2017

High Performance Multiview Video Coding

Caoyang Jiang
Michigan Technological University, cjiang@mtu.edu

HIGH PERFORMANCE MULTIVIEW VIDEO CODING

By

Caoyang Jiang

A DISSERTATION

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

In Electrical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2017

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Electrical Engineering.

Department of Electrical and Computer Engineering

Dissertation Advisor:	*Dr. Saeid Nooshabadi*
Committee Member:	*Dr. Timothy Havens*
Committee Member:	*Dr. Zhuo Feng*
Committee Member:	*Dr. Soner Onder*
Department Chair:	*Dr. Daniel R. Fuhrmann*

Dedication

I dedicate this to Wei He, my wife; Tao Jiang, my father; And Lingjiang Cao, my mother

Contents

List of Figures

List of Tables

Preface

This dissertation presents my research work in pursuing the PhD degree in Electrical Engineering at Michigan Technological University. This dissertation includes previously published articles in Chapter 2, Chapter 3 and Chapter 4. All the research works described herein were conducted under the supervision of my advisor Professor Saeid Nooshabadi.

Chapter 2 and Chapter 3 contain journal articles published in *IEEE Transactions on Circuits and Systems for Video Technology* and *IEEE Transcations on Parallel and Distributed Systems*, respectively. Chapter 4 contains a conference paper published in *2016 IEEE Data Compression Conference*. As the first author of the papers and manuscript, with the guidance from my adivser, I completed the proposal, implementation, and analysis of the algorithms and methodologies. The articles were completed collaboratively with the help from my advisor Dr. Saied Nooshabadi.

Acknowledgments

First and foremost, I would like to thank my advisor Professor Saeid Nooshabadi for his support and guidance while I was pursing my PhD degree at Michigan Tech.

I would like to thank all my committee members, Professor Timothy Havens, Professor Zhuo Feng and Professor Soner Onder for their time, invaluable advices and encouragements. I am also greatly indebted to the help and support from the department chair Dr.Fuhrmann.

I also want to thank my friends at Michigan Tech, Lengfei Han, Xueqian Zhao, Yonghe Guo, Solmaz Hajmohammadi, Shuo Wang for their help and supports. Xueqian Zhao offered me many useful advices for the preparation of my Phd qualification exam. Lengfei had been a very supporting friend of mine. His advices on writing PhD dissertation had been a great help.

Finally, and most importantly, I would like to thank my family. Without the love and support from my wife, it is impossible for me to become who I am today. I coudn't compensate her enough for the years of sarifices she made to enable the success of my career. My parents has always been a strong source of encouragement and support when I needed.

Abstract

Following the standardization of the latest video coding standard *High Efficiency Video Coding* in 2013, in 2014, multiview extension of HEVC (MV-HEVC) was published and brought significantly better compression performance of around 50% for multiview and 3D videos compared to multiple independent single-view HEVC coding. However, the extremely high computational complexity of MV-HEVC demands significant optimization of the encoder. To tackle this problem, this work investigates the possibilities of using modern parallel computing platforms and tools such as single-instruction-multiple-data (SIMD) instructions, multi-core CPU, massively parallel GPU, and computer cluster to significantly enhance the MVC encoder performance. The aforementioned computing tools have very different computing characteristics and misuse of the tools may result in poor performance improvement and sometimes even reduction. To achieve the best possible encoding performance from modern computing tools, different levels of parallelism inside a typical MVC encoder are identified and analyzed. Novel optimization techniques at various levels of abstraction are proposed, non-aggregation massively parallel motion estimation (ME) and disparity estimation (DE) in prediction unit (PU), fractional and bi-directional ME/DE acceleration through SIMD, quantization parameter (QP)-based early termination for coding tree unit (CTU), optimized resource-scheduled wave-front parallel processing for CTU, and workload balanced, cluster-based multiple-view parallel are proposed. The result shows proposed parallel optimization techniques, with insignificant loss to coding efficiency, significantly improves the execution time performance. This , in turn, proves modern parallel computing platforms, with appropriate platform-specific algorithm design, are valuable tools for improving the performance of computationally intensive applications.

Chapter 1

Introduction

Standardized in 2014, the latest international video coding standard, High Efficiency Video Coding (HEVC/H.265 [1], [2]) takes over the place of previous standard, Advanced Video Coding (AVC/H.264 [3]) as the most efficient video coding standard. The design goal of HEVC/H.265 is to provide 50% better compression performance than its predecessor AVC/H.264 to satisfy ever growing demand for ultra high definition videos such as 4k and 8K resolution. Limited by the viewing angle of single camera capturing system, to further enhance the viewing experience, a multiview system where multiple cameras are deployed to capture the same scene in a synchronized fashion draws high interests [4] in the recent years. Common application of multiview coding includes free view TV, immersive teleconference, and virtual reality. The computational complexity and coding efficiency for video encoders are closely related and achieving better compression performance requires significantly more computations. By taking multiple views into encoding consideration, the complexity increases exponentially.

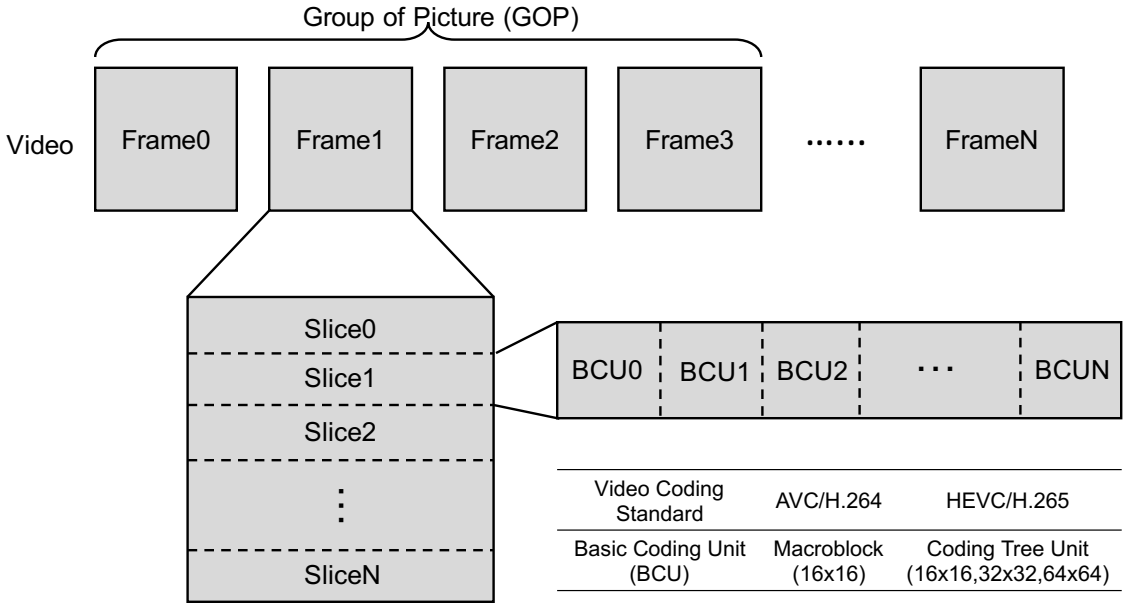


Figure 1.1: Video structure: video, group of picture, slice, and standard-specific basic coding units.

The challenge is thus in encoder optimization to trade minimal amount of computational cost for the most coding efficiency. This is specially critical for real-time application where delivering time has to meet strict deadlines. Conventional methodologies attempt to identify and skip non-effective computations with minimal cost to coding efficiency. The emerging parallel computing tools and platforms, such as multimedia instructions, multi-core central processing unit (CPU) and massively parallel architecture (MPA) offer new opportunities and guidances for video encoder optimization, where the calculations are accelerated by carrying out computations in parallel, leading to significant time reduction. However, designing efficient architecture-specific algorithms require careful analysis of modern video encoder architectures and identification of parallelizable procedures in the encoding process.

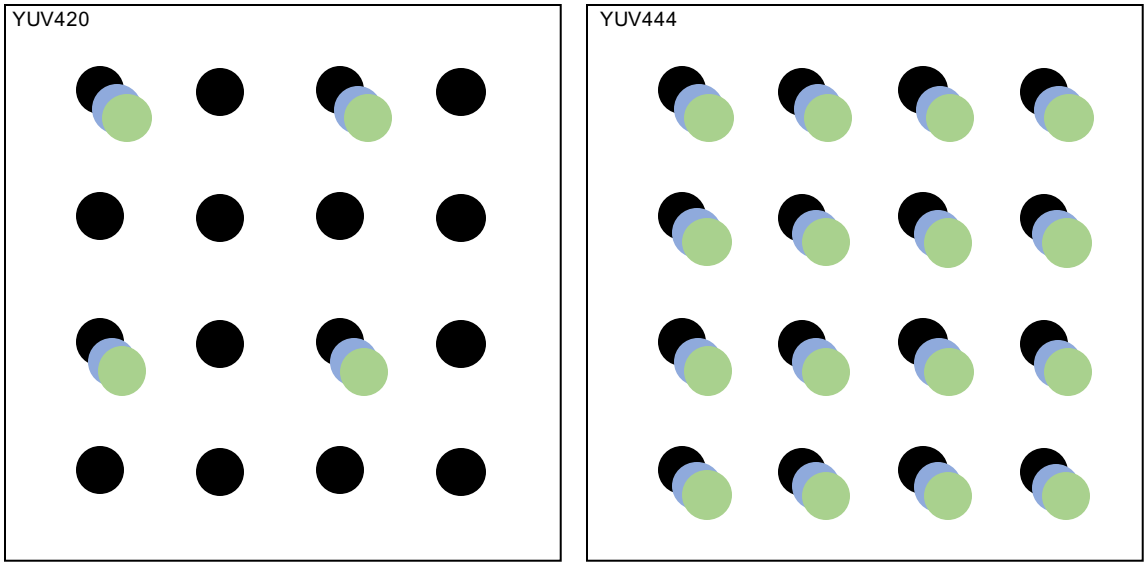


Figure 1.2: YUV420 and YUV444 commonly used for storing raw video data.

1.1 Video Coding

1.1.1 Video Structure

A video sequence is produced by camera that continuously projecting the real world 3-D scene onto a 2-D imaging sensor and saving the resulting pixel values digitally for post-processing [5]. Many color space are devised for displaying and storing raw video sequences using multi-channel scheme such as Luminance plus two chrominance (YUV) and Red-Green-Blue (RGB). Any color can be produced by varying the magnitude of the channels. The most frequently used color space scheme for storing raw video data is YUV420. This representation takes the advantage of the less sensibility to the color variation in the human visual system and subsamples the two chrominance components, reducing the raw video sequence size by half. YUV444, on the other hand, keeps full captured color information in Y, U and V channels. Fig. 1.2

illustrates the structure of YUV420 and YUV444 color space. In this figure, black circles, blue circles, and green circles refer to Y, U, and V channel data, respectively. The YUV444 and YUV420 are the only supported source video formats in both AVC/H.264 [3] and HEVC/H.265 standard [6].

1.1.1.1 Group of Picture

Knowing a video source consists of frames, it is possible to group the frames periodically to form so called group of pictures (GOP). A GOP is considered independent from other GOPs and can be encoded and decoded without knowing the existences of other GOPs. As a result, only frames within a GOP may dependent on each other and the type of dependence is further specified.

1.1.1.2 Slice

A frame can be further divided into multiple slices. Each slice consists of a rectangular region of pixels extending full frame width. Error-resiliency is improved with frame slicing where the corruption of a slice do not lead to the corruption of the entire frame.

1.1.1.3 Coding Units

A slice consists of integer number of square blocks of pixels, forming the basic coding unit for encoding and decoding. The basic coding unit sizes are 16×16 (referred to as Macroblock) for AVC/H.264 and 64×64 (referred to as Coding Tree Unit) for

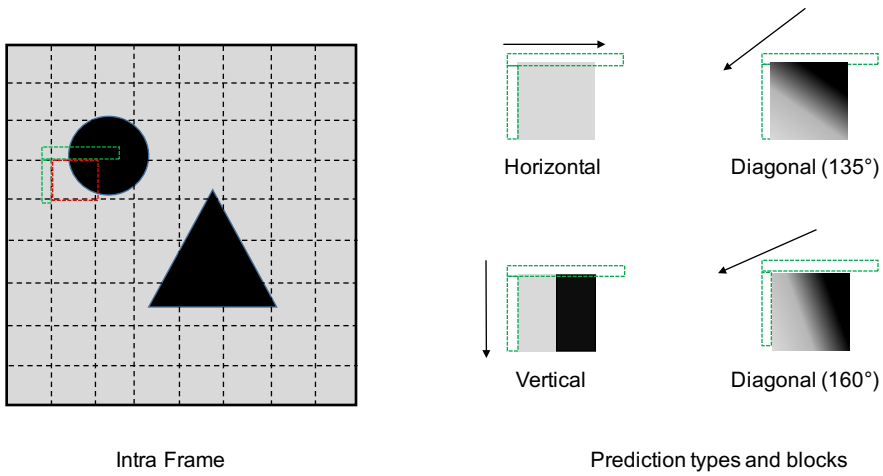


Figure 1.3: Intra-frame prediction uses neighboring pixels from top and left to make prediction.

HEVC/H.265. The compression of a video frame is the result of compressing all basic coding units within the frame. Video coding takes the advantage of *intra-frame* and *inter-frame* redundancies within basic coding units to significantly reduce its size.

1.1.2 Intra-frame Prediction

1.1.2.1 Intra-frame Redundancy

For natural scene, there exists many homogenous regions with little to no color variation such as a white wall in the background. It is thus possible to use few pixels to predict the values for majority of the other pixels within the homogenous region. Coding and storing only the pixels for prediction and prediction differences (errors) achieves the goal of compression. The use of intra-frame prediction for encoding is called, *intra-frame coding* [2] [3].

1.1.2.2 Prediction Modes

To obtain good compression with intra-frame prediction requires reducing the number of prediction pixels and increasing prediction accuracy (resulting in smaller prediction error).

Given coding units are rectangular, it is natural to use border pixels as the source for prediction. In both AVC/H.264 and HEVC/H.265 standard, pixels immediately to the left and top of the current basic unit are used for intra-prediction. An important reason for selecting these pixels is that they all belong to previously encoded basic unit and do not change for the remaining encoding process. Without knowing the pattern of the homogenous region, it is necessary to make trial predictions using various angles (modes) for prediction. The mode associated with the best match, in terms of smallest prediction error, is chosen and coded together with the prediction error. 16 prediction angles (or modes) are specified for AVC/H.264 [3] and 33 for HEVC/H.265 [6]. Encoder has the freedom to try all or a fraction of the modes. Fig. 1.3 shows four intra-prediction modes. The horizontal and vertical green boxes refer to the column pixels immediately to the left and top of the candidate basic unit. The main difference between the modes is the angle that defines the pattern of prediction blocks. Flexibility in intra-mode evaluations and various other mode evaluations within the encoder reflects an important design aspect in the modern video standard[2], [3]. The flexility makes room for both simple encoder designs (possibly with lower compression) that evaluate less modes and sophisticated encoder designs (possibly with higher compression) that evaluate more modes.

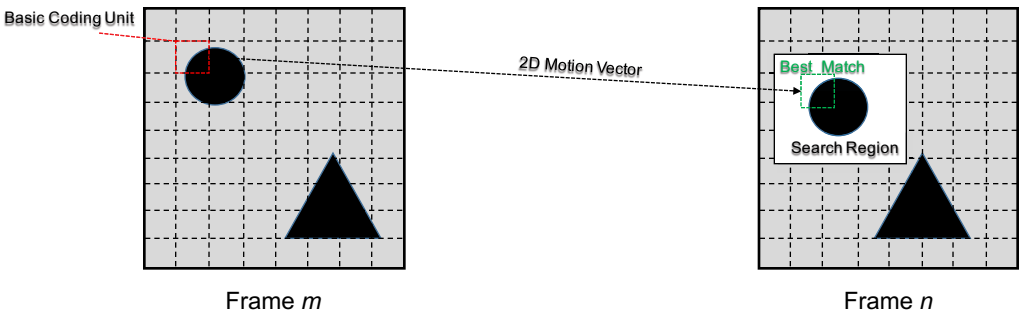


Figure 1.4: Motion estimation for a block of pixels.

1.1.3 Inter-frame Prediction

The speed of frame capturing is referred to as frame rate and commonly measured in unit of frame per second (FPS). Human visual system can't detect the subtle change between frames once the frame rate exceeds about 30 FPS. Between the transition of two consecutive frames, a significant amount of visual content undergoes little to no change due to the relatively slow motion of natural objects. For more static applications such as video conferencing and news reporting, the change between consecutive frames is close to none. The similarity between frames (and by extension, coding units within frame) is the second type of redundancy in the video and is referred to as *inter-frame redundancy*. Due to the motion of objects in the scene, it is possible to form a prediction for a coding unit in another frame represented by a displacement in the 2-D image space (motion vector) [2], [3]. By coding and storing only the motion vector, reference frame ID, and prediction error achives the goal of compression. The use of inter-frame prediction for encoding is called, *inter-frame coding*.

1.1.3.1 Inter-frame Redundancy

Similar to mode evaluation in intra-frame prediction, the compression in the inter-frame coding is achieved through a trial-and-error process called, *motion estimation*, where the coding unit is compared against blocks of the same size at various locations in a predefined search region in other frame(s). Fig. 1.4 illustrates the concept of the search process. The red box and green box refer to the coding unit and its best match in the search region, respectively. The white area in *Frame n* is the search region. Encoder attempts to find the most accurate match for the candidate block inside this region. Larger search window size and more reference frames increase the chance of finding better match (hence better compression) at the cost of higher computation. Usually, a coding unit may contain motion content in various directions. In this case, using a single motion vector for the entire block may lead to high prediction error and thus leads to inferior compression performance. It is, thus, natural to sub-divide a coding unit into smaller blocks and assign motion vector to each of the sub-divided blocks to achieve finer prediction. The sub-divided blocks are called sub-partitions in AVC/H.264 [3] and coding units in HEVC/H.265 [2]. Note that sub-divided blocks from coding units can be further recursively divided to achieve even finer prediction. Each sub-divided coding unit undergoes an independent motion estimation, adding another layer of complexity to the motion estimation process for HEVC/H.265 standard.

1.1.3.2 Integer-pixel Motion Estimation

The motion estimation where the displacement is an integer number of pixels is called integer-pixel motion estimation. This process contributes to the majority of the motion estimation time due to the large number of search locations.

1.1.3.3 Sub-pixel Motion Estimation

Once the integer-pixel motion vector is found through integer-pixel motion estimation, the resulting best match block is interpolated at the sub-pixel locations and motion is further estimated at sub-pixel level. Sub-pixel motion estimation achieves better compression by compensating for the loss of motion accuracy as a result of discrete sampling of the camera sensor. Half, quarter, and one-eighth sub-sampling are included in the modern video coding standards [2] [3].

1.1.3.4 Motion Vector

The size of the search window and the level of sub-sample determines the number of bits required for a motion vector. With a square search region size of R, the number of bits for integer motion vector is $2\times\log(R)$ and the value 2 is for horizontal and vertical component of the motion vector. Each sub-sampling adds one additional bit and three level of sub-sampling requires 3 bits. In total each motion vector requires $2\times\log(R) + 3$ bits. Due to the motion coherency in the neighboring pixels representing the same moving object, there can be similarities in the motion vectors from neighboring blocks. To reduce the size for storing motion vector, a virtual motion vector is synthesized from previously encoded neighboring blocks to the left and top. Only the motion vector difference (MVD) between the synthesized motion vector and the evaluated motion vector is stored.

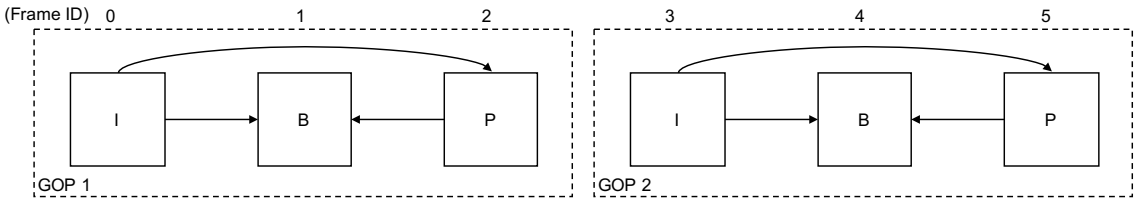


Figure 1.5: A 2-GOP IBP prediction structure consists of an intra-prediction frame (I-frame), a predictive frame (P-frame) and a bi-predictive frame (B-frame). The display order is 0-1-2-3-4-5 and the encoding order is 0-2-1-3-5-4.

1.1.3.5 Motion Residual

The difference between the prediction block and the best match block, referred to as motion residual, is further processed and encoded.

1.1.3.6 Prediction Structures

The frame referencing within a GOP is implemented through prediction structure where the dependency among frames within GOPs is established. The prediction structure determines the number and the type of referencing before encoding process starts. Usually a more sophisticated prediction structure where multiple reference frames are assigned leads to better compression performance. A I-frame uses purely intra-prediction coding while a predictive frame (P-frame) can use both intra- and inter-prediction coding. A predictive frame uses more than one reference frame for inter-prediction is called, bi-predictive frame (B-frame). Fig. 1.5 shows a simple IBP prediction structure demonstrating the usage of all three prediction frame types. Note that the source of arrow is a reference frame for the destination frame. As can be seen, frame encoding order (0-2-1-3-5-4) does not match with display order (0-1-2-3-4-5) due to the prediction dependency. This requires frame buffering at the encoder and

appropriate frame reorder at decoder.

1.1.4 Frequency Domain Coding

Prediction error from both intra-frame prediction and inter-frame prediction are further transformed into frequency domain signals. This process packs the large amount of low frequency signals in the prediction error into few coefficients and helps subsequent entropy coder to achieve better compression. A modified integer discrete cosine transform and discrete sine transform are adopted in the AVC/H.264 and HEVC/H.265 standards.

1.1.5 Quantization

The frequency domain coefficients are (optional) quantized to reduce size. Note that quantization in frequency domain has less negative effect on the visual quality than quantization in spatial domain. This is due to the fact prediction error has very low energy high frequency components. Quantization is an effective tool in managing tradeoff between bitrate and quality. Both AVC/H.264 and HEVC/H.265 standards define 52 distinct quantization levels to allow fine control over the tradeoff.

1.1.6 Entropy Coding

The transformed and quantized prediction error is further entropy coded. AVC/H.264 offers variable length coding and arithmetic coding. The practice of AVC/H.264 shows variable length coding in most scenarios are less effective than arithmetic coding. For

this reason, variable length coding is excluded from HEVC/H.265 standard. The entropy encoded bits along with prediction modes, motion vector, reference frame ID and other assisting parameters are stored in the final output bitstream.

1.1.7 Decoding

The decoder reverses the encoding process by performing entropy decoding, inverse quantization, inverse transform, inverse prediction to reconstruct each basic coding unit and frames. If quantization is applied, the reconstructed video is data-wise not exactly the same as the video data before encoding, resulting in the so called *lossy compression*.

1.1.8 Standard Specifics

The evolution from AVC/H.264 to HEVC/H.265 brings up to 50% better compression performance. The main contributing factor to the improvement is the increased basic coding unit size, from 16×16 in AVC/H.264 to 64×64 in HEVC/H.265. As video resolution increase, larger basic coding unit while achieving same performance as in AVC/H.264 with coding unit subdivision is capable of capturing larger coherent redundancies using less bits. In addition, HEVC/H.265 separates the concepts of unit for coding, unit for motion estimation and unit for transformation, offering significantly more coding freedom which are necessary to achieve higher compression efficiency.

1.1.9 Complexity Analysis

At the low level, the motion estimation process contributes significantly to the overall computation. For coding unit of size N×N, a match in a single location has O(N^2) complexity. Using a search region of size R, the total calculation amounts to O($N^2 \times R^2$). Adding F number of reference frame further increases complexity to O(F×$N^2 \times R^2$). At the high level, the rich choice of intra-frame and inter-frame prediction modes and recursive coding unit subdivisions potentially made HEVC/H.265 significantly more complex than AVC/H.264.

1.2 Multiview Coding

Multiview coding (MVC) shares a majority of coding tools and techniques used in single view coding such as coding unit division, frequency domain coding, quantization and entropy coding [4]. To successfully gain additional compression performance in MVC requires a careful design of inter-view prediction structure (while the prediction structure within a view between frames still applies). A simple four-view capturing system along with a uni-directional (from top to bottom) inter-view prediction structure is shown in Fig. 1.6.

1.2.0.1 Inter-view Redundancy

Frames captured by multiple cameras simultaneously from different perspective are highly correlated. The similarity between frames in different views is referred to as

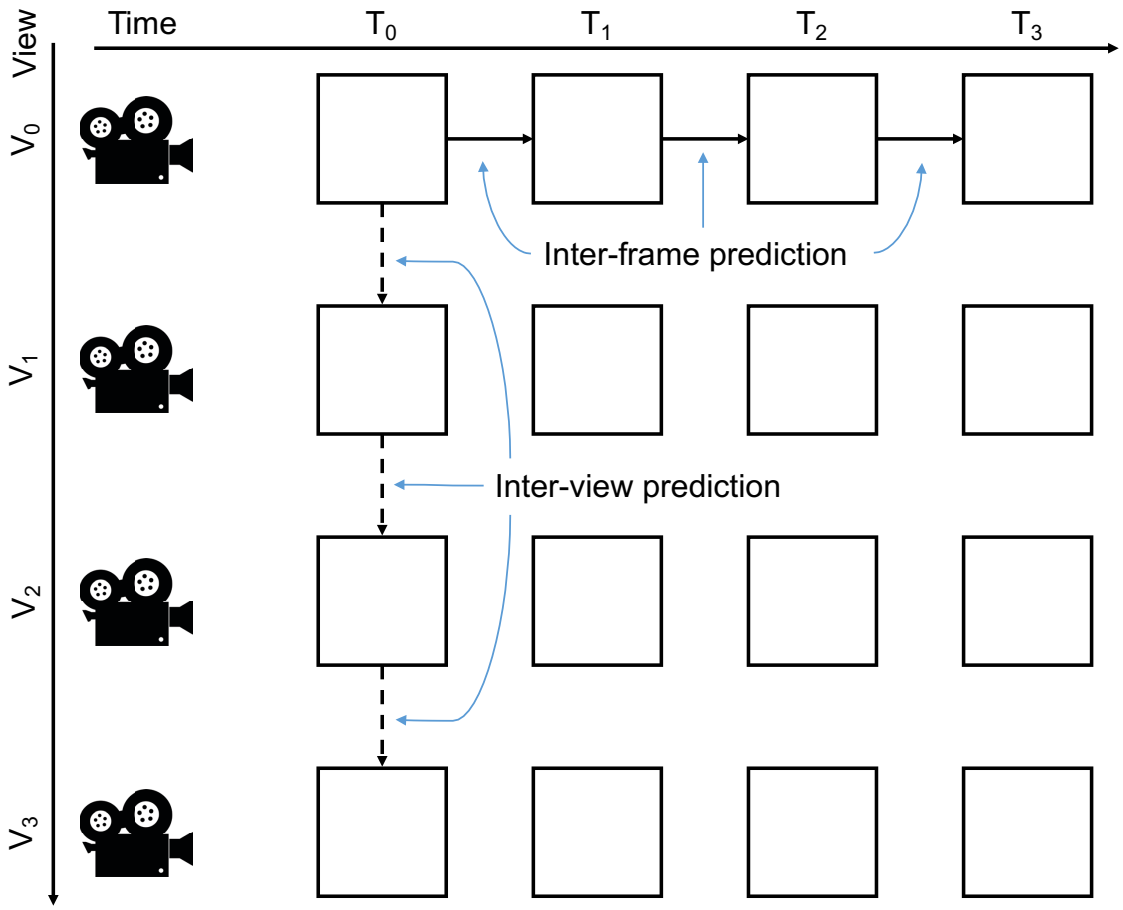


Figure 1.6: Multiview system and uni-directional inter-view prediction (IPP).

inter-view redundancy and can be exploited to further increase compression performance when multiview is available.

1.2.1 Inter-view Prediction

Similar to inter-frame prediction used in single view coding, the same type of prediction using reference frame from another view is called inter-view prediction. Fig. 1.6 indicates both inter-frame prediction and inter-view prediction. The prediction

process is referred to as disparity estimation and the displacement obtained from inter-prediction is referred to as disparity vector.

1.2.1.1 Disparity Estimation

In multiview coding with disparity estimation, the reference frames (the source of the dashed arrow in Fig. 1.6) are provided by neighboring view(s) captured at the same time instance. It is possible to include frame(s) from neighboring view at different time instance to the reference frame list. The benefit of doing so usually do not worth the increased amount of computation and is usually not considered.

1.2.1.2 Prediction Structures

Fully exploiting the inter-view redundancy requires the understandings of the camera array formation and appropriate prediction structure selection. For linear deployed camera array, one-direction linear inter-view prediction structure performs better. Detailed analysis of a wide variety of inter-view prediction structure on the coding performance are given in Chapter 4.

1.2.2 Standard Specifics

Specification for multiview coding in the two generation of standards are similar. The reference view ID is signaled in the bitstream and the first view must conform to its single view coding standard.

1.2.3 Complexity Analysis

The complexity increase in MVC mainly come from the disparity estimation process where the number of reference frame increased for all frames in all views (except non-dependent views such as the first view). Incorporating neighboring view frames in the reference frame list increases the complexity of motion/disparity estimation in each dependent view linearly (O((K+ F)$\times N^2 \times R^2$) where K and F are the number of inter-view and temporal referencing views, respectively). Evaluation at such complexity is impractical even for non real-time applications and fast and accurate motion/disparity estimation algorithm is necessary for any practical encoder design. In this work, a significant amount of effort is put into designing better parallel motion/disparity estimation algorithms to reduce encoding time for MVC while maintaining good compression performance.

1.3 Multi-level Parallelism

In the recent years, the advancement in the semiconductor industry led to the advent of high performance multi-core central processing unit (CPU) and massive cores graphical processing unit (GPU). While the type of suited problem for the two architectures are different, they both process tasks in parallel and reduce execution time over the sequential version (that runs on single core). Achieving better execution performance using parallel computing tools requires the identification of possible parallelisms and appropriate design of parallel algorithms to accommodate a particular architecture. In this section, four levels of parallelism in a modern video encoder are identified.

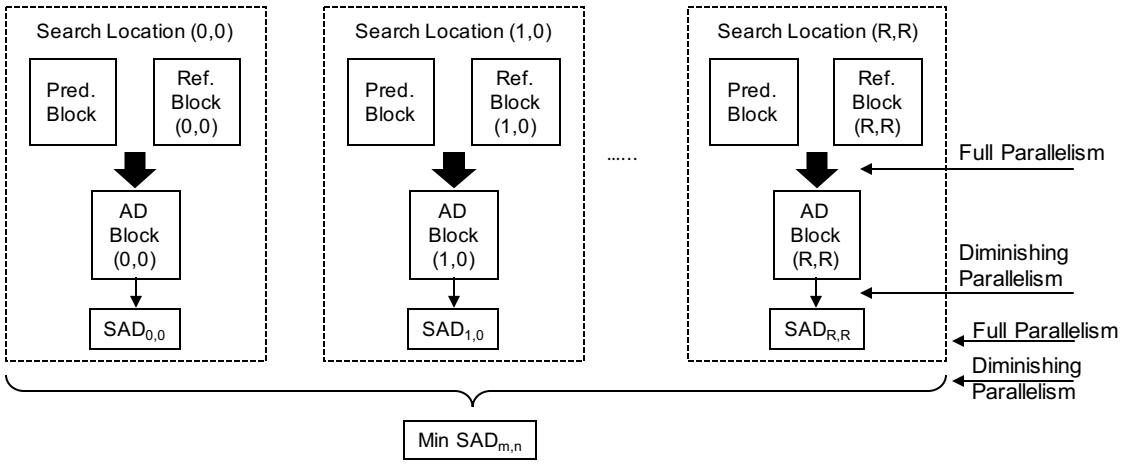


Figure 1.7: Parallelism in SAD calculation and minimal SAD finding.

1.3.1 Pixel Level Parallelism in ME/DE

In the search for best match in the pre-defined search window, the similarity between coding unit and reference block is measured with sum of absolute difference (SAD). The SAD calculations and the process of finding the best match in terms of lowest SAD exhibits significant amount of pixel level parallelism and are illustrated in Fig. 1.7.

1.3.1.1 Sum of absolute difference

To compute the sum of absolute difference for one search location, the first step is to find the pixel-wise absolute difference (AD). The AD calculation for individual pixel is completely independent and can be carried out in full parallelism. The subsequent summation of the AD into a SAD has a diminishing parallelism characteristic. The SAD calculation for individual search location is completely independent and can be fully parallelized. Note that the degree of the parallelism in this case equals to the area of the search window (O(R^2) where R is the search window size) and is typical

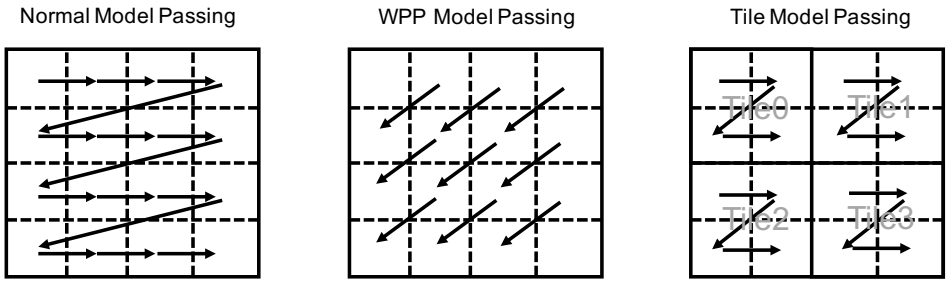


Figure 1.8: Probability Model Passing for normal coding, coding with WPP and coding with tiling.

very high (For typical $R = \pm 64$, area is 16384).

