syst. Video Technol., vol. 4, no. 4, pp. 438-442, August 1994.

[B. Liu 93] B. Liu and A. Zaccarin, “New fast algorithms for the estimation ob block motion vectors”, IEEE Trans. Circuits Syst. Video Technol., vol.3, pp.148-157, April 1993.

[L. Liu 96] L.Liu and E. Feig, “ A block based gradient descent search algorithm for block motion estimation in video coding”, IEEE Trans. Circuits Syst. Video Technol., vol. 6, no. 4, pp. 419-423, August 1996.

[Manacher 67] Manacher, G.K. “Production and Stabilization of real task schedulers”, Journal of ACM, pp. 841-848, July 1967

[Manimaran 98] G.Manimaran and C.Murthy, “An efficient dynamic scheduling algorithm for multiprocessor real-time systems”, IEEE Transactions on Parall. and Distrib. Sys., vol. 9, no. 3, pp. 312-319, March 1998.

[Mitchell 96] J. Mitchell, W. Pennebaker, C. Fogg, and D. LeGall, MPEG Video Compression Standard, Chapman & Hall, Digital Multimedia Standards Series, 1996.

[Moulin 95] P. Moulin, A.T. Ogielsky, G. Lilienfeld, J. W. Woods “Video signal processing and coding on data parallel computers”, Digital Signal Processing, vol. 5, pp. 118-129, April 1995.

[Musmann 85] H.G. Musmann, P.Pirsh, and H.-J. Grallert “Advances in picture coding”, Proceedings of IEEE, vol. 73, pp. 523-548, Apr. 1985

[Palis 96] M.Palis, J.Liou and S. Wei, " Task clustering and scheduling for distributed memory parallel architectures", IEEE Transactions on Parall. and Distrib. Sys., vol. 7, no. 1, pp. 46-55, January 1996.

[Papadimitriou 79] C. Papadimitriou, and M. Yannakakis, "Scheduling interval ordered tasks", SIAM Journal of Computing, vol. 8, pp. 405-409, 1979.

[Po 96] L.M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation", IEEE Trans. Circuits Syst. Video Technol., vol. 6, no. 3, pp. 313-317, June 1996.

[Puri 97] A. Puri, R.L. Schmidt and B.G. Haskell, "Improvements in DCT based video coding", Proc. SPIE Visual Commun. and Image Processing, San Jose, pp. 676-687, February 1997.

[Rabbani 91] M. Rabbani and P.W. Jones, Digital Image Compression Techniques, Tutorial Texts in Optical Engineering, Vol, TT 7, SPIE Optical Engineering Press, Bellingham, WA, 1991.

[Ramamritham 90] K.Ramamritham, J. Stankovic and P. Shiah, "Efficient scheduling algorithms for real-time multiprocessor systems, IEEE Transactions on Parall. and Distrib. Sys., vol. 1, no. 2, pp. 184-194, April 1990.

[Ramamritham 95] K. Ramamritham, " Allocating and sceduling of precedense-related periodic tasks", IEEE Transactions on Parall. and Distrib. Sys., vol. 6, no. 4, pp. 412-420, April 1995.

[Rec 601, 90] CCIR, Recommendation 601-2: Encoding parameters of digital television for studio. International Consultative Committee for Radio, 1990.

[Sarkar 89] V. Sarkar, Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors, MIT press, MA, 1989.

[Scafer 95] R. Scafer and T. Sikora, "Digital video coding standards and their role in video communications", Proceedings of IEEE, vol. 83, pp. 907-924, June 1995.

[Shen 96] K. Shen and E.J.Delp, "A Spatial-Temporal Approach for Real-Time MPEG Video Compression", Proceedings of the 25th International Conference on Parallel Processing, pp. 100-107, August 13-15, 1996.

[Sih 93] G.Sih and E.Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures", IEEE Trans. Parall., and Distrib. Sys., vol 4, no. 2, pp. 75-187, February 1993.

[Simulation 98] Simulation software "Text of IS14496-5 (MPEG-4 simulation software) final committee draft", Doc ISO/MPEG N2205, MPEG Tokyo Meeting, March 1998.

[Smoliar 94] S.W. Smoliar and H. Zhang, "Content-based video indexing and retrieval", IEEE Multimedia, pp. 62-72, Summer 1994.

