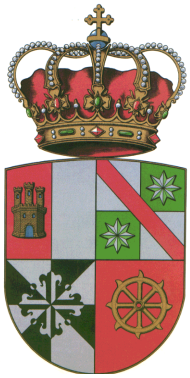
UNIVERSIDAD DE CASTILLA-LA MANCHA

Departamento de Sistemas Informáticos



2D and 3D H.264 Inter Prediction Algorithms for GPU–based heterogeneous architectures

Tesis Doctoral
presentada al Departamento de Sistemas Informáticos
de la Universidad de Castilla-La Mancha
para la obtención del título de
Doctor en Informática

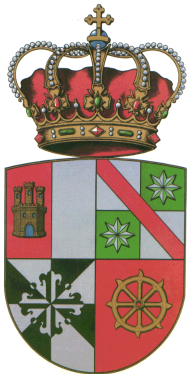
Presentada por:
Rafael Rodríguez Sánchez

Dirigida por:
Dr. D. José Luis Martínez Martínez
Dr. D. José Manuel Claver Iborra
Dr. D. José Luis Sánchez García

Albacete, febrero de 2013

UNIVERSITY OF CASTILLA-LA MANCHA

Computing Systems Department



2D and 3D H.264 Inter Prediction Algorithms for GPU–based heterogeneous architectures

A dissertation for the degree of Doctor of Philosophy in
Computer Science to be presented with due permission of the
Department of Computing Systems, for public examination and debate.

Author:

Rafael Rodríguez Sánchez

Advisors:

Dr. José Luis Martínez Martínez
Dr. José Manuel Claver Iborra
Dr. José Luis Sánchez García

Albacete, February 2013

This thesis was author typeset using LaTeX

Printed in the Kingdom of Spain

A mis padres y mi hermano
A Gema

II

Agradecimientos

La tesis finalmente está llegando a su fin y con estas líneas me gustaría dar las gracias a todas aquellas personas que de una manera u otra han hecho posible que sea así.

Para comenzar, me parece de recibo dar las gracias a los tutores de esta tesis, que tanto han trabajado en ella. Han hecho un gran esfuerzo y he aprendido infinidad de cosas de ellos. Sin su colaboración hoy no estaría escribiendo estas líneas. Gracias José Luis, gracias Pepelu y gracias José Manuel. También querría mencionar a Gerardo que ha colaborado durante su desarrollo.

Por otro lado, me gustaría dar las gracias al grupo de investigación RAAP que ha sido el que me ha dado la oportunidad de realizar esta tesis y a todos sus miembros por el buen ambiente que han creado. No voy a mencionar a nadie ya que son muchos y seguro que me faltaría alguien por nombrar. Gracias a los que todavía están y a los que ya se han ido a otros lugares.

También, quiero dar las gracias al Multimedia Lab de la universidad de Gante y muy especialmente a Jan, que me acogieron durante 3 meses en su laboratorio. Ha sido una bonita experiencia en la que he aprendido muchas cosas. En cuanto a mi estancia en Gante también me gustaría dar las gracias a Jaime y a Laura por los buenos ratos que hemos pasado; sin ellos se me habrían hecho los 3 meses mucho más largos.

Fuera del ámbito universitario, quiero dar las gracias a todos mis amigos y amigas por su apoyo y fidelidad no sólo durante estos 4 años, sino desde que os conocí. Sois muchos y tendría que usar muchas páginas para nombraros a todos. ¡Gracias!

Para ir terminando, qué voy a decir de mi familia, parte indispensable en mi vida y fuente de apoyo incondicional. Gran porcentaje de esta tesis es vuestro y no habría llegado hasta aquí sin vosotros. Gracias Papá, gracias Mamá y gracias Javi. De manera especial quiero mencionar a los que ya no están, seguro que desde su "pedacito" de cielo me han ayudado a que esto salga adelante.

Gema, siempre te dejo para el final, pero supongo que será porque eres de lo más importante en mi vida y me reservo las últimas palabras para ti. Gracias por tu cariño, por tu apoyo, por tu comprensión, por estar ahí ¡¡¡Gracias por todo!!!. También quiero dar las gracias a Amado, a Trini, a Marina y a la abuelita que también me habéis ayudado.

A todos vosotros y a todos los que me han ayudado ¡¡¡Gracias!!!

Resumen

En los últimos años, los contenidos multimedia han crecido de manera espectacular. Este crecimiento ha sido provocado por el rápido avance de las redes de telecomunicación y el rápido desarrollo de los dispositivos móviles y de visualización. Cada día, millones de vídeos son transmitidos por Internet o son emitidos desde una estación de televisión. Sin embargo, un vídeo en bruto consume demasiados recursos (espacio en disco y ancho de banda en las redes de comunicación), por lo que es necesario comprimirlo antes de ser transmitido o almacenado, reduciendo el consumo de recursos.

H.264/AVC es el último estándar de codificación de vídeo establecido, el cual es capaz de obtener mayor compresión que otros estándares previos para una misma calidad. Esta eficiencia en la codificación se logra mediante el uso de nuevas técnicas o mediante la mejora de algunas ya existentes. Esta mejora en la compresión se obtiene a base de aumentar la complejidad de los codificadores. Es cierto que el futuro estándar H.265/HEVC reemplazará a su predecesor (H.264/AVC), pero actualmente H.264/AVC es el estándar más utilizado.

Desde que H.264/AVC fue establecido en 2003, ha habido muchas propuestas que han tratado de reducir el coste computacional de los codificadores H.264/AVC y muchas de ellas se han centrado en el módulo de inter predicción. Estas propuestas en su mayoría han tratado de identificar ciertos modos que no es muy probable que fueran a ser utilizados para no tener que evaluarlos o definir algoritmos de búsqueda rápidos. Sin embargo, con la aparición de las GPUs se ha abierto una nueva vía para acelerar este módulo.

En el mercado existen varios fabricantes de GPUs, pero la mayor atención se la han llevado las desarrolladas por NVIDIA debido a su facilidad de programación. La principal característica de estas GPUs es disponer de una gran cantidad de elementos de proceso en un mismo chip a costa de una reducción significativa de otros elementos, por ejemplo la memoria cache. Estas GPUs son construidas siguiendo el modelo SIMD y son usadas normalmente como co–procesador.

En este escenario, donde es necesario reducir el tiempo de ejecución de los codificadores H.264/AVC, el uso de las GPUs es la solución ideal. Además, es importante mantener la eficiencia en la codificación. Para ello, las GPUs van a ser usadas conjuntamente con varios algoritmos diseñados para reusar la información de movimiento obtenida para ciertos modos con el fin de obtener la información de movimiento de otros modos. Esta tesis propone varias técnicas dependiendo del tipo de imágenes que se use en la codificación: se ha propuesto un algoritmo para imágenes tipo P, otro para tipo B y otro para vídeos 3D.

Los algoritmos propuestos en esta tesis reducen considerablemente los tiempos de ejecución de los algoritmos que se han tomado como referencia (búsqueda completa y UMHexagonS), mientras que proporcionan una buena eficiencia de codificación. Además, las propuestas son capaces de mejorar a las propuestas previas que hay en la literatura. Finalmente, se ha demostrado la eficiencia energética de las propuestas ya que, al usarlas, el codificador consume menos energía que usando los algoritmos de referencia.

Summary

In the last few years, multimedia content has grown dramatically. This growth has been driven by both the rapid advance in telecommunication networks and the quick development of portable devices and screens. Every day, millions of videos are transmitted over the Internet or are broadcast from a TV station. However, raw video consumes too many resources (storage capacity and network bandwidth), so video content is encoded before being transmitted or stored, reducing the resources used.

H.264/AVC is the latest–established video coding standard, and it achieves much higher compression than other previously existing video coding standards at the same quality. This compression efficiency is obtained by adopting some new or improved coding techniques. This improved performance is achieved by increasing the computational complexity of the encoders. It is true that the future H.265/HEVC standard will replace its predecessor (H.264/AVC), but at present the H.264/AVC standard is the most widely used.

Since the H.264/AVC standard was established in 2003, there have been many proposals that have sought to reduce the computational complexity of the H.264/AVC encoder and most of them focus on the inter prediction process. These proposals aim at discarding some coding modes before being checked, or at proposing fast search algorithms. However, with the emergence of GPUs, a new door has been opened for speeding up this quite complex process.

In the consumer market there are different GPU manufacturers, but most attention has been focused on the GPUs made by NVIDIA due to their programmability, the main feature of these GPUs being the large number of processing elements integrated into a single chip at the expense of a significant reduction in other components e.g. cache memory. These GPUs are built following the SIMD programming model and are normally used as coprocessors to assist the CPUs with massive data computations.

In this scenario, it is necessary to reduce the encoding time employed by an H.264/AVC encoder, and GPUs are the ideal solution. Moreover, the coding efficiency should be maintained. For this purpose, the parallel and powerful engine of NVIDIA GPUs will be used jointly with algorithms designed to reuse the motion information of the smallest inter prediction mode to obtain the motion information for the other higher inter prediction modes. This thesis proposes several techniques depending on the frame/slice types used to encode the video sequences, including an algorithm developed for P frames, an algorithm developed for B frames and an algorithm developed for 3D stereo video sequences.

The proposals presented in this thesis greatly reduce execution time when compared with the two reference algorithms tested (full search and UMHexagonS) and deliver an acceptable coding efficiency. The proposed algorithms obtain almost the same coding efficiency as the full search algorithm and surpass the coding efficiency of the UMHexagonS algorithm. Moreover, the proposed algorithms outperform previous approaches available in the literature. Additionally, the energy efficiency of the GPU solutions is also demonstrated, since the GPU–based H.264/AVC encoder consumes less energy than the reference H.264/AVC encoder.

Contents

List of Figures

List of Tables

Acronyms

3D *3–Dimensional.*

API *Application Programming Interface.*

ASO *Arbitrary Slice Order.*

AVC *Advance Video Coding.*

AVS *Audio Video Standard.*

CABAC *Context–Adaptive Binary Arithmetic Coding.*

CAVLC *Context–Adaptive Variable Length Coding.*

Cell BE *Cell Broadband Engine.*

CIF *Common Intermediate Format.*

COM *Component Object Model.*

CPU *Central Processing Unit.*

CUDA *Compute Unified Device Architecture.*

DBS *Diamond–Based Search.*

DCT *Discrete Cosine Transform.*

DDR *Double Data Rate.*

DE *Disparity Estimation.*

DRAM *Dynamic Random Access Memory.*

DTV *Digital TeleVision.*

DVD *Digital Versatile Disc.*

DVFS *Dynamic Voltage and Frequency Scaling.*

EPZS *Enhanced Predictive Zonal Search.*

FIR *Finite Impulse Response.*

FME *Fractional Motion Estimation.*

FMFSA *Fast Multi–reference Frame Selection Algorithm.*

FMO *Flexible Macroblock Ordering.*

FPGA *Field Programmable Gate Array.*

FRExt *Fidelity Range Extension.*

FS *Full Search.*

GOP *Groups Of Picture.*

GPGPU *General–Purpose computing on Graphics Processing Unit.*

GPU *Graphic Processing Unit.*

HBS *Hexagon–Based Search.*

HD *High Definition.*

HDT *Hadamard Transform.*

HDTV *High Definition TeleVision.*

HEVC *High Efficiency Video Coding.*

IME *Integer Motion Estimation.*

ISO/IEC *International Organization for Standardization / International Electrotechnical Commission.*

ITU–T *International Telecommunication Union–Telecommunication.*

JVT *Joint Video Team.*

LDPS *Line Diamond Parallel Search.*

MAE *Mean of Absolute Errors.*

MB *MacroBlock*.

MBAFF *Macroblock Adaptive Frame Field Coding*.

MC *Motion Compensation*.

ME *Motion Estimation*.

MET *Mode–correlation–based early Termination*.

MPEG *Moving Picture Experts Group*.

MSE *Mean Square Error*.

MV *Motion Vector*.

MVC *Multi View Coding*.

MVP *Motion Vector Predictor*.

NAL *Network Abstraction Layer*.

NNS *Nearest Neighbour Search*.

PC *Personal Computer*.

PCI *Peripheral Component Interconnect*.

PE *Processing Element*.

PHODS *Parallel Hierarchical One-Dimensional Search*.

PSNR *Peak Signal Noise Ratio*.

PSU *Power Supply Unit*.

QCIF *Quarter Common Intermediate Format*.

QP *Quantization Parameter*.

QWVGA *Quarter Wide Video Graphics Array*.

RAM *Random Access Memory*.

RD *Rate Distortion*.

RMS *Root Mean Square*.

SAD *Sum of Absolute Differences*.

SADT *Sum of Absolute Differences of the Transformed residual data.*

SAE *Sum of Absolute Errors.*

SDK *Software Development Kit.*

SIMD *Single Instruction Multiple Data.*

SLI *Scalable Link Interface.*

smpUMHexagonS *Simplified Unsymmetrical Multi-Hexagon Search.*

SP *Stream Processor.*

SPEC *Standard Performance Evaluation Corporation.*

SVC *Scalable Video Coding.*

TR *Time Reduction.*

TSS *Three Step Search.*

TV *TeleVision.*

UMHexagonS *Unsymmetrical Multi-Hexagon Search.*

USB *Universal Serial Bus.*

VCEG *Video Coding Experts Group.*

VGA *Video Graphics Array.*

VLC *Variable Length Code.*

VLSI *Very Large Scale Integration.*

WVGA *Wide Video Graphics Array.*

Chapter 1

Introduction

IN this chapter, we introduce the main reasons that have motivated this work. Then, we define the objectives we wish to accomplish in the course of this PhD, and finally we outline the organization of this thesis.

1.1. Motivation

In recent years, we have witnessed a particularly rapid advance in telecommunications, caused by the rapid development of the Internet, telephony networks, mobile devices (smart–phones, tablets and laptops) and screens. This has led to new market opportunities for companies which provide all kinds of information to customers spread across the globe using these networks.

Multimedia traffic in general, and digital video in particular, has increased spectacularly in recent years. This video traffic, both on the Internet and on mobile networks, increases daily. Nowadays, video coding is the main technology behind a wide range of applications, which include video conferencing, video streaming, and more [Wiegand et al. 03] [Sullivan et al. 06].

Moreover, *Digital TeleVision* (DTV), whether terrestrial or satellite, has been imposed on the domestic *TeleVision* (TV) market. At present, users have the possibility to demand the type of content they want to view at any time. *High Definition TeleVision* (HDTV) is the latest trend regarding DTV, representing a revolution in the domestic TV market.

HDTV offers qualities close to those that can be seen in the best cinema, implying an increment in the resolution of domestic TVs. The images for old TVs were transmitted with a resolution of 720x576 pixels, while the images for recent HDTVs are transmitted with a resolution of up to 1920x1080 pixels. That means that the image resolution, and as a consequence the amount of information being transmitted or stored, is increased by a factor of 5. If we take into account that one pixel is represented using at least 3 bytes, it implies that one *High Definition* (HD) image is represented using nearly 6 Mbytes.

Moreover, knowing that a video sequence is sampled at least 25 times per second, this implies that a one hour video requires over 520 Gbytes to be stored, or 150 Mbytes per second. Additionally, in recent years *3–Dimensional* (3D) videos have emerged in the industry in order to provide the sensation of immersion in the video scene. Usually, a 3D video is sampled using two different viewpoints (stereo), doubling again the amount of information being transmitted or stored.

This huge amount of information must be transferred and processed in real time for an appropriate visualization of the video content. However, such an amount of information cannot be transmitted by existing interconnection networks or handled by existing devices. This is where video compression standards become extremely useful. Companies use these standards to store the information in a compressed form, minimizing the amount of information sent to the clients.

A decade ago, MPEG–2 [ISO/IEC 99] was the most widely used video compression standard. It was able to offer very good quality with an acceptable level of resource utilization (network bandwidth or disc storage). However, companies demanded a new video compression standard able to maintain the quality offered by MPEG–2, but able to further reduce resource consumption. This new standard is the H.264 or MPEG–4 part 10 /*Advance Video Coding* (AVC) video compression standard [ITU–T 03] [ISO/IEC 03] and was established in 2003. It is true that the future H.265/*High Efficiency Video Coding* (HEVC) [ITU–T and ISO/IEC 12] standard will replace its predecessor (H.264/AVC), but at present most architectures and video coding solutions are implemented using the H.264/AVC standard.

The main purpose of H.264/AVC is to offer a good standard of quality and to be able to considerably reduce the output bit rate of the encoded sequences, compared with previous standards, with a view to being used in a variety of applications such as *Digital Versatile Disc* (DVD), video-streaming, HDTV, etc., while exhibiting a substantially improved definition of quality and image. However, these improved capabilities are obtained at the expense of an increment in the computational complexity of the encoder. H.264/AVC adopts many video coding techniques, such as multiple reference frames, weighted prediction, a de-blocking filter, variable block size and quarter-pixel precision for motion compensation, causing encoding time to become very high for applications that demand real-time encoding.

Fortunately, the high computational cost introduced by the H.264/AVC encoders can be efficiently reduced by adapting the sequential source code to parallel architectures. In recent years, different high performance computing platforms have been used for this purpose, such as clusters or supercomputers. However, these parallel architectures are usually very expensive and their computing power is considerably above what an H.264/AVC encoder requires. In this scenario, it is necessary to find other cheaper and smaller alternatives.

It is in the last five years when a small and cheap device with high computing capabilities has been introduced in high performance computing. These devices are known as *Graphic Processing Unit*s (GPUs) and are normally incorporated in modern graphics cards. As a consequence, GPUs are an interesting and low cost alternative to accelerate

some processes that are part of the video encoding process. These devices, designed and traditionally used for graphics applications, have recently been incorporating many processing elements that can efficiently exploit the data parallelism inherent in many applications. The progress of GPUs is now the focus of a great deal of attention and GPUs are moving from being exclusively used in graphics applications to being used in what is now called *General–Purpose computing on Graphics Processing Unit* (GPGPU) [GPGPU 07]. They have changed from fixed pipelines to programmable pipelines. GPUs consist of hundreds of highly decoupled processing cores that are able to achieve immense parallel computing performance. Moreover, these GPUs provide high memory bus widths, high speed memory chips, and high processor clock speeds. Graphics cards deliver specifications never seen before.

GPUs are highly parallel and are normally used as coprocessors to assist the *Central Processing Unit* (CPU) with massive data computations. CPUs and GPUs have different instruction set architectures, forming what it is known as a heterogeneous computing platform. Now, the two together can be used to accelerate various numerical and signal processing applications [Shen et al. 05] [Krüger and Westermann 03], among others.

In the consumer market different graphics card manufacturers, such as NVIDIA and ATI/AMD, have proposed their different GPU implementations. However, most of the attention has been focused on the GPUs made by NVIDIA, since they have developed an architecture called *Compute Unified Device Architecture* (CUDA) [NVidia 12], which can be easily programmed using CUDA C. This programming language is a C–based high level programming language designed to maintain a low learning curve for programmers familiar with standard C. The main feature of these NVIDIA GPUs is a large number of processing elements integrated into a single chip at the expense of a significant reduction in cache memory. Each core executes the same instruction at the same clock cycle but on different data. GPUs also have an external *Dynamic Random Access Memory* (DRAM) memory which can be classified depending on its access mode.

In this scenario, the goal of this thesis is to reduce the computational complexity of an H.264/AVC encoder as much as possible, and it is known that most of this complexity is intended to remove the redundancies of the video sequences. One of these redundancies is temporal redundancy, which is removed by the inter prediction module of an H.264/AVC encoder. This module iteratively checks how movement takes place within a search area. To reduce H.264/AVC encoder complexity, an NVIDIA GPU will be used, but also a technique to reduce computation. This technique uses the motion information of some encoding modes to obtain the motion information of other encoding modes. Moreover, it is desirable for the encoding efficiency of H.264/AVC to be maintained as far as possible, so the encoded bit stream will be evaluated in terms of image quality and of the number of bits required to store or transmit it. Additionally, the energy efficiency of the proposed H.264/AVC encoder will be evaluated, since this is a strong criticism that has been made of GPU solutions.

There are many potential applications of this thesis. Nowadays, GPUs are available in a wide variety of environments, ranging from small and cheap personal computers, to

large and expensive supercomputers [TOP500 12]. NVIDIA manufactures different GPU solutions to satisfy the requirements of these different environments. As a consequence, the solutions proposed in this thesis can be used by personal computers when performing a video conference, by more powerful servers dedicated to video streaming or by large supercomputers in video storage industry.

1.2. Objectives

As was mentioned at the end of the previous section, the main goal of this thesis is to reduce the encoding complexity on an H.264/AVC encoder, taking into account that the proposed H.264/AVC encoder should be able to maintain the encoding efficiency of the reference H.264/AVC encoder as far as possible. The main objectives are summarized as follows:

- To analyse the different GPU architectures available on the consumer market to find the most suitable device to develop a GPU–based H.264/AVC encoder. Different aspects are important for this, such as their hardware design, their performance capabilities (memory bandwidth and peak performance) or their programming facilities.
- To carry out an analysis of H.264/AVC. The main aim of this analysis is to check if the most suitable module for being accelerated using GPUs is the inter prediction module, since a priori it is known that it is the most time consuming module. Different aspects are important for this, such as the data dependencies, the execution time or to identify if its source code is suitable for the programming model of a GPU (e.g. branch instructions are not desirable).
- To review the most recent state of the art methodologies developed for both GPU computing and H.264/AVC video coding. This review provides the starting point for this thesis and an overview of the most recent and prominent related proposals. This thesis aims to obtain better results than the ones presented in the related articles.
- To propose several GPU–based inter prediction algorithms. These algorithms are aimed at reducing H.264/AVC encoder execution time, but also at maintaining the coding efficiency of the reference algorithms. These algorithms are:
  - An inter prediction algorithm developed for P frames.
  - An inter prediction algorithm developed for B frames.
  - An inter prediction algorithm developed for 3D stereo video sequences, using both P and B frames.
- To evaluate the proposed algorithms using a wide variety of tests. Each algorithm is tested using different video resolutions and is analysed in terms of execution time, coding efficiency and energy consumption. Moreover, the different algorithms are

evaluated using different H.264/AVC profiles, since they are intended for the requirements of different applications.

- To compare the results obtained by using the proposed algorithms with the ones reported in the most recent and prominent related proposals, aiming to improve upon their results.

1.3. Organization of this document

The thesis is organized into seven chapters and one appendix. In what follows, a brief introduction of each is provided:

- *Chapter 1.* The first chapter introduces the thesis. This chapter provides the motivation, objectives and organization of the thesis.
- *Chapter 2.* This chapter provides a review of the video compression standards used in this thesis (H.264/AVC and its extension, *Multi View Coding* (MVC)). Moreover, this chapter provides an introduction to GPU computing.
- *Chapter 3.* This chapter summarizes the most recent and prominent work related to the scope of this thesis.
- *Chapter 4.* This chapter presents the inter prediction algorithm developed for P frames. The algorithm is based on reusing the motion information of some encoding modes and is optimized for GPU execution. Then, the proposal is evaluated, including a comparison with the most recent related proposals where possible.
- *Chapter 5.* This chapter presents the inter prediction algorithm developed for B frames. This algorithm is an extension of the one developed for P frames, and is also optimized for GPU execution. At the end, the proposal is evaluated, including a comparison with the most recent related proposals where possible.
- *Chapter 6.* This chapter presents the final proposal of this thesis. The proposal is a modification to the algorithms presented in previous chapters, developed to mitigate both the temporal and the inter–view dependences inherent in 3D video sequences. Finally, the proposal is evaluated.
- *Chapter 7.* This chapter presents the main conclusions derived from this thesis, and provides future directions in which the work presented in this thesis can be extended. Finally, the publications derived from this thesis are listed.
- *Appendix A.* Provides extra results for this thesis. The H.264/AVC encoder is configured to simulate, as far as possible, the encoding condition of the future H.265/HEVC standard.

Chapter 2

Technical Background

SINCE H.264/AVC and its extension, MVC, are involved in the development of this thesis, they will be described in this chapter. Moreover, this thesis has been developed using modern GPUs, and these are also described. An extension of the contents of this chapter can be found in [Richardson 04], in [Richardson 10] and in [NVidia 12].

2.1. H.264/AVC

In the past thirty years, there has been a marked increase in multimedia content, encouraged by the quick deployment of the Internet, mobile networks and mobile devices. However, to enable the storage and transmission of this multimedia content it is necessary to employ compression techniques. Nowadays, image and video compression are very active research fields.

During these thirty years many codecs have been standardized. This standardization specifies the syntax of the coded information and how to decode the information in order to reconstruct it. In this sense, there are two standardization organizations established exclusively to develop video coding standards, the *International Telecommunication Union–Telecommunication* (ITU–T) *Video Coding Experts Group* (VCEG) and the *International Organization for Standardization / International Electrotechnical Commission* (ISO/IEC) *Moving Picture Experts Group* (MPEG). Both organizations have developed their own video codec standards separately, but in some cases they have worked together to create common standards. In order to establish the specifications of these common standards, the *Joint Video Team* (JVT) was created with video experts from both organizations. Figure 2.1 shows the evolution of the video coding standards developed by these organizations.

H.264 [ITU–T 03] or MPEG–4 part 10 *Advance Video Coding* (AVC) [ISO/IEC 03] is a compression video standard developed jointly by the ITU–T VCEG and the ISO/IEC MPEG. The first preliminary H.264/AVC implementation was released in 2003 [JVT 11].

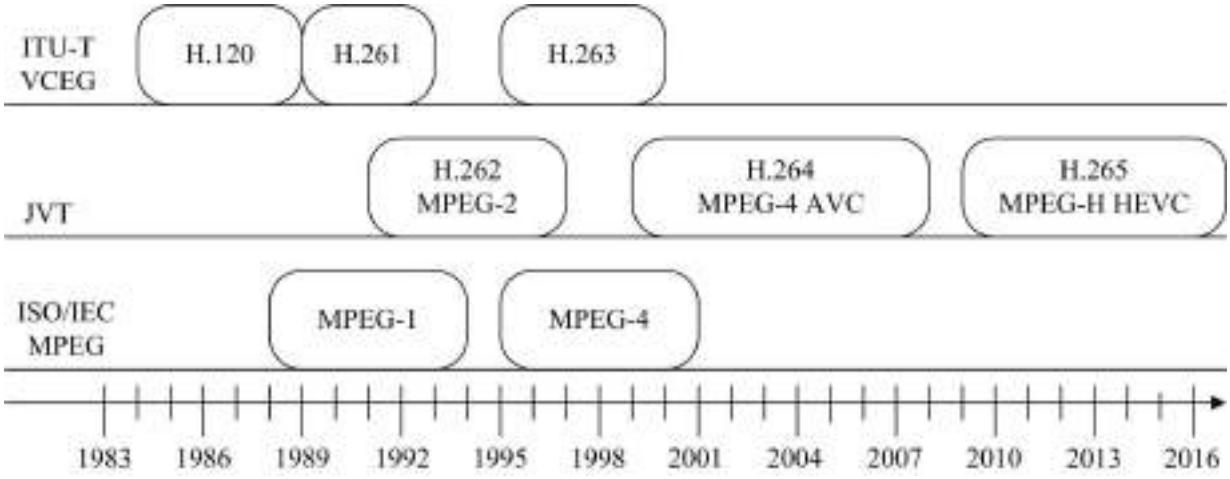


Figure 2.1: Video coding standards.

H.264/AVC emerged with the objective of creating a standard able to provide good video quality for the encoded video sequence, as well as being able to greatly reduce the bit stream produced by an H.264/AVC encoder, when compared with previous standards. Secondly, it was intended to achieve this goal without increasing the complexity of the encoder. Finally, as an additional goal, it was intended that H.264/AVC could be used in a wide variety of applications such as video–telephony, video–conferencing, video–streaming, HDTV, DVD storage and digital cinema, among others.

H.264/AVC represents a significant advance in terms of quality and distortion compared with the commercial standards previously most in use, such as MPEG–2 [ISO/IEC 99]. In efficiency terms, H.264/AVC increases the compression achieved by previous standards (up to 200%), while also increasing the image definition and quality of the encoded video sequence [Wiegand et al. 03].

Subsequently to the definition of the H.264/AVC standard, some modifications or extensions were proposed. In September 2004, the *Fidelity Range Extension* (FREXt) [Sullivan et al. 04] was introduced. This extension makes it possible to use 10 and 12 high definition bit samples for the sampling formats YUV 4:2:2 and 4:4:4, to use Intra–prediction using 8x8 blocks, and to support additional color spaces.

In 2007, the *Scalable Video Coding* (SVC) [ITU–T and ISO/IEC 07] extension was introduced. SVC supports spatial, temporal and quality scalability, which allows the adaptation of the encoded bit stream to the transmission conditions, or to the application's requirements.

In 2009, the *Multi View Coding* (MVC) [ITU–T and ISO/IEC 09] extension was introduced. MVC enables efficient video encoding of video sequences obtained by using more than one camera in a single bit stream. In MVC, there is a large quantity of inter–view dependencies which can be exploited in order to mitigate the redundancy between the different viewpoints (cameras). The key of MVC is to combine temporal, spatial and inter–view predictions.

Currently, a new standard is under development. Just as happened with H.264/AVC, the ITU–T VCEG and ISO/IEC MPEG are working together on the development of this new standard. The name for this new standard is H.265, or MPEG–H HEVC [ITU–T and ISO/IEC 12].

2.1.1. H.264/AVC overview

Digital video usually represents a visual scene recorded from the real world, sampled at regular intervals of time. The visual scene may be sampled in two different ways: sampling complete frames, which is known as *progressive video*, or sampling interlaced fields, which is known as *interlaced video*. In an interlaced video sequence, half of the data in a frame belongs to a field (odd lines within a frame) and the other half of the data to a different field (even lines within a frame), each representing half of the information in a complete frame, doubling the perceived frame rate. This sampling creates the video sequence, which is the video stream.

The complete video sequence is organized in *Groups Of Picture*s (GOPs), which specify the order in which the different predictions are arranged. Each frame may be further divided into one or more slices, and each slice is encoded depending on its type. In H.264/AVC there are 5 slice types, but these slice types are restricted to the profile used to encode the video sequences (further description of the H.264/AVC profiles can be found in Section 2.1.9).

Table 2.1 shows a general description of each slice type available in H.264/AVC, as well as the profile on which each slice type can be used. This division into slices makes it possible to control the complexity of the encoders and decoders. Therefore, the time

Table 2.1: H.264/AVC slice modes.

Slice type	Description	Profile(s)
I (Intra)	Contains only I blocks (each block is predicted from previously coded data within the same slice)	All
P (Predicted)	Contains P blocks (each P block is predicted from previous frames) and/or I blocks	All
B (Bi–Predictive)	Contains B blocks (each B block is predicted from previous and/or future frames) and/or I blocks	Extended, Main
SP (Switching P)	Facilitates switching between coded streams; Contains P and/or I blocks	Extended
SI (Switching I)	Facilitates switching between coded streams; Contains SI blocks (a special type of intra coded blocks)	Extended

employed for encoding video sequences may vary considerably, as well as the compression ratio, error resilience, and the quality of the encoded video sequence.

The frames may be acquired using different resolutions (width and height of the frames), such as full–HD 1920x1080 pixels, HD 1280x720 pixels, *Video Graphics Array* (VGA) 640x480 pixels, *Common Intermediate Format* (CIF) 352x288 pixels, or *Quarter Common Intermediate Format* (QCIF) 176x144 pixels. Additionally, the input frames are divided into equal–size blocks, called *MacroBlock*s (MBs). The size of an MB is 16x16 pixels, and most of the mechanisms applied by the H.264/AVC encoders and decoders are applied at this level. Figure 2.2 summarizes the layer structure of an H.264/AVC video sequence.

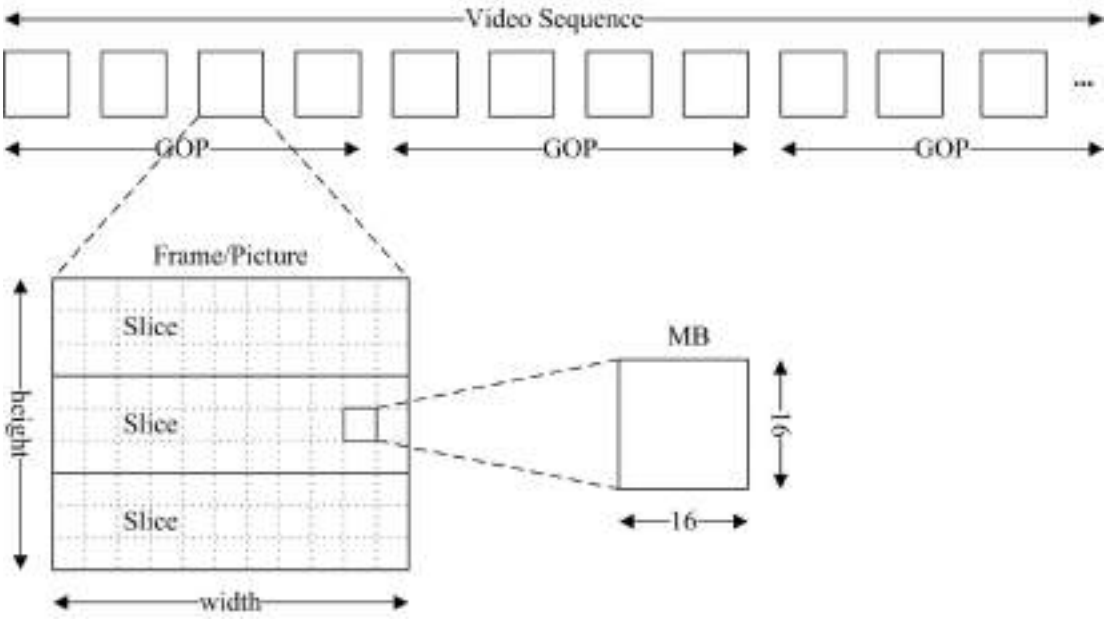


Figure 2.2: Layer structure of an H.264/AVC video sequence.

Finally, each MB is typically represented using three components: Y, U, and V; Y is the luminance component; U and V are the chrominance components. However, it is known that the human visual perception system is more sensitive to the luminance component than to the chrominance ones. Therefore, it is usual to sub–sample the chrominance components. The most common sampling formats are cited below:

- 4:4:4. Luminance and chrominances are sampled using the same frequency.
- 4:2:2. Chrominances are sampled halving the frequency in the horizontal direction.
- 4:2:0. Chrominances are sampled halving the frequency in both directions (horizontal and vertical).
- 4:1:1. Chrominances are sampled quartering the frequency in the horizontal direction.
- 4:0:0. No chrominances used, only luminance is used.

H.264/AVC employs a hybrid block–based compression technique, which is able to eliminate the temporal (inter prediction) and the spatial (intra prediction) redundancies in video sequences. In H.264/AVC, an MB will be either inter predicted, or intra predicted, or skipped. If the MB is inter predicted, a displacement vector referring to a previously encoded frame and its residual are obtained and stored. If an MB is intra predicted, the MB is encoded referring to neighbouring blocks in the same frame. Finally, if the MB is skipped, no further mechanisms are used to encode this MB.

The residual of a prediction is the difference between the current block and the reference block. Before being stored or transmitted, the residual is transformed, quantified and encoded using an entropy encoder. Figure 2.3 shows the block diagram of the mechanisms previously described, which is the basic block diagram of a generic H.264/AVC encoder.

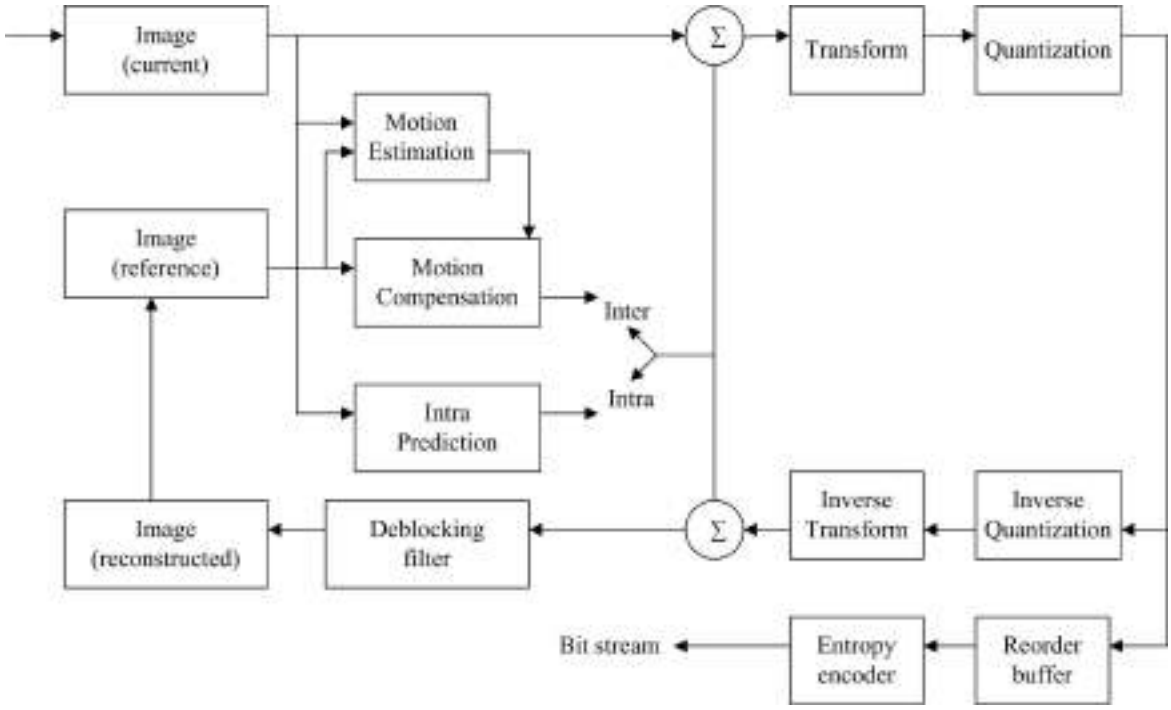


Figure 2.3: H.264/AVC encoder block diagram.

An H.264/AVC encoder has two paths, known as *forward path* and *reconstructed path*. Both paths start with the same input frame but their ends are different. The forward path finishes with the encoded bit stream, which will be stored or transmitted. The reconstructed path finishes with the reconstructed frame, which will be used as reference frame, for further frames.

On the other hand, a generic H.264/AVC decoder only has a forward path. This path starts with the encoded bit stream and finishes with the reconstructed frame, which will be stored or reproduced. Note that the forward path in the decoder is quite similar to the reconstructed path of the encoder. Figure 2.4 shows the block diagram of a generic H.264/AVC decoder.

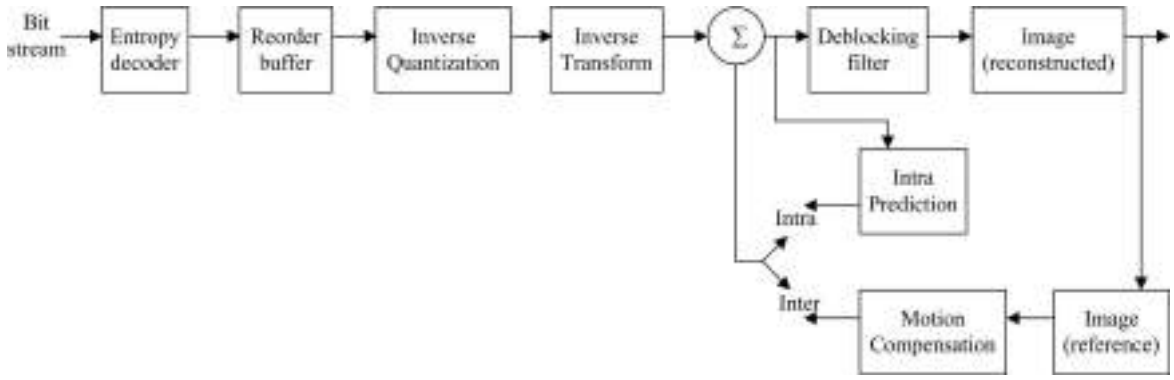


Figure 2.4: H.264/AVC decoder block diagram.

Originally, the H.264/AVC standard defined a set of three profiles (Baseline, Main, and Extended), each of them supporting a set of coding mechanisms and specifying what is required of an encoder and decoder that follows each profile. Later, in 2004, with the definition of the *Fidelity Range Extension* (FREXt) [Sullivan et al. 04], four new profiles were added (High, High10, High4:2:2, and High4:4:4). Finally, at the beginning of 2009 with the last standard update, more profiles were added to make a total of twelve profiles. Additionally, two new extensions were added to the standard (SVC and MVC extensions), adding extra profiles. Two profiles were added with the MVC extension: the Stereo High profile and the Multi–View High profile, which provide support for MVC.

2.1.2. Inter Prediction

Motion Estimation (ME) and *Motion Compensation* (MC) are the mechanisms responsible for eliminating the temporal redundancies between independent frames in a video sequence. ME and MC look for a pattern indicating how the block movement occurs, obtaining a *Motion Vector* (MV). Both mechanisms make up the module known as *inter prediction* and can be identified in the block diagram of a generic H.264/AVC encoder (Figure 2.3). However, only the MC module can be identified in the block diagram of a generic H.264/AVC decoder (Figure 2.4).

Each block, into which a frame can be divided, is compared with previous and future frames (reference frames), inside a search range. This mechanism is known as ME and obtains the MV which provides the best prediction for the current block (a region that minimises the differences between the current block and the predicted block). ME estimates each block by using a block located in a previous frame (P and B slices/frames), by using a block located in a future frame (B slices/frames), or by using a combination of both (B slices/frames) which is known as *Bi-directional prediction*. A graphical description of the ME mechanism can be found in Figure 2.5, where the chosen region (which is a sub–set of the search area) is subtracted from the current block to obtain the residual. The residual block and the associated MV are stored or transmitted.

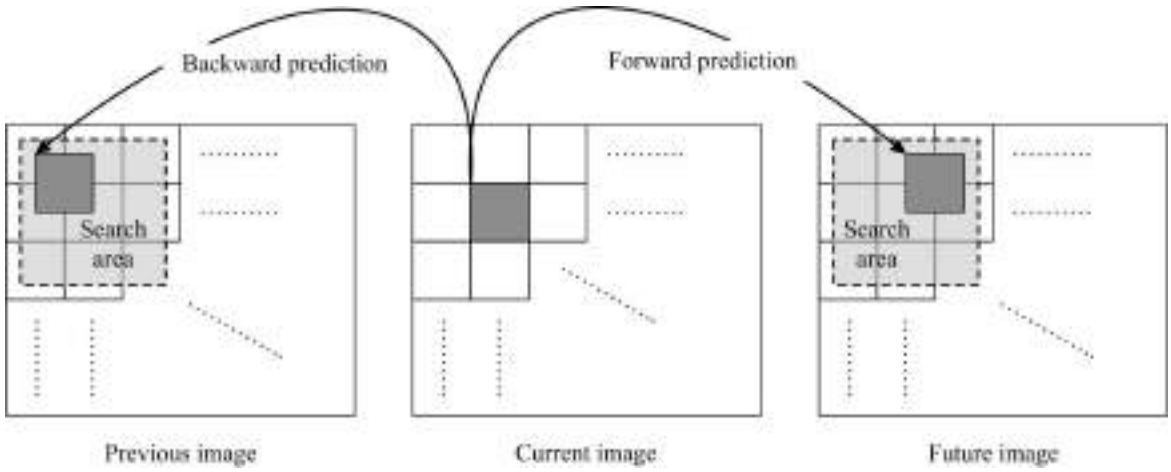


Figure 2.5: Motion Estimation mechanism.

The H.264/AVC standard supports motion compensation block sizes ranging from 16x16 to 4x4, with many options available between them. This procedure is known as the *tree structured motion compensation algorithm*, and it is able to search for the optimal matching block by close prediction (where the block size is variable). These smaller blocks, when compared with the blocks used by previous standards, are able to contain and isolate the movement, providing greater flexibility in the MB partitions, as well as more precise MVs.

H.264/AVC inter prediction supports motion compensation block sizes of 16x16, 16x8, 8x16, and 8x8, where each of the sub-divided blocks is an MB partition. Additionally, if the 8x8 mode is chosen, each of the four 8x8 partitions within the MB may be further split in four ways: 8x8, 8x4, 4x8, and 4x4, which are known as *MB sub-partitions*. ME is carried out for all MB partitions and sub-partitions. Figure 2.6 shows the motion compensation block sizes supported by the H.264/AVC standard.

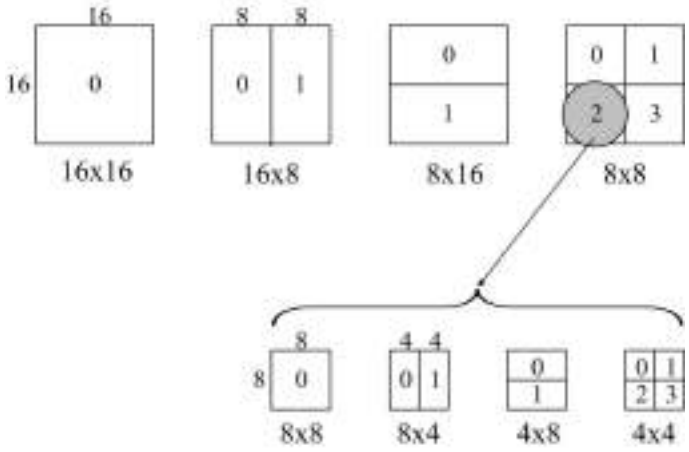


Figure 2.6: H.264 MB partitions and MB sub–partitions.

Note that an MB belonging to a slice/frame marked as inter predicted (P or B slices /frames) may be inter predicted, intra predicted, skipped or encoded using the DIRECT mode (only available in B slices/frames). The decision is made by analysing the encoding cost. An MB belonging to a slice/frame marked as intra predicted may only be intra predicted.

The mechanisms described above are applied for the luminance component, but can also be applied for the chrominance components. Remember that the chrominance components can be sampled using different frequencies from the ones used for the luminance component, so the block sizes for the chrominances may vary. For example, if the 4:2:0 sampling format is used, the block sizes for the chrominance components are 8x8, 8x4, 4x8, 4x4, 4x2, 2x4, and 2x2.

In order to further improve compression, the H.264 /AVC standard is based on the idea that the best match can be found in a region offset from the current MB by an integer number of pixels, which is known as *Integer Motion Estimation* (IME). However, for many MBs, a better match can be found by searching a region interpolated to sub–pixel accuracy. This is known as *sub–pixel ME* or *Fractional Motion Estimation* (FME), and it supports quarter–pixel accuracy.

The input frames in a video sequence are sampled at pixel level, so it is necessary to interpolate the frames in order to obtain the frames with sub–pixel accuracy. The sub–pixels with half–pixel accuracy are obtained by means of a 6–tap *Finite Impulse Response* (FIR) filter with weights (1/32, -5/32, 5/8, 5/8, -5/32, 1/32). For example, half–pixel a in Figure 2.7a is calculated from six horizontal integer–pixels, and half–pixel b in Figure 2.7a is calculated from six vertical integer–pixels:

$$a = round((G - 5H + 20C + 20I - 5J + K)/32), \tag{2.1}$$

$$b = round((A - 5B + 20C + 20D - 5E + F)/32), \tag{2.2}$$

Note that integer–pixels are denoted in Figure 2.7 using upper case letters, half–pixels using lower case letters, and quarter–pixels using numbers.

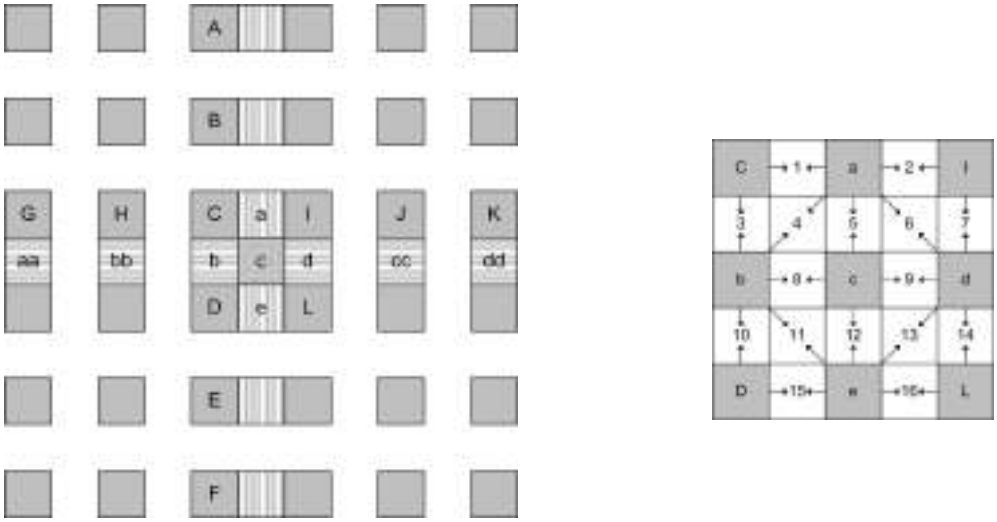
When all of the samples horizontally and vertically adjacent to integer–pixels have been calculated, the remaining half–pixels are calculated using the previously calculated half–pixels (vertically or horizontally adjacent). For example, half–pixel c in Figure 2.7a is calculated from six horizontal half–pixels:

$$c = round((aa - 5bb + 20b + 20d - 5cc + dd)/32), \tag{2.3}$$

Once all the half–pixels are available, the sub–pixels with quarter–pixel accuracy are obtained by linear interpolation. Quarter–pixels horizontally or vertically adjacent to half– or integer–pixels are linearly interpolated between these adjacent pixels. For example, in Figure 2.7b quarter–pixels 1 and 3 are calculated as:

$$1 = round((C + a)/2) \quad //horizontally, \tag{2.4}$$

$$3 = round((C + b)/2) \quad //vertically, \tag{2.5}$$



(a) Half–pixel generation

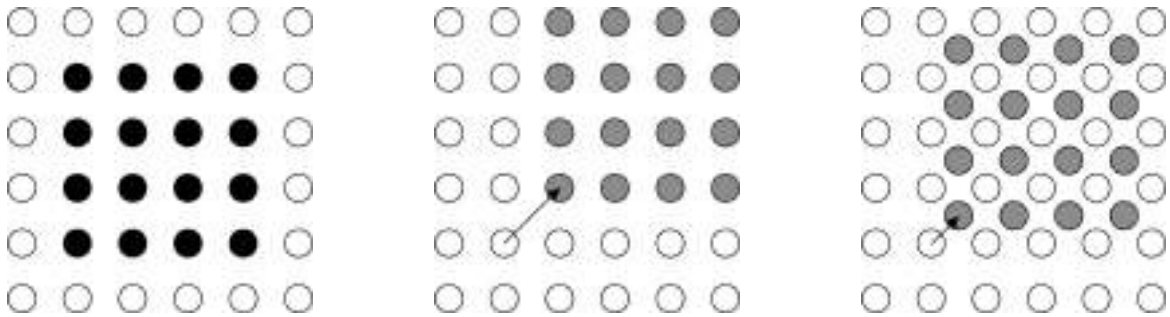
(b) Quarter–pixel generation

Figure 2.7: Interpolation of the luminance component.

Finally, the remaining quarter–pixels are linearly interpolated between a pair of diagonally opposite half–pixels. For example, in Figure 2.7b the quarter–pixel 4 is calculated as:

$$4 = round((a + b)/2), \tag{2.6}$$

Figure 2.8 shows two prediction examples. In Figure 2.8a, a 4x4 block in the current frame is predicted from a neighbouring region to the current position in the reference frame. If the two components of the MV are integers, an integer prediction was performed, MV (1,-1) in Figure 2.8b. If one or both components are not integers, a sub–pixel prediction was performed, MV (0.5,-0.5) in Figure 2.8c. For each MB partition and MB sub–partition described above, the ME process with full–pixel accuracy is applied (IME) and then, using the best full–pixel MV obtained as a starting point, the sub–pixel ME is applied (FME). FME can be viewed as a refinement of IME.



(a) Current block

(b) MV (1, -1)

(c) MV (0.5, -0.5)

Figure 2.8: 4x4 sample of integer and sub–pixel prediction.

Encoding one MV for each MB partition can increase the number of bits required to encode a frame, specially if small partition sizes are chosen. However, it is known that MVs from neighbouring partitions are often highly correlated and each MV can be predicted using the MVs of neighbouring partitions. Therefore, a *Motion Vector Predictor* (MVP) can be calculated, and the differences between the MV obtained and the MVP calculated are encoded. The MVP forming method depends on the availability of nearby MVs and on the partition size.

Figure 2.9 shows an MB and its neighbouring MBs involved in the MVP calculation. If there is more than one partition in the neighbouring MBs (in Figure 2.9, the left and upper MBs are divided into more than one partition), the nearest partition to the top–left corner of the current MB is selected in order to calculate the MVP (see A and B partitions in Figure 2.9). The MVP is calculated as the median of the three selected partitions (A, B, and C partitions in Figure 2.9).