1.3.1.2 Minimal Cost Reduction

Once all the SADs for all locations within the search window are found, finding the best match in terms of lowest SAD becomes a tree reduction problem and thus has a diminishing parallelism.

1.3.2 Coding Tree Unit Parallelism

While parallel encoding of macroblocks is unsupported by the AVC/H.264 standard, HEVC/H.265 introduced two parallel processing tools,*wavefront parallel processing* (WPP) and *tiling processing*, for its coding tree units (CTU). Both parallel tools require slight modification to the entropy coding processing where the probability model is passed in a order different from the conventional raster-scan order. The passing direction for various parallel processing tools are shown in Fig. 1.8. Each block in a grid represents a coding unit and the arrow indicates the direction of probability model passing for the coding units.

1.3.2.1 Wavefront Processing

Wavefront parallel processing (WPP) is a parallel coding tool focused on improving the capabilities for parallel processing with minimal cost in the coding efficiency [6]. Independent encoding and decoding of CTUs is possible if the processing from one CTU row to the next offsets by two consecutive CTUs, creating a wave front processing pattern. The probability models are passed diagonally from one row to the next and has the same global coverage as normal raster-scan order passing, thus preserving the coding efficiency.

1.3.2.2 Tiling Processing

In tiling approach, a frame is flexibly subdivided into rectangular region of CTUs and coding dependency between CTUs are restricted. Unlike the loose dependency between CTUs in different rows in WPP, tiled regions do not require communication between processors for CTU-level entropy coding and thus can be encoded completely independently. However, as the number of tiles increase (size of each tile reduces), the coding efficiency deteriorates due to severe breaking of dependencies in entropy coding. For this reason, WPP is usually superior and used extensively in designing high performance H.265/HEVC codecs [5].

1.3.3 Frame Level Parallelism

The three aforementioned frame prediction types (I, P and B) implies that the number of simultaneous encodable frames heavily depends on the prediction structure. Four

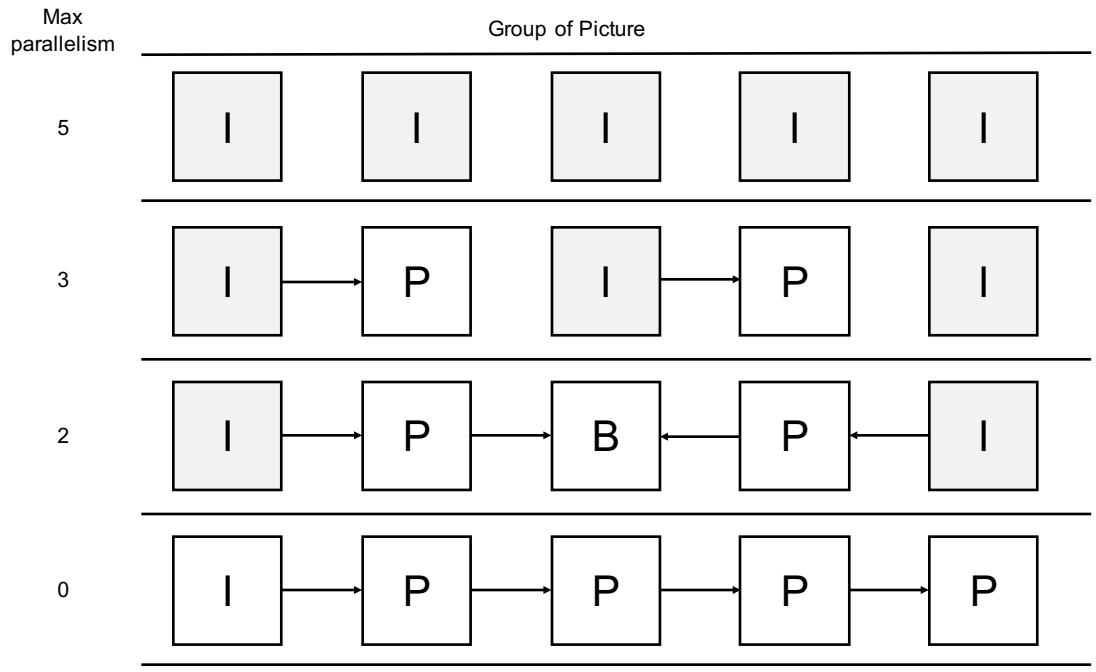


Figure 1.9: Four example inter-frame prediction structures with diminishing degree of parallelism are listed from top to bottom. Each box in the figure represents a frame and shaded boxes are the parallel encodable frames within a GOP.

inter-frame prediction structures with diminishing degree of parallelism (from top to bottom) are shown in Figure 1.9.

1.3.3.1 Frame Independencies

Frame dependency exists only when an encoder uses P-frame and B-frame for coding. As the number of P and B frames with in a GOP increases, the amount of dependencies increase and frame level parallelism decrease. In the scenario for max frame parallelism, all frames are encoded in I-frame type. However, such prediction structure results in high coding inefficiency compared to using partially P and B frames for coding. A tradeoff between coding efficiency and degree of frame level parallelism is a crucial design factor for frame level parallelization.

1.3.3.2 Group of Picture Independencies

A possible solution to frame level parallel processing inherited in the coding structure is the GOP parallel processing. As defined by the standards, GOPs are mutually independent and frames forming the GOPs are inherently independent and can be processed in parallel. The difficulty in single view coding to prepare multiple GOPs can be easily solved in multiview coding, providing a huge potential for speeding up if GOP parallelism can be harnessed correctly.

1.3.4 View Level Parallelism

A view becomes dependent on another view if frames within the dependent view use inter-view prediction. The type of inter-view prediction for all views are specified through inter-view prediction structure and is user-defined (not standard-defined). An in-depth analysis on four commonly used inter-view prediction structures are made in Chapter 3. With flexible inter-view prediction where a view may depend on multiple neighboring views, multiview coding is able to achieve significant better overall coding efficiency. Generally, as the dependency between views increase, the coding efficiency improves and view level parallelism reduces. In the maximum view parallelism scenario, all views are independently encoded without any use of inter-view prediction and no extra compression performance is gained.

1.3.4.1 View Independencies

The type of inter-view dependency is not defined by the standard and can be flexibly specified by the encoding application, bringing a wide range of possible inter-view prediction structure. A common inter-view prediction structure called *IPP* (see dash arrows in Fig. 1.6), where each view makes reference only to the previous view when encoding, reported the highest coding gain [7]. Due to the one-directional chained dependency, the amount of frame parallelism between views are very small. Another popular inter-view structure, *IBP* eases the inter-view dependency with slight loss of coding efficiency.

1.4 Related work

1.4.1 Fast ME/DE

To accelerate the block matching in ME/DE many sub-optimal (CPU-based) techniques have been proposed for AVC/H.264 that are generally applicable to HEVC/H.265. An iterative Hexagon search was proposed in [8] to enhance the RD performance of the earlier diamond search [9]. More sophisticated search algorithms use multiple simple search patterns and local correlations to further improve RD performance. The technique in [10] proposes unsymmetrical-cross multi-hexagon-grid search (UMHexagonS). The work in [11] gains further improvement through enhanced predictive zonal search (EPZS). There are exploratory implementations of ME/DE algorithms on MPA platform of graphical processing units (GPU) [12], [13], [14], [15], [16], [17]. All these implementations except [15] are purposed for single

view video, and have not even been integrated into a complex platform of JMVC, and therefore, not compared with TZsearch, (which is the gold standard for MVC). These algorithms, while demonstrate the potential of MPA, do not exhibit impressive performance comparing to the CPU-based full search. The best performance speed up with respect to sequential full search reported in these references range from 9 to 17.

The main reason for the low performance of these algorithms is that they fail to fully exploit the massive-parallelism afforded by the programming environment of MPA of GPU by performing many parallelizable tasks sequentially [12], [17]. Works in [12] and [13] lack due consideration to the algorithmic scalability within the capabilities of the underlying hardware, resulting in unnecessary resource saturation and scaling limitations. Some of these algorithms also gain speedup at the expense of significant compromise in bit-rate and PSNR quality. That is because these algorithms fail to properly handle inter-partition MV/DV dependencies in MVC [12], [13], [16] (see Section 2.3). The best implementation of MVC, to date, in terms of speed, bit-rate, and PSNR quality, is the JMVC reference software with TZsearch mode for ME/DE. Therefore, all performance evaluations of proposed algorithm in this paper is made with respect to JMVC reference software.

1.4.2 Fast Mode Decision

A rich choice of hierarchical partitioning modes within the CTU is the main reason for higher coding efficiency as well as the high computational complexity in HEVC. To improve the execution time of HEVC requires additional optimization steps beyond the efficient processing of ME, through early termination of partitioning within CTU. The scheme in [18] stops further partitioning of CU into smaller CUs if the skip mode

has been selected. An early termination scheme where the partitioning mode with largest PU size is first checked, was proposed in [19]. If in this mode PU produces a coded-block-flag equal to zero, the processing of all PUs within this CU is skipped. The work in [20] improves upon this termination scheme by halting the processing of all other PUs if both the MV difference and coded-block-flag are turned out to be equal to zero. Further, the latest related work on HEVC coding in [21] proposed a ME technique that skips the processing of all CU of size 8×8. For the remaining larger CUs all 17 possible symmetric partitioning modes are evaluated. Three modes with lowest costs collectively determine the early termination decision for the processing of CTU subtrees.

Above algorithms are estimated to yield a speedup factor of about 1.6 to 3 with varying loss in the RD performance. There are also fast mode decisions proposed for intra-prediction [22]. However, intra-prediction consumes very little time in comparison with ME/DE processing. These efforts reveal the potential of reducing encoding time by appropriate skipping of some of CUs and PUs. However, the increasing number of views in MV-HEVC brings significantly more inter-prediction for each PU within a CTU, potentially slowing down the processing of PUs for the existing algorithms.

1.4.3 Multiview Coding Scheduling

To maximum computational resource usage and reduce unnecessary stalls involved in multiview coding due to inter-view and temporal dependencies, works in [23] and [24] present scheduling algorithms for parallel MVC encoding at the frame level on a multi-processor system for a given prediction structure. In these works a prediction structure is used to build a directed acyclic graph where frames across the temporal

and inter-view domains form the vertices of the graphs and the coding dependencies form the edges. Starting with I-frames vertices in view 0 as pair of roots, the encoding scheduler inspects all the neighboring vertices and assigns them to one of the available CPU cores. Then, for each of those neighbor vertices in turn, it inspects their neighbor vertices which are not yet coded, and so on. The process continues until all the GOPs across all the views are encoded. This requires a complex scheduler that traverses the graph to discover and schedule the frames that are ready to be dispatched for the execution on one of the many identical CPU cores. Further, in this scheme the workload, in terms of the number of frames ready to be scheduled, at each coding stage varies greatly across the stages. This results in under-utilization of CPU cores or inadequate number of cores for efficient parallelism depending on the coding stage. To alleviate this problem by creating enough workload to keep all the CPU cores busy, the work in [23] proposes the processing of multiple GOPs across all the views in parallel, further complicating the scheduler. The scheduler task becomes even more cumbersome considering the fact that at each stage of encoding frame vertices take widely different execution times depending on the number of their immediate descendants in the graph and the nature of edge dependencies (temporal or inter-view).

In contrast, this work presents a simple scheduling scheme where the number of frames to be encoded does not change across the coding steps. As will be described in the Section 3, this is achieved through a simple encoding step time shift. The simplicity of the proposed parallel scheduling scheme results in the more complex prediction structure of IPP to have a more efficient parallel implementation compared with IBP.

1.5 Scope of this work

The goal of this work is to investigate and develop highly efficient algorithms and optimization techniques for the previous generation video encoder (based on H.264/AVC) and current generation video encoder (based on H.265/HEVC) by exploiting a variety of parallelisms within the encoders.

1.5.1 Highly Parallel ME/DE Algorithms

1.5.1.1 Massively Parallel Integer-pixel ME/DE Design for AVC/H.264

To solve extremely high computation due to the ME/DE, a full block search based massively parallel motion estimation algorithm is proposed. In this work, each computing thread handles the search for one location and collectively a search area number of threads processes the entire search range. In each thread, the cost for all sizes of coding units are progressively aggregated, reducing computation by a factor of seven compared to sequential aggregation. To solve the problem of losing neighbor motion vector information as the result of progressive aggregation, the motion vector for the largest block is used for all cost calculations. The results show, with insignificant loss in coding efficiency, the proposed algorithm outperforms the full search and TZsearch estimations on a sequential processor, by a factor of 300 and 4, respectively. An improved version with adaptively adjustable search range achieved another two times speedup with insignificant coding efficiency loss.

1.5.1.2 Massively Parallel Integer-pixel ME/DE Design for HEVC/H.265

When moving to HEVC/H.265 standard, significantly more complex coding unit structure fully saturates the computing hardware, leading to reduced performance. To overcome this problem, a predicate algorithm for skipping search on static coding units is proposed. Furthermore, inter-pixel similarity within the prediction is exploited to reduce the workload on all computings thread. The combination of the two algorithms yields significant speedup with no loss to the coding performance.

1.5.1.3 Accelerating bi-directional and sub-pixel ME/DE with Multimedia Instructions

Massively parallel processing on MPA is ideal for integer ME/DE due to the existence of a large search region, with each core responsible for up to thousands of SAD computations. However, for sub-pixel and bi-directional ME/DE, the number of search points for a PU is a tiny fraction of its integer ME counterpart, and thus not suitable for processing on MPA. Sub-pixel and bi-directional ME/DE are also not suitable candidates for multi-threading on multicores as the small processing workload does not justify the significant execution overhead. On the other hand, streaming SIMD extensions (SIMD) instructions set offered by all modern CPUs that allow packed data to execute in a parallel fashion provide a promising alternative for sub-pixel and bi-directional ME/DE proceeding. SIMD optimized sub-pixel and bi-directional ME/DE are proposed to further improve ME/DE procedure.

1.5.1.4 Multi-threaded Wavefront Parallel Processing Design

A novel implementation of multi-core parallel processing of CTU is proposed and implemented. Independent CTUs on the wavefront are grouped into batches and processed in parallel (in any order). In comparison to traditional multi-core parallelizing strategy, this method improves the computing resources usage at the early phase of WPP and provides a speed of three for multiview coding.

1.5.2 GOP Level Parallelization

To overcome the huge computational cost associated with ME/DE in the multiview coding, computer cluster with heterogeneous computing components to provide concurrency and multi-level parallelism at coarse grain is adopted. A multiple-view-parallel, multiple-interleaved group of pictures (multiple-IGOP) scheduling scheme is proposed for MVC. When evaluated over eight views, with no loss in rate distortion (RD) performance, the proposed scheme outperforms view-sequential coding by a factor of up to 12.4 and 12.3, respectively, for two popular prediction structures, IBP and IPP.

1.5.3 Quantization Parameter Based Fast Mode Decision

Further improvement of HEVC/H.265 relies on optimization at the global scope where multiple procedures such as intra-prediction, inter-prediction, DCT, quantization are selectively skipped. A majority of high level optimization focuses on using motion information to early determine prediction modes, avoiding the necessity to evaluate

all modes. By carefully examining the effect of quantization parameter on the coding time performance, a novel quantization parameter based fast mode decision is proposed. This algorithm sets constrain on the depth of coding unit modes based on the quantization parameter and achieved up to 6 times speedup. To avoid coding efficiency loss for video with highly complex content, special type of coding unit are defined and mode decision for those coding units are allowed to evaluate further. The main advantage of this fast mode decision algorithm is its ability to adapt to the quantization parameter and video content.

1.6 Overview of Chapters

This dissertation consists of four chapters. Chapter 2 presents the design of two novel parallel motion estimation and disparity estimation algorithms to significantly reduce encoding time. Both algorithms are applicable to AVC/H.264 multiview extension and HEVC/H.265 multiview extension. Chapter 3 discusses the use of GOP level parallelism in conjunction with an efficient scheduling scheme to achieve efficient parallel encoding on computer cluster. Chapter 4 discusses high level optimization techniques to reduce the evaluation for multiple functional units and proposes ME/DE algorithms tailored for HEVC/H.265. Chapter 5 concludes this dissertation by summarizing the key points for this work.

Chapter 2

Massively Parallel Motion and Disparity Estimation Algorithms[1]

Multiview video coding (MVC) has recently received considerable attention. It is proposed as an extension of H.264/AVC standard for multiple video source compression. To resolve the extremely high computational complexity of MVC (and in fact other advanced video coding techniques), requires development of suitable parallel algorithms that are amenable to implementation on low cost massively parallel architecture (MPA); platforms that have found a common place due to recent advances in the parallel computer architecture. The high complexity of MVC is due to its prediction structure, where motion estimation (ME) between the frames, and disparity estimation (DE) between the views, contribute to more than 99% of overall complexity of the coder. This chapter presents the development and implementation of a scalable massively parallel fast search algorithm to, significantly, reduce the computational cost of ME/DE over the current best available full block matching,

[1]The material contained in this chapter was previously published in "*IEEE Transactions on Circuits and Systems for Video Technology*"

and sub-optimal fast search algorithms. The proposed massively parallel fast search algorithm (DZfast), when evaluated over eight views, outperforms the existing full search and fast search MVC algorithms by a factor up to 245.8 and 8.4, respectively. This speedup comes at no or minute loss in rate-distortion (RD) performance.

2.1 Introduction

Multiview video coding (MVC) is defined in Annex H of the state-of-the-art video coding standard H.264/AVC [3]. Common application of MVC are 3D movies/TV, free view TV, and immersive teleconferencing [25], [26]. In these applications, multiple cameras are deployed to capture dynamic scenes simultaneously. MVC in addition to taking the advantage of inter-frame temporal similarity in motion estimation (ME), as in conventional video coding techniques, exploits inter-view similarity to achieve about twice the higher coding efficiency compared with coding each view independently (simulcast). Inter-view similarities are used in disparity estimation (DE) in between the neighboring view video sequences. The most common technique for ME/DE is *block matching* [25]. This technique computes motion vector (MV) and disparity vector (DV) between two best matched blocks of pixels in two frames (temporal or inter-view). Block matching, even in single view video coding, is a time-consuming process amounting to about 80% to 90% of the total encoding time.

Table 2.1
Execution Time profiling of JMVC: Contribution of full search block matching in ME and DE for video sequence"Ballroom"

View ID	**0**			**1**			**2**		
Search Range	±32	±64	±128	±32	±64	±128	±32	±64	±128
DE + ME % Time	97.25	98.73	99.76	99.25	99.48	99.84	97.74	99.38	99.80

When combining ME and DE over multiple views, block matching takes even longer, consuming almost the entire computation time. Table 2.1 shows the profiling of MVC test-bench, the joint multiview video coding (JMVC) software suite [27] for three

views and for three search ranges, taking up to 99.80% of the computation time. To accelerate ME/DE in the block matching, many sub-optimal techniques have been proposed in literature [8], [9]. These algorithms trade video quality and bit-rate for faster computation time. JMVC implements four sub-optimal search algorithms. They are full search, log search, spiral search and TZsearch. Among the sub-optimal algorithms for MVC, TZsearch provides the fastest implementation with the lowest degradation in terms of peak signal to noise ratio (PSNR) and bit-rate. Because of its superiority, TZsearch is also adopted for the next generation video coding standard (H.265) test software, high efficiency video coding (HEVC) test model (HM) [2],[28]. While a highly efficient algorithm, such as TZsearch, is sufficient for model testing, it does not provide a solution for real-time implementation of complex coding techniques needed for MVC.

To ease the coding complexity, application specific integrated circuit (ASIC) solutions are proposed for MVC [29], [30]. However, such hardware designs require significant simplification of coding process. The work in [29], as an example, presents the design of a MVC on an ASIC platform. To achieve a high processing rate, the design has made several major simplifications to the MVC coding when compared with the MVC reference software, JMVC, resulting in significant degradation in bit-rate and PSNR, defeating the purpose of achieving high coding efficiency in MVC. The adoption of simple prediction structure in the temporal domain, and small search region for the fast ME/DE algorithm [31], [32], [33] results in about 2.5 to 3 dB degradation in PSNR over the full range of bit-rates. In addition, the architecture in [29], to implement an efficient pipeline for processing video sequences from multiple views, uses the original image pixels instead of reconstructed pixels for prediction, resulting in a further degradation in video quality by up to 0.77 dB in PSNR.

Similarly, ASIC based ME/DE design in [30] achieves its speedup at the expense of performing the TZsearch based block matching in the search region for ME/DE for

only a small fraction of reference frames (15%). For the other 85% of reference frames, TZsearch over the full search region is replaced by a block matching using a maximum of 15 predictors from the neighborhood in spatial (within the current frame), temporal (same video sequence) and inter-view (neighboring video sequences) domains. This simplification leads to an 11% increase in bit-rate and 0.11 dB in PSNR loss [30]. In general, ASIC implementations achieve their high frame rates due to their highly optimized architecture, at the expense of reduced flexibility, programmability and scalability.

The availability of cost-effective MPA computing platforms [34], [35], provides an opportunity for the development of MVC parallel algorithms that are fast, and nearly-optimum, without sacrificing video quality or bit-rate. Characteristics of MPA platforms are the availability of a large number of computing cores that are generally organized as single instruction multiple data (SIMD), or single program multiple data (SPMD) computing paradigm. MPA embeds several blocks of fast shared memory allowing efficient coordination of multiple streams of the same program by the computing cores. In addition, MPA gains unprecedented performance through high bandwidth memory (Gigabytes per second (GB/s)).

There are exploratory implementations of block matching algorithms on the MPA platform of graphical processing units (GPU) [17], [13], [15]. All these implementations, except [15], are proposed for single view video, and have not even been integrated into the complex platform of JMVC, and therefore, not compared with TZsearch, (which is the gold standard for MVC). Even reference [15], the only paper on the GPU implementation of MVC, does not report the prediction model (see next section). Further, this paper reports the comparison with the outdated EPZS [11] which is no longer a search option in the JMVC software suite [27]. These algorithms, while demonstrating the potential of MPA, do not even exhibit impressive performance compared to the CPU-based full search. The best performance speedup

with respect to sequential full search reported in these references range from 9 to 17. As will be shown in this chapter, to gain an advantage over TZsearch an improvement in speedup over the sequential full search by at least one order of magnitude over the existing reported implementations is needed.

The main reason for the poor performance of algorithms in [17], [13], [15] is that they lack due consideration to the algorithmic scalability within the capabilities of the underlying hardware, and the inefficient use of memory bandwidth, resulting in unnecessary resource saturation and scaling limitations. These works achieve large scale parallelism by processing huge number of macroblocks simultaneously, thereby, saturating the resources of an MPA. However, they fail to process each macroblock in an efficient manner, by performing many parallelizable tasks sequentially. Further, these algorithms gain speedup at the expense of bit-rate and PSNR. This is due to the fact that, due to the hardware limitation of MPA, the parallel processing of a large number of macroblocks comes at the expense of significant reduction of the size of the search region (*e.g.* 20.48 fold reduction [15]). Further, parallel processing of macroblocks, as employed in [17], [13], [15], results in poor handling of spatial MV/DV dependencies in MVC, as required by H.264/AVC. (See Sec. 2.3). From our investigation the best implementation of MVC, to-date, in terms of speed, bit-rate, and PSNR quality, is the JMVC reference software with TZsearch mode for ME/DE, on a single-thread, single-core CPU. Therefore, all performance evaluations of proposed algorithms are made with respect to JMVC reference software as an anchor.

In this chapter, first, the methodology for the parallel full search (GPUfull) is explored and the limitation of the naïve parallel full search schemes, so far explored in the literature, is highlighted. Next, a highly efficient parallel fast search algorithm (DZfast) for ME/DE is proposed. This algorithm fully exploits the massive-parallelism of MPA, and employs a dynamic programming technique for data reuse to achieve significant

improvement with respect to the existing sequential full search and best performing sub-optimum search algorithm (TZsearch) for MVC to reduce encoding time. This reduction in the execution time comes with no, or insignificant, degradation in picture quality or bit-rate. Further, the scalability of the proposed algorithm to multiple MPA computing units when they are available, is demonstrated.

In this work it will be shown that parallel processing, even on an MPA with hundreds of cores, is not just a naïve parallel implementation of an algorithm such as parallel full search. The development of efficient fast parallel algorithms, similar to their sequential counterparts, requires a careful analysis of the multimedia algorithm, with a good understanding of the resource features and limitations of the underlying MPA.

This chapter is organized as follows. Section 2.2 presents a brief description of MVC prediction structures. Section 2.3 discusses the opportunities for massive parallelism in the full search algorithm. It also presents the implementation and performance results of the massively parallel full search. Further, it highlights the limitations of naïve parallel full parallel search implementation. Section 2.4 presents the proposal for the massively parallel fast search algorithm. Section 2.5 covers the performance analysis for the proposed algorithm, and compares the results with alternative algorithms. Section 2.6 discusses the applicability of the proposed technique to HEVC.

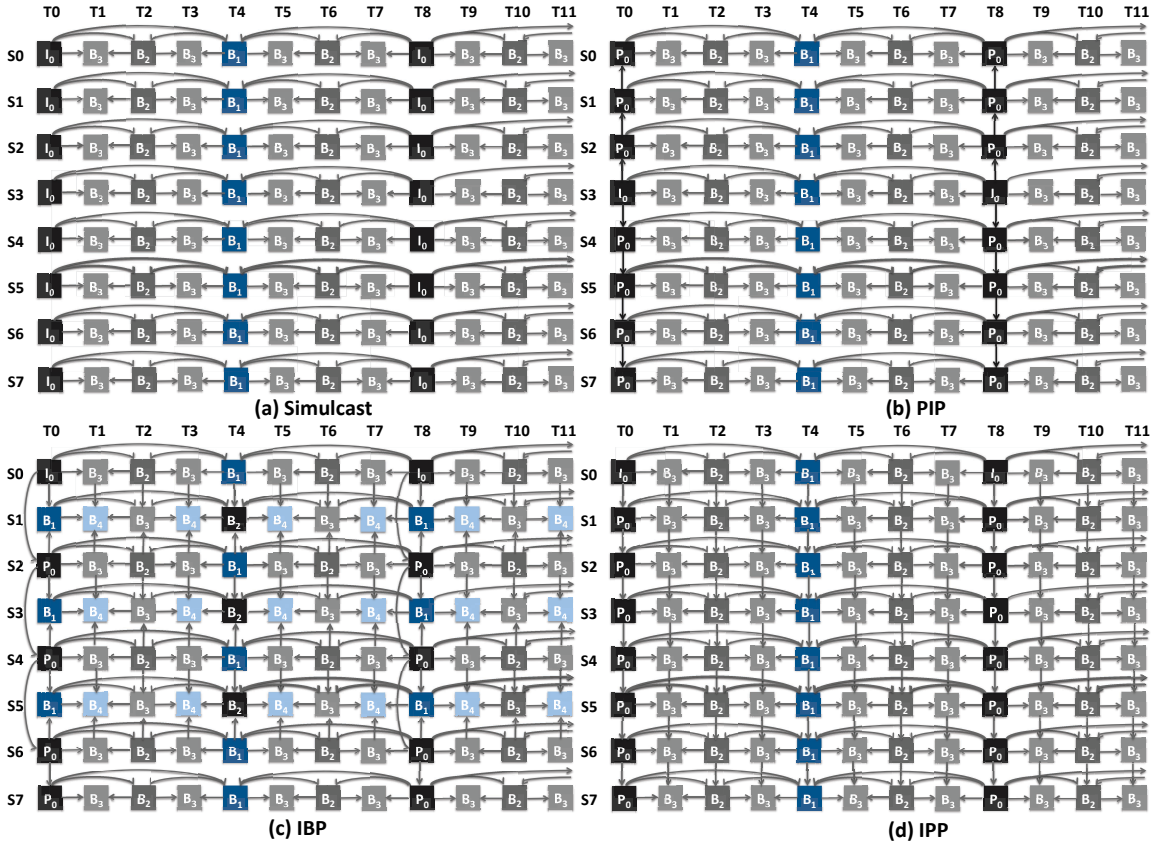


Figure 2.1: Multiview prediction structures: (a) Simulcast: independently coded without inter-view prediction, (b) Inter-view prediction for key frames only with PIP structure, (c) Inter-view prediction for all key and non-key frames with IBP structure, (d) Inter-view prediction for all key and non-key frames with IPP structure.

2.2 Multiview Prediction Structures

2.2.1 Temporal and Inter-view Prediction Structures

Fig. 3.2 presents several prediction schemes for MVC. Focusing specifically on Fig. 3.2 (a), in the temporal domain (T0 to T8), a group of picture (GOP) is fenced by two consecutive intra-coded picture frames (I frames), known as key frames (T0 and

T8). Frames between key frames are non-key frames (T1 to T7). For any single view, the coding concept of hierarchical-B frames was introduced in [33], where all non-key frames are B picture frames, and are typically predicted by using the two nearest picture frames of the next higher level in temporal domain as references. This temporal prediction structure implies that the coding of a picture frame has to be preceded by the coding of its reference frames, and therefore, makes references to the reconstructed versions of the reference picture frames, instead of the original reference picture frames.

Having multiple cameras capturing the same scene simultaneously, predictions can be formed between the video sequences (inter-view domain) in addition to temporal domain within a single video sequence. However, as the number of views increases, the prediction structure becomes highly diverse. The straightforward solution is to encode each view independently using the state-of-the-art codec H.264/AVC, as GOP, as shown in the *simulcast* prediction structure of Fig. 3.2 (a). However, this method fails to exploit the inter-view dependencies and results in significantly higher bit-rate [25], [26].

In addition to GOP formation and hierarchical-B frames for temporal prediction in a single view, it is possible to also include inter-view prediction in the coding process. Depending on whether inter-view prediction is applied to non-key frames (T1 to T7 in Fig. 3.2), two types of strategies are proposed in [33]. Fig. 3.2 (b) with no inter-view prediction for the non-key frames, has a PIP (P frame,..., P frame, I frame, ..., P frame) prediction structure. Fig. 3.2 (c), and (d), on the other hand, where inter-view prediction is applied to non-key frames, respectively, have IBP (I frame, ..., [B frame, P frame], ..., [B frame, P frame], P frame) and IPP (I frame, P frame, P frame, ..., P frame) structures. A significant increase in the rate-distortion (RD) performance can be obtained with inter-view prediction [33], with IPP being the best performing structure. From the encoding implementation point of view, IPP structure suffers

from the dependency chain among the views. IBP and PIP structures ease the inter-view dependency with a slight trade off in RD performance (a degradation of 0.25 and 0.5 dB, respectively, for the same bit-rate). IBP prediction structure is adopted in JMVC [27].

2.2.2 Multiview Video Coding (MVC)

In the block matching technique involving temporal ME or inter-view DE, a macroblock of size 16×16 is divided into seven different partition sizes; 4×4, 4×8, 8×4, 8×8, 16×8, 8×16, and 16×16, resulting in 41 different partitions. For each partition, ME, and/or DE, is performed in the temporal ME and/or inter-view DE domain, respectively, to find the best match in the search region using a RD cost function.

Consider a partition, p, of size $\mathbf{B}_p$ in a macroblock. For this partition, let MV denote the MV/DV for a location in the search region, $\mathbf{S}$, in one of the reference frames in the multiview prediction structure. Let $\mathbf{X}_p$ and $\mathbf{C}_p(MV)$, respectively, denote the pixel matrices for partition p, and the one identified by MV in the search region of the reference frame. The parameter MVP_p is used to represent the MV/DV predictor associated with this partition. In H.264/AVC, MVP_p [3], [5] is calculated from the median of MV/DV of the available three neighboring partitions to the left (MV_A), top (MV_B), top-right (MV_C) and possibly top-left (MV_D), if one of the other three MV/DVs is not available. The RD Lagrangian cost function is evaluated as,

$$J_p(MV|\lambda) = SAD(\mathbf{X}_p, \mathbf{C}_p(MV)) + \lambda Rate(MVD_p) \tag{2.1}$$

where,

$$SAD(\mathbf{X}_p, \mathbf{C}_p(MV)) = \sum_{i \in \mathbf{B}_p} |\mathbf{X}_p^i - \mathbf{C}_p^i|$$

and MV/DV difference ($MVDp$) is expressed as,

$$MVD_p = MV - MVP_p$$

The minimal cost for partition p is found by,

$$(J_{min,p}, MV_{min,p}) = \arg\min_{MV \in \mathbf{S}} J_p(MV|\lambda) \tag{2.2}$$

A RD Lagrangian cost function formulation similar to (2.1) [36] uses this set of 41 $(J_{min,p}, MV_{min,p})$ pairs in the mode decision process, as specified in the H.264 standard. The MVD_p and the pixel residuals corresponding to selected partitions are coded, and transmitted in the bit-stream.

2.3 Implementation of Multiview Video Coding on Massively Parallel Architectures

2.3.1 Exploiting Parallelism in Multiview Video Coding

There are ample opportunities for parallelism in MVC at several levels. These range from frame parallel processing in temporal and inter-view domains, to parallel processing of macroblocks in a frame, to parallelism within a macroblock at the pixel-level.

Due to the prediction structure of MVC and the coding dependencies, the scope for parallel processing at the macroblock and frame levels is limited, and best implemented through familiar thread-level parallelism using the multi-core structures with dynamic thread-scheduling reported in literature (*e.g.* [37], [38], [23]). Parallelism at the higher levels is ill-suited for mapping to MPA computing platforms. Threads on MPA platforms, unlike on multi-core systems, are lightweight, with very little creation overhead. MPA platforms also need thousands of threads for full efficiency. Pixel-level data parallelism within a macroblock, on the other hand, is well-suited for massive parallelism on an MPA platform. This chapter focuses on the pixel-level parallelism within a macroblock on an MPA platform.