[Srinivasan 85] R. Srinivasan, and K. Rao, "Predictive coding based on efficient motion estimation", IEEE Trans. on Commun. vol COM-33. Pp. 888-896, August 1985.

[Stein 92] D. Stein and D. Shah, “Implementing lightweight threads”, SunSoft Inc, USENIX, 1992.

[Steinmentz 94] R. Steinmentz, “Data Compression in Multimedia Systems”, Multimedia Systems, pp. 166-172, 1994.

[Tan 95] M. Tan, J.M. Siegel, and H.J.Siegel, “Parallel implementation of block-based motion vector estimation for video compression on the MasPar MP-1, and PASM”, International Conference on Parallel Processing, pp. 14-18, August 1995.

[Tekalp 95] A.M . Tekalp, Digital Video Processing. New Jersey: Prentice Hall PTR, 1995

[Touparis 99] A. Tourapis, O. C.Au and M.L. Liou, “Fast motion estimation using circular zonal search”, In Procs. of SPIE Visual Commun. and Image Processing, pp.1497-1505 January 1999.

[Ulman 75] J. Ulman, “NP-complete scheduling problems”, Journal Comput. and Sys. Sciences, vol 10, pp. 384-393, 1975.

[Uz 91] K.Uz, M. Vetterli and D. LeGall, “Interpolative multiresolution coding of advanced television with compatible subchannels “,”, IEEE Trans. Circuits Syst. Video Technol., vol. 1, pp. 86-99, March 1991.

[Watkinson 95] John Watkinson, Compression in Video and Audio, Focal Press, 1995.

[Wilkinson 96] J.H Wilkinson, “The optimal use of I, P, and B frames in MPEG-2 coding”, Proceedings in International Broadcast Convention, pp. 444- 449, IEE, 1996.

[Williams 90] S.A. Williams, “Programming models for parallel systems”, WILEY Series in Parallel Computing, 1990.

[Wu 90] M. Wu and D. Gajski “ Hypertool: A programming aid for Message-Passing Systems”, IEEE Trans. Parall. and Distrib. Sys., vol 1, no. 3, pp. 330-343, July 1990.

[Xie 92] K.Xie, L. Eycken, and A. Oosterlinck, “A new block-based motion estimation algorithm”, Signal Processing: Image Commun., vol. 4. pp. 507-517, 1992.

[Yu 94] Y. Yu and D. Anastassiou, “Software implementation of MPEG-2 video encoding using socket programming in LAN”, In Proc. of SPIE Digital Video Compression on Personal Computers: Algorithms and Technologies, vol. 2187, pp. 229-240, San Jose, February 1994.

[Zafar 91] S. Zafar, Y.Q. Zhang, and J.S. Baras, “Predictive block matching motion estimation for TV coding-Part I: Inter block prediction”, IEEE Trans. Broadcast, vol. 37, pp. 97-101, September 1991.

[Zafar 93] S. Zafar, Y. Q. Zhang, and B. Jabbari, “Multiscale video representation using multiresolution motion compensation and wavelet decomposition”, IEEE Journal. Select. Areas Commun., vol. 11, pp. 24-35, January 1993.

[Zhang 91] Y.Q.Zhang and S. Zafar, “Predictive block matching motion estimation for TV coding-Part II: Interframe prediction”, IEEE Trans. Broadcast, vol. 37, pp. 102-105, September 1991.

[Zhao 87] W.Zhao, K. Ramamritham, and J.Stankovic, “Scheduling tasks with resource requirements in hard real-time systems”, IEEE Trans. on Soft. Engineering, vol. SE-13, no. 5, pp. 564-577, May 1987.

[Zhu 97] S. Zhu and K. K. Ma, “A new diamond search algorithm for fast block matching motion estimation”, Intern. Conference on Information Commun. and Signal Processing, ICICS, pp. 292-317, Singapore, Sept 1997.

[11172-2, 93] ISO/IEC 11172-2, Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s. MPEG (Moving Pictures Expert Group), International Organisation for Standardisation, 1993 (MPEG-1 Video).

[13818-2, 94] ISO/IEC 13818-2, Generic coding of moving pictures and associated audio information. MPEG (Moving Pictures Expert Group), International Organisation for Standardisation, 1994. (MPEG-2 Video).

[14469-2, 99] ISO/IEC, ISO/IEC International Standard 14469-2, MPEG-4 Visual, Spring 1999.