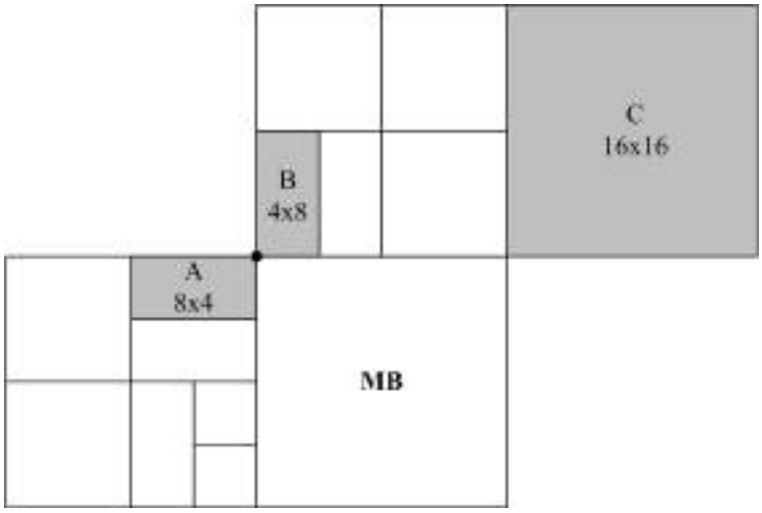


Figure 2.9: Current and neighbouring partitions.

Finally, the H.264/AVC standard allows the use of more than one reference frame per encoded frame. The number of reference frames has been considerably increased in comparison with previous standards, namely up to 16 reference frames. This mechanism is able to greatly reduce the number of bits required to store the information. Previous standards require a bigger number of bits to encode the information when there is a clear variation in the movement. By using multiple reference frames, the probability of finding a good prediction considerably increases. This mechanism is quite useful when applied in the following situations:

- Movements that are periodically repeated.
- Interpretation of movements and obstructions.
- Switching between different camera angles.

The following subsections show how to find the best MV and the cost metrics most commonly used for this. The best MV is the one with the lowest cost, i.e. the MV which requires less bits to encode both the MV and the residual block.

Cost metrics

The main objective of the MC mechanism is to minimize the energy of the residual transformed coefficients after quantization (see the block diagram of a generic H.264/AVC encoder in Figure 2.3), which depends on the energy in the residual block before being transformed. Moreover, ME aims at finding the best match which obtains the lowest cost, i.e. the MV which minimizes the energy in the residual block. Therefore, ME evaluates the residual energy using different offsets (MVs).

There are different metrics to evaluate the energy in the residual block. These metrics affect the computational complexity and the accuracy of the ME algorithm. The most common are the *Mean Square Error* (MSE) metric, the *Mean of Absolute Errors* (MAE) metric, and the *Sum of Absolute Errors* (SAE) metric. Equations 2.7, 2.8, and 2.9 describe the three energy measurements, where the block size is N x N; and C and R are the current and reference blocks, respectively.

$$MSE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} (C_{ij} - R_{ij})^2, \tag{2.7}$$

$$MAE = \frac{1}{N^2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |Cij - R_{ij}|, \tag{2.8}$$

$$SAE = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |Cij - R_{ij}|, \tag{2.9}$$

Probably, the most commonly used metric is SAE, since it gives good results and it is the least complex. The SAE metric is usually implemented as depicted in Equation 2.9, and it is also known as the *Sum of Absolute Differences* (SAD) metric. However, this metric can also be implemented as the *Sum of Absolute Differences of the Transformed residual data* (SADT). The transformed residual increases the computational complexity but improves the accuracy of the energy measurement. Usually, a multiplication–free transform is used, and the extra computational cost is not excessive. Hadamard transforms are examples of these multiplication–free transforms.

ME algorithms

This subsection provides a review of several well–known ME algorithms, which represent the most computationally expensive task of an H.264/AVC encoder. More details can be found in [Richardson 04].

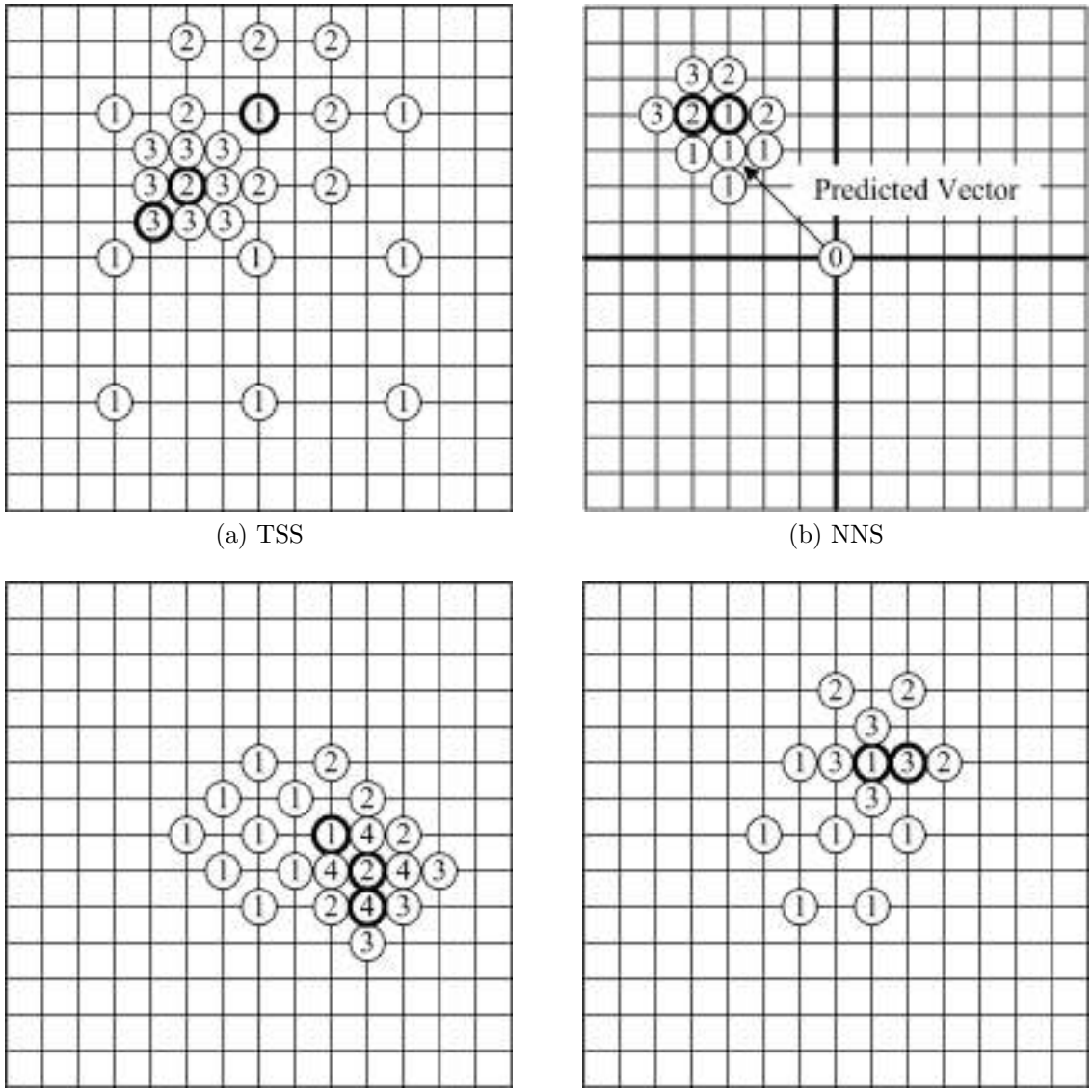
- *Full Search* (FS) is the most straightforward search algorithm, and involves evaluating the cost metric at every point inside a search area. The search area is defined as $\pm S$ samples surrounding the current MB location, in the reference frame. This algorithm always finds the minimum cost since it evaluates all possible positions within the search area. However, it is very computationally intensive because the cost is calculated at every one of the $(2S + 1)^2$ positions.

  Fortunately, the computation of the FS algorithm can be simplified by applying what it is known as *early-out termination*. If before completing the evaluation of a specific search area position, the accumulated cost is higher than the best cost previously obtained for another position, this position can be skipped. However, even with early termination, FS is too computationally expensive and it is not suitable for many applications. Therefore, in the literature it is possible to find some search algorithms which are aimed at reducing the computational cost and only evaluate a subset of the locations within the search area. These algorithms are known as *fast search* algorithms.

- *Three Step Search* (TSS), sometimes described as *N–Step Search*, is depicted in Figure 2.10a. First, the cost metric is calculated in the center of the search area, and at eight locations whose distance to the center of the search area is half of the search range ($\lceil searchrange/2 \rceil$). In Figure 2.10a, the search range is 7, the search distance is 4, and the nine locations are labelled as 1. In the second step, the position that gives the smallest cost becomes the center of the search area, the search distance is halved, and a further eight locations are searched (these locations are labelled as 2 in the figure). The algorithm is repeated until the search distance cannot be further divided.

- *Nearest Neighbour Search* (NNS) is a low complexity ME algorithm which closely approaches the performance of FS in MPEG–4 using the *Simple profile*. In NNS, a predictor vector is calculated based on the MVs of surrounding blocks (similar to the MVP calculation). The cost metric is evaluated in the center of the search area (labelled as 0 in Figure 2.10b). Then, the center of the search area is moved to the predicted position, and surrounding positions in a diamond shape are evaluated (labelled as 1 in Figure 2.10b). Further algorithm steps depend on the location of the minimum cost. If the minimum cost is located in the center of the diamond, the algorithm finishes, otherwise an extra iteration is performed.

- *Diamond–Based Search* (DBS) uses two diamond shaped search patterns, a large diamond with nine points for coarse grain search, and a small diamond with five points for fine grain search. The algorithm starts by locating the large diamond at the center of the search area, and the point with the lowest cost becomes the new center of the search area (labelled as 1 in Figure 2.10c). Then, if the new center of the search area is located at a diamond vertex, five additional points are used to perform a new coarse grain search (labelled as 2 in Figure 2.10c). If the new center of the search area is located at a diamond face, three additional points are used to perform



(a) TSS (b) NNS

(c) DBS (d) HBS

Figure 2.10: Motion estimation search algorithms.

a new coarse grain search (labelled as 3 in Figure 2.10c). Finally, if the center of the search area is located at the diamond's center, four additional points are checked. The small diamond is used to perform the final fine grain search (labelled as 4 in Figure 2.10c).

- *Hexagon–Based Search* (HBS) is similar to DBS, but checks less search area points before finishing, so it is less complex. HBS uses two hexagon shaped search patterns, a large hexagon with seven checking points for coarse grain search, and a small hexagon with five points for fine grain search. The algorithm first performs the coarse grain

search by locating the large diamond at the center of the search area (labelled as 1 in Figure 2.10d). Then, if the best matching point is located at a hexagon vertex, three additional points are checked to perform the next coarse grain search (labelled as 2 in Figure 2.10d). If the best matching point is located at the hexagon's center, the fine grain search is performed by adding four new checking points (labelled as 3 in Figure 2.10d).

2.1.3. Intra Prediction

In some situations, the inter prediction mechanism is not able to efficiently remove the redundancies which may exist inside a frame, since no movement has occurred at a certain time instant. Therefore, the inter prediction mechanism is useless and a different mechanism should be carried out. This mechanism is intra prediction, which is responsible for eliminating the spatial redundancies inside a frame. An intra prediction module can be identified in both the H.264/AVC encoder (Figure 2.3) and the H.264/AVC decoder (Figure 2.4), but its purpose is different in each case. In the encoder, its purpose is to eliminate the spatial redundancies within the frame, while in the decoder it is to reconstruct the encoded frames. Intra prediction is carried out in I slices, but is also carried out in P or B slices where no motion has occurred.

The intra prediction module is carried out at MB level, and tries to predict the current block by using neighbouring sample values that have been already processed, following a set of predefined directions. The H.264/AVC standard uses three different block sizes for intra prediction (16x16, 8x8 and 4x4), which are known as *intra prediction modes*. For each component (YUV) within an MB, one intra prediction mode and one prediction direction have to be obtained.

The luminance component (Y) can be intra predicted using the 16x16 intra mode, the 8x8 intra mode, and the 4x4 intra mode. The chrominance components (UV) can be intra predicted using only the 8x8 intra mode. Both chrominance components should select the same direction, but it is not necessary for there to be any relation between the direction selected for the luminance component and the direction selected for the chrominance components.

The H.264/AVC encoder typically selects the prediction mode for each block that minimizes the differences between the current block and neighbouring pixels. The best direction is selected based on a cost metric similar to the ones explained in the inter prediction section. In fact, the most commonly used is the SAE metric.

There are four possible directions for the 16x16 intra mode: vertical, horizontal, DC and plane. Table 2.2 explains the operations needed to calculate the cost metric and Figure 2.11 shows the available 16x16 prediction directions. *H and V* are the neighbouring samples that are set as references, which have been previously encoded and reconstructed (these samples in the encoder will be exactly the same as those a decoder will form when decoding the video sequence), and the shaded samples are the samples to be predicted. This mode is useful to eliminate the spatial redundancies of homogeneous areas in an image.

Table 2.2: 16x16 intra directions.

Direction	Description
0 (vertical)	Extrapolation from upper samples (H values).
1 (horizontal)	Extrapolation from left samples (V values.)
2 (DC)	Mean of upper and left samples (H + V values.)
3 (Plane)	A linear "plane" function is fitted to the upper and left samples.

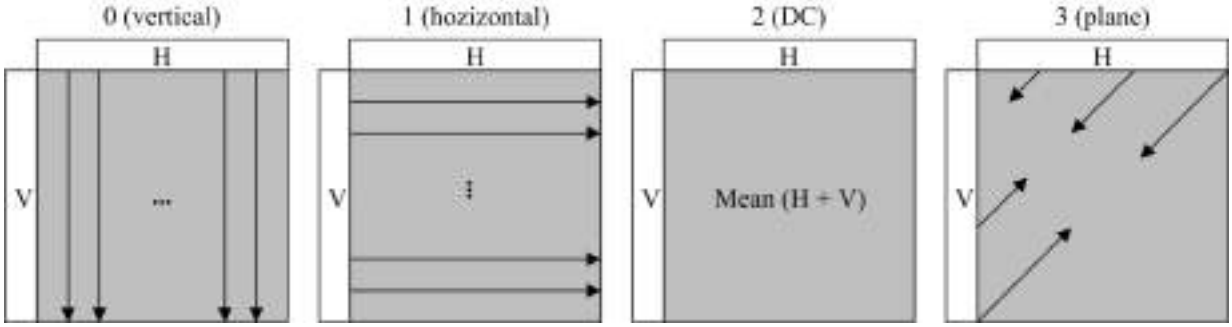


Figure 2.11: 16x16 intra directions.

For the 4x4 intra mode, there are nine possible directions. The first three directions (vertical, horizontal and DC) are the same as those used in the 16x16 intra mode. However, new prediction directions are introduced in this mode. Table 2.3 explains the operations needed to calculate the cost metric and Figure 2.12 shows the available 4x4 prediction directions. Note that the top–left sub–figure in Figure 2.12 is not a prediction direction, it shows the sample's distribution involved in the cost calculation of the 4x4 intra mode. *A to M* are the neighbouring samples that are set as references, which have been previously encoded and reconstructed (these samples in the encoder will be exactly the same as those a decoder will form when decoding the video sequence), and *a to p* are the samples to be predicted.

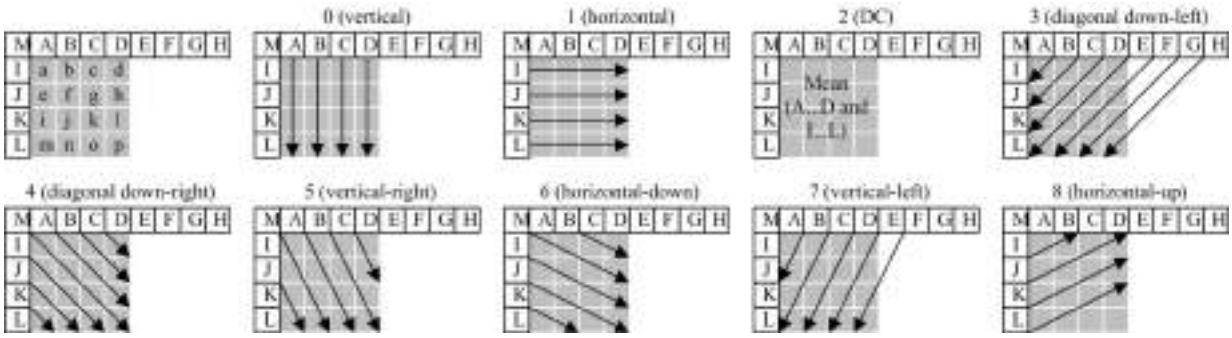


Figure 2.12: 4x4 intra directions.

Table 2.3: 4x4 intra directions.

Direction	Description
0 (vertical)	The upper samples A, B, C, and D are extrapolated vertically.
1 (horizontal)	The left samples I, J, K, and L are extrapolated horizontally.
2 (DC)	All samples (a to p) are predicted by the mean of samples A, B, C, D, I, J, K, and L.
3 (Diagonal down–left)	The samples are interpolated at a 45° angle between lower–left and upper–right.
4 (Diagonal down–right)	The samples are interpolated at a 45° angle down and to the right.
5 (vertical–right)	Extrapolation at an angle of approximately 26.6° to the left of vertical (width/height = 1/2).
6 (horizontal–down)	Extrapolation at an angle of approximately 26.6° below horizontal.
7 (vertical–left)	Extrapolation (or interpolation) at an angle of approximately 26.6° to the right of vertical.
8 (horizontal–up)	Extrapolation at an angle of approximately 26.6° above horizontal.

Finally, the 8x8 intra mode is different depending on whether it is applied to the luminance component or to the chrominance components. The 8x8 luminance intra prediction is only available in the High profiles, and it is carried out using nine prediction modes which are very similar to the nine modes previously described for 4x4 intra prediction, except that each 8x8 luminance block is filtered (using a low–pass filter) to improve prediction performance. The 8x8 chrominance intra prediction uses four prediction modes which are very similar to the four modes previously described for 16x16 intra prediction, except that the numbering of these four modes is different. The modes are 0 (DC), 1 (horizontal), 2 (vertical) and 3 (plane).

2.1.4. Transform

After applying the inter and intra prediction mechanisms, the H.264/AVC standard defines a transform module. The main aim of this module is to minimize the spatial redundancies of the predicted residual. The H.264/AVC standard defines two kinds of transforms:

- A direct transform, which is used to encode the video sequence and can be found in the block diagram of a generic H.264/AVC encoder (Figure 2.3).

- An inverse transform, which is used to decode the video sequence and can be found both in the block diagram of a generic H.264/AVC decoder (Figure 2.4) as well as in the reconstructed path of the block diagram of a generic H.264/AVC encoder (Figure 2.3).

H.264/AVC uses different multiplication–free transform algorithms in this module, depending on which residual is being applied. The first version of the standard defined the following transforms:

- 4x4 *Hadamard Transform* (HDT) for the luminance DC coefficients in MBs coded in intra 16x16 mode.
- 2x2 HDT for the chrominance DC coefficients in any MB.
- 4x4 integer *Discrete Cosine Transform* (DCT) for all other blocks.

Additionally to the transforms mentioned above, a later extension of the standard defined an 8x8 integer DCT similar to the one defined for 4x4 blocks. This newly defined transform is available in High profiles for MBs that are coded using the 8x8 intra prediction mode or any inter prediction mode. Note that previous standards, such as MPEG–2 and MPEG–4, defined 8x8 DCT algorithms, but these were floating–point–based transforms [Haskell et al. 96].

An important feature of this module is that the transforms are multiplication–free, i.e. only integer operations are used, reducing the number and complexity of the mathematical operations used (only additions and shifts are performed), and avoiding drift errors which can occur when performing floating–point operations.

2.1.5. Quantization

Quantization removes irrelevant information from the transformed blocks and is carried out by a scalar quantiser, known as the *Quantization Parameter* (QP). As in the case of the transform module, there are two kinds of quantization. A direct quantization which is used to encode the video sequence (encoder only) and an inverse quantization which is used to decode the video sequence (encoder and decoder).

The H.264/AVC defines a total of 52 values for the QP, from 0 to 51. The quantization module divides each transformed coefficient by the selected QP. Usually, a different QP value is configured for the different slice types (I, P and B) available in a video sequence. Additionally, hierarchical GOP patterns may vary the QP depending on the hierarchical level on which the current frame is located. The quantization step is doubled in size for every increment of 6 in QP.

The large number of QPs available makes it possible to control the bit rate and quality of the encoded video sequence. In general, a high QP produces lower bit rates and quality than a low QP, since the quantization step is reduced or increased, respectively.

2.1.6. De–blocking filter

After quantization, the H.264/AVC standard defines a homogenization module, which is known as the *de–blocking filter*. This filter operates at MB level, as well as on each of the sixteen 4x4 blocks into which an MB can be further divided. The de–blocking filter can be found in the block diagram of a generic H.264/AVC decoder (Figure 2.4), and in the reconstructed path of the block diagram of a generic H.264/AVC encoder (Figure 2.3).

Most of the mechanisms defined by the H.264/AVC standard are block–based and can lead to visible artefacts. The aim of this module is to reduce the visibility of these undesired artefacts (blocking artefacts). At the MB level, the de–blocking filter tries to eliminate the visible artefacts produced by the different predictions which may occur between adjacent blocks. At the 4x4 block level, the de–blocking filter tries to eliminate the visible artefacts produced by the different predictions, which may occur between adjacent blocks, as well as the artefacts produced by the transform and quantization processes.

The de–blocking filter usually modifies up to two bordering pixels of the MB/4x4 block on which it is being applied, by using a non–linear adaptive filter. This filter is able to increase the image sharpness, and to avoid the formation of undesired objects which might damage the subjective image quality. All in all, this filter prevents the blocking effect that may occur due to the fragmentation of the image into blocks for processing.

2.1.7. Entropy coding/decoding

The entropy coding and decoding modules use fixed– or variable–length binary codes to encode/decode the video stream. The entropy encoder can be found in the generic block diagram of an H.264/AVC encoder (Figure 2.3), while the entropy decoder can be found in the generic block diagram of an H.264/AVC decoder (Figure 2.4).

The H.264/AVC standard specifies several methods for coding the output of the quantization module. These methods are as follows:

- Fixed length code. An element is converted into a binary code with a specific length (n bits).

- Exponential–Golomb *Variable Length Code* (VLC). An element is represented using a Golomb codeword with a variable number of bits. Usually, the symbols which occur most frequently use the shortest codewords.

- *Context–Adaptive Variable Length Coding* (CAVLC). It is a method designed to code transformed coefficients in which different VLCs use context–adaptation. The variable length codes are chosen depending on the statistics of recently–coded coefficients.

- *Context–Adaptive Binary Arithmetic Coding* (CABAC). It is an arithmetic coding method in which probability models are updated based on previous coding statistics.

VLCs, such as Exponential–Golomb, may be used as a solution to encode data with varying probabilities. Frequent symbols are assigned to short codewords and less common symbols are assigned to long codewords.

CAVLC is used to encode residual blocks, scan ordered blocks, or transformed coefficients, and it is available in all profiles. Entropy coding is applied using 4x4 blocks (if the 8x8 DCT is used, entropy coding is performed by dividing the block into four 4x4 blocks), which are scanned using a zigzag or a field scan and converted into a series of VLCs, based on VLC tables which depend on already encoded elements. Moreover, it is designed to take advantage of certain characteristics of the quantized blocks:

- After quantization, blocks often contain mostly zeros.
- The highest non–zero coefficients after the block scan are often sequences of ± 1.
- The number of non–zero coefficients in neighbouring blocks is correlated.
- The level or magnitude of non–zero coefficients tends to be larger at the start of the scanned array, near the DC coefficient, and smaller towards the higher frequencies.

CABAC is an optional entropy coding mode available in Main and High profiles. CABAC achieves good compression by selecting probability models for each syntax element according to the element's context, by adapting probability estimations based on local statistics and by using arithmetic coding rather than VLC. Coding an element involves the following steps:

- Binarization. Each element is converted into a binary code. It is similar to the process of converting a data element into a VLC. However, the following stages are repeated for each bit of the binarized elements.
- Context model selection. A context model is a probability model for one or more bits of the binarized element and it is chosen from a selection of available models depending on the statistics of recently–coded data elements.
- An arithmetic coder encodes each bit according to the selected probability model.
- The selected context model is updated.

CABAC can provide improved coding efficiency compared with CAVLC at the expense of greater computational complexity.

2.1.8. Other mechanisms/tools

The modules described above provide great flexibility to the different applications which make use of the H.264/AVC standard. However, this standard also improves the coding efficiency of previous standards, such as MPEG–2 [Wiegand et al. 03]. In the following lines, other mechanisms/tools which contribute to the coding efficiency and robustness of the H.264/AVC standard are briefly mentioned:

- The standard defines a *Network Abstraction Layer* (NAL) which provides mechanisms (header information for transportation) to keep the video syntax, regardless of the network type on which it is being transmitted or the media storage used.
- The standard supports *Flexible Macroblock Ordering* (FMO) inside a frame, which is aimed at minimizing the impact of the errors which may occur during transmission.
- The standard makes it possible to divide the data into different packets which are handled depending on their importance.
- The standard makes it possible to include redundant frames to mitigate errors in the reconstruction process.

2.1.9. Profiles and levels

The main purpose of the H.264/AVC standard is to offer a good quality standard that is able to considerably reduce the output bit rate of the encoded video sequences, compared with previous standards. Moreover, it was designed with a view to being applied in a wide variety of applications such as DVD, video–streaming, HDTV, etcetera. Therefore, if it was designed to be applied in a wide variety of applications, that means that it was designed to satisfy the different requirements of these applications. For example, the requirements of DVD are completely different from those of a video streaming application.

In this context, the concept of profile was introduced. A profile is a set of coding tools defined to satisfy the requirements of a set of applications, defining what is required on both the encoder and decoder sides. In 2003, the H.264/AVC standard was originally defined with three profiles (Baseline, Main and Extended). Later, in 2004, with the definition of FREext, four new profiles were added (High, High10, High4:2:2, and High4:4:4). Finally, in 2009, with the last standard update, five new profiles were added (Constrained Baseline, High10 Intra, High4:2:2 Intra, High4:4:4 Intra and CAVLC4:4:4 Intra). Roughly, these twelve profiles can be grouped into four main profiles: Baseline, Main, Extended and High. Moreover, the MVC and SVC extensions, added in 2009, introduce their own profiles.

Figure 2.13 shows a schematic view of the tools/modules available in the most common profiles defined by the H.264/AVC standard. In the center of the figure some tools that are available in all profiles can be identified. In the later standard update, these common tools were grouped to form the Constrained Baseline profile. These common tools are:

- I slices. 16x16 and 4x4 intra predictions are available in all profiles, 8x8 intra prediction is only available in the High profiles.
- P slices. Inter prediction using previous reference frames is available in all profiles.
- De–blocking filter.
- CAVLC as entropy encoder/decoder is available in all profiles.

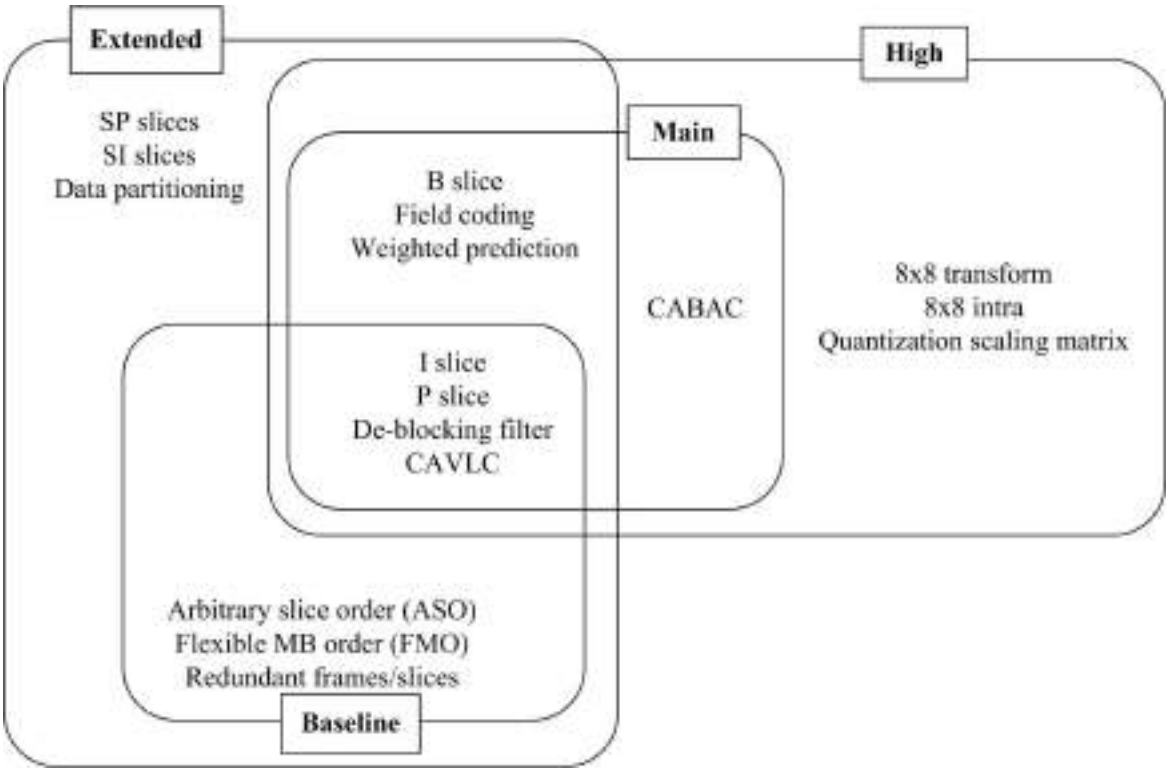


Figure 2.13: H.264/AVC profiles.

The Baseline profile was designed for low–complexity applications which demand real–time encoding and decoding, such as video telephony, video conferencing, wireless applications, and mobile applications. This profile includes the common tools, as well as the following ones:

- *Arbitrary Slice Order* (ASO). The slices within a frame may follow any order.
- FMO aimed at minimizing the impact of the errors which may occur during transmission.
- Redundant frames/slices can be used to avoid errors during transmission or storage. The redundant frames/slices can be a complete frame or a part of it.

The Main profile was designed for broadcasting applications, such as DTV and video storage. This profile includes the common tools, as well as the following ones:

- B slices. This profile supports inter prediction using previous frames, future frames or a combination of both as reference frames. This slice type improves the coding efficiency of I and P slices, but it also increases the computational cost.
- Field coding/interlaced video. This profile makes it possible to sample the video sequence as a sequence of interlaced fields. In an interlaced video sequence, half of

the data in a frame belongs to a field (odd lines within a frame) and the other half of the data to a different field (even lines within a frame), each representing half of the information of a complete frame, doubling the perceived frame rate.

- Weighted prediction. This mechanism makes it possible to modify the prediction data in P or B slices. This mechanism penalizes frames which are far away from the current frame and benefits frames which are close to the current frame.
- CABAC as entropy encoder/decoder is available.

The Extended profile was designed for multimedia streaming applications, i.e. applications that require high compression and high reliability. This profile includes the tools available in the Baseline and in the Main profiles (except that CABAC is not supported as entropy encoder/decoder), as well as the following tools:

- SP and SI slices, which are special slice types that allow efficient switching between video streams and efficient random access for video decoders.
- Data partitioning. The encoded data may be split into different partitions to improve the robustness of the video stream transmission.

Finally, the High profile (and all profiles whose name begins with High) was designed for professional applications, such as HD broadcast and disc storage (HD DVD and Blu–ray disc), video acquisition and edition, and ultra high quality broadcast applications that demand lossless video. This profile includes the tools available in the Main profile, as well as the following ones:

- 8x8 intra prediction.
- 8x8 transform.
- Quantization scaling matrices to improve the subjective quality of the video sequence. Different scales are used according to specific frequencies associated with the transformed coefficients.
- More than 8 bits per sample to obtain a more accurate video representation (10 bits are supported in the High10 and High4:2:2 profiles, and 12 bits in the High4:4:4 profile).
- Extra sampling formats (YUV 4:2:2 is supported in the High4:2:2 profile and YUV 4:4:4 is supported in the High4:4:4 profile) and color spaces (RGB).

An H.264/AVC level specifies an upper limit on the frame size, processing rate (number of frames or blocks which can be decoded per second) and working memory required to decode a video sequence. A particular decoder can only decode H.264/AVC bit streams up to a certain combination of profile and level.

Table 2.4 shows the upper limits of the different levels defined in the H.264/AVC standard. The decoding speed column shows the maximum number of MBs that can be decoded per second; the frame size column shows the maximum number of MBs within a frame; the bit rate column shows the maximum bit rate supported for non High profiles (note that the maximum bit rate for the High profile is 1.25 times that of the Baseline, Extended and Main profiles; 3 times for the High10 profile; and 4 times for the other High profiles); and finally the last main column shows the maximum resolution allowed and its corresponding frame rate taking into account the previous limits. For example, level 1 supports up to 99 MBs per frame, and a 176x144 video sequence is composed of 99 MBs (176/16 = 11 MBs, 144/16 = 9 MBs and 11x9=99 MBs), the frame rate for the highest resolution supported is calculated dividing the decoding speed and the frame size (1,485/99 = 15). Lower resolutions, than the maximum, for a specific level allow higher frame rates.

Table 2.4: H.264/AVC levels.

Level	Decoding speed (MBs/s)	Frame size (MBs)	Bit rate (Kbits/s) Non High profiles	High resolution @ frame rate
1	1,485	99	64	176x144 @ 15.0
1b	1,485	99	128	176x144 @ 15.0
1.1	3,000	396	192	352x288 @ 7.5
1.2	6,000	396	384	352x288 @ 15.2
1.3	11,880	396	768	352x288 @ 30.0
2	11,880	396	2,000	352x288 @ 30.0
2.1	19,800	792	4,000	352x576 @ 25.0
2.2	20,250	1,620	4,000	720x576 @ 12.5
3	40,500	1,620	10,000	720x576 @ 25.0
3.1	108,000	3,600	14,000	1,280x720 @ 30.0
3.2	216,000	5,120	20,000	1,280x1,024 @ 42.2
4	245,760	8,192	20,000	2,048x1,024 @ 30.0
4.1	245,760	8,192	50,000	2,048x1,024 @ 30.0
4.2	522,240	8,704	50,000	2,048x1,080 @ 60.0
5	589,824	22,080	135,000	3,672x1,536 @ 26.7
5.1	983,040	36,864	240,000	4,096x2,304 @ 26.7
5.2	2,073,600	36,864	240,000	4,096x2,304 @ 56.3

2.2. Multi View Coding (MVC)

MVC [ITU–T and ISO/IEC 09] is a compression standard which provides a compact representation for multiple views of a video scene, using a single bit stream. It was developed as an extension of the H.264/AVC standard (Annex H) and, like the H.264/AVC standard, was jointly developed by the ISO/IEC MPEG and the ITU–T VCEG. The first draft was released in 2008, while the final draft came out in November 2009.

The MVC extension enables inter–view prediction to improve coding efficiency, and supports traditional temporal (inter) and spatial (intra) predictions. This extension is backward compatible with H.264/AVC, so legacy devices are able to decode MVC bit streams by discarding the information which does not belong to the base view (a further description of the base view can be found in the following sections). MVC applications include 3D television, advanced surveillance systems, immersive teleconferencing and gaming.

In the literature some potential implementations of MVC can be found, and these include:

- Stereoscopic video. A stereo pair of views of the visual scene are combined. This combination can be achieved by using data glasses or an autostereoscopic display, thus giving the 3D sensation. Unfortunately, this 3D visualization has a limited viewing angle (Figure 2.14a).
- 3D video. Multiple real or rendered views of the visual scene are combined and presented to the viewer. This combination can be carried out by using virtual reality glasses or an advanced autostereoscopic display. In this scenario, the viewer has the feeling of immersion in the 3D scene (Figure 2.14b).
- Free viewpoint video. Some views of the scene are available and the viewer may select an arbitrary viewing angle. If the selected viewing angle does not exist, it is rendered or created from other existing views (Figure 2.14c). Examples of this representation include multiple cameras at a sports game or multiple surveillance cameras.

(a) Stereoscopic view (b) 3D view

(c) Free viewpoint / rendered view

Figure 2.14: Multi–view video: view examples.

In a similar way to the H.264/AVC standard, MVC has inherent redundancy (temporal and spatial), but it adds a new kind of redundancy which is known as *inter–view redundancy*. The purpose of MVC is to take advantage of this redundancy and efficiently encode the scene. Figure 2.15 shows a multi–view scene composed of three views. Each view can be encoded as a separate H.264/AVC bit stream. However, the inter–view correlation/redundancy can be exploited by using a single MVC bit stream offering better coding efficiency. If the cameras are close together, frame 0 of view 0 may be strongly correlated to frame 0 of view 1 and frame 0 of view 1 may be strongly correlated to frame 0 of view 2, etcetera. In fact, this correlation may appear in all the frames of a multi–view video.

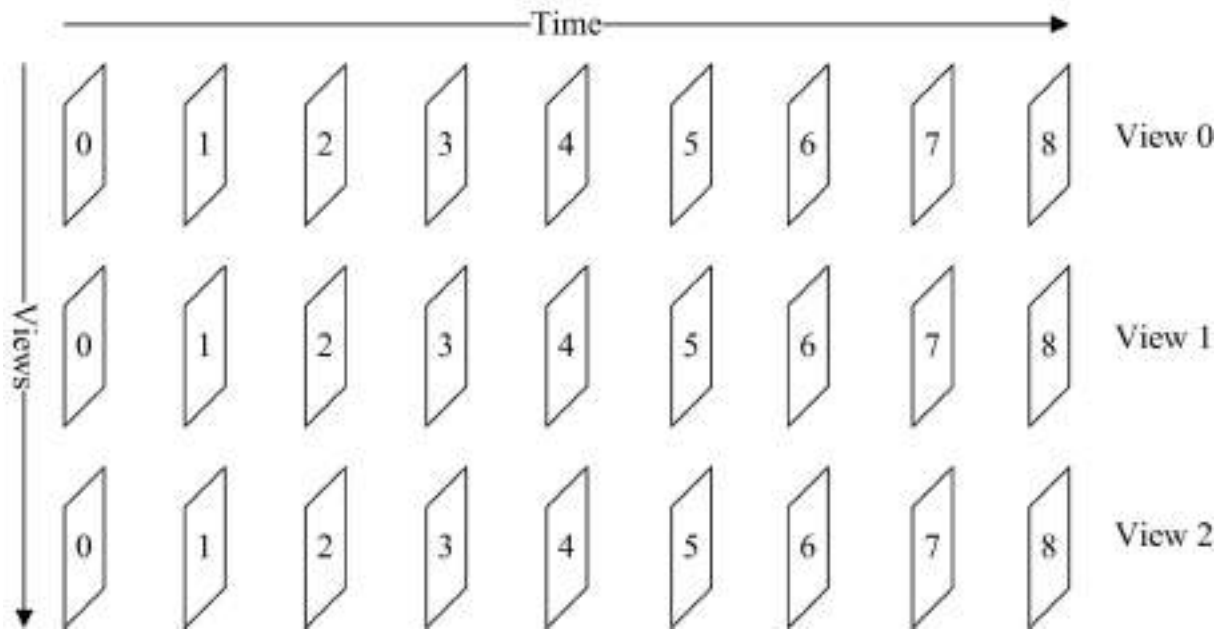


Figure 2.15: MVC, Views and frames.

The MVC extension modifies the basic H.264/AVC syntax to support multi–view coding as follows:

- Parameters set.
- The Reference Picture List is modified to support prediction from different views.
- NAL Units are modified to include extra information about the base view (a legacy H.264/AVC decoder must be able to discard secondary views by checking the extra information provided by the MVC encoder).
- Picture numbering and reference index are modified to support multiple views.

2.2.1. Inter–view prediction

The redundancies in a multi–view scene can be exploited by introducing predictions between views, and this is commonly known as *inter–view prediction*. This inter–view prediction is the main contribution of the MVC extension to the H.264/AVC standard.

Figure 2.16 shows an example of inter–view prediction. Each view is predicted using a hierarchical GOP pattern [Schwarz et al. 06], which is composed of a key frame (I or P frame) followed by seven B frames (the predictions for view 1 are not shown for clarity), where the B frames are encoded using a hierarchical structure, i.e. the 4^{th} frame is predicted using the key frames, the 2^{nd} and 6^{th} frames are predicted using a key frame and the 4^{th} frame, and so on. Note that view 0 is predicted using conventional H.264/AVC tools and no frames are predicted using frames from a different view, which means that this view can be decoded by any H.264/AVC decoder or by an MVC decoder, and is considered as the base view.

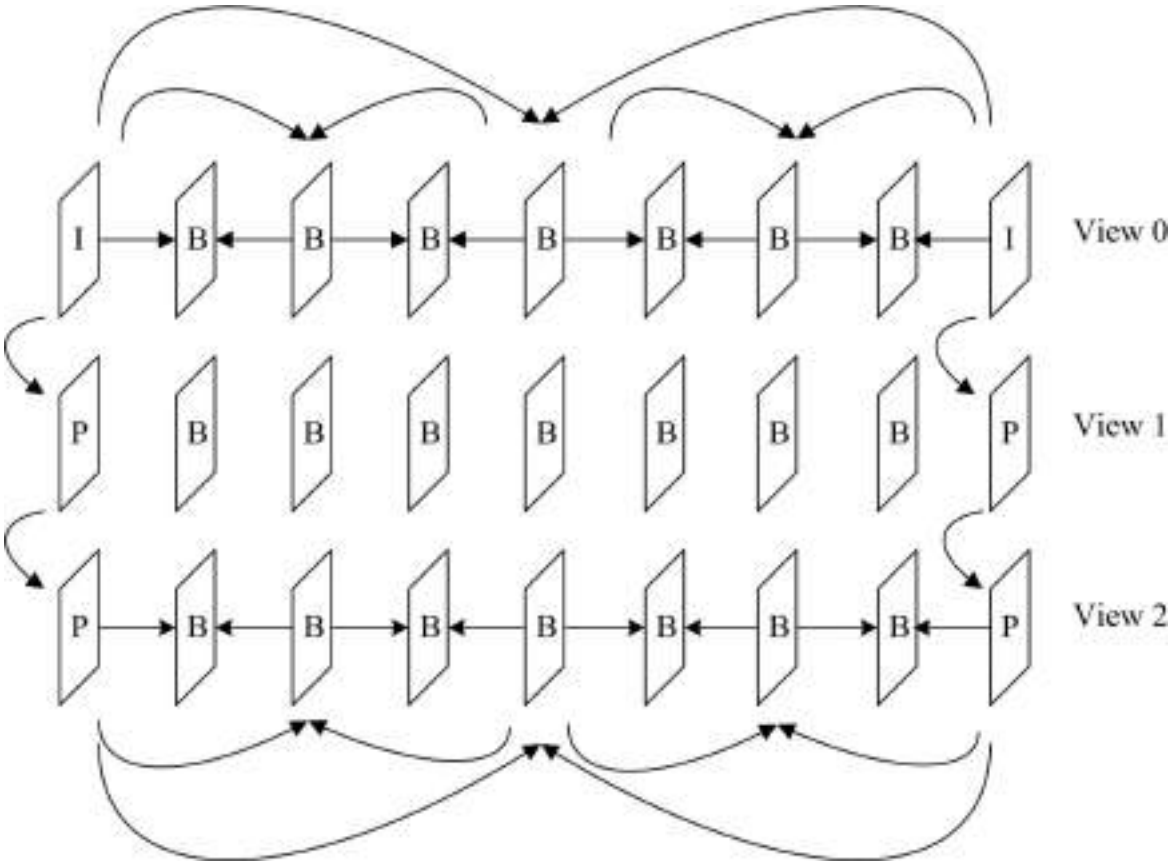


Figure 2.16: Inter–view prediction of key frames.

A more complex prediction structure can be defined including inter–view predictions for all B frames (except for the B frames included in the base view). More details about different inter -view prediction structures can be found in [Vetro et al. 11], in [Richardson 10] and in [Merkle et al. 07].

MVC inherits many video coding techniques from H.264/AVC, such as variable block–size matching ME. However, in the case of MVC, when the ME process is applied using a reference frame from a different view (inter–view prediction), it is more commonly known as *Disparity Estimation* (DE) since it estimates the differences between adjacent viewpoints/-cameras. As with ME, variable block-size DE is carried out using eight inter prediction modes (SKIP, Inter 16x16, Inter 16x8, Inter 8x16, Inter 8x8, Inter 8x4, Inter 4x8, and Inter 4x4), which are depicted in Figure 2.6.

2.2.2. Profiles and Levels

As with H.264/AVC, a profile determines the subset of coding tools that must be supported by conforming encoder/decoders. The MVC extension added two new profiles based on the H.264/AVC standard. These profiles are:

- Multi–View High profile, which provides support for an arbitrary number of views.
- Stereo High profile, which provides support for two-view stereoscopic video. This profile was selected by the Blu–Ray Disc Association [Blu 05] as the coding format for 3D video with HD resolution.

As mentioned at the beginning of this section, this standard is backward compatible, so legacy devices must be able to decode at least one view (the base view). In order to facilitate this compatibility, the base view must be encoded following the High profile or the Constrained Baseline profile.

Figure 2.17 shows a schematic view of the tools available in the profiles defined in the MVC extension. In general, the Stereo High profile may be viewed as a superset of the High profile, but adding support for inter–view predictions using two views. The Multi–View High profile may be viewed as a generalization of the Stereo High profile adding support for more than 2 views. However, the Multi–View High profile does not support field coding/interlaced video, and therefore it does not support the *Macroblock Adaptive Frame Field Coding* (MBAFF) mode. Note that the base view cannot use field coding and MBAFF since it is not supported in the Constrained Baseline profile to ensure compatibility with previous devices. It is possible to obtain an encoded bit stream that satisfies both profiles when using two views and not using interlaced coding tools in the secondary view.

The levels defined in MVC are very similar to the ones defined for the H.264/AVC standard (see Table 2.4). Some limits have not been modified, such as the maximum bit rate, and other limits have been multiplied by a factor of 2, such as the decoding speed. This scale factor makes it possible to decode a stereo view sequence using the same level as specified for single–view video sequences, at the same resolution. However, this factor does not scale with the number of views used. To decode a higher number of views (more than 2), one would use a higher level, and/or reduce the video sequence resolution, and/or reduce the frame rate.

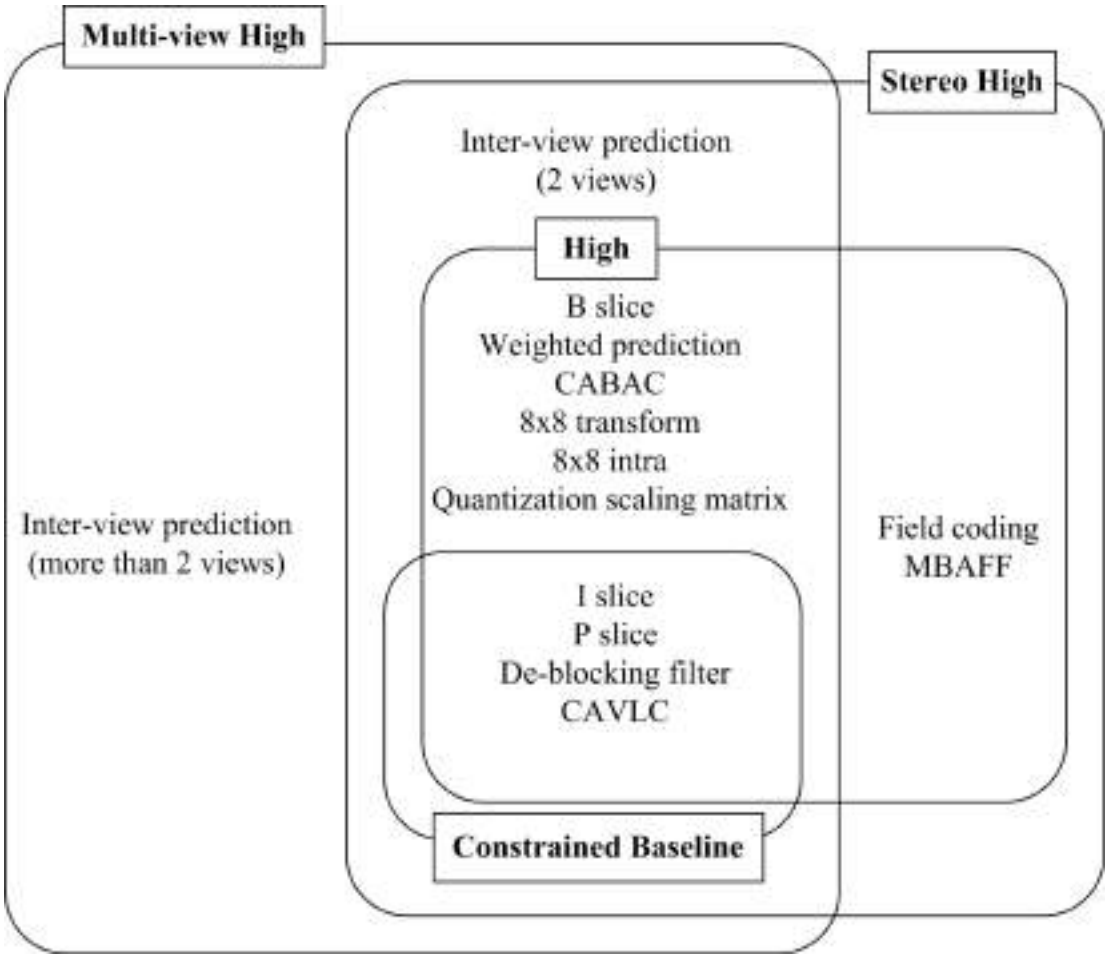


Figure 2.17: MVC profiles.

2.3. Graphic Processing Units (GPUs)

Since the beginning of modern computing, most computer programs have been written using a serial programming model. However, the availability of new, cheap and more powerful parallel platforms has meant that the serial programming model has suffered from certain limitations in performance terms when compared with the parallel programming model. These parallel platforms include supercomputers, clusters and many–core processors. Therefore, migrating sequential algorithms to parallel platforms is becoming very common.

In the past few years, new heterogeneous architectures have been introduced in high performance computing [Feng and Manocha 07]. Examples of this type of architectures are GPU–based platforms, *Cell Broadband Engine* (Cell BE), and *Field Programmable Gate Array* (FPGA)–based platforms. Modern desktop (or server) graphics cards include a many–core processor chip, which is known as a GPU. This processor chip is built following

the *Single Instruction Multiple Data* (SIMD) architecture model and it is able to perform arbitrary and programmable operations on data sent to it.

Recently, there has been a marked increase in the performance and capabilities of GPUs, such that they have attracted a lot of attention. Modern GPUs are able to achieve up to 3 TFLOPS, working in simple precision mode, and up to 190 GB/s of memory transfer rate. The GPU's performance increase has been much higher than the CPU's performance increase. In fact, over the last ten years, the performance of GPUs has doubled every six months, while the performance of CPUs has doubled every eighteen months.

Although GPUs were originally designed for multimedia and computer or console gaming, the rapid increase in the GPU's performance has spawned a research community that has successfully mapped a wide range of complex applications for them. In this way, a new trend has recently appeared in the GPU community focusing on general purpose tasks, leading to what is referred to as the GPGPU [GPGPU 07]. It is because of these improvements that more and more people use GPUs for everything and not just for graphics. General–purpose applications development for GPUs has recently gained momentum as a cost–effective approach for accelerating data– and compute–intensive applications, and this has been driven by the introduction of C–based programming environments. This emerging world of highly parallel systems requires a programming model that scales from one generation of parallel architectures to the next.

Nowadays, parallel processing on multi-core processors is one of the biggest software challenges in the industry. In recent years, some approaches to parallel processing have appeared in the industry. Examples of these parallel platforms are the RapidMind multi-core development platforms, as parallel processing for the x86 technology [Monteyne 08]; the PeakStream math libraries for graphics processors [Papakipos 07]; the Fujitsu remote procedure calls [Koeda 07]; the Ambric development-driven CPU architecture [Ambric 06]; and the Tilera tiled mesh network [Tilera 08]. In the near future, all *Personal Computer* (PC) processors and game consoles will include graphics cores, and therefore graphics processors will be integrated in the consumer market.

The GPU philosophy differs so much from the philosophy of mono–core CPUs, multi–core CPUs or even superscalar CPUs, which may route the instruction stream through multiple pipelines. In a GPU there is only one instruction stream. Fortunately, the main GPU manufacturers (NVIDIA and ATI/AMD) have developed their own tools for transparent programming, proposing new languages or even extensions for the most common high level programming languages. In this respect, NVIDIA proposes *Compute Unified Device Architecture* (CUDA) [NVidia 12], which is a software/hardware platform for massively parallel high-performance computing on their company's powerful GPUs. CUDA is steadily winning customers in scientific and engineering fields.