2.3.2 Exploiting Parallelism in Full Block Search

Evaluation of (2.1) in the full search scheme is slow in terms of the search speed, as every location in the search region has to be visited once. The algorithm has a complexity of $O(SR^2)$, requiring large computation time, even for a small search range (SR), as shown in Table 2.1. JMVC employs TZsearch [27], a sub-optimum but efficient fast search algorithm, that reduces the overall computational complexity, by a large factor of up to 70, while maintaining good RD performance.

Fast search algorithms available to-date (including TZsearch) have been designed to reduce the number of computations. Their early termination conditions and step-by-step search patterns are naturally sequential and leave little room for massive parallelization. Therefore, these fast search algorithms are not well-suited for mapping to MPA. On the other hand, full search algorithm has the advantage of being amenable to mapping to fine-grain parallelism of MPA, and can be accelerated with high efficiency. Hence, to be a candidate for implementation on an MPA platform, a

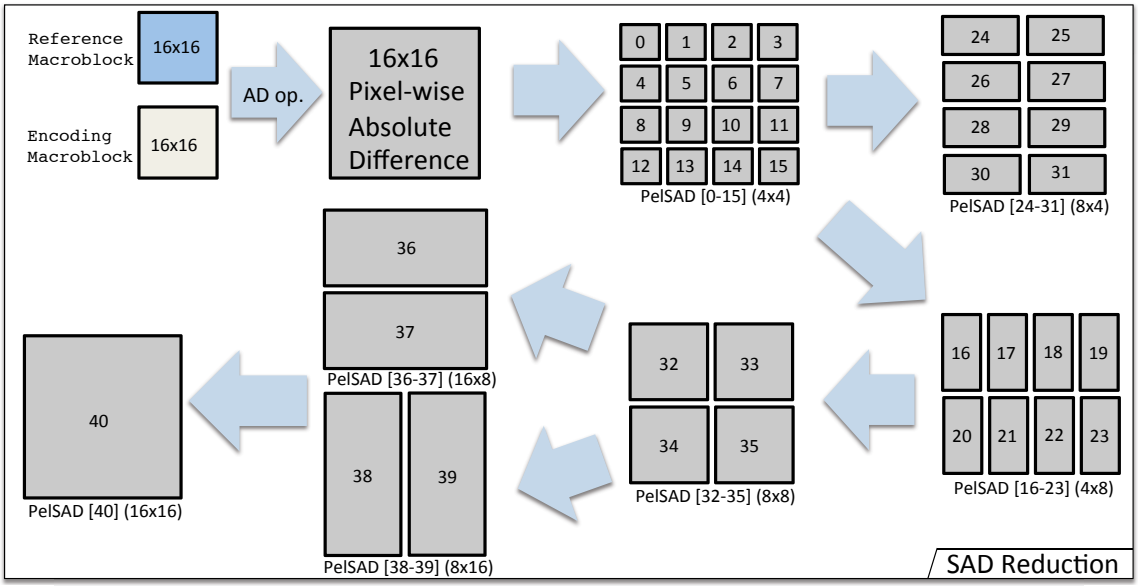


Figure 2.2: Sum of absolute difference (SAD) reduction.

suitable algorithm should not exclusively aim to reduce the number of computations, but rather increase opportunities for fine-grain parallelism.

This section presents the proposed massively parallel and highly efficient full search scheme for MVC. The scheme is characterized as follows:

i) From (2.1), it can be inferred that for a given MV, the sum of absolute differences (SAD) for larger partitions are, progressively, aggregated by the summation of SAD values of smaller partitions. Therefore, a dynamic programming technique [39] as shown in Fig. 2.2 can be employed to perform this progressive aggregation. The minimal partition size, defined by the standard, is 4×4. This indicates that as the SADs for 16 4×4 partitions are progressively computed, SADs for other partitions can be computed simply through the aggregation of SADs for these smallest partitions. The absolute difference for each pixel is highly parallelizable and can be easily implemented on an MPA. This is in contrast with the work in [15] where SADs for 41 partitions are computed separately, thereby, increasing the work load significantly.

ii) The calculation in (2.1) can be carried out independently for different values of MV in the search region. For MVs from the same neighborhood, $\mathbf{C}_p(MV)$ values share a large number of reference pixels. Thus, as will be seen later, they can be loaded into the high-speed shared memory of MPA for use by many computing cores.

iii) To compute (2.1), for each partition p, MVD_p must first be calculated from MVP_p. As was said before, in H.264/AVC standard, MVP_p is derived from the median of three spatial neighboring partitions that are already encoded. This creates a dependency between macroblocks and the partitions within a macroblock, that does not yield to parallelism. Parallel processing of macroblocks requires setting the MVP_p to zero, resulting in significant degradations in PSNR and/or bit-rate [5]. This is a major drawback of work in [17], [13], and [15], where macroblocks are processed in parallel. To break this dependency, the MVD_p for the 16×16 partition (which always exists) is applied on all partitions within the same macroblock. The simulations show that this a good approximation, with negligible loss of PSNR or bit-rate. This simple approximation, as will be demonstrated later, allows for massive parallelization of (2.1).

2.3.3 Implementation of Full Search on the MPA Platform

2.3.3.1 GPU parallel programming paradigm

The huge opportunities for parallelism available in MVC, can be exploited for implementation on any MPA computing platform that supports fine-grain parallelism. This work uses the NVIDIA GPU parallel computing tool CUDA [2]. In CUDA, parallel

[2]Compute Unified Device Architecture (CUDA™) [40]

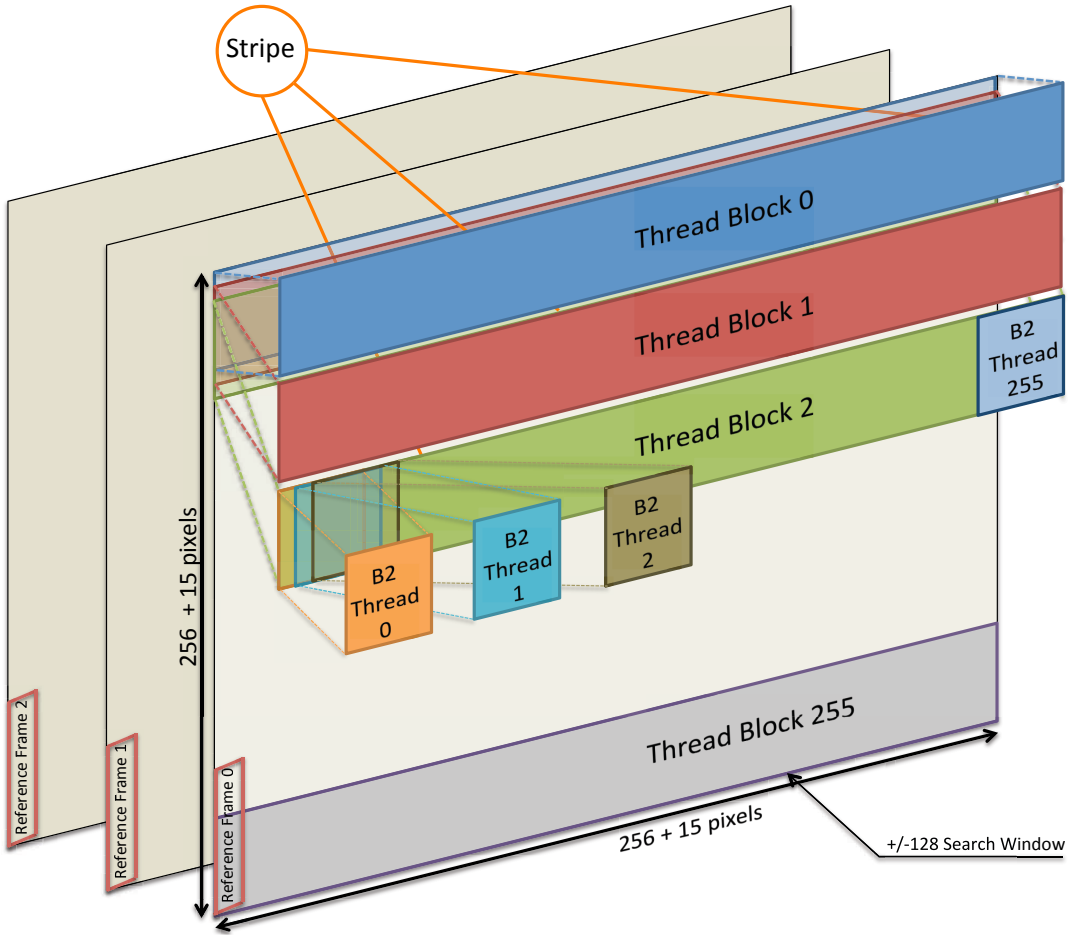


Figure 2.3: Lagrangian cost function kernel for $SR = \pm 128$.

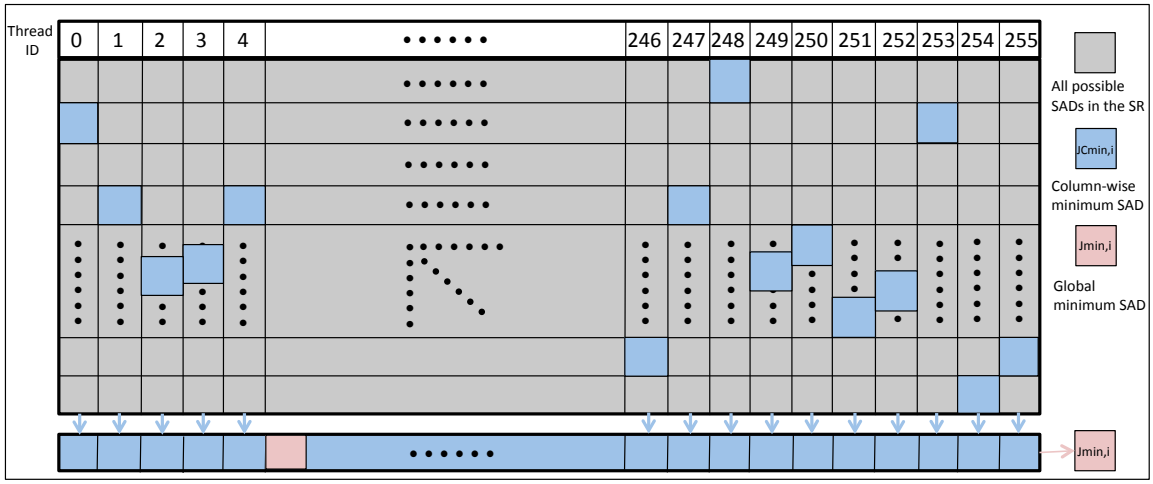


Figure 2.4: Minimizer kernel for $SR = \pm 128$.

programs are encapsulated in *kernel* functions that model single instruction multiple threads (SIMT) computing paradigm. All copies of parallel program, *threads*, execute the same set of instructions, however, on different data. Threads are further grouped into *thread-blocks*. Thread-blocks are in turn arranged in a *grid* [40]. Thread-blocks are executed on the GPU's streaming multiprocessors with each having 32 computing cores, and executing 32 threads, (a *thread warp*), simultaneously.

2.3.3.2 GPU-based full search

With this GPU architecture, the process of parallel GPU-based full search (GPUfull) is carried out in four phases using two CUDA kernels [41].

Phase i: Transfer of the macroblock and its reference frames into GPU global memory.

Phase ii: Launch of the RD Lagrangian cost function kernel to compute all $J_p(MV|\lambda)$ in the search region for 41 partitions according to (2.1) as depicted in Fig. 2.3.

Phase iii: Launch of the minimizer kernel to find the ($J_{min,p}$, $MV_{min,p}$) pairs for all 41 partitions, according to (2.2) as shown in Fig. 2.4.

Phase iv: Transfer of the data structure $\{J_{min,p}, MV_{min,p}|p = 0...40\}$ back to CPU.

Subsequent to these four phases on the GPU, all necessary computation for the RD Lagrangian cost functions, to select the best mode, are evaluated on the CPU.

Two separate kernels help to achieve the best performance through appropriate allocation of resources, by specifying the best arrangement of threads in thread-blocks and thread-blocks in the grid for each kernel. Details of these two GPU kernels for full

search are discussed in [41]. In the RD Lagrangian cost kernel, search region is divided into cascade of overlapping "strips" with each strip having a width of search region and the height of macroblocks (Fig. 2.3). Consecutive strips are offset by one pixel in the vertical direction. A strip consists of all necessary pixels to perform ME/DE for one row of MVs in the search region for 41 partitions. The cost function computation for a single strip is assigned to one CUDA thread-block. Each thread in a thread-block computes all the RD Lagrangian cost functions in (2.1) for 41 partitions for one value of MV. The number of thread-blocks and thread in a thread-block, are exactly twice the absolute value of SR. Upon the completion of this kernel, all $J_p(MV|\lambda)$ in the search region for 41 partitions are computed. For a $SR = \pm n$ this kernel generates $41 \times (2n)^2$ cost values ($= 2,686,976$ for $n = \pm 128$ in the experimentation conducted).

The second kernel finds the ($J_{min,p}$, $MV_{min,p}$) pairs for all 41 partitions according to (2.2). Unlike the minimum finder function built in the CUDA library [3], and the implementations in [17], and [13] that operate on a single vector, the minimum finder in this kernel operates on multiple vectors in parallel. (one vector for each of 41 partition). This parallel processing on multiple vectors results in large performance improvement of this kernel [41]. The completion of this kernel produces 41 pairs of ($J_{min,p}$, $MV_{min,p}$), one for each partition.

The reason for significant improvement of GPUfull over implementations in [17], [13], [15] is that two kernels in GPUfull are designed with one aim in mind; to maximize parallelization of various computations required for the ME/DE for a single macroblock. With the efficient use of GPU resources, the proposed algorithm is able to increase the search region from 32×32 in [13], 64×64 in [17], and 128×25 in [15], to 256×256, and at the same time achieve one order of magnitude better performance. Note that implementations in [17], [13] and [15], all the macroblocks in a frame are scheduled to the GPU in parallel. Concurrent launch of thousands of macroblocks

[3] `Isamin()` function in CUDA basic linear algebra subprogram (BLAS) library [42]

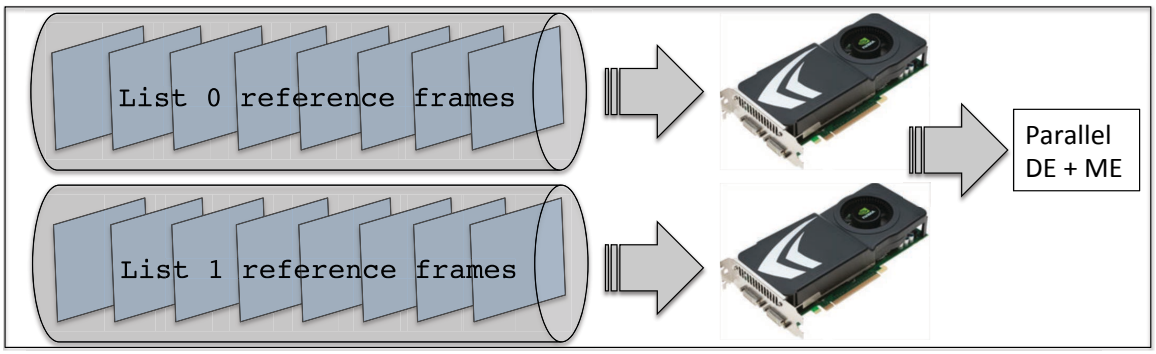


Figure 2.5: Dual GPU ME/DE solution.

Table 2.2
Encoder Setting and Experimental Conditions

Software	JMVC8.5 (executed on a single-thread, single-core), Prediction structure: IBP, GOPSize: 8, NumberReferenceFrames: 1, FrameRate: 25, FramesTobeEncoded: 33, SymbolMode: CABAC, BiPredIter: 2, IterSearchRange: 1, TZsearch: Raster search step size (L_{step}): 3, Algorithms: Star Search, Raster Search and Star Refinement, QP parameter $\in$ {24, 28, 32, 37}, Test video Sequence (480p)$\in$ {Ballroom, Exit, Vassar, Race1}, ME/DE search range [-128,127]/[-128,+127](horizontal/vertical)
Hardware	Dual NVIDIA Fermi™ C2075 SLI with 5 GB GDDR5, Intel® Xeon® 6-core CPU x5650 @ 2.67GHz with 50 GB DDR3

saturates the GPU resources and yields no performance advantage. GPUfull, on the other hand, processes one macroblock at a time. However, GPUfull, with efficient implementation of its two kernels, where the number of threads per thread-blocks and thread-blocks in the grid are no more than 256, outperforms the other schemes that focus on parallel processing of huge number of macroblocks, by a minimum factor of 10.

Table 2.3
Encoding Results for TZsearch, CPU-based Full Search, and GPU-based Full Search (GPUfull)

View	Search Scheme	Ballroom												Vassar											
		Full search				TZ search				GPUfull (2-GPU|1-GPU)				Full search				TZ search				GPUfull (2-GPU|1-GPU)			
0	QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
	PSNR	39.852	38.058	36.258	34.117	39.848	38.055	36.257	34.114	39.848	38.054	36.259	34.107	39.324	37.941	36.665	35.155	39.323	37.940	36.665	35.153	39.323	37.940	36.665	35.153
	Bit-Rate	1552.7	962.6	598.3	344.2	1555.1	962.9	598.8	344.2	1554.9	961.9	599.4	344.4	1135.9	570.1	305.0	160.1	1136.7	570.0	305.2	160.1	1135.9	570.0	305.1	160.0
	Time	11646	11606	11640	11637	199	188	182	169	91|145	91|145	91|145	90|144	11533	11555	11632	11539	83	88	76	75	92|146	90|145	90|145	90|145
1	QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
	PSNR	39.655	38.049	36.288	34.047	39.656	38.042	36.276	34.042	39.659	38.045	36.288	34.025	39.221	37.915	36.569	34.837	39.216	37.897	36.554	34.819	39.216	37.908	36.569	34.828
	Bit-Rate	1157.5	620.6	346.2	180.6	1155.9	619.8	346.1	180.9	1159.3	621.3	347.0	180.0	995.4	411.5	181.0	74.3	996.7	414.2	182.6	73.9	993.6	414.9	183.1	74.2
	Time	25545	25317	25166	25411	1281	1204	1106	954	169|280	169|280	168|279	168|279	25198	25125	25545	25472	991	813	667	522	169|280	168|280	167|278	167|278
2	QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
	PSNR	39.994	38.313	36.499	34.217	39.987	38.315	36.485	34.188	39.987	38.313	36.487	34.201	39.644	38.331	36.975	35.215	39.639	38.325	36.948	35.178	39.639	38.324	36.975	35.216
	Bit-Rate	1334.6	764.7	441.2	234.7	1336.3	765.5	442.1	234.7	1335.5	766.9	442.7	235.0	965.19	425.44	201.87	86.27	967.9	427.6	203.2	86.0	966.6	427.0	202.4	87.0
	Time	12679	12553	12666	12736	240	229	213	196	99|149	99|149	98|148	98|148	12705	12653	12743	12655	132	108	96	86	99|149	98|148	98|148	97|147
3	QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
	PSNR	39.771	37.958	36.089	33.877	39.769	37.955	36.068	33.860	39.767	37.966	36.079	33.883	39.636	38.319	37.025	35.330	39.636	38.308	37.020	35.317	39.634	38.329	37.032	35.334
	Bit-Rate	1123.7	591.4	319.5	168.4	1123.1	591.7	320.9	169.1	1125.2	592.3	320.5	168.9	875.6	361.4	161.2	66.6	875.9	359.5	162.9	67.2	875.1	362.6	162.8	66.7
	Time	25518	25299	25157	25431	2031	1153	1061	926	170|278	168|278	168|277	167|277	25278	25119	25337	25371	914	757	625	495	168 |278	168|277	167|276	166|276
4	QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
	PSNR	39.647	37.803	35.903	33.554	39.642	37.793	35.876	33.524	39.637	37.791	35.882	33.548	39.550	38.113	36.673	34.823	39.550	38.107	36.649	34.794	39.551	38.114	36.647	34.820
	Bit-Rate	1492.4	853.5	499.3	264.8	1498.1	855.6	497.6	266.9	1494.2	853.6	497.3	268.2	1030.9	477.7	227.1	98.9	1030.8	476.3	228.8	98.3	1030.5	474.9	226.8	99.4
	Time	12635	12581	12609	12708	241	229	217	206	100|147	99|147	98|146	98|146	12635	12761	12609	12708	125	109	98	87	99|146	98|145	98|145	97|145
5	QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
	PSNR	40.272	38.478	36.598	34.241	40.260	38.465	36.576	34.221	40.268	38.467	36.591	34.245	40.480	39.210	37.831	35.887	40.488	39.194	37.829	35.844	40.489	39.201	37.826	35.864
	Bit-Rate	1076.6	583.4	340.7	196.2	1077.8	585.1	341.7	196.5	1079.6	585.4	340.8	196.2	746.3	346.6	172.0	78.9	746.7	345.7	173.3	78.8	745.4	345.7	172.7	78.3
	Time	25464	25261	25150	25434	1271	1196	1101	961	169|277	169|277	168|276	168|276	25164	25141	25210	25374	1091	924	774	610	169|278	168|277	167|276	166|276
6	QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
	PSNR	39.315	37.772	36.037	33.758	39.307	37.765	36.039	33.714	39.310	37.770	36.033	33.741	38.944	37.675	36.560	34.740	38.941	37.660	36.422	34.720	38.945	37.668	36.422	34.737
	Bit-Rate	1546.2	858.7	499.8	275.2	1548.3	861.2	504.4	275.9	1548.4	861.1	503.1	275.9	1209.3	527.1	254.6	114.4	1215.4	529.2	257.3	114.5	1210.8	525.5	255.0	115.2
	Time	12628	12595	12710	12739	261	243	231	216	99|148	99|147	99|147	98|146	12628	12725	12649	12739	147	122	106	93	99|146	98|145	97|145	98|145
7	QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
	PSNR	39.484	37.604	35.552	33.071	39.484	37.595	35.524	33.048	39.483	37.591	35.543	33.050	39.543	38.136	36.736	34.833	39.536	38.140	36.720	34.810	39.539	38.137	36.719	34.823
	Bit-Rate	1637.4	934.1	539.8	282.4	1640.1	935.3	539.6	284.1	1641.1	935.9	542.0	286.1	1079.7	529.8	266.8	110.9	1079.1	531.4	266.0	109.8	1078.5	531.1	266.0	111.2
	Time	12635	12559	12541	12760	254	238	226	215	100|148	99|148	98|147	98|147	12695	12739	12660	12700	138	120	106	92	100|146	99|145	98|145	97|145

PSNR is expressed in dB, Bit-rate in kilo-bits/secound (kbs), and Time in seconds.

2.3.4 Multi-GPU Implementation

The discussion for the full search, so far, has been with respect to a single reference frame. As seen from Fig. 3.2, in MVC all B-frames have multiple reference frames. So it is possible to perform the ME/DE of a macroblock across multiple reference frames simultaneously using multiple GPUs. With respect to the current frame, reference frames in the forward and backward directions (both in the temporal and the inter-view sense), are listed in *list0* and *list1*, respectively. In the experimentation two GPUs have been employed to process frames from two lists in parallel (Fig. 2.5). The exchange of data between the CPU and GPUs are performed in parallel using the CUDA asynchronous memory transfers. ME/DE in two lists are carried out simultaneously following the GPUfull search described earlier. This results in a speedup of about two in the ME/DE estimation part of the MVC. As seen from Fig. 3.2 (c), for view 1, the number of reference frames is four. More efficient MVC schemes require even more reference frames. Therefore, with the availability of more GPUs, the performance of ME/DE can be scaled up proportionally by processing several reference frames in parallel.

The use of multi-GPU processing is not limited to block matching in ME/DE in a single macroblock with respect to two reference frames in *list0* and *list1*. It can also be applied to block matching in ME/DE for multiple macroblocks in the same frame in parallel, provided coding dependencies are resolved. One approach to overcome the coding dependencies among the macroblocks in a frame is the concept of diagonal wavefront processing, proposed in [43]. This approach may even require out-of-order processing of macroblocks from different slices, when such slices contain few rows of macroblocks, as required for improved error-resiliency [3]. In our experience, the wavefront processing is likely to result in extra scheduling overhead in the non-GPU part of the MVC process.

2.3.5 Performance Analysis for GPU-based Full Search

Table 2.2 presents the experimental conditions: parameters for MVC software coding tools and CPU/GPU hardware platforms. The BiPredIter and iterSearchRange parameters in Table 2.2 relate to the iterative bi-prediction process. JMVC uses a highly efficient iterative scheme for bi-prediction. Based on the $(J_{min,p}, MV_{min,p})$ pairs found in each direction, subsequent bi-directional search using a very small SR is carried out, iteratively, back and forth between two reference frames from the two lists, *list0* and *list1*. This improves the coding efficiency by ensuring that the best linear combination of two predictions are used [36]. Parameters iterSearchRange and BiPredIter, respectively, specify the value of SR and number of iterations for bi-prediction. To save the execution time, the number of frames to be encoded (FramesTobeEncoded) is chosen such that it exactly contains four GOPs, but can be set to any value. Four video sequences are carefully chosen, with Ballroom [44] and Race1 [45] describing highly dynamic scenes that contain complex features, Vassar [44] a more static scene, and Exit [44] in between. In this chapter the full results for only Ballroom and Vassar, and comparative results for all four video sequences are shown. The metric for evaluating encoding performance proposed in [1] is adopted for the comparison of various techniques. Encoding is performed for eight views but can be expanded to more.

Table 2.3 shows the performance results for the proposed GPUfull, CPU-based full search, and the state-of-the-art CPU-based TZsearch fast search algorithm. Data is presented for two video sequences, four quantization parameters (QPs), and eight views. It should be noted that there are negligible differences in PSNR and bit-rate values between GPUfull, TZsearch and the CPU-based full search for all views and QPs. On the other hand, there are large differences in the execution times between the three schemes. It should also be noted that the variations in the execution times

for CPU-based and GPUfull across two video sequences are relatively small. This is as expected, since these two algorithms search every location in the search region. On the other hand, the corresponding variations in search times for TZsearch are wide. This is due to the non-deterministic nature of TZsearch algorithm where the search time depends on the complexity of the video sequence and the QP value used. The reason for non-deterministic execution time of TZsearch becomes apparent in the next section.

One single most important observation about Table 2.3 is that the speedup for views with bi-directional inter-view prediction, (views 1, 3, and 5 in Fig. 3.2) is up to 3.5 times higher than the speedup for the other views. This suggests that with more complex prediction structures, with larger number of reference frames, GPUfull will perform even better than TZsearch.

Tables 2.4 and 2.5 summarize the data in Table 2.3 by averaging over eight views. They also include the summary of the results for two other video sequences, Exit and Race1. As seen from Table 2.4, for all video sequences, GPUfull on a single GPU performs about 87 to 90 times better than the CPU-based full search for the QPs values ranging from 24 to 37. It should be noted that for both schemes, being full search, as expected, the speedup remains relatively constant across all QP values and video sequences tried. From Table 2.5 it can be observed that the speedup of the GPUfull over the TZsearch covers a wide range of 1.3 to 3.9 over the same range of QPs values and video sequences. This is due to the sensitivity of the TZsearch execution time to QP and video content. It should also be noted that speedup factors for the video sequence with the least dynamic scene, Vassar, are the lowest among the four video sequences. This is due to the fact that non-deterministic TZsearch terminates fast for video sequences that are more static. Also note that for all cases considered, 2-GPU implementation improves the performance by an additional factor of 1.6.

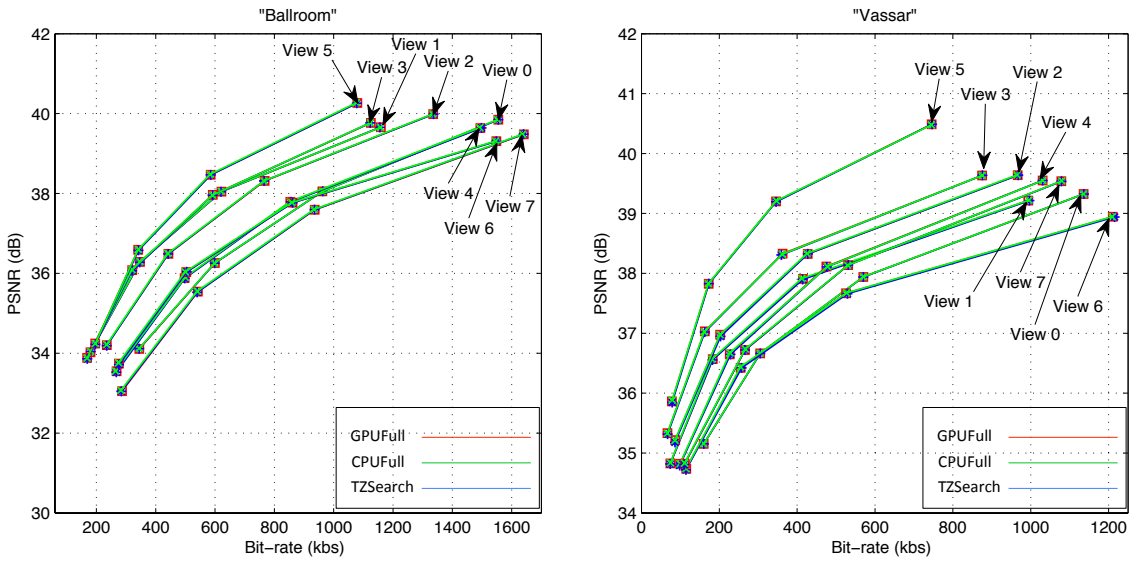


Figure 2.6: Rate-distortion performance for multiple views for CPU-based full search (CPUfull), TZsearch and GPUfull.

Table 2.4
Comparison of GPUfull to CPU-based Full Search Averaged Over Eight Views

	Ballroom				**Race1**				**Exit**				**Vassar**			
QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
DPSNR (dB)	-0.0216				-0.037				-0.0263				-0.0113			
DBR (%)	+0.5356				+0.53				+0.8681				+0.5256			
Speedup (1-GPU)	88	88	88	89	89	89	89	90	88	88	89	89	88	88	88	89
Speedup (2-GPU)	139	139	139	141	138	138	139	141	140	140	141	147	139	140	141	142

Table 2.5
Comparison of GPUfull to TZsearch Averaged Over Eight Views

	Ballroom				**Race1**				**Exit**				**Vassar**			
QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
DPSNR (dB)	+0.0057				0				-0.0104				+0.0085			
DBR (%)	-0.1589				0				+0.2542				-0.3325			
Speedup (1-GPU)	3.7	3.0	2.8	2.5	3.9	3.6	3.4	3.0	3.0	2.7	2.5	2.0	2.3	1.9	1.6	1.3
Speedup (2-GPU)	5.8	4.7	4.4	3.9	6.0	5.6	5.3	4.7	4.8	4.3	4.0	3.3	3.6	3.1	2.6	2.1

RD performance plots for Ballroom and Vassar video sequences, for three search algorithms, and for eight views are shown in Fig. 2.6. The CPU-based full search is used as a reference, as it searches every location in the search region, and uses the MV/DV predictor at partitions smaller than 16×16 and, therefore, stands for

the highest RD performance achievable. As seen from the figure, for all views, the plots for all three algorithms completely overlap each other. When compared with CPU-based full search, on average, for the Ballroom video sequence, the GPUfull loses only 0.02 dB in PSNR with an increase in bit-rate of 0.5%. The degradations for the Race1, Vassar, and Exit video sequences are similar. Minor degradation in RD performance results from the breaking of the dependencies in MVP_p values, where all smaller partitions inherit the MVP for 16×16 partition. A no less important characteristic of plots in Fig. 2.6 is that the views with more reference frames (view 1 and 3, and 5 in Fig. 3.2) generally have better RD performance.

The implementation of GPUfull exposes the potential of MPA for ME/DE in MVC. This implementation relies on some innovative techniques such as dynamic programming and parallel multi-vector reduction, and combines them with efficient use of GPU resources. The biggest advantage of GPUfull is its fixed execution time, independent of the type of video sequences, an important design factor in real-time systems with a hard deadline. However, GPUfull is essentially a straightforward mapping of the full search algorithm into a parallel architecture. Nevertheless, GPUfull is used as the basis for the development and evaluation of an innovative parallel fast search ME/DE algorithm in the next section.