2.3.1. Programming model

CUDA was introduced in 2006 by NVIDIA, and is a general purpose parallel computing architecture that makes it possible to solve many complex computational problems by

using the parallel engine available on NVIDIA GPUs. CUDA C is a C–based high level programming language designed to maintain a low learning curve for programmers familiar with standard C. Additionally, NVIDIA GPUs can also be programmed using other high level programming languages, such as CUDA FORTRAN, OpenGL, OpenCL, OpenAcc, or DirectCompute.

CUDA C includes C/C++ software development tools, function libraries, and a hardware abstraction mechanism that hides the GPU hardware from developers, like an *Application Programming Interface* (API). Moreover, this also has the advantage of being able to be extended from one generation to the next. Although CUDA requires programmers to write special code for parallel processing, it does not require them to explicitly manage threads in the conventional sense, which greatly simplifies the programming model. CUDA development tools work alongside a conventional C/C++ compiler, so programmers can mix GPU code with general purpose code for the host CPU. At this point, the programmers do not explicitly manage threaded code, as a hardware thread manager handles the threads automatically, which is an important feature of CUDA.

In CUDA, the calculations are distributed in a grid of thread blocks, where each thread block has the same size (number of threads). These threads execute the GPU code, known as the *kernel*. All the threads within a thread block execute the same GPU code over different data (SIMD philosophy). Figure 2.18 shows this organization for two GPU kernels (labelled as Kernel 1 and Kernel 2), where each GPU kernel is composed of a grid of thread blocks and each thread block is composed of a constant number of threads. The grid

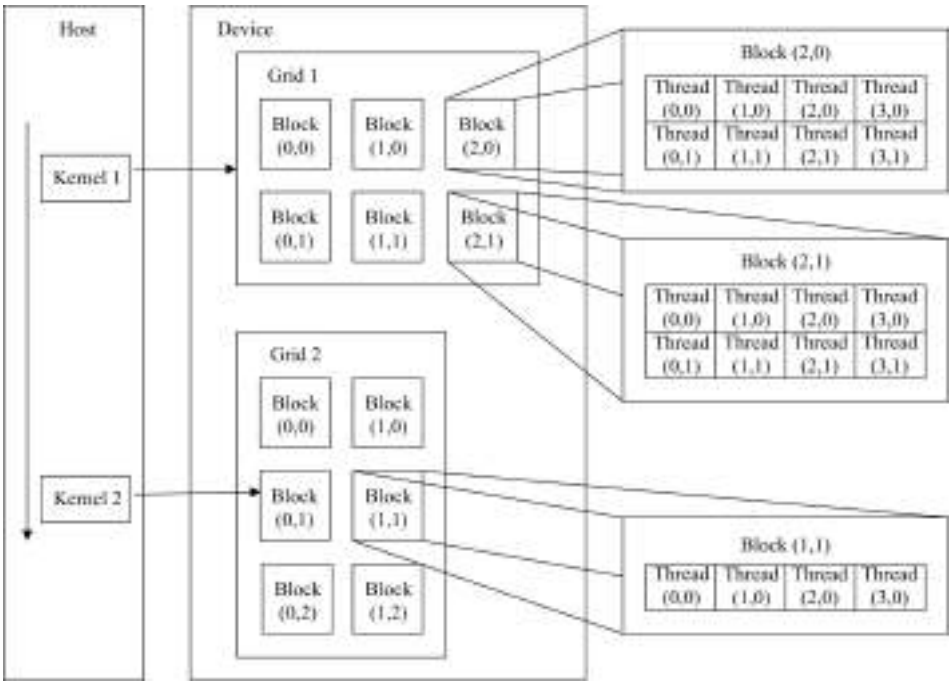


Figure 2.18: Thread organization in a GPU.

dimensions and the number of threads within each thread block must be carefully chosen in order to obtain the best performance, and this depends on the algorithm to be executed.

The compute capability of a computing device says to which core architecture a device belongs, and this is very important because the availability of some of the GPU features depends on it. The compute capability is defined by a major revision number and a minor revision number. The major revision number for devices belonging to the Kepler architecture is 3, the major revision number for devices belonging to the Fermi architecture is 2 (2.x), previous devices are all of compute capability 1.x. The minor revision number corresponds to certain improvements in the core architecture. A list containing all CUDA–enabled devices and their compute capability can be found in [NVidia 12].

In CUDA C, GPU Kernels are C functions that are executed $N \cdot M$ times, by $N \cdot M$ different threads, where N is the number of thread blocks configured and M is the number of threads within each thread block. A kernel is defined using the *__global__* specifier and it is invoked specifying the number of threads that will execute that kernel. Each thread has its own ID which is accessible through built–in variables. The following code is an example of how to declare and call a kernel.

```
//Kernel definition
__global__ void Function(int* a){
        ...
}
int main() {
        ...
        //Kernel invocation, where
        //N is the number of thread blocks, and
        //M is the number of threads within the thread block
        Function<<<N,M>>>(a);
        ...
}
```

The number of thread blocks and the number of threads within a thread block can be defined by using an integer or a 3–dimensional vector. However, there are some limits to both of them, because the available resources in a GPU are limited. Table 2.5 summarizes these limits. The maximum number of threads per thread block is limited to 1024, but fortunately the limit in the number of thread blocks is considerably higher, allowing the programmers to configure some billion of concurrent threads in a GPU kernel.

In general, the GPU can be viewed as a logical grid, each thread block as a multi–core processor, and each thread as a processing element. Each multi–processor executes a set of thread blocks in time slots, but a specific thread block is always executed on the same multi–processor. Furthermore, the threads within the thread block are further divided into warps, where each warp executes the same instruction over different data at a time stamp. The warp is the scheduling unit used by the hardware scheduling manager. Usually, the

Table 2.5: Kernel limits.

Technical limit	Compute capability						
	1.0	1.1	1.2	1.3	2.0	2.1	3.0
Maximum dimensionality of a grid of thread blocks	2				3		
Maximum x-dimension of a grid of thread blocks	65535						$2^{31} - 1$
Maximum y–, or z–dimension of a grid of thread blocks	65535						
Maximum dimensionality of a thread block	3						
Maximum x- or y-dimension of a thread block	512				1024		
Maximum z-dimension of a thread block	64						
Maximum number of threads per thread block	512				1024		

number of threads within each thread block is higher than the warp size, so a thread block is split into multiple scheduling units.

Threads within a thread block can share data through on–chip memory (fast access memory) and can synchronize their execution in order to coordinate memory accesses. However, threads of different thread blocks can only communicate through off–chip memory (DRAM).

In a system composed of multiple CUDA–enabled devices, all of them are accessible as independent devices. However, there are some restrictions when the system is in *Scalable Link Interface* (SLI) mode. Some instructions must be provided in order to select the device used by the application.

The CUDA programming model assumes a system composed of a host (CPU) and a device (GPU). The device is a coprocessor that executes the CUDA kernels and the host executes a C program. Usually, the host and device codes are executed concurrently. However, in order to facilitate concurrency, some function calls are asynchronous. These are:

- Kernel launches.
- Device to device memory copies.
- Host to or from device memory copies with a maximum size of 64 kBytes.
- Memory copies performed by functions ended with the suffix *__Async*.
- Memory set function calls.

Concurrency can be globally disabled through an environment variable. This feature is provided for debugging purposes and should never be used for released software. CUDA debuggers and profilers automatically disable concurrency and all launches are synchronous.

2.3.2. Hardware model

This section describes the most relevant hardware specifications of NVIDIA GPUs. More information regarding this topic can be found in [NVidia 12].

GPU architecture is based on having a large quantity of processing elements, or cores, integrated on a single chip at the expense of a significant reduction in cache memory (see Figure 2.19). Therefore, there are more transistors for data processing than for data caching and flow control. In fact, GPUs were originally designed for compute–intensive and highly parallel computations (graphic rendering).

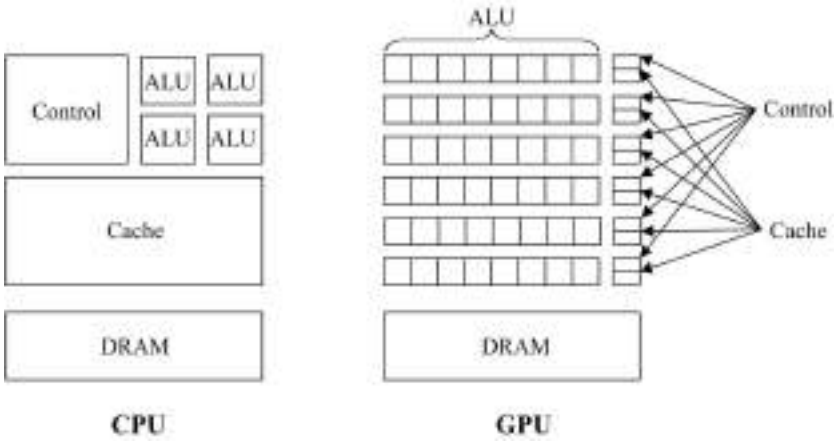


Figure 2.19: Comparison between a 4 core CPU and a generic GPU architecture.

The GPU architecture is determined by the graphics applications for which they were originally designed, which are completely different from the applications of a traditional CPU. More specifically, the GPU is designed to address problems expressed as data–parallel computations with high arithmetic intensity (the same operation is executed on many data elements, and the number of arithmetic operations is higher than the number of memory transactions). In this scenario, a sophisticated flow control and big data caches are not necessary since the same operation is executed for each data element and the memory latency can be hidden with arithmetic operations. GPU architecture is highly segmented and follows the SIMD programming model.

Modern graphics cards are composed of a GPU chip and an external DRAM memory. The GPU and the DRAM memory are connected via a high speed I/O interface (typically *Peripheral Component Interconnect* (PCI) Express). Figure 2.19 shows a schematic comparison between a 4-core CPU architecture and a generic GPU architecture.

Figure 2.20 shows the main components of any NVIDIA GPU, which is composed of a set of SIMD multi–processors known as *Stream Processor*s (SPs). Each SP is composed of a set of *Processing Element*s (PEs) or cores, which execute the same instruction on different data, on every clock cycle. The number of SIMD multi–processors and the number of PEs within the SIMD multi–processors depends on the compute capability of the device, as well as on the commercial model of the graphics card. Moreover, each SP has a set of resources shared by all PEs within the SP:

- 32–bit registers.
- A high speed local memory shared by all PEs within each SP. Note that this memory can be used as an L1 global memory cache for devices of compute capability 2.0 and above.
- A read only constant memory cache. The constant cache memory working set per SP is 8 kB.
- A read only texture memory cache. The texture cache memory working set per SP is device dependent, between 6 kB and 8 kB.

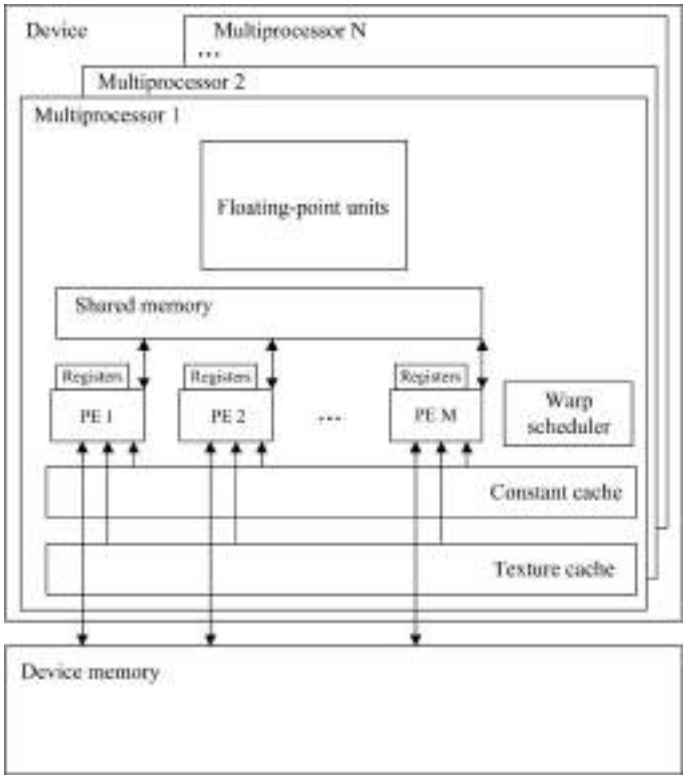


Figure 2.20: Basic hardware model of a GPU.

- Floating–point units. Single–precision floating–point arithmetic operations are supported by all devices, but double–precision floating–point arithmetic operations are only supported by devices of compute capability 1.3 and above.
- Warp schedulers.

The number and size of most of these resources depends on the compute capability of the device, and are summarized in Table 2.6.

Table 2.6: SP technical specifications per compute capability.

Technical Specification	Compute capability						
	1.0	1.1	1.2	1.3	2.0	2.1	3.0
# of cores per SP	8				32	48	192
# of 32–bit registers per SP	8k		16k		32k		64k
Maximun amount of shared memory per SP	16 kB				48 kB		
# of shared memory banks	16				32		
Maximun amount of local memory per thread	16 kB				512 kB		
# of floating–point units	2				4	8	32
Warp scheduler	1				2		4

On the other hand, the number of threads that can be in execution/scheduling within each SP, on every clock cycle, can vary. The SP creates, manages, schedules, and executes threads in groups of 32 parallel threads called *warps*, which are the minimum execution unit. A thread block is divided into warps in order to schedule thread execution. When a kernel begins its execution, a warp is marked as active and its execution starts. More warps are marked as active when an SP is idle (memory transactions, synchronization instructions, or the active warps have finished their execution). However, the maximum number of active warps depends on the compute capability of the device and the main restrictions are summarized in Table 2.7. Moreover, a warp can be marked as active if and only if there are resources available for its execution (registers and shared memory).

Table 2.7: Thread specifications per compute capability.

Technical Specification	Compute capability						
	1.0	1.1	1.2	1.3	2.0	2.1	3.0
Warp size	32						
Maximun # of active thread blocks per SP	8						
Maximun # of active warps per SP	24		32		48		64
Maximun # of active threads per SP	768		1024		1536		2048

For devices of compute capability 1.x, in order to execute one instruction on all threads of a warp, the warp scheduler issues the instruction over:

- 4 clock cycles for integer and single–precision floating–point instructions.
- 16 clock cycles for single–precision floating–point transcendental instructions.
- 32 clock cycles for double–precision floating–point instructions.

For devices of compute capability 2.0 at every instruction issue time, each scheduler issues one instruction, while for devices of compute capability 2.1 and 3.0, each scheduler issues two instructions. Active warps are distributed among the available warp schedulers before being planned for their first execution.

Finally, the external device memory is implemented using a *Double Data Rate* (DDR)3 or a DDR5 DRAM memory. More details about this memory and its access modes can be found in the next section.

2.3.3. Memory Hierarchy

The CUDA programming model assumes a system composed of a host (CPU) and a device (GPU), both of them with their separate memory spaces. Allocation, deallocation, and memory transactions between both memory spaces must be explicitly performed by using CUDA runtime functions. Device memory can be allocated by using linear memory or by using CUDA arrays.

Device memory can be classified depending on its access mode: global memory, local memory, constant memory, texture memory, or shared memory. All threads have a private local memory and access to the global memory. Constant and texture memory are read–only cached memory spaces and can be accessed by all threads. Global, texture, and constant memory are optimized for different memory usages. Shared memory can be accessed by all threads within each thread block.

The lifetime of global, constant, and texture memory is the lifetime of the application, whereas the lifetime of local and shared memory is the lifetime of the thread and the lifetime of the thread block, respectively. Therefore, global, constant, and texture memory are persistent across kernel launches. Local, and shared memory are not. The Memory hierarchy of an NVIDIA GPU is depicted in Figure 2.21.

The memory throughput varies considerably depending on the memory access pattern used and on the memory type. The memory transactions between the host and the device are slower than the memory transactions between the device memory and the device, so re–using data is very important. Moreover, the memory transactions between off–chip memory (global memory, constant memory, and texture memory) and the device are slower than the memory transactions between on–chip memory (shared memory and caches) and the device.

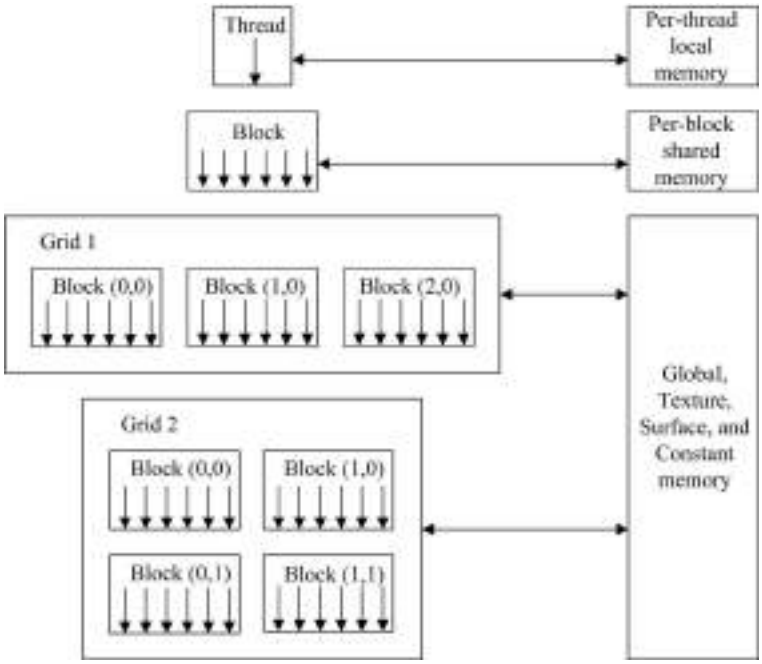


Figure 2.21: Memory hierarchy.

Shared memory

Shared memory is located on the GPU chip itself, so accessing this memory space is faster than accessing all other memory spaces, which are located in the external DRAM with the exception of cache memories, which are also located on the GPU chip.

Shared memory is organized in memory modules, called *banks*, which can be accessed simultaneously. For devices of compute capability 2.1 and below, the memory banks have a bandwidth of 4–bytes per two clock cycles; for devices of compute capability 3.0 the memory banks have a bandwidth of 8–bytes per clock cycle. The number of memory banks depends on the compute capability of the device, and are shown in Table 2.6.

Accessing this memory space is as fast as a register access, when no bank conflicts occur. If two or more threads within a warp attempt to access the same memory bank at the same time, the accesses are serialized, and hence the effective memory bandwidth decreases.

Global memory

Global memory resides in device memory and is accessed using 32–, 64–, or 128–byte memory transactions. However, global memory instructions read and write using words of 1, 2, 4, 8, or 16 bytes. When a warp of threads accesses global memory, these memory instructions coalesce into one or more memory transactions depending on the size of the words accessed and the distribution of memory accesses across the threads. Note that the first address of each memory transaction is a multiple of its size (32–, 64–, or 128–

byte), i.e. the memory transactions are aligned to the DRAM size. In general, the more memory transactions generated by a read or write instruction, the more unused words are transferred to the device, decreasing the instruction throughput.

The global memory coalescing requirements depend on the compute capability of the device used. The requirements for devices of compute capability 1.0 and 1.1 are stricter than the requirements for devices of higher compute capability. When the memory coalescing conditions are satisfied, a global memory request is split into two memory requests, one for each half–warp, and they coalesce into one or more memory transactions. If the coalescing conditions are not satisfied, 16 independent 32–byte memory transactions are performed. The coalescing conditions for devices of compute capability 1.0 and 1.1 are listed below:

- The size of the word must be 4, 8, or 16 bytes.
- If the word size is:
  - 4, the 16 words must lie in the same 64–byte segment.
  - 8, the 16 words must lie in the same 128–byte segment.
  - 16, the first 8 words must lie in the same 128–byte segment, and the last 8 words in the following 128–byte segment.
- The K^{th} thread must access the K^{th} word, i.e. the accesses must be in order.

For devices of compute capability 1.2 and above the coalescing conditions are less strict: the accesses do not have to be in order, or be in the same memory segment.

Additionally, the global memory for devices of compute capability 2.0 and above is cached, so locality can be exploited to reduce the impact on instruction throughput. The memory cache can be configured at compile time, using L1 and L2 cache levels (by default this option is used), or using L2 only. Physically, L1 cache and shared memory are the same on–chip memory. Therefore, if L1 cache is not used, the amount of shared memory available is 48 kB (see Table 2.6), otherwise the amount of shared memory available is 16 kB and the memory cache size is 32 kB. Global memory accesses using L1 and L2 are issued using 128–byte memory transactions, but are issued using 32–byte memory transactions if only L2 is used.

Local memory

Local memory is used for automatic variables, and is located in device memory. Local memory is subject to the same coalescing requirements described for global memory. As with global memory, local memory is cached for devices of compute capability 2.0 and above. The compiler uses this memory space automatically for the following variables:

- Kernels that use more registers than those available.

- Large arrays that would consume too many registers.
- Arrays which are not indexed with constant values.

Constant, Texture and Surface memory

Constant and texture memory spaces are located in device memory, and are cached for all devices. Surface memory space is also located in device memory, but is not cached. The GPU provides specific texturing hardware, which the GPU uses for graphics, to access the texture and the surface memory spaces. Therefore, accessing these memory spaces instead of global memory can have performance benefits.

Constant and texture memory spaces are read–only, and therefore there is no coherency mechanism between the device memory and its cache. An access to these memory spaces only reads from device memory on a cache miss, otherwise the word is read from its cache. Surface memory can be read and written. Texture and surface memory use the same texturing hardware for their memory transactions, but the texture cache is not kept coherent with respect to surface memory writes.

Texture memory is accessed using device functions called *texture fetches*, and fetches over objects called *texture references*. A texture reference is bound to a region of device memory called *texture*. Surface memory is accessed using device functions called *surface fetches*, and fetches over objects called *surface references*. A surface reference is bound to a region of device memory called *surface*.

A texture and a surface can be addressed using linear memory or CUDA arrays, but the latter are optimized for texture and surface fetching. The dimensionality, which specifies whether the texture and surface are addressed, can be one–, two–, or three–dimensional.

Page–locked memory

CUDA runtime allows the programmers to allocate and free page–locked host memory, instead of regular pageable host memory. The benefits of using page–locked host memory are:

- Copies between page–locked host memory and device memory can be overlapped with kernel execution.
- Page–locked host memory can be mapped into the device memory space, avoiding memory copies between both memory spaces.
- The memory bandwidth between host and device memory can be higher on systems with a front–side bus.

This type of memory must be used carefully, as allocating too much page–locked memory reduces overall system performance because it reduces the amount of physical memory that the operating system can use.

Chapter 3

Related Work

IN the H.264/AVC standard, inter prediction is the most computationally expensive task [Wiegand et al. 03]. Inter prediction is carried out in two complex steps: ME and MC. In the literature, many approaches have been proposed in order to accelerate these processes. But, up to now, there have not been many solutions making use of GPUs, which is the major focus of this thesis: to exploit the powerful GPU architecture to accelerate traditional video coding algorithms, such as H.264/AVC inter prediction, by using CUDA.

In order to describe the state of the art developed in the framework of fast H.264/AVC inter prediction algorithms, this chapter is divided into different sections: Section 3.1 reviews some inter prediction proposals developed for H.264/AVC itself; Section 3.2 reviews some proposals presented for MVC; Section 3.3 reviews the most important approaches that make use of GPUs to accelerate the H.264/AVC inter prediction algorithm; and Section 3.4 reviews some proposals that make use of parallel architectures.

Furthermore, in recent years, the computer's power and energy consumption has attracted a lot of attention. Developers are increasingly aware of the energy and power consumption of their solutions. Section 3.5 reviews the most relevant proposals that try to model and/or reduce energy consumption in the framework of parallel processing and GPU computing.

In a nutshell, there are not many approaches focusing on H.264/AVC implementation on GPUs using CUDA, and most of them fail to give *Rate Distortion* (RD) performance (quality and bit rate of the encoded video sequence). Time reduction or speed–up is a very important feature that can be achieved by using GPUs. However, all approaches must keep the RD as close as possible to the sequential approach. Moreover, there are some approaches that do not focus on video coding standards and only try to parallelize a part of them, such as the ME module. In the H.264/AVC video coding standard, the ME is only a part of the whole encoding algorithm, and therefore all the approaches should try to combine all of the encoding tools in order to show how this affects the other modules.

3.1. Faster H.264/AVC Inter Prediction algorithms - Soft–Computing techniques

In the literature, many approaches have been presented to accelerate the H.264/AVC inter prediction algorithm. Most of them are based on estimating data by using faster algorithms, determining which MB partition or partitions are not suitable to be selected (based on certain features), or determining stopping criteria for some algorithms. The H.264/AVC standard was standardized in 2003, so the proposals that involve this standard begin in 2003 and are organized by the year of publication.

In 2003:

Lim et al. [Lim et al. 03] proposed a fast mode decision algorithm based on the information supplied by the intra prediction and an edge map. However, due to the small probability of intra modes in inter frames, they suggest that this practice may have a certain limit in reducing the computational complexity of the H.264/AVC inter prediction.

Lee and Jeon in [Lee and Jeon 03] proposed a pruned mode decision algorithm capable of early selecting some coding modes, which can be discarded from being checked. This method reduces the computational complexity of the inter prediction algorithm by skipping the RD cost calculation of the discarded modes, while maintaining the coding efficiency. Experimental results report a computational complexity reduction of up to 75% in the RD cost calculation.

Yin et al. in [Yin et al. 03] proposed a scheme to jointly optimize the ME and the mode decision. The authors define some thresholds to determine which is the most suitable mode to be selected, and use a fast ME algorithm which is based on the well-known *Enhanced Predictive Zonal Search* (EPZS) ME algorithm [Tourapis 02]. Experimental results report up to a 90% reduction in the computational complexity of the RD cost calculation and a bit rate increment of up to 3%, when compared with the reference ME algorithm.

In 2004:

Kim et al. in [Kim et al. 04], based on the properties of an all-zero coefficient blocks that are produced by quantisation, proposed a fast mode decision able to effectively eliminate unnecessary modes. However, in this method, there are several threshold values that should be predefined. Furthermore, the transform coefficients must be available to make a fast decision (see the basic block diagram of a generic H.264/AVC encoder in Chapter 2). Experimental results show that this algorithm is two times faster than the complete mode decision of the JM 7.2 reference encoder, with negligible coding loss. This article also includes a comparison against the proposals presented in [Lim et al. 03] and [Lee and Jeon 03], outperforming both proposals in terms of execution time and coding efficiency.

Yu in [Yu 04] proposed a fast block size selection algorithm for inter frame coding. The algorithm relies on two predictive factors: the intrinsic complexity of the MB and

the knowledge of the mode decision carried out for already encoded frames. The first P frame within each GOP is not coded using the proposed algorithm since it requires the mode decision resulting from the previous frame, which is not available (intra frame). This algorithm can save up to 31% of the total encoding time, when compared with the JM 6.1 reference encoder.

Lee and Jeon in [Lee and Jeon 04] proposed two techniques in order to reduce encoder complexity, with small coding loss. The first one is an early SKIP mode decision, which makes it possible to omit all the remaining RD calculations, motion estimations and intra predictions. The second one is a selective intra mode decision which avoids the need to calculate all the inter frame modes when there is a correlation between surrounding pixels in the spatial direction. The first technique obtains a time reduction of 14% of the total encoding time, the second one a time reduction of 20%, and the combination of both techniques obtains a time reduction of 29% when compared with the JM 6.1 reference encoder.

Chang et al. in [Chang et al. 04] proposed a method to speed up the mode decision carried out for P frames. It exploits the inter and intra correlations between neighbouring blocks. This method requires the computation of two thresholds for measuring the goodness of the prediction. Simulation results show that this method can achieve a time reduction of between 33% and 59% depending on the video sequence, with negligible bit rate increments.

In 2005:

Wu et al. in [Wu et al. 05], using the spatial homogeneity of a video object's textures and the temporal stationary characteristics inherent in video sequences, proposed a fast inter mode decision algorithm for H.264/AVC able to decide the best mode in inter frame coding (P and B frames). However, this method suffers from a drawback, since it requires an edge image for texture information and a different image for the temporal stationary characteristics. This proposal is able to obtain on average a time reduction of 30%, when compared with the JM 5.0 reference encoder.

Grecos and Yang in [Grecos and Yang 05] presented a fast inter mode decision algorithm for P slices, which exploits spatio–temporal neighbourhood information jointly with a set of SKIP mode conditions for an enhanced SKIP mode selection. In a first step, they select some MBs to be coded as SKIP using spatio–temporal information. The idea of using spatio-temporal neighbourhood information is based on the observation that the areas of a picture that consist of MBs encoded using SKIP mode slowly change over time. In a second step, they use the conditions presented in [Lee and Jeon 03] in order to identify more MBs suitable to be coded using the SKIP mode. Finally, for the remaining MBs, they try to predict their respective modes while trying to avoid as much computation as possible. Experimental results achieve a 35–58% reduction in execution time, when compared with the JM 8.2 reference encoder.

In 2006:

Kuo and Lu in [Kuo and Lu 06] presented a SKIP mode decision algorithm, which is able to reduce the huge complexity resulting from the variable block size ME algorithm. This algorithm decides the SKIP mode based on the MVs of 8x8 blocks, which are calculated in advance. In contrast with previous proposals, this algorithm does not need predefined thresholds. On average, this algorithm is able to obtain a time reduction of 27%, when compared with the JM 9.2 reference encoder.

Nieto et al. in [Nieto et al. 06] presented a fast mode decision algorithm based on *Peak Signal Noise Ratio* (PSNR) predictions. First of all, the algorithm performs the SKIP and DIRECT modes, and then decides if more modes are needed. The decision is based on the generation of predictions of the expected PSNR of the current frame, which are calculated using the distortion costs obtained for SKIP and DIRECT modes, and the distortion costs obtained for previous frames. On average, this proposal obtains a time reduction of 36%, when compared with the JM 10.1 reference encoder. A bit rate increment of 1.78% and a PSNR decrement of 0.140 dB are also reported.

Ri and Ostermann in [Ri and Ostermann 06a] proposed a method to predict the best coding mode of an MB by using the best mode and the RD cost of neighbouring MBs (in time and space). The algorithm is based on the correlation of neighbouring MBs. In order to mitigate error propagation, an exhaustive mode decision is periodically carried out. This method can save up to 53% of the total encoding time, when compared with the JM 10.1 reference encoder. This paper shows a comparison with [Chang et al. 04], outperforming its results in terms of time savings and RD distortion.

Ri and Ostermann in [Ri and Ostermann 06b] proposed an improvement to the algorithm previously presented in [Ri and Ostermann 06a]. They improved the algorithm in order to take into account when the optimal mode for a concrete MB is different from the best mode of its neighbours. This method outperforms the previously presented method in execution time (i.e. can save up to 57%), but not in coding efficiency.

Kuo and Chan in [Kuo and Chan 06] proposed a method which determines a suitable inter prediction mode according to the motion field distribution and to the correlation inside an MB. The motion field distribution is obtained after applying a DBS over the sixteen 4x4 partitions into which an MB can be divided. The algorithm, by analysing the correlation between the sixteen MVs obtained, is able to carry out the mode decision. The algorithm is tested against the Fast FS algorithm implemented in JM 9.8 and the total execution time is 2.33 times shorter (time reduction of 57%), while, on average, it obtains a bit rate increment of 1%.

In 2007:

Lin et al. in [Lin et al. 07] presented an algorithm which discards some inter and intra modes by using a threshold. The algorithm exploits the correlation of co–located MBs in previously coded frames. In contrast with previous proposals, this proposal uses an

adaptive threshold based on the statistical dependency of co–located MBs. This algorithm does not exploit the correlation of MBs located in the same frame. The algorithm is tested against the well–known EPZS algorithm [Tourapis 02] implemented in JM 11, and it obtains a time reduction of 43%, while on average obtaining a bit rate increment of 1.5%.

Wang et al. in [Wang et al. 07] proposed a fast mode decision algorithm, when the RD optimization is turned on, by using the MB motion characteristics. First of all, the residual of the MB is obtained by applying the ME algorithm, then the residual is carefully analysed in order to obtain the MB motion characteristics. The residual is obtained and analysed using 8x8, 4x4, and 2x2 blocks. This division into blocks aims at extracting the edge directions within the MB. The proposal is tested against the Fast FS algorithm implemented in JM 8.6. Experimental results achieve a 40–63% reduction in execution time, with a bit rate increment of up to 6.7%.

Zhan et al. [Zhan et al. 07] presented a fast mode decision algorithm based on the correlation between MBs. The algorithm dynamically defines a set of candidate modes depending on the correlation of co–located MBs in the reference frame. This method can save up to 65% of the total encoding time when compared with the FS algorithm implemented in JM 10.1.

In 2008:

Bystrom et al. [Bystrom et al. 08] proposed a method to early detect the SKIP mode by using Bayesian networks. The Bayesian network requires some thresholds to determine the most suitable mode classification. These thresholds depend on the QP used and on the activity factor, which can be easily obtained. This approach obtains the best results for low–motion video sequences, since the probability of SKIP mode is high. The evaluation reports time savings ranging from 5% to 80% when compared with the FS algorithm implemented in JM 9.0, with a bit rate increment of up to 1%.

Zhan et al. in [Zhan et al. 08] presented a novel fast intra and inter mode decision algorithm for H.264/AVC. The proposed algorithm reduces the number of calculations carried out in both intra and inter predictions. First, the authors analyse the correlation between MBs, and define the mode decision depending on the distance between the current MB and the neighbouring MBs. Finally, the algorithm performs an MV adjustment in order to detect and properly encode objects which are in movement. This paper includes a comparison against [Grecos and Yang 05], outperforming its results with the same encoding conditions. The evaluation reports time savings of 50% when compared with the FS algorithm implemented in JM 10.1.

In 2009:

Cai et al. [Cai et al. 09] presented a fast ME algorithm. First, the algorithm selects some inter prediction modes which are not suitable to be used in homogeneous regions. Then, a hierarchical block matching scheme and a spatial neighbour searching scheme

are used to find the best MV with full–pixel accuracy. Finally, a direction–based rule is used to reduce the number of candidate positions with sub–pixel accuracy. However, this algorithm uses seven thresholds, one for each of the seven MB inter prediction modes, which are empirically obtained using the sequences used in the performance evaluation (the algorithm is trained for these sequences). Experimental results show a time reduction of up to 88% in the ME time, when compared with the JM 10.2 reference encoder. It also reports a bit rate increment of up to 0.5%.

Liu et al. in [Liu et al. 09] proposed an inter mode decision algorithm based on the motion homogeneity evaluated on a normalized MV field, which is generated using the MVs from the 4x4 partition. Three directional motion homogeneity measurements derived from the MV field are used to determine the candidate modes. Experimental results show a time reduction of up to 40% when compared with the JM 9.4 reference encoder. It also reports a bit rate increment of up to 2.7%. This article also includes a comparison against the proposal presented in [Wu et al. 05], outperforming its results in terms of execution time but not in terms of coding efficiency.

In 2010:

Hsia and Hung in [Hsia and Hung 10] proposed a fast ME algorithm for multiple reference frames. The algorithm is based on combining some well–known search algorithms such as FS, TSS and DBS. A control flow is proposed to select the search algorithm depending on video features. Moreover, it defines an adaptive search area adjustment to cover the motion displacement to achieve higher accuracy. The evaluation reports speed–ups ranging from 6x to 15x when compared with the FS algorithm implemented in JM 8.6, using 5 reference frames.

In 2011:

Husemann et al. in [Husemann et al. 11] presented an optimized high performance small diamond topology zonal search motion prediction algorithm. The algorithm is an adaptation of the DBS on which only three new checking positions are added on each algorithm iteration. Moreover, the SAD calculation is based on internal orthogonal 64-bit vector operations called *Linear Vectors*, which are particularly useful for Intel and AMD PC platforms. The proposed algorithm has worse coding efficiency than the *Unsymmetrical Multi-Hexagon Search* (UMHexagonS) algorithm implemented in JM16.2 (up to 0.3 dB), but on average is 5 times faster.

In 2012:

Hilmi et al. in [Hilmi et al. 12] proposed a method that reduces the number of candidate modes using direct information from co–located MBs. The proposed algorithm tries to detect moving objects. Depending on a previously performed motion complexity analysis,

the authors divide the available modes into three categories: SKIP mode; 16x16, 16x8 and 8x16 modes; 8x8, 8x4, 4x8 and 4x4 modes. As a consequence the algorithm does not completely evaluate all inter prediction modes. Experimental results show a time reduction of up to 78%, while reporting bit rate increments of up to 1%.

3.2. Faster Inter Prediction algorithms for the MVC Extension of H.264/AVC - Soft–Computing techniques

The MVC extension of the H.264/AVC standard was completely finalized in November 2009. Therefore, the proposals related to the MVC extension are recent. In fact, the MPEG issued the call for proposals in October 2005. The major focus of all of them was how to deal with the inter–view dependencies and how to remove the redundancies between adjacent views.

In 2006:

Merkle et al. in [Merkle et al. 06] presented a coding scheme aimed at improving the quality of multi–view video sequences by using H.264/AVC codecs. First of all, the authors analysed the temporal and inter–view dependencies. Then, by exploiting the spatio–temporal correlations obtained in the analysis, they defined the coding scheme. The proposed scheme is based on hierarchical B frames to exploit both temporal and inter–view dependencies. Experimental results report PSNR gains of up to 3.2 dB when compared with simulcast.

Lai and Ortega in [Lai and Ortega 06] proposed a fast predictive search algorithm to reduce the complexity in MVC. The authors proposed to predict the ME using the DE or vice versa. In this way, the algorithm obtains good candidate vectors to perform the estimation on the other field with very low complexity. Then, the algorithm carries out a reduced estimation using a search window with up to 9x9 candidate positions. Simulation results show that ME generally provides better block matching estimation than DE, therefore the best option is to predict the DE using the ME.

In 2007:

Lu et al. in [Lu et al. 07] proposed a DE technique to accelerate the disparity search by using epipolar geometry. Epipolar geometry has been widely studied in computer vision [Hartley and Zisserman 04] and is the only geometry constraint between a stereo pair of images of a single scene. The proposed epipolar–geometry–based DE can greatly reduce the search region and effectively track large and irregular disparity, which is very common in multi–view scenarios. Experimental results show a speed–up of up to 12x when compared with the fast full search DE algorithm and a speed–up of up to 3x when compared

with the UMHexagonS DE algorithm [Rahman and Badawy 05] (both algorithms available in JM 10.1), while reporting a bit rate increment of up to 1.6%.

Li et al. in [Li et al. 07] presented a fast inter prediction algorithm for both ME and DE. First, the prediction type is selected depending on the reference frames, as regions with fast motion are best handled with inter–view predictions (DE). On the other hand, homogeneous and stationary regions are best handled with temporal predictions (ME). The reason is that regions with fast motion may be predicted using small block sizes and large MVs, which decreases coding efficiency. Then, some unuseful search regions in the view direction are discarded from being analysed, based on the displacement of the cameras which recorded the 3D scene. Finally, a fast mode decision algorithm is performed based on the previously determined prediction type, applying the ME or DE processes only to a subset of the available inter prediction modes. Experimental results show a time reduction of up to 69% for the total encoding time, while reporting a bit rate increment of up to 4.2%.

In 2008:

Ding et al. in [Ding et al. 08] proposed a content–aware prediction algorithm for the inter–view mode decisions. The proposed algorithm is able to save unnecessary computation by exploiting the correlation between the different views in MVC. The ME modes and their corresponding MVs are predicted by using the DE and the coding information of neighbouring views. Therefore, ME computational complexity can be greatly reduced, since some MBs may be early identified as SKIP, INTRA or DE modes. Experimental results show a time reduction of over 98% for the ME time, with a quality loss of up to 0.06 dB.

Han and lee in [Han and Lee 08] presented a fast mode decision algorithm for MVC. The proposed algorithm uses a global disparity vector and a frame segmentation algorithm to identify the changes among the different viewpoints (cameras). The available modes in MVC are divided into two categories: the first category is composed of the ME and DE modes, and the second one is composed of the remaining modes (INTRA, SKIP and DIRECT). The mode decision algorithm selects and carries out one of these two categories. Experimental results report on average a time reduction of 40% with a PSNR degradation of about 0.05 dB.

In 2009:

Huo et al. in [Huo et al. 09] presented a scalable prediction structure for MVC in which the inter–view prediction may be disabled if the inter–view redundancy can be almost eliminated by temporal and intra prediction. In this way, the time employed for DE may be saved by reducing encoder complexity. The authors use a hierarchical GOP pattern and propose not to carry out the DE in one or more of the highest temporal layers of the hierarchical GOP pattern, since they observed that the percentage of temporal predictions

increases with an increment in the temporal layer index. Experimental results report on average a time reduction of 30.60% and a bit rate increment of 1.55, when not performing the DE in the highest level in a GOP pattern of five levels (GOP 16).

Shen et al. in [Shen et al. 09] proposed a fast DE and ME algorithm based on the correlation between the inter prediction modes and motion homogeneity. MBs with homogeneous motion usually select temporal predictions with large block sizes, and MBs with complex motion usually select inter–view predictions or temporal predictions with small block sizes. The proposal uses the spatial properties of the motion field, which is generated by obtaining the motion information of the 4x4 inter prediction mode. On average, experimental results show a time reduction of 63% and a bit rate increment of up to 2%.

In 2010:

Deng et al. in [Deng et al. 10] proposed a fast ME and DE algorithm, which exploits the correlation between temporal and inter–view reference frames. First, the algorithm obtains a predictor by taking into account the correlation of motion and disparity vectors of neighbouring MBs. Then, an iterative search algorithm is run to find the optimal motion and disparity vectors. The iterative search algorithm is performed using small window sizes of 5x5, which is sufficient for maintaining the coding efficiency. The algorithm is only implemented for the 16x16 inter prediction mode. Therefore, it does not support variable block size ME and DE. Simulation results show that the overall average encoding time saving is 86%, while reporting a bit rate increment of up to 6%.

Zeng et al. in [Zeng et al. 10] presented a fast mode decision algorithm called *Mode–correlation–based early Termination* (MET). First, for each MB the SKIP mode is evaluated. Then, if the encoding cost of the SKIP mode is below an adaptive threshold, the other modes may be discarded from being checked. The threshold is based on the mode correlation of adjacent MBs in the current view and in neighbouring views. Experimental results report a reduction in computational complexity of about 65%, while on average reporting bit rate increments of 1%.

In 2011:

Zhang et al. in [Zhang et al. 11] proposed a *Fast Multi–reference Frame Selection Algorithm* (FMFSA) for hierarchical bi–directional prediction in MVC. Due to the correlations within an MB, there is a high probability that the different inter prediction modes available in MVC select the same reference frame and direction as the 16x16 mode would select. Therefore, this algorithm carries out the 16x16 mode, and then only evaluates a sub–set of the reference frames (typically 1), thus saving computation. Experimental results report a reduction in computational complexity ranging from 68% to 79%, while on average reporting bit rate increments of 0.5%.

Shen et al. in [Shen et al. 11] presented a low complexity mode decision algorithm to reduce the complexity of ME and DE in MVC. The proposed algorithm is based on

four decision techniques: early SKIP mode decision, adaptive early–out termination, fast mode size decision, and selective intra coding in inter frames. The authors evaluate each technique separately, and as a final step evaluate the results of the overall algorithm. On average, the experimental results report a reduction in computational complexity of about 76%, while on average reporting bit rate increments of 1.3%.

In 2012:

Deng et al. in [Deng et al. 12] proposed an iterative search strategy designed to speed up the uni–directional prediction and a selective bi–directional prediction algorithm. The uni–directional prediction algorithm is carried out in some iterations, the best vectors of each MB partition from one iteration become the base vectors for the next one (similar to an MVP). Then, by extracting some motion information from the uni–directional MVs and disparity vectors, the bi–directional prediction algorithm is executed. Simulation results show that the speed of the proposed algorithm on average is 88 times above that of the FS algorithm in JMVC, while reporting bit rate increments of up to 2%.

Liu et al. in [Liu et al. 12] presented a high-speed mode decision algorithm for the inter–view predictions of multi-view video sequences. Some candidate modes are discarded from being checked to reduce the encoding cost calculations and an early stop mode decision is made by using multiple parameters related to the estimation of the final optimal mode. These parameters are: the MB residual and the temporal, spatial and inter–view correlations. On average, the experimental results report a reduction in computational complexity of about 92%, while reporting bit rate increments of up to 2.5%.

3.3. GPU–based H.264/AVC Inter Prediction algorithms

As mentioned in Chapter 2, in the last decade the processing power of GPUs has grown faster than the processing power of contemporary CPUs. Therefore, GPUs have attracted a lot of attention. In 2003, *Ian Buck* in a presentation at *Graphics Hardware 2003* and in [Buck et al. 04] estimated that the peak performance of the Nvidia GeForce FX 5900 was nearly 20 Gflops, which was equivalent to a 10-GHz Pentium 4 processor. This was the starting point at which GPUs started moving from being exclusively used for graphics applications to being used for high performance computing.

In the framework of video processing using GPUs, there are not many solutions available in the literature. Some of them use graphic APIs for programming the GPU, such as OpenGL or Microsoft Direct X. However, with the emergence of Nvidia CUDA in 2006, a new door was opened for high performance computing using GPUs. The related proposals are organized by the year of publication.

In 2004:

Kelly and Kokaram in [Kelly and Kokaram 04] proposed the use of computer graphics hardware for fast image interpolation. Basically, the authors, by using the OpenGL API, implemented a bilinear interpolation algorithm needed to obtain sub–pixel accurate MVs. However, the GPU does not produce identical images to those produced on the CPU. These errors appear due to round–off problems and precision issues of the GPU texturing units. Moreover, the GPU used in this work uses a PCI bus, which becomes the bottleneck of the system. The results show a speed–up of up to fourfold, when sub–pixel ME is performed. The authors do not include the algorithm in any H.264/AVC encoder.

In 2006:

Ho et al. in [Ho et al. 06] presented an H.264/AVC ME algorithm using programmable graphics hardware. The algorithm is run at MB–level to overcome the dependencies between adjacent MBs (MVPs), and is implemented using the OpenGL API. The MBs are transferred to the GPU when they are going to be encoded, dividing them into sixteen 4x4 blocks. The reference frame is transferred to the GPU texture units at the beginning of coding each frame. Then, the FS ME algorithm is executed for all inter prediction modes in parallel. The authors also provide a mechanism to adjust the arithmetic intensity of the algorithm depending on the processing capabilities of the GPU used. Experimental results show that the proposal is 10 times faster that an optimized SIMD implementation, which they also developed. The authors do not include the algorithm in any H.264/AVC encoder.

In 2007:

Lee et al. in [Lee et al. 07] proposed a multi-pass and frame parallel algorithm to accelerate some ME tools available in an H.264/AVC encoder, by using the OpenGL API. They unroll and rearrange the multiple nested loops of the ME algorithm using a multi-pass method; IME is implemented using a two–pass method; FME is implemented using a six–pass method; and ME using multiple reference frames is implemented using a two–pass method. The algorithm is implemented using a multi–pass method because the total number of instructions is higher than the GPU instruction limit. However, the algorithm does not support variable block size ME and it is not integrated into any H.264/AVC encoder. Experimental results show that the proposed GPU–based ME algorithm achieves a speed–up of up to 56x, when using a search range of 16 and 3 reference frames.

In 2008:

Ryoo et al. in [Ryoo et al. 08] presented one of the pioneering approaches that make use of CUDA to accelerate general purpose applications. They presented certain optimization principles of multi–threaded GPUs using CUDA, for a wide variety of applications. They

tackle the challenge of striking the optimum balance between resource utilization and the number of simultaneous active threads, but also try to reorder the memory accesses to combine them into adjacent memory transactions (coalescing accesses). One of the applications used in this work is the *H.264ref* benchmark from the *Standard Performance Evaluation Corporation* (SPEC) CPU2006 [SPEC 06], obtaining a speed–up of 20x for the parallelized part of the H.264/AVC encoder, which means a speed–up of 1.5x for the complete encoder.

Chen and Hang in [Chen and Hang 08] proposed an implementation of the H.264/AVC ME algorithm using CUDA. The algorithm is based on an efficient block-level parallel algorithm for the variable block size ME in H.264/AVC, supporting IME and FME. They decompose the H.264/AVC ME algorithm into 5 steps, so they can achieve highly parallel computation. However, they use many sequential kernels, thus reducing the parallel computations and increasing the memory transfers between the GPU and its DRAM memory. Moreover, the algorithm is not included in any H.264/AVC encoder, so it is not possible to show any result concerning RD performance. The proposal does not deal with the problem of MVPs in H.264/AVC, so the cost metric cannot be accurate. Experimental results show a speed–up of up to 12x for the ME module implemented in this work.

Kung et al. in [Kung et al. 08], by using the OpenGL API, proposed a GPU–based ME for H.264/AVC rearranging the encoding order of 4x4 blocks. Unlike previous works, this proposal deals with the dependencies between adjacent MBs (MVP) and can be applied to FS ME and to other fast ME algorithms, such as TSS. The algorithm, instead of using the raster scan order to code the 4x4 blocks, uses a complete 4x4 diagonal block list from top–left to bottom–right, so the parallelism depends on the video sequence resolution. The proposal does not support variable block size ME, and is only implemented for the 4x4 mode. Experimental results show that this proposal is 45 times faster than their CPU implementation using FS, and is 15 times faster when compared with the TSS algorithm.

In 2009:

Schwalb et al. in [Schwalb et al. 09] presented a GPU–based ME approach for the purpose of H.264/AVC video coding using the DirectX 9 API. To exploit the available parallel computing power and memory bandwidth of modern GPUs, a small diamond search is adapted to their programming model. The authors cannot obtain the maximum expected performance because this algorithm uses irregular/random memory access patterns and the unavailability of real while loops inhibits the possibility of an early–out termination strategy, which means that unnecessary operations and memory transactions are carried out. Experimental results show a speed–up factor ranging from 1.5 to 3 when compared with UMHexagonS [Rahman and Badawy 05] implemented in JM 9.0. The authors do not provide a proper RD performance analysis.

Momcilovic and Sousa in [Momcilovic and Sousa 09] proposed a scalable parallelization approach for H.264/AVC ME on multi–core CPUs, and its implementation on GPUs using CUDA. The algorithm obtains the MVs by applying data reusing techniques and exploiting

the computing capabilities of GPUs. The proposal assumes no dependencies between adjacent MBs, which is not true for H.264/AVC ME. The MV (0,0) is used as MVP, i.e. the proposal does not use MVPs. Therefore, the RD performance can be considerably affected. Each frame is divided into an equal–size number of MBs depending on the number of SPs available in the GPU, and the MBs are assigned to a specific SP. The MBs mapped to each SP are sequentially estimated since the proposal copies the MB itself and its associated search area candidates to the SP shared memory, which is a limited resource. Moreover, this procedure limits the maximum dimensions of the search area. Experimental results show that this proposal can achieve real–time ME when using 4-CIF sequences, 5 reference frames and 8 as search range. The authors do not provide a proper RD performance analysis.

Pieters et al. in [Pieters et al. 09] presented a motion search architecture, capable of executing H.264/AVC ME, supporting variable size ME for multiple GPUs using CUDA. The motion search architecture processes MBs serially in the CPU threads (one CPU thread is generated per slice within a frame), but uses previously calculated data in parallel on the GPU. The motion search algorithm iterates from fast, but inaccurate MVs, to slow but accurate MVs, depending on the available time. The algorithm uses a spiral search pattern, dividing the complete search area into multiple iterations. These iterations make it possible to implement an early–out termination behaviour based on the MVs obtained in previous iterations, and to deal with the dependencies between adjacent MBs because approximate MVPs can be calculated using the best MVs of previous iterations. Experimental results show that real-time ME for 720p sequences (1280x720 pixels) is achieved using a search range of 16 running on a system with four GPUs. However, the algorithm is not integrated into any H.264/AVC encoder so the RD performance is not evaluated and some drift errors are introduced. It does not use real MVPs, and uses a predefined threshold value for the early–out termination mechanism.

In 2010:

Cheung et al. in [Cheung et al. 10] proposed a GPU–based version of the *Simplified Unsymmetrical Multi-Hexagon Search* (smpUMHexagonS) ME algorithm [Yi et al. 05], implemented in JM 14.2 using CUDA. This algorithm uses several techniques in order to save computation, including MVPs, different search patterns (cross, hexagon and diamond) and an early–out termination mechanism. The authors divide the current frame into multiple tiles, such that each tile is processed by a single GPU thread and different tiles are processed by different independent threads concurrently on the GPU. The number of tiles used affects the algorithm's performance, both in terms of execution time and in terms of RD performance. The more tiles used, the faster the algorithm as greater parallelism can be achieved. However, RD performance is worse because less MBs are predicted using real MVPs. The authors evaluate the algorithm using different numbers of tiles, ranging from 3 to 3600 tiles. Experimental results show that the algorithm is up to 3 times faster than the reference algorithm when coding 720p video sequences (1280x720 pixels), while

it reports significant bit rate increments of up to 12% with a penalty in quality of up to 0.4dB depending on the sequence and the tile length.

In 2011:

Lu and Hang in [Lu and Hang 11] presented a GPU–based inter prediction algorithm for MVC. The algorithm is implemented using integer precision; no sub–pixel ME or DE is carried out using the GPU. The algorithm is based on the *Parallel Hierarchical One-Dimensional Search* (PHODS) proposed by *Chen et al.* in [Chen et al. 91], which is a fast ME algorithm designed to reduce the number of sequence steps and search points. The proposed algorithm is adapted to have a regular flow control and a fixed number of instructions for each iterative process. In this way, each iterative process can be independently executed by independent GPU threads. Experimental results show that the proposed algorithm can obtain a speed–up of 9–20x for integer ME and DE algorithms when compared with the reference EPZS [Tourapis 02] algorithm implemented in JMVC, while reporting a coding quality loss of up to 0.026 dB.

Massanes et al. in [Massanes et al. 11] proposed a fast implementation of the block-matching ME algorithm for multiple GPUs using CUDA. The authors proposed a very simple ME algorithm aimed at real–time encoding which does not support variable block–size ME. A GPU thread is generated for each search position for each block in a frame, and each of them computes the encoding costs as the SAD. The block and search area sizes are set depending on the sequence resolution. This proposal aims at real–time encoding and does not take RD performance into account. The authors do not use MVPs and do not analyse the RD performance of the encoded video sequences. Experimental results report a speed–up of about 200x when compared with a custom FS algorithm (no early–out termination mechanism implemented), and report real–time ME at 30 fps for 720x480 format when using 2 Tesla C1060 GPUs, the blocks and the search area are set to 5% and 10% of the image size, respectively.

Monteiro et al. in [Monteiro et al. 11] presented a way to partition the steps of the FS block matching algorithm in the CUDA architecture. The proposed scheme uses a library called *Thrust* [Thrust 11] for data manipulation between the CPU and the GPU, and the algorithm is divided into two steps: SAD values calculation and SAD comparison. The algorithm is developed for 4x4 blocks as the basic unit for block matching, using search areas ranging from 24x24 to 256x256 pixels. The algorithm is not integrated into any H.264/AVC encoder and as a consequence no RD performance is evaluated. Experimental results report a speed–up of about 200x when compared with a custom FS algorithm (no early–out termination mechanism implemented) when using a search area of 64x64 pixels.

In 2012:

Pieters et al. [Pieters et al. 12] presented a multi–view encoding framework for high quality video game remote rendering. The key–aspect is that most video games make use

of a 3D engine, which is typically accelerated on a GPU, containing information on the composition of the 3D scene and its objects as well as their motion. This paper shows how to analyse and extract that information, aiming to exploit it in order to successfully offload the most time consuming tasks of an MVC encoder. They propose two approaches: the first one completely eliminates motion search from the encoding process and is called the *forced* version; and a second one on which a refinement search process is carried out, and this one is called the *forced+refinement* version. Experimental results show that the forced version reports a speed–up of 11x when compared with FS and the forced+refinement version reports a speed–up of 3.2x (the maximum search area used is four times smaller than the one used by FS). On the other hand, the algorithms report PSNR decrements ranging from 0.5 dB to 1.0 dB.

3.4. H.264/AVC Inter Prediction algorithms for other parallel architectures

Other proposals for other parallel and/or heterogeneous architectures can be found in the literature. These architectures include multi–core processors, FPGA prototypes and *Very Large Scale Integration* (VLSI) prototypes.

In 2004:

Chen et al. in [Chen et al. 04] presented a multi–level parallelization of an H.264/AVC encoder for the Intel Hyper–threading architecture. The authors parallelize the H.264/AVC encoder using the OpenMP programming model, which makes it possible to leverage the advanced compiler technologies in the Intel C++ compiler. First of all, the authors present the design considerations, and then two multi–level data partitioning methods. Experimental results show speed–ups ranging from 3.7x to 4.5x. The quality and bit rate of the encoded video sequence is the same as the reference implementation. However, the frames are divided into multiple slices in order to obtain as much data parallelism as possible, which means that the dependencies between MBs are broken, increasing the bit rate by a factor of up to 1.5.

In 2005:

Tseng et al. in [Tseng et al. 05] presented an FPGA prototype which supports variable–block–size ME for multiple reference frames. It supports P and B slices, quarter–pixel accuracy using a 6–tap filter for interpolation, and weighted bi–directional prediction. The algorithm pays special attention to the memory system design in order to optimize memory usage and bandwidth.

In 2008:

Zheng et al. in [Zheng et al. 08] presented a VLSI architecture of MC for multiple standards including MPEG–2 [ISO/IEC 99], H.264/AVC, and the Chinese *Audio Video Standard* (AVS) [AVS-Group 05]. The proposed design has an MB–level pipelined structure which consists of an MVP calculation unit, a cache-based fetch, and a pixel interpolation unit. The proposed architecture exploits the parallelism in the MC algorithm to accelerate the processing speed, and uses the dedicated design to optimize memory accesses. The MVP unit can cover all prediction for the three standards with a small error; the cache–based fetch can save up to 25% of the required memory bandwidth; and the pixel interpolation unit is designed to avoid redundant calculations. Experimental results show that real time is achieved when decoding HDTV 1080i (1920x1080 pixels) video sequences.

In 2011:

Ruiz and Michell in [Ruiz and Michell 11] presented an H.264/AVC VLSI processor chip for IME. The proposal is based on the FS block matching algorithm and uses a 2D array to obtain a high data reuse of the search area. A three–directional scan is supported in the 2D array, reducing memory accesses, and thus saving clock cycles. The algorithm obtains the motion information for the smallest MB partitions (4x4), and using this information it obtains the motion information of other higher partitions. The authors do not deal with MVPs in order to remove the dependencies between adjacent MBs. Finally, a custom mode decision is carried out taking into account the estimated cost of each MB partition. The proposed VLSI processor chip has enough processing capacity for 1080p (1920x1080 pixels) real–time IME with a search range of 16.

Kalva et al. in [Kalva et al. 11] presented an overview of parallel programming. More specifically, languages and tools for parallel programming are introduced within the scope of multimedia applications. First of all, the computing capabilities of modern general purpose processors are analysed keeping in mind the special requirements of multimedia applications. Multimedia applications are the kind of applications that can benefit from parallel implementations and are expected to drive the demand for multi–core processors and parallel implementations. Then, the parallelism in multimedia applications is analysed. Finally, some languages and tools for parallel programming are presented as a solution for speeding up multimedia applications. The most common tools are the OpenMP API, the SIMD programming model, the CUDA GPU programming model and the usage of cloud computing for multimedia applications.

In 2012:

Atitallah et al. [Atitallah et al. 12] proposed an FPGA–based implementation of the *Line Diamond Parallel Search* (LDPS) algorithm [Werda et al. 07], with variable block size ME. The FPGA prototype is based on two modules: the loading module and the search module. The first one is used to load the MB and the search area. The second

one finds the suitable MV according to the LDPS algorithm. The proposed algorithm is implemented to support variable block size ME, but does not support all the modes defined by the H.264/AVC standard. The proposed FPGA processor chip has enough processing capacity for 1080p (1920x1080 pixels) real–time ME with a search range of 8.

3.5. Power and Energy consumption

In recent years, research efforts have not been exclusively focused on performance (speed–up) and on the migration from the sequential programming model to the parallel programming model. Studies that deal with power and energy consumption are becoming very common, which is a new requirement from processor manufacturers.

In 2005:

Feng et al. in [Feng et al. 05] presented a prototype for direct and automatic profiling of power consumption for non–interactive scientific applications on high–performance distributed systems. The prototype, for each available node of the distributed system, profiles all components in the system (CPU, memory, disk, and interconnection network). Experimental results indicate that the profiling is often regular, depending on the application's characteristics and on the problem sizes. If the number of nodes increases, the energy consumption increases too, but the performance does not always improve. This aspect suggests smart schedulers could be used to optimize the energy consumption while maintaining performance. The prototype is limited by the sampling frequency of the hardware and by its scalability. It cannot be used for profiling large parallel systems.

In 2008:

Rofouei et al. in [Rofouei et al. 08] presented an experimental investigation into the power and energy cost of GPU operations, and a cost/performance comparison against a CPU-only system. The investigation was carried out using a platform called *LEAP–Server*, which is a novel architecture that incorporates standard server functionalities with high fidelity real–time energy monitoring of the system components, such as the CPU, GPU, motherboard and *Random Access Memory* (RAM). Simulation results show that using a GPU results in energy savings if the performance gains are above a threshold.

In 2009:

Huang et al. in [Huang et al. 09] presented an empirical study of the performance, power, and energy characteristics of GPUs for scientific computing. The authors migrate a sequential CPU code to a hybrid CPU+GPU environment. The sequential code is a widely-used biological application, called *GEM*. It calculates the electrostatic potential generated

inside a macromolecule, and is based on numerical solutions of the Poisson-Boltzman equation. First, a multi–threaded version of the code is implemented, and then a GPU version is implemented. The study shows the impact of changing the number of thread blocks and the number of threads within each thread block of the GPU implementation. Experimental results show that the GPU implementation delivers an energy-delay product that is multiple orders of magnitude better than the CPU implementation. The analysis also shows that the GPU implementation is more efficient as many thread blocks and threads within each thread block are configured in both execution time and energy consumption.

Ma et al. in [Ma et al. 09] presented a statistical power consumption analysis of a GPU-based framework. The authors recorded the GPU power consumption and the workload of different test programs, and developed the power consumption model. Then, by using the generated model, they can predict the energy consumption for a specific GPU program. However, this model cannot accurately predict energy consumption peaks, and the authors cannot find an explanation.

In 2010:

Nagasaka et al. in [Nagasaka et al. 10] proposed another statistical approach for estimating the power consumption of GPU kernels. The authors use performance counters, which can be easily obtained in CUDA applications. The counters used are: the number and size of global memory transactions, the number of local memory transactions, the number of divergent and non–divergent branches, the number of instructions used and the number of shared memory bank conflicts. The evaluation is carried out using 49 testing programs available in the CUDA *Software Development Kit* (SDK) and in the *Rodinia suite*, obtaining an average error ratio lower than 5%. However, this model fails when running GPU programs which use texture memory accesses because of the lack of performance counters for monitoring texture accesses.

In 2011:

In 2011, *Anzt et al.* in [Anzt et al. 11] presented an analysis and optimizations of power consumption for the iterative solution of sparse linear systems on multi–core and many–core platforms. First of all, the authors analyse the computational and power consumption requirements of some iterative linear solvers applied to sparse systems. Then, they evaluate the gains yield by modifying the voltage and frequency of the CPU (*Dynamic Voltage and Frequency Scaling* (DVFS) method). DVFS potentially lowers the energy consumption when the GPU is in execution (CPU is idle) and the total execution time does not vary. However, the CPU continues in a highly–inefficient busy–wait status, as the CPU executes a polling algorithm in order to know when the GPU has finished its execution. Finally, they propose to set the CPU to sleep when the GPU is working, so they avoid the execution of the polling algorithm. Experimental results show that the DVFS technique can obtain an energy consumption reduction of up to 5%, and using both techniques jointly they can obtain an energy consumption reduction of up to 25%.

Chapter 4

P frame Inter Prediction

IN this chapter, the proposal developed for P frames is presented. First of all, a brief introduction to the proposed and the reference inter prediction algorithms is given. Then, the proposal is described in detail. Finally, it is evaluated, including a comparison with the most recent and prominent related proposals.

4.1. Introduction

ME sequentially obtains the motion information (encoding costs) for all available MB partitions and sub-partitions, for all MBs in a frame. For each MB partition and sub-partition, a search area is defined and a search algorithm is executed. The search algorithm looks for a region that minimizes the differences between the current partition or sub-partition and the chosen region.

The main challenge of this approach is to efficiently support the tree-structured MC algorithm executed in the H.264/AVC JM 17.2 reference software encoder [JVT 11] for P frames. The idea is to concurrently obtain the motion information for all MBs at the beginning of coding each P frame. Additionally, in order to work with a GPU, some data must be explicitly allocated to its DRAM in advance, since the CPU and the GPU have their own independent memory spaces.

4.2. Proposed algorithm

As was mentioned in Section 2.3.3, in a GPU there are several kinds of memory, and each kind of memory is suitable for different kinds of memory accesses. The data which does not change during the complete encoder execution is transferred once per sequence and is allocated to the GPU constant memory: frame dimensions, search range, search area distribution, QPs. On the other hand, before coding each frame, the frame to be encoded and the reference frame(s) are transferred to the GPU texture memory. Texture memory

is selected since it is bigger than constant memory, and there is a sufficient amount of memory to allocate the frames; both kinds of memory have associated read-only memory caches. Finally, some structures, which should be filled in by the proposed algorithm, are allocated to GPU global memory: namely encoding costs and MVs.

Figure 4.1 shows the activity diagram of the proposed algorithm, which is executed once per frame. The MVs for all MB partitions and sub–partitions in a frame are obtained in parallel. Then, the MC mechanism is sequentially executed for all MB partitions and sub–partitions available in a complete frame. The algorithm is divided into three main parts: MVP calculation, IME and FME.

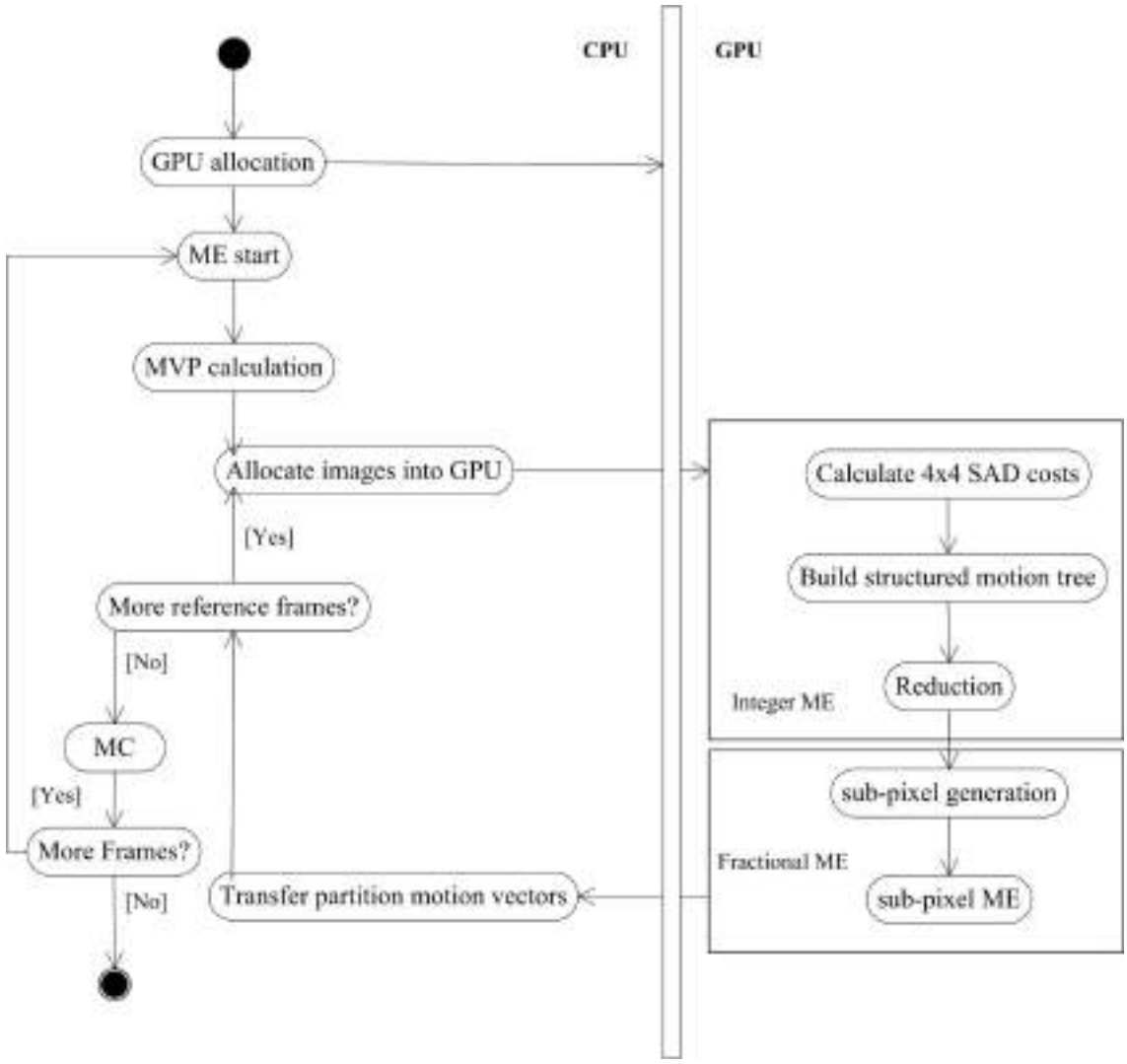
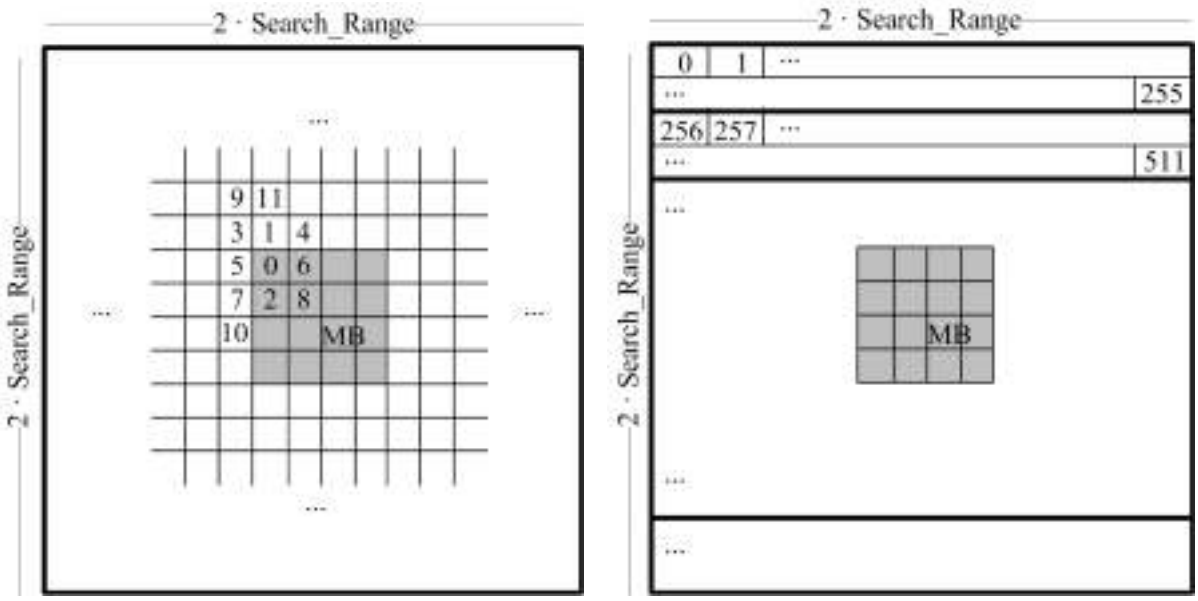


Figure 4.1: Activity diagram of the proposed encoder.

As a starting point, the FS algorithm is considered, since it is the one which provides the best coding efficiency. The reference FS algorithm evaluates all search area positions following a spiral pattern, as described in Figure 4.2a. This search area distribution starts

checking the positions close to the center of the search area and ends with the ones which are further away. This distribution is appropriate for exploiting the early–out termination mechanism implemented in the H.264/AVC JM 17.2 reference encoder, since it is more likely to find the best prediction near the center of the search area than near the search area borders. However, this distribution does not have locality in the memory accesses, which is very important for an efficient GPU execution.

The proposed algorithm does not implement the early–out termination mechanism, since a complete frame is going to be concurrently evaluated on the GPU, such that each search area position is going to be evaluated by a different GPU thread. Therefore, a new search area distribution must be defined. Figure 4.2b shows the proposed search area distribution, which follows a raster scan order; position 0 corresponds to the top-left corner of the search area, and the positions are distributed in rows. The position values shown in Figure 4.2 are an example, and they depend on the search range used.



(a) Spiral pattern (b) Proposed pattern

Figure 4.2: Search area distribution.

4.2.1. MVP calculation

As was described in Section 2.1.2, it is known that encoding one MV for each MB partition can increase the number of bits required to encode a frame, especially if small partition sizes are chosen. However, it is also known that MVs from neighbouring partitions are often highly correlated and each MV can be predicted using the MVs of neighbouring partitions. Therefore, an MVP can be calculated, and the differences between the MV obtained and the MVP calculated are encoded.

The reference FS algorithm calculates one MVP for each MB partition and sub–partition of each MB in a frame, the encoding costs of all of them using Equation 4.1, and then encodes the differences between the MV obtained and the MVP calculated. The encoding costs are obtained as follows:

$$Cost = SAD + \lambda \cdot R, \tag{4.1}$$

where SAD is the metric used to calculate the differences (see Section 2.1.2 for more details), λ is an encoder parameter which depends on the QP used and R is the number of bits required to encode the MV minus the MVP. However, when the GPU is in execution it is not possible to calculate the MVP for a given partition because the MVs of neighbouring partitions are being calculated concurrently.

In the literature, a solution to solve the problem of the MVP calculation can be found. [Pieters et al. 09] proposed carrying out the ME in a number of steps, and after each step the MVPs are updated. This solution requires extra synchronization points and the execution time may be affected. However, the proposed algorithm uses two different MVPs and is executed in one iteration: one MVP to obtain the best MV of each MB partition and sub–partition, and one MVP to encode the best MV obtained.

At the beginning of coding each P frame, the MVPs for all MBs in a frame are calculated and transferred to the GPU global memory. The MVPs are calculated using the motion information from the previously encoded frame, unlike in the sequential execution in which the motion information of neighbouring MBs can be used. The proposal uses as MVP the 16x16 MV of the MB located in the same position in the previously encoded frame. This MVP is used to calculate the encoding costs (Equation 4.1). Moreover, a common search is defined for each MB, i.e. all MB partitions and sub–partitions of a given MB use the same MVP to locate its search area (in the reference algorithm different search areas may be used) and to calculate the encoding costs.

On the other hand, when the MC mechanism is carried out the real MVPs can be calculated (MC is sequentially carried out on the CPU, and the MVs of neighbouring MBs are available). Therefore, the encoding costs of the best MVs obtained for all MB partitions and sub–partitions are recalculated, taking into account the real MVP. As a consequence the encoding costs are more accurate and a better mode decision can be carried out.

Note that, despite these real MVPs, the final encoding costs may be different from the ones calculated by the reference algorithm because the best MVs obtained may be different (the best MV is selected by using a non real MVP). Finally, if the best MV obtained differs, the MVP calculated for other MBs will also differ, and the differences will propagate.

4.2.2. Integer Motion Estimation

Once all the required data is allocated and transferred to the GPU memory, the IME is carried out. As depicted in Figure 4.1, IME is divided into three steps which need to be executed sequentially following a highly–parallel procedure by using the GPU: to

calculate 4x4 SAD costs, to build the structured motion tree and to perform a reduction to obtain the best MV for all MB partitions and sub–partitions with integer–pixel accuracy. Moreover, IME is carried out using two GPU kernels. Each thread block of the first GPU kernel obtains and reduces the motion information of 256 adjacent search area positions of a given MB; if more than 256 search area positions are defined in the search area, a second kernel performs a final reduction.

4x4 SAD costs

In order to execute the first GPU kernel, a GPU thread is generated for each search area position (there are $(2 \cdot Search\_Range)^2$ search area positions per MB) and 256 threads are grouped into a thread block. At the beginning of executing each thread block of the first GPU kernel, all threads cooperate to allocate to the GPU shared memory the MB which is going to be estimated and the portion of the search area needed to estimate the MB. Figure 4.3 shows an example of the search area allocated to shared memory, when the Search_Range is set to 32. The horizontal width of the search area is $2 \cdot Search\_Range = 64$, and as a consequence, $256/64 = 4$ complete search area rows are going to be evaluated by each thread block. However, fifteen horizontal and vertical extra pixels must be allocated to the GPU shared memory in order to have all the required data. The pixels are stored on the GPU shared memory using a 4–byte data structure, since the GPU shared memory can be simultaneously accessed using 4–byte memory transactions when no bank conflicts occur (see Section 2.3.3).

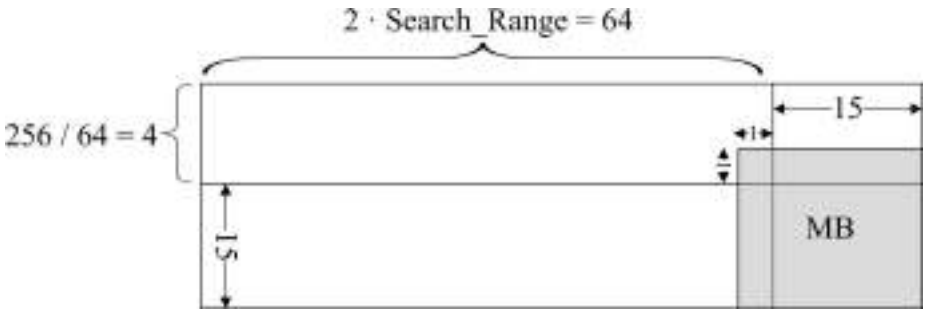


Figure 4.3: Example of the search area allocated to shared memory.

The search area is located using the MVPs calculated before executing this first GPU kernel. This allocation is possible due to the redistribution of the search area presented above (Figure 4.2b), otherwise correlative positions would not be adjacent in memory.

The SAD calculation is carried out in 4x4 blocks, as is depicted in Figure 4.4. Each GPU thread obtains the SAD costs of the 16 4x4 blocks into which an MB can be divided (b_0 to b_{15} in Figure 4.4). Contiguous threads are mapped to contiguous search area positions (P_0 to P_{255} in Figure 4.4). Note that Figure 4.4 is a graphical description of the calculation for the complete MB, which is carried out by a number of thread blocks. Therefore, each column of the matrix shown in Figure 4.4 corresponds to one position checked inside the

search area, which is calculated by the same GPU thread. Intermediate results of this step are stored in GPU registers for faster memory access in the next step.

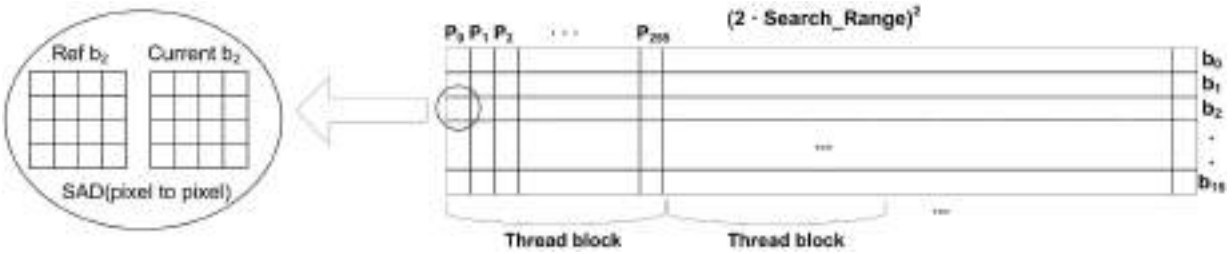


Figure 4.4: Obtaining 4x4 SAD costs.

This step calculates the SAD costs of the 4x4 sub–partitions, it does not calculate their encoding costs.

Structured motion tree

The second step uses the information previously obtained in the first step to obtain the encoding costs of all MB partitions and sub–partitions. It is executed by the same GPU kernel as in the first step, so each thread block will obtain the motion information of 256 adjacent search area positions.

Figure 4.5 shows how to obtain the SAD costs of a given position (Pst) for the different MB partitions and sub-partitions, starting with the 4x4 SAD costs. This procedure is possible because the proposed algorithm uses a common search area for all MB partitions and sub–partitions within a frame. In order to obtain the motion information for the eight 4x8 and for the eight 8x4 sub–partitions, it is only necessary to add two 4x4 SAD costs for each of them, e.g. by adding #0 and #2 SAD costs from the 4x4 sub–partition, the #0

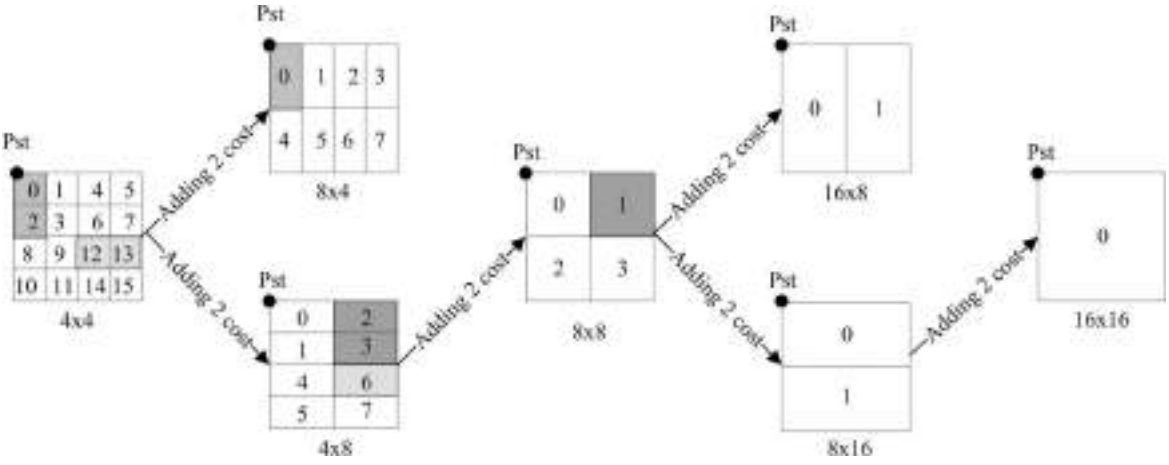


Figure 4.5: SAD cost building for different MB partitions.

SAD cost for the 8x4 sub–partition is obtained (see shaded boxes); to obtain the four 8x8 partitions it is only necessary to add two 4x8 SAD costs, and so on.

Before storing the motion information, the final encoding costs are calculated. The encoding cost calculation is based on Equation 4.1. Therefore, the encoding costs to store the MVs ($\lambda \cdot R$) are added to the previously calculated SAD costs.

The encoding costs are stored on the GPU shared memory using a 4–byte data structure. This structure uses two unsigned short data containing the encoding cost and its associated position (2 bytes each). This allocation is chosen to avoid shared memory bank conflicts.

Reduction

Finally, the last step performs a reduction to obtain the best MV of each MB partition and sub–partition. This step is typically carried out using two GPU kernels. The same GPU kernel which carried out the previous steps performs a reduction by a factor of 256. That is, the GPU kernel returns the best MV of each MB partition and sub–partition of the 256 search area positions evaluated by each thread block configured. Additionally, an independent GPU kernel performs the final reduction, if needed.

Figure 4.6 shows one iteration of the reduction procedure. After the second step, there are 256 positions per MB partition and sub–partition, which is the initial size (S) of the reduction procedure. In order to reduce the information, m comparisons are needed for each row used in shared memory (b_0 to b_{N-1}). Note that m is half of the remaining positions in any of the eight iterations needed to reduce from 256 positions to 1. As a result, on each

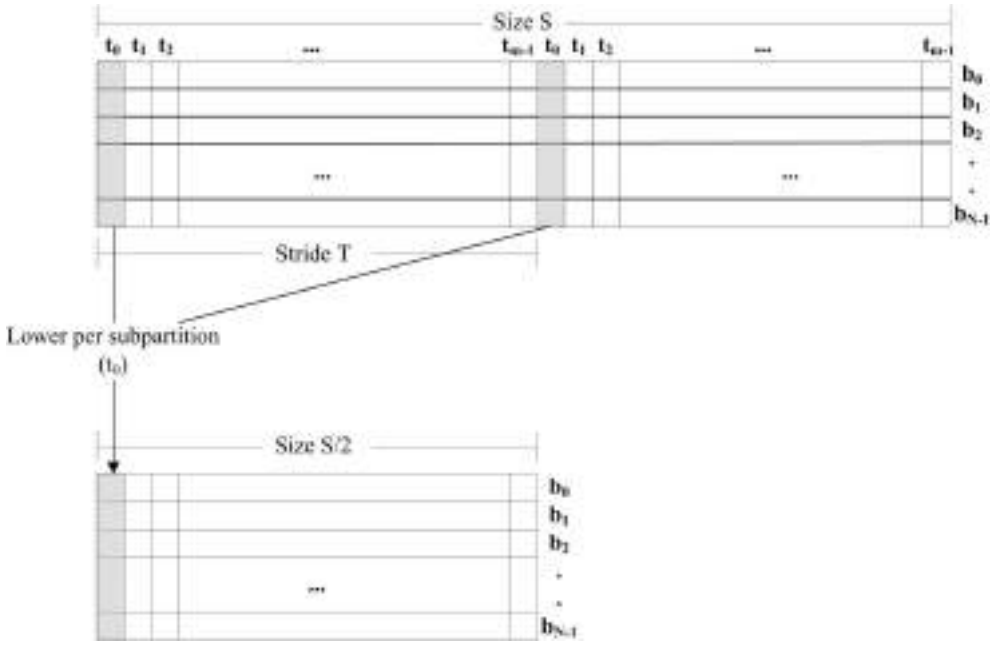


Figure 4.6: Binary reduction scheme.

algorithm iteration $m \cdot N$ comparisons are carried out. These comparisons are uniformly distributed on the available threads.

In order to complete the reduction process, eight iterations are needed, starting with 256 (2^8) encoding costs per partition and sub–partition, and finishing with one encoding cost, where the size (S) is reduced by half in each iteration. These reductions are performed with S = 256, 128, 64, 32, 16, 8, 4, and 2 using T strides, such that T=S/2. These strides, jointly with the threads distribution, are chosen to avoid shared memory bank conflicts. The code for the eight iterations is unrolled to avoid unnecessary loop climbs. Intermediate results are allocated to shared memory, updating the shared memory, while final results are allocated to global memory for the final reduction procedure.

Additionally, if there are more than 256 positions inside the search area of each MB, an independent GPU kernel is carried out, applying the same reduction procedure described above. This second kernel starts with $(2 \cdot Search\_Range)^2/256$ positions and finishes with the best MV with integer accuracy for each MB partition and sub–partition for each MB in a frame.

4.2.3. Fractional Motion Estimation

Once IME has finalized its execution, FME is carried out. FME can be viewed as a refinement of IME. However, before starting the FME algorithm the reference frame must be sub–sampled to quarter–pixel accuracy. That is, each pixel is converted into sixteen sub–pixels, which means that the image size is multiplied by four in each dimension (see Figure 4.7). These sixteen sub-pixels are classified into full-pixel, sub-pixels with half-pixel accuracy and sub-pixels with quarter-pixel accuracy.

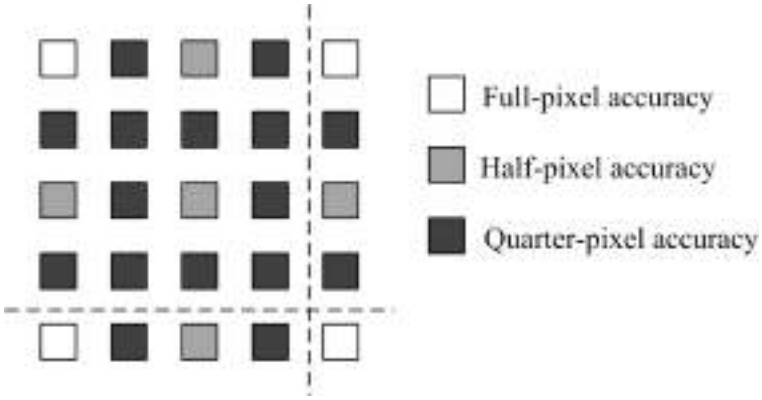


Figure 4.7: Sub-pixel generation.

Experimental results show that it is faster to obtain the sub–pixels by using GPU kernels than transferring the images with sub–pixel accuracy from CPU memory (the images are sixteen times bigger). As was explained in Section 2.1.2, half–pixels are obtained by means of a six–tap FIR filter and quarter–pixels by means of a bilinear filter. Remember that there are dependencies in the sub–pixel generation procedure and three GPU kernels are

used to obtain all sub–pixels. The half–pixel located in the middle of Figure 4.7 is obtained using other half–pixels and the quarter–pixels are obtained using half–pixels. One GPU thread per pixel in the original frame is generated.

FME is carried out in two steps: half–pixel refinement and quarter–pixel refinement. The algorithm for both steps is the same but applied over different pixels. The best full-pixel MV (obtained by the IME algorithm) is used as the starting point for the half–pixel refinement and the algorithm searches over half–pixels to obtain the best MV with half–pixel accuracy. The best half-pixel MV (obtained by the half–pixel refinement) is used as the starting point for the quarter–pixel refinement and the algorithm searches over quarter–pixels to obtain the best MV with quarter–pixel accuracy. A graphical description can be found in Figure 4.8. The best MV with quarter–pixel accuracy of each partition and sub–partition is the required output of the proposed algorithm. Each step is carried out using one GPU kernel.

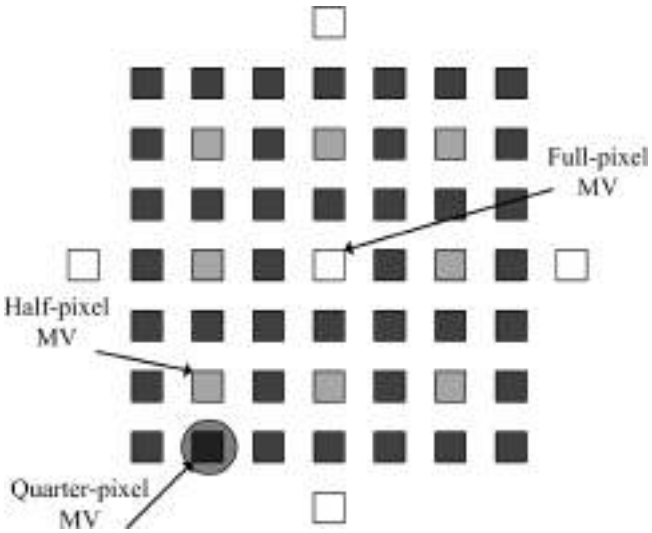


Figure 4.8: Sub-pixel MV refinement.

FME is carried out using 4x4 blocks like IME, but in this case the number of search area positions is considerably smaller, and each thread only calculates the cost of one 4x4 block. If we pay attention to Figure 4.8, eight half–pixels surround each full–pixel and eight quarter–pixels surround each half–pixel. Therefore, nine search area positions are defined for FME. The encoding cost of the starting point must be recalculated since the QP may vary, and as a consequence, the final encoding cost may also vary. The encoding costs are calculated using Equation 4.1, but the SADT metric using Hadamard transforms is used instead of SAD. The SADT metric is more complex than the SAD metric, but provides greater coding efficiency. By default, the H.264/AVC JM 17.2 reference encoder uses the SAD metric for IME and the SADT metric for FME.

The MB is divided into sixteen 4x4 blocks and each one takes as a starting point the appropriate MV. For the 16x16 partitions all blocks within an MB take the same MV, while for the 4x4 sub–partitions all blocks within an MB may take different MVs.

As a consequence of this divergence, the proposed algorithm cannot reuse the motion information of the smallest partition to obtain the encoding costs of the higher partitions, and the reference pixels cannot be allocated to shared memory (the texture memory is used). Each partition has different starting points (full-pixel MV or half-pixel MV), the costs must be recalculated for each partition and sub–partition and are merged using GPU atomic operations allocating the resulting encoding costs to shared memory. No merging is necessary for the 4x4 sub–partitions and the costs of all 4x4 blocks are added for the 16x16 partitions. The same reduction procedure used for IME is used to obtain the final MV.

4.3. Performance evaluation

In this section, the results of applying the proposal described in this chapter are presented. First of all, the encoding conditions and the metrics used to evaluate the proposal are shown. Then, the evaluation is carried out, including a comparison with related proposals.

4.3.1. Encoding conditions

In order to evaluate the proposed inter prediction algorithm developed for P frames, it has been integrated into the H.264/AVC JM 17.2 reference encoder [JVT 11]. The H.264/AVC encoding parameters used for the evaluation are those included in the Baseline profile of the mentioned reference encoder. However, some parameters are changed in the configuration file and these are described below:

- The number of reference frames is set to 1 in order to keep the complexity as low as possible. An analysis using more reference frames is possible, since the algorithm can iterate over multiple reference frames. The conclusions obtained will be the same regardless of the number of reference frames configured. At the end of this chapter, a comparison with related proposals which uses multiple reference frames is shown.
- RD–Optimization is disabled for the same reason as the number of reference frames. However, in Appendix A the algorithms proposed in this thesis are evaluated enabling this option.
- The GOP pattern configured is 1 I frame followed by 11 P frames (I11P).
- The tests are carried out with popular sequences in QCIF (176x144 pixels), CIF (352x288 pixels), VGA format (640x480 pixels), 720p format (HD, 1280x720 pixels) and 1080p format (full-HD, 1920x1080 pixels). The first frame of the sequences used in the evaluation is shown in Figure 4.9, in Figure 4.10 and in Figure 4.11, and these sequences have different characteristics (content and movement features). Note that the sequences used for the evaluation of QCIF, CIF and VGA format are the same.

(a) Canoe

(b) Fast food

(c) Football

(d) M. calendar

(e) Racing

(f) Scrolltext

(g) Skyline

(h) Softfootball

Figure 4.9: QCIF, CIF and VGA sequences used to evaluate the proposal.

(a) City

(b) Crew

(c) Dolphins

(d) Harbour

(e) Mobcal

(f) Night

(g) Park run

(h) Shields

Figure 4.10: 720p sequences used to evaluate the proposal.

(a) Crowd

(b) Ducks

(c) Into tree

(d) Old town

(e) Park joy

(f) Pedestrian

(g) Riverbed

(h) Tractor

Figure 4.11: 1080p sequences used to evaluate the proposal.

- The frame rate parameter is set to 30 for QCIF, CIF and VGA format (30 Hz), and it is set to 50 for 720p and 1080p format (50 Hz).

- The QP is varied between 28, 32, 36 and 40 according to [Bjontegaard 01], [Sullivan and Bjontegaard 01] and [JVT Test Model Ad Hoc Group 03].

- The search range is set to 16, which means 1024 positions inside the search area of each MB partition, when encoding QCIF and CIF video sequences. For the other formats, the search range is set to 32, which means 4096 positions inside the search area of each MB partition.

- The proposed inter prediction algorithm is tested against two search algorithms implemented in the H.264/AVC JM 17.2 reference encoder: FS [Richardson 10] and UMHexagonS [Rahman and Badawy 05].

In order to make a proper comparison, an unmodified H.264/AVC JM 17.2 reference encoder is run on the same machine as the H.264/AVC JM using the proposed algorithms, with the same encoding configuration and with no calls to the GPU.

The following development environment is used to test the proposed inter prediction algorithm: the host machine used is an Intel Core i7 running at 2.80 GHz with 6GB of DDR3 memory. The GPU used is an NVIDIA GTX480 with an NVIDIA driver with CUDA support (260.19). The operating system for this scheme is Linux Ubuntu 10.4 x64 with GCC 4.4. Finally, Table 4.1 shows the main GPU features.

Table 4.1: GTX480 features.

Characteristic	Value
Compute capability	2.0
Global memory	1.5 GB
Number of multiprocessors	15
Number of cores	480
Constant memory	64 kB
Shared memory per block	48 kB
Registers per block	32,768
Max. active threads per multiprocessor	1,536
Clock rate	1.40 Ghz

4.3.2. Metrics

Different metrics have been used to evaluate the proposal. These metrics are the *Time Reduction* (TR) and speed–up, RD function, ΔPSNR and Δbit rate, mode decisions, and power and energy consumption. These metrics are defined below.

Time reduction and speed–up

In order to evaluate the time saved by the proposed inter prediction algorithm, two metrics are going to be used. The TR metric shows the percentage of time saved by the proposal and is based on Equation 4.2. The speed–up metric shows how many times the proposed algorithm is faster than the reference algorithms and is based on Equation 4.3.

$$TR(\%) = \frac{T_{JM} - T_P}{T_{JM}} \cdot 100, \tag{4.2}$$

$$speed - up = \frac{T_{JM}}{T_P}, \tag{4.3}$$

where T_{JM} denotes the coding time used by the H.264/AVC JM 17.2 reference encoder, and T_P is the time taken by the JM 17.2 encoder using the proposed algorithm. T_P also includes all the computational costs for the operations needed in order to prepare the information required by our proposal, including memory allocations on the GPU and memory transferences from/to GPU memory.

In the evaluation, these two metrics are applied at different levels. The first one shows the TR and speed–up for the complete H.264/AVC encoder, while the second one shows them for the time spent exclusively by the functions where the proposal is applied.

RD function

RD provides the theoretical bounds on the compression rates that can be achieved using different methods. In RD theory, the rate is usually understood as the number of bits per data sample to be stored or transmitted. The notion of distortion is subject to on–going discussion. In the simplest case, and the one actually used in most cases, the distortion can be simply defined as the mean squared error of the difference between the input and the output signals. In the definition of the RD function, the PSNR is the distortion for a given bit rate. The average global PSNR is based on Equation 4.4.

$$PSNR = \frac{4 \cdot PSNR_Y + PSNR_U + PSNR_V}{6} \tag{4.4}$$

The Luminance PSNR_Y is multiplied by four, since the YUV input files are sampled using the 4:2:0 sampling format, which is composed of four 8x8 blocks for the luminance component and of only one 8x8 block for each chrominance component.

ΔPSNR and Δbit rate

The experiments are carried out on the test sequences using four quantization parameters, namely, QP = 28, 32, 36 and 40. The detailed procedures for calculating bit rate and PSNR differences can be found in the work by Bjøntegaard [Bjontegaard 01], and make use of Bjøntegaard and Sullivan's common test rule [Sullivan and Bjontegaard 01]. These procedures have been recommended by the JVT Test Model Ad Hoc Group [JVT Test Model Ad Hoc Group 03]. The YUV files used for comparing the PSNR results are the original YUV file at the input of the H.264/AVC JM 17.2 reference encoder and the one obtained after decoding the H.264/AVC video stream using the H.264/AVC JM 17.2 reference decoder.

A positive Δbit rate means that the number of bits required to encode the video sequence has increased, and vice versa. A negative ΔPSNR means that the quality of the encoded video sequence has decreased, and vice versa.

Mode decisions

Another way to analyse how the proposed algorithm works is by showing the MB mode decision image generated. The MB modes generated by the H.264/AVC JM 17.2 reference encoder and by the H.264/AVC JM 17.2 encoder using the proposed algorithm are compared in order to measure the accuracy of the MB mode classification tree. A grid image showing the MB modes overlaid on a corresponding frame is used to visually compare the MB mode classification.

Power and energy consumption

With the objective of sampling the current consumed at a given time by the whole test computer including the *Power Supply Unit* (PSU), we developed a data logger device capable of transmitting this data in a reliable and easy way to a different computer. The principle of this device is based on the analysis of the magnetic field produced by an electric current flowing through a straight conductor and it is capable of sampling and reconstructing the resulting wave, whatever its form, and processing it in order to obtain an average value.

We use a sensor capable of translating these magnetic changes extracted from the supply cable into a proportional voltage level to simplify the procedure. The sensor used is the Allegro Microsystems Inc A1301, a continuous time ratio–metric linear Hall-effect sensor with a sensitivity of 2.5 mV/G. The sensor response is given by the following equation:

$$v = y_0 + \alpha \cdot I, \tag{4.5}$$

where v is the voltage at the output of the sensor, $y_0 = 2.4610$ V, $\alpha = 0.4185$ V/A, and I is the current flowing through the conductor we want to measure. The constants y_0 and α are obtained by linear regression from a test in which the output voltage of the A1301

sensor was measured for different current values. Both output voltage and current values were measured using high precision digital multi–meters.

The sensor output is tied to the analog-to-digital converter module of a Microchip PIC12F683 micro–controller, which is responsible for sampling the voltage data with a resolution of 10–bit length (4.88 mV per bit), and sending it to the user by using an RS232 interface which adjusted its data transfer rate to 115200 bauds.

In order to transfer the sampling data to the host computer we used the Future Technology Devices International Ltd FT232 chip, which makes it possible to adapt the RS232 interface to the *Universal Serial Bus* (USB) interface. The data is received through a virtual *Component Object Model* (COM) port which is created when the installation of the sensor device in the host computer is completed. The software used to collect the data is the Eltima software RS232 data logger. Figure 4.12 shows the basic scheme of our profiler system. Note that two computers were used: the first one (tested computer) is the computer on which the H.264/AVC encoder is running and whose energy usage needs to be measured; the second one (data computer) is used in order to collect and process the data from the sensor device.

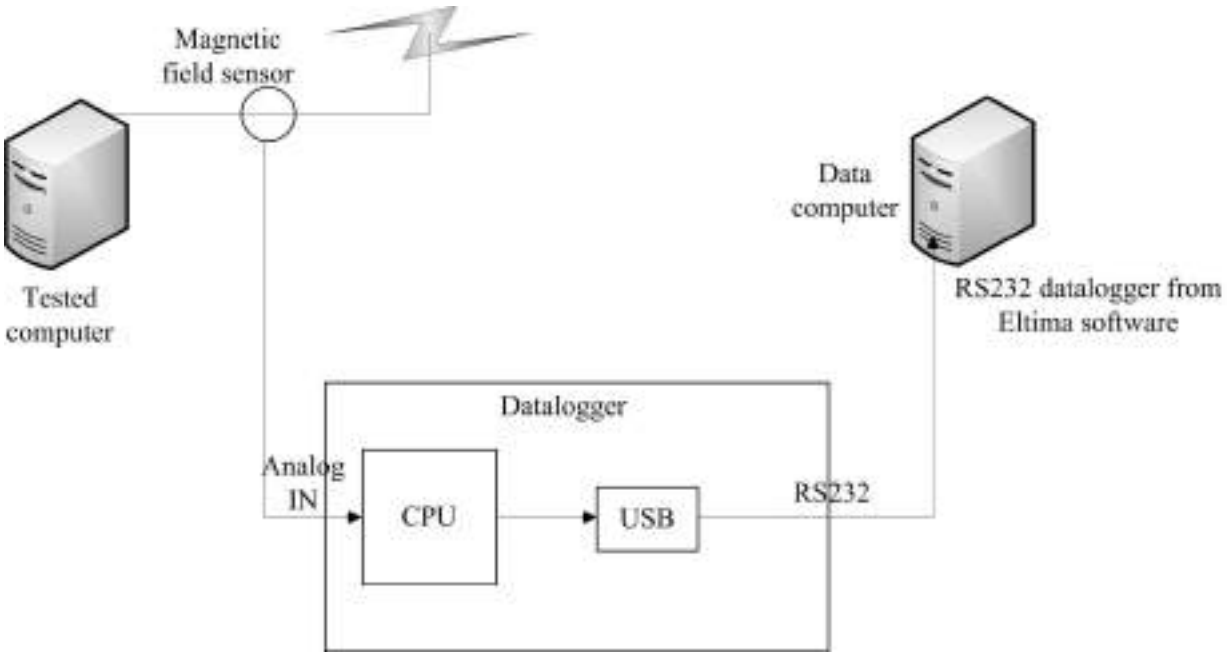


Figure 4.12: Profiler system scheme.

Once the voltage data is obtained, it must be processed to obtain the *Root Mean Square* (RMS) or quadratic mean value for a given cycle. When working with periodic and symmetrical waves, such as an electric wave, the RMS value must be calculated in cycles. Our data logger device takes 2k samples per second, that is, 40 samples per cycle (1 cycle = 20 msec). Thus, we calculate the V_{RMS} in a cycle with Equation 4.6.

$$V_{RMS} = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{n} v_i^2} = \sqrt{\frac{v_1^2 + v_2^2 + \cdots + v_N^2}{N}} \tag{4.6}$$

Operating with Equation 4.5 and using the V_{RMS} value of the voltage previously obtained, the current flowing through the conductor at any cycle can be obtained as:

$$I = \frac{V_{RMS} - 2.4610}{0.4185} \tag{4.7}$$

Once the data for the current intensity is known, we can apply equation 4.8 to obtain the power consumed by the whole system at any cycle, where V is equal to 230 V. Finally, we can obtain the energy consumption applying equation 4.9, where T is the time consumed by the H.264/AVC encoder and P is the average power consumed.

$$P = V \cdot I \tag{4.8}$$

$$E = P \cdot T \tag{4.9}$$

4.3.3. Results

This section presents the results obtained when coding different video sequences using the proposed inter prediction algorithm developed for P frames.

Timing results

From Table 4.2 to Table 4.8 the timing results of the proposed algorithm are presented. The proposed algorithm is tested against two search algorithms implemented in the H.264/AVC reference encoder, so the results are divided into two main parts: the comparison against the FS algorithm and the comparison against the UMHexagonS algorithm. Moreover, the timing results are further divided, showing the timing results focusing exclusively on the proposed algorithm (*ME module* column), and the timing results of the complete H.264/AVC encoder (*Complete encoder* column).