It should also be noted that while over eight views, GPUfull has a better overall performance than TZsearch, it does not perform faster for every view. For example, consider the execution times for the Vassar video sequence in Table 2.3, a more static scene where TZsearch converges fast. For certain individual views and higher QP values, the execution time of TZsearch is marginally better than GPUfull search. The reason for this is that MPAs, while having hundreds of computing cores, their resources (cores, shared memory, registers, cache, memory bandwidth), are limited. Due to resource limitations, the number of resident (active) thread-blocks on the fly, at any given time, on a GPU multiprocessors is only a fraction of total number of

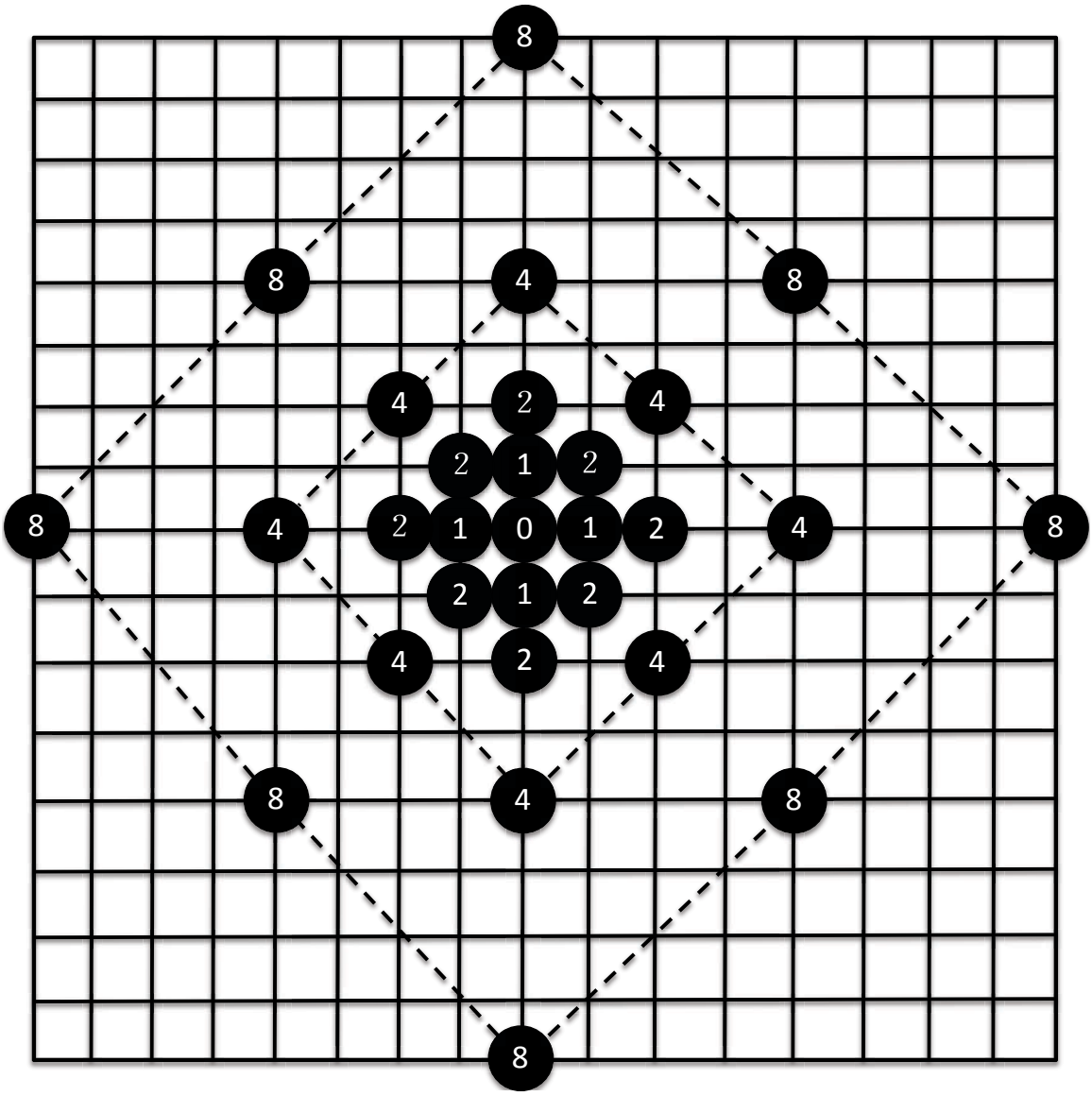


Figure 2.7: Star search pattern

thread-blocks processed by the scheduler. Therefore, when there is a large number of thread-blocks they are scheduled on the multiprocessor cores in a batch-sequential (warp-sequential) manner. As macroblocks are assigned to thread-blocks for execution on the GPU, by the same token, there is little gain in flooding the GPU resources through the processing of multiple macroblocks in parallel.

2.4 Massively Parallel Fast Search using Motion Vector Predictors

From the foregoing discussion, it should be noted that to improve the execution time performance of GPUfull requires reducing the number of thread-blocks significantly. This, in turn, requires developing a parallel algorithm that abandons GPUfull for a faster search that has little effect on PSNR or bit-rate.

The work in [15] reduces the number of search points through a sub-optimum search scheme. However, the search is rather arbitrarily skewed against the motion in the vertical direction by searching points in only 13 rows in a non-square search region of 128×25. Further, the number of points searched is fixed, independent of the video sequence. However, as the results in Table 2.5 for TZsearch shows, the search time (and, therefore, the number of search points) for a sub-optimum scheme strongly depends on the video content. Failing to account for the video content yields a degraded bit-rate and PSNR for complex video sequences with dynamic scenes. This is specially noticeable for inter-view references, where number of searches for TZsearch are significantly higher. From Table 2.5, the execution times for views 1, 3, and 5 are about six times larger than the other views for TZsearch, whereas for CPU-based and GPUfull they are only twice larger.

The efforts presented here in the development of the parallel fast search is inspired by approach in sequential fast algorithms, such as TZsearch, to reduce the number of search points. To this end an analysis of TZsearch is provided first.

2.4.1 Analysis of TZsearch

TZsearch scheme is a highly accurate and efficient ME/DE algorithm, and due to its outstanding performance, has been implemented in JMVC [27] and HM [28]. TZsearch is also a highly configurable search scheme that offers the freedom to fine tune the algorithm. To simplify the analysis of TZsearch, the general configuration which is also the default setting for JMVC is considered. The TZsearch process can be described as a star search with conditional raster search. Four of the steps involved in TZsearch are described below.

i) *Test near motion vectors*: This step moves the search center in the search region to the most probable location from among the candidate vectors: MV (0,0) (no displacement), MV_A, MV_B, MV_C, and MVP_p (see Sec. 2.2). Among these five candidates, the best MV (called *neighbor predictor* (NP)), in terms of lowest cost from (2.1), is chosen as the search center. This simple technique incurs little computation cost, as it involves the evaluation of the cost function in (2.1) for only five MVs. It is an effective technique, as candidate MVs are drawn from the neighboring partitions that usually have high correlation with the MV of the current partition under consideration.

ii) *Star search*: Typically fast search algorithms use diamond as the search pattern (square and hexagon are also used). Only the points on the vertices, middle of the edges, and the center of the diamond (search center) are included in the search. This search is repeated in an iterative fashion by moving the center of the search pattern to the point with minimum cost according to (2.1). The process stops when the center of the diamond has the lowest cost. The common drawback of the diamond algorithm is that it often settles in a local minimum. The star tries to avoid this problem by using several concentric diamonds with different step

sizes (Fig. 2.7). In the star search, the step sizes grow exponentially in powers of two (*i.e.* $\{1, 2, 4, ..., SR\}$). As seen from Fig. 2.7 the number of points that are searched in the star algorithm is $8log_2SR - 4$, which is significantly lower than the full search. Also note that most search points are clustered around the center of the star. For a static scene (*e.g.* Vassar) the TZsearch typically ends with the star search step. This is a significant factor in the lower execution time of TZsearch compared with GPUfull in Table 2.3.

iii) *Raster refinement search*: Star search is a rather coarse global search. Therefore, the minimum search point, if located away from the center, has a highly diminished probability of being close to the global minimum. Therefore, in the event that the location of the minimum point is at a distance above a threshold value, the star search is considered unsuccessful. Instead a finer grain raster refinement search is performed. Similar to full search, raster refinement covers the whole search region, but only searches every L_{step} horizontally and vertically. Hence, the computational complexity of raster refinement is only $1/(L_{step})^2$ of the full search. The raster refinement minimum point is denoted as *global predictor* (GP), as opposed to NP mentioned in the first step. Raster refinement incurs additional execution time, resulting in a non-deterministic encoding time for TZsearch.

iv) *Star refinement*: Since raster search skips many points, an iterative star refinement search is performed to enhance the result. Star refinement starts with a center that is the minimum search point found by the raster refinement step. Next, like diamond search, the center of star search pattern moves to the next best point. The process terminates when the best search point is evaluated at the center of the star. This is another factor contributing to the non-deterministic encoding time of TZsearch.

2.4.2 Decision Zone-based Fast Parallel Search

From the foregoing discussion it can be concluded that for a static scene, the global minimum is likely to be located in the vicinity of the origin of the search region. As a scene becomes more dynamic, the location of the global minimum becomes significantly less certain, and has a diminished probability of being found by the star search that allocates more search points around the origin of the search region. Therefore, for a dynamic scene what is needed is an algorithm that allocates search points uniformly within the search region.

2.4.2.1 Star versus raster search

Consider a search region with dimension $\pm SR$. As SR increases, it is easy to see that the number of search points covered by raster search increases much faster compared with the number of points for the star search. The break-even value of SR for a given value of L_{step} can be found by solving for SR in $8log_2SR - 4 = (2SR/L_{step})^2$. For $L_{step} = 3$ (the default value for TZsearch in JMVC), $4 < SR < 8$ is obtained. Therefore, it is obvious that even for moderate value of $L_{step} = 3$, the size of the search region where star search becomes less effective than the raster search, is rather small. So, for a dynamic scene, a raster based algorithm is needed to cover more search points, albeit, generally at a higher computational cost.

2.4.2.2 Cost maps

Since for every search point in the search region there is a cost, a cost map can be developed. Fig. 2.8 presents the cost maps for four different macroblocks for the Ballroom video sequence, for a partition size of 16×16. These maps identify several cost regions. It should be noted that dark blue regions with the lowest cost cover large areas in the search region. This suggests that the motion variations in all four macroblocks are relatively complex. The mapping for typical macroblock from a static scene has its dark blue regions coalesced around the origin. It should also be noted that the global minimum (marked by green arrow) resides in one of the lowest cost regions that are shaded dark blue (called region of interest (ROI)). The red arrow, on the other hand, identifies the raster search minimum for $L_{step} = 8$. The probability of the raster search finding the global minimum in the dark blue area diminishes with $((1/Lstep)^2)$.

2.4.2.3 Fast decision zone-based search

It is evident that to find the global minimum, first, the ROI from all the regions with the lowest cost using a coarse search needs to be identified. Once the ROI is identified a refinement is performed to search for the global minimum. From the discussion on the analysis of TZsearch, recall that NP was used as the center of star search and GP (obtained through raster search) was used for the center of the star refinement search. In the coarse part of the algorithm the use of NP and GP are combined to find the best MV/DV that has a high probability of being located within the ROI.

As it will be shown later for great majority of macroblocks, GP and NP have identical costs according to (2.1), albeit different MV/DV values. NP is chosen as default, as

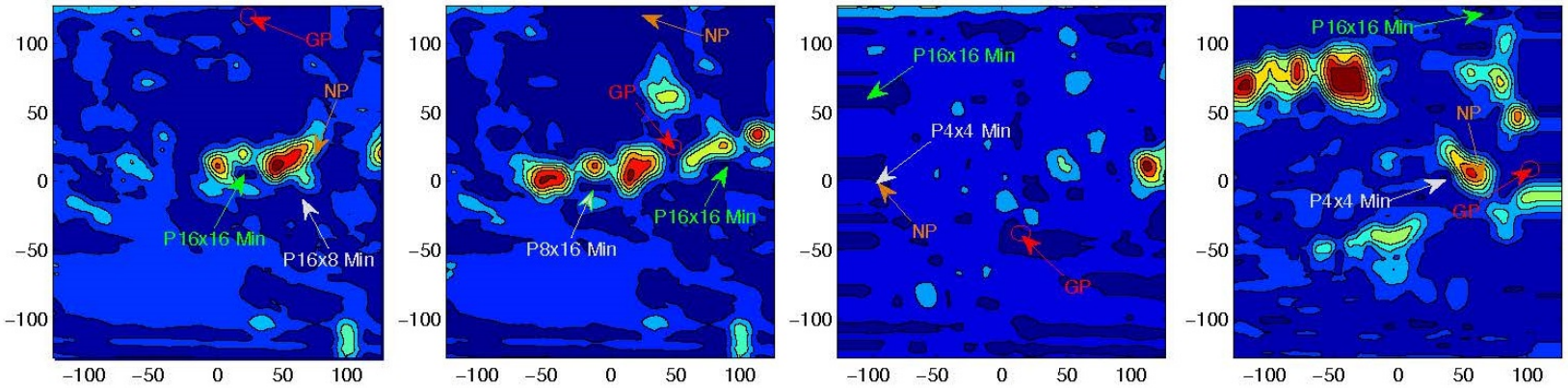


Figure 2.8: Cost maps for typical macroblocks from the Ballroom video sequence, for a partition size of 16 × 16. Red and green arrows identify raster search minima (GP) for $L_{step} = 8$, and global minima (P16×16 Min), respectively. Also shown on the maps are the locations for the global minima for 16×8 (P16×8 Min), 8×16 (P8×16 Min) and 4×4 (P4×4 Min) partitions, identified by white arrows. Orange color arrows identify the location of neighbor predictors (NP).

it correlates better with the neighbor macroblocks. However, for significant number of cases GP provides a MV/DV with the lowest cost. Maps in Fig. 2.8 are four examples of such cases.

For the refinement search this work uses the concept of *decision zone*. Fig. 2.9 illustrates the partitioning of the search region into multiple non-overlapping decision zones. *ZoneSize* defines the dimensions of the inner zone, as well as increments to the outer boundary of the next zone. The zone is selected to include the location of MV/DV obtained through the coarse search (NP or GP). Corresponding to each decision zone, SR is set for the refinement full search as shown in Fig. 2.9. With the aim of fully encompassing ROI within the search region, the value of SR is chosen sufficiently larger than the dimension of the decision zone selected. This provides a high probability of finding the global minimum within the ROI. Fig. 2.9 also illustrates the inclusion of MV/DV and ROI in the decision zone and in its corresponding search region, respectively.

On an MPA computing platform, the raster search routine to find GP requires only a minor modification of GPUfull discussed in the previous section. Further, GPUfull is used as the refinement search algorithm with appropriate selection of parameters $ZoneSize$ and SR. The phases involved in the proposed decision zone-based search (DZfast) algorithm are,

Phase i *Raster search - GP*: perform GPU-accelerated raster search with a step size of L_{step} to find GP.

Phase ii *NP/GP selection*: select the best MV/DV out of GP and five neighbor motion vectors ($MV_{(0,0)}$, MV_A, MV_B, MV_C, and MVP_p).

Phase iii *Decision phase*: select the decision zone that encompasses the best MV/DV. Set the corresponding SR for the refinement search.

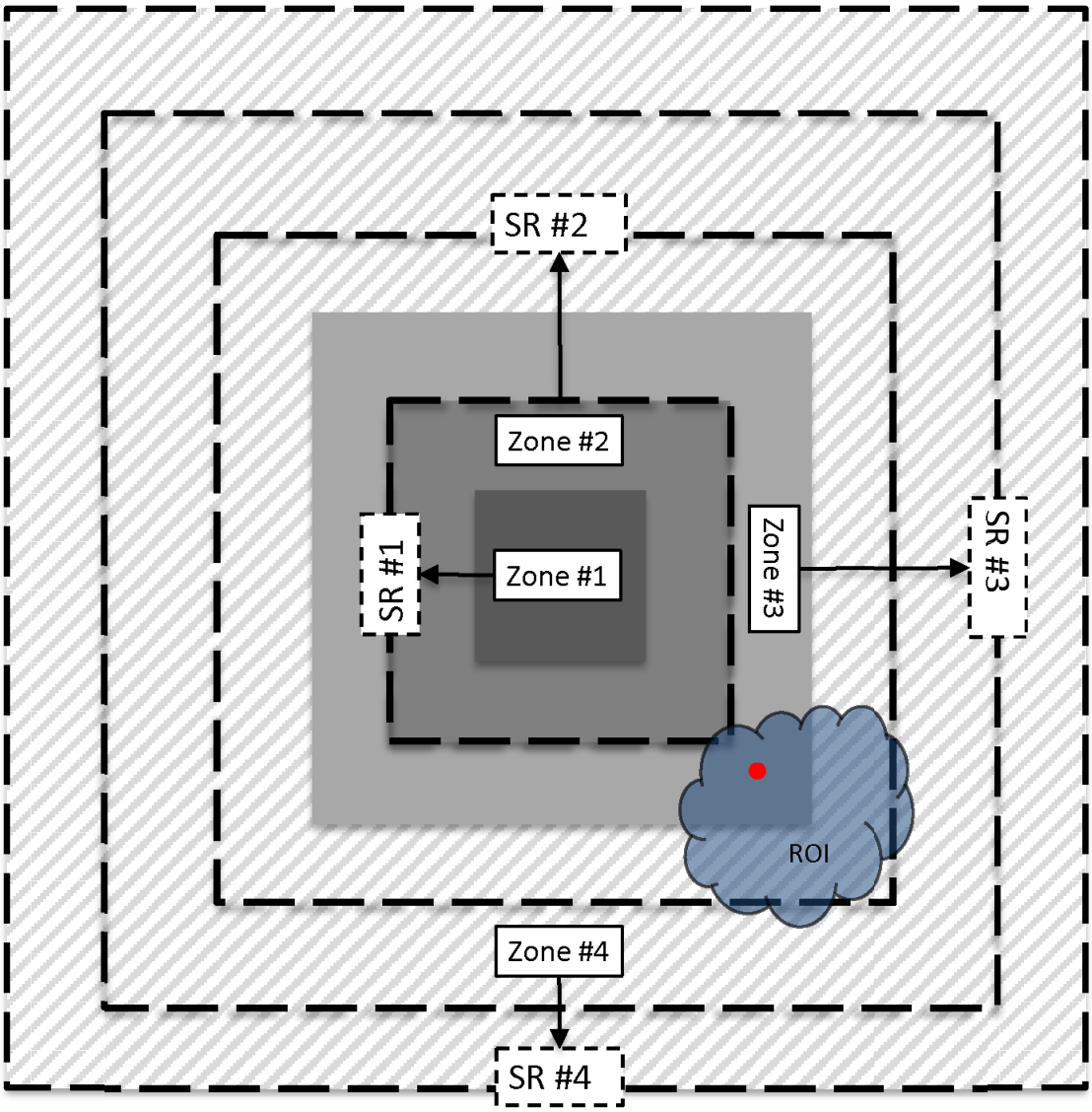


Figure 2.9: Decision zone layout for DZfast.

Phase iv *Refinement*: perform the refinement search using GPUfull.

The computational complexity of raster search for finding GP $((2*SR/L_{step})^2)$ reduces as inverse square of L_{step}. However, the performance of DZfast in terms of bit-rate and PSNR is insensitive to L_{step} for range of values between two to eight, for the range of video sequences tried. $L_{step} = 8$ is used for the experimentation.

It should also be noted that performance of GPUfull, in terms of bit-rate and PSNR, is insensitive to the exact choice of SR. Therefore, to suit the architecture of the GPU and the memory block alignment, $SR \in \{\pm 32, \pm 64, \pm 96, \pm 128\}$ is chosen. By the same token, the performance of the algorithm is affected very little by the exact choice of $ZoneSize$. Therefore, the value of $ZoneSize = \frac{1}{2}\min(SR)$ is conveniently chosen.

To reduce the execution time, partition size of 16×16 for Phases i to iii of the DZfast algorithm is chosen. As will be shown later it has little impact on the coding quality or the bit-rate. However, the refinement Phase iv proceeds as GPUfull search in parallel for 41 partitions.

The execution time for DZfast can be modeled as,

$$T_{total} = P_{32}T_{32} + P_{64}T_{64} + P_{96}T_{96} + P_{128}T_{128} + T_{GP,NP} \tag{2.3}$$

where T_{SR} is the GPU execution time of the refinement search for $SR \in \{\pm 32, \pm 64, \pm 96, \pm 128\}$ and $T_{GP,NP}$ the execution time of the combined GP and NP coarse search. These values are the characteristic of the MPA platform. P_{SR} represents the probability of selection of a search region with a given SR value. P_{SR} depends on the statistics of the video sequence under consideration. Due to resource limitation, there is a upper limit on the rate of parallel processing (represented as number of locations searched per second) that can be sustained on an MPA. On the GPU test platform used in this work, for small values of $SR < 96$ the parallel processing rate remains well below that limit, and therefore, the execution time of T_{32} and T_{64} remain relatively low and close to each other. For $SR = \pm 96$ the processing rate begins to saturate. For $SR = \pm 128$ the processing rate is fully saturates, and any additional workload has to be serviced in a batch-sequential manner. Fig. 2.10

illustrates the effect of the limitation of GPU resources on the saturation of processing rate with increase in SR for the first three views. Saturation of processing rate results in a sharp increase in the execution time of T_{128} as shown in Fig. 2.11. Also, note that the execution time follows the view complexity in the prediction structure of Fig. 3.2 (c), with view 1 having twice the number of reference frames for view 0, requiring twice the execution time. The red line in the figure marks the baseline time which is the total execution time excluding the refinement search part of DZfast. It is easy to see that any advantage of DZfast over GPUfull (that uses $SR = 128$) comes from the relative magnitude of P_{SR} values.

2.4.2.4 Discussion on DZfast

Note that in DZfast the origin is used as the center of the search rather the best MV/DV found in the coarse search, as it is typical in the fast search algorithms such as TZsearch. To explain this choice, the following discussion is in order.

Intuitively, centering the refinement search around the MV/DV obtained from the coarse search should result in a smaller SR. However, this is of little concern on the MPA computing platform of GPU, as long as the SR remains below the level that saturates the resources. Therefore, moving the center to the origin does not increase the execution time significantly for the range of SR values used in DZfast. In addition, GPU memory is organized for high bandwidth memory transfer. However, it requires memory access to be aligned at certain block sizes. Centering the search around the origin with the chosen SR values will ensure efficient memory alignment.

Further, as was said before, the coarse search is restricted to the partition size of 16×16, and use the result for the other smaller partitions. Smaller partitions, however, can have values of GP, NP and global minima that are significantly different from that

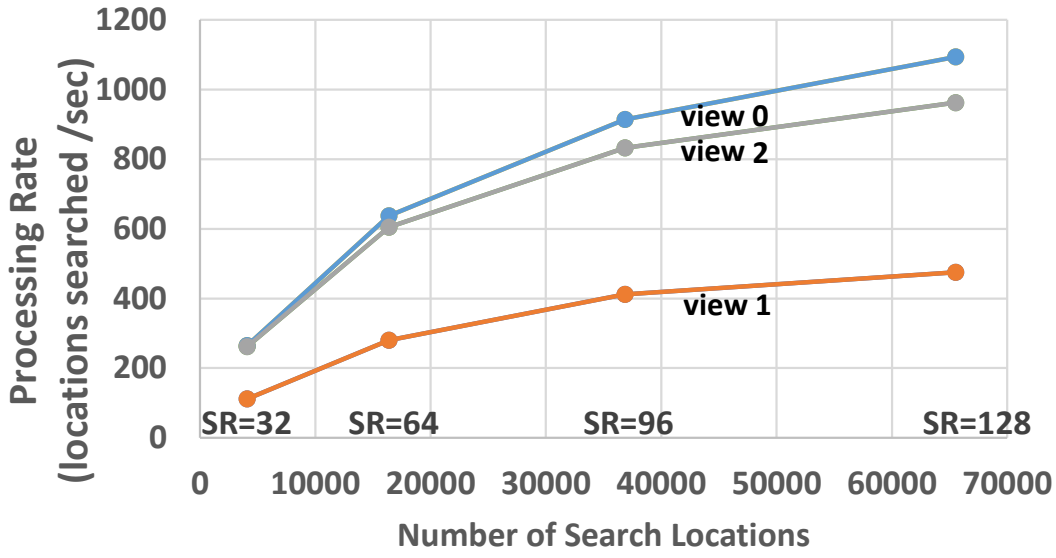


Figure 2.10: Processing rate for the refinement search part of DZfast, for various search ranges for the first three views.

of 16×16. In Fig. 2.8 white arrows show the location of the global minima for 16×8, 16×16 and 4×4 partitions. As seen, setting the search center at the MV/DV for GP for 16×16 is likely to fail to cover these points. This is a definite drawback of solution in [29] where the small search region with $SR = \pm 16$ is centered around MV/DV from NP. So centering the search around the MV/DV obtained from the combined GP/NP coarse search does not guarantee to find the ROI for every partition. Centering the search around the origin and setting the region for the refinement search sufficiently large, as it is done in DZfast, will have a high likelihood of enclosing the ROI for any partition size.

2.5 Performance Analysis of DZfast

For the experimental set up the DZfast parameters are configured as $L_{step} = 8$ (a good compromise between search time and accuracy), $ZoneSize = \pm 16$, and $SR \in$

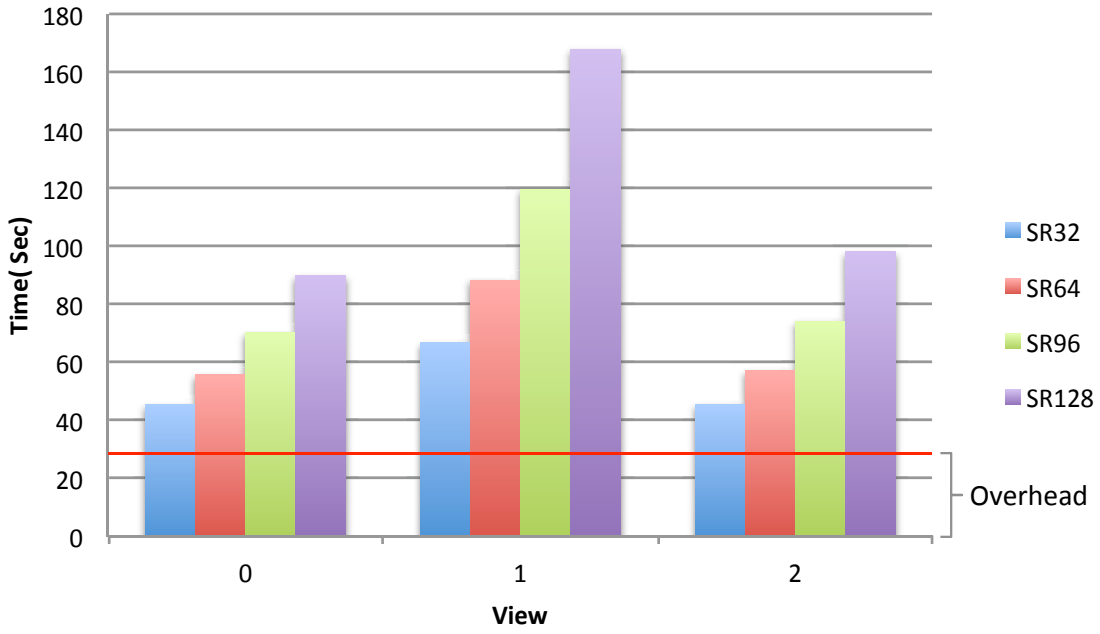


Figure 2.11: Execution time for various search ranges for GPUfull. The red line marks the baseline time that includes all timing components other than refinement search part of DZfast.

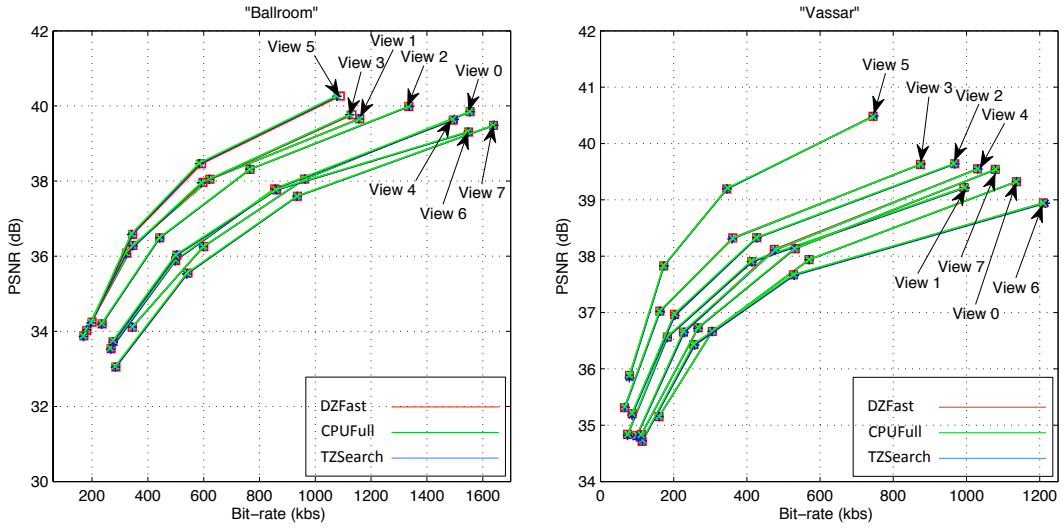


Figure 2.12: Rate-distortion performance for multiple views for DZfast, TZsearch and CPU-based full search (CPUfull).

Table 2.6
Percentage Distribution for Global Predictor (GP), Neighbor Predictor (NP), and Decision Zones for DZfast

Percentage Distribution														
	Ballroom							Vassar						
View	NP	NP=GP	GP	Zone #1	Zone #2	Zone #3	Zone #4	NP	NP=GP	GP	Zone #1	Zone #2	Zone #3	Zone #4
0	12.92	78.43	8.66	81.44	5.16	2.38	11.01	1.32	97.75	0.94	98.85	0.58	0.20	0.37
1	50.56	39.48	9.96	54.37	12.14	7.82	25.67	38.51	57.30	4.19	70.64	9.57	5.47	14.33
2	16.88	70.28	12.84	74.06	6.28	3.30	16.36	5.85	89.62	4.53	90.93	1.99	0.99	6.09
3	47.11	40.84	12.06	60.44	12.96	9.51	17.09	36.21	58.95	4.84	74.46	10.31	4.64	10.59
4	17.32	68.50	14.18	73.07	7.23	4.23	15.48	5.92	89.82	4.26	91.03	1.92	1.10	5.95
5	49.36	38.81	11.84	57.97	14.28	8.85	18.90	43.44	53.48	3.08	66.33	11.87	6.61	15.20
6	18.81	67.86	13.33	73.12	7.66	4.09	15.14	2.77	93.74	3.49	90.75	2.02	1.25	5.99
7	18.02	68.40	13.58	74.28	9.58	4.98	11.17	6.36	90.02	3.62	91.73	2.23	1.74	4.31

$\{\pm 32, \pm 64, \pm 96, \pm 128\}$. The rest of the parameters are listed in Table 2.2.

Table 2.6 presents the statistical distribution of MV/DV at the end of coarse search, between GP and NP, where a great majority of the cases are contributed from the NP pool. The majority of NP cases, however, belong to the GP=NP category, where the cost of NP and GP from (2.1) are the same, albeit having different MV/DV. These cases default to NP as it has better correlation with neighbor macroblocks. For Vassar video sequence, over 95% of MV/DVs come from NP or NP=GP. For Ballroom this value reduces to 86%, a direct result of complex nature of this video sequence. Table 2.6 also presents the distribution of the best MV/DV from the coarse search, among four decision zones. For all views both in Vassar and Ballroom, a majority of partitions fall in first decision zone. However, for the Ballroom video sequence, there is a significant leakage from the decision zone #1 to the other decision zones, with most going to zone #4 with $SR = \pm 128$. This results in 20% to 25% increase in the execution time for the Ballroom over the Vassar as shown in Table 2.7, and expected from (2.3). This is inline with the results for TZsearch in Table 2.3.