Before showing the results, we would like to mention certain characteristics of each reference algorithm used in the evaluation. The execution time of the FS algorithm is highly dependent on the video sequences and may even double, affecting the TRs and speed–ups obtained. The FS algorithm is implemented using an early–out termination mechanism, which is able to discard search area positions before being completely checked, saving computation. The execution time of the UMHexagonS algorithm is also sequence dependent, but less than that of the FS algorithm. The UMHexagonS algorithm is carried out in a number of steps, and the number of steps depends on the video sequence being

Table 4.2: Timing results of the proposed encoder for QCIF sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 16								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
Canoe	98.39	62.02	89.69	9.70	90.05	10.05	57.68	2.36
Fast food	97.88	47.07	87.00	7.70	88.14	8.43	53.51	2.15
Football	98.40	62.75	89.66	9.67	89.62	9.64	56.36	2.29
M. calendar	98.25	57.16	88.66	8.81	87.48	7.99	51.22	2.05
Racing	98.20	55.70	88.83	8.95	88.54	8.72	54.59	2.20
Scrolltext	97.58	41.38	85.50	6.90	85.82	7.05	49.02	1.96
Skyline	97.41	38.62	84.46	6.43	84.45	6.43	45.97	1.85
Softfootball	98.46	65.08	90.06	10.06	89.84	9.84	56.91	2.32
Average	98.07	51.89	87.98	8.32	87.99	8.33	53.16	2.13

Table 4.3: Timing results of the proposed encoder for CIF sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 16								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
Canoe	98.82	84.44	90.48	10.51	93.25	14.82	62.30	2.65
Fast food	98.34	60.39	87.42	7.95	91.40	11.62	56.80	2.32
Football	98.75	80.21	90.10	10.10	92.48	13.30	59.84	2.49
M. calendar	98.64	73.48	89.14	9.21	90.37	10.38	53.17	2.14
Racing	98.85	87.31	90.84	10.91	92.66	13.62	60.46	2.53
Scrolltext	97.80	45.53	84.17	6.32	88.36	8.59	49.48	1.98
Skyline	97.92	47.99	84.85	6.60	87.69	8.12	47.83	1.92
Softfootball	98.90	91.18	91.08	11.21	93.16	14.63	61.81	2.62
Average	98.50	66.83	88.51	8.70	91.17	11.33	56.46	2.30

encoded. On the other hand, the execution time of the proposed algorithm is almost constant, since it always evaluates all possible search area positions in parallel.

Table 4.2 and Table 4.3 show the timing results when coding the same QCIF and CIF video sequences using 16 as search range. The results show that the proposed algorithm outperforms both algorithms in both resolutions. When coding QCIF video sequences, the H.264/AVC encoder using the proposed algorithm on average is 8.3x times faster than the

H.264/AVC encoder using the FS algorithm (TR of nearly 88%), and is 2.1 times faster than the H.264/AVC encoder using the UMHexagonS algorithm (TR of over 53%). When coding CIF video sequences, the H.264/AVC encoder using the proposed algorithm on average is 8.7x times faster than the H.264/AVC encoder using the FS algorithm (TR of over 88.5%) and is 2.3 times faster than the H.264/AVC encoder using the UMHexagonS algorithm (TR of nearly 56.5%).

Table 4.4 shows the average execution time increments of the reference algorithms (FS and UMHexagonS) and the average execution time increments of the proposed algorithm, when coding CIF video sequences instead of coding QCIF video sequences. The resolution of the video sequences is incremented by four (CIF format is four times bigger than QCIF), so a priori, the execution time should also increase by a factor of four. However, the expected execution time increment is not obtained.

Table 4.4: ΔTime from QCIF to CIF video sequences.

H.264/AVC JM 17.2 Main profile – Search range = 16		
Search algorithm	ΔTime	Expected ΔTime
FS	3.88	4
UMHexaonS	4.09	
Proposal	3.00	

The ΔTime obtained when using the FS algorithm is 3.88. It is lower than 4, which means that the early–out termination mechanism is able to save more computation when coding CIF video sequences than when coding QCIF video sequences.

The ΔTime obtained when using the UMHexagonS algorithm is 4.09. It is greater than 4, which means that the algorithm requires more iterations to find the best MVs when coding CIF video sequences than when coding QCIF video sequences.

The ΔTime obtained when using the proposed algorithm is 3.00. It is considerably lower than 4, which means that the algorithm obtains a higher instruction throughput when coding CIF video sequences than when coding QCIF video sequences. In other words, when coding QCIF video sequences the maximum instruction throughput that this algorithm can obtain for the GPU used is not achieved. The instruction throughput has increased when the GPU's computational load has also increased.

As a final conclusion, the ΔTime obtained when using the proposed algorithm is lower than the ΔTime obtained when using the reference algorithms. So the speed–ups obtained are greater when coding CIF video sequences than when coding QCIF video sequences (see Table 4.2 and Table 4.3).

Table 4.5 shows the timing results when coding the same VGA video sequences used in the QCIF and CIF evaluation. However, in this case the search range is set to 32, increasing the number of search area positions by a factor of 4. In comparison with the

analysis shown above for QCIF and CIF video sequences, the GPU's computational load has increased not only because the sequence's resolution has increased, but also because the search area of each MB partition is 4 times bigger.

Table 4.5: Timing results of the proposed encoder for VGA sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
Canoe	99.17	119.90	95.85	24.08	90.19	10.19	65.17	2.87
Fast food	98.88	89.36	94.58	18.44	86.92	7.65	58.22	2.39
Football	99.15	117.28	95.77	23.62	88.43	8.64	60.97	2.56
M. calendar	99.11	112.59	95.45	21.99	85.20	6.76	53.30	2.14
Racing	99.25	132.53	96.23	26.52	89.78	9.78	64.12	2.79
Scrolltext	98.40	62.44	92.47	13.28	80.92	5.24	47.99	1.92
Skyline	98.53	68.02	93.05	14.40	79.07	4.78	45.15	1.82
Softfootball	99.26	135.63	96.27	26.81	90.24	10.25	65.02	2.86
Average	98.97	96.88	94.96	19.83	86.34	7.32	57.49	2.35

On average, the algorithm's speed–up is nearly 97x when compared with the FS algorithm (TR of nearly 99%), which means a speed–up of nearly 20x (TR of nearly 98%) for the complete H.264/AVC encoder. On the other hand, it obtains a speed–up of over 7x when compared with the UMHexagonS algorithm (TR of over 86%), which means a speed–up of over 2.3x (TR of nearly 55.5%) for the complete H.264/AVC encoder.

Table 4.6 shows the average execution time increments of the reference algorithms (FS and UMHexagonS) and the average execution time increments of the proposed algorithm, when coding VGA video sequences instead of coding CIF video sequences. The resolution of the video sequences is incremented by three (VGA format is three times bigger than CIF), so a priori, the execution time should also increase by a factor of three. Moreover, the search range is also incremented by two, which means that the search area is four times bigger. As a result, the expected ΔTime is twelve.

The ΔTime obtained when using the FS algorithm is 8.09. It is considerably lower than 12, which means that the early–out termination mechanism is able to save considerably more computation when coding VGA video sequences using 32 as search range than when coding CIF video sequences using 16 as search range.

The ΔTime obtained when using the UMHexagonS algorithm is 3.55. It is only a little greater than 3, which means that the algorithm is able to process a search area which is four times bigger using only a few extra iterations of the algorithm. Remember that VGA format is three times bigger than CIF.

Table 4.6: ΔTime from CIF to VGA video sequences.

H.264/AVC JM 17.2 Main profile		
Search algorithm	ΔTime	Expected ΔTime
FS	8.09	12
UMHexaonS	3.55	
Proposal	5.50	

The ΔTime obtained when using the proposed algorithm is 5.50. This is considerably lower than 12, which means that the algorithm obtains a considerably higher instruction throughput when coding VGA video sequences than when coding CIF video sequences. In other words, when coding CIF video sequences the maximum instruction throughput that this algorithm can obtain for the GPU used is far from being achieved.

The ΔTime obtained when using the proposed algorithm is lower than the ΔTime obtained when using the FS algorithm, so the speed–ups obtained are greater when coding VGA video sequences than when coding CIF video sequences. The ΔTime obtained when using the proposed algorithm is greater than the ΔTime obtained when using the UMHexagonS algorithm, so the speed–ups are lower (see Table 4.3 and Table 4.5).

Finally, Table 4.7 and 4.8 show the timing results when coding HD video sequences. Note that the video sequences analysed are different from the ones used for lower resolutions, so a comparison with previous results would not be fair.

Table 4.7: Timing results of the proposed encoder for 720p sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
City	98.90	91.28	94.52	18.25	79.34	4.84	44.96	1.82
Crew	98.93	93.42	94.70	18.86	82.69	5.78	50.01	2.00
Dolphins	98.74	79.40	93.73	15.94	81.98	5.55	47.95	1.92
Harbour	98.88	89.57	94.36	17.73	78.07	4.56	42.88	1.75
Mobcal	98.98	98.03	94.85	19.42	81.55	5.42	47.40	1.90
Night	98.75	80.16	93.79	16.11	78.40	4.63	43.60	1.77
Park run	99.23	130.52	95.95	24.67	84.59	6.49	51.66	2.07
Shields	98.94	93.97	94.61	18.55	82.59	5.74	48.84	1.95
Average	98.92	92.60	94.56	18.39	81.15	5.30	47.16	1.89

Table 4.8: Timing results of the proposed encoder for 1080p sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
Crowd	99.15	117.21	95.37	21.61	83.27	5.98	49.04	1.96
Ducks	99.31	145.90	96.12	25.80	84.25	6.35	49.47	1.98
Into tree	99.13	114.36	95.36	21.53	80.33	5.08	44.59	1.80
Old town	98.92	92.24	94.34	17.67	77.17	4.38	40.48	1.68
Park joy	99.25	133.87	95.86	24.13	85.18	6.75	51.75	2.07
Pedestrian	98.59	70.81	92.82	13.92	79.80	4.95	44.19	1.79
Riverbed	99.33	149.19	96.37	27.51	89.06	9.14	60.46	2.53
Tractor	98.84	85.95	93.66	15.77	85.86	7.07	52.80	2.12
Average	99.06	106.81	94.99	19.95	83.11	5.92	49.10	1.96

The algorithm's speed–ups range from 79x to 130x when compared with the FS algorithm for coding 720p video sequences, and from 70x to 145x when compared with the FS algorithm for coding 1080p video sequences. For both resolutions, the overall speed–ups obtained for the complete H.264/AVC encoder range from 13x to 27x.

When compared with the UMHexagonS algorithm, the algorithm's speed–ups range from 4.5x to 6.5x for coding 720p video sequences, and from 4.3x to 9.1x for coding 1080p video sequences. For both resolutions, the overall speed–ups obtained for the complete H.264/AVC encoder range from 1.6x to 2.5x.

RD results

From Table 4.9 to Table 4.13 the RD results of the proposed algorithm are presented and organized in a similar way to the previous section. The results are divided into two main parts: the comparison against the FS algorithm and the comparison against the UMHexagonS algorithm.

Table 4.9 and Table 4.10 show the RD results when coding the same QCIF and CIF video sequences using 16 as search range. The results show that the proposed algorithm obtains a slightly lower encoding efficiency than the FS algorithm (the RD degradation is negligible if the computational savings are taken into account), but it surpasses the encoding efficiency obtained by the UMHexagonS algorithm.

When compared with the FS algorithm for coding QCIF video sequences, the average bit rate increment obtained is 2.73%, while it is 2.46% for coding CIF video sequences. That is, the bit rate increments improve slightly when the resolution is increased. The bit rate increments are similar for all video sequences, ranging from 0.78% to 4.37%.

Table 4.9: RD results of the proposed encoder for QCIF sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 16				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
Canoe	2.39	-0.083	-0.69	0.027
Fast food	3.24	-0.114	-2.07	0.070
Football	4.37	-0.146	-0.93	0.030
M. calendar	1.25	-0.050	1.37	-0.054
Racing	4.21	-0.146	-4.63	0.157
Scrolltext	0.78	-0.031	-2.13	0.085
Skyline	2.03	-0.067	-0.43	0.013
Softfootball	3.54	-0.118	-0.71	0.022
Average	2.73	-0.094	-1.28	0.044

Table 4.10: RD results of the proposed encoder for CIF sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 16				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
Canoe	1.25	-0.044	-2.04	0.069
Fast food	3.01	-0.097	-5.20	0.166
Football	3.22	-0.103	-3.32	0.102
M. calendar	1.98	-0.078	1.45	-0.059
Racing	2.99	-0.108	-16.13	0.676
Scrolltext	1.15	-0.042	-5.72	0.217
Skyline	3.09	-0.098	-0.14	0.002
Softfootball	2.99	-0.095	-2.23	0.074
Average	2.46	-0.083	-4.17	0.156

On the other hand, when compared with the UMHexagonS algorithm for coding QCIF video sequences, the average bit rate decrement obtained is 1.28%, while it is 4.17% for coding CIF video sequences. That is, the bit rate decrements improve when the resolution is increased. The bit rate decrements are more uneven than when using the FS algorithm and range from 0.14% to 16.13%. Note that the M. calendar video sequence is a special

case in which the proposed algorithm is not able to outperform the encoding efficiency of the UMHexagonS algorithm.

Table 4.11 shows the RD results when coding the same VGA video sequences used in the QCIF and CIF evaluation. However, in this case the search range is set to 32, increasing the number of search area positions by a factor of 4. In this case, besides increasing the resolution of the video sequences tested, the search area is also increased. However, the behavior is similar to the one previously observed when coding QCIF and CIF video sequences.

Table 4.11: RD results of the proposed encoder for VGA sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 32				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
Canoe	0.18	-0.003	-3.14	0.099
Fast food	2.44	-0.063	-5.70	0.159
Football	2.20	-0.059	-3.84	0.103
M. calendar	2.36	-0.082	0.49	-0.019
Racing	1.90	-0.064	-15.25	0.561
Scrolltext	1.96	-0.066	-10.25	0.391
Skyline	3.52	-0.096	-2.60	0.073
Softfootball	1.22	-0.039	-2.72	0.080
Average	1.97	-0.059	-5.38	0.181

When compared with the FS algorithm for coding VGA video sequences, the average bit rate increment obtained is 1.97%, while the average bit rate decrement is 5.38% when compared with the UMHexagonS algorithm.

Finally, Table 4.12 and Table 4.13 show the RD results when coding HD video sequences. Note that the video sequences analysed are different from the ones used for lower resolutions, so a comparison with previous results would not be fair.

The bit rate increments, when compared with the FS algorithm, range from 0.90% to 2.59% for coding 720p video sequences, and from 1.20% to 4.49% for coding 1080p video sequences. Note that the Riverbed video sequence in 1080p format is a special case in which the proposed algorithm is able to slightly outperform the encoding efficiency obtained by the FS algorithm.

On the other hand, when compared with the UMHexagonS algorithm for coding 720p video sequences, the bit rate decrements range from 0.08% to 5.07%, and from 0.12% to 17.5% for coding 1080p video sequences. Note that Park run and Shields in 720p format,

Table 4.12: RD results of the proposed encoder for 720p sequences.

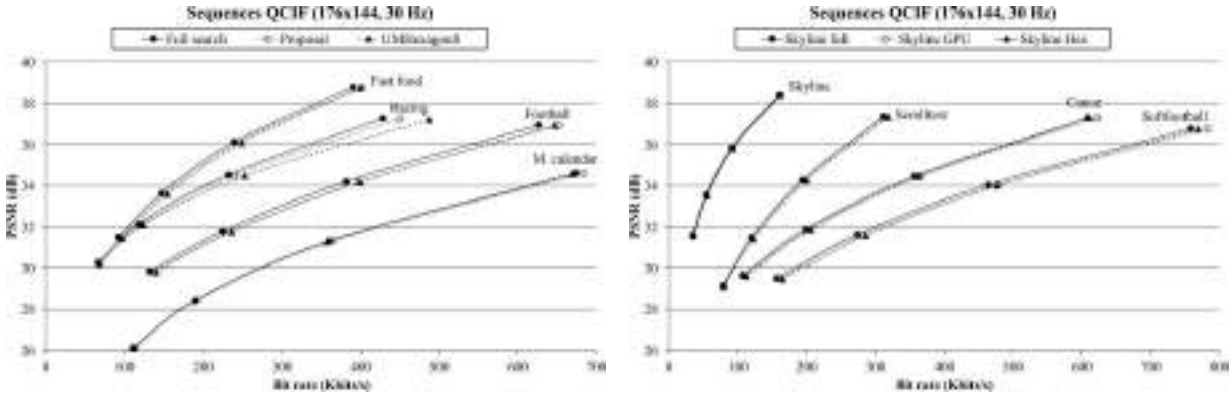
H.264/AVC JM 17.2 Baseline profile – Search range = 32				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
City	2.29	-0.068	-0.08	-0.004
Crew	2.59	-0.059	-1.60	0.038
Dolphins	2.28	-0.065	-5.07	0.152
Harbour	0.90	-0.029	-0.90	0.028
Mobcal	1.85	-0.048	-1.94	0.053
Night	1.55	-0.046	-2.06	0.062
Park run	1.16	-0.035	0.67	-0.020
Shields	1.44	-0.040	1.29	-0.047
Average	1.76	-0.049	-1.21	0.033

Table 4.13: RD results of the proposed encoder for 1080p sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 32				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
Crowd	3.86	-0.118	-1.25	0.035
Ducks	1.20	-0.037	-0.12	-0.002
Into tree	3.66	-0.074	-1.18	0.026
Old town	2.31	-0.048	0.36	-0.017
Park joy	2.33	-0.077	-0.61	0.017
Pedestrian	2.27	-0.051	-6.79	0.168
Riverbed	-0.03	0.000	-0.91	0.024
Tractor	4.49	-0.127	-17.50	0.557
Average	2.51	-0.067	-3.50	0.101

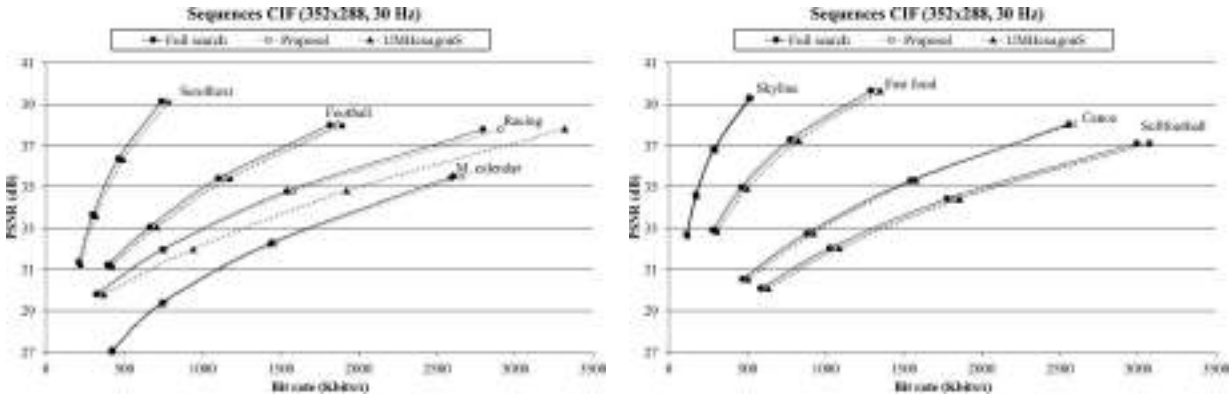
and Old town in 1080p format, are special cases in which the proposed algorithm is not able to outperform the encoding efficiency obtained by the UMHexagonS algorithm.

From Figure 4.13 to Figure 4.17 the RD graphic results for the proposed algorithm and for the reference algorithms are presented. The graphs represent the PSNR versus bit rate curves from a value of 28 to 40 for QP. For clarity the RD curves of each resolution are split into two independent graphs.



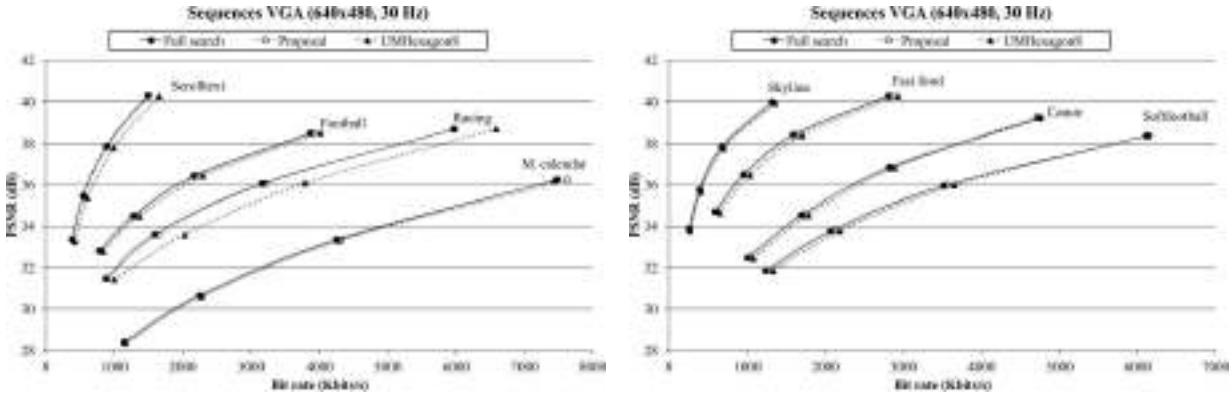
(a) Part I (b) Part II

Figure 4.13: RD graphic results of the proposed encoder for QCIF sequences.



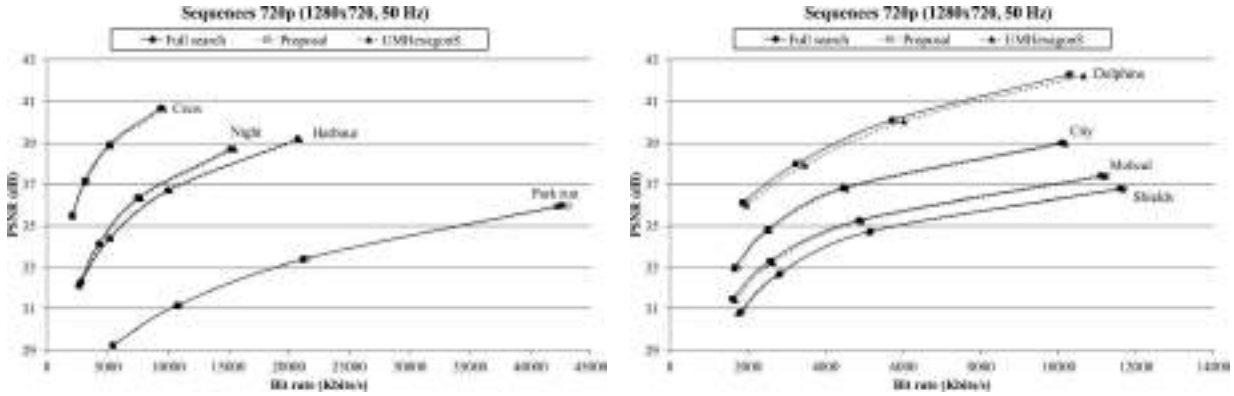
(a) Part I (b) Part II

Figure 4.14: RD graphic results of the proposed encoder for CIF sequences.



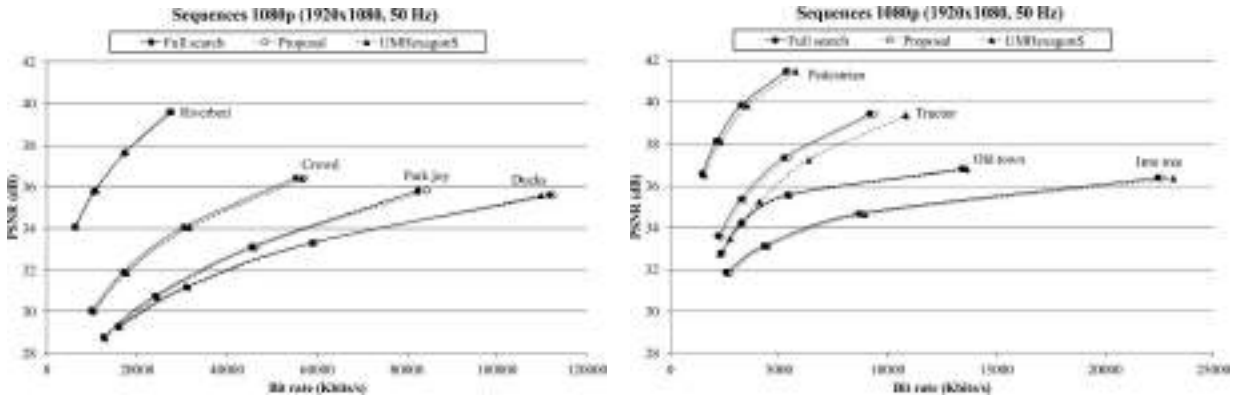
(a) Part I (b) Part II

Figure 4.15: RD graphic results of the proposed encoder for VGA sequences.



(a) Part I (b) Part II

Figure 4.16: RD graphic results of the proposed encoder for 720p sequences.



(a) Part I (b) Part II

Figure 4.17: RD graphic results of the proposed encoder for 1080p sequences.

The PSNR versus bit rate obtained with the proposed H.264/AVC encoder, based on our algorithm, deviates slightly from the results obtained when applying the sequential reference encoder; the curves obtained by the proposed encoder are located in the middle of the curves obtained by the other search algorithms. The curves of the proposed encoder show that the proposed encoder obtains better bit rate results for a given PSNR than the curves obtained by the UMHexagonS algorithm (it improves upon the results of the UMHexagonS algorithm). The curves of the proposed encoder show that the proposed encoder obtains slightly worse bit rate results for a given PSNR than the curves obtained by the FS algorithm. Note that, as expected after the analysis of the RD tables, there are some exceptions, such as the Riverbed video sequence in 1080p format, in which the proposed algorithm outperforms the RD results when compared with the FS algorithm. On the other hand, when encoding the M. calendar video sequence the proposed algorithm is not able to outperform the encoding efficiency of the UMHexagonS algorithm.

Mode decisions

Figures 4.19, 4.20, 4.21 and 4.22 show the inter prediction mode decisions for one frame made by the H.264/AVC encoder using the FS algorithm (Figures 4.19a, 4.20a, 4.21a and 4.22a) and the inter prediction mode decisions made by our proposed algorithm (Figures 4.19b, 4.20b, 4.21b and 4.22b). The mode decisions are obtained when analysing the previously obtained output bit streams in order to carry out the timing and RD evaluation, with a QP value of 28.

Figure 4.19, Figure 4.20 and Figure 4.21 show the inter prediction mode decisions for the 5th frame of the Racing video sequence in QCIF, CIF and VGA format, respectively; Figure 4.22 shows them for the 5th frame of the Crew video sequence in 720p format. The Racing and Crew video sequences are chosen since these video sequences are the ones which produce one of the greatest bit rate increments in comparison with the FS algorithm; the 5th frame is selected, since it is a P frame located in the middle of the configured GOP pattern. On the other hand, Figure 4.18 shows the different kinds of inter prediction modes available for P frames. Note that, in order to obtain the mode decisions carried out by the reference and proposed H.264/AVC encoders, a free-ware software is used which does not support 1080p sequences.

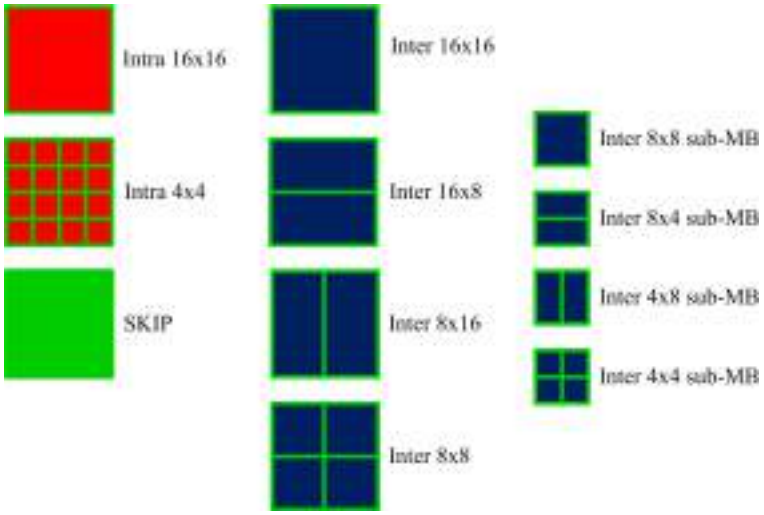
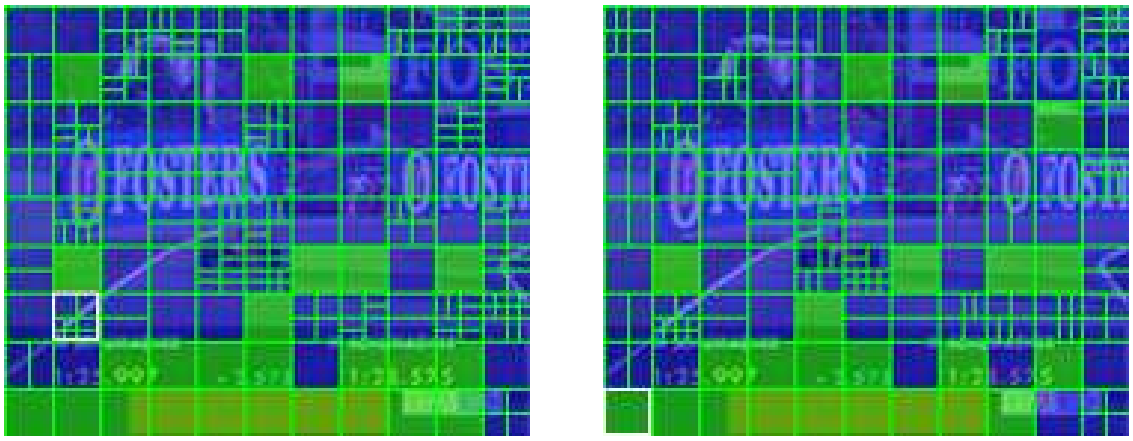


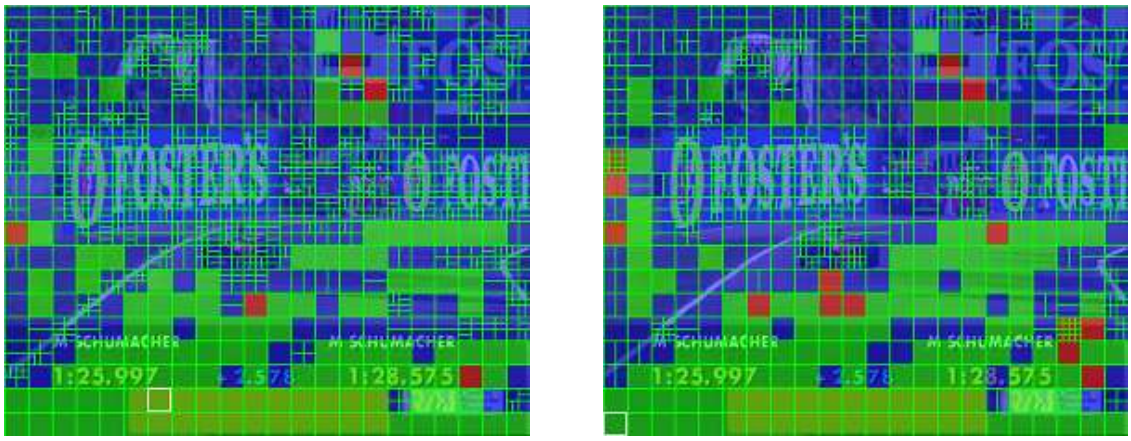
Figure 4.18: Different kinds of predictions in P frames.

As is shown in the figures, different MB labels are generated by our approach, but these partitions are close to the partitions made by the H.264/AVC encoder using the FS algorithm. As a result, although the label partitions are not the same, our closed decisions do not have a significant impact on the final quality performance. The bit rate impact is due to the lack of real MVPs in the cost calculations (see Equation 4.1).



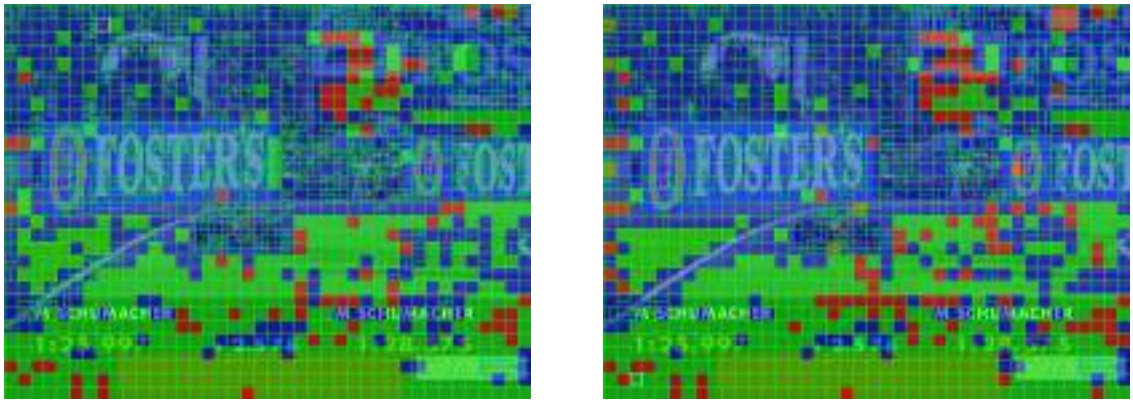
(a) Full search (b) Proposal

Figure 4.19: Mode decisions of Racing video sequence in QCIF format.



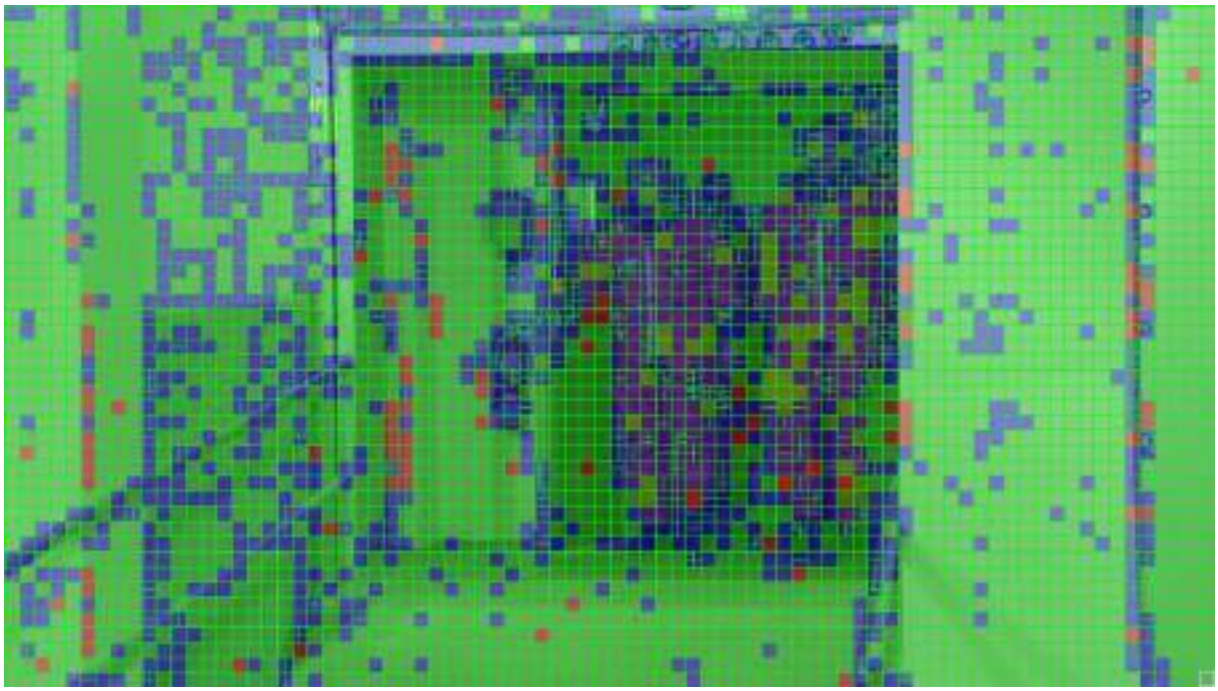
(a) Full search (b) Proposal

Figure 4.20: Mode decisions of Racing video sequence in CIF format.

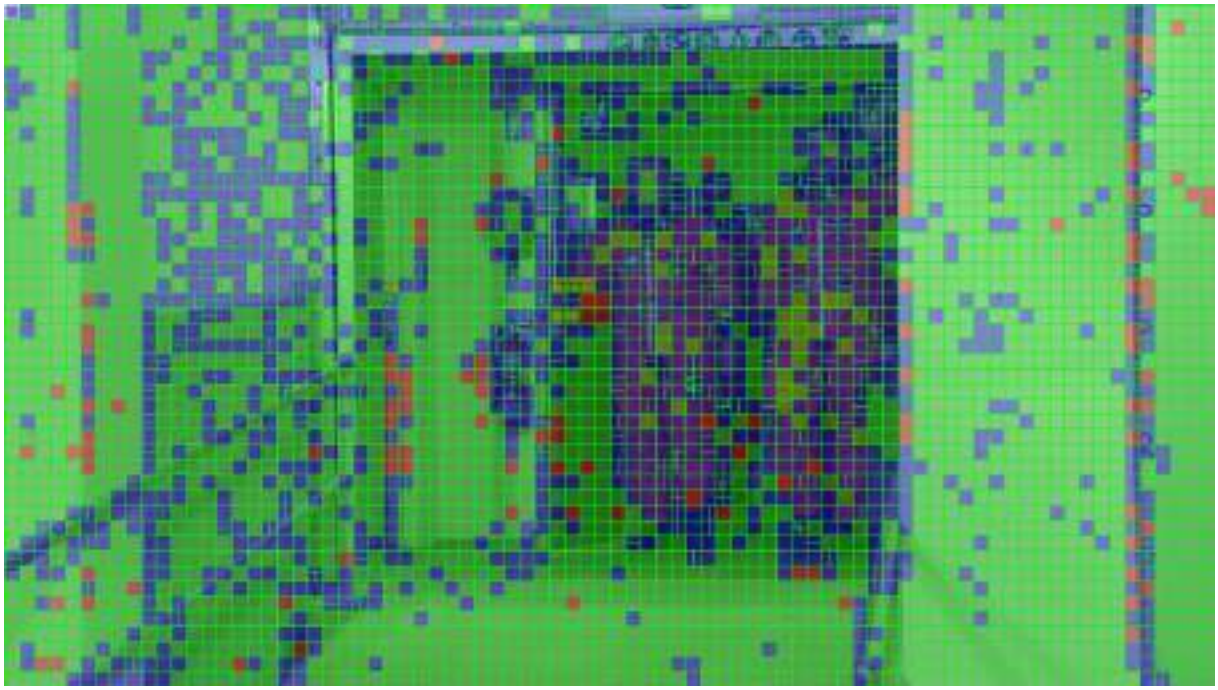


(a) Full search (b) Proposal

Figure 4.21: Mode decisions of Racing video sequence in VGA format.



(a) Full search



(b) Proposal

Figure 4.22: Mode decisions of Crew video sequence in 720p format.

Power and energy results

From Table 4.14 to Table 4.17 the power and energy consumption results of the proposed algorithm are presented. These tables show the average power consumption, the execution time and the total energy consumed when coding one GOP (12 frames) for the complete test computer when coding the tested sequences in CIF, VGA, 720p and 1080p format. An analysis for QCIF video sequences is not provided since the execution time of the GPU kernels is considerably shorter than the cycles of the electric current, and the power consumption peaks cannot be properly reconstructed, which is one of the major aims of this analysis, along with showing the overall energy consumption of the test computer. The power consumption peaks cannot be fully reconstructed for CIF and VGA video sequences, but at least they can be identified in the power consumption graphic results (Figure 4.23a and Figure 4.23b).

The first main column shows the results for the H.264/AVC encoder using the FS algorithm, the second main column shows them for the H.264/AVC encoder using the UMHexagonS algorithm, and the third one shows them for the H.264/AVC encoder using the proposed algorithm. Additionally, there are two extra results for the proposed encoder, which show the ratio between the energy consumed by the H.264/AVC encoder using the FS algorithm and the proposed algorithm (*Ratio FS* column), and the ratio between the energy consumed by the H.264/AVC encoder using the UMHexagonS algorithm and the proposed algorithm (*Ratio UMHS* column).

Table 4.14: Energy consumption for coding a GOP. CIF sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 16											
Sequence	Full search			UMHexagonS			Proposal				
	Power (W)	Time (s)	Energy (J)	Power (W)	Time (s)	Energy (J)	Power (W)	Time (s)	Energy (J)	Ratio FS	Ratio UMHS
Canoe	181.97	7.00	1,273.79	179.63	1.62	291.00	229.56	0.69	158.40	8.04	1.84
Fast food	181.29	5.63	1,020.66	180.44	1.22	220.14	221.96	0.67	148.71	6.86	1.48
Football	181.77	6.58	1,196.05	179.45	1.33	238.67	227.73	0.69	157.13	7.61	1.52
M. calendar	181.75	6.29	1,143.21	179.68	1.24	222.80	232.43	0.69	160.38	7.13	1.39
Racing	181.54	5.22	947.64	179.35	1.25	224.19	236.64	0.66	156.18	6.07	1.44
Scrolltext	180.83	3.36	607.59	179.77	0.98	176.17	229.67	0.66	151.58	4.01	1.16
Skyline	180.90	4.25	768.83	179.26	1.06	190.02	230.67	0.66	152.24	5.05	1.25
Softfootball	181.56	7.17	1,301.79	180.03	1.35	243.04	230.95	0.70	161.67	8.05	1.50
Average	181.45	5.69	1,032.45	179.70	1.26	225.75	229.95	0.68	155.79	6.60	1.45

The average power consumption for the H.264/AVC encoder using the reference algorithms (FS and UMHexagonS) is very similar and ranges from 179 W to 185 W, regardless of the video sequence resolution used. However, the execution time varies considerably, and as a consequence the overall energy consumption varies greatly. For example, when

Table 4.15: Energy consumption for coding a GOP. VGA sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 32											
Sequence	Full search			UMHexagonS			Proposal				
	Power (W)	Time (s)	Energy (J)	Power (W)	Time (s)	Energy (J)	Power (W)	Time (s)	Energy (J)	Ratio FS	Ratio UMHS
Canoe	183.42	55.41	10,163.30	182.68	5.26	960.90	245.12	2.01	492.69	20.63	1.95
Fast food	184.69	50.50	9,326.85	182.23	4.17	759.90	246.18	2.03	499.75	18.66	1.52
Football	183.37	53.67	9,841.47	182.16	4.40	801.50	246.72	2.05	505.78	19.46	1.58
M. calendar	184.19	56.90	10,480.41	182.48	4.09	746.34	244.84	2.10	514.16	20.38	1.45
Racing	184.04	41.50	7,637.66	182.32	4.08	743.87	248.47	2.01	499.42	15.29	1.49
Scrolltext	183.50	25.40	4,660.90	182.41	3.03	552.70	249.07	1.92	478.21	9.75	1.16
Skyline	183.73	32.68	6,004.30	182.02	3.18	578.82	248.95	1.95	485.45	12.37	1.19
Softfootball	184.45	57.24	10,557.92	182.11	4.44	808.57	249.45	2.08	518.86	20.35	1.56
Average	183.92	46.66	8,584.10	182.30	4.08	744.08	247.35	2.02	499.29	17.11	1.49

Table 4.16: Energy consumption for coding a GOP. 720p sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 32											
Sequence	Full search			UMHexagonS			Proposal				
	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Ratio FS	Ratio UMHS
City	184.00	124.85	22.97	178.91	10.92	1.95	247.32	6.04	1.49	14.42	1.31
Crew	184.93	154.92	28.65	179.36	11.49	2.06	249.43	5.86	1.46	19.62	1.41
Dolphins	185.13	125.22	23.18	179.71	12.39	2.23	250.06	5.99	1.50	15.45	1.49
Harbour	185.47	138.12	25.62	179.94	10.85	1.95	251.97	6.06	1.53	16.75	1.27
Mobcal	184.52	159.79	29.48	180.30	10.80	1.95	252.95	5.93	1.50	19.65	1.30
Night	184.40	114.17	21.05	180.61	10.67	1.93	253.45	5.93	1.50	14.03	1.29
Park run	185.37	224.09	41.54	180.94	13.85	2.51	255.27	6.25	1.60	25.96	1.57
Shields	185.53	172.36	31.98	181.56	12.88	2.34	256.46	5.89	1.51	21.18	1.55
Average	184.92	151.69	28.06	180.17	11.73	2.12	252.11	5.99	1.51	18.38	1.40

coding one GOP in 1080p format, the total energy consumption ranges from 47.66 kJ to 95.5 kJ, because the execution time ranges from 257.45 s to 516.38 s.

The average power consumption for the H.264/AVC encoder using the proposed algorithm is also very similar and ranges from 246 W to 256 W when coding video sequences in 720p and 1080p format, and is a bit less when coding CIF and VGA video sequences, since the power consumption peaks are not fully reconstructed. However, in this case the execution time is also very similar, and as a consequence the overall energy consumption does not vary significantly. For example, when coding one GOP in 1080p format, the total

Table 4.17: Energy consumption for coding a GOP. 1080p sequences.

H.264/AVC JM 17.2 Baseline profile – Search range = 32											
Sequence	Full search			UMHexagonS			Proposal				
	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Ratio FS	Ratio UMHS
Crowd	185.52	389.10	72.19	182.62	25.79	4.71	246.55	14.44	3.56	20.28	1.32
Ducks	185.61	429.46	79.71	182.96	27.63	5.06	250.56	14.41	3.61	22.08	1.40
Into tree	184.23	393.30	72.46	183.02	23.39	4.28	251.67	13.60	3.42	21.19	1.25
Old town	184.61	364.80	67.35	182.78	22.42	4.10	252.87	13.46	3.40	19.81	1.21
Park joy	184.57	396.95	73.27	182.57	28.36	5.18	253.13	13.95	3.53	20.76	1.47
Pedestrian	185.11	257.45	47.66	183.17	25.67	4.70	250.22	13.41	3.36	14.18	1.40
Riverbed	184.94	516.38	95.50	182.85	35.44	6.48	250.82	14.02	3.52	27.13	1.84
Tractor	184.16	277.72	51.14	183.20	28.62	5.24	251.46	13.90	3.50	14.61	1.50
Average	184.84	378.15	69.91	182.90	27.17	4.97	250.91	13.90	3.49	20.01	1.42

energy consumption ranges from 3.36 kJ to 3.61 kJ, because the execution time ranges from 13.41 s to 14.4 s.

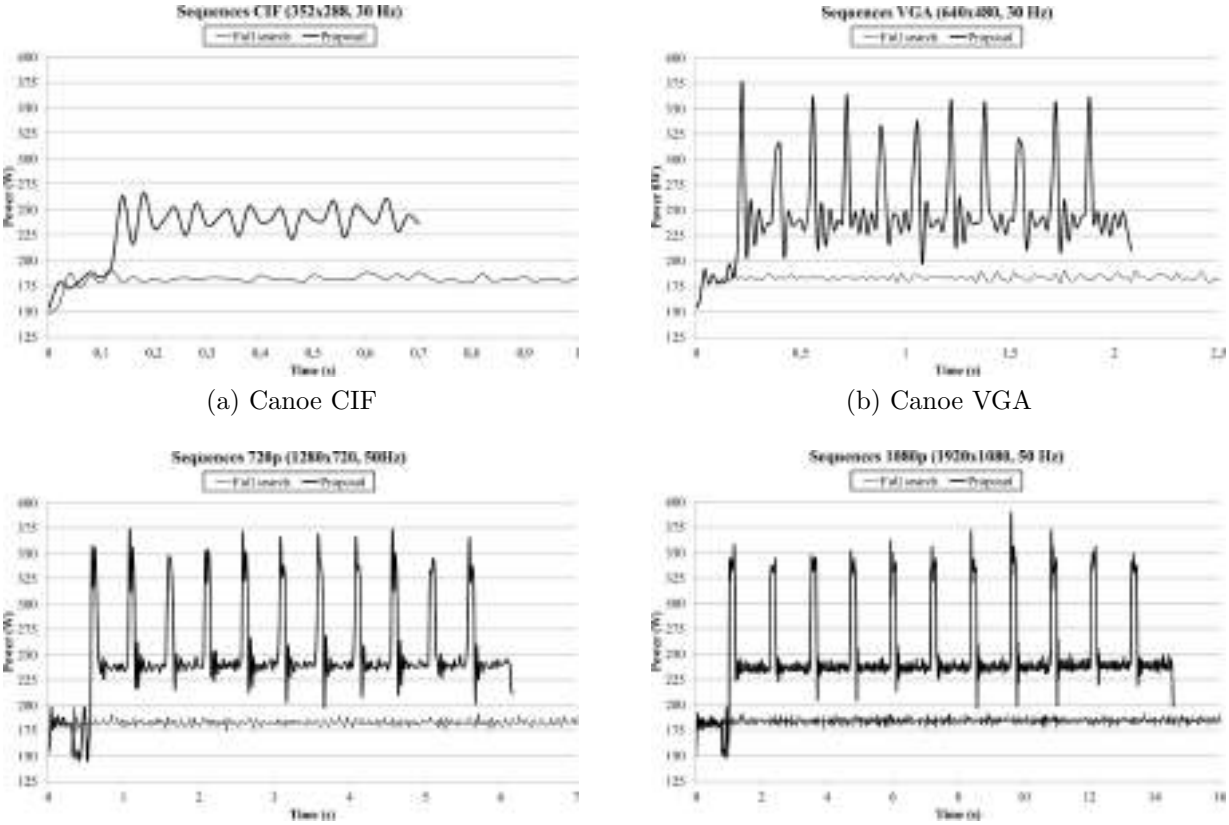
On average, when coding VGA, 720p and 1080p video sequences, the energy consumption for the H.264/AVC encoder using the proposed algorithm ranges from 17 to 20 times better than for the H.264/AVC encoder using the FS algorithm; and is 6 times better when coding CIF video sequences (remember that the search range for CIF video sequences is set to 16, while the search range for the other resolutions is set to 32). On the other hand, for all video formats, the energy consumption for the H.264/AVC encoder using the proposed algorithm ranges from 1.4 to 1.5 times better than for the H.264/AVC encoder using the UMHexagonS algorithm.

Note that there is not much difference in the average power consumption between the H.264/AVC encoder using the reference algorithms (179 W to 185 W) and the proposed GPU-based encoder (229 W to 252 W), and the reason is that the GPU is processing less than 20% of the total encoding time.

Figure 4.23 shows an extract from the power consumption over time for the complete test computer when coding one video sequence of each resolution for the H.264/AVC encoder using the FS algorithm and for the H.264/AVC encoder using the proposed algorithm. Note that the power consumption over time when using the UMHexagonS algorithm is not shown in these graphs because it is similar to the one shown when using the FS algorithm, although the execution time is shorter (see Tables 4.14 to 4.17).

When the encoder process begins all encoders consume the same power (around 180–185 W), but when the GPU starts working the power consumption of the proposed GPU–based encoder increases. It has a power consumption of around 325-375 W, except for CIF format in which case the power consumption peaks are not fully reconstructed (Figure 4.23). The configured GOP pattern is composed of 1 I frame followed by 11 P frames where the

proposed algorithm is executed, so 11 power consumption peaks can be found on each graph in Figure 4.23. Finally, we should mention that the power consumption for the CPU code in the GPU–based encoder is around 235 W, which is higher than for the reference execution (180–185 W) because the GPU is always active, waiting for new kernels.



(a) Canoe CIF (b) Canoe VGA

(c) City 720p (d) Crowd 1080p

Figure 4.23: Power consumption graphic results.

4.4. Comparison with other known results

This section compares the results reported by our proposed algorithm developed for P frames with the ones reported in the most recent and relevant related articles, which are described in Chapter 3. The comparison is divided into two main parts: the first part compares against works that do not use a GPU in order to reduce the H.264/AVC encoding time, and the second one with proposals that do.

4.4.1. Soft-Computing H.264/AVC Inter Prediction algorithms

As mentioned in Section 3.1, the design of fast MB mode decision algorithms for the inter frame prediction of H.264/AVC video encoders is a major area of research. In this section, we undertake a comparative analysis of our proposal with some of the most prominent ones presented for an H.264/AVC encoder scenario. Throughout our experiments, the encoding conditions of each article have been reproduced as far as possible, and a reasonably objective and fair comparison is possible by following Bjøntegaard and Sullivan's common test rule [Sullivan and Bjontegaard 01]. Unfortunately, there are only two articles that make use of this test rule. The comparison metrics have been produced and tabulated based on the TRs, ΔPSNR and ΔBit rate differences. The video sequences used are those used in the related publications.

Table 4.18 and Table 4.19 summarize our main findings. As seen from the tables, our proposal significantly outperforms those previously reported in terms of TR (%) for all video sequences. However, for some video sequences, greater PSNR decrements and bit rate increments are reported, but this extra penalty is negligible if the time savings are taken into account. Note that for some video sequences [Bystrom et al. 08] outperforms or maintains the coding efficiency of the reference encoder because some sequences are the same as those used to train the Bayesian network on which the proposal is based.