It is also noted that in Table 2.6, views 1, 3, and 5 exhibit a drastic redistribution

Table 2.7
Encoding Results for the DZfast

View		Ballroom				Vassar			
	QP	24	28	32	37	24	28	32	37
	PSNR	39.8477	38.0545	36.2573	34.109	39.3229	37.9389	36.6639	35.1534
0	**Bit-Rate**	1554.4	961.9	599.9	344.2	1136.3	569.8	305.3	160.1
	Time								
	2-GPU|1-GPU	61|71	58|70	59|59	58|58	51|58	50|56	50|56	49|56
	QP	24	28	32	37	24	28	32	37
	PSNR	39.6578	38.0512	36.2872	34.0225	39.2193	37.9045	36.5657	34.8382
1	**Bit-Rate**	1159.2	622.2	346.9	180.8	992.8	413.2	182.9	74.1
	Time								
	2-GPU|1-GPU	123|154	121|153	120|151	117|147	110|134	106|129	101|122	95|115
	QP	24	28	32	37	24	28	32	37
	PSNR	39.9881	38.3115	36.4902	34.1987	39.6418	38.3274	36.9692	35.2061
2	**Bit-Rate**	1335.3	766.1	442.9	235.8	967.7	426.8	202.4	86.5
	Time								
	2-GPU|1-GPU	68|79	67|78	67|77	66|75	58|64	57|63	57|63	57|63
	QP	24	28	32	37	24	28	32	37
	PSNR	39.763	37.966	36.082	33.881	39.630	38.326	37.024	35.313
3	**Bit-Rate**	1130.9	597.6	323.806	170.770	874.3	361.3	162.1	65.6
	Time								
	2-GPU|1-GPU	116|147	114|145	113|143	109|138	107|129	103|125	98|118	92|112
	QP	24	28	32	37	24	28	32	37
	PSNR	39.624	37.792	35.884	33.539	39.551	38.122	36.651	34.818
4	**Bit-Rate**	1494.3	854.3	499.5	268.5	1030.2	474.8	226.7	99.4
	Time								
	2-GPU|1-GPU)	68|79	67|77	66|77	65|75	58|64	57|63	56|63	57|63
	QP	24	28	32	37	24	28	32	37
	PSNR	40.268	38.462	36.579	34.243	40.484	39.195	37.829	35.884
5	**Bit-Rate**	1089.4	592.8	345.6	200.5	744.6	345.3	173.0	79.3
	Time								
	2-GPU|1-GPU	116|148	115|146	113|145	112|141	111|136	109|134	106|129	102|124
	QP	24	28	32	37	24	28	32	37
	PSNR	39.307	37.765	36.026	33.726	38.945	37.671	36.432	34.714
6	**Bit-Rate**	1549.0	861.0	503.8	275.6	1210.1	527.3	255.0	113.8
	Time								
	2-GPU|1-GPU	68|79	67|77	66|76	65|75	59|65	57|64	57|63	57|63
	QP	24	28	32	37	24	28	32	37
	PSNR	39.484	37.595	35.550	33.051	39.541	38.133	36.728	34.835
7	**Bit-Rate**	1638.7	935.5	543.8	285.5	1078.4	532.1	266.8	111.6
	Time								
	2-GPU|1-GPU	65|75	64|74	64|73	63|72	57|64	56|63	55|62	55|61

from zone #1 to other zones, with a large percentage going to zone #4. This stems from the prediction structure in Fig. 3.2 (c), where these views use more inter-view reference frames. Multiple views capture the same scene from different disparate viewpoints and, therefore, the resulting DV is larger than the MV in the temporal

Table 2.8
Comparison of DZfast to CPU-based Full Search Averaged Over Eight Views

	Ballroom				Race1				Exit				Vassar			
QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
DPSNR (dB)	-0.0346				-0.049				-0.0314				-0.0110			
DBR (%)	+0.942				+1.14				+0.9219				+0.3925			
Speedup (1-GPU)	167	168	172	178	105	106	109	115	164	167	170	174	199	204	205	211
Speedup (2-GPU)	203	205	206	212	145	146	147	151	199	202	206	210	226	232	236	246

Table 2.9
Comparison of DZfast to TZsearch Averaged Over Eight Views

	Ballroom				Race1				Exit				Vassar			
QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
DPSNR (dB)	-0.0093				-0.0322				-0.0142				+0.0103			
DBR (%)	+0.2556				+0.87				+0.3461				-0.4581			
Speedup (1-GPU)	6.9	5.7	5.4	4.9	4.5	4.3	4.2	3.9	5.6	5.2	4.9	3.9	5.2	4.5	3.9	3.1
Speedup (2-GPU)	8.4	7.0	6.5	5.9	6.3	6.0	5.6	5.1	6.8	6.3	5.9	4.7	5.9	5.1	4.3	3.7

Table 2.10
Comparison of DZfast to GPUfull Averaged Over Eight Views

	Ballroom				Race1				Exit				Vassar			
QP	24	28	32	37	24	28	32	37	24	28	32	37	24	28	32	37
DPSNR (dB)	-0.014				-0.0276				+0.0011				+0.0031			
DBR (%)	+0.3762				+0.73				+0.1038				-0.1008			
Speedup (1-GPU)	1.5	1.5	1.5	1.5	1.1	1.1	1.1	1.1	1.4	1.4	1.5	1.4	1.6	1.7	1.7	1.7
Speedup (2-GPU)	1.9	1.9	2.0	2.0	1.2	1.2	1.2	1.3	1.9	1.9	1.9	1.9	2.3	2.3	2.4	2.4

domain.

Fig. 2.12 presents the RD performance of DZfast along with TZsearch and CPU-based full search (CPUfull). Three algorithms virtually have the identical performance. Tables 2.8, 2.9, and 2.10 present the comparison of the performance of DZfast with respect to PSNR, bit-rate, and execution time speedup averaged over eight views, with CPU-based full search, TZsearch and GPUfull, respectively.

Notable from Table 2.8, in comparison with CPU-based full search, DZfast loses no more than 0.05 dB in PSNR and 1.14% increase in bit-rate, averaged over eight views. It should also be noted that DZfast performs better for the video sequences with more

static content like Vassar. This is inline with the results in Table 2.5 in relation to GPUfull and TZsearch.

As seen from Table 2.9, in comparison to TZsearch, DZfast gains a speedup of 3.1 to 6.9 over the single GPU and 3.7 to 8.4 over dual GPU, over the range of QPs and the set of four video sequences tried. The execution times for TZsearch and DZfast follow similar trends with respect to the complexity in the video content. Therefore, their speedups remain relatively independent of video sequence. Therefore, unlike Tables 2.4, 2.5 and 2.8, where there is a clear correlation between the content of the video sequence and its corresponding speedup values, Table 2.9 conveys no such correlation. This can be easily verified by comparing the results for Race1 with Exit and Ballroom.

Table 2.10 indicates DZfast maintains the same RD performance as GPUfull in Section 2.3 with speedup factor of 1.1 to 1.7 with single GPU and 1.2 to 2.4 with dual GPU. It should be noted that highly dynamic video sequence, Race1, with high values of P_{96} and P_{128}, results in less improvement of DZfast over GPUfull than the other video sequences. This is inline with the results in Table 2.8 in relation to DZfast and CPU-based full search.

As a last point on DZfast, it should be highlighted that the scalability features of GPUfull across the multi-GPU platform (Section 2.3.4) is also directly applicable to DZfast.

2.6 Application to High Efficiency Video Coding (HEVC)

As stated earlier, the state-of-the-art video coding standard, H.264/AVC, includes MVC as one of its extensions. This section provides a brief discussion of the applicability of MPA techniques, as the technology moves to its successor, HEVC [46]. HEVC inherits a majority of its coding features from H.264, with some modifications to improve the coding efficiency. The changes that are relevant to ME/DE are inclusion of additional partition types and larger sizes [46]. HEVC supports up to 12 variable block sizes ranging from 4×8/8×4 to 64×64. Therefore, the ME/DE process in GPUfull and DZfast are directly applicable to MVC extension of HEVC. The exploratory work in [47] proposes the use of GPU along side of CPU for HEVC. To adapt DZfast to HEVC, nevertheless, requires minor modifications to improve its performance. The number and size of decision zones must increase to handle additional partition types and larger sizes in HEVC. Also, MV/DV predictor may require a redesign to better handle the increase in number of partition types. Notwithstanding those minor modifications, the search algorithms in DZfast are completely reusable and require no changes.

2.7 Conclusion

This chapter introduced two efficient parallel algorithms for ME and DE, suitable for implementation on the MPA of GPU. Two algorithms exhibit negligible difference in PSNR and bit-rate in comparison to the state-of-the-art TZsearch, gaining an overall speedup by a factor of up to 7 for the range of video sequences tried. The advantage

of the proposed DZfast algorithm is the adaptivity to individual macroblocks in a video sequence. The search range is dynamically adjusted depending on the amount of motion variation in the macroblock. The proposed parallelization demonstrated a way to break the dependency in partition MVP_p values, with an insignificant amount of loss in the RD performance. Algorithms proposed in this chapter are easily scalable to multiple GPUs. The modification of the algorithms for HEVC is straightforward.

Chapter 3

High Level Parallelization exploiting GOP Level Parallelism [1]

This chapter presents the use of a computer cluster with heterogeneous computing components to provide concurrency and multi-level parallelism at coarse grain and massive fine-grain for multiview video coding (MVC) applications. MVC involves coding of multiple video sequences that are taken from the same scene but different perspective. In addition to motion estimation (ME) used in conventional video coding for single view video for exploiting inter-frame temporal similarities, MVC adopts disparity estimation (DE) to further increase compression. To overcome the huge computational cost associated with ME and by extension with DE, attention has been mainly focused on developing fast ME/DE algorithms. Although fast ME/DE algorithms bring substantial speedup, to achieve realtime MVC encoding, it requires further acceleration of the coding process at higher levels. Towards this end, this chapter discusses a multiple-view-parallel, multiple-interleaved group of pictures

[1] The material contained in this chapter was previously published in "*IEEE Transactions on Parallel and Distributed Systems*" ©2016 IEEE. See Appendix A.2 for copies of the copyright permission from IEEE.

(multiple-IGOP) scheduling scheme for MVC. When evaluated over eight views, with no loss in rate distortion (RD) performance, the proposed scheme outperforms view-sequential coding by a factor of up to 12.4 and 12.3, respectively, for two popular prediction structures, IBP and IPP.

3.1 Introduction

High level trends in media mining systems such as applied analytics for intelligent video surveillance (IVS) systems depends on several underlying enabling computational techniques and technologies such as video mining, computational intelligence, computer vision, physical simulation, all of which involve extraction of meaningful and actionable knowledge from large amounts of streaming multimedia and sensory data such as multiview video [48, 49]. These applications are highly complex, require a huge amount of computing power, and demand realtime or even super-realtime processing capability. Our recent work on multiview video processing [41, 50] at frame resolution of $576p$ (720×576), showed that the requirement is Giga-operations per second to deliver realtime performance. Serial processing speed on today's typical state-of-the-art general purpose processors is about two frames per second per view, about 480 times slower than realtime (30 frames per second per view) for an 8-view multiview video processing. Therefore, it is highly desirable to accelerate these time consuming media mining workloads on a multitude of heterogeneous platforms to achieve realtime streaming performance. In the particular domain of multiview video IVS computational hardware requires significant sensing and processing capabilities. Additionally, the hardware from multiple views need to exchange data for efficient multiview compression and video analytics.

However, one characteristic of media mining algorithms such as multiview systems

that helps us in this regard is their significant amount of data parallelism, at multitude of levels of granularity, which can be leveraged to enable realtime stream processing and parallel computation on heterogenous high performance computing platforms, equipped with general purpose processors and specialized hardware accelerators. The primary source of improved computational performance for data analytics will come from parallel processing of streaming data on a potentially bewildering and constantly evolving hierarchy of commodity parallel computing resources. For instance, typical computing resources currently include multicore central processing units (CPUs) operating in a multiple-instructions multiple-data (MIMD) model, together with a massively-parallel architecture (MPA) of many-core graphical processing units (GPU) with significant fast shared memory which operate most efficiently in a single-instruction multiple-data (SIMD) or single-program multiple-data (SPMD) mode with a given vector length [51, 52, 53]. These resources in instances are augmented with specialized processors / accelerators / coprocessors that are application-specific, *e.g.*, image processing, video coding, compression/decompression, pattern-matching, cryptographic, extensible markup language (XML) accelerators, and/or host Ethernet accelerator (HEA) packet processors [54, 55, 56, 57, 58, 59, 60, 61].

The number of CPU cores and the shared-memory hierarchy between these cores (each of which has access to a CPU vectorized floating-point coprocessor in addition to the GPU resources) is currently undergoing revolutionary changes [59]. The number of GPU cores is increasing very rapidly which is motivating increased flexibility in the hierarchical GPU programming model; this is best exemplified by the NVIDIA Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL) standards [62] which now allows hierarchical grids of blocks (1, 2, and 3-dimensional) of GPU kernels to be initiated and controlled from the GPU [63, 64, 65]. As a result, current standard desktop computers have at least seven readily accessible hierarchical programming levels—first level on the CPU and the second to seventh levels on the GPU—on two device classes—say a 4-core CPU and a 480-core GPU—connected

through a complex memory hierarchy. Integrated multicore CPU/GPU devices, such as the Intel Xeon processors with Intel Xeon Phi coprocessors and the AMD Fusion family of application processing units (APUs), further complicate data processing for technical computing problems.

In this chapter a case study is presented for the use of a heterogeneous computing platform, consisting of cluster of CPUs and GPUs, for multiview applications in which there is a requirement to process huge amounts of multimedia data, such as media mining applications.

Multiview video coding (MVC) involves the coding of a scene that is simultaneously captured by multiple cameras. Among the applications of MVC, 3D video is most well-known. MVC has been incorporated into the latest high efficiency video coding (HEVC) standard commonly known as H.265/HEVC [46, 66]. Similar to its predecessor advanced video coding (AVC) H.264/AVC standard [3], HEVC standard has adopted an MVC coding technique that is the extension of single view video coding but with added syntax to support inter-view prediction. The H.264/AVC MVC extension is chosen as the test platform, to investigate some common MVC prediction structures and their suitability for parallel processing.

A typical advanced video encoder has four major functional elements; intra-prediction, inter-prediction, transformation and entropy coding. Among the four functional elements, inter-prediction incurs the highest computational cost but yields the most coding performance gain. Inter-prediction involves the process of forming a prediction block from previously coded picture(s). A motion vector is calculated from the displacement between the current block and the predicted block. Only the motion vector and the difference (residual) between current block and the predicted block, are coded and transferred into the bitstream. The process of searching for the best motion vector is called motion estimation (ME), and the use of spatial displacement

motion vectors to form a prediction is known as motion compensation (MC). When extending this concept to multiview video coding, the method of exploiting disparity between the views from neighboring cameras to form a prediction is called disparity compensation (DC). Similarly, the process of estimating disparity vector between the views from neighboring cameras is called disparity estimation (DE). The DE uses the same block matching technique used for ME [25].

Existing works for reducing encoding time are mainly focused on reducing the complexity of ME/DE by employing sub-optimal block matching algorithms such as diamond search [9], hexagon search [8], and UMHexagonS [10]. A substantial speedup, with minimal rate distortion (RD) performance loss, is achieved by exploiting low level parallelism in the block matching process. Hardware accelerated block matching algorithms are proposed in [29] and [67]. There are also several recent works on the use of MPA of GPUs to accelerate the ME/DE [50] (and references thereof), [41], [47], [15].

Low level parallelism, however, limits the parallel processing to the pixels in a coding unit (CU) (or macroblock)[2], and therefore, does not scale well with the computational resources of multicore central processing units (CPUs) that do not feature MPA. Even computational resources (computing cores, shared memory, registers, cache, and memory bandwidth) of MPA such as GPU saturate when the search range for the block matching process reaches a point [50]. Therefore, further increase in the speedup in the MVC requires exploitation of coarse-grain parallelism at the higher levels of hierarchy.

High level parallelism includes CU, slice, and frame level parallelism. Parallel processing of CUs within the same frame using the MPAs is possible, provided coding

[2]Coding unit (CU) is the terminology used in H.265/HEVC standard which is similar to macroblock in H.264/AVC

dependencies are resolved. One approach to overcome the coding dependencies among the CUs in a frame is the concept of diagonal wavefront processing, proposed in [43] a concept that was adopted in HEVC [66]. To have enough concurrent CUs available for coding, this approach may even require out-of-order processing of CUs from different slices, when such slices contain few rows of CUs, as required for improved error-resiliency. In our experience [50], within the limitation of the state-of-the-art MPAs, with just hundreds of cores, there may be little to be gained by CU level parallelism.

This chapter focuses on frame level parallelism using group of pictures (GOP). This is the most suitable level of parallelism for a multicore and cluster computing platform. The strategy for frame level parallelism presented in this chapter, as will be demonstrated, can be easily extended to slice level parallelism, with little modification, by breaking the frames in a GOP into multiple slices.

A computing cluster with its underlying heterogeneous computational units (CPU(s) and GPU(s)), provides an ideal opportunity to accelerate data processing units in presence of both low-level massively parallel, and sequential workloads, when multitude of tasks as various levels of hierarchy can be executed concurrently [68, 69, 70, 71].

This chapter is organized as follows. Section 3.2 presents a brief description of MVC prediction structures. Section 3.3 discusses the opportunities for exploiting frame level parallelism using view-parallel scheduling scheme. Section 3.4 presents a scheme for multiple-view-parallel, multiple-interleaved GOP (multiple-IGOP) scheduling. Section 3.5 presents the latency and memory considerations for the multiple-IGOP scheme. Section 3.6 discusses the platform specific issues. Section 3.7 presents our implementation and performance results of the view-parallel and multiple-IGOP encoding schemes. This section also covers an in-depth analysis for the proposed schemes. Section 3.8 provides a brief description of slice and CU level parallelism. Section 4.7

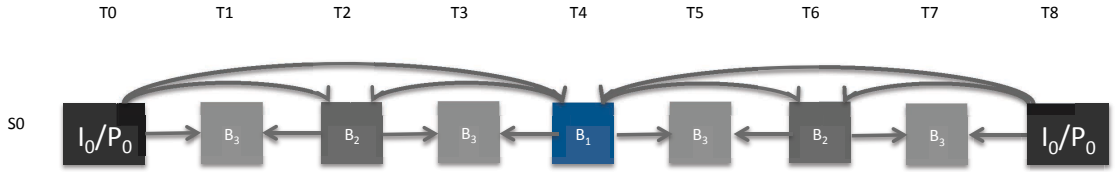


Figure 3.1: Hierachical B-frames temporal prediction structure

concludes the chapter.

3.2 MVC Prediction Structure

In an advanced video encoder, two prediction types are commonly used, intra-prediction and inter-prediction. Intra-prediction involves forming the prediction block using reconstructed samples from the current frame. The method of forming intra-predicted blocks is standardized in the state-of-the-art video standards. Inter-prediction involves forming prediction block using reconstructed samples from previously coded picture(s). It is also known as temporal prediction. Unlike intra-prediction, temporal prediction structure is left unspecified in the standards. This leaves room for a wide range of choices between coding performance and computational complexity. Similarly, modern video coding standards do not specify an inter-view prediction structure. The choice of prediction structure for MVC plays an important role in coding performance and complexity of the GOP level parallelism.

3.2.1 Temporal Prediction Structure

In the temporal domain the common prediction structure is IBBP (I-frame, [B-frame, B-frame, P-frame], ... [B-frame, B-frame, P-frame]) [72]. However, this is not the most efficient prediction structure. To improve the coding efficiency a hierarchical

B-frames prediction structure was introduced in [7]. A typical hierarchical B-frames structure with three levels of hierarchy is shown in Fig. 3.1. The first frame in this figure (I-frame) is intra-coded and it is known as a key frame. The key frame and all frames before the occurrence of the next key frame, (a total of eight frames), form a GOP. Frames between two key frames are called nonkey frames. In hierarchical B-frames prediction structure, all nonkey frames are B-frames (bi-directional), and are predicted using only the pictures of the same or higher temporal level of hierarchy as reference. Using the comparison metric in [1] hierarchical B-frames prediction structure achieves video quality that is better than the IBBP by 0.5 to 1 dB in peak-signal-to-noise-ratio (PSNR). Alternatively, it results in a saving of 15% to 22% in bitrate. The improvement in coding performance comes at the cost of increased coding complexity, as a picture's reference frame(s) must be coded, prior to the coding of the picture itself.

3.2.2 Inter-view Prediction Structure

Extension to multiview system results in a diverse range of prediction structures. Fig. 3.2 shows four commonly used prediction structures, ranging from the simplest (Fig. 3.2 (a)) to the most coding efficient (Fig. 3.2 (d)). In the figure views are indicated as V_0-V_7 and frames within the views are identified as T_0-T_{11}. The straightforward solution is to encode each view independently by forming GOPs, as shown in the simulcast prediction structure of Fig. 3.2 (a). However, this method fails to exploit inter-view dependencies and results in significantly higher bitrate [25] [26].

To improve the coding efficiency for the multiview system, the hierarchical B-frames prediction structure using GOP for temporal domain can be combined with an inter-view prediction structure. Depending on whether inter-view prediction is applied to

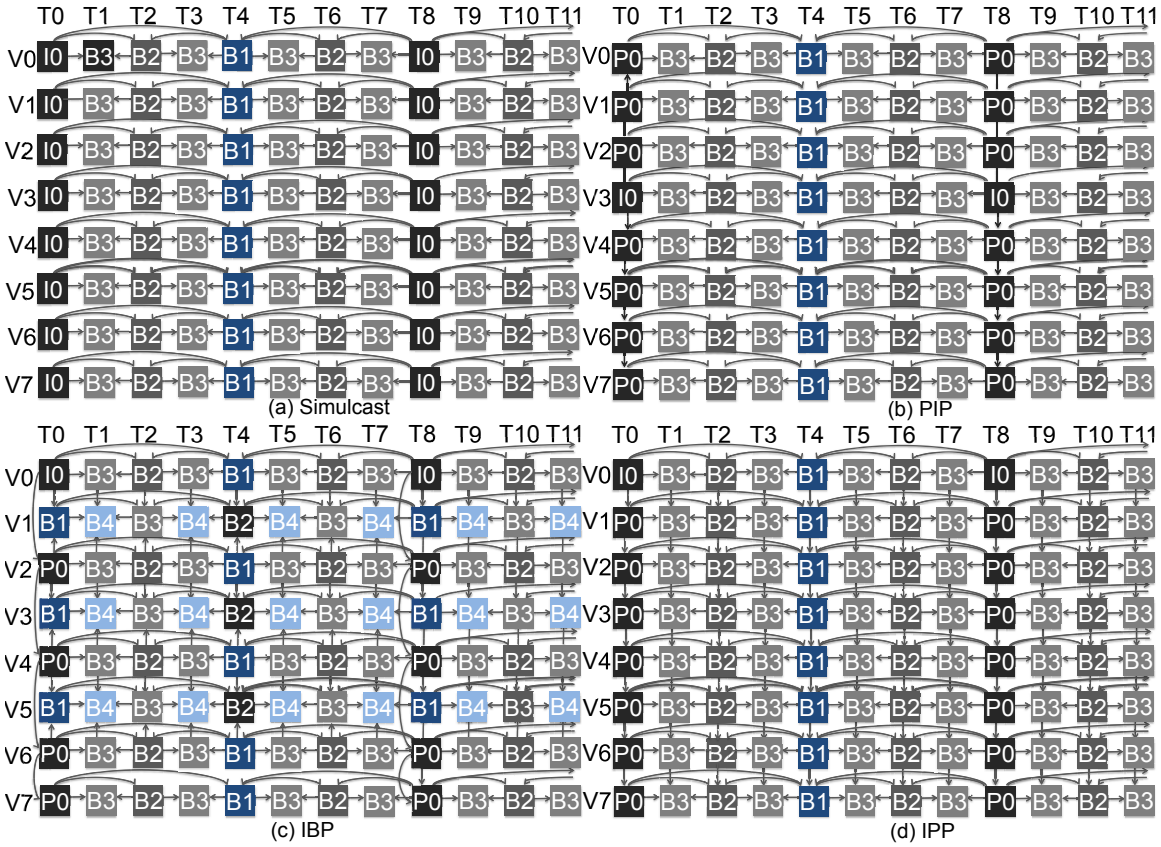


Figure 3.2: Multiview predication structure: (a) simulcast: independently coded without inter-view prediction, (b) inter-view prediction for key frames only with PIP structure, (c) inter-view prediction for all nonkey frames with IBP structure, (d) inter-view prediction for all nonkey frames with IPP structure

nonkey pictures, two types of structures are proposed in [7]. Fig. 3.2 (b) with no inter-view prediction for nonkey pictures, has a PIP (P-frame,...,P-frame, I-frame, ..., P-frame) prediction structure. In Fig. 3.2 (c) and (d), on the other hand, where inter-view prediction is applied to nonkey pictures, will have, respectively, IBP (I-frame, ..., [B-frame, P-frame], ..., [B-frame, P-frame], P-frame) and IPP (I-frame, P-frame, P-frame, ..., P-frame) structures. A significant increase in the rate-distortion (RD) performance can be obtained with the inter-view prediction [7], with IPP being the best performing structure (0.25 and 0.5 dB, respectively, with respect to IBP and PIP, for the same bitrate). From an implementation point of view. It should be noted that

IPP, IBP and PIP structures have different inter-view dependencies that need to be accounted for efficient GOP level parallelization. From Fig. 3.2 it should be clear that IPP is the most complex coding structure as it has more inter-view references than the other prediction structures.

Works in [23] and [24] present scheduling algorithms for parallel MVC encoding at the frame level on a multi-processor system for a given prediction structure. In these works a prediction structure is used to build a directed acyclic graph where frames across the temporal and inter-view domains form the vertices of the graphs and the coding dependencies form the edges. Starting with I-frames vertices in view 0 as pair of roots, the encoding scheduler inspects all the neighboring vertices and assigns them to one of the available CPU cores. Then, for each of those neighbor vertices in turn, it inspects their neighbor vertices which are not yet coded, and so on. The process continues until all the GOPs across all the views are encoded. This requires a complex scheduler that traverses the graph to discover and schedule the frames that are ready to be dispatched for the execution on one of the many identical CPU cores. Further, in this scheme the workload, in terms of the number of frames ready to be scheduled, at each coding stage varies greatly across the stages. This results in under-utilization of CPU cores or inadequate number of cores for efficient parallelism depending on the coding stage. To alleviate this problem by creating enough workload to keep all the CPU cores busy, the work in [23] proposes the processing of multiple GOPs across all the views in parallel, further complicating the scheduler. The scheduler task becomes even more cumbersome considering the fact that at each stage of encoding frame vertices take widely different execution times depending on the number of their immediate descendants in the graph and the nature of edge dependencies (temporal or inter-view).

In contrast, this work develops a simple scheduling scheme where the number of frames to be encoded does not change across the coding steps. As will be described

in the next section this is achieved through a simple encoding step time shift. The simplicity of our parallel scheduling scheme results in the more complex prediction structure of IPP to have a more efficient parallel implementation compared with IBP. This is in spite of the fact that execution time for sequential processing of frames for the IPP prediction structure on a single processor, having four more inter-view references is more than that of IBP. To afford a scalable implementation, computing cluster is employed where computing nodes can be added as required. Further, our simple scheduling scheme allows for efficient use of heterogeneous computing platform where each cluster node consists of several CPU cores and specialized GPUs.

3.3 View-parallel Model of MVC

For the purpose of studying GOP parallelism, two most efficient prediction structures, IBP and IPP are selected. The basis for comparison is given to the simulcast scheme of Fig. 3.2 (a) that yields itself to the highest level of GOP parallelism, as it has no inter-view prediction structure.

The straightforward high level parallel implementation of multiview video coding for inter-view IBP prediction structure with eight views is shown in Fig. 3.3. In this implementation pictures from each view are processed by a separate compute node. The assignment of views to compute nodes are shown through their identification (ID) numbers. Each vertical slice corresponds to a coding time step. In each time step, one full video frame is coded by each of the compute nodes. The stretch of time steps (T_0 to T_{11}) for encoding eight GOPs for eight views are indicated by light blue cells in Fig. 3.3. Yellow cells refer to the previous and next GOPs. In each compute node, coding sequence follows the hierarchical B-frames prediction structure ($T_{8/0}$,

Figure 3.3: View-parallel processing for IBP prediction structure (a) scheduling across compute nodes, (b) workload analysis across compute nodes

T_4, T_2, T_1, T_3, T_6, T_5 and T_7). Dark blue, light gray and dark gray color coded compute nodes correspond to views that are, respectively, coded independently, through one-directional inter-view referencing, and bi-directional inter-view referencing. Red

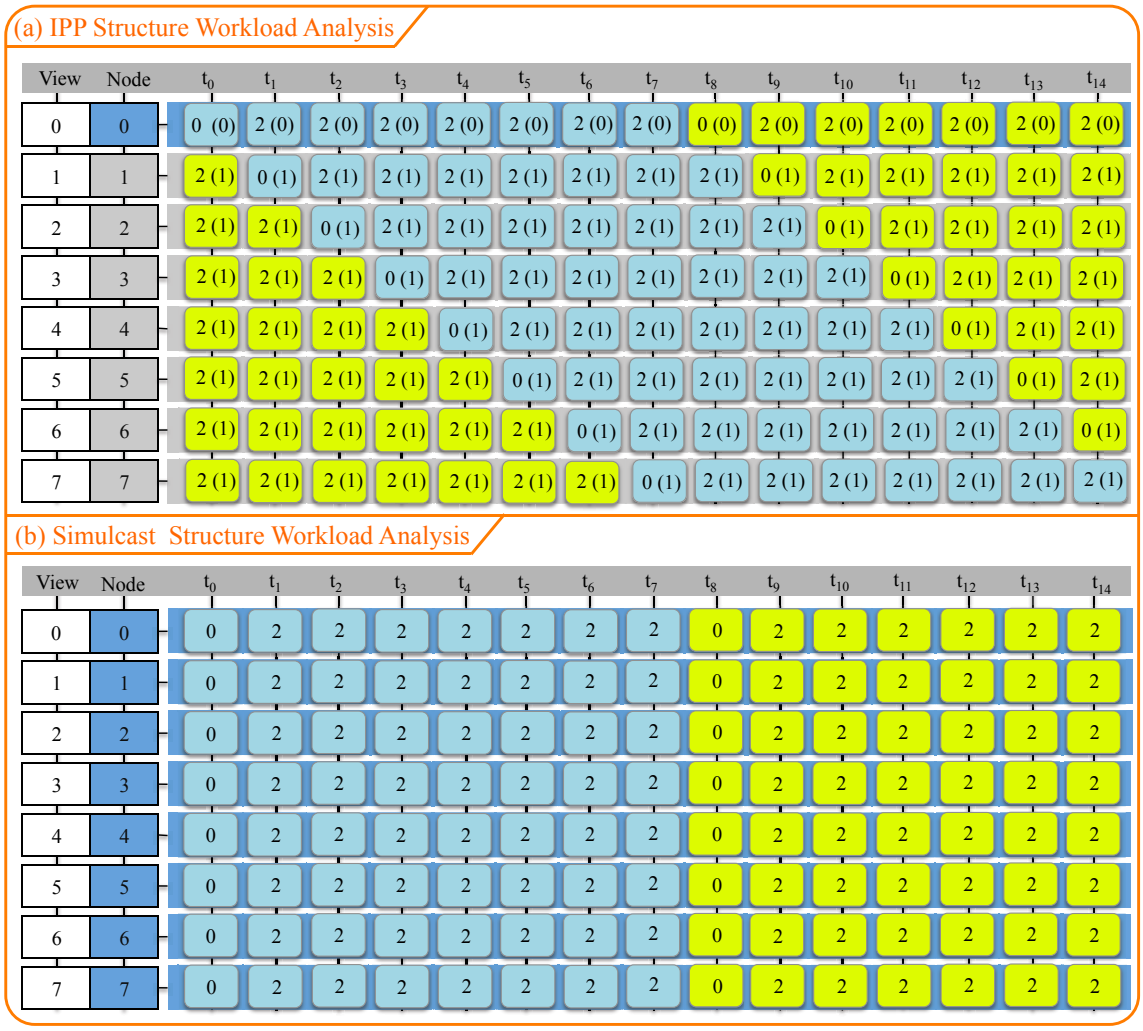


Figure 3.4: Workload analysis across compute nodes for view-parallel processing for (a) IPP prediction structure and (b) simulcast prediction structure

and green arrows in Fig. 3.3 (a) correspond to bi-directional and one-directional referencing to the neighboring views, respectively, for IBP in Fig. 3.2 (c)[3]. Inter-view dependencies are handled by an appropriate alignment of the view-frames in coding time steps across the compute nodes. This can be achieved through a simple scheduling scheme of Fig. 3.3 (a) where compute nodes keep in lock-step with each other

[3]Note that the green arrow has one source and one destination compute node. The red arrow, on the other hand, has two sources and one destination compute node. For example, the sources for red arrow for compute node 4 at coding time step T_3 are compute nodes 1 and 3 at time steps T_1 and T_2, respectively.

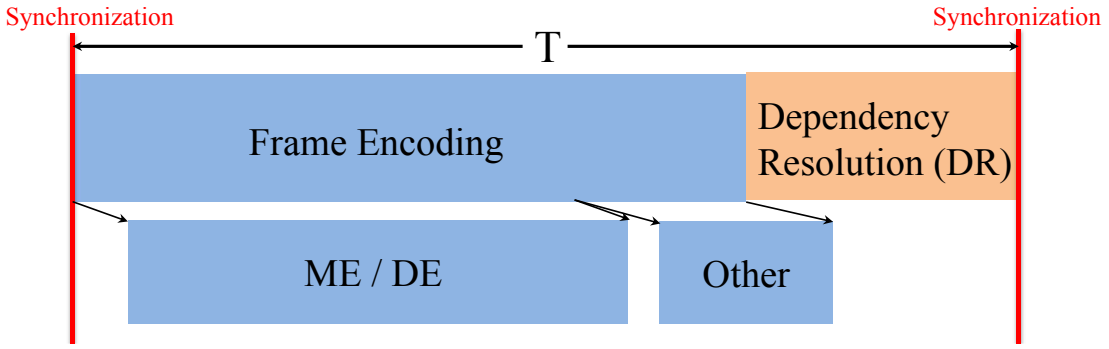


Figure 3.5: Frame processing epoch

by an appropriate hand-shake message passing for the purpose of sending/receiving reference frame(s) to/from each other.

The parallel scheduling model in Fig. 3.3 (a) results in the workload distribution for the compute nodes in various time steps as shown in Fig. 3.3 (b). The workloads for the compute nodes in each time step are indicated by a pair of numbers R_T (R_V) that correspond to the number of required references to the neighboring frames in the temporal (inter-view) domain in that coding time step. Each reference corresponds to an ME/DE process. The ME/DE process in MVC consumes the majority of the overall coding time (up to 99%) [41]. Therefore, the execution time for each coding time step is proportional to the weighted sum of numbers in R_T (R_V) pair. It is noted in each coding time step the workload distribution across the compute nodes are not uniform and tends to be higher for the dark gray coded nodes. For example, for coding time steps T_5 to T_7 where the workloads for all the compute nodes reach their maximum, the workloads for the dark blue, light gray and dark gray color coded compute nodes correspond to two, two and four reference frames, respectively. This uneven workload distribution results in slow down of dark blue, light gray compute nodes to keep in lock-step with the dark gray compute nodes. The workload distribution becomes even more uneven during the other coding time steps. For example the corresponding workload pairs of references in coding time step T_0 are zero, two, and four for the same sequences of color coded compute nodes.

To better analyze the parallel scheduling scheme of Fig. 3.3 (b) each coding time is modeled as a two-part frame coding and dependency resolution (DR) epoch as shown in Fig. 3.5. Coding part involves the coding of CUs in the frame. The DR part accounts for the exchange of reference frames required by the inter-view prediction structures for the next coding time step. At the start of each coding time step, compute nodes are synchronized with each other. Dependencies in the temporal domain, within a single view, do not need resolutions across the compute nodes. As will be seen later in our implementation platform, the DR part constitutes a tiny fraction of the overall coding time ($< 0.1\%$).