Table 4.18: Comparison against [Bystrom et al. 08]

Sequence	Format	Method	TR (%)	ΔPSNR (dB)	Δbit rate (%)
Akiyo	QCIF	Proposal	91.97	-0.021	0.66
		[Bystrom et al. 08]	84.42	0.010	-0.18
Container	QCIF	Proposal	93.02	-0.010	0.32
		[Bystrom et al. 08]	80.51	0.060	-1.42
Foreman	QCIF	Proposal	94.48	-0.100	3.61
		[Bystrom et al. 08]	37.57	0.000	0.23
M. calendar	QCIF	Proposal	96.08	-0.090	2.23
		[Bystrom et al. 08]	9.10	-0.150	2.75
News	QCIF	Proposal	93.26	-0.073	2.10
		[Bystrom et al. 08]	78.81	-0.020	0.28
Silent	QCIF	Proposal	94.13	-0.049	1.64
		[Bystrom et al. 08]	71.00	0.020	-0.49

Table 4.19: Comparison against [Liu et al. 09]

Sequence	Format	Method	TR (%)	ΔPSNR (dB)	Δbit rate (%)
Coastguard	CIF	Proposal	96.27	-0.048	1.60
		[Liu et al. 09]	33.10	-0.022	0.64
Foreman	CIF	Proposal	94.50	-0.061	2.40
		[Liu et al. 09]	37.70	-0.049	1.48
M. calendar	CIF	Proposal	95.80	-0.044	1.10
		[Liu et al. 09]	28.20	-0.073	1.77
News	CIF	Proposal	92.35	-0.018	0.52
		[Liu et al. 09]	49.20	-0.114	2.43
Paris	CIF	Proposal	93.84	-0.024	0.68
		[Liu et al. 09]	41.00	-0.072	1.81
Silent	CIF	Proposal	94.13	-0.014	0.51
		[Liu et al. 09]	49.60	-0.036	0.91
Stefan	QCIF	Proposal	95.36	-0.089	2.30
		[Liu et al. 09]	33.00	-0.082	1.62

4.4.2. GPU–based H.264/AVC Inter Prediction algorithms

As with the comparison made with related works that do not use a GPU to reduce the H.264/AVC encoding time, a comparative study with some of them is not possible because of several reasons. Some of them are not integrated into any H.264/AVC encoder and as a consequence do not evaluate the RD performance, some of them do not support all MB partitions and sub–partitions available in H.264/AVC and some of them use different cost metrics. Moreover, all the works prior to 2006 do not use CUDA since CUDA emerged in 2007. As a consequence of these reasons, a comparison with those articles would not be fair.

Fortunately, a comparison against [Cheung et al. 10] is possible. In order to obtain an adequate evaluation, the comparison is carried out encoding the same video sequences (720p), using the same encoding conditions, and is achieved when comparing the results against the smpUMHexagonS algorithm implemented in the H.264/AVC JM reference encoder with their results against smpUMHexagonS too. Remember that, as was explained in Chapter 3, the encoding efficiency and the time savings reported in [Cheung et al. 10] depend on the number of tiles in which they divide each frame, so the results for their

proposal are divided depending on that. Their algorithm is faster the more tiles are used, but the encoding efficiency is lower.

Table 4.20 shows the execution time for the experiments carried out and also shows the average execution time for each configuration. Before the analysis note that the peak performance for our GPU is 1350 GFlops and the peak performance for the GPU used in [Cheung et al. 10] is 345.6 GFlops, which means that our GPU is 3.9 times more powerful. For this reason, and for a fair comparison, an extra column labelled Index has been included in Table 4.20, which shows the ratio between the average execution time obtained by their implementation for a certain encoder configuration and the average execution time for our implementation using the same encoder configuration. Higher values than 3.9 for this index mean that the proposed algorithm is faster than their algorithm.

In conclusion, our algorithm is as fast as their best configuration (index of 3.85), and it outperforms the execution time for the other configurations (higher index than 3.9). The execution time using 3 and 12 tiles is not specified in [Cheung et al. 10]; however, higher execution times are expected since they use less GPU threads.

Table 4.20: Execution time comparison with [Cheung et al. 10] results.

	Sequences				Average GPU time (ms)	Index
	Crew	City	Harbour	Night		
Number of tiles	GPU time (ms)	GPU time (ms)	GPU time (ms)	GPU time (ms)		
3,600	835.05	927.32	1,248.95	1,688.50	1,174.95	3.85
900	959.16	1,005.55	1,341.45	1,975.95	1,320.53	4.33
225	2,169.25	2,108.71	2,763.79	4,175.44	2,804.30	9.19
90	4,373.63	4,165.28	5,318.38	6,920.73	5,194.51	17.02
12	Unknown					
3	Unknown					
Proposal	305.99	306.16	304.96	304.71	305.23	

Table 4.21 shows the RD results for their algorithm as well as our RD results using the same encoding conditions. [Cheung et al. 10] obtains more degradation as many tiles are used due to the dependencies between neighbouring MBs. However, the proposed algorithm outperforms the RD performance obtained by their fastest configurations (90 or more tiles); our algorithm has lower bit rate increments and lower PSNR losses than their algorithm for all video sequences.

Table 4.21: RD comparison with [Cheung et al. 10] results.

	Sequences							
	Crew		City		Harbour		Night	
Number of tiles	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
3,600	3.14	-0.082	12.93	-0.407	5.58	-0.221	4.64	-0.170
900	3.08	-0.079	11.12	-0.352	2.39	-0.094	3.55	-0.130
225	3.12	-0.080	11.17	-0.350	2.25	-0.089	3.42	-0.125
90	3.22	-0.083	10.82	-0.339	2.21	-0.087	3.40	-0.124
12	0.63	-0.016	1.41	-0.044	0.57	-0.022	1.19	-0.043
3	0.09	-0.003	0.26	-0.008	0.07	-0.003	0.16	-0.006
Proposal	3.08	-0.071	6.68	-0.309	0.88	-0.028	1.55	-0.047

4.5. Conclusions

In order to reduce the high complexity of the H.264/AVC JM 17.2 reference encoder, this chapter proposed a GPU–based inter prediction algorithm developed for P frames. The proposed algorithm has been tested using different video resolutions and the most important conclusions are summarized as follows:

- The proposed algorithm for all video resolutions achieves a time reduction of over 97% when compared with the FS algorithm, and over 85% when compared with the UMHexagonS algorithm. The TRs increase when the GPU computational load also increases.
- The proposed algorithm for all video resolutions achieves negligible RD drop penalties if the time savings are taken into account, when compared with the FS algorithm. On the other hand, it surpasses the coding efficiency of the UMHexagonS algorithm.
- The GPU–based algorithm raises the average power consumption of the complete system. However, as the execution time is shorter, the H.264/AVC JM 17.2 using the proposed algorithm consumes less energy than when using the reference algorithms (FS and UMHexagonS).
- Finally, the proposal is tested against related proposals. In a first section, it is tested against proposals that do not use a GPU to accelerate the inter prediction module, outperforming their timing results and maintaining their coding efficiency. On the other hand, in a second section, it is tested against one proposal that uses a GPU to accelerate the inter prediction module, equalling its best timing results and outperforming its coding efficiency.

Chapter 5

B frame Inter Prediction

IN this chapter, the proposal developed for B frames is presented. First, some observations about the proposed and the reference inter prediction algorithms are made. Then, the proposal is described in detail. Finally, it is evaluated, including a comparison with the most recent and prominent related proposals.

5.1. Introduction

As was mentioned in Chapter 4, ME sequentially obtains the motion information (encoding costs) for all available MB partitions and sub-partitions, for all MBs in a frame. For each MB partition and sub-partition, a search area is defined and the search algorithm chosen is executed. The search algorithm looks for a region that minimizes the differences between the current partition or sub-partition and the chosen region. However, in a B frame there are two different ways of doing this. The first one uses one frame to perform the prediction and is called *forward and backward prediction*; the second one uses two frames to perform the prediction and is called *bi–directional prediction*. Moreover, the bi–directional prediction is carried out in two steps: the first one searches in a future frame and the second one searches in a previous frame.

The main challenge of the approach presented in this chapter is to efficiently support the tree-structured MC algorithm executed in the H.264/AVC JM 17.2 reference software encoder [JVT 11] for B frames. The idea is to concurrently obtain the motion information for all MBs at the beginning of coding each B frame. Forward, backward and bi–directional predictions are executed sequentially following a highly–parallel procedure on the GPU.

The proposed algorithm developed for B frames has the same starting point as the one developed for P frames, i.e. the FS algorithm. As a consequence, the same memory allocations and search area distribution are used to implement the proposed algorithm.

5.2. Proposed algorithm

This section describes the algorithm developed for B frames. The algorithm is divided into two main parts: forward and backward prediction, and bi–directional prediction.

5.2.1. Forward and backward prediction

The algorithm developed for forward and backward prediction is based on the one presented for P frames in Chapter 4. In fact, the only difference is where the reference frames are located. The algorithm developed for forward and backward prediction uses previous or future frames to estimate the current frame, while the algorithm developed for P frames only uses previous frames. Note that this manner of prediction means that the frames are encoded out of order (the encoding order is different from the temporal order), otherwise the forward prediction could not be performed because the prediction is carried out using a previously encoded reconstructed frame (see the block diagram of a generic H.264/AVC encoder (Figure 2.3) shown in Section 2.1).

In order to encode a B frame, all its reference frames (previous and future) are encoded in advance, which means that the motion information for all of them is available and can be used to calculate the MVPs of the current frame.

The MVP for backward prediction is calculated using the MV of the 16x16 partition of the MB located in the same position, but in the temporally previous frame. The MVP for forward prediction is calculated using the MV of the 16x16 partition of the MB located in the same position, but in the temporally future frame.

5.2.2. Bi–directional prediction

Bi–directional prediction is carried out in two steps. The first one estimates each MB partition and sub–partition using a block located in the temporally previous frame (the dimensions of the block are the same as those of the MB partition or sub–partition being estimated) and a search area located in the temporally future frame. In what follows, we will refer to the block as the opposite block. Bi–directional prediction is performed by searching in the future frame. In the second one, the opposite block and the search area interchange their locations, the opposite block is located in the future frame and the search area is located in the previous frame. Bi–directional prediction is performed by searching in the previous frame. The output of both steps is two MVs, one referring to the previous frame and one referring to the future frame.

As with the algorithm described for P frames in Chapter 4, the bi–directional prediction algorithm is divided into three main parts: MVP calculation, IME and FME. The description of the algorithm focuses on the first step, but an analogous description for the second step could be provided.

5.2.2.1. MVP calculation

The first task in order to run the proposed algorithm is to locate the opposite block and the search area, i.e. to obtain an MVP for each of them. The H.264/AVC JM 17.2 reference encoder locates the opposite block using the MV obtained with the backward or the forward prediction, depending on where the opposite block is located, and its dimensions. The search area is located using the motion information of previously encoded neighbouring MBs using the bi–directional prediction algorithm (the same procedure used to calculate the real MVPs in Chapter 4, and explained in Section 2.1.2). However, the motion information of neighbouring MBs is not available in parallel execution and a different MVP calculation is needed.

Figure 5.1 shows how to locate the search area and the opposite block in the first step of the algorithm, where MVP_1 is used to locate the opposite block and MVP_2 is used to locate the search area. As MVP for the opposite block, we propose to use the MV of the partition being encoded. If the opposite block is located in the previous frame (MVP_1), the MVP was obtained with the backward prediction; if it is located in the future frame (MVP_2), the MVP was obtained with the forward prediction. As MVP for the search area, we propose to use the MV of the 16x16 partition of the MB located in the same position. If the search area is located in the previous frame (MVP_1), the MVP was obtained with the backward prediction; if it is located in the future frame (MVP_2), the MVP was obtained with the forward prediction. Note that the MVP of the search area is common for all MB partitions and sub–partitions, which means that all partitions and sub–partitions within an MB use the same search area.

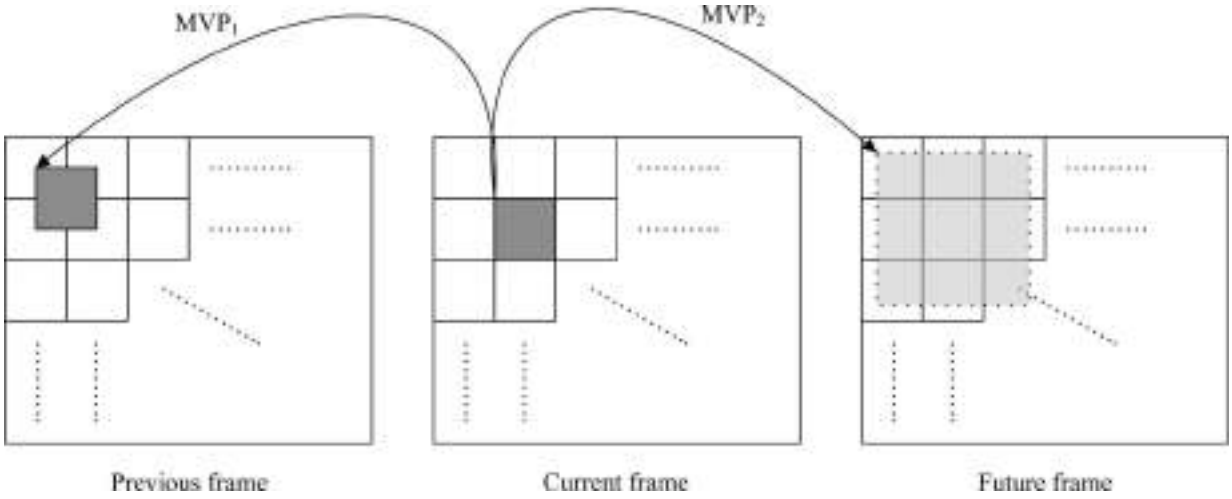


Figure 5.1: Search area and opposite block locations.

However, there is a problem in the proposed MVP calculation, as forward and backward predictions must be completely finalized before starting the bi—directional prediction, and with the H.264/AVC JM encoding order this does not occur. The encoding order implemented in the H.264/AVC JM reference encoder is: backward prediction, bi—directional

prediction (first step), forward prediction and bi—directional prediction (second step). The solution is to modify the encoding order implemented: firstly backward and forward predictions are performed, and then the bi–directional predictions are performed.

Finally, the encoding costs are calculated using Equation 5.1. Note that the main difference in comparison with the encoding costs calculated for P frames (or calculated for forward and backward prediction) is that the output of the bi–directional prediction algorithm is two MVs, which are used to obtain the encoding costs.

$$Cost = SAD + \lambda \cdot R_1 + \lambda \cdot R_2, \tag{5.1}$$

where SAD is the metric used to calculate the differences (see Section 2.1.2 for more details), λ is an encoder parameter which depends on the QP used, R_1 is the number of bits required to encode MV_1 minus MVP_1, and R_2 is the number of bits required to encode MV_2 minus MVP_2.

5.2.2.2. Integer Motion Estimation

In a similar way to the algorithm developed for P frames in Chapter 4, IME is divided into a number of steps which are executed sequentially following a highly–parallel procedure by using the GPU. However, in the bi—directional prediction algorithm it is not possible to calculate the encoding costs of higher MB partitions using the ones obtained for the 4x4 sub–partition. Different MVPs are used to locate the opposite block(s) of each MB partition and sub–partition. For this algorithm the steps are: obtain the encoding costs, and then perform a reduction to obtain the best MV.

As with the algorithm presented in Chapter 4, IME is carried out using two GPU kernels. Each thread block of the first GPU kernel obtains and reduces the motion information of 256 adjacent search area positions of a given MB; if more than 256 search area positions are defined in the search area, a second kernel performs a final reduction. The reduction procedure used to obtain the best MV for all MB partitions and sub–partitions with integer–pixel accuracy is the same as the one described in Chapter 4, so this section focuses on how to obtain the motion information of all MB partitions and sub–partitions.

The first GPU kernel is carried out in a number of iterations, which depends on the number of MB partitions and sub–partitions configured. For example, if all MB partitions and sub–partitions are enabled in the configuration file, seven iterations are needed. The last part of each iteration is the reduction procedure.

For each algorithm iteration, the current block, the opposite block and the search area are required. The current block and the search area are the same in all algorithm iterations, so at the beginning of this first GPU kernel all GPU threads collaborate to allocate them to shared memory. Note that the search area is allocated using the same structure as the one described in Figure 4.3. On the other hand, the opposite block is filled in on each algorithm iteration. A memory matrix is defined for that in shared memory, which is either filled in with the 16x16 opposite block, or with the two 16x8 opposite blocks, or with the two 8x16 opposite blocks, and so on. All threads also collaborate to allocate the opposite block(s).

On each algorithm iteration, each GPU thread obtains the SAD costs for the sixteen 4x4 blocks into which both the current block and the opposite block can be divided for a specific position. Additionally, if the opposite block was filled in with the 16x16 partition, the sixteen SAD costs are added in order to obtain the motion information of the 16x16 partition; if the opposite block was filled in with the two 16x8 partitions, eight SAD costs are added in order to obtain the motion information of the upper 16x8 partition, and eight SAD costs are added in order to obtain the motion information of the lower 16x8 partition; and so on. When the SAD of the current partition or sub–partition is properly obtained, the final encoding is calculated using Equation 5.1.

Finally, a pseudo–code of the proposed algorithm is presented. Note that this pseudo–code only covers the first step of the proposed algorithm. The pseudo–code for the second step is the same, but the search are is located using MVP_1 (the algorithm searches in the previous frame) and the opposite block(s) is(are) located using MVP_2.

Algorithm Proposed IME algorithm for B frames

```
Bi–prediction (Step 1)
    For each thread block configured – Kernel 1
        Transfer current block from GPU texture memory to shared memory
        Transfer search area from GPU texture memory to shared memory (MVP2)
        For each MB partition and sub–partition
            Transfer opposite block from GPU texture memory to shared memory (MVP1)
            For each thread within the thread block
                Calculate motion information (SAD cost)
                Merge SAD costs + calculate encoding costs
                Reduction (1 MV per MB–partition within the thread block)
    Final reduction – Kernel 2 (1 MV per MB–partition)
```

5.2.2.3. Fractional Motion Estimation

Once the execution of IME has finalized, FME is carried out. The algorithm is based on the one explained in Chapter 4 for FME. However, in this case, when calculating the encoding costs, the opposite block (which is located in shared memory) is taken into account. The current block continues in shared memory and the sub–pixel search area continues being checked from texture memory. SADT using Hadamard transforms is the metric used to obtain the encoding costs.

Finally, we should mention that it is not necessary to sub–sample the reference frames because they have been previously sub–sampled when executing the forward and backward prediction algorithms.

5.3. Performance evaluation

In this section, the results of applying the proposal described in this chapter are presented. First of all, the encoding conditions and the metrics used to evaluate the proposal are described. Then, the evaluation is carried out.

5.3.1. Encoding conditions

In order to evaluate the proposed inter prediction algorithm developed for B frames, it has been integrated into the H.264/AVC JM 17.2 reference encoder [JVT 11]. The H.264/AVC encoding parameters used for the evaluation are those included in the Main profile of the mentioned reference encoder. However, some parameters are changed in the configuration file and these are described below. In general, the encoding conditions are similar to the ones used in Chapter 4, but using a different profile.

- The number of reference frames is set to 4 for P frames, and to 2 in both directions (forward and backward) for B frames. We show that the algorithms presented in this thesis are also appropriate for using multiple reference frames
- RD–Optimization is disabled in order to keep the complexity as low as possible. However, in Appendix A the algorithms proposed in this thesis are evaluated enabling this option.
- The number of B frames inserted between each I or P frame is set to 7 and the intra period is set to 32. Therefore, in each GOP there are 28 B frames, 3 P frames and 1 I frame. Figure 5.2 shows the first 9 frames of a GOP. In what follows we will refer to this structure as *mini GOP*. Therefore, each GOP is divided into 4 *mini GOPs*.
- The GOP pattern used is full hierarchy, and is depicted in Figure 5.2. Note that in the figure only the reference for the closest reference frames is depicted, but as mentioned above, multiple reference frames are used.
- The QP for P and I frames is varied among 28, 32, 36, and 40, according to [Bjontegaard 01], [Sullivan and Bjontegaard 01] and [JVT Test Model Ad Hoc Group 03]; the QP for B frames depends on the hierarchical level on which the B frame is located (see Figure 5.2), and is incremented by 1 per hierarchical level in reference with the one configured for P and I frames.
- The tests are carried out with popular sequences in QCIF (176x144 pixels), CIF (352x288 pixels), VGA format (640x480 pixels), 720p format (HD, 1280x720 pixels) and 1080p format (full-HD, 1920x1080 pixels). These sequences are the same as those used in the performance evaluation section in Chapter 4, which are sequences with different characteristics (content and movement features).

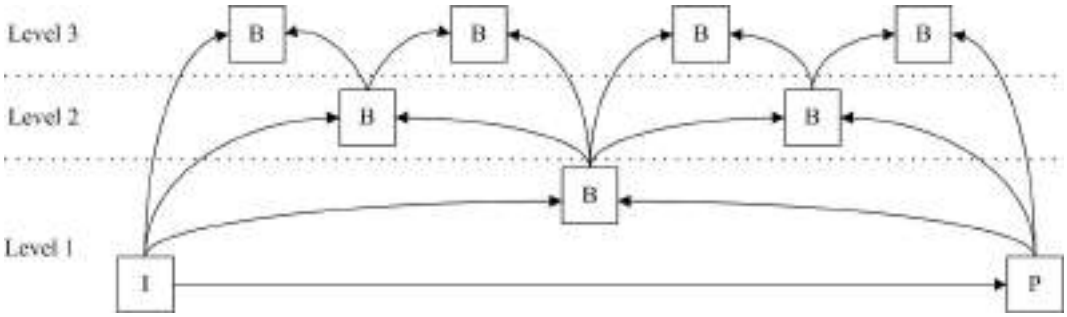


Figure 5.2: Configured GOP pattern.

- The frame rate parameter is set to 30 for QCIF, CIF and VGA format (30 Hz), and to 50 for 720p and 1080p format (50 Hz).
- The search range is set to 32, which means 4096 positions inside the search area of each MB partition.
- The proposed inter prediction algorithm is tested against two search algorithms implemented in the H.264/AVC JM 17.2 reference encoder: FS [Richardson 10] and UMHexagonS [Rahman and Badawy 05].

In order to make a proper comparison, an unmodified H.264/AVC JM 17.2 reference encoder is run on the same machine as the H.264/AVC JM using the proposed algorithms, with the same encoding configuration and with no calls to the GPU. The development environment, including the GPU, is the same as the one used in the evaluation of Chapter 4.

5.3.2. Metrics

Several metrics have been used to evaluate the proposal. These metrics are the TR and speed–up, RD function, ΔPSNR and Δbit rate, mode decisions and power and energy consumption. All these metrics have been previously defined in Section 4.3.2.

5.3.3. Results

This section presents the results obtained when coding different video sequences using the proposed inter prediction algorithm developed for B frames.

Timing results

As in the study of Chapter 4, the execution time of the reference algorithms (FS and UMHexagonS) is sequence dependent, while the execution time of the proposed algorithm is almost constant. As a consequence, the TRs and speed–ups reported depend on that.

From Table 5.1 to Table 5.3 the timing results of the proposed algorithm for coding the same QCIF, CIF and VGA video sequences are presented. The results are mainly divided depending on the reference algorithm used for testing (FS and UMHexagonS); and are further divided showing the timing results for the complete H.264/AVC encoder and focusing exclusively on the proposed algorithm.

Table 5.1: Timing results of the proposed encoder for QCIF sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
Canoe	99.04	103.86	98.24	56.79	81.82	5.50	71.54	3.51
Fast food	98.70	77.07	97.65	42.61	79.28	4.83	68.43	3.17
Football	98.97	96.68	98.12	53.11	80.57	5.15	69.99	3.33
M. calendar	98.66	74.89	97.58	41.33	75.64	4.10	63.84	2.77
Racing	98.78	82.05	97.81	45.58	78.34	4.62	67.38	3.07
Scrolltext	98.29	58.45	96.95	32.74	72.43	3.63	60.36	2.52
Skyline	97.99	49.63	96.41	27.85	68.71	3.20	56.09	2.28
Softfootball	99.05	105.08	98.26	57.31	81.89	5.52	71.59	3.52
Average	98.68	76.00	97.63	42.12	77.33	4.41	66.15	2.95

Table 5.2: Timing results of the proposed encoder for CIF sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
Canoe	99.12	113.91	98.35	60.51	84.58	6.48	74.88	3.98
Fast food	98.77	81.58	97.73	44.02	81.26	5.34	70.66	3.41
Football	99.01	101.01	98.16	54.21	82.41	5.68	72.10	3.58
M. calendar	98.79	82.85	97.75	44.48	78.30	4.61	66.77	3.01
Racing	99.07	107.33	98.27	57.67	82.56	5.73	72.37	3.62
Scrolltext	98.15	53.94	96.63	29.67	73.39	3.76	61.36	2.59
Skyline	98.06	51.47	96.46	28.29	70.72	3.42	58.21	2.39
Softfootball	99.15	118.26	98.41	62.93	84.54	6.47	74.89	3.98
Average	98.77	81.01	97.72	43.85	79.72	4.93	68.90	3.22

Table 5.3: Timing results of the proposed encoder for VGA sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
Canoe	99.06	106.37	98.23	56.63	85.08	6.70	75.65	4.11
Fast food	98.85	86.76	97.86	46.64	82.01	5.56	71.58	3.52
Football	98.98	97.69	98.09	52.44	82.08	5.58	71.65	3.53
M. calendar	98.75	80.23	97.66	42.72	78.87	4.73	67.27	3.06
Racing	99.05	105.73	98.23	56.48	83.35	6.01	73.33	3.75
Scrolltext	98.04	51.07	96.43	27.99	73.95	3.84	61.83	2.62
Skyline	98.02	50.42	96.37	27.52	70.80	3.42	58.06	2.38
Softfootball	99.13	114.43	98.35	60.75	84.86	6.61	75.28	4.04
Average	98.73	79.02	97.65	42.60	80.13	5.03	69.33	3.26

For all of them, when compared with the FS algorithm, the speed–ups of the algorithm range from 49x to 118x, which means that the speed–ups for the complete H.264/AVC encoder range from 27x to 60x. When compared with the UMHexagonS algorithm, the speed–ups of the algorithm are ranging from 3.2x to 6.7x, which means that the speed–ups for the complete H.264/AVC encoder range from 2.2x to 4.1x.

Table 5.4 shows the average execution time increments of the reference algorithms (FS and UMHexagonS) and the average execution time increments of the proposed algorithm, when coding CIF video sequences instead of coding QCIF video sequences. The resolution of the video sequences is incremented by four (CIF format is four times bigger than QCIF), so a priori, the execution time should also increase by a factor of four. However, the expected execution time increment is not obtained.

Table 5.4: ΔTime from QCIF to CIF video sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32		
Search algorithm	ΔTime	Expected ΔTime
FS	3.92	4
UMHexaonS	4.10	
Proposal	3.67	

The ΔTime obtained when using the FS algorithm is 3.92. It is lower than 4 due to the fact that the early–out termination mechanism is able to save more computation when coding CIF video sequences than when coding QCIF video sequences.

The ΔTime obtained when using the UMHexagonS algorithm is 4.10. It is greater than 4, which means that the algorithm requires more iterations to find the best MVs when coding CIF video sequences than when coding QCIF video sequences.

The ΔTime obtained when using the proposed algorithm is 3.67, which means that the algorithm obtains a higher instruction throughput when coding CIF video sequences than when coding QCIF video sequences. In other words, when coding QCIF video sequences the maximum instruction throughput that this algorithm can obtain for the GPU used is not achieved.

As a final conclusion, the ΔTime obtained when using the proposed algorithm is lower than the ΔTime obtained when using the reference algorithms, so the speed–ups obtained are greater when coding CIF video sequences than when coding QCIF video sequences (see Table 5.1 and Table 5.2).

Table 5.5 shows the average execution time increments of the reference algorithms (FS and UMHexagonS) and the average execution time increments of the proposed algorithm, when coding VGA video sequences instead of coding CIF video sequences. The resolution of the video sequences is incremented by three (VGA format is three times bigger than CIF), so a priori, the execution time should also increase by a factor of three. However, the expected execution time increment is not obtained.

Table 5.5: ΔTime from CIF to VGA video sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32		
Search algorithm	ΔTime	Expected ΔTime
FS	2.92	3
UMHexaonS	3.03	
Proposal	2.97	

The conclusions for the ΔTime when using the reference algorithms are similar to the ones obtained in the previous analysis. However, the ΔTime obtained when using the proposed algorithm is near the expected one, which means that when coding CIF video sequences the maximum instruction throughput is nearly obtained.

The ΔTime obtained when using the proposed algorithm is greater than the ΔTime obtained when using the FS algorithm, so the speed–ups obtained are lower when coding VGA video sequences than when coding CIF video sequences. The ΔTime obtained when using the proposed algorithm is lower than the ΔTime obtained when using the UMHexagonS algorithm, so the speed–ups are greater (see Table 5.2 and Table 5.3).

Finally, Table 5.6 and Table 5.7 show the timing results when coding HD video sequences. Note that the video sequences analysed are different from the ones used for lower resolutions, so a comparison with previous results would not be fair.

Table 5.6: Timing results of the proposed encoder for 720p sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
City	98.58	70.20	97.32	37.31	72.78	3.67	59.93	2.50
Crew	98.60	71.67	97.40	38.39	74.54	3.93	62.12	2.64
Dolphins	98.49	66.05	97.17	35.38	74.36	3.90	61.80	2.62
Harbour	98.45	64.37	97.09	34.31	69.45	3.27	56.19	2.28
Mobcal	98.63	73.20	97.44	38.99	74.80	3.97	62.35	2.66
Night	98.34	60.33	96.92	32.42	69.72	3.30	56.69	2.31
Park run	98.99	99.27	98.07	51.83	78.54	4.66	66.51	2.99
Shields	98.65	74.17	97.46	39.40	76.68	4.29	64.57	2.82
Average	98.59	71.01	97.36	37.84	73.86	3.83	61.27	2.58

Table 5.7: Timing results of the proposed encoder for 1080p sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
Crowd	98.94	94.62	97.94	48.64	78.26	4.60	65.70	2.92
Ducks	99.09	110.31	98.22	56.20	78.87	4.73	66.25	2.96
Into tree	98.90	91.29	97.90	47.54	73.47	3.77	60.35	2.52
Old town	98.53	68.08	97.18	35.49	69.67	3.30	55.76	2.26
Park joy	99.10	111.11	98.22	56.21	80.28	5.07	68.04	3.13
Pedestrian	98.51	67.31	97.19	35.60	74.32	3.89	61.38	2.59
Riverbed	99.09	110.09	98.24	56.74	82.60	5.75	71.44	3.50
Tractor	98.80	83.32	97.66	42.71	79.78	4.94	67.57	3.08
Average	98.87	88.67	97.82	45.85	77.15	4.38	64.56	2.82

The algorithm's speed–ups range from 66x to 99x when compared with the FS algorithm for coding 720p video sequences, and from 67x to 111x for coding 1080p video sequences. For both resolutions, the overall speed–ups obtained for the complete H.264/AVC encoder range from 32x to 56x.

When compared with the UMHexagonS algorithm, the algorithm's speed–ups range from 3.2x to 4.6x for coding 720p video sequences, and from 3.3x to 5.7x for coding 1080p video sequences. For both resolutions, the overall speed–ups obtained for the complete H.264/AVC encoder range from 2.2x to 3.5x.

RD results

From Table 5.8 to Table 5.12 the RD results of the proposed algorithm are presented. The proposed algorithm is tested against two search algorithms implemented in the H.264/AVC reference encoder, so the results are divided into two main parts: the comparison against the FS algorithm and the comparison against the UMHexagonS algorithm.

Table 5.8, Table 5.9 and Table 5.10 show the RD results when coding the same QCIF, CIF and VGA video sequences. In general, the results show that the proposed algorithm obtains a slightly lower encoding efficiency than the FS algorithm (the RD degradation is negligible if the computational savings are taken into account), but it surpasses the encoding efficiency obtained by the UMHexagonS algorithm.

Table 5.8: RD results of the proposed encoder for QCIF sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
Canoe	1.57	-0.052	-4.45	0.151
Fast food	2.76	-0.093	-5.31	0.173
Football	2.62	-0.084	-3.25	0.105
M. calendar	2.75	-0.094	0.14	-0.004
Racing	3.90	-0.121	-6.63	0.228
Scrolltext	2.20	-0.075	-8.24	0.337
Skyline	3.70	-0.107	-3.03	0.100
Softfootball	2.38	-0.074	-3.61	0.118
Average	2.74	-0.088	-4.30	0.151

When compared with the FS algorithm for coding QCIF and CIF video sequences, the bit rate increments range from 1.57% to 5.43%, which is an acceptable RD degradation. However, when coding VGA video sequences there are some cases which lead to a greater RD degradation. These sequences have a background which is in movement, or the camera is in movement, or both at the same time. These situations, in which real MVPs are not available, cause certain bit rate increments (see Equation 5.1). Note that, when comparing these sequences with the execution of the UMHexagonS algorithm, bit rate decrements over 10% are obtained for all of them.

On the other hand, when compared with the UMHexagonS algorithm, the proposed algorithm obtains bit rate decrements for all tested sequences, except for M. calendar in QCIF. The bit rate decrements obtained are of up to 18.9%.

Table 5.9: RD results of the proposed encoder for CIF sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
Canoe	1.95	-0.068	-4.74	0.161
Fast food	3.65	-0.107	-7.57	0.253
Football	2.46	-0.076	-4.61	0.148
M. calendar	2.53	-0.093	-3.20	0.121
Racing	5.43	-0.175	-18.93	0.803
Scrolltext	4.64	-0.166	-16.37	0.697
Skyline	4.85	-0.142	-6.07	0.193
Softfootball	2.99	-0.095	-3.78	0.122
Average	3.56	-0.115	-8.16	0.312

Table 5.10: RD results of the proposed encoder for VGA sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
Canoe	1.45	-0.044	-7.40	0.247
Fast food	3.20	-0.078	-8.31	0.224
Football	2.43	-0.063	-5.16	0.142
M. calendar	2.10	-0.072	-10.86	0.354
Racing	6.37	-0.196	-15.43	0.566
Scrolltext	11.63	-0.394	-15.84	0.622
Skyline	7.44	-0.201	-10.69	0.325
Softfootball	2.56	-0.073	-4.95	0.150
Average	4.65	-0.140	-9.83	0.329

Finally, Table 5.11 and Table 5.12 show the RD results when coding HD video sequences. Note that the video sequences analysed are different from the ones used for lower resolutions, so a comparison with previous results would not be fair.

The bit rate increments range from 0.75% to 5.12% when compared with the FS algorithm for coding 720p video sequences, and from 0.42% to 9.76% for coding 1080p video sequences. Note that, as was the case when coding VGA video sequences, there is one video sequence for which a greater Δbit rate is reported (Tractor in 1080p format). This Δbit

Table 5.11: RD results of the proposed encoder for 720p sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
City	2.30	-0.063	-10.62	0.306
Crew	4.51	-0.107	-5.71	0.136
Dolphins	3.60	-0.095	-7.24	0.207
Harbour	1.71	-0.050	-2.06	0.059
Mobcal	5.12	-0.131	-15.83	0.482
Night	1.39	-0.039	-4.99	0.147
Park run	0.75	-0.020	-4.03	0.115
Shields	3.37	-0.094	-8.70	0.248
Average	2.84	-0.075	-7.40	0.213

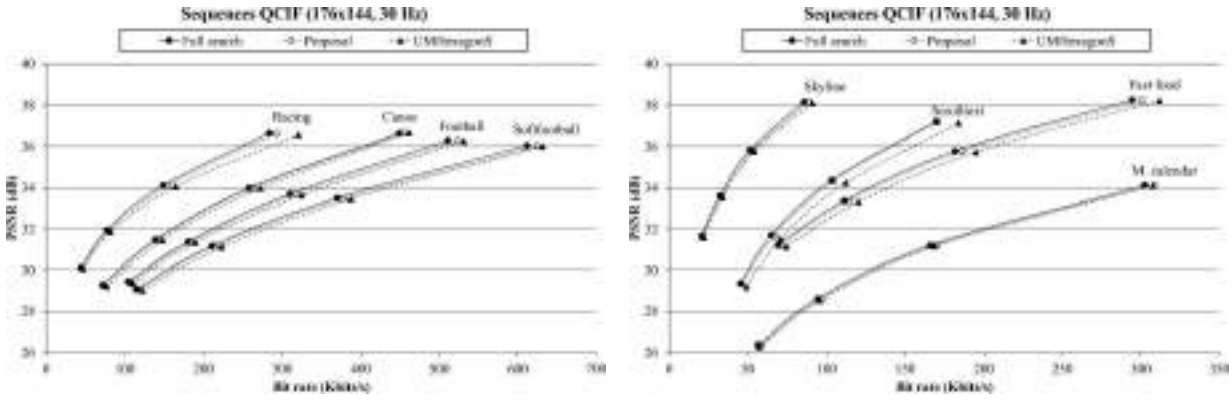
Table 5.12: RD results of the proposed encoder for 1080p sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
Crowd	3.08	-0.090	-6.63	0.203
Ducks	0.42	-0.011	-1.80	0.046
Into tree	6.01	-0.119	-13.45	0.297
Old town	3.76	-0.076	-5.73	0.111
Park joy	2.53	-0.076	-6.12	0.195
Pedestrian	4.24	-0.095	-6.21	0.148
Riverbed	1.29	-0.033	-1.31	0.035
Tractor	9.76	-0.268	-22.63	0.769
Average	3.89	-0.096	-7.99	0.226

rate is obtained because this sequence has a homogeneous background and the camera is in movement, which is not the optimal scenario for the proposed MVPs.

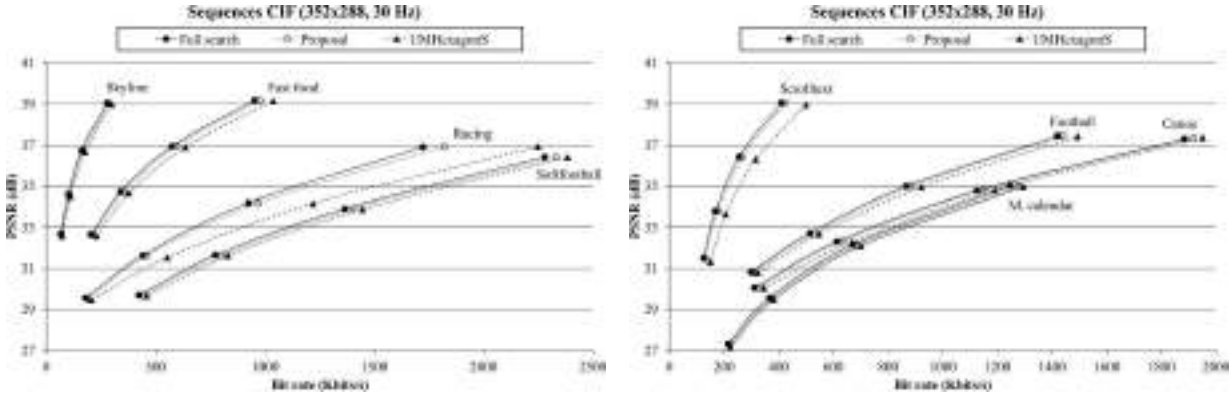
On the other hand, when compared with the UMHexagonS algorithm for coding 720p video sequences, the bit rate decrements range from 2.06% to 15.83%, and from 1.31% to 22.63% for coding 1080p video sequences.

From Figure 5.3 to Figure 5.7 the RD graphic results for the reference algorithms (FS and UMHexagonS) and for the proposed algorithm are presented. The results are presented for all video sequence resolutions, from a value of 28 to 40 for QP. For clarity the RD curves of each resolution are split into two independent graphs.



(a) Part I (b) Part II

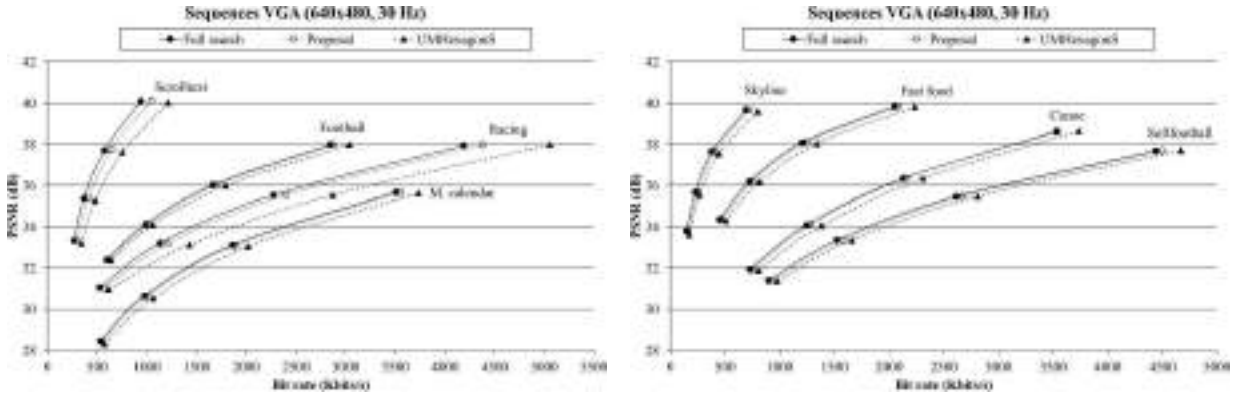
Figure 5.3: RD graphic results of the proposed encoder for QCIF sequences.



(a) Part I (b) Part II

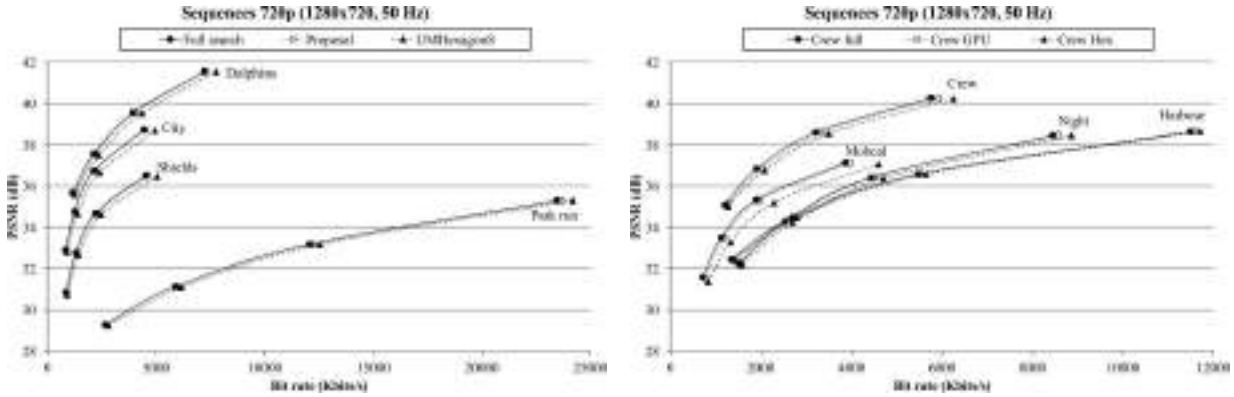
Figure 5.4: RD graphic results of the proposed encoder for CIF sequences.

As can be seen from the figures, the PSNR versus bit rate obtained with the proposed H.264/AVC encoder, based on our algorithm, in general slightly deviates from the results obtained when using the FS algorithm, except in some cases in which greater bit rates are required for a given PSNR value (e.g. the Racing and Scrolltext video sequences for low resolutions). However, when compared with the results obtained when using the UMHexagonS algorithm, certain differences can be appreciated, as the H.264/AVC encoder using the proposed algorithm is able to obtain similar PSNR values while using lower bit rates.



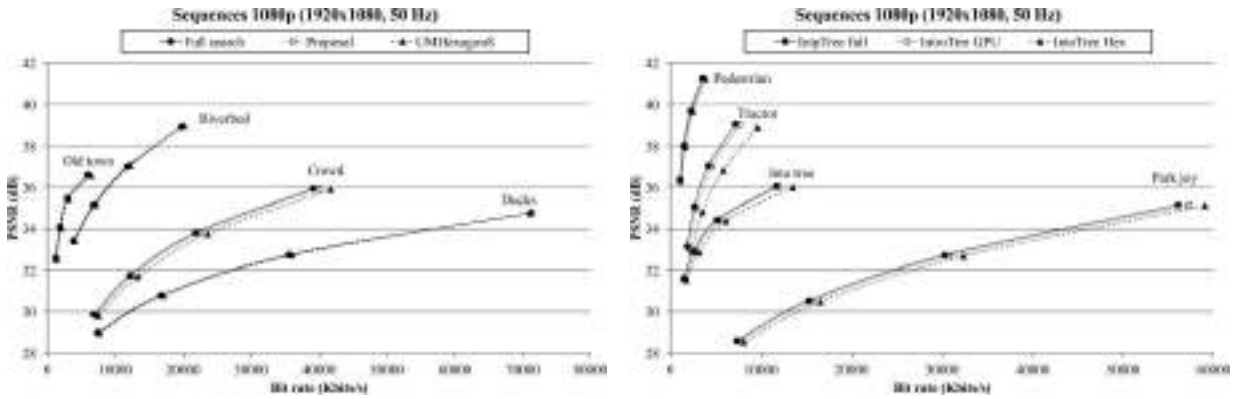
(a) Part I (b) Part II

Figure 5.5: RD graphic results of the proposed encoder for VGA sequences.



(a) Part I (b) Part II

Figure 5.6: RD graphic results of the proposed encoder for 720p sequences.



(a) Part I (b) Part II

Figure 5.7: RD graphic results of the proposed encoder for 1080p sequences.

Mode decisions

Figures 5.9 to 5.12 show the inter prediction mode decisions made for one frame by the H.264/AVC encoder when using the FS algorithm (Figures 5.9a, 5.10a, 5.11a and 5.12a) and for one frame made by our proposed algorithm (Figures 5.9b, 5.10b, 5.11b and 5.12b). The mode decisions are obtained when analysing the output bit streams obtained in order to carry out the timing and RD evaluation, with a QP value of 28.

Figure 5.8 shows the different kinds of inter prediction modes available for B frames. The prediction modes are the same as for P frames, but adding the Direct mode. Note that, in order to obtain the mode decisions carried out by the reference and the proposed H.264/AVC encoders, a free-ware software is used which does not support 1080p sequences.

Figure 5.9, Figure 5.10 and Figure 5.11 show the mode decisions for the 28th frame of the Scrolltext video sequence in QCIF, CIF and VGA format, respectively; Figure 5.12 shows them for the 5th frame of the Crew video sequence in 720p format. The Scrolltext and Crew video sequences are chosen since these video sequences are the ones which produce one of the greatest bit rate increments in comparison with the FS algorithm. The 5th frame is selected since it is a B frame located in the middle of the GOP pattern configured, while the 28th is selected because of the same reason but also because the background movement in this sequence starts around the 20th frame. The background movement is responsible for the bit rate increments reported by this video sequence.

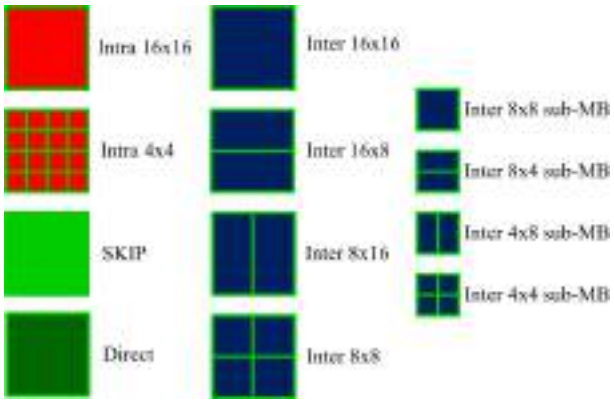
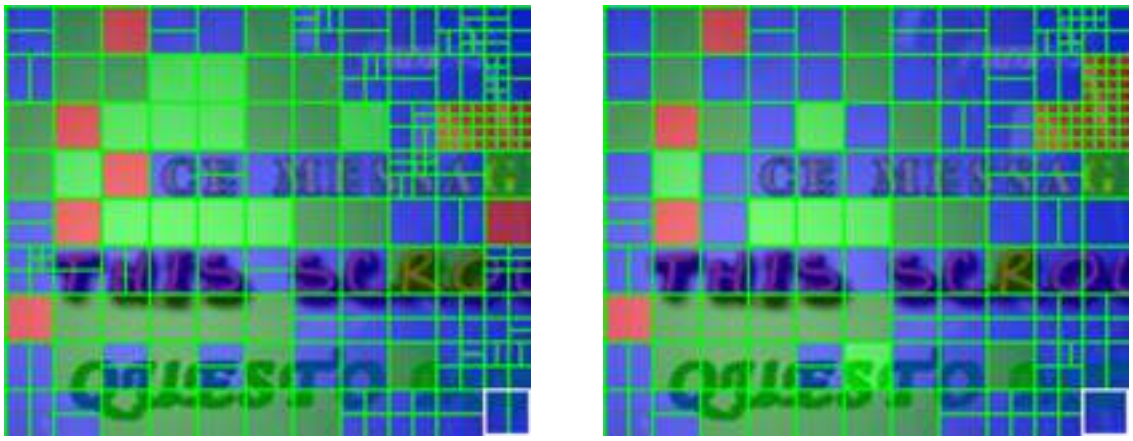


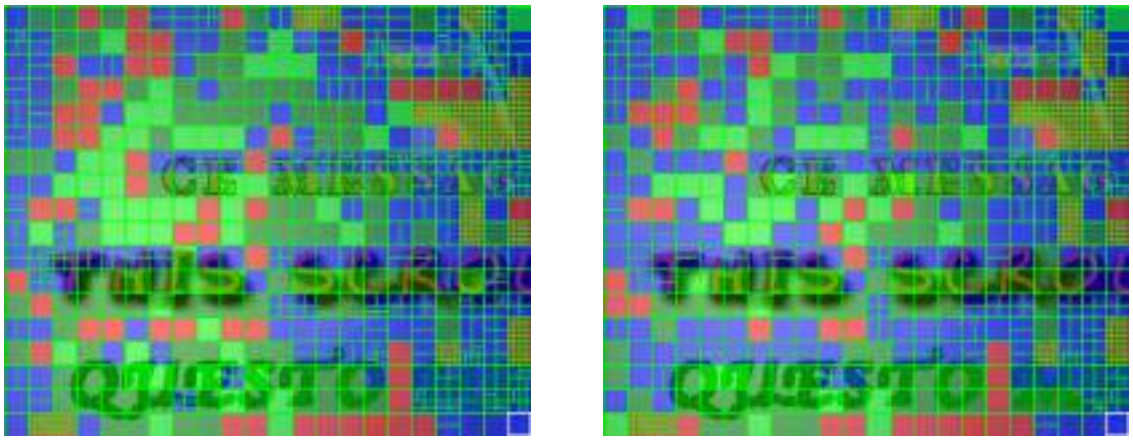
Figure 5.8: Different kinds of predictions in B frames.

As is shown in the figures, different MB labels are generated by our approach, but in general these partitions are close to the partitions made by the H.264/AVC encoder using the FS algorithm. As a result, although the label partitions are not the same, our closed decisions do not have a significant impact on the final quality performance. The different decision is mainly due to the lack of real MVPs in the cost calculations. In this case two MVs are required to calculate the encoding cost when the Bi–directional prediction is selected (see Equation 5.1) and one when the forward or backward prediction is selected (see Equation 4.1). Moreover, multiple reference frames also affect the mode decisions, since more MVs are candidates to be selected and more encoding costs must be calculated.



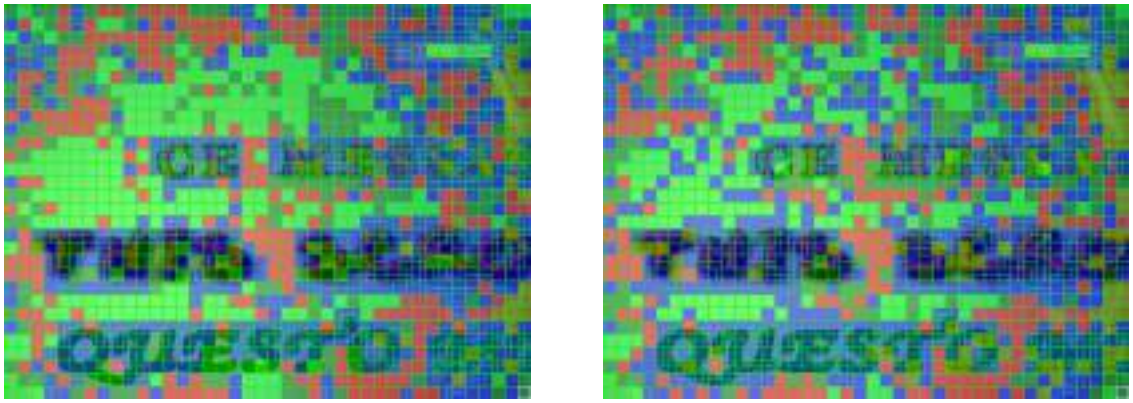
(a) Full search (b) Proposal

Figure 5.9: Mode decisions for Scrolltext sequence in QCIF format.



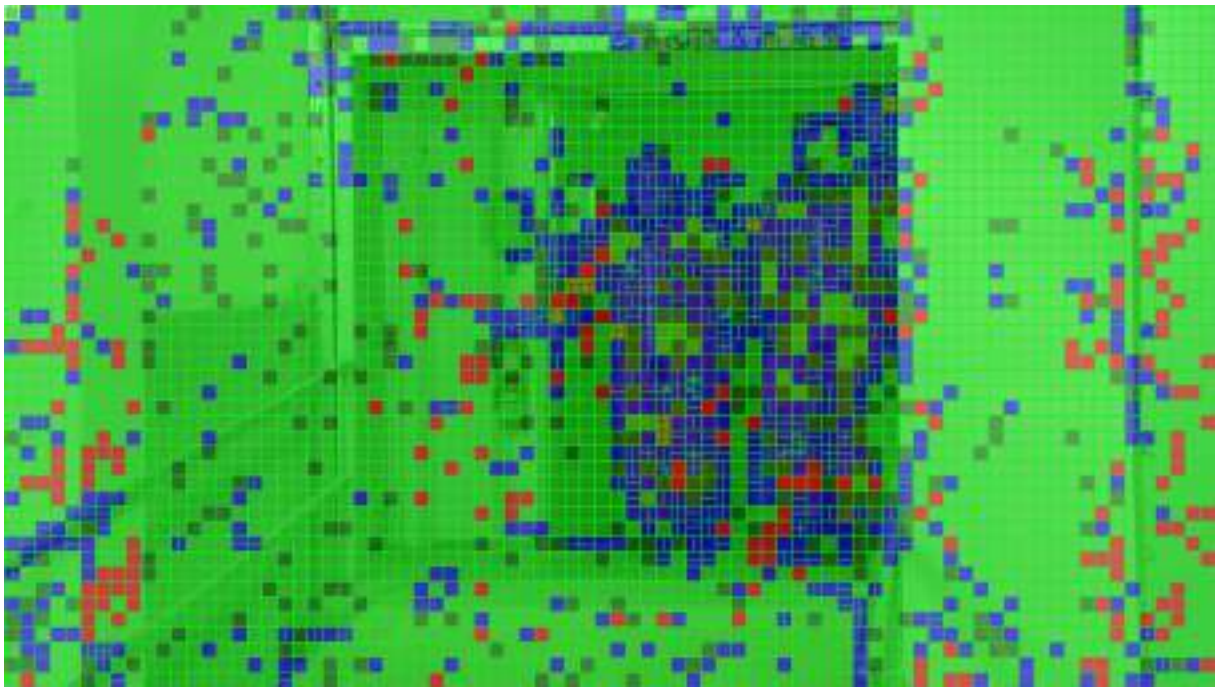
(a) Full search (b) Proposal

Figure 5.10: Mode decisions for Scrolltext sequence in CIF format.

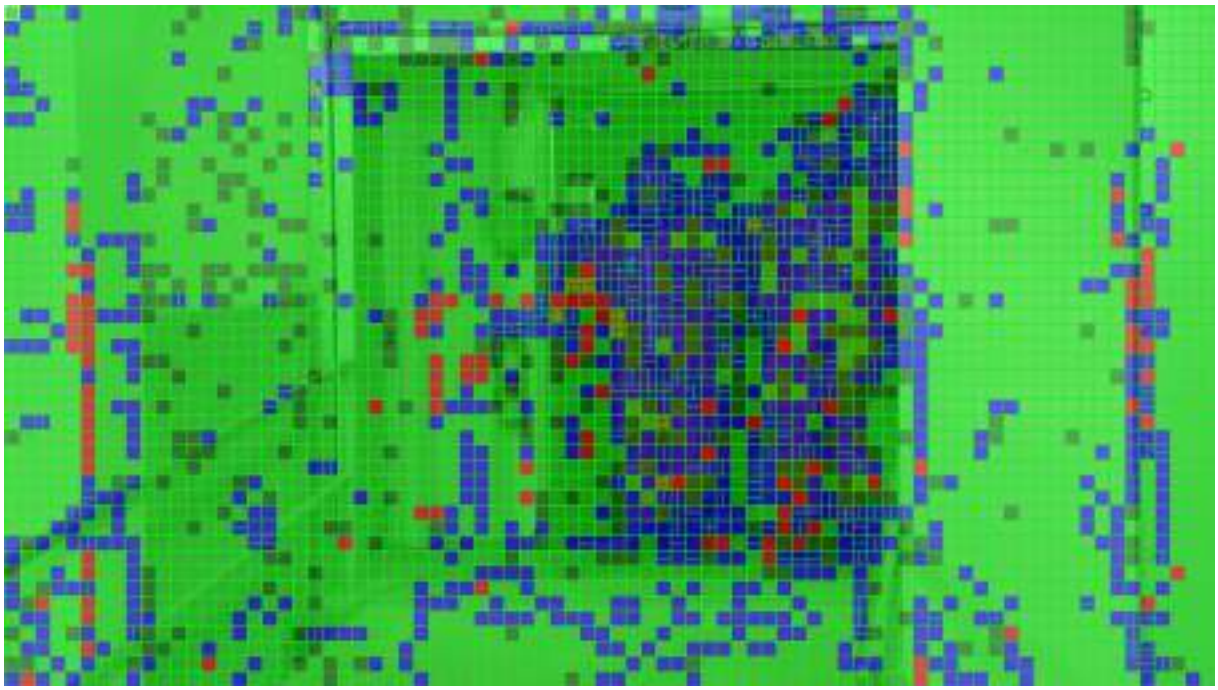


(a) Full search (b) Proposal

Figure 5.11: Mode decisions for Scrolltext sequence in VGA format.



(a) Full search



(b) Proposal

Figure 5.12: Mode decisions for Crew sequence in 720p format.

Power and energy results

From Table 5.13 to Table 5.16 the power and energy consumption results of the proposed algorithm are presented. These tables show the average power consumption, the execution time and the total energy consumed when coding one *mini GOP* (9 frames) for the complete test computer when coding the tested sequences in CIF, VGA, 720p and 1080p format. As in Chapter 4, the analysis for QCIF video sequences is not shown. All the tables are mainly divided depending on the algorithm used to encode the video sequences and include two columns showing by how many times the proposed algorithm consumes less energy than the reference algorithms (FS and UMHexagonS).

Table 5.13: Energy consumption for coding a GOP. CIF sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32											
Sequence	Full search			UMHexagonS			Proposal				
	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Ratio FS	Ratio UMHS
Canoe	182.05	79.86	14.54	182.43	4.91	0.90	271.08	1.47	0.40	36.48	2.25
Fast food	182.82	54.27	9.92	181.36	3.16	0.57	277.83	1.29	0.36	27.68	1.60
Football	183.15	68.76	12.59	181.73	3.65	0.66	272.71	1.31	0.36	35.25	1.86
M. calendar	183.38	59.55	10.92	182.01	3.23	0.59	275.10	1.30	0.36	30.54	1.64
Racing	183.35	51.62	9.46	181.55	3.42	0.62	278.60	1.27	0.35	26.75	1.75
Scrolltext	183.29	28.61	5.24	181.13	2.30	0.42	280.25	1.26	0.35	14.85	1.18
Skyline	182.84	35.29	6.45	181.73	2.51	0.46	279.34	1.27	0.35	18.19	1.29
Softfootball	183.30	78.07	14.31	181.21	3.77	0.68	274.35	1.32	0.36	39.52	1.89
Average	183.02	57.00	10.43	181.64	3.37	0.61	276.16	1.31	0.36	28.66	1.68

The average power consumption for the H.264/AVC encoder using the reference algorithms (FS and UMHexagonS) is almost constant and ranges from 181 W to 187 W, regardless of the video sequence resolution. However, the execution time considerably varies, and as a consequence the overall energy consumption varies greatly. For example, when coding one *mini GOP* in 1080p format, the total energy consumption ranges from 221.9 kJ to 334.2 kJ, because the execution time ranges from 1,201.47 s to 1,819.60 s.

The average power consumption for the H.264/AVC encoder using the proposed algorithm increases slightly when the resolution is increased. The GPU's instruction throughput has reached its maximum, and the execution time increments for the different formats are slightly greater than expected (this overhead is caused by the memory transference from/to the GPU). As a consequence, the percentage of time in which the GPU is working increases slightly, raising the average power consumption. However, in this case the execution time is very similar, and as a consequence, the overall energy consumption does not vary significantly. For example, when coding one *mini GOP* in 1080p format, the total energy consumption ranges from 7.14 kJ to 7.65 kJ, because the execution time ranges from 24.83 s to 25.94 s.

Table 5.14: Energy consumption for coding a GOP. VGA sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32											
Sequence	Full search			UMHexagonS			Proposal				
	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Ratio FS	Ratio UMHS
Canoe	183.97	231.25	42.54	183.40	14.30	2.62	278.07	3.74	1.04	40.91	2.52
Fast food	185.23	176.36	32.67	182.58	9.90	1.81	279.41	3.68	1.03	31.77	1.76
Football	185.48	197.44	36.62	182.38	10.79	1.97	279.70	3.73	1.04	35.10	1.89
M. calendar	184.76	180.40	33.33	182.16	9.79	1.78	283.40	3.73	1.06	31.53	1.69
Racing	185.34	156.02	28.92	182.08	10.51	1.91	284.48	3.67	1.04	27.70	1.83
Scrolltext	185.19	86.30	15.98	181.95	7.04	1.28	286.35	3.57	1.02	15.63	1.25
Skyline	185.22	112.45	20.83	181.99	7.40	1.35	281.68	3.60	1.01	20.54	1.33
Softfootball	185.73	211.42	39.27	182.15	11.45	2.09	284.83	3.72	1.06	37.06	1.97
Average	185.12	168.96	31.27	182.34	10.15	1.85	282.24	3.68	1.04	30.03	1.78

Table 5.15: Energy consumption for coding a GOP. 720p sequences.

H.264/AVC JM 17.2 Main profile – Search range = 32											
Sequence	Full search			UMHexagonS			Proposal				
	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Ratio FS	Ratio UMHS
City	185.36	438.19	81.22	182.95	26.81	4.90	281.72	10.76	3.03	26.79	1.62
Crew	185.93	550.55	102.36	182.87	25.66	4.69	286.70	10.69	3.06	33.40	1.53
Dolphins	185.80	527.52	98.01	182.97	31.78	5.81	286.10	10.97	3.14	31.23	1.85
Harbour	185.77	456.68	84.84	183.41	25.17	4.62	290.11	10.94	3.17	26.73	1.45
Mobcal	185.63	654.55	121.50	183.28	26.85	4.92	292.26	10.70	3.13	38.85	1.57
Night	185.32	438.36	81.24	183.29	24.80	4.55	293.21	10.79	3.16	25.68	1.44
Park run	184.80	829.99	153.38	183.34	35.29	6.47	292.81	11.03	3.23	47.49	2.00
Shields	185.75	700.23	130.07	183.57	34.11	6.26	294.69	10.75	3.17	41.06	1.98
Average	185.55	574.51	106.58	183.21	28.81	5.28	289.70	10.83	3.14	33.90	1.68

On average, the energy consumption for the H.264/AVC encoder using the proposed algorithm ranges from 28 to 35 times better than for the H.264/AVC encoder using the FS algorithm. On the other hand, the energy consumption for the H.264/AVC encoder using the proposed algorithm ranges from 1.6 to 1.7 times better than for the H.264/AVC encoder using the UMHexagonS algorithm for all video formats.

Note that there is a certain difference in the average power consumption between the H.264/AVC reference encoder (181 W to 187 W) and the proposed GPU–based encoder

Table 5.16: Energy consumption for coding a GOP. 1080p sequences.

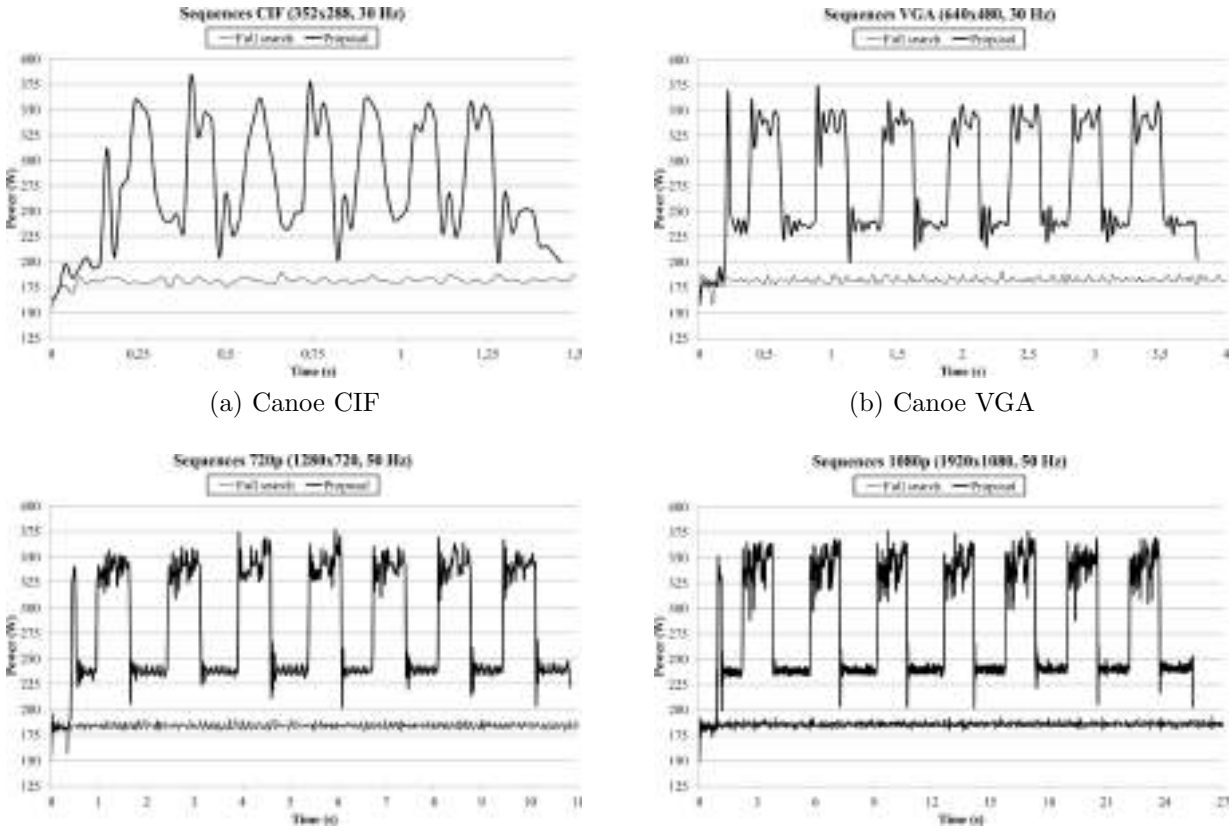
H.264/AVC JM 17.2 Main profile – Search range = 32											
Sequence	Full search			UMHexagonS			Proposal				
	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Ratio FS	Ratio UMHS
Crowd	186.87	1,356.48	253.49	181.80	62.25	11.32	282.02	25.30	7.14	35.53	1.59
Ducks	187.40	1,471.55	275.77	182.19	65.89	12.00	287.82	25.55	7.35	37.50	1.63
Into tree	185.84	1,376.36	255.78	182.24	53.79	9.80	291.50	24.64	7.18	35.61	1.36
Old town	184.30	1,229.63	226.62	182.11	51.85	9.44	292.75	24.53	7.18	31.56	1.31
Park joy	183.68	1,819.60	334.22	182.29	76.12	13.88	293.71	25.36	7.45	44.87	1.86
Pedestrian	184.16	1,225.78	225.74	182.34	68.04	12.41	295.21	24.83	7.33	30.80	1.69
Riverbed	185.07	1,745.24	322.99	183.66	86.40	15.87	294.91	25.94	7.65	42.22	2.07
Tractor	184.73	1,201.47	221.95	183.49	73.01	13.40	295.96	25.36	7.51	29.57	1.78
Average	185.26	1,428.26	264.57	182.52	67.17	12.26	291.74	25.19	7.35	35.96	1.66

(276 W to 291 W), and the reason is that the GPU is processing for about 50% of the total encoding time.

Figure 5.13 shows an extract from the power consumption over time for the complete test computer, when coding one video sequence of each resolution for the H.264/AVC encoder using the FS algorithm and for the H.264/AVC encoder using the proposed algorithm. Note that the power consumption over time when using the UMHexagonS algorithm is not shown in these graphs because it is similar to the one shown when using the FS algorithm, but the execution time is shorter (see Tables 5.13 to 5.16).

When the encoder process begins, all the encoders consume the same power (around 180–185 W), but when the GPU starts working the power consumption of the proposed encoder increases. It has a power consumption of around 325-375 W, except for CIF format in which case the power consumption peaks are not fully reconstructed (Figure 5.13). The configured GOP pattern is composed of 1 I frame followed by 7 B frames and 1 P frame where the proposed algorithms are executed, so 8 power consumption peaks can be found on each graph in Figure 5.13. Note that the encoding order is different from the visualization order, and the P frame is encoded before the B frames. The algorithm presented in Chapter 4 for P frames is less complex than the algorithm presented in this chapter for B frames. As a consequence, the power consumption peaks generated by P frames are thinner than the ones generated by B frames (see in Figure 5.13 the first power consumption peak in comparison with the other peaks).

As in the previous chapter, we would like to mention that the power consumption for the CPU code in the GPU–based encoder is around 235 W, which is higher than for the reference execution (180–185 W) because the GPU is always active, waiting for new kernels.



(a) Canoe CIF (b) Canoe VGA

(c) City 720p (d) Crowd 1080p

Figure 5.13: Power consumption graphic results of the proposed encoder.

5.4. Comparison with other known results

This section compares the results reported by our proposed algorithm developed for B frames with the ones reported in the most recent and relevant related articles. However, in the literature there are not many works aimed at reducing the H.264/AVC encoding time when using B frames. In fact, there is only one approach and it is not GPU–based.

The encoding conditions of [Liu et al. 09] have been reproduced as far as possible, and a reasonably objective and fair comparison is possible by following Bjøntegaard and Sullivan's common test rule [Sullivan and Bjontegaard 01].The comparison metrics have been produced and tabulated based on the TRs, ΔPSNR and ΔBit rate differences. The video sequences used are those used in the related publication.

Table 5.17 summarizes our main findings. As seen from the table, our proposal greatly outperforms those previously reported in terms of TR (%) for all video sequences. However, for some video sequences, greater PSNR decrements and bit rate increments are reported, but this extra penalty is negligible if the time savings are taken into account.

Table 5.17: Comparison against [Liu et al. 09]

Sequence	Format	Method	TR (%)	ΔPSNR (dB)	Δbit rate (%)
Coastguard	CIF	Proposal	98.13	-0.043	1.54
		[Liu et al. 09]	30.70	-0.063	1.67
Foreman	CIF	Proposal	97.01	-0.107	4.14
		[Liu et al. 09]	33.60	-0.068	2.08
M. calendar	CIF	Proposal	98.02	-0.089	2.58
		[Liu et al. 09]	28.80	-0.078	2.09
News	CIF	Proposal	95.75	-0.024	0.77
		[Liu et al. 09]	41.60	-0.126	2.73
Paris	CIF	Proposal	96.85	-0.046	1.35
		[Liu et al. 09]	39.10	-0.130	2.85
Silent	CIF	Proposal	96.83	-0.025	0.89
		[Liu et al. 09]	43.30	-0.034	0.90
Stefan	QCIF	Proposal	97.78	-0.103	2.76
		[Liu et al. 09]	35.20	-0.098	1.99

5.5. Conclusions

In order to reduce the high complexity of the H.264/AVC JM 17.2 reference encoder, this chapter proposed a GPU–based inter prediction algorithm developed for B frames. The proposed algorithm has been tested using different video resolutions and the most important conclusions are similar to the ones presented in Chapter 4, but there are some differences:

- When compared with the FS algorithm, the proposed algorithm for all video resolutions achieves a greater time reduction than when only using P frames (over 98%), since the algorithm developed for B frames is more complex (the percentage of time in which the GPU is in execution is higher) and the maximum GPU computational load is reached for low resolutions.
- The proposed algorithm for all video resolutions achieves acceptable RD drop penalties if the time savings are taken into account, when compared with the FS algorithm. However, for some video sequences Δbit rates of up to 11% are obtained. These increments occur because the proposed MVPs are not able to accurately calculate the encoding costs when the movement takes place in the background of the video sequences and the background is homogeneous. On the other hand, the proposed algorithm greatly surpasses the coding efficiency of the UMHexagonS algorithm, as average Δbit rate values of over 7% are obtained for all video resolutions.

- The GPU–based algorithm raises the average power consumption of the complete system, but the execution time is shorter. The average power consumption when using B frames is higher than when only using P frames because the percentage of time in which the GPU is in execution is higher. However, the TRs are higher and as a consequence the energy savings are also higher.
- Finally, the proposal is tested against one related proposal, which does not use a GPU to accelerate the inter prediction module. The proposal outperforms its timing results and equals its coding efficiency.

Chapter 6

3D Inter Prediction

IN this chapter, the proposal developed for 3D video sequences is presented. First of all, some observations about the proposed and the reference inter prediction algorithms are made. Then, the proposal is described in detail. Finally, it is evaluated. Note that this chapter does not include a comparison against related proposals because all of them have been proposed using a different reference software and the comparison would not be fair.

6.1. Introduction

As was explained in Section 2.2, when the ME algorithm is applied, in a 3D video sequence, using a reference frame from a different view (inter–view prediction), a different kind of redundancy can be eliminated. This prediction is more commonly known as DE, since it estimates the differences between adjacent viewpoints/cameras, i.e. the disparity between the different views.

ME and DE sequentially obtain the motion information (encoding costs) for all available MB partitions and sub–partitions, for all MBs in a frame. Both algorithms search for a region that minimizes the differences between the current block and the chosen block in a similar manner. In fact, the H.264/AVC JM 17.2 reference source code of both algorithms is the same.

The main challenge for the approach presented in this chapter is to efficiently support the tree-structured MC algorithm executed in the H.264/AVC JM 17.2 reference software encoder [JVT 11] for 3D video sequences.

6.2. Proposed algorithm

As has been mentioned above, the reference source code of the ME and DE algorithms is identical, so the proposed algorithm for 3D video sequences will use the proposals presented

in Chapters 4 and 5 for P frames and B frames, respectively. However, a new MVP calculation is defined, since the MVP calculation is based on previously encoded frames and the redundancies eliminated when using the ME algorithm are different from the ones eliminated when using the DE algorithm. An inter–view predicted frame cannot be properly estimated using an MVP obtained from a temporally predicted frame, or vice versa.

Due to the fact that there are two different ways of applying the proposed algorithms (ME and DE), the proposal updates the MVP in two different ways. When using a reference frame from the same view, the MVP is calculated using the 16x16 MV of the MB located in the same position but in the previously temporally predicted frame. When using a reference frame from a different view, the MVP is calculated using the 16x16 MV of the MB located in the same position but in the previously inter–view predicted frame.

Figure 6.1 shows the MVP calculation described above when using I and P frames. The same procedure is easily extended to also using B frames.

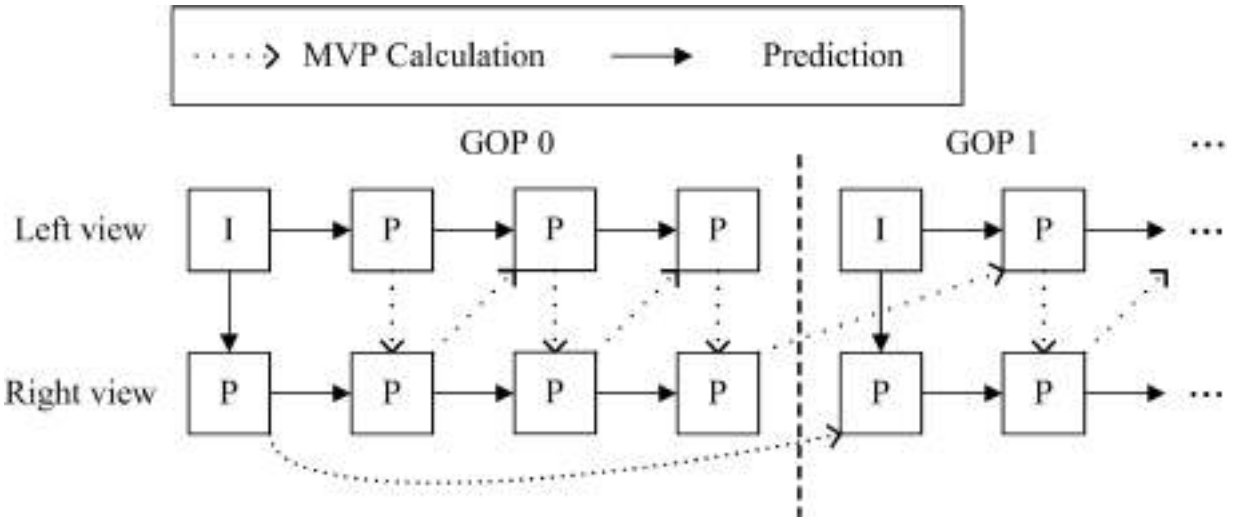


Figure 6.1: 3D MVP calculation.

6.3. Performance evaluation

In this section, the results of applying the proposal described in this chapter are presented. First of all, the encoding conditions and the metrics used to evaluate the proposal are described. Then, the evaluation is carried out.

6.3.1. Encoding conditions

In order to evaluate the proposed inter prediction algorithm developed for 3D sequences, it has been integrated into the H.264/AVC JM 17.2 reference encoder [JVT 11]. The

H.264/AVC encoding parameters used for the evaluation are those included in the Stereo High profile of the said reference encoder. Furthermore, two different coding scenarios have been used to test the proposal: a simple scenario in which only P frames are used (this scenario is based on the one used in the performance evaluation section of Chapter 4) and a more complex scenario in which P and B frames are used, as well as multiple reference frames (this scenario is based on the one used in the performance evaluation section of Chapter 5).

First of all, the common conditions for both scenarios are described, and after that, the specific conditions for each of them are detailed:

- RD–Optimization is disabled to keep the complexity as low as possible. However, in Appendix A the algorithms proposed in this thesis are evaluated enabling this option.
- The tests are carried out with popular stereo (2 views) video sequences in 1080p format (full-HD, 1920x1080 pixels). The first frame of each of these sequences is shown in Figure 6.2. These sequences have different characteristics (content and movement features).
- The frame rate parameter is set to 25 for each view (25 Hz).
- The search range is set to 32, which means 4096 positions inside the search area of each MB partition.
- The proposed inter prediction algorithm is tested against two search algorithms implemented in the H.264/AVC JM 17.2 reference encoder: FS [Richardson 10] and UMHexagonS [Rahman and Badawy 05].

In order to make a proper comparison, an unmodified H.264/AVC JM 17.2 reference encoder is run on the same machine as the H.264/AVC JM using the proposed algorithms, with the same encoding configuration and with no calls to the GPU. The development environment, including the GPU, is the same as that used in the evaluation of Chapter 4.

P frames scenario

- The number of reference frames is set to 1 in order to keep the complexity as low as possible. An analysis using more references is possible since the algorithm can iterate over multiple reference frames. The conclusions obtained will be the same regardless of the number of reference frames configured.
- The QP is varied between 28, 32, 36 and 40, according to [Bjontegaard 01], [Sullivan and Bjontegaard 01] and [JVT Test Model Ad Hoc Group 03].
- The configured GOP pattern of the left view in the stereo video sequences is 1 I frame followed by 11 P frames (I11P), while the one configured for the right view is composed of 12 P frames. The first P frame of the right view in each GOP is inter–view predicted using the I frame of the left view, and is shown in Figure 6.3.

(a) Beergarden

(b) Cafe

(c) Car park

(d) Hall

(e) Street

Figure 6.2: Stereo 1080p sequences used to evaluate the proposal.

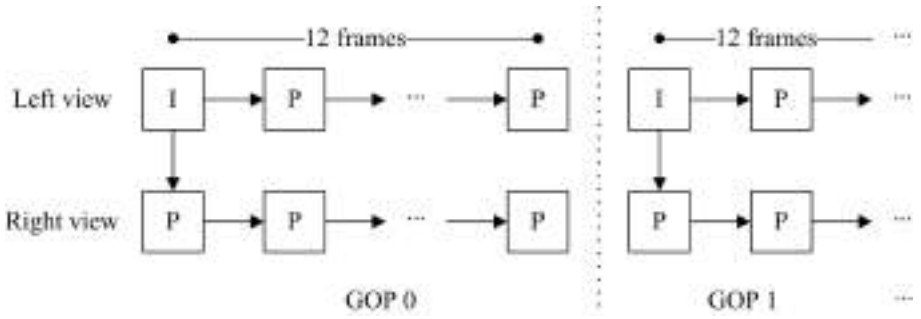


Figure 6.3: Configured GOP pattern. Only I and P frames.

P and B frames scenario

- The number of reference frames for P frames is set to 4 , while for B frames it is set to 2 in each direction, giving a total of 4 frames.

- The number of B frames inserted between each I or P frame is set to 7, and the intra period is set to 32. As a consequence, for the the left view in each GOP there are 28 B frames, 3 P frames and 1 I frame, while for the right view there are 28 B frames and 4 P frames; divided into 4 *mini GOPs*. Figure 6.4 shows the first 18 frames of the GOP configured, 9 frames per view. The first P frame of the right view in each GOP is inter–view predicted using the I frame of the left view, and is shown in Figure 6.4.

- The GOP pattern used is full hierarchy, and is depicted in Figure 6.4. Note that in the figure only the reference for the closest reference frames is depicted, but as mentioned above, multiple reference frames are used.

- The QP for P and I frames is varied among 28, 32, 36, and 40, according to [Bjontegaard 01], [Sullivan and Bjontegaard 01] and [JVT Test Model Ad Hoc Group 03]; the QP for B frames depends on the hierarchical level on which the B frame is located (see Figure 6.4) and is incremented by 1 per hierarchical level in reference with the one configured for P and I frames.

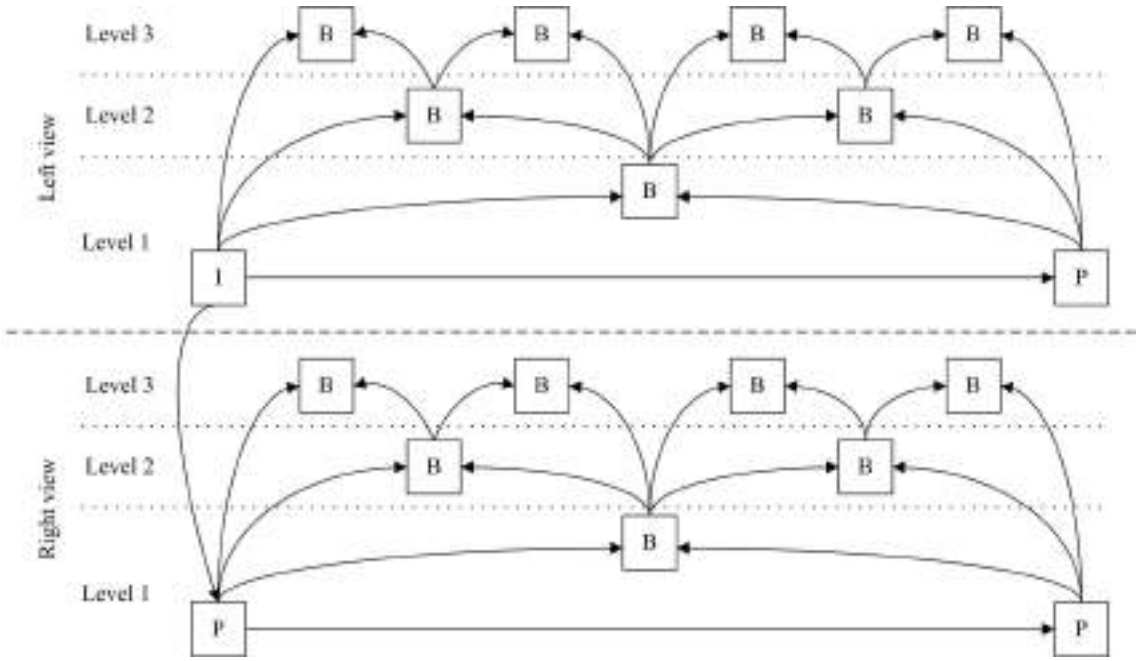


Figure 6.4: Configured GOP pattern. I, P and B frames.

6.3.2. Metrics

Different metrics have been used to evaluate the proposal. These metrics are the TR and speed–up, RD function, ΔPSNR and Δbit rate, and power and energy consumption. All these metrics have been previously defined in Section 4.3.2.

6.3.3. Results

This section presents the results obtained when coding different 3D (stereo) video sequences using the proposed inter prediction algorithms developed for P and B frames.

6.3.3.1. P frames scenario

This section presents the results obtained when coding different 3D stereo video sequences using P frames.

Timing results

Table 6.1 shows the timing results of our proposed H.264/AVC encoder when coding five 3D stereo full HD video sequences. The proposed algorithm is tested against two search algorithms, and therefore the results are divided depending on the reference algorithm used. Moreover, the results are further divided into two parts: the timing results focusing exclusively on the proposed algorithm (*ME module* column), and the timing results focusing on the complete H.264/AVC encoder (*Complete encoder* column).

Table 6.1: Timing results of the proposed encoder. I and P frames.

H.264/AVC JM 17.2 Stereo High profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
Beergarden	98.83	85.82	93.64	15.73	77.15	4.38	39.26	1.65
Cafe	97.97	49.18	89.62	9.63	71.77	3.54	33.75	1.51
Car park	98.60	71.56	92.47	13.29	73.35	3.75	34.67	1.53
Hall	98.02	50.59	89.90	9.91	73.25	3.74	35.20	1.54
Street	98.59	70.96	92.34	13.06	74.18	3.87	35.42	1.55
Average	98.40	62.64	91.60	11.90	73.94	3.84	35.66	1.55

As expected from the analysis made in Chapter 4, the proposed algorithm outperforms both search algorithms. When compared with the FS algorithm, the proposed algorithm obtains a speed–up of over 62x (TR of 98.40%) on average, which means a speed–up

of nearly 12x (TR of 91.60%) for the complete H.264/AVC encoder. Also, the proposed algorithm obtains a speed–up of over 3.8x (TR of 73.94%) on average, which means a speed–up of over 1.5x (TR of 35.66%) for the complete H.264/AVC encoder, when compared with the UMHexagonS algorithm.

RD results

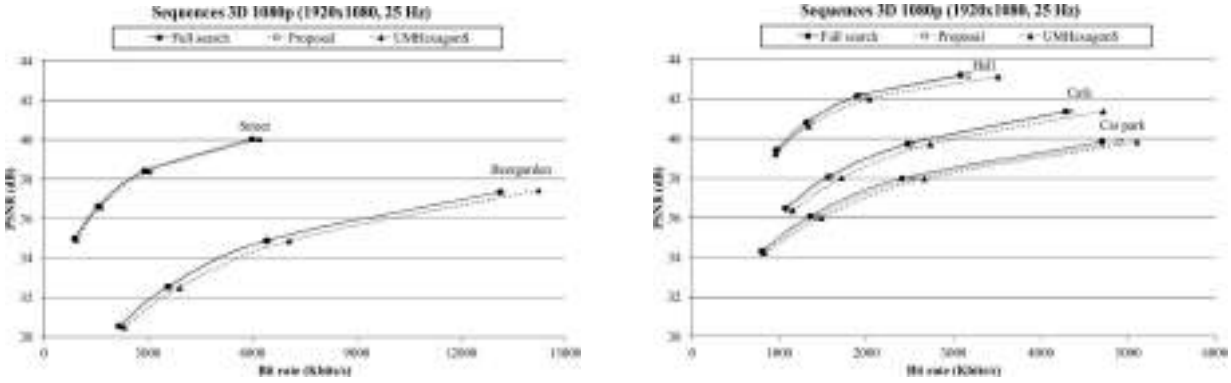
Table 6.2 shows the Δbit rate and ΔPSNR results of our proposed H.264/AVC encoder when coding five 3D stereo full HD video sequences. As in Table 6.1, the proposed algorithm is tested against the two above–mentioned search algorithms implemented by the H.264/AVC reference encoder.

Table 6.2: RD results of the proposed encoder. I and P frames.

H.264/AVC JM 17.2 Stereo High profile – Search range = 32				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
Beergarden	1.00	-0.032	-8.10	0.276
Cafe	1.74	-0.040	-8.35	0.214
Car park	4.49	-0.108	-5.25	0.138
Hall	2.30	-0.041	-6.65	0.125
Street	1.24	-0.030	-3.82	0.088
Average	2.15	-0.050	-6.43	0.168

The proposed algorithm obtains slightly worse results when compared with the FS algorithm and surpasses the results obtained by the UMHexagonS algorithm. The proposed algorithm obtains, on average, a bit rate increment of 2.15% and a PSNR loss of 0.050 dB when compared with the FS algorithm. On the other hand, the proposed algorithm obtains, on average, a bit rate decrement of 6.43% and a PSNR gain of 0.168 dB when compared with the UMHexagonS algorithm.

Figure 6.5 shows the RD graphic results obtained when using the reference algorithms (FS and UMHexagonS) and when using the proposed approach, for different 3D stereo sequences in 1080p format, from a value of 28 to 40 for QP. For clarity the RD graphs are split into two sub–figures. As can be seen from the figure, the PSNR versus bit rate obtained with the proposed encoder, based on our algorithm, deviates slightly from the results obtained when applying the FS algorithm and improves upon the results when applying the UMHexagonS algorithm, requiring lower bit rates for a given PSNR value.



(a) Part I (b) Part II

Figure 6.5: RD graphic results of the proposed encoder. I and P frames.

Power and energy results

Table 6.3 shows the average power, time and energy consumed when coding one GOP (24 frames, 12 frames per view) for the complete test computer when coding five 3D stereo 1080p sequences. The first main column shows these results for the H.264/AVC encoder using the FS algorithm, the second main column shows them for the H.264/AVC encoder using the UMHexagonS algorithm and the third main column shows them for the H.264/AVC encoder using the proposed algorithm. Additionally, Table 6.3 includes two columns showing by how many times the proposed algorithm consumes less energy than the reference algorithms (FS and UMHexagonS).

Table 6.3: Energy consumption for coding a GOP. I and P frames.

H.264/AVC JM 17.2 Stereo High profile – Search range = 32											
Sequence	Full search			UMHexagonS			Proposal				
	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Ratio FS	Ratio UMHS
Beergarden	184.59	613.68	113.28	184.92	60.29	11.15	246.69	40.57	10.00	11.32	1.11
Cafe	184.13	427.17	78.65	182.30	57.44	10.47	247.04	40.63	10.04	7.84	1.04
Car park	184.65	531.32	98.11	183.93	56.32	10.36	248.66	40.98	10.19	9.63	1.02
Hall	183.82	482.67	88.72	184.77	54.81	10.13	249.83	40.30	10.07	8.81	1.01
Street	183.62	583.45	107.13	184.83	56.38	10.42	250.96	40.91	10.27	10.43	1.02
Average	184.16	527.66	97.18	184.15	57.05	10.51	248.64	40.68	10.11	9.61	1.04

On average, the energy consumption for the GPU–based encoder is 9.61 times better than for the H.264/AVC encoder using the FS algorithm, and is almost the same as for the H.264/AVC encoder using the UMHexagonS algorithm. This behaviour when compared with the UMHexagonS algorithm is due to the fact that the speed–ups obtained are not good enough to reduce energy consumption.

Figure 6.6 shows the power consumption over time for the complete test computer, when coding the Beergarden sequence for both the H.264/AVC encoder using the FS algorithm and for the H.264/AVC encoder using our proposed algorithm. Note that only an extract of the complete graph is shown in order to analyse in detail the power consumption. The H.264/AVC encoder using the UMHexagonS algorithm is not included in the graph since its power consumption is almost the same as the one obtained for the H.264/AVC using the FS algorithm, but the execution time is shorter.

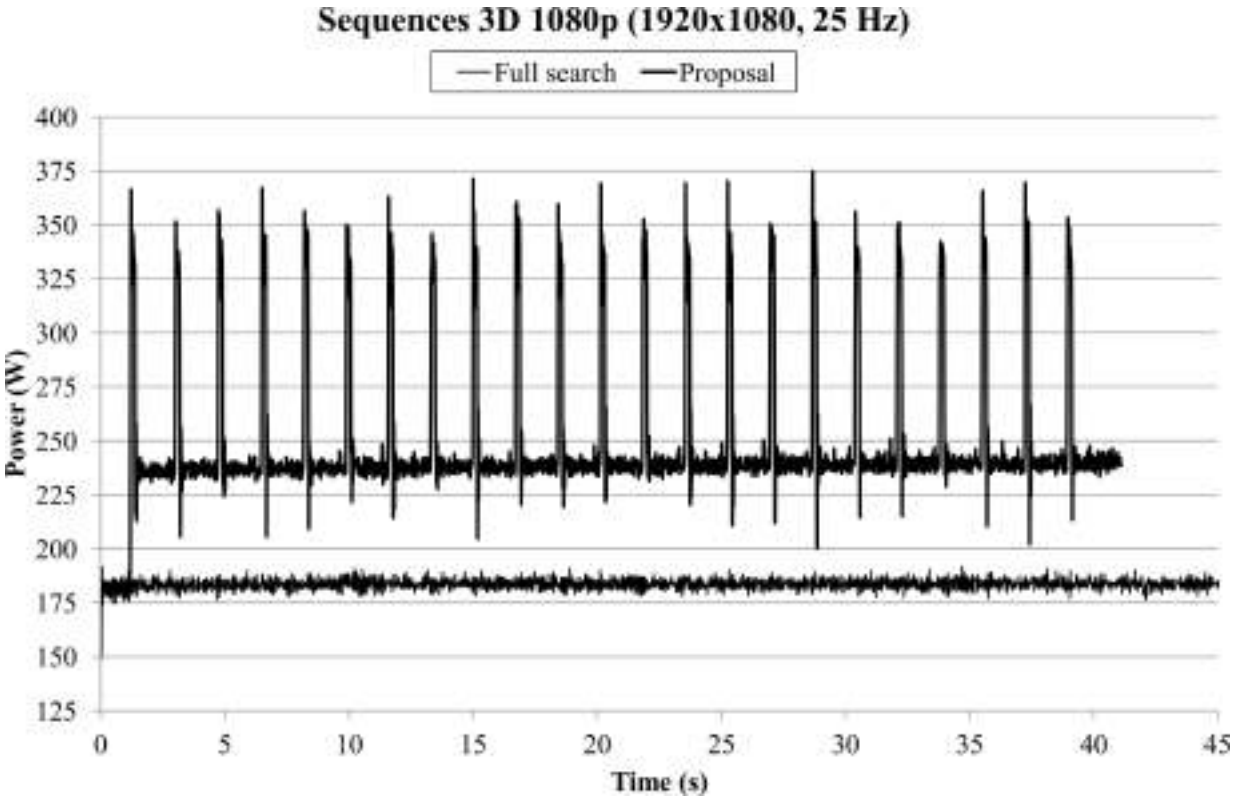


Figure 6.6: Power consumption graphic results of the proposed encoder. I and P frames of Beergarden sequence.

When the encoder process begins, both encoders consume the same power (180-185 W), but when the GPU starts working the power consumption of the proposed encoder increases, having a power consumption around 325-375 W (see power consumption peaks in Figure 6.6). 23 power consumption peaks can be identified in Figure 6.6 since in the configured GOP pattern there are one I frame, and 23 P frames where the proposed algorithm is executed.

6.3.3.2. P and B frames scenario

This section presents the results obtained when coding different 3D stereo video sequences using P and B frames.

Timing results

Table 6.4 shows the timing results of our proposed H.264/AVC encoder when coding five 3D stereo full HD video sequences. The proposed algorithm is tested against two search algorithms, and therefore the results are divided depending on the reference algorithm used. Moreover, the results are further divided into two parts: the timing results focusing exclusively on the proposed algorithm (*ME module* column), and the timing results focusing on the complete H.264/AVC encoder (*Complete encoder* column).

Table 6.4: Timing results of the proposed encoder. I, P and B frames.

H.264/AVC JM 17.2 Stereo High profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
Beergarden	98.37	61.43	96.73	30.61	66.82	3.01	51.42	2.06
Cafe	96.97	33.01	94.09	16.92	60.25	2.52	45.33	1.83
Car park	97.98	49.48	95.99	24.96	61.05	2.57	45.99	1.85
Hall	97.39	38.27	94.89	19.57	65.66	2.91	50.77	2.03
Street	97.97	49.16	95.97	24.79	62.80	2.69	47.61	1.91
Average	97.74	44.15	95.53	22.40	63.32	2.73	48.23	1.93

As expected from the analysis made in Chapter 5, the proposed algorithm outperforms both search algorithms. However, the TRs and speed–ups obtained for the proposed algorithm are lower than the ones obtained when using the *P frames scenario*. This behaviour occurs because the reference algorithms can save computation due to the fact that there are multiple reference frames configured. The execution time of the proposed algorithm is almost constant (it is content independent), while the reference search algorithms are content dependent. The FS algorithm is implemented using an early–out termination which is able to skip some search area positions based on the cost obtained for previously checked positions, and the UMHexagonS algorithm can carry out less algorithm iterations.

On the other hand, the TRs and speed–ups obtained for the complete H.264/AVC encoder are greater than the ones obtained when using the *P frames scenario* because the percentage of time spent by the ME module is greater.

The proposed algorithm obtains a speed–up of over 44x (TR of 97.74%) on average, which means a speed–up of over 22x (TR of 95.53%) for the complete H.264/AVC encoder, when compared with the FS algorithm. Also, the proposed algorithm obtains a speed–up of over 2.7x (TR of 63.32%) on average, which means a speed–up of nearly 2x (TR of 48.23%) for the complete H.264/AVC encoder, when compared with the UMHexagonS algorithm.

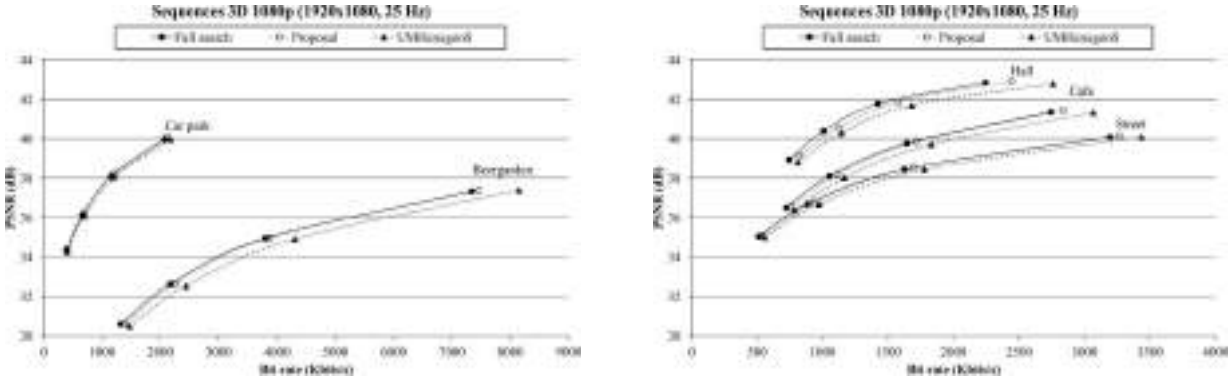
RD results

Table 6.5 shows the Δbit rate and ΔPSNR results of our proposed H.264/AVC encoder when coding five 3D stereo full HD video sequences. As in the previous table, the proposed algorithm is tested against the two above–mentioned search algorithms implemented by the H.264/AVC reference encoder.

Table 6.5: RD results of the proposed encoder. I, P and B frames.

H.264/AVC JM 17.2 Stereo High profile – Search range = 32				
	Full search		UMHexagonS	
Sequence	Δbit rate (%)	ΔPSNR (dB)	Δbit rate (%)	ΔPSNR (dB)
Beergarden	2.21	-0.069	-10.21	0.354
Cafe	2.64	-0.058	-9.04	0.224
Car park	-0.07	0.002	-5.63	0.149
Hall	6.96	-0.125	-9.71	0.182
Street	1.55	-0.033	-7.26	0.178
Average	2.66	-0.057	-8.37	0.217

Figure 6.7 shows the RD graphic results obtained when using the reference algorithms (FS and UMHexagonS) and when using the proposed approaches, for different 3D stereo sequences in 1080p format, from a value of 28 to 40 for QP. For clarity the RD graphs are split into two sub–figures. As can be seen from the figure, the PSNR versus bit rate obtained with the proposed encoder is slightly better than the results obtained for the *P frames scenario* when compared with the UMHexagonS algorithm, and slightly worse when compared with the FS algorithm.



(a) Part I (b) Part II

Figure 6.7: RD graphic results of the proposed encoder. I, P and B frames.

The RD analysis when using the *P and B frames scenario* is similar to the one obtained when analysing the *P frames scenario*. However, when compared with the FS algorithm the bit rate increments and PSNR losses are slightly greater; when compared with the UMHexagonS algorithm the bit rate decrements and PSNR gains are also slightly greater.

Power and energy results

Table 6.6 shows the average power, time and energy consumed when coding one *mini GOP* (18 frames, 9 frames per view) for the complete test computer when coding five 3D stereo 1080p sequences. The first main column shows these results for the H.264/AVC encoder using the FS algorithm, the second main column shows them for the H.264/AVC encoder using the UMHexagonS algorithm and the third main column shows them for the H.264/AVC encoder using the proposed algorithm. Additionally, Table 6.6 includes two columns showing by how many times the proposed algorithm consumes less energy than the reference algorithms (FS and UMHexagonS).

Table 6.6: Energy consumption for coding a GOP. I, P and B frames.

H.264/AVC JM 17.2 Stereo High profile – Search range = 32											
Sequence	Full search			UMHexagonS			Proposal				
	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Power (W)	Time (s)	Energy (kJ)	Ratio FS	Ratio UMHS
Beergarden	184.42	1,748.44	322.45	181.58	101.84	18.49	282.31	52.13	14.72	21.91	1.26
Cafe	185.98	1,248.06	232.11	182.54	94.21	17.20	288.99	51.32	14.83	15.65	1.16
Car park	186.18	1,508.51	280.85	182.77	90.83	16.60	293.46	51.22	15.03	18.68	1.10
Hall	185.33	1,318.38	244.34	182.78	89.88	16.43	294.30	50.83	14.96	16.33	1.10
Street	184.69	1,671.97	308.80	182.52	91.76	16.75	295.46	51.38	15.18	20.34	1.10
Average	185.32	1,499.07	277.71	182.44	93.70	17.09	290.90	51.38	14.94	18.58	1.14

On average, the energy consumption for the GPU–based encoder is 18.58 times better than for the H.264/AVC encoder using the FS algorithm, and is 1.14 times better than for the H.264/AVC encoder using the UMHexagonS algorithm. In comparison with the results obtained when using the *P frames scenario*, the energy savings are greater. This behaviour is due to the fact that the speed–ups obtained for the complete H.264/AVC encoder are bigger than the ones obtained when using the *P frames scenario* because the percentage of time spent by the ME module is greater. Note that there is a bit more difference in the average power consumption between the reference encoder (180-185 W) and the proposed GPU–based encoder (290 W), and the reason is that the GPU is in execution about 50% of the total encoding time.

Figure 6.8 shows the power consumption over time for the complete test computer when coding the Beergarden sequence for both the H.264/AVC encoder using the FS algorithm and for the H.264/AVC encoder using our proposed algorithm. Note that only an extract

of the complete graph is shown in order to analyse in detail the power consumption. The H.264/AVC encoder using the UMHexagonS algorithm is not included in the graph since its power consumption is almost the same as the one obtained for the H.264/AVC using the FS algorithm, but the execution time is shorter.