It is also noted that the time associated with the frame encoding in Fig. 3.5 consists of two parts; the part associated with the ME/DE processing for the frame references and the other part associated with the rest of coding functional elements (transformation and entropy coding, etc). For a given ME/DE search algorithm, the ME/DE processing time depends on the number and type of references (prediction structure). The other part of coding is much smaller and weakly depends on the number of references.

The scheduling schemes in Fig. 3.4 show the similar workload distributions in terms of number and type of reference frames for IPP and simulcast prediction structures. It is clear that simulcast, having no inter-view dependency, has an uniform workload distribution across all views (requiring only one compute node color coding), which reaches a maximum of two references. The workload for IPP as can be seen is more evenly distributed, where two compute node color codings suffice. This is for the fact that IPP as can be seen in Fig. 3.2 (d) has no bi-directional references in inter-view domain.

To compare the compute performance of various inter-view prediction structures using the view-parallel scheme, in terms of balance in the workload distribution among

Table 3.1
Average α Values of TZsearch and DZfast for Different Video Sequences

Video Sequence	Ballroom	Vassar	Exit
α_{DZfast}	1.1	1.1	1.1
$\alpha_{TZsearch}$	2.5	4.5	3.5

compute nodes, for each time step, the quantity *balance index* (BI) is introduced as a measure of deviation from a balanced workload distribution among the compute nodes in the view-parallel scheme. The BI for a given time step is expressed as,

$$\mathrm{BI} = (\mathrm{CT} - \mathrm{CT}_{avg})/\mathrm{CT}_{avg} \tag{3.1}$$

where CT and CT_{avg}, respectively, correspond to actual and average compute times, expressed in units of number of references per frame for view-parallel scheduling for inter-view prediction schemes. Further, CT is represented as,

$$\mathrm{CT} = R_T + \alpha R_V \tag{3.2}$$

where α is a multiplicative factor that accounts for the relative complexity of the DE compared with ME. The value of α depends on the algorithm used for ME/DE processing (full search vs sub-optimum fast search) and the nature of the video sequence. For the search algorithm employed and the range of video sequences [44] tried in this work (Section 3.6), the value of α, as obtained from running the encoder, varies from 1 to 4.5 as shown in Table 3.1. In this analysis it is assumed that the execution times for given values of R_T and R_V are fixed across the views and encoding time steps. This assumption while not quite valid for a single view, is sufficiently accurate across several views.

Starting with scheduling scheme for the simulcast structure in Fig. 3.4 (b), it is noted

Table 3.2
Load Balance Analysis for View-parallel IBP Prediction Structure

	$\mathbf{T}_0$	T_1	T_2	T_3	T_4	T_5	T_6	T_7
CT	4	4	4	4	4	4	4	4
CT_{avg}	2.50	2.63	2.38	2.38	2.38	2.75	2.75	2.75
BI	0.60	0.52	0.68	0.68	0.68	0.45	0.45	0.45
GOP Average	CT(GOP): 4,			CT_{avg}(GOP): 2.56,			BI: 0.57	

that for an eight-step cycle T_0 to T_7, for all views, there is one step with no references and seven steps with two references. Hence, Simulcast for all time steps has BI = 0, a perfect workload balance. For IPP for scheduling in Fig. 3.4 (b), assuming $\alpha = 1$, for all timing steps CT and CT_{avg} are, respectively, 3 and 2.63 in units of references, yielding BI = 0.14.

For IBP in Fig. 3.3 CT = 4 for all timing steps. On the other hand, values for CT_{avg} and BI vary across the timing steps as shown in Table 3.2. Since the range of values for CT_{avg} and BI pair across the eight time steps are close to each other, it is convenient to average them across the GOP, *i.e.* $\mathrm{CT}_{avg}(\mathrm{GOP}) = 2.56$ and $\mathrm{BI}(\mathrm{GOP}) = 0.56$. It can also be written as $\mathrm{CT}(\mathrm{GOP}) = CT = 4$. Irrespective of the value of CT_{avg}, to keep the compute nodes in lock-step with each other, the actual workload for each timing step is determined by the value of CT = 4 in units of references for $\alpha = 1$.

In conclusion view-parallel implementation of IPP is more balanced than IBP. This should come as no surprise as IBP involves a mixture of one and bi-directional inter-view referencing whereas IPP has only one-directional inter-view referencing.

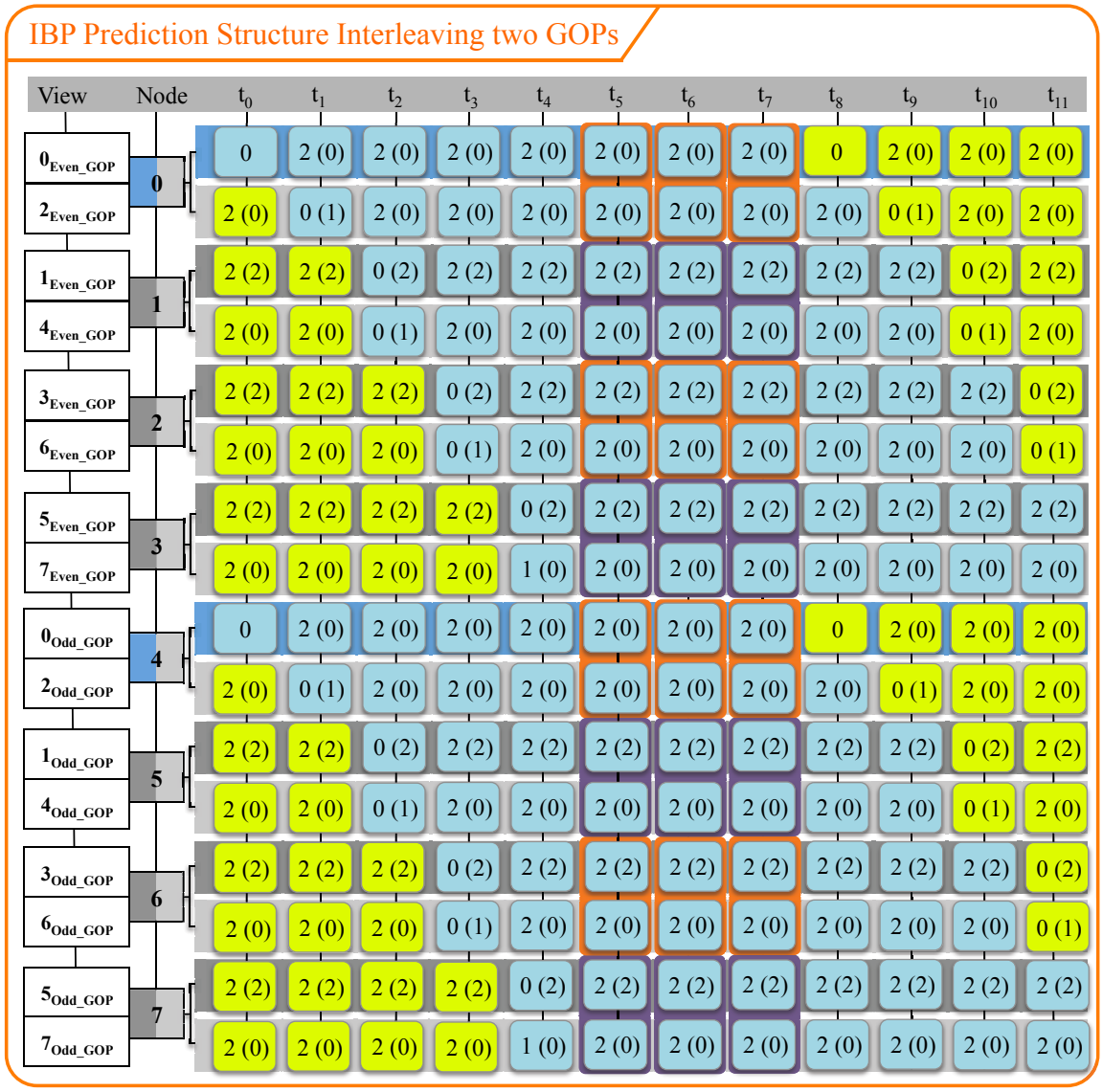


Figure 3.6: Two-view-parallel, two-GOPs interleaved (2-IGOP) parallel processing for IBP prediction structure

3.4 Multiple-view-Parallel, Multiple-inter leaved-GOP Model

To balance the workload among the compute nodes, in this section, a multiple-view-parallel strategy that exploits GOP parallelism is proposed. It is observed in the temporal domain using hierarchical B-frames prediction structure eight pictures in the GOP are encoded at a time. Since GOPs are coded independent of each other, it is, therefore, possible to interleave the processing of several GOPs with no sacrifice to the bitrate or video quality. Further, in this scheme, to balance the workloads across the compute nodes, multiple-views are assigned to each node.

A two-view-parallel, two-GOPs interleaved (2-IGOP) strategy for IBP prediction structure is given in Fig. 3.6 where views are paired together according to the inter-view prediction structure to balance the workloads across the compute nodes. In this scheduling scheme even GOPs are assigned to compute nodes 0 to 3 and odd GOPs are assigned to compute nodes 4 to 7. At the coding time steps T_5 to T_7, where all compute nodes reach their maximum workload, it can be seen that compute nodes 0 and 4 have a workload of four reference frames, and the rest of the compute nodes have six reference frames. This workload distribution of four to six is more balanced than two to four for IBP view-parallel structure in Fig. 3.3. The CT_I and CT_{avg}(GOP) for this multiple-view-parallel scheme are 3 and 2.56 in units of reference frames, with $\alpha = 1$. The corresponding BI value is reduced from 0.56 to 0.17 which is similar to the value obtained for IPP scheduling of Fig. 3.4.

Extending this concept to four-view-parallel, four-interleaved GOPs (4-IGOP), the CT_I and CT_{avg}(GOP) change to 2.88 and 2.56, with $\alpha = 1$. The corresponding BI value drops further to 0.12 indicating a substantially improved balance in the

workload, compared to view-parallel scheduling. With 8-view-parallel, 8-interleaved GOPs (8-IGOP), the corresponding values for CT_I and $CT_{avg}(GOP)$ converge to a single value of 2.56, resulting in BI = 0. The is a perfect balance in the workload, as far as the number of references is concerned.

It can be seen that the change in the BI value in going from a 4-IGOP to an 8-IGOP scheme is very little. Considering the various computations required by the other coding elements beyond ME/DE and the resource limitations of the underlying implementation platform, as will be demonstrated in the next section, going from a 4-IGOP to an 8-IGOP scheme does not produce significant change in performance.

Table 3.3 provides a summary of the load balance analysis for various multiple-view-parallel GOP-interleaved strategies for IBP, IPP and simulcast, for $\alpha = 1$. While IBP structure has a wide range of BI values, the corresponding range for the IPP structure is very limited. It starts from a low value of BI = 0.14 for the view-parallel scheme and does not change at all in going to 2-IGOP scheme. Migration to a higher multiple-view-parallel produces similar results as IBP (with a slight edge over IBP).

The simple view-parallel and multiple-IGOP models developed for the parallel processing of MVC has the advantage of being independent of the implementation platform. This allows to deal with the parallelism and balance in the workload using a higher level algorithmic abstraction in the suitable unit of the number of references in temporal and inter-view domains, for the various prediction structures in Fig. 3.2, using a general concept of compute nodes. This abstraction has the advantage of demonstrating how the choice of MVC algorithm and high level parallel processing concepts influence the coding performance. It also helps to highlight the limits of parallel processing from a platform independent, algorithmic and architectural abstraction.

Table 3.3
Load Balance Analysis for Various View-parallel GOP-interleaved Strategies for IBP, IPP and simulcast

	IBP			IPP			Simulcast		
	CT_I	CT_{avg}	BI	CT_I	CT_{avg}	BI	CT_I	CT_{avg}	BI
View-parallel	4	2.56	0.56	3	2.63	0.14	1.75	1.75	0
2-IGOP	3	2.56	0.17	3	2.63	0.14	1.75	1.75	0
4-IGOP	2.88	2.56	0.12	2.88	2.63	0.10	1.75	1.75	0
8-IGOP	2.56	2.56	0	2.63	2.63	0	1.75	1.75	0

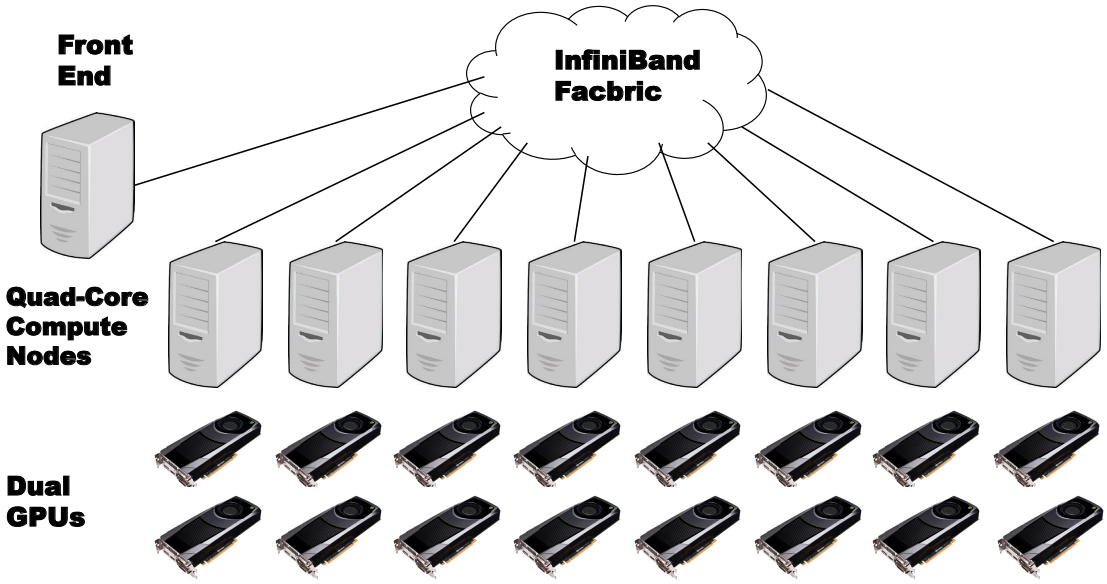


Figure 3.7: Eight compute node implementation platform

3.5 Latency and Memory Considerations

For a single view video stream the maximum latency in HEVC [66] is defined as the maximum number of frames before any frame in output order but follows in decoding order. For example in Fig. 3.1 in the 8-frame GOP consisting of frames T_1 to T_8, the frame T_8 is the last frame in the output but the first frame that needs to be decoded. This results in a latency of seven frames. This latency syntax is signaled in the coded bitstream. Extending this definition to multiview coding, it is a fact

that the views can only be decoded in a particular order according to the prediction structure. However, there is no requirement as to the order that they appear in the output. A syntax is adopted to take advantage of this fact, where the views are placed in the output in the same order that they are decoded. In this syntax for all view-sequential, view-parallel and multiple-IGOP schemes presented in this chapter, the maximum frame latency is no more than that of the single view stream.

However, adopting a definition of the latency akin to work in [24], the latency L is defined as the sum of the time to capture required number of GOPs across the multiple views, L_{cap}, and the time for the number of time steps required to encode frames in those GOPs, L_{epo}. L is expressed as,

$$L = L_{cap} + L_{epo} \tag{3.3}$$

with

$$L_{cap} = IF/R$$

and

$$L_{epo} = IST_{epo_I}$$

where I defines the interleaving factor (1, 2, 4, and 8 for view-parallel, 2-IGOP, 4-IGOP, and 8-IGOP, respectively), F the number of frames per GOP, R the frame rate (number of frames per second), T_{epo_I} the average epoch time per view for the multiple-IGOP scheme, and S the number of the time steps required to process all the frames in all the views in one GOP. From Fig. 3.3 and Fig. 3.4 for an 8-view system the value of S corresponds to 12, 15, and 8 for the IBP, IPP, and simulcast prediction schemes. For the view-sequential scheme, values of $I = 1$ and $S = 64$ (for an 8-view system with 8-frame GOP) are applicable.

The multiple-IGOP schemes rely on the fact that there is enough video buffer to hold the captured frames before the coding. Memory requirement can be expressed as $IFVPB$, where I and F where define before, V the number of views, and P the number of pixels in a frame, and B the average number of bytes per pixel. For the 8-IGOP scheme, having eight views with an 8-frame GOP, a video frame of 640×480 in YUV420 format, and average 12 bits per pixel, the total size of the memory buffer required is 225 Megabytes.

3.6 Implementation Platform Specific Analysis

3.6.1 Platform Level Specific Issues

So far, it is assumed that the ME and DE processing have the same level of complexity ($\alpha = 1$). This is certainly true of the full search algorithm for ME/DE, or almost true for a parallel fast algorithm of [50]. This section dives into the details of the implementation platform and demonstrate how platform dependent parallel processing features influence the execution performance of MVC. Here, the aim is to use the platform specific parallel features to reduce the epoch time in Fig. 3.5. Joint multiview video coding (JMVC) reference software suite [27] is adopted for experimentation. JMVC comes with added syntax to support inter-view prediction for AVC [3].

Fig. 3.7 depicts our implementation platform consisting of a compute cluster of eight nodes, with each node consisting of a 4-core CPU and two GPUs. Table 4.3 provides the details of the implementation platform. In our implementation platform the ME/DE processing part of frame encoding is performed using two different techniques.

The first technique uses the resources of GPUs by employing our parallel fast search algorithm, DZfast, in [50]. In this technique $\alpha \approx 1$ for the video sequences tried in this work. The second technique uses sequential TZsearch[4] [27], with $\alpha > 1$, on the CPU cores and is typically five times slower than DZfast [50].

3.6.2 Platform Level Issues for Parallel DZfast

In [50] the massively parallel architecture of GPU was utilized to develop a fast search algorithm (DZfast) for the ME/DE part of the coding in Fig. 3.5. Further, for a range of video sequences, DZfast achieves a value of $\alpha \approx 1$. The other part (combined contribution from the other coding functional elements) is executed on one of the CPU cores. With two GPUs per compute node there is the potential for executing two ME/DEs for CUs for two references in parallel. Further, with four cores it is possible to execute the other part of the coding of the CUs for up to four views in parallel.

However, heterogeneous nature of the computing platform complicates the analysis of the overall encoding time. This analysis requires a closer look at the overhead of interactions between the CPU cores and the GPU accelerators, the distribution of tasks between the computing resources, and the bottlenecks when there are contentions for use of same resources. The analysis as will be shown here requires observation of the coding process at fine granularity of CUs.

It is noted that, employing a single CPU core and a single GPU, with parallel DZfast on the GPU, the frame encoding time in Fig. 3.5 varies within a wide range from 0.2

[4]TZsearch is employed by JMVC [27], is a sub-optimum, but efficient fast search algorithm that reduces the overall computational complexity greatly, by a factor of up to 70 over the full search, while maintaining good RD performance.

to 5 seconds (s) with an average of 2.4 s[5]. The reason for this is the fact that ME/DE processing time for a frame varies largely depending on the number of references ranging from zero (for an I-frame) to 3.6 s (for four reference frames) on a single GPU. The average ME/DE processing time for a frame is 1.7 s. It should be also noted that the time associated with the rest of coding functional elements (transformation and entropy coding, etc) weakly varies with the number of reference frames ranging from 0.5 s to 0.9 s with an average of 0.7 s per frame.

With dual-GPU and quad-core CPU on a compute node, the ME/DE processing can be effectively overlapped for two or more reference frames, and the other coding part from multiple-views (up to four views) to reduce the epoch time in Fig. 3.5. Using our implementation platform it is easy to see that for view-parallel schemes in Figs. 3.3 and 3.4, all B-frames pair of ME/DE processing can be executed on two GPUs in parallel. For example for view 1 in Fig. 3.3 four references "2(2)" form two pairs of form "1(1)" with frames in each pair allocated evenly to two GPUs. This results in a significant reduction in the ME/DE time per frame with an average of 1.3 s. Note that the reduction going from one GPU to two GPUs is less than half. The reason for this is that first not all the references are even in number. Second there is an additional overhead involved in going from one to two GPUs. However, the other part of the frame encoding time for a single view can only run on a single CPU core and remains unchanged. Thus the overall encoding time reduces to 1.9 s. It is also easy to see that with two GPUs the execution times for actual workload of three references per coding time step for IPP and four references for IBP are closer to each other than the ratio of 3/4 indicated in Table 3.3. As will be seen in the next section this ratio is about 0.85.

For 2-IGOP scheme in Fig. 3.6 for IBP, execution of frame for encoding both views

[5]Note that these timing values are only indicative and can vary depending on the platform, and from one video scene to the other.

are carried out on the same node but overlap, as they run on two separate CPU cores. CPU cores, however, dispatch the ME/DE processing part of the frame coding to the same pair of GPUs. With up to six references per frame encoding time step, there is an obvious contention for the resources of dual-GPU. Therefore, ME/DE processing for pairs of references are executed on the dual-GPUs in a pair-sequential manner.

3.6.2.1 Coding unit level analysis

To analyze the situation further it is a fact that the actual coding process, on both CPU core and GPU, takes place with granularity of CU. Acknowledging that ME/DE process is carried out on GPU, a timeline model is constructed for the processing of multiple CUs on the parallel platform of Fig. 3.7 as shown in Fig. 3.8. In Scenario #1 ME/DE processing on one or two GPU(s) is followed by the processing of remainder of coding tasks on a CPU core. The view-parallel scheduling schemes in Figs. 3.3 and 3.4 with one view per node are typical of where Scenario #1 is applicable. This scenario presents the strict sequential processing nature of CUs on a CPU core, and GPUs are used to reduce the ME/DE processing time. Note that in this case $i = j$ in Scenario #1 in Fig. 3.8.

In Scenario #2 in Fig. 3.8, the execution of ME/DE for two or more CUs overlap each other. This scenario is typical of multiple-IGOP scheduling (*e.g.* Fig. 3.6). This overlap is due to fact that the CU workload for GPU(s) comes from different views. For the GPU architecture in Fig. 3.7, with specification in Table 4.3, ME/DE processing can be scheduled concurrently on the same GPU but are only processed sequentially. It should also be noted that Scenario #1 is also applicable to multiple-view-parallel, multiple-IGOP scheduling depending on the number of reference frames at any given time. Also from Fig. 3.8 in Scenario #2 the non ME/DE part of coding for views executed on one CPU core can run concurrently with processing of non

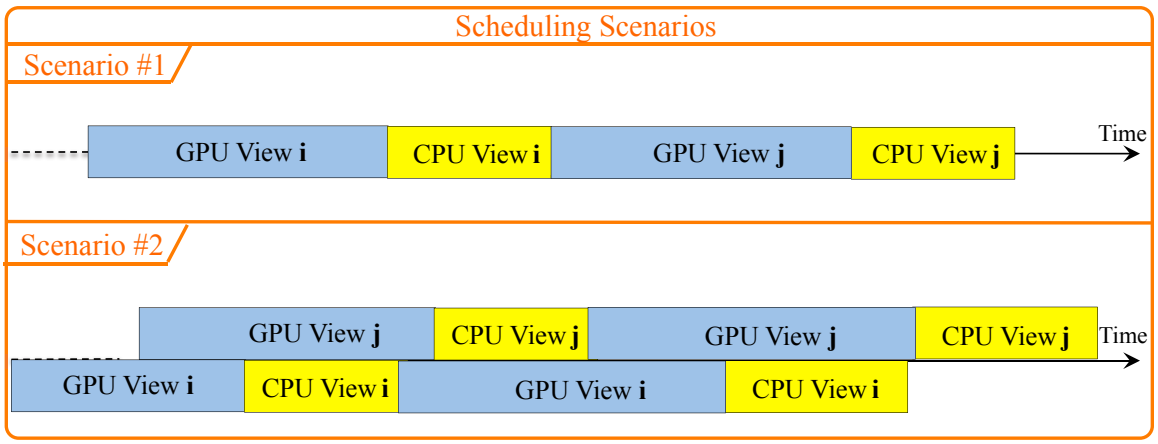


Figure 3.8: GPU Scheduling scenarios

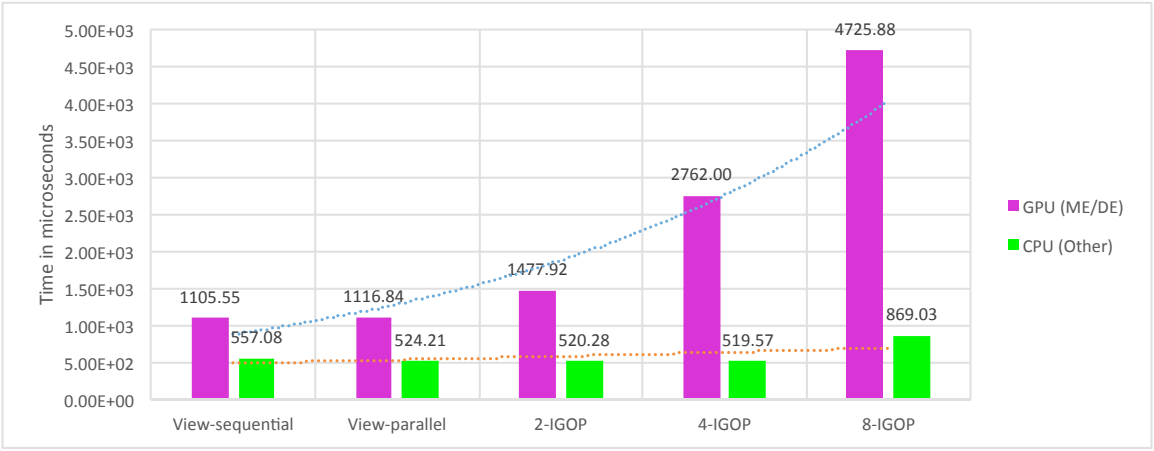


Figure 3.9: Average GPU (ME/DE) and CPU (other) compute time per CU per CPU core for various parallel coding strategies with IBP prediction structure

ME/DE parts from the other views on the remaining CPU cores. It can also run concurrently with the ME/DE processing from the other views on the GPUs.

As the number of views in the multiple-view-parallel, multiple-IGOP scheduling increases from two to four, the contention for GPU resource becomes even more severe, further elongating the overlap in the ME/DE processing execution periods for GPUs. The other parts of coding for four views run on four separate CPU cores concurrently. In going from a 4-IGOP to an 8-IGOP scheme, in addition to elongation in

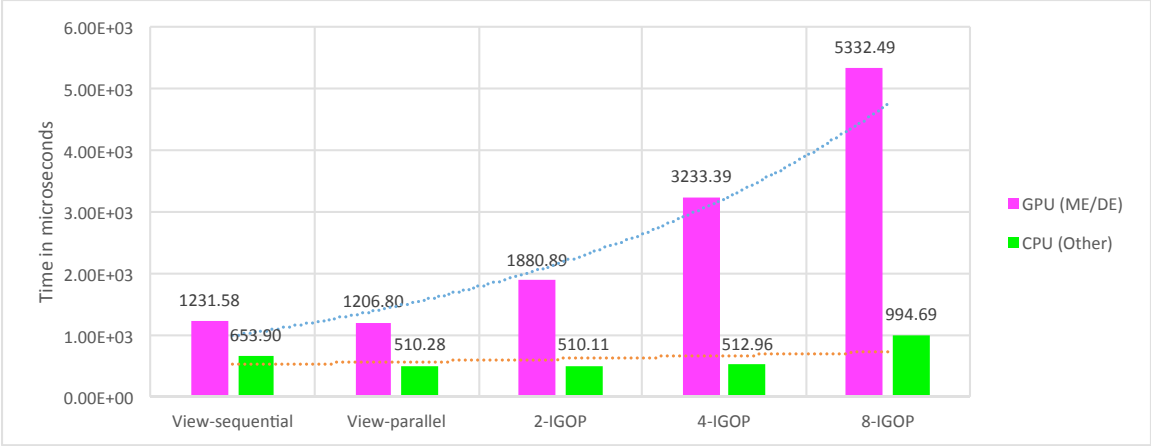


Figure 3.10: Average GPU (ME/DE) and CPU(other) compute time per CU per CPU core for various parallel coding strategies with IPP prediction structure

Table 3.4
Experimental Condition

Hardware:	Michigan Tech Immersive Visual Studio (IVS) Computing Cluster (eight nodes) with one front end, eight compute nodes, each equipped 4 CPU cores (Intel® i7® 4-core CPU i7-3820 @ 3.60GHz with 32 GB DDR3 Memory, and two NVIDIA™ (GTX 680 SLI with 4 GB GDDR5), supported by an eight TB RAID60 storage for computation and visualization. It also features a 40 Gb/s InfiniBand network that serves its computing needs and an gigabit ethernet backend network serves the administrative needs of this cluster.
Operating System:	Rocks 5.4.2 (Maverick) ++ CentOS 5.5
Software:	H.264/AVC MVC extension, test and validation suite JMVC8.5 GOPSize: 8, NumberReferenceFrames: 1, QP: 37 FrameRate: 25, FramesTobeEncoded: 1025, SymbolMode: CABAC, BiPredIter: 2, IterSearchRange: 1 ME/DE Algorithm: DZfast [41],[50], TZsearch [27] Test Sequence ∈ {Ballroom (480p), Exit (480p), Vassar (480p)} in YUV420 format, ME/DE search range [-128,127]/[-128,+127](horizontal/vertical) Intel® MPI library, Intel® C&C++ Compiler

the ME/DE processing time, the processing of the other parts for two views are scheduled on a single CPU core and have to be executed sequentially. The experimental results in support of this observation are provided in the next section.

From the foregoing discussion the average epoch time per view T_{epo_I} for the multiple-IGOP is modeled as,

$$T_{epo_I} = \frac{\frac{T_G}{\beta_G} \times \frac{\mathrm{CT}_I \times I}{G} + T_E \times \left\lceil \frac{I}{C} \right\rceil}{I} \tag{3.4}$$

where T_G and T_E, respectively, are the ME/DE processing time on single GPU (using DZfast) for a single reference frame, and rest of coding functional elements time on a single CPU core (Fig. 3.5). The parameter β_G is the ratio of speedup in going from single GPU to multi-GPU. The parameters G and C, respectively, represent the number of GPU and CPU cores on a compute node.

3.6.3 Platform Level Issues for Sequential TZsearch

Using TZsearch for ME/DE, the entire frame encoding process in Fig. 3.5 is carried out only on the CPU cores. The analysis of coding scenario for TZsearch becomes easy considering our homogeneous nature of the computing grid where all nodes are identical. This creates an homogeneous computing environment with total of 32 identical cores, allowing for up to 32 CPU-based independent frame encodings simultaneously. In this scheme it is possible to take the advantage of the availability of more cores to increase the level of GOP interleaving and achieves further reduction in the encoding time.

In view-parallel encoding, ME/DEs for each frame across multiple views are assigned to one of the compute nodes and processed by one of the CPU cores in that node. The limiting factor to performance is the node coding the view with the maximum weighted sum of temporal and inter-view reference frames at each timing step. For 2-IGOP, workload for each compute node is partitioned into two CPU cores with each

core processing one frame from one view. This should result in overall reduction in the processing time by a factor of two with respect to view-parallel scheme. It should be noted that as views are executed concurrently on different cores, the workload distribution per core remains unchanged from the view-parallel scheme. Therefore, the limiting factor to performance is still the view with the maximum number of weighted sum of references. Going from 2-IGOP to 4-IGOP, the encoding time should further reduce by a factor of two, as all four CPU cores are utilized for processing four views. With maximum of four cores in each node, in going from a 4-IGOP to an 8-IGOP scheme, any change in the speedup can only come from the change in the workload distribution according to (3.2) from one view per core to two views per core in an interleaved fashion.

From the foregoing discussion the average epoch time per view T_{epo_I} for the multiple-IGOP for TZsearch is modeled as,

$$T_{epo_I} = \frac{(T_C \times \mathrm{CT}_I + T_E) \times \left\lceil \frac{I}{C} \right\rceil}{I} \tag{3.5}$$

where T_C and T_E, respectively, are the ME/DE processing time for a single reference frame, and rest of coding functional elements time, on a single CPU core using the TZsearch search algorithm.

3.7 Implementation and Results

3.7.1 Experimental Setup

The proposed strategies are integrated into JMVC and deployed on Michigan Tech's Immersive Visual Studio (IVS) computing cluster. The overall experimental setup and conditions are given in Table 4.3. Message passing interface (MPI) is adopted as the framework for dependency resolution and transferring reference frames between nodes. According to our measurements each transfer between the nodes that involves the transfer of 640×480 frame size, using the 40 Gb/s bandwidth of the Infiniband fabric, takes about 304 μs. This constitutes an insignificant time compared with the cost of the ME/DE processing for a single reference frame.

Three well-known 8-view video sequences ("Ballroom", "Vassar", and "Exit" [44]), in YUV 4:2:0 format, have been selected. To obtain accurate results, each view in each video sequence is concatenated five times to form 128 GOPs, consisting 1025 frames. With 128 GOPs, it is possible to perform 2-IGOP scheduling with two 64-GOP sets from two views assigned to two CPU cores, on a single compute node. For 4/8-IGOP scheduling the assignment sets are 32/16 GOPs from four/eight views assigned to four CPU cores on one compute node. Three prediction structures, IBP, IPP and simulcast are examined.

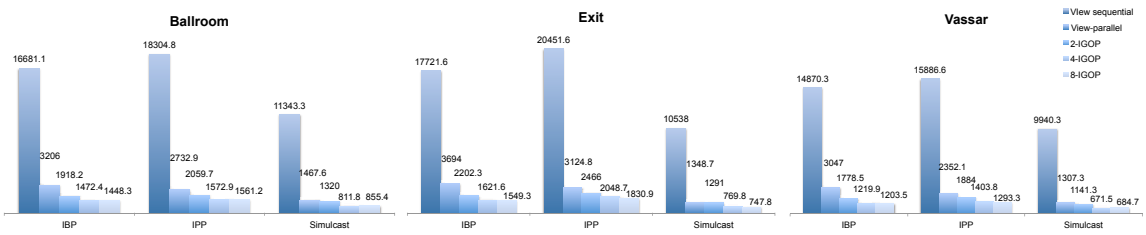


Figure 3.11: Encoding Times for Various Coding Scheduling Strategies for 1025 Frames (128 GOPs) (seconds) Using DZfast

Table 3.5
Speedup Comparison for DZfast (with Respect to its View-sequential Coding)

	view-parallel			**multiple-IGOPs**			
	IBP	IPP	Simulcast	IGOPs	IBP	IPP	Simulcast
Ballroom	5.2	6.7	7.7	2	8.7	8.9	8.6
				4	11.3	11.6	14.0
				8	11.5	11.7	13.3
Exit	4.8	6.5	7.8	2	8.0	8.3	8.2
				4	10.9	10.0	13.7
				8	11.4	11.2	14.1
Vassar	4.9	6.8	7.6	2	8.4	8.4	8.7
				4	12.2	11.3	14.8
				8	12.4	12.3	14.5

3.7.2 Results and Observations for Parallel DZfast

The impact of various workload distribution in the computation time for coding of a CU, for IBP and IPP prediction structures are shown in Fig. 3.9 and Fig. 3.10, respectively. The trends for both prediction structures are identical where the increase in GPU processing time grows exponentially (with powers of two) from view-parallel to 8-IGOP. It should be noted that CPU time for processing the other parts of coding remains constant up to 4-IGOP. Going to 8-IGOP almost doubles the CPU time due to the CPU resource limitation of four cores. In order to handle eight views on a

compute node, pairs of views are sequentially scheduled to a single CPU core.

The experimental results are presented in Fig. 3.11 and Table 3.5. The results approximately track the model in (3.4) with $T_G = 1.7s$, $\beta_G = 1.6$, $T_E = 0.7s$, and CT_I from Table 3.3. First, note that the workload distribution for IBP and IPP structures, as was noted in Table 3.3, play a primary role. It is also noted that IBP and IPP prediction structures in view-sequential coding show similar encoding time because they have similar number of reference frames per GOP (164 and 168 for IBP and IPP, respectively). However, since ME/DE processing is performed on the two available GPUs in parallel, four and three references, for views 1, 3, and 5, respectively, for IBP and IPP take same amount of time. For all other views IBP has only two references versus three for IPP. This results in reduction in the encoding time for IBP with respect to IPP beyond the ratio of $164/168 = 0.98$ as evidenced in Fig. 3.11. The simulcast execution time is expected to be much lower. The speed advantage for encoding time of simulcast comes at the cost of degradation in RD performance [7].

From Fig. 3.11 for view-parallel scheduling the ratio of execution time of IPP over IBP is about 0.85, which is more than 0.75 expected from Table 3.3. This is because computations of ME/DE are executed concurrently on two GPUs in pairs. Therefore, processing time for three references for IPP is only slightly less than four references for IBP. For multiple-view-parallel, multiple-IGOP scheduling the execution times for IBP and IPP are very similar. This is in keeping with actual workload in Table 3.3.

Next the attention is focused on execution times across the scheduling structures. Compared with view-sequential coding, view-parallel achieves significant, five, six and seven fold speedups for IBP, IPP and simulcast prediction structures, respectively. The relative speedup differences from view-sequential to view-parallel coding for the

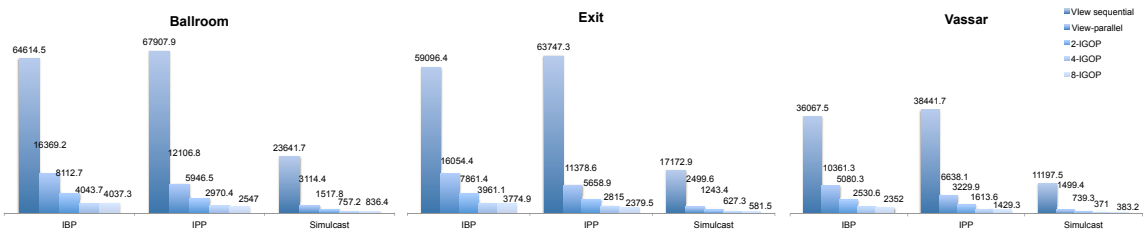


Figure 3.12: Encoding Times for Various Coding scheduling Strategies for 1025 Frames (128 GOPs) (seconds) Using TZsearch

three prediction structures follow the workload distribution in Table 3.3. In view-parallel coding, simulcast achieves a speedup of 7.8 approaching theoretical value of eight. In going from the view-parallel to the 2-IGOP scheduling, reductions in execution time by a factor of about 1.67 and 1.33 are achieved, respectively, for IBP and IPP. For the IBP the speedup is partly due to reduction in the workload per core from four to three, and partly due to execution concurrency between one CPU core for one view and the CPU core and/or the GPU(s) from other views (Scenario #2 in Fig. 3.8). IPP exhibits a lower speed up as there is no change in the workload and all the gain comes from CPU core concurrency.

Further, going from a 2-IGOP to a 4-IGOP scheme, results in similar gains factor for IBP and IPP of about 1.33. Since the reduction in the workload is little (Table 3.3), this improvement primarily comes from CPU core concurrency where up to four views can be coded simultaneously on four CPU cores. However, further interleaving to 8-IGOP yields small gain. That is because the only gain is from small improvement in balancing the workload distribution. There is no additional gain from CPU core concurrency, as two views have to be scheduled to one CPU core, and their executions need to be scheduled sequentially, as evidenced in Fig. 3.9 and Fig. 3.10.

Table 3.6
Speedup Comparison for TZsearch (with Respect to its View-sequential Coding)

	view-parallel			**multiple-IGOPs**			
	IBP	IPP	Simulcast	IGOPs	IBP	IPP	Simulcast
Ballroom	3.9	5.6	7.6	2	7.9	11.3	15.6
				4	15.9	22.7	31.2
				8	15.9	26.5	28.3
Exit	3.7	5.6	6.9	2	7.5	11.2	13.8
				4	14.9	22.5	27.4
				8	15.6	26.6	29.5
Vassar	3.5	5.7	7.5	2	7.1	11.8	15.1
				4	14.2	23.6	30.2
				8	15.3	26.7	29.2

3.7.3 Results and Observations for Sequential TZsearch

The experimental results for TZsearch are presented in Fig. 3.12 and Table 3.6. For view-sequential coding in Fig. 3.12, the ratios of the encoding times of IBP over IPP for three video sequences are between 0.93 and 0.96. These ratios track well with the ratios of weighted sum of temporal and inter-view reference frames in a GOP for IBP and IPP, with α set according to Table 3.1, for video sequences Ballroom, Exit and Vassar.

For view-parallel and 2-IGOP/4-IGOP/8-IGOP encoding in 3.12, the results track the model in (3.5) accurately with $T_C = 5.4s$, $T_E = 0.7s$, and CT_I from (3.2).

Given that CPU cores are the only computation units at work, the overall encoding is limited by the core with the biggest weighted sum of temporal and inter-view reference frames in any given time step. For the range of α values for the three video sequences, the ratios of execution times of IPP over IBP are between 0.93 to 0.96, which are

Table 3.7
Speedup of DZfast over TZsearch

	view-sequential			**view-parallel**			**multiple-IGOPs**			
	IBP	IPP	Simulcast	IBP	IPP	Simulcast	IGOPs	IBP	IPP	Simulcast
Ballroom	3.9	3.7	2.1	5.1	4.5	2.1	2	4.3	2.9	1.1
							4	2.8	1.9	0.9
							8	2.9	1.7	1.0
Exit	3.3	3.1	1.6	4.4	3.7	1.9	2	3.6	2.3	1.0
							4	2.5	1.4	0.8
							8	2.5	1.4	0.8
Vassar	2.4	2.4	1.1	3.4	2.8	1.1	2	2.9	1.7	0.6
							4	2.1	1.2	0.6
							8	2.0	1.2	0.6

significantly more than 0.75 expected from Table 3.3 where $\alpha = 1$ was assumed.

The relative speedup differences for the three predictions structures follow the work-load distribution obtained from (3.2) with appropriate values of α. As expected, going from view-parallel to 2-IGOP and 4-IGOP schemes improves the performance by a factor of two and four respectively. This results is different from that of DZfast where there was contention for the resources of GPU from multiple views executing on multiple cores, and where the speedups were about 1.67 and 1.33 for IBP and IPP.

For DZfast, going from a 4-IGOP to an 8-IGOP scheme resulted in little improvement in performance. However, in the case of TZsearch, due to reduction in the work load distribution on the CPU cores, according to (3.2), it is expected a modest improvement in the performance. However, due to larger values of $\alpha > 3.5$ the improvement is very little for IBP, and only a factor no more than 1.17 better for IBP.

3.7.4 Comparative Discussion on TZsearch and DZfast

Table 3.7 presents the comparative evaluation of DZfast and TZsearch for three prediction structures. As seen for the sequential coding the DZfast is a clear winner by up to factor of 4.3 for IPP. However, as the multiple-view-parallelism increases the DZfast becomes less effective in comparison to TZsearch. For example the speedup of DZfast over TZsearch is no more than 1.7 for the 8-IGOP scheme. The reason for this behavior is the contention for the resources of GPU from the concurrent processing of eight views. Increasing the number of GPUs will reduce pressure on the resources of GPUs and improve the performance of DZfast.

The use of parallel search algorithms such as suboptimum DZfast or optimum GPUfull [50] allows us to set $\alpha = 1$ in (3.1) and (3.2). This has the advantage of allowing the use of a general concept of compute nodes to deal with the parallelism and balance in the workload using a higher level algorithmic abstraction in the suitable unit of the number of references in temporal and inter-view domains, for the various prediction structures in Fig. 3.2.

3.8 Slice-Level and Coding Unit Parallelism

View-parallel and multiple-IGOPs processing schemes for IBP and IPP can be easily extended to lower slice-level parallelism. It can be accommodated by multiple partial reference frame transfers. This is possible as CUs are coded in raster-scan order. This, however, comes at the cost of additional transfer and book keeping overheads. It is easy to see that slice-level parallelism does not change the relative workload distribution presented in Table 3.3.

It is also worthwhile to briefly discuss the influence of implementation platform on slice level parallelism. From the discussion so far it is clear that the limit to parallel processing on a node is four cores where CPU-concurrency loses its effectiveness. It is observed that as a frame is divided into slices and processed in parallel on a node, the computing resources saturate quickly, and beyond four slices, coding schedule become sequential on a node. Therefore, it is easy to conclude that with four (or multiple thereof) slices, the only factor contributing to the performance is, platform independent, prediction structure workload distribution from Table 3.3. This concludes that provided the number of slices are sufficiently large, it is possible to analyze the performance of MVC on computing cluster purely from its prediction structure and workload distribution across the compute nodes.

The provision of wavefront processing of coding units (CU) in HEVC presents an opportunity for parallelization at a finer level of granularity. This, however, requires resources to process multiple coding units in parallel. Similar to the discussion on slice level parallelism, a 4-core CPU limits parallel processing of CUs to four. From the discussion in Section 3.6 it should be noted that to implement the coding of CUs using a fast fine grain massively parallel technique requires allocation of multiple MPAs resources to a compute node to accelerate the wavefront processing.

3.9 Conclusion

By exploiting group of pictures (GOP) parallelism in the multiview coding, in this chapter, the contribution is a multiple-view-parallel, multiple-interleaved GOP (multiple-IGOP) scheduling scheme that will produce a balanced workload. In addition, the resources of multicore CPUs and multi-GPUs are leveraged to extract the maximum performance from a computer cluster. A substantial decrease of overall

encoding time by a factor of 12 is observed with no cost to the bitrate or video quality when compared with the implementation on a single node. The improvement factor of 12 is more than the number of nodes (eight). Furthermore, this strategy decouples the optimizations at the lower levels from those at higher levels, allowing the deployment of any search algorithm for the ME/DE processing.

Chapter 4

Multi Level MVC Encoder Optimization[1]

Standardized in 2014, multiview extension of high efficiency video coding (MV-HEVC) offers significantly better compression performance of around 50% for multiview and 3D videos compared to multiple independent single-view HEVC coding. However, the extreme high computational complexity of MV-HEVC, demands significant optimization of the encoder. Novel optimization techniques at various levels of abstraction. Non-aggregation massively parallel motion estimation (ME) and disparity estimation (DE) in prediction unit (PU), fractional DE and bi-directional ME/DE, quantization parameter (QP)-based early termination of coding tree unit (CTU), and optimized resource-scheduled wave-front parallel processing for CTU, are proposed in this chapter. When evaluated over three views for all MV-HEVC available test sequences, proposed optimization outperforms the anchor encoder by average factor of 5.4 at the cost of 4.4% bitrate (DBR) increase, or equivalent a PSNR degradation of

[1]The material contained in this chapter was previously published in "*2016 IEEE Data Compression Conference*"

0.12 dB.

4.1 Introduction

High efficiency video coding (HEVC)/H.265 [6] is the latest state-of-the-art video coding standard, providing up to 50% better compression over its predecessor advance video coding (AVC)/H.264 [3] to satisfy ever growing demands for higher resolution in videos such as $4K$ and $8K$ [25]. The multiview extension to HEVC (MV-HEVC) was defined in the Annex G of HEVC/H.265 [6] [73] [5] in 2014. Common applications of multiview coding (MVC) are free view TV, 3D movies/TV, and immersive teleconferencing [25] [26]. In these applications, multiple cameras commonly arranged in linear, grid or arc formation are deployed to capture the same scene simultaneously. In addition to exploiting temporal similarity using motion estimation (ME) and motion compensation, MVC exploits inter-view similarity using disparity estimation (DE) and disparity compensation, achieving notably higher coding efficiency compared with coding of multiple views as separate video streams. The ME algorithms designed for temporal prediction can, with little or no modifications, be applied to DE for inter-view prediction.

The profiling of AVC/H.264 MVC in our previous work [50] showed that 99% of execution time is spent on integer ME/DE and its sole optimization is enough to gain significant speedup. However, as the profiling result for MV-HEVC/H.265 in Table 4.1 shows, the execution time is spread across many functional modules where integer ME/DE contribution to the overall coding execution time is only 47%. Fractional ME/DE and bi-directional ME/DE also make a significant 15% contribution to the execution time. A significant 38% of execution time is contribution from the other functional modules such as intra-prediction, discrete cosine transform (DCT)

Table 4.1
Execution Time Profiling of MV-HEVC/H.265 (HTM) 16.2 at QP=32 for multiview video sequence Shark

Integer ME	Fractional ME	Bi-directional ME	Other
47%	5%	10%	38%

and context-adaptive binary arithmetic coding (CABAC) . Therefore, sole optimization of integer ME/DE in MV-HEVC/H.265 while still needed, is obviously not enough. This chapter proposes several low level techniques that reduce execution time of integer, fractional and bi-directional ME/DE. Further, several high level optimization techniques that affect all or a significant number of functional modules in MV-HEVC/H.265 is considered. For example by selective evaluation of coding units execution of all required functional modules are skipped. Another available tool is, of course, concurrent coding of functional modules.

There have been efforts to improve the execution performance of HEVC single view coding [21] [74] (and references thereof). However, these efforts only deal with single view HEVC, and thus do not take the characteristics of inter-view correlation into account, and are, therefore, inadequate for MV-HEVC. This chapter presents three major contributions that reduce the computational complexity of MV-HEVC. While MV-HEVC has been the focus of complexity reduction in this chapter, the contributions in this chapter provide significant benefit to the single view HEVC as well. In this chapter, a methodology for designing a novel massively parallel architecture (MPA) based ME/DE algorithm from a completely new perspective is explored. A single instruction multiple data (SIMD) approach is proposed for sub-pixel and bi-directional ME/DE. Next, quantization parameter (QP)-based early termination of coding tree unit (CTU) is proposed, where the coding units (CU) below a certain depth are not processed, depending on values of their QP. This strategy exploits the exponential decrease in motion residuals with increase in QP to skip ME in prediction units (PU) within the CUs at a certain CTU depth. Finally, on the implementation

platform, a resource-optimized multi-threaded execution scheduling for the wave front processing (WPP) for the implementation on a multicore processor is proposed.

This chapter is organized as follows. Section 4.2 briefly presents the relevant concepts in HEVC and MV-HEVC. Section 4.3 presents a survey of related works in improving the runtime performance of HEVC video coding. Section 4.4 presents our efforts on scalable massive parallelism and the opportunity that it provides for fast ME/DE algorithm from a new perspective. Section 4.5 presents our proposal for QP-based early termination of CTU. Section 4.6 discusses the proposal for an optimized WPP scheduling and implementation. Section 4.7 concludes the chapter.

4.2 High efficiency Video Coding (HEVC) and its Multiview Video Coding (MVC)

HEVC inherits its block based hybrid coding model from AVC. Among the innovative features of HEVC [6], increased flexibility of block partitioning for prediction and transform coding have contributed to more than half of the average bitrate savings. In HEVC the CTU size ranges from 16×16 to 64×64, the CU size from the size of CTU down to 8×8, and the PU size from the size of the CU down to $4 \times 8/8 \times 4$. For the transform unit (TU) (required for DCT) the size extends further from CU all the way down to 4×4 [73] [5]. However, HEVC gains exceptional compression efficiency at the cost of higher computational complexity. Fig. 4.1 shows all possible PU and CU partitioning modes for the CTU of size 64×64 in HEVC. There are four possible CU modes (8×8, 16×16, 32×32 and 64×64), amounting to a total of 85 distinct CUs ($64 + 16 + 4 + 1 = 85$). There are 24 PU modes, amounting to a total of 593 distinct PUs ($320 + 208 + 52 + 13 = 593$).

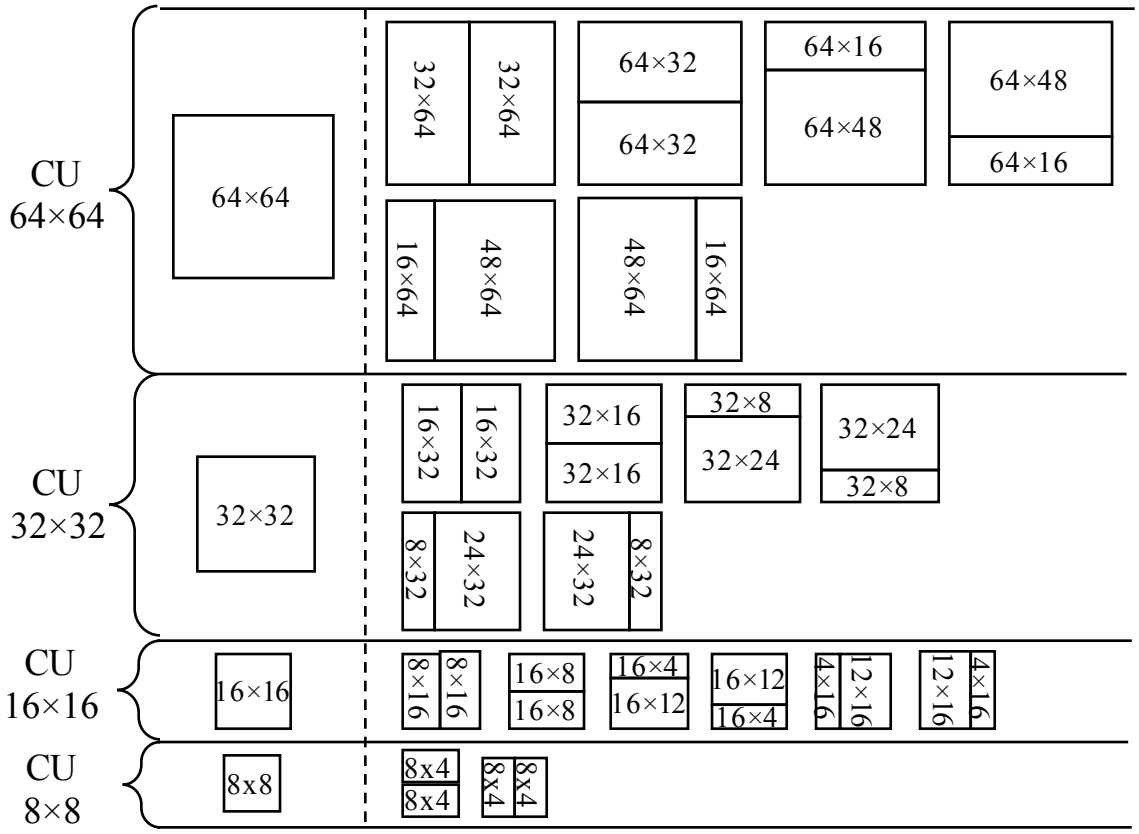


Figure 4.1: Enumeration of all CU and PU modes in a CTU

The incorporation of MV-HEVC in HEVC is achieved through a high-level syntax extension, which is shared with other multi-layer extensions (scalable extension (S-HEVC) and 3D-HEVC). MV-HEVC is designed to allow reuse of existing HEVC encoders and decoders with no major modifications [73]. The design principle of MV-HEVC follows that of the multiview extension of AVC. The higher compression efficiency is achieved by exploiting redundancy between views of the same scene (inter-view similarities) as shown in Fig. 4.2 (a).

Fig. 4.2 (b) presents a prediction structure for an MV-HEVC system. All pictures associated with the same capture time instance are stored in an access unit (AU) and have the same picture order count (POC). In MV-HEVC a picture with all its color-components is referred to as *layer*. The first view or base layer within a AU

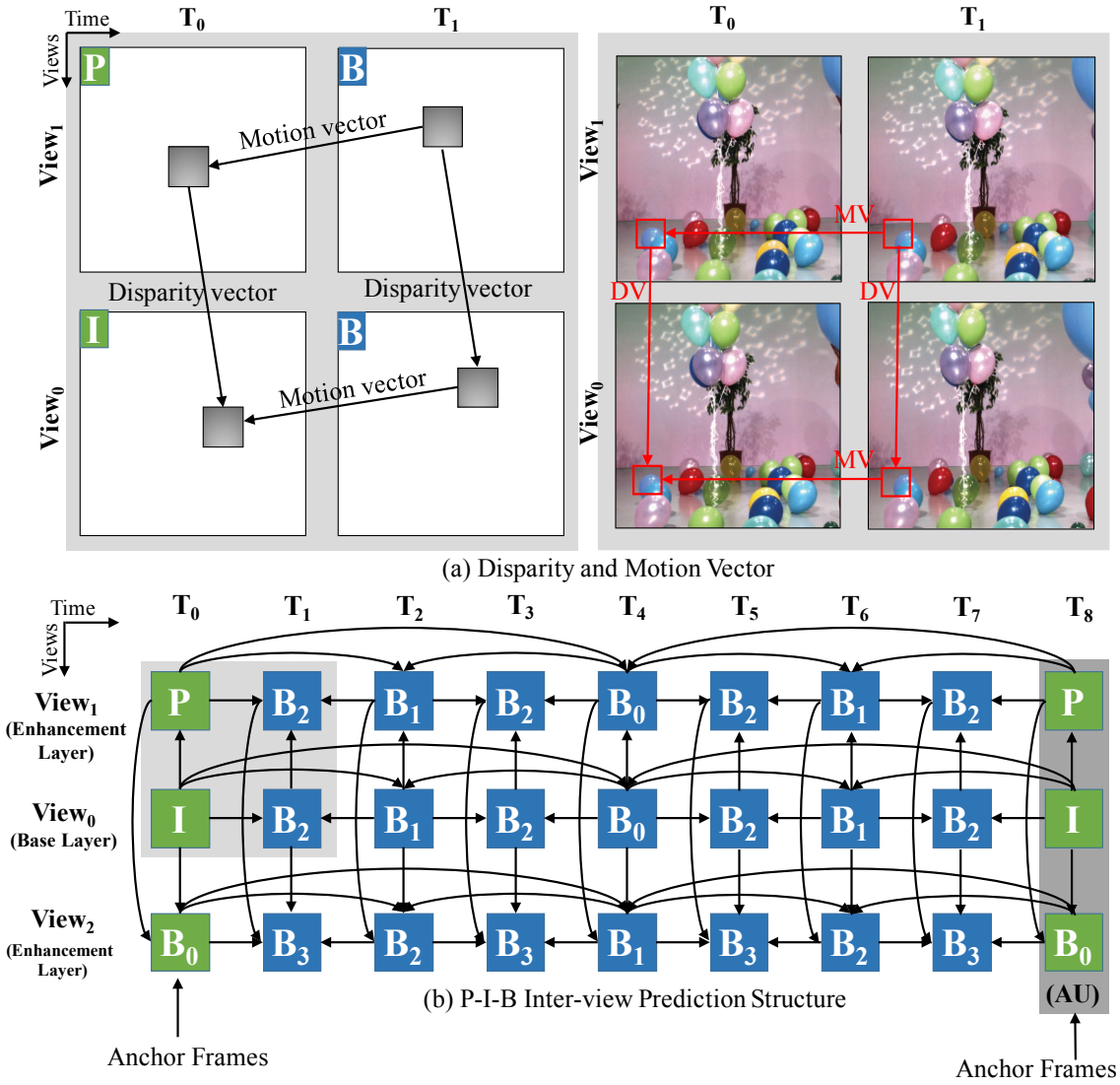


Figure 4.2: Multiview prediction, (a) motion and disparity and vectors (ME/DV), (b) P-I-B inter-view prediction structure

must conform to HEVC single-layer coding specifications. The layers from other views, within the same AU, following the base layer are referred to as enhancement layers or nonbase layers, and must conform to specifications for multiview extension. Each layer follows a temporal hierarchical B-frames prediction structure [7] where eight frames are grouped as a group of pictures (GOP). Each GOP is fenced by two consecutive intra-coded frames (I-frames).

As seen from Fig. 4.2 (b), in addition to having a temporal hierarchical B-frames prediction structure that uses the frames within the same view for referencing, MV-HEVC requires an inter-view prediction structure (not defined by the standard). The addition of inter-view frames of the same time instance in the reference picture list (RPL) for the prediction in the enhancement layers results in a significant bitrate saving for these layers [73] [50] [75]. Inclusion of inter-view prediction structure is enabled through the flexible reference picture management capabilities of HEVC. The complex reference hierarchy in the prediction structure presented in Fig. 4.2 (b), is formed to fully exploit various temporal and inter-view redundancies. This prediction structure is built through deeper temporal layering, increasing number of reference frames in the temporal and inter-view domains, and larger decoded picture buffer, all leading to higher computational complexity and more storage space.

4.3 Related Work

4.3.1 ME/DE Algorithm

The ME/DE is usually carried out through an iterative block matching process where a vector is computed between two blocks of pixels, in the current and reference frames. The best match in the rate-distortion (RD) sense is selected and signaled in the bitstream. Block matching even in the single view video coding is a time-consuming process, and as said, amounts to up to 50% of the total encoding time.

To accelerate the block matching in ME/DE many sub-optimal techniques have been proposed for AVC that are generally applicable to HEVC. An iterative Hexagon search was proposed in [8] to enhance the RD performance of the earlier diamond

search [9]. More sophisticated search algorithms use multiple simple search patterns and local correlations to further improve RD performance. The technique in [10] proposes unsymmetrical-cross multi-hexagon-grid search (UMHexagonS). The work in [11] gains further improvement through enhanced predictive zonal search (EPZS). In high efficiency model (HM) [28] and its multiview extension (HTM) [4], reference software suites, the method of TZsearch, a variation of diamond search, is adopted. Compared with AVC, the computational complexity of ME in HEVC has increased considerably due to higher hierarchical complexity associated with the CTU. Much higher complexity also results from the very high video resolutions that are targeted by HEVC, (and by extension MV-HEVC). The complexities associated with MV-HEVC demand new ME/DE techniques beyond TZsearch.

The availability of massively of MPA computing platforms [34] [35], has provided an excellent opportunity to accelerate application with high data-level parallelism such as ME/DE. In our previous works, a range of high performing ME/DE schemes for multiview extension of AVC are proposed to significantly improved execution time, with negligible impact on the RD performance [50] [41]. The recent GPU-base algorithm for HEVC in [21] employs two different techniques for ME processing of the larger and smaller PUs. A parallel GPU/CPU based ME processing was proposed in [74] for HEVC.

All aforementioned ME/DE algorithms share a common characteristic, the calculation of sum of absolute differences (SAD) for all PUs are carried through an aggregation of SADs from the smaller PUs to compute the SADs for the larger PUs (see [50] for details). This approach while avoids redundant calculations of SADs has three drawbacks. First, the aggregation process imposes a strict hierarchical dependency on the processing of PUs. This removes the opportunity to skip ME/DE process for a PU with content that has little or no motion. Second, parallel SAD aggregation is unable to use motion vectors (MV) or disparity vectors (DV) from the neighboring PUs, (as

specified in the standard), during the calculation of cost function, resulting in some loss in RD performance [50]. Third, as will be seen in Section 4.6 the implementation of aggregation of PUs in WPP (introduced for the first time in HEVC), on a typical multicore CPU/GPU heterogenous platform [76] significantly hampers the coding time performance.

4.3.2 Fast Mode Decision

A rich choice of hierarchical partitioning modes within the CTU is the main reason for higher coding efficiency as well as the high computational complexity in HEVC. To improve the execution time of HEVC requires additional optimization steps beyond the efficient processing of ME, through early termination of partitioning within CTU. The scheme in [18] stops further partitioning of CU into smaller CUs if the skip mode has been selected. An early termination scheme where the partitioning mode with largest PU size is first checked, was proposed in [19]. If in this mode PU produces a coded-block-flag equal to zero, the processing of all PUs within this CU is skipped. The work in [20] improves upon this termination scheme by halting the processing of all other PUs if both the MV difference and coded-block-flag are turned out to be equal to zero. Further, the latest related work on HEVC coding in [21] proposed a ME technique that skips the processing of all CU of size 8×8. For the remaining larger CUs all 17 possible symmetric partitioning modes are evaluated. Three modes with lowest costs collectively determine the early termination decision for the processing of CTU subtrees.

Above algorithms are estimated to yield a speedup factor of about 1.6 to 3 with varying loss in the RD performance. There are also fast mode decisions proposed for

intra-prediction [22]. However, intra-prediction consumes very little time in comparison with ME/DE processing.

These efforts reveal the potential of reducing encoding time by appropriate skipping of some of CUs and PUs. However, the increasing number of views in MV-HEVC brings significantly more inter-prediction for each PU within a CTU, potentially slowing down the processing of PUs for the existing algorithms.

4.4 Fast Massively Parallel Motion/Disparity Estimation (ME/DE)

4.4.1 Architecture and Programming Model

An MPA features a large number of processing cores (on the order of 1000) which are organized in a hierarchical fashion, in an array of *multi-processors* (MP), with each MP housing a number of processing cores (typical 32 to 48 cores). The MP cores operate in SIMD mode or more appropriately single program multiple data (SPMD). When an MP becomes available, one of the thread schedulers dispatches a stream of identical instructions for an entire set of program threads to the cores in that MP [77]. In addition, MPA gains unprecedented performance through high bandwidth memory access in the order of Giga bytes per second (GB/s). The memory hierarchy of MPA features a large number of registers, MP local memory, L1 and L2 caches, and dynamic random access memory (DRAM), listed in the order of the fastest to the slowest in terms of access latency. In ME/DE processing the fast storage such as registers and L1 cache can be leveraged to aggregate SADs [50].

Memory access latency variation on the MPA is large, and therefore, plays a significant role in the parallel algorithm design. Arithmetic operations incur a latency of about 20 cycles on average, if the operands are available in the registers [77]. The corresponding latency when the operands are on the DRAM is about 400 to 800 cycles. A good measure of computational efficiency of a parallel algorithm on an MPA is the compute-to-memory-ratio (CMR) which is represented as the number of arithmetic operations per a DRAM access. In order to increase CMR of an algorithm requires maximum data reuse once it is loaded from the DRAM to the local memory attached to an MP. Each partial accumulation of absolute difference operation for ME/DE consists of two read memory accesses (reference block pixel and prediction block pixel) and three arithmetic operations (subtraction, absolute value and addition). To improve the CMR of SAD in ME/DE on the MPA the number of memory accesses needs to be reduced. It should be noted that due to large cache size the CMR is not a significant issue for the CPU-based algorithms.

The MPA platform used is NVIDIA GPU with compute unified device architecture (CUDA) programming model as shown in Fig. 4.3. All copies of the parallel program (*Threads*) execute the same set of instructions, however, on different data. Threads are further grouped into thread-blocks. Thread-blocks are in turn placed on a grid. Thread-blocks are executed on the GPU's *streaming multi-processors* SMP, with a minimum of 32 threads per SMP (*a thread warp*) executing simultaneously. *Occupancy* is used as a measure for parallel execution concurrency on the GPU architecture and is represented as the number of threads simultaneously residing on a SMP. It mainly depends on total number of threads and thread-blocks, and allocation of resources for each thread and thread-block. Higher occupancy generally leads to significantly better performance and is considered a crucial design factor.

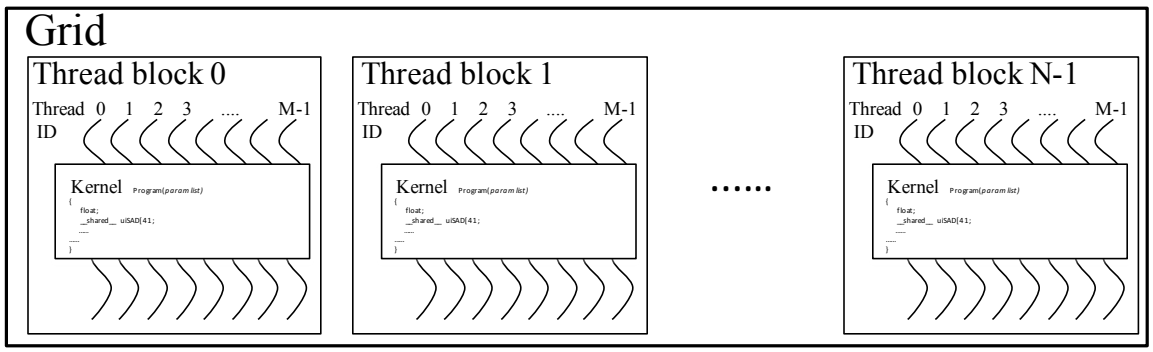


Figure 4.3: Threads, blocks, and grid configuration

4.4.2 Aggregation Parallel ME/DE

The SAD evaluation in the ME/DE can be divided into two categories. The full block matching (FBM) can use the method of *aggregating SAD evaluation* (ASAD) as shown in Fig. 4.1, where all pixel-wise absolute differences for all locations are evaluated and progressively aggregated to form the SADs for all PUs within a CU and CTU, avoiding any redundant computation for SADs.

The development of parallel ME/DE algorithms have generally followed the ASAD approach due to its ease of implementation on the massively parallel architecture (MPA) [50] [41] [21] [74]. In this case PUs of the smallest size can be computed in parallel and aggregated in a hierarchical fashion. This, however, comes at the cost of reduction in the RD performance. This is because in the parallel PU processing the MV/DV information from the neighboring PUs to minimize the ME cost function are not available *a-priori*. The cost in RD performance was of minor concern in AVC with a much smaller macroblock size of 16×16 [50]. However, in HEVC with the CTU of 64×64 this cost cannot be ignored.

In addition, despite of seemingly large number of cores, a massively parallel ASAD

based FBM can easily saturate the resource of an MPA [50]. To reduce the likelihood of saturation of resources on the GPU, in [50] a fast massively parallel ME/DE search algorithm to significantly reduce the complexity of ASAD using an adaptive search region is proposed. However, in HEVC the aggregation hierarchy (Fig. 4.1) is much deeper than the AVC (8 versus 4). This corresponds to parallel ME/DE cost evaluations for 593 PUs compared to only 41 in AVC. This indicates that employing ASAD method would always saturates the resources of the GPU, even if our previous method in [50] is applied. Therefore, it is desirable to explore a good method that provides flexibility to simplify the ME/DE cost evaluations for certain PUs whose contents have remained static.

4.4.3 Non-aggregation Fast Massively Parallel ME/DE

The objective is to seek a fast massively parallel solution through the method of *Independent SAD evaluation* (ISAD) for the search algorithms, where each PU within a CU follows a separate search in the search region without reliance on the aggregation of SAD from the smaller PUs within the same CU and CTU. The technique of ISAD can significantly reduce the number of ME/DE cost evaluations from its maximum of 593 by simplifying the ME/DE process for the PUs that have little motion content. Further, the method of ISAD can totally avoid the ME/DE cost evaluations for 168 PUs that correspond to HEVC asymmetric partition modes (see Fig. 4.1), which are rarely invoked. Further, ISAD allows use of the MV/DV information from the neighboring PUs to minimize the ME cost function, because the PUs within a CU are coded in the right order as specified in the standard [6]. To avoid the drawbacks of ASAD based approaches, for the first time, this work proposes a parallel fast ISAD algorithm (*Predicate Algorithm*) that is based on exploitation of inter-pixel similarity in a frame.

4.4.3.1 Predicate Algorithm for Motion/Disparity Estimation

With hierarchical nature of CTU in HEVC, where a large number of PUs of various sizes exists, it is possible to take the advantage of the algorithmic flexibility of ISAD to substantially reduce the complexity of ME/DE for PUs with little or no motion. To achieve this, a preprocessing predicate is inserted before initiating a high accuracy, high computational complexity ME/DE using the FBM technique. The expensive FBM is avoided if the predicate produces a MV/DV that is within an user specified range (*iRaster*). The computational overhead of predicate algorithm is significantly lower than the subsequent high accuracy FBM, resulting in much saving in the average search time. The proposed ISAD-based predicate algorithm for complexity reduction in the ME/DE of PUs is much more flexible and content-adaptive, offering trade-off between accuracy and computational complexity. Algorithm 1 presents the details of the proposed scheme.

The predicate scheme, by taking the advantage of the correlation between the four neighboring PUs, computes the cost associated with points obtained from their MVs/DVs in addition to the cost for the point for the zero displacement MV/DV. This is followed by the check of another 20 search points through a concentric diamond search around the center of the search area at steps of one, two, and four. Among all these 25 search points the one with the lowest cost has its MV/DV compared with the *iRaster* threshold. If the MV/DV associated with this point is larger than *iRaster* threshold a FBM is initiated. Limiting the steps size for the diamond search to four makes is found to yield a good trade-off between RD and execution time performance. Table 4.2 shows the RD performance with *iRaster* set to 1, 2 and 3, for four MV-HEVC sequences for a QP value of 32. The experiments are carried out at a commonly used mid-range QP value of 32. The RD performance of TZSearch is also presented for the purpose of comparison. As can be seen, significant skip rate

Algorithm 1: Predicate Based Algorithm for Motion/Disparity Estimation

Input: Search region, S; Prediction Block, C; Skip threshold value, $iRaster$; Search width and height, $(SRWidth, SRHeight)$; Neighbors and zero MVs/DVs, $(MV_A, MV_B, MV_C, MV(0,0))$

Output: Motion Cost, $Cost$; Motion/Disparity vector, MV

Initialization: $Cost = \infty$; $MV = (0,0)$
(Start Predicate)
for $tmpMV \in (MV_A, MV_B, MV_C, MV(0,0))$ **do**
    $tmpCost =$`getCost`$(tmpMV, S, C)$;
    if $tmpCost < Cost$ **then**
        $Cost = tmpCost$; $MV = tmpMV$;
    end
end
$iDist = \{1, 2, 4\}$;
$(lCost, lMV) =$ `DiamondSearch`$(iDist, MV, S, C)$;
(End Predicate)
if $(lMV.X > iRaster) || (lMV.Y > iRaster)$ **then**
    $CostArray =$ `gpuScaleFastKernel`$(SRWidth, SRHeight, S, C)$;
    $(Cost, MV) =$ `MinimalCost`$(CostArray)$;
else
    (Skip full block search for this prediction block)
end

with minimal to no RD performance loss is achieved at $iRaster = 1$. For all test sequences, going from $iRaster = 1$ to $iRaster = 2$ and $iRaster = 3$ further increases skip rate, with little sign of significant loss in RD performance. $iRaster = 3$ is used all simulation. This indicates that the predicate algorithm is effective in skipping unnecessary work in ME/DE.

The fraction of PUs that do not undergo ME/DE using the FBM in the predicate scheme varies with the content in MV-HEVC multiview video. The average skip rates are 92%, 88%, 97%, and 93% for Balloons, Kendo, PoznanHall2, and PoznanStreet, respectively. For a more dynamic MV-HEVC video sequence such as Kendo the PU skip rate is lower.

Table 4.2
Rate-distortion (RD) performance at three *iRaster* values for MV-HEVC test sequence at QP=32

Video Sequence	iRaster	PSNR	BR (Kbps)	Skip Rate
Balloons	1	41.13	882	90%
	2	41.13	883	92%
	3	41.13	881	92%
	TZSearch	41.13	879	
Kendo	1	41.90	1092	85%
	2	41.90	1094	88%
	3	41.90	1096	88%
	TZSearch	41.89	1090	
PoznanHall2	1	42.43	790	95%
	2	42.43	794	97%
	3	42.43	793	97%
	TZSearch	42.43	791	
PoznanStreet	1	38.78	2526	90%
	2	38.78	2534	93%
	3	38.78	2535	93%
	TZSearch	38.78	2519	

4.4.3.2 Inter-pixel Similarities and ScaleFast Search

Optimization guidelines for the MPA provided earlier in this section suggest that a good parallel ME/DE algorithm must focus on reducing the computation cost while maintaining a regular memory access pattern to fetch the pixel values. The fast CPU-based simple fast search algorithms such as diamond search [9], or more complex search algorithms such as UMHexagon [11] unfortunately cannot be efficiently parallelized on the MPA platform due to irregular prediction chain and memory access patterns.

To maintain a regular memory access pattern and reduce the number of computations on the MPA, an improved ME/DE scheme that is based on the use of *inter-pixel* similarity is introduced. In addition to the spatial and temporal similarity across the

PUs, there is a high correlation between a picture and its lower resolution versions. The number of memory accesses and computations can be reduced by a factor four, if allowing only the participation of every alternative pixel in both directions in a PU in the evaluation of SAD. Note that in this scheme the search region, and therefore, the number of points searched have not changed. However, for each point searched, the number of absolute difference computations has scaled down. Our experimentation over several video sequences has shown that the MVs/DVs for the best search points for the original PU and its scaled down version are very similar. It is possible to continue along this path by requiring the participation of every four and eight pixels in each direction in the computation of SAD. Fig. 4.4 presents the effect of down-scaling of a PU by factors of two, four and eight on the visual content of the picture. The red box in Fig. 4.4 identifies the block with best match in the reference search region. As can be seen the similarity between the MVs/DVs in the original PU and its factor of four and eight scaled down versions are still maintained, due to inter-pixel similarity. The ME at different resolution is also reported in [78]. Three downsampling methods, *left-top*, *discrete wavelet transform*, and *averaging filter* are presented in [78]. The downsampling in [78] was used to enhance the ME accuracy, whereas in this work the technic is employed to improve execution performance. The low pass filtering is skipped for the sake of maximizing execution performance. Our results shows that use of this technique has little to no effect on RD performance.

The scaling factor trades computation time for the RD performance and can be specified as an user option. It should be noted if a scaling factor of eight is specified, a PU with a side equal to four is scaled by a factor of four in that side. For example a PU of 8×4 is scaled by a factor of eight and four. The consequence of this nonuniform scaling across the PUs when the scaling factor is eight or higher requires normalization of SADs to account for this non-uniformity during the evaluation of the cost function. Another implication of nonuniform scaling, as will be seen in the next section, is that it loses its effectiveness in reducing the number of computation in SAD for the PUs at

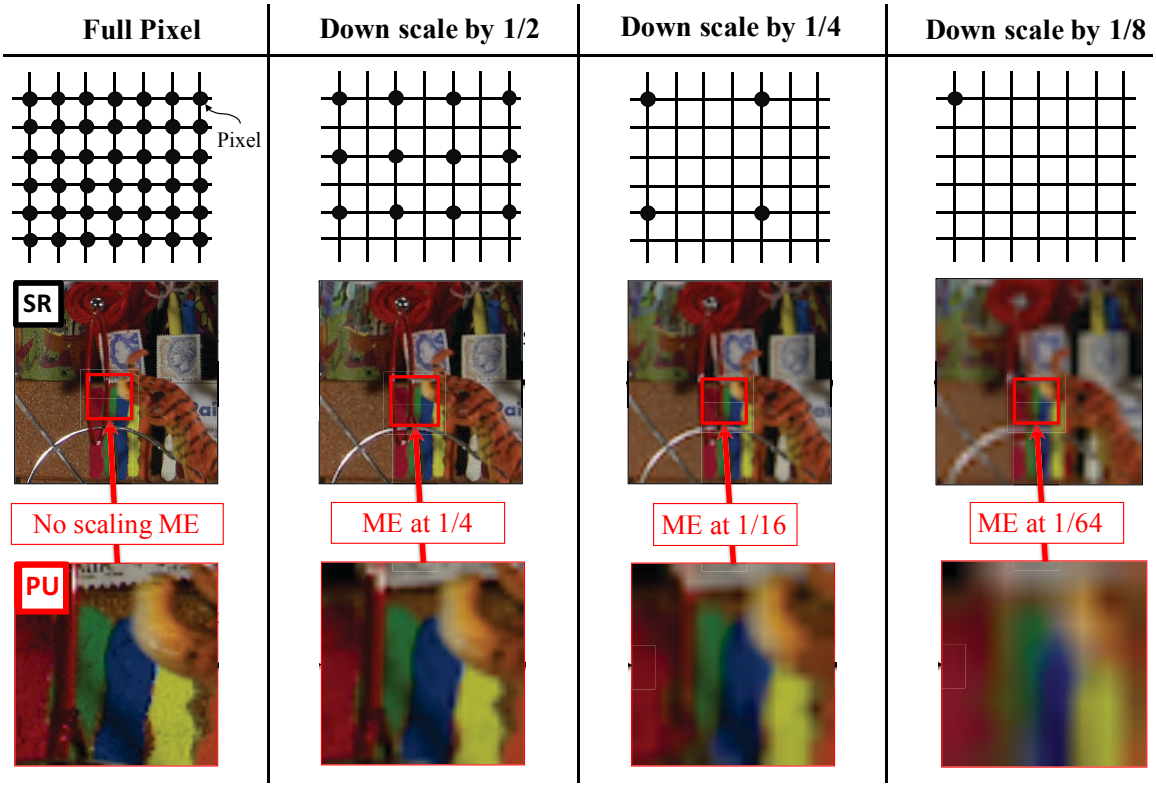


Figure 4.4: Motion/Disparity (ME/DE through the exploitation of inter-pixel similarity

the bottom of CTU hierarchy that have one of their sides equal to four (for a scaling factor of eight).

4.4.4 Architecture Optimization

Next discussion is focused on the GPU architecture optimization for a highly parallel ISAD-based algorithm on the MPA.

Device configuration: To tune the performance of FBM in the proposed predicate algorithm, the GPU device is configured to match the search window size. Each thread within a thread-block is dedicated to the computation of SAD for a PU in one

search location. All threads in a thread-block process a row of search locations in the search window. thread-blocks collectively process all the rows of the search window. For example, for a search window of ±64, there are 128 threads within a thread-block and 128 thread-blocks within the grid. This configuration has three advantages: 1) regardless of the size of the PU, a total of 128^2 threads is enough to fully occupy any modern MPAs; 2) it scales well with the search window size without affecting occupancy (described earlier) ; 3) it ensures reduction in the workload for a thread with the proposed pixel down-scaling ME/DE.

With the chosen configuration an occupancy of 4 thread-blocks per SM is achieved, where typically the performance of the GPU saturates. The CMR, for a PU of size $N \times M$, and scaling factor of SF for the *ScaleFast*, a search window width of SR, is obtained as,

$$\text{Computation} = \frac{3 \times N \times M \times \text{SR}}{\text{SF}^2}$$
$$\text{Memory Access} = \frac{M \times \text{SR}}{f^2}$$
$$CMR = \frac{\text{Computation}}{\text{Memory Access}} = 3N$$

The factor of 3 refers to the number of operations (subtract, taking absolute value and add) involved in the calculation of single SAD. Each thread calculates the SAD for PU at one search location amounting to 3×N×M. A thread-block collectively calculates a total of SR number of locations. A single SAD calculation requires the reading of a pixel in the search window and a pixel in the PU. However, pixels in a PU are shared among all threads and thus are loaded onto the fast shared memory all at once. The number of memory accesses comes from reading the reference pixels in the search window amounting to M×SR. Note that the CMR solely depends on the

width of the PU (scaling factor cancels each other). This indicates CMR is unchanged for PU as scaling factor increases (to reduce computation). The typical values of N are 64, 32, 16 and 8.

Cost reduction: The evaluation of SADs is followed by the Lagrangian RD cost optimization to select the MV/DV with the minimum cost. This optimization can be performed completely on the GPU immediately following the computation of all SADs. However, the device configuration for the SAD evaluations are not best for the computations for the cost optimization. Alternatively, all Lagrangian RD cost optimization can be performed on the CPU after the transfer of required data from the GPU to the CPU memory. In our implementation the best approach to perform a partial cost optimization on the GPU and carry the rest on the CPU. In this approach all row-wise cost reduction are performed on the GPU maintaining the same device configuration. The row-wise reduction is carried out by 128 threads in the thread-block, via the binary-tree reduction in only seven steps. All partial cost values are next transferred to CPU for column-wise cost reduction. In this way the amount of data movement between the GPU device and the CPU memory is reduced by a factor of 128, resulting in significant reduction in the transfer time. This also eliminate the high latency associated with the relaunch of the new kernel on the GPU when this latency is significantly higher than the execution time of the kernel.

Motion cost and vector byte packing: SAD cost varies with QP. However, from our observation, from a large number of trials, it requires no more than 20 significant bits. Hence, the allocation of 24 bits is more than sufficient to cover a wide range of QPs. Even if the actual cost requires more bits, saturation of SAD cost to 24 bits does not degrade the RD performance as a MV/DV with such a high cost $(2^{24} - 1)$ is most unlikely to be the best choice in the final cost optimization step. On the other hand, the storage for the MV/DV is limited by the search window size. For the implementation employed the row-wise partial reduction on the GPU device requires

only maintaining the record of the MVs/DVs along the row. With eight bits of storage a search window as large as ±128 can be accommodated. To reduce the memory transfer between the GPU device and the CPU, a 32-bit packed data structure is employed where the most-significant three bytes are allocated for the SAD cost, and least significant byte for the MV/DV. On the CPU the packed data structure are unpacked to do the final column-wise cost reduction.

Other optimization measures: 1) the memory transfer is further reduced by passing all parameters to the GPU device kernel function through registers; 2) all RPLs are allocated only once before start of the encoding and de-allocated only at the very end of the encoding; 3) local shared memory has declared to be accessed through name aliases to serve a dual purpose, storing the PU pixels and later on SAD values.

4.4.5 SIMD-Parallel Sub-pixel and Bi-direction ME

The work in [74] achieves fast sub-pixel motion estimation by skipping search locations that incurs RD performance loss. In this work SIMD acceleration is introduced to evaluate all search locations and mitigate the loss of RD performance.

Massively parallel processing on MPA is ideal for integer ME/DE due to the existence of a large search region, with each core responsible for up to thousands of SAD computations. However, for sub-pixel and bi-directional ME/DE, the number of search points for a PU is a tiny fraction its integer ME counterpart, and thus not suitable for processing on MPA. Sub-pixel and bi-directional ME/DE are also not suitable candidates for multi-threading on multicores as the small processing workload does not justify the significant execution overhead. On the other hand, streaming SIMD extensions (SSE) [79] instructions set offered by all modern CPUs that allow

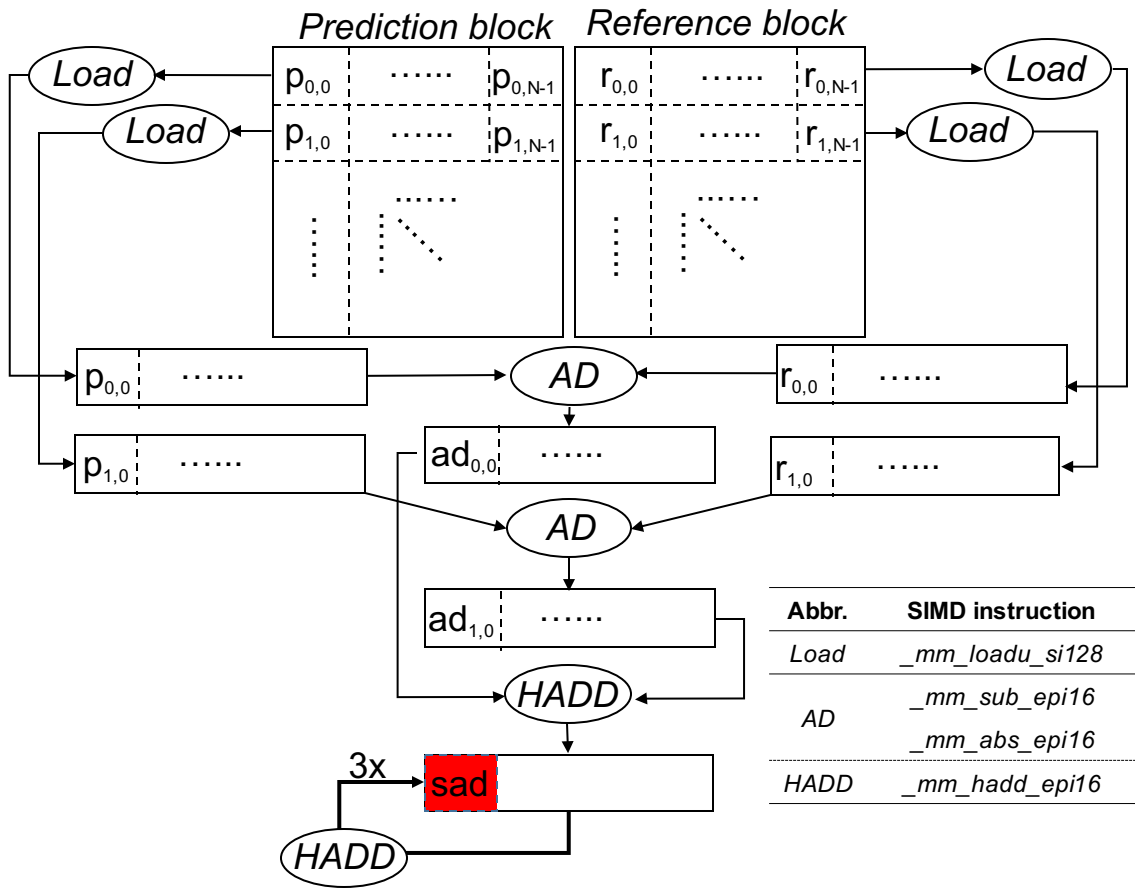
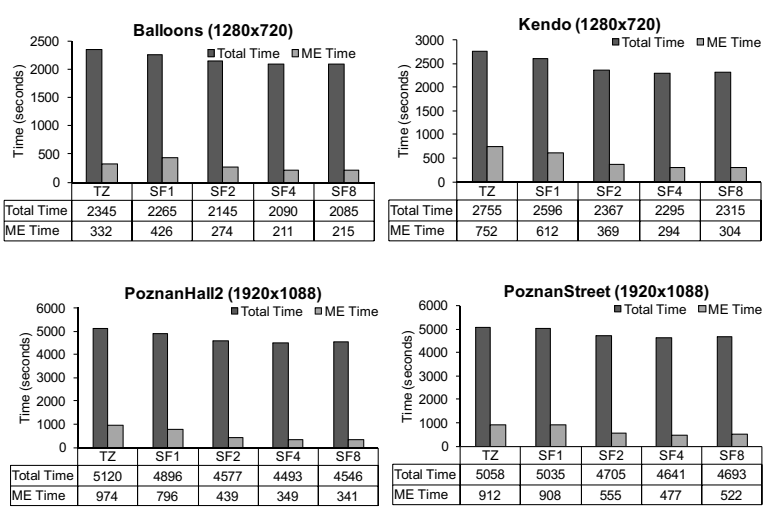


Figure 4.5: Single-instruction-multiple-data (SIMD) SAD calculation implementation.

packed data to execute in a parallel fashion provide a promising alternative for sub-pixel and bi-directional ME/DE proceeding. First, unlike the data transfers in MPA, the overhead of transferring data from cache/memory to SIMD register is minimal. Second, decent amount of parallelism needed for the task at hand can be achieved on the modern processors that support wider register (up to 512-bit). With a register width of $W = 128$ and pixel bit depth of $B = 16$ the sub-pixel processing can be improved by up to a factor of $W/B = 8$.

This work, therefore, takes the advantage of SSE instruction set to efficiently reduce the number of instructions required for sub-pixel and bi-directional ME/DE. As shown in Fig. 4.5, to perform partial SAD on a batch of 16 pixels (16-bit pixels)



Video Name	Algorithm	PSNR	Bitrate	Skip rate
Balloons (1280x720)	TZ	41.13	879	N/A
	SF=1	41.13	882	92%
	SF=2	41.13	884	
	SF=4	41.13	881	
	SF=8	41.13	888	
Kendo (1280x720)	TZ	41.89	1090	N/A
	SF=1	41.90	1095	88.8%
	SF=2	41.90	1096	
	SF=4	41.90	1096	
	SF=8	41.90	1098	
PoznanHall2 (1920x1088)	TZ	42.43	791	N/A
	SF=1	42.44	793	97%
	SF=2	42.43	794	
	SF=4	42.43	793	
	SF=8	42.43	798	
PoznanStreet (1920x1088)	TZ	38.78	2519	N/A
	SF=1	38.78	2532	93%
	SF=2	38.78	2535	
	SF=4	38.78	2535	
	SF=8	38.78	2547	

Figure 4.6: Execution time, PSNR and birate performance measures for four 3-view MV-HEVC multiview test video sequences for a QP value of 32

Table 4.3
Experimental Condition

Hardware:	Intel® Xeon® CPU x5650® 12 cores (2 sockets, 6 cores/socket) with no-hyper-threading, and SSE4 SIMD instruction set with 128-bit, @2.67GHz and NVIDIA Fermi ™ C2075 SLI with 5 GB GDDR5,
Operating System:	Red Hat Enterprise Linux 7.2 (Maipo)
Software:	HEVC test and validation software suite (HM) 16.3 [28] and its MVC extension suite (HTM) 16.2 [4], GOPSize: 8, Temporal prediction structure: hierarchical B-frames, Inter-view prediction structure: IPP, BiPredIter: 2, IterSearchRange: 4, FastEncoderDecision: 0, FastDecisonMerge: 0, Anchor ME/DE Algorithm: TZsearch [4] [28], ME/DE search range [-64,64]/[-64,+64](horizontal/ vertical)
Test Sequence:	HEVC Video Sequence ∈ {PeopleOnStreet (2560×1920, Cactus (1920×1080), RaceHorses (832×480), BlowingBubbles (415×240) [80]} Multiview Video Sequence ∈ {1920×1088×25: PoznanStreet, Dancer, GTFly, Shark; 1024×768×30: Balloons, PoznanHall2, Newspaper, Kendo [81]}

only four types of SSE SIMD instructions are needed with a total of 12 operations (four *Load*, two *Absolute*, two *Difference*, and four Horizontal Add (*HADD*) as shown in Fig. 4.5 with their corresponding intrinsic SSE2 instruction). For PUs of 64×64,

64×32, 32×64, 32×32, 32×16, 16×32, 16×16, 16×8, 8×16, 8×8, 8×4, and 4×8, the number of SSE2 SIMD 16 pixel batch processing required to compute the global SAD value for one search location are, 256, 128, 128, 64, 32, 32, 16, 8, 8, 4, 2, 4, respectively. It should be noted that Intel® SSE instruction set offers a dedicated SIMD instruction (`_mm_SAD_epu8`) for 8-bit SAD calculations. However, this instruction is not suitable for 16-bit pixel representation. The chosen sequence of instructions work across a range of the pixel bit-depth from 8 to 12 bits as required by HEVC standard. First, PU and its reference block pixels are loaded onto 128-bit vector registers [2] via `LOAD` instruction. Eight 16-bit pixel-wise absolute value of differences are evaluated in parallel using subtraction and absolute (*AD* in Fig. 4.5) instructions and stored in a 128-bit vector register. Another eight absolute value of differences are computed in the same manner. The resulting two 128-bit vector registers containing 16 absolute differences are reduced to a single partial SAD value through a sequence of four horizontal add (*HADD* [79]) instructions. The *HADD* instruction performs binary reduction by adding eight pairs of neighboring absolute differences in 16-bit sub-registers and placing the results in lower four 16-bit sub-registers. The first execution of this instruction reduces two 128-bit vector registers in this fashion simultaneously, where results from the second register are stored in the higher four 16-bit sub-registers. The subsequent three invocations of this instruction reduces all the 16 absolute differences into a single partial SAD in the lowest 16-bit sub-register of SAD register, highlighted red in Fig. 4.5. This partial SAD is subsequently aggregated to the global SAD. On average, enabling SIMD acceleration yields an additional 20% execution time reduction with no cost to RD performance.

[2] The CPU on the platform used in the experiment (Table 4.3) supports up to Intel® SSE4 SIMD instruction set with maximum register width of 128-bit.

4.4.6 Results

For the experimental setup, the parameters in Table 4.3 are used. Four 3-view MV-HEVC multiview video sequences [81] are tried. The QP = 32 is selected for proving the effectiveness of *ScaleFast*. Fig. 4.6 for MV-HEVC multiview shows the execution time, peak signal to noise ratio (PSNR) and bitrate performance results of optimized predicate algorithm by integrating the scaled SAD computation into predicate algorithm. The figure also present the fraction of PUs that do not undergo integer ME/DE using the FBM in the predicate scheme.

A speedup factor of two in the integer ME with no loss in RD performance is observed for MV-HEVC test video sequences for a scaling factor of two. A scaling factor of four shows an average speedup factor of 2.2 for the integer ME with less than 0.02 dB loss in PSNR or alternatively, a negligible, 0.1% increase in bitrate. The highest skip rate of 97% is observed in *PoznanHall2* sequence with no obvious impact on the RD performance.

Regardless the type of video sequence, the speedup factor saturates beyond the scaling factor of four for the reason of non-uniformity in the scaling factor at the bottom of the CTU hierarchy as mentioned before.

Table 4.4 presents a comparative summary of the RD performance and speedup of the proposed *SacleFast* algorithm, when augmented with SIMD processing for the fractional and bi-directional ME/DE, with respect to the anchor TZsearch for eight multiview test sequences using Bjontegaard metric [1] over QP values of 22, 28, 32, and 37. An average speedup of 2.5 in ME/DE is observed with a minimal effect on PSNR and bitrate.

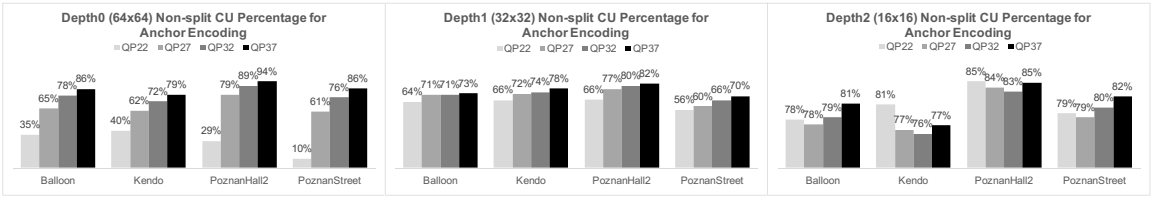


Figure 4.7: Percentage of non-splitting CU at *Depth0*, *Depth1*, *Depth2* for four 3-view MV-HEVC test video sequences, *Balloons*, *Kendo*, *PoznanHall2*, and *PoznanStreet* for anchor encoding.

4.5 Quantization Parameter (QP)-Based Early Termination of Coding Tree Unit (CTU)

4.5.1 Quantization

The PU residuals are transformed into the frequency domain by an integer transform operation that approximates the familiar 2D DCT. The QP determines the step size for associating the transformed coefficients with a finite set of steps. A large value of QP represents big steps that crudely approximate the transform coefficients in the spatial domain using a smaller number of bits, at the cost of higher distortion. A small value of QP approximates the block's spatial frequency spectrum more accurately at the cost of more bits. A rate control algorithm can dynamically adjust the encoder's QP as the most effective mean to achieve a target bitrate. In the quantization process, transformed coefficients are first divided by a quantization step size (Q_{step}) and then rounded. The exponential relation between the step size for the transformed coefficients Q_{step} and the QP (with value ranging from 0 to 51) for the video sequences with an 8-bit color depth is given as,

Table 4.4
Comparison of SIMD Augmented ScaleFast RD and Execution Time with the Anchor TZsearch Using Bjontegaard Metric [1] over QP values of 22, 28, 32, and 37

Video Sequence	**DPSNR**	**DBR**	**Speedup**	
			ME	Total
Balloons	-0.01	0.5%	2.1	1.4
PoznanHall2	-0.01	0.4%	1.8	1.3
Newspaper	-0.02	0.5%	2.6	1.4
Kendo	-0.01	0.4%	2.3	1.5
PoznanStreet	-0.01	0.6%	1.8	1.3
Dancer	-0.02	0.7%	1.9	1.3
GTFly	-0.02	1.0%	2.1	1.4
Shark	-0.03	0.9%	2.3	1.4
Average	-0.016	0.63%	2.11	1.4

Table 4.5
Coarse-Grain QP-based Early Termination Depth Selection

QP Sub-range	**Allowed Depth**	**Disallowed Depth**
$[0, 12]$	0, 1, 2, 3	None
$[13, 26]$	0, 1, 2	3
$[27, 40]$	0, 1	2, 3
$[41, 51]$	0	1, 2, 3

$$Q_{step}(\mathrm{QP}) = 2^{(\frac{\mathrm{QP}-4}{6})} \tag{4.1}$$

An unit increment in QP increases the quantization step size by approximately 12%. For an average QP of around 25, the reduction in the bitrate is approximately 80%. it should be, however, noted that the source of bitrate reduction is the transformed residual and not the MV/DV. Further, with QP greater than 25, the transformed residuals associated with the finer PU partitions (such as 16×16 and 8×8), have a higher likelihood of being zero compared with PUs of larger size (such as 64×64 and 32×32). Therefore, MVs/DVs associated with small PUs are more likely to be less useful.

4.5.2 Coarse-grain Early Termination of Coding Tree Unit (CTU)

In the anchor HM encoder, all CU and associated PU modes (see Table 4.1) are tried to determine the best modes in terms of lowest RD cost. This exhaustive search comes at the expense of high computational complexity. It is observed in the coding experiments that the best selected CU mode is highly correlated with the selected QP value. Fig. 4.7 shows the percentage of non-splitting CU after mode decision at three different depths for four commonly used QP values and four 3-view MV-HEVC multiview test sequences. As can be seen, at *Depth0* where CU size is 64×64, percentage of