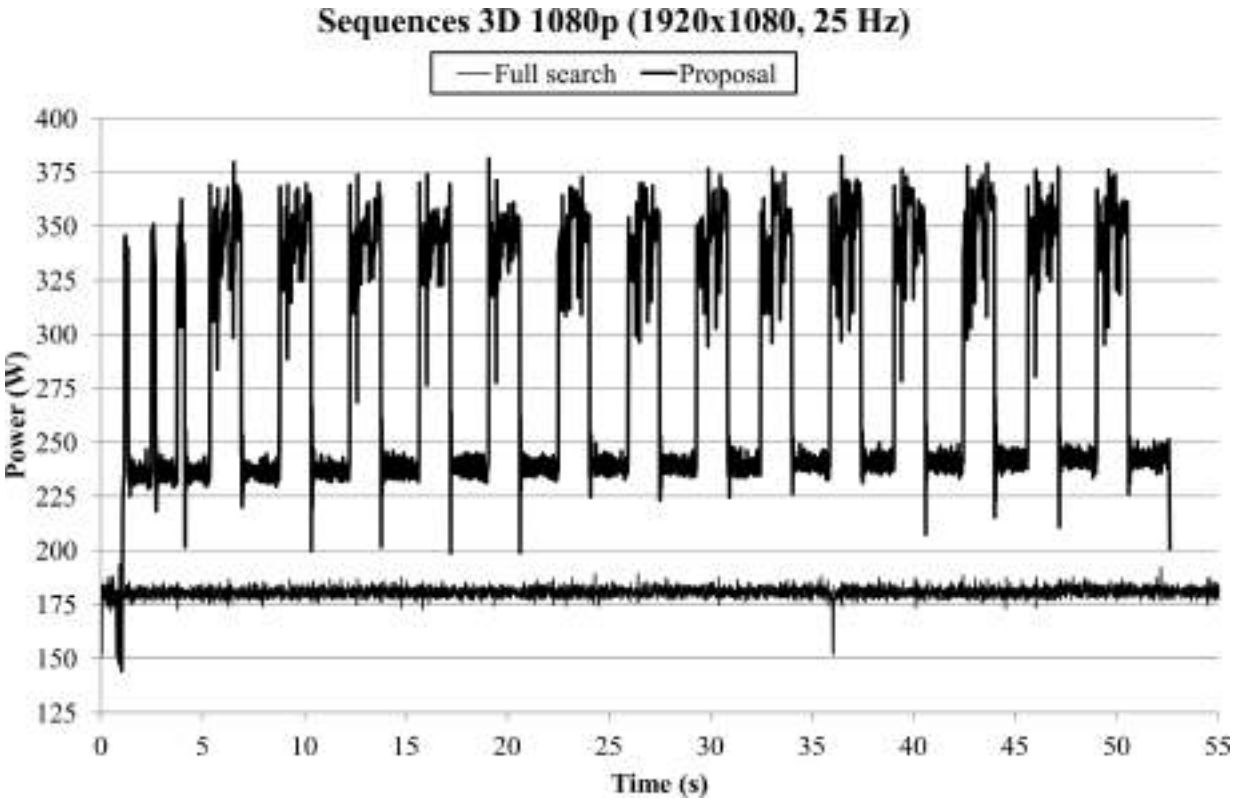


Figure 6.8: Power consumption graphic results of the proposed encoder. I, P and B frames of Beergarden sequence.

When the encoder process begins, both encoders consume the same power (180-185 W), but when the GPU starts working the power consumption of the proposed encoder increases, having a power consumption of around 325-375 W (see power consumption peaks in Figure 6.6). 17 power consumption peaks can be identified in Figure 6.8 since in the configured GOP pattern there are 1 I frame, and 17 P and B frames where the proposed algorithm is executed. The first 3 power consumption peaks correspond to the execution of P frames and are thinner than the others, which correspond to the execution of B frames. The execution time of the algorithm developed for P frames is considerably shorter than the one developed for B frames.

6.4. Conclusions

In order to reduce the high complexity of the H.264/AVC JM 17.2 reference encoder, this chapter proposed a GPU–based inter prediction algorithm developed for 3D video sequences. The proposed algorithm uses the algorithms presented in previous chapters, but a new MVP calculation is needed.

The aim of this chapter has been to obtain as much coding efficiency as possible when encoding 3D video sequences. For this purpose, two different ways of calculating the MVPs are presented, since the proposed algorithms eliminate two different kinds of redundancies (temporal and inter–view).

The conclusions regarding TR, RD performance and energy savings are similar to the ones presented in previous chapters.

Chapter 7

Conclusions and Future Work

THIS chapter concludes the thesis. Firstly, we will present the main conclusions which can be drawn from this thesis. Afterwards, we discuss which tasks could be developed as future work. Finally, we provide the list of the publications that have been derived from the thesis.

7.1. Conclusions

Several conclusions have been drawn in the course of this thesis. In the following, the most relevant conclusions are stated:

- First of all, different GPU architectures were analysed in detail, taking into account different GPU manufacturers such as NVIDIA and ATI/AMD. In this thesis, NVIDIA GPUs were chosen, since they can be programmed using CUDA C, which is a C–based high level programming language designed to maintain a low learning curve for programmers familiar with standard C. NVIDIA GPUs can be programmed using other high level programming languages such as OpenCL, but CUDA is the best way to exploit the GPU capabilities.

- Once the GPU architecture has been selected, a deep analysis of the H.264/AVC standard was made, paying special attention to the encoder side. The analysis focused on whether the inter prediction module is the one that best fits the GPU philosophy, obtaining a positive response. The inter prediction module has less data dependencies than other modules such as the intra prediction module, and does not use many divergent branch instructions. Moreover, the inter prediction module is the most time consuming module of an H.264/AVC encoder.

- In Chapter 4, an inter prediction algorithm developed for P frames is presented, implementing both the IME and FME algorithms on the GPU. The IME algorithm reuses the motion information calculated for the smallest MB sub–partition (4x4) to

obtain the motion information of the other higher MB partitions and sub–partitions. The algorithm uses data reusing techniques and is designed to exploit GPU capabilities. The algorithm is evaluated using the Baseline profile of the H.264/AVC JM 17.2 reference encoder, obtaining the following main results:

- The proposed algorithm obtains for the complete encoder a time reduction of over 84% when compared with the execution of the Full Search algorithm and of over 40% when compared with the execution of the UMHexagonS algorithm.
- The proposed algorithm achieves negligible RD drop penalties when compared with the Full Search algorithm, and surpasses the coding efficiency when compared with the UMHexagonS algorithm.
- The complete test computer using the proposed algorithm consumes less energy than when using the reference algorithms. The energy consumption of the GPU–based encoder ranges from 4 to 25 times better than when compared with the execution of the Full Search algorithm, and is around 1.4 when compared with the execution of the UMHexagonS algorithm.
- Additionally, the proposal is tested against related proposals. In a first section, it is tested against proposals that do not use a GPU to accelerate the inter prediction module, outperforming their timing results and equalling their coding efficiency. On the other hand, in a second section, it is tested against one proposal that uses a GPU to accelerate the inter prediction module, equalling its best timing results and outperforming its coding efficiency.

- In Chapter 5, an inter prediction algorithm for B frames is presented. The algorithm is an extension of the one previously presented for P frames, using similar data reusing techniques. However, for bi–directional prediction it is not possible to use the motion information of the smallest MB sub–partition to obtain that of other higher MB partitions or sub–partitions, since the opposite blocks of each MB partition and sub–partition may be different. The algorithm is evaluated using the Main profile of the H.264/AVC JM 17.2 reference encoder, obtaining the following results:

  - The proposed algorithm obtains for the complete encoder a time reduction of over 96% when compared with the execution of the Full Search algorithm, and of over 55% when compared with the execution of the UMHexagonS algorithm.
  - The proposed algorithm achieves acceptable RD drop penalties when compared with the Full Search algorithm, and surpasses the coding efficiency of the UMHexagonS algorithm.
  - The complete test computer using the proposed algorithm consumes less energy than when using the reference algorithms. The energy consumption of the GPU–based encoder is around 30 times better when compared with the execution of the Full Search algorithm, and is around 1.7 times better when compared with the execution of the UMHexagonS algorithm.

  - Additionally, the proposal is tested against one related proposal. It is tested against one proposal that does not use a GPU to accelerate the inter prediction module, outperforming its timing results and equalling its coding efficiency.

- In Chapter 6, an inter prediction algorithm for 3D stereo video sequences is presented. The algorithm presented in this chapter uses the ones presented in previous chapters, but they are adapted to a 3D scenario. The algorithms were adapted to remove temporal and inter–view redundancies.

- The hardware of NVIDIA GPUs has been continuously evolving during the development of this thesis. As a consequence, the proposals have evolved to exploit their potential as much as possible. In an early stage of this thesis an old NVIDIA 8800GTX was used, whose compute capability was 1.1. Later, an NVIDIA GTX285 was used, whose compute capability was 1.3. Finally, an NVIDIA GTX480 is used to obtain the performance results included in this thesis. GTX480 is based on the Fermi architecture, which provides some improvements in comparison with previous non–Fermi GPUs.

7.2. Future Work

The work that we have presented in this thesis can be expanded in several ways. In the following, we present some research lines that could be followed in the future:

- *To port the proposals to other high level programming languages that execute across heterogeneous platforms.* The thesis was implemented using CUDA, but as has been mentioned above, CUDA is not the only way of programming GPUs. There are other high level programming languages such as OpenCL, which are platform independent and can be executed on ATI/AMD GPUs and even on the CPU. CUDA programs can only be executed on NVIDIA GPUs.

- *To upgrade the GPU used.* During the last few months of the development of this thesis, NVIDIA released a new GPU core architecture called *Kepler*. The Kepler architecture redesigns again the GPU hardware to provide better performance. NVIDIA GPUs are backward compatible (the proposals can be executed on this new core architecture), but an analysis of this new architecture is needed to obtain all its potential.

- *To use multiple GPUs.* The algorithms are implemented for execution on a single GPU, so extending the algorithms for using on multiple GPUs may be a good option to further increase the time reductions obtained without affecting the encoding efficiency.

- *To consider other H.264/AVC modules.* At the beginning of this thesis, the ME module was identified as the most suitable for execution on a GPU, since it is the

most time consuming module of an H.264/AVC encoder and fits well with the GPU philosophy. However, there are other modules that are less time consuming, but after applying the proposals described in this thesis have become the most time consuming modules. Examples of these modules are the intra prediction and the de–blocking filter modules.

- *To consider other search algorithms.* The starting point of this thesis is the Full Search algorithm. However, in the literature other search algorithms can be found. These algorithms include the UMHexagonS [Rahman and Badawy 05] and the EPZS [Tourapis 02] algorithms. These algorithms are known as fast search algorithms and do not have a regular search pattern, as Full Search has, offering new challenges for GPU computing.

- *To adapt the proposals to make them work with RD–Optimization, constant bit rate and weighted prediction.* These options can be activated in the encoder configuration file, but they have not been taken into account during the development of this thesis. The proposals have been designed to work in SAE mode. A deeper analysis of these mechanisms and some algorithm modifications can be made to improve the coding efficiency of the proposed algorithms.

- *To adapt the proposals to make them work for other standards.* The proposals of this thesis have been developed for the H.264/AVC standard, but at the moment of presenting this thesis a new standard is near to being finalized. This future standard will be known as H.265/HEVC, and will have an even more complex ME module than the one defined for the H.264/AVC standard.

7.3. Publications

The proposals described in this thesis have led to the publication of several journal articles and participation in a number of conferences. In the following, we list all these publications, providing a brief description of the main contributions.

7.3.1. Journals

1. **H.264/AVC inter prediction for heterogeneous computing systems**

   - Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Gerardo Fernández Escribano; José Manuel Claver Iborra; José Luis Sánchez García
   - Journal: The Journal of Supercomputing
   - Year: 2012 On–line
   - Impact factor: 0.578 (JRC 2011)
   - Ranking: 68/99 in Computer Science, Theory and Methods category

This paper [Rodríguez-Sánchez et al. 12a] presents the IME algorithm developed for P frames and aimed at non–Fermi GPUs. It is an extension of the one presented at the CMMSE 2011 conference, paying special attention to the power and energy consumption of the H.264/AVC encoder.

2. **Optimizing H.264/AVC interprediction on a GPU-based framework**

   - Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Gerardo Fernández Escribano; José Luis Sánchez García; José Manuel Claver Iborra; Pedro Díaz Sánchez
   - Journal: Concurrency and Computation: Practice and Experience
   - Year: 2012 On–line
   - Impact factor: 0.636 (JRC 2011)
   - Ranking: 64/99 in Computer Science, Theory and Methods category

   This paper [Rodríguez-Sánchez et al. 12e] describes the different optimizations carried out to obtain the final IME algorithm developed for P frames using Fermi GPUs. The final version described in this paper is the one presented in this thesis in Chapter 4.

3. **H.264/AVC inter prediction on accelerator-based multi-core systems**

   - Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Gerardo Fernández Escribano; José Luis Sánchez García; José Manuel Claver Iborra
   - Journal: Multimedia Tools and Applications
   - Year: 2012 On–line
   - Impact factor: 0.617 (JRC 2011)
   - Ranking: 65/99 in Computer Science, Theory and Methods category

   This paper [Rodríguez-Sánchez et al. 12d] presents the complete inter prediction algorithm developed for P frames focusing on the Fermi GPU, including both IME and FME. The results of this article are a sub–set of the ones presented in Chapter 4.

7.3.2. Journals under review

1. **3D High Definition video coding on a GPU–based heterogeneous system**

   - Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Jan De Cock; Gerardo Fernández Escribano; Bart Pieters; José Luis Sánchez García; José Manuel Claver Iborra; Rik Van de Walle

- Journal: Computers and Electrical Engineering (selected at ISPA2012 conference)
- Impact factor: 0.837 (JRC 2011)
- Ranking: 26/50 in Computer Science, Hardware and Architecture category

This paper was selected to extend the proposal presented at the ISPA 2012 conference for 3D video sequences. It includes a more accurate way of obtaining the encoding costs which uses different Hadamard transforms.

2. **Adapting Hierarchical Bidirectional Inter Prediction on a GPU—based Platform for 2D and 3D H.264 Video Coding**

   - Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Jan De Cock; Gerardo Fernández Escribano; Bart Pieters; José Luis Sánchez García; José Manuel Claver Iborra; Rik Van de Walle
   - Journal: EURASIP Journal on Advances in Signal Processing
   - Impact factor: 0.811 (JRC 2011)
   - Ranking: 151/245 in Engineering, Electrical and Electronic category
   - Status: minor revisions

   This paper shows the complete inter prediction algorithm developed for B frames focusing on Fermi GPUs. The results of this article are a sub–set of the ones presented in Chapter 5 and in Chapter 6.

7.3.3. Conferences

1. **Accelerating H.264 Inter Prediction in a GPU using CUDA**

   - Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Gerardo Fernández Escribano; José Manuel Claver Iborra; José Luis Sánchez García
   - Conference: IEEE International Conference on Consumer Electronics (ICCE 2010)
   - ISBN: 978-1-4244-4314-7
   - Las Vegas (EEUU), January 2010

   This paper [Rodríguez-Sánchez et al. 10] presents the initial IME algorithm approach developed for P frames. The proposed algorithm does not deal with the MVP calculation and was developed for a non–Fermi GPU (GTX285).

2. **H.264/AVC Full-pixel Motion Estimation for GPU Platforms**

   - Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Gerardo Fernández Escribano; José Manuel Claver Iborra; José Luis Sánchez García

- Conference: The 11th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE 2011)
- ISBN: 978-84-614-6167-7
- Benidorm, Alicante (Spain), June 2011

This paper [Rodríguez-Sánchez et al. 11a] describes the final IME algorithm developed for non–Fermi GPUs. The proposed algorithm deals with the MVP calculation and has several optimizations in comparison with the one presented at the ICCE 2010 conference.

3. **Reducing Complexity in H.264/AVC Motion Estimation by using a GPU**

   - Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Gerardo Fernández Escribano; José Manuel Claver Iborra; José Luis Sánchez García
   - Conference: IEEE 13th International Workshop on Multimedia Signal Processing (MMSP 2011)
   - ISBN: 978-1-4577-1433-7
   - Hangzhou (China), October 2011

   This paper [Rodríguez-Sánchez et al. 11b] describes the initial IME approach developed for Fermi GPUs. The proposed algorithm deals with the MVP calculation and includes an evaluation using multiple reference frames.

4. **A Fast GPU-Based Motion Estimation Algorithm for H.264/AVC**

   - Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Gerardo Fernández Escribano; José Luis Sánchez García; José Manuel Claver Iborra
   - Conference: The 18th International Conference in Multimedia Modeling (MMM 2012). Lecture Notes in Computer Science vol. 7131
   - ISBN: 978-3-642-25958-6
   - Klagenfurt (Austria), January 2012

   This paper [Rodríguez-Sánchez et al. 12b] presents a complete GPU–based inter prediction algorithm developed for P frames, including both IME and FME. The algorithm is developed for Fermi GPUs and includes some optimizations in comparison with the IME presented at the MMSP 2011 conference. This proposal is described in Chapter 4

5. **A Fast GPU-Based Motion Estimation Algorithm for HD 3D Video Coding**

   - Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Gerardo Fernández Escribano; José Luis Sánchez García; José Manuel Claver Iborra

- Conference: The 10th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2012)
- ISBN: 978-0-7695-4701-5
- Leganés, Madrid (Spain), July 2012

This paper [Rodríguez-Sánchez et al. 12c] shows the inter prediction algorithm developed for 3D video sequences described in Chapter 6. The paper explores three different solutions to eliminate the temporal and inter–view dependencies until the final solution is achieved.

7.3.4. Other publications

1. **A GPU-Based DVC to H.264/AVC Transcoder**

   - Alberto Corrales García; Rafael Rodríguez Sánchez; José Luis Martínez Martínez; Gerardo Fernández Escribano; José Manuel Claver Iborra; José Luis Sánchez García
   - Conference: International Conference on Hybrid Artificial Intelligence Systems (HAIS 2010). Lecture Notes in Computer Science vol. 6077
   - ISBN: 978-3-642-13802-7
   - San Sebastian (Spain), June 2010

   This paper [Corrales-García et al. 10] presents a GPU-based DVC to H.264/AVC transcoder. The transcoding process is accelerated in both parts of the transcoder. The DVC part is accelerated by using a multi–core CPU, providing some information to the H.264/AVC encoder (MVPs). Then, by using the MVPs previously obtained in the DVC part of the transcoder, the IME algorithm presented at the ICCE 2010 conference is carried out.

Appendix A

Additional evaluation

In this chapter, an extra evaluation of our proposals is presented. First of all, the encoding conditions and the metrics used to evaluate the proposals are described. Then, the evaluation is carried out. The aim of this appendix is to evaluate the proposals using the common test conditions defined for the future H.265/HEVC standard. These common test conditions include different tools, such as RD–Optimization and Weighted prediction, which have not been taken into consideration when developing the algorithms presented in this thesis.

A.1. Encoding conditions

In order to further evaluate the proposals presented in this thesis an extra scenario is configured for testing. The H.264/AVC JM 17.2 encoder is configured to simulate, as far as possible, the encoding conditions of the future H.265/HEVC standard. For this purpose, the H.264/AVC JM encoder is configured using the parameters specified in [JCT-VC 12] and an HM–like configuration file is used as the base configuration. The base configuration is the one called *Low–delay B – High efficiency* in [JCT-VC 12], and is provided jointly with the latest H.264/AVC JM encoder releases. No modifications are made to the base configuration except that the search algorithms for the comparisons are the FS and UMHexagonS algorithms. In what follows a brief description of the configuration is provided:

- High profile is used
- The number of reference frames is set to 4 in both directions (forward and backward) for B frames.
- RD–Optimization is activated.
- Weighted prediction is activated.

- The GOP pattern is one I frame followed by 31 B frames.
- The tests are carried out with popular sequences in *Quarter Wide Video Graphics Array* (QWVGA) format (416x240 pixels), in *Wide Video Graphics Array* (WVGA) format (832x480 pixels) and in 720p format (HD, 1280x720 pixels). The sequences used are the ones specified in [JCT-VC 12], which are grouped into different Classes. QWVGA video sequences belong to class D, WVGA video sequences belong to class C and 720p video sequences belong to class E. Note that all classes defined in the document are not evaluated, since the evaluation made in this appendix ranges from low to high resolutions, providing an appropriate overview of the proposed algorithms.
- The search range is set to 32, which means 4096 positions inside the search area of each MB partition.

In order to make a proper comparison, an unmodified H.264/AVC JM 17.2 reference encoder is run on the same machine as the H.264/AVC JM using the proposed algorithms, with the same encoding configuration and with no calls to the GPU. The development environment, including the GPU, is the same as that used in the evaluation of Chapter 4.

A.2. Metrics

Different metrics have been used to evaluate the proposals. These metrics are the TR and speed–ups, which have been previously defined in Section 4.3.2. Moreover, in order to analyse the encoding efficiency of the proposed algorithms, [JCT-VC 12] recommends two RD cost metrics. The document is delivered jointly with an evaluation sheet on which two RD metrics are calculated. This evaluation shows the one labelled as *piecewise cubic* since the results for both metrics are quite similar.

A.3. Results

This section presents the results obtained when coding different video sequences, divided into two main categories: timing results and RD results.

Timing results

From Table A.1 to Table A.3 the timing results of the proposed algorithms for coding the analysed video sequences are presented. The results are mainly divided depending on the reference algorithm used for testing (FS and UMHexagonS); and are further divided showing the timing results for the complete H.264/AVC encoder and focusing exclusively on the proposed algorithm.

Table A.1: Timing results of the proposed encoder for class C.

H.264/AVC JM 17.2 High profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
BasketballDrill	98.94	94.57	91.34	11.55	80.08	5.02	32.09	1.47
BQMall	98.82	84.55	90.40	10.42	82.18	5.61	35.09	1.54
PartyScene	99.12	113.65	92.11	12.67	81.78	5.49	32.26	1.48
RaceHorses	99.21	126.91	93.27	14.85	86.11	7.20	41.33	1.70
Average	99.02	102.36	91.78	12.16	82.54	5.73	35.19	1.54

Table A.2: Timing results of the proposed encoder for class D.

H.264/AVC JM 17.2 High profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
BasketballPass	99.05	104.76	92.33	13.04	83.54	6.08	37.91	1.61
BlowingBubbles	98.99	99.23	91.36	11.58	79.82	4.96	30.35	1.44
BQSquare	98.90	90.52	90.45	10.47	78.73	4.70	29.01	1.41
RaceHorses	99.17	120.67	93.09	14.47	85.42	6.86	40.59	1.68
Average	99.03	102.68	91.81	12.21	81.88	5.52	34.46	1.53

Table A.3: Timing results of the proposed encoder for class E.

H.264/AVC JM 17.2 High profile – Search range = 32								
	Full search				UMHexagonS			
	ME module		Complete encoder		ME module		Complete encoder	
Sequence	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up	TR (%)	speed–up
FourPeople	97.42	38.69	81.16	5.31	63.94	2.77	17.94	1.22
Johnny	97.23	36.12	80.45	5.12	62.57	2.67	17.59	1.21
KristenAndSara	97.24	36.26	80.38	5.10	63.84	2.77	18.18	1.22
Average	97.30	36.99	80.66	5.17	63.45	2.74	17.91	1.22

For classes C and D, when compared with the FS algorithm, the speed–ups of the algorithm range from 84x to 126x, which means that the speed–ups for the complete H.264/AVC encoder range from 10x to 14x. However, for class E they are smaller because these sequences have low motion and the reference algorithms spend less time to encode

them. On the other hand, for classes C and D, when compared with the UMHexagonS algorithm, the speed–ups of the algorithm range from 4.7x to 7.2x, which means that the speed–ups for the complete H.264/AVC encoder range from 1.4x to 1.7x. The behaviour for class E is similar to the one shown when compared with the FS algorithm, obtaining smaller speed–ups.

RD results

From Table A.4 to Table A.6 the RD results of the proposed algorithms for coding the analysed video sequences are presented. Positive values mean that a higher number of bits are required to encode each color component for a certain PSNR value, and negative values mean that a smaller number of bits are required to encode each color component for a certain PSNR value.

Table A.4: RD results of the proposed encoder for class C.

H.264/AVC JM 17.2 High profile – Search range = 32						
	Full search			UMHexagonS		
Sequence	Y (%)	U (%)	V (%)	Y (%)	U (%)	V (%)
BasketballDrill	3.9	4.2	4.1	-2.7	-3.2	-3.5
BQMall	5.8	4.9	5.5	-2.4	-2.8	-2.9
PartyScene	5.0	3.9	3.9	1.6	0.7	0.3
RaceHorses	6.9	7.2	7.3	-3.0	-3.6	-3.2
Average	5.4	5.0	5.2	-1.6	-2.2	-2.3

Table A.5: RD results of the proposed encoder for class D.

H.264/AVC JM 17.2 High profile – Search range = 32						
	Full search			UMHexagonS		
Sequence	Y (%)	U (%)	V (%)	Y (%)	U (%)	V (%)
BasketballPass	4.9	4.2	4.8	0.1	-1.0	-0.6
BlowingBubbles	5.9	3.5	4.2	3.1	1.6	1.7
BQSquare	4.1	2.3	3.5	1.4	1.3	1.4
RaceHorses	6.9	6.0	5.9	0.4	-0.8	-0.5
Average	5.4	4.0	4.6	1.3	0.3	0.5

Table A.6: RD results of the proposed encoder for class E.

H.264/AVC JM 17.2 High profile – Search range = 32						
	Full search			UMHexagonS		
Sequence	Y (%)	U (%)	V (%)	Y (%)	U (%)	V (%)
FourPeople	2.4	1.7	2.1	0.3	-0.1	0.1
Johnny	6.3	3.3	2.6	0.7	1.6	1.4
KristenAndSara	4.5	5.8	5.3	-0.5	0.3	0.0
Average	4.4	3.6	3.3	0.2	0.6	0.5

In general, the results show that the proposed algorithms have a lower encoding efficiency than the FS algorithm, reporting average values ranging from 3.3% to 5.4%; the results also show that for some video sequences the proposed algorithms surpass the encoding efficiency obtained by the UMHexagonS algorithm, while not for others, reporting average values ranging from -2.3% to 1.3%.

RD degradation, when it occurs, is negligible if the computational savings are taken into account. Note that in the configuration file some mechanisms that are not taken into account when developing the proposed algorithms are activated, affecting RD performance. These mechanism are RD-optimization and weighted prediction, which are left as future work.

Bibliography

[Ambric 06] Ambric. Ambric First TeraOPS-Class Chip Using GALS Architecture, October 2006.

[Anzt et al. 11] H. Anzt, V. Heuveline, J. Aliaga, M. Castillo, J. Fernandez, R. Mayo, E. Quintana-Orti: Analysis and optimization of power consumption in the iterative solution of sparse linear systems on multi-core and many-core platforms. In *International Green Computing Conference and Workshops (IGCC '11)*, pp. 1 –6, July 2011.

[Atitallah et al. 12] A.B. Atitallah, S. Arous, H. Loukil, N. Masmoudi: Hardware implementation and validation of the fast variable block size motion estimation architecture for H.264/AVC. *AEU - International Journal of Electronics and Communications*, Vol. 66, No. 8, pp. 701–710, 2012.

[AVS-Group 05] AVS-Group. Information Technology - Advanced Coding of Audio and Video – Part 2: Video; advanced Audio and Video Standard (AVS1-P2), 2005.

[Bjontegaard 01] G. Bjontegaard. Calculation of Average PSNR Differences between RD-Curves. 13th VCEG-M33 Meeting, 2001.

[Blu 05] Blu–Ray Disc Association, 2005. http://www.blu-raydisc.com/en/index.aspx.

[Buck et al. 04] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, P. Hanrahan: Brook for GPUs: stream computing on graphics hardware. *ACM Transactions on Graphics*, Vol. 23, No. 3, pp. 777–786, August 2004.

[Bystrom et al. 08] M. Bystrom, I. Richardson, Y. Zhao: Efficient mode selection for H.264 complexity reduction in a Bayesian framework. *Signal Processing: Image Communication*, Vol. 23, No. 2, pp. 71–86, February 2008.

[Cai et al. 09] C. Cai, H. Zeng, S.K. Mitra: Fast motion estimation for H.264. *Signal Processing: Image Communication*, Vol. 24, No. 8, pp. 630–636, 2009.

[Chang et al. 04] C.Y. Chang, C.H. Pan, H. Chen: Fast Mode Decision for P-frames in H.264. In *Picture Coding Symposium (PCS '04)*, December 2004.

[Chen and Hang 08] W.N. Chen, H.M. Hang: H.264/AVC motion estimation implmentation on Compute Unified Device Architecture (CUDA). In *IEEE International Conference on Multimedia and Expo (ICME '08)*, pp. 697–700, April 2008.

[Chen et al. 91] L.G. Chen, W.T. Chen, Y.S. Jehng, T.D. Chiuch: An efficient parallel motion estimation algorithm for digital image processing. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 1, No. 4, pp. 378 –385, December 1991.

[Chen et al. 04] Y.K. Chen, X. Tian, S. Ge, M. Girkar: Towards efficient multi-level threading of H.264 encoder on Intel hyper-threading architectures. In *Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, 63, April 2004.

[Cheung et al. 10] N.M. Cheung, X. Fan, O. Au, M.C. Kung: Video Coding on Multicore Graphics Processors. *IEEE Signal Processing Magazine*, Vol. 27, No. 2, pp. 79 –89, March 2010.

[Corrales-García et al. 10] A. Corrales-García, R. Rodríguez-Sánchez, J.L. Martínez, G. Fernández-Escribano, J.M. Claver, J.L. Sánchez: A GPU-Based DVC to H.264/AVC transcoder. In *Proceedings of the 5th international conference on Hybrid Artificial Intelligence Systems - Volume Part II (HAIS '10)*, pp. 233–240, Berlin, Heidelberg, 2010. Springer-Verlag.

[Deng et al. 10] Z.p. Deng, K. Jia, Y.L. Chan, C.H. Fu, W.C. Siu: Fast motion and disparity estimation for multiview video coding. *Frontiers of Computer Science in China*, Vol. 4, pp. 571–579, 2010.

[Deng et al. 12] Z.P. Deng, Y.L. Chan, K.B. Jia, C.H. Fu, W.C. Siu: Iterative search strategy with selective bi-directional prediction for low complexity multiview video coding. *Journal of Visual Communication and Image Representation*, Vol. 23, No. 3, pp. 522 – 534, 2012.

[Ding et al. 08] L.F. Ding, P.K. Tsung, S.Y. Chien, W.Y. Chen, L.G. Chen: Content–Aware Prediction Algorithm With Inter-View Mode Decision for Multiview Video Coding. *IEEE Transactions on Multimedia*, Vol. 10, No. 8, pp. 1553 –1564, December 2008.

[Feng and Manocha 07] W. Feng, D. Manocha: High–performance computing using accelerators. *Parallel Computing*, Vol. 33, No. 10–11, pp. 645–647, 2007.

[Feng et al. 05] X. Feng, R. Ge, K. Cameron: Power and Energy Profiling of Scientific Applications on Distributed Systems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS '05)*, pp. 34–43, April 2005.

[GPGPU 07] GPGPU. *GPGPU*, 2007. `http://gpgpu.org/`.

[Grecos and Yang 05] C. Grecos, M.Y. Yang: Fast inter mode prediction for P slices in the H264 video coding standard. *IEEE Transactions on Broadcasting*, Vol. 51, No. 2, pp. 256 – 263, June 2005.

[Han and Lee 08] D.H. Han, Y.L. Lee: Fast Mode Decision Using Global Disparity Vector for Multiview Video Coding. In *Second International Conference on Future Generation Communication and Networking Symposia (FGCNS '08)*, Vol. 3, pp. 209 –213, December 2008.

[Hartley and Zisserman 04] R.I. Hartley, A. Zisserman: *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition, 2004.

[Haskell et al. 96] B. Haskell, A. Puri, A. Netravali: *Digital video: an introduction to MPEG-2*. Springer, 1996.

[Hilmi et al. 12] B. Hilmi, K. Goswami, J.H. Lee, B.G. Kim: Fast inter-mode decision algorithm for H.264/AVC using macroblock correlation and motion complexity analysis. In *IEEE International Conference on Consumer Electronics (ICCE '12)*, pp. 90 –91, January 2012.

[Ho et al. 06] C.W. Ho, O. Au, S.H. Gary Chan, S.K. Yip, H.M. Wong: Motion Estimation for H.264/AVC using Programmable Graphics Hardware. In *IEEE International Conference on Multimedia and Expo (ICME '06)*, pp. 2049 –2052, July 2006.

[Hsia and Hung 10] S.C. Hsia, Y.C. Hung: Fast multi-frame motion estimation for H264/AVC system. *Signal Image and Video Processing*, Vol. 4, No. 2, pp. 167–175, June 2010.

[Huang et al. 09] S. Huang, S. Xiao, W. Feng: On the energy efficiency of graphics processing units for scientific computing. In *Proceedings of 23rd International Parallel and Distributed Processing Symposium (IPDPS '09)*, pp. 1–8, 2009.

[Huo et al. 09] J. Huo, Y. Chang, M. Li, Y. Ma: Scalable prediction structure for multiview video coding. In *IEEE International Symposium on Circuits and Systems (ISCAS '09)*, pp. 2593 –2596, May 2009.

[Husemann et al. 11] R. Husemann, V. Roesler, R. Kintschner, H. Frohlich, A. Susin: High performance H.264/AVC encoding motion prediction algorithm. In *18th IEEE International Conference on Image Processing (ICIP '11)*, pp. 957 –960, September 2011.

[ISO/IEC 99] ISO/IEC. ISO/IEC 14486-2 PDAM1:1999, Infomation Technology - Generic Coding of Audio-Visual Objects - Part 2: Visual, 1999.

[ISO/IEC 03] ISO/IEC. ISO/IEC 14496-10:2003, Information technology – Coding of audio–visual objects – Part 10: Advanced Video Coding, 2003.

[ITU–T 03] ITU–T. ITU–T Recomendación H.264, Advanced Video Coding for Generic Audiovisual Services, 2003.

[ITU–T and ISO/IEC 07] ITU–T, ISO/IEC. ITU–T and ISO/IEC 14496-10:2007, Advanced Video Coding for Generic Audiovisual Services (SVC extension), 2007.

[ITU–T and ISO/IEC 09] ITU–T, ISO/IEC. ITU–T and ISO/IEC 14496-10:2007, Advanced Video Coding for Generic Audiovisual Services (MVC extension), 2009.

[ITU–T and ISO/IEC 12] ITU–T, ISO/IEC. High efficiency video coding (HEVC) text specification draft 6, February 2012.

[JCT-VC 12] JCT-VC. Common test conditions and software reference configurations. Join collaborative Team on video Coding 9th meeting, Doc. JCTVC-I1100, 2012.

[JVT 11] JVT. Join collaborative Team on video coding, Reference software to committee draft, version 17.2, 2011.

[JVT Test Model Ad Hoc Group 03] JVT Test Model Ad Hoc Group. Evaluation Sheet for Motion Estimation. Draft version 4, 2003.

[Kalva et al. 11] H. Kalva, A. Colic, A. Garcia, B. Furht: Parallel programming for multimedia applications. *Multimedia Tools Applications*, Vol. 51, No. 2, pp. 801–818, 2011.

[Kelly and Kokaram 04] F. Kelly, A. Kokaram: Fast Image Interpolation for Motion Estimation using Graphics Hardware. In *Proceedings of IS&T/SPIE Electronic Imaging – Real–Time Imaging VIII*, Vol. 5297, pp. 184–194, January 2004.

[Kim et al. 04] Y.H. Kim, J.W. Yoo, S.W. Lee, J. Shin, J. Paik, H.K. Jung: Adaptive mode decision for H.264 encoder. *Electronics Letters*, Vol. 40, No. 19, pp. 1172–1173, September 2004.

[Koeda 07] Y. Koeda. Operating System of the VX/VPP300/VPP700 Series of Vector-Parallel Supercomputer Systems, April 2007.

[Krüger and Westermann 03] J. Krüger, R. Westermann: Linear algebra operators for GPU implementation of numerical algorithms. *ACM Transactions on Graphics*, Vol. 22, No. 3, pp. 908–916, July 2003.

[Kung et al. 08] M. Kung, O. Au, P. Wong, C.H. Liu: Block based parallel motion estimation using programmable graphics hardware. In *International Conference on Audio, Language and Image Processing (ICALIP '08)*, pp. 599–603, July 2008.

[Kuo and Chan 06] T.Y. Kuo, C.H. Chan: Fast Variable Block Size Motion Estimation for H.264 Using Likelihood and Correlation of Motion Field. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 16, No. 10, pp. 1185 –1195, October 2006.

[Kuo and Lu 06] T.Y. Kuo, H.J. Lu. Efficient H.264 Encoding Based on Skip Mode Early Termination. In *Advances in Image and Video Technology*, Vol. 4319 of *Lecture Notes in Computer Science*, pp. 761 – 770. Springer Berlin Heidelberg, 2006.

[Lai and Ortega 06] P.L. Lai, A. Ortega: Predictive fast motion/disparity search for multi-view video coding. In *SPIE Visual Communications and Image Processing (SPIE VCIP '06)*, 2006.

[Lee and Jeon 03] J. Lee, B. Jeon. Pruned Mode Decision based on Variable Block Sizes Motion Compensation for H.264. In *Interactive Multimedia on Next Generation Networks*, Vol. 2899 of *Lecture Notes in Computer Science*, pp. 410–418. Springer Berlin Heidelberg, 2003.

[Lee and Jeon 04] J. Lee, B. Jeon: Fast mode decision for H.264. In *IEEE International Conference on Multimedia and Expo (ICME '04)*, Vol. 1 – 3, pp. 1131–1134, June 2004.

[Lee et al. 07] C.Y. Lee, Y.C. Lin, C.L. Wu, C.H. Chang, Y.M. Tsao, S.Y. Chien: Multi-Pass and Frame Parallel Algorithms of Motion Estimation in H.264/AVC for Generic GPU. In *IEEE International Conference on Multimedia and Expo (ICME '07)*, pp. 1603–1606, July 2007.

[Li et al. 07] X. Li, D. Zhao, X. Ji, Q. Wang, W. Gao: A Fast Inter Frame Prediction Algorithm for Multi–View Video Coding. In *IEEE International Conference on Image Processing (ICIP '07)*, Vol. 3, pp. 417–420, October 2007.

[Lim et al. 03] K.P. Lim, S. Wu, D.J. Wu, S. Rahardja, X. Lin, F. Pan, Z.G. Li. Fast Inter Mode Selection, Document JVT-I020. Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG 9th Meeting, September 2003.

[Lin et al. 07] Y.C. Lin, T. Fink, E. Bellers: Fast Mode Decision for H.264 Based on Rate-Distortion Cost Estimation. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '07)*, Vol. 1, pp. 1137–1140, April 2007.

[Liu et al. 09] Z. Liu, L. Shen, Z. Zhang: An Efficient Intermode Decision Algorithm Based on Motion Homogeneity for H.264/AVC. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 19, No. 1, pp. 128 –132, January 2009.

[Liu et al. 12] X. Liu, L.T. Yang, K. Sohn: High-speed inter-view frame mode decision procedure for multi-view video coding. *Future Generation Computer Systems*, Vol. 28, No. 6, pp. 947 – 956, 2012.

[Lu and Hang 11] C.T. Lu, H.M. Hang: Multiview encoder parallelized fast search realization on NVIDIA CUDA. In *IEEE Visual Communications and Image Processing (VCIP '11)*, pp. 1–4, November 2011.

[Lu et al. 07] J. Lu, H. Cai, J.G. Lou, J. Li: An Epipolar Geometry-Based Fast Disparity Estimation Algorithm for Multiview Image and Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 17, No. 6, pp. 737 –750, June 2007.

[Ma et al. 09] X. Ma, M. Dong, L. Zhong, Z. Deng: Statistical Power Consumption Analysis and Modeling for GPU-based Computing. In *Workshop on Power Aware Computing and Systems (HotPower '09)*, 2009.

[Massanes et al. 11] F. Massanes, M. Cadennes, J. Brankov: Compute-unified device architecture implementation of a block-matching algorithm for multiple graphical processing unit cards. *Journal of electronic imaging*, Vol. 20, No. 3, 2011.

[Merkle et al. 06] P. Merkle, K. Muller, A. Smolic, T. Wiegand: Efficient Compression of Multi-View Video Exploiting Inter-View Dependencies Based on H.264/MPEG4-AVC. In *IEEE International Conference onMultimedia and Expo (ICME '06)*, pp. 1717 –1720, July 2006.

[Merkle et al. 07] P. Merkle, A. Smolic, K. Muller, T. Wiegand: Efficient Prediction Structures for Multiview Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 17, No. 11, pp. 1461–1473, Novermber 2007.

[Momcilovic and Sousa 09] S. Momcilovic, L. Sousa: Development and evaluation of scalable video motion estimators on GPU. In *IEEE Workshop on Signal Processing Systems (SiPS '09)*, pp. 291 –296, October 2009.

[Monteiro et al. 11] E. Monteiro, B. Vizzotto, C. Diniz, B. Zatt, S. Bampi: Applying CUDA Architecture to Accelerate Full Search Block Matching Algorithm for High Performance Motion Estimation in Video Encoding. In *23rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD '11)*, pp. 128 –135, October 2011.

[Monteyne 08] M. Monteyne. White Paper: RapidMind Multi-Core Development Platform, February 2008. `http://www.rapidmind.net/pdfs/WP_RapidMindPlatform.pdf`.

[Nagasaka et al. 10] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, S. Matsuoka: Statistical power modeling of GPU kernels using performance counters. In *International Green Computing Conference (IGC '10)*, pp. 115–122, August 2010.

[Nieto et al. 06] M. Nieto, L. Salgado, J. Cabrera: Fast Mode Decision on H.264/AVC Main Profile Encoding Based on PSNR Predictions. In *Proceedings of the International Conference on Image Processing (ICIP '03)*, pp. 49 –52, October 2006.

[NVidia 12] NVidia. *NVidia CUDA C programming guide 4.2*. NVidia, April 2012. `http://developer.nvidia.com/`.

[Papakipos 07] M. Papakipos. The PeakStream Platform: High-Productivity Software Development for Multi-Core Processors, April 2007.

[Pieters et al. 09] B. Pieters, C.F. Hollemeersch, P. Lambert, R. van de Walle: Motion estimation for H.264/AVC on multiple GPUs using NVIDIA CUDA. In *Society of Photo-Optical Instrumentation Engineers (SPIE '09) Conference Series*, Vol. 7443, 2009.

[Pieters et al. 12] B. Pieters, C. Hollemeersch, J. De Cock, P. Lambert, R. Van de Walle, P. Alface, C. Stevens: Multiview Video Coding Using Video Game Context Information. In *2012 IEEE International Conference on Multimedia and Expo Workshops (ICMEW '12)*, pp. 1 –6, July 2012.

[Rahman and Badawy 05] C. Rahman, W. Badawy: UMHexagonS algorithm based motion estimation architecture for H.264/AVC. In *Proceedings. of the Fifth International Workshop on System-on-Chip for Real-Time Applications, 2005*, pp. 207–210, July 2005.

[Ri and Ostermann 06a] S.H. Ri, J. Ostermann. Fast mode decision for H.264/AVC using mode prediction. In *Proceedings of the 8th International Conference on Advanced Concepts for Intelligent Vision Systems* , Vol. 4179 of *Lecture Notes in Computer Science*, pp. 354 – 363. Springer Berlin Heidelberg, 2006.

[Ri and Ostermann 06b] S.H. Ri, J. Ostermann: Fast Mode Decision for H.264/AVG using Mode and RD Cost Prediction. In *First International Conference on Communications and Electronics (ICCE '06)*, pp. 264 – 269, October. 2006.

[Richardson 04] I.E.G. Richardson: *H.264 and MPEG-4 Video Compression: Video Coding for Next-Generation Multimedia*. John Wiley and Sons, Ltd, 2004.

[Richardson 10] I.E.G. Richardson: *The H.264 Advanced Video Compression Standard, 2nd Edition*. John Wiley and Sons, Ltd, 2010.

[Rodríguez-Sánchez et al. 10] R. Rodríguez-Sánchez, J.L. Martínez, G. Fernández-Escribano, J.M. Claver, J.L. Sánchez: Accelerating H.264 inter prediction in a GPU by using CUDA. In *IEEE International Conference on Consumer Electronics (ICCE '10)*, pp. 463 –464, January 2010.

[Rodríguez-Sánchez et al. 11a] R. Rodríguez-Sánchez, J.L. Martínez, G. Fernández-Escribano, J.M. Claver, J.L. Sánchez: H.264/AVC Full-pixel Motion Estimation for GPU Platforms. In *The 11th International Conference Computational and mathematical methods in Science and Engineering (CMMSE '11)*, June 2011.

[Rodríguez-Sánchez et al. 11b] R. Rodríguez-Sánchez, J.L. Martínez, G. Fernández-Escribano, J.M. Claver, J.L. Sánchez: Reducing complexity in H.264/AVC motion estimation by using a GPU. In *IEEE 13th International Workshop on Multimedia Signal Processing (MMSP '11)*, pp. 1 –6, October 2011.

[Rodríguez-Sánchez et al. 12a] R. Rodríguez-Sánchez, J.L. Martínez, G. Fernández-Escribano, J.M. Claver, J.L. Sánchez: H. 264/AVC inter prediction for heterogeneous computing systems. *The Journal of Supercomputing*, pp. 1–10, 2012.

[Rodríguez-Sánchez et al. 12b] R. Rodríguez-Sánchez, J.L. Martínez, G. Fernández-Escribano, J.L. Sánchez, J.M. Claver. A Fast GPU-Based Motion Estimation Algorithm for H.264/AVC. In *Advances in Multimedia Modeling (MMM '12)*, Vol. 7131 of *Lecture Notes in Computer Science*, pp. 551–562. Springer Berlin Heidelberg, 2012.

[Rodríguez-Sánchez et al. 12c] R. Rodríguez-Sánchez, J.L. Martínez, G. Fernández-Escribano, J.L. Sánchez, J.M. Claver: A Fast GPU-Based Motion Estimation Algorithm for HD 3D Video Coding. In *IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA '12)*, pp. 166 –173, July 2012.

[Rodríguez-Sánchez et al. 12d] R. Rodríguez-Sánchez, J.L. Martínez, G. Fernández-Escribano, J.L. Sánchez, J.M. Claver: H.264/AVC inter prediction on accelerator-based multi-core systems. *Multimedia Tools and Applications*, pp. 1–21, 2012.

[Rodríguez-Sánchez et al. 12e] R. Rodríguez-Sánchez, J.L. Martínez, G. Fernández-Escribano, J.L. Sánchez, J.M. Claver, P. Díaz: Optimizing H.264/AVC interprediction on a GPU-based framework. *Concurrency and Computation: Practice and Experience*, Vol. 24, No. 14, pp. 1607–1624, 2012.

[Rofouei et al. 08] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, M. Sarrafzadeh: Energy-aware high performance computing with graphic processing units. In *Proceedings of the 2008 conference on Power aware computing and systems (HotPower'08)*, pp. 11–15, 2008.

[Ruiz and Michell 11] G. Ruiz, J. Michell: An efficient VLSI processor chip for variable block size integer motion estimation in H.264/AVC. *Signal Processing: Image Communication*, Vol. 26, No. 6, pp. 289–303, 2011.

[Ryoo et al. 08] S. Ryoo, C.I. Rodrigues, S.S. Baghsorkhi, S.S. Stone, D.B. Kirk, W.m.W. Hwu: Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming (PPoPP '08)*, pp. 73–82, 2008.

[Schwalb et al. 09] M. Schwalb, R. Ewerth, B. Freisleben: Fast Motion Estimation on Graphics Hardware for H.264 Video Encoding. *IEEE Transactions on Multimedia*, Vol. 11, No. 1, pp. 1–10, January 2009.

[Schwarz et al. 06] H. Schwarz, D. Marpe, T. Wiegand: Analysis of hierarchical B pictures and MCTF. In *IEEE International Conference on Multimedia and Expo (ICME '06)*, pp. 1929–1932. IEEE, 2006.

[Shen et al. 05] G. Shen, G.P. Gao, S. Li, H.Y. Shum, Y.Q. Zhang: Accelerate video decoding with generic GPU. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 15, No. 5, pp. 685 – 693, May 2005.

[Shen et al. 09] L. Shen, Z. Liu, S. Liu, Z. Zhang, P. An: Selective Disparity Estimation and Variable Size Motion Estimation Based on Motion Homogeneity for Multi–View Coding. *IEEE Transactions on Broadcasting*, Vol. 55, No. 4, pp. 761 –766, December 2009.

[Shen et al. 11] L. Shen, Z. Liu, P. An, R. Ma, Z. Zhang: Low-Complexity Mode Decision for MVC. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 21, No. 6, pp. 837 –843, June 2011.

[SPEC 06] SPEC. *CPU2006*, 2006. `http://www.spec.org/cpu2006/`.

[Sullivan and Bjontegaard 01] G. Sullivan, G. Bjontegaard. Recommended Simulation Common Conditions for H.26L Coding Efficiency Experiments on Low–Resolution Progressive–Scan Source Material. ITU-T VCEG, Doc. VCEG-N81, 2001.

[Sullivan et al. 04] G.J. Sullivan, P. Topiwala, A. Luthra: The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. In *SPIE conference on Applications of Digital Image Processing XXVII*, pp. 454–474, September 2004.

[Sullivan et al. 06] G. Sullivan, J.R. Ohm, A. Ortega, E. Delp, A. Vetro, M. Barni: dsp Forum - Future of Video Coding and Transmission. *IEEE Signal Processing Magazine*, Vol. 23, No. 6, pp. 76–82, November 2006.

[Thrust 11] Thrust. Thrust – Code at the speed of light, 2011. http://code.google.com/p/thrust/wiki/QuickStartGuide.

[Tilera 08] Tilera. TILEPro64 Processor, 2008. `http://www.tilera.com/products/processors/TILEPRO64`.

[TOP500 12] TOP500. The Top500 List: Twenty Years of Insight into HPC Performance, November 2012. http://www.top500.org/.

[Tourapis 02] A.M. Tourapis: Enhanced predictive zonal search for single and multiple frame motion estimation. In *SPIE Visual Communications and Image Processing (SPIE VCIP '02)*, Vol. 4671, pp. 1069–1079, 2002.

[Tseng et al. 05] H.C. Tseng, C.R. Chang, Y.L. Lin: A hardware accelerator for H.264/AVC motion compensation. In *IEEE Workshop on Signal Processing Systems Design and Implementation*, pp. 214–219, November 2005.

[Vetro et al. 11] A. Vetro, T. Wiegand, G. Sullivan: Overview of the stereo and multiview video coding extensions of the H. 264/MPEG-4 AVC standard. *Proceedings of the IEEE*, Vol. 99, No. 4, pp. 626–642, 2011.

[Wang et al. 07] X. Wang, J. Sun, Y. Liu, R. Li: Fast Mode Decision for H.264 Video Encoder Based on MB Motion Characteristic. In *IEEE International Conference on Multimedia and Expo (ICME '07)*, pp. 372 – 375, July 2007.

[Werda et al. 07] I. Werda, H. Chaouch, A. Samet, M. Ali, B. Ayed, N. Masmoudi: Optimal DSP-Based Motion Estimation Tools Implementation For H.264/AVC Baseline Encoder. *International Journal of Computer Science and Network Security,*, Vol. 7, No. 5, pp. 141–150, May 2007.

[Wiegand et al. 03] T. Wiegand, G. Sullivan, G. Bjontegaard, A. Luthra: Overview of the H.264/AVC video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, pp. 560 –576, July 2003.

[Wu et al. 05] D. Wu, F. Pan, K.P. Lim, S. Wu, Z.G. Li, X. Lin, S. Rahardja, C.C. Ko: Fast intermode decision in H.264/AVC video coding. *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 15, No. 7, pp. 953 – 958, july 2005.

[Yi et al. 05] X. Yi, J. Zhang, N. Ling, , W. Shang. Improved and simplified fast motion estimation for JM. Technical report, JVT-P021, JVT Meeting, Poznan, Poland, July 2005.

[Yin et al. 03] P. Yin, H.Y. Tourapis, A. Tourapis, J. Boyce: Fast mode decision and motion estimation for JVT/H.264. In *Proceedings of the International Conference on Image Processing (ICIP '03)*, Vol. 3, pp. 853 – 856, September 2003.

[Yu 04] A.C. Yu: Efficient block-size selection algorithm for inter-frame coding in H.264/MPEG-4 AVC. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '04)*, Vol. 3, pp. 169 – 172, May 2004.

[Zeng et al. 10] H. Zeng, K.K. Ma, C. Cai: Mode-correlation-based early termination mode decision for multi-view video coding. In *17th IEEE International Conference on Image Processing (ICIP '10)*, pp. 3405 –3408, September 2010.

[Zhan et al. 07] B. Zhan, B. Hou, R. Sotudeh: Fast mode decision for H.264 based on correlation between macroblocks. In *Norchip*, pp. 1 –4, November 2007.

[Zhan et al. 08] B. Zhan, B. Hou, R. Sotudeh: A fast intra/inter mode decision algorithm of H. 264/AVC for real-time applications. In *IEEE International Conference on Communications (ICC '08)*, pp. 510–514, 2008.

[Zhang et al. 11] Y. Zhang, S. Kwong, G. Jiang, H. Wang: Efficient Multi-Reference Frame Selection Algorithm for Hierarchical B Pictures in Multiview Video Coding. *IEEE Transactions on Broadcasting*, Vol. 57, No. 1, pp. 15 –23, March 2011.

[Zheng et al. 08] J. Zheng, W. Gao, D. Wu, D. Xie: A novel VLSI architecture of motion compensation for multiple standards. *IEEE Transactions on Consumer Electronics*, Vol. 54, No. 2, pp. 687 –694, May 2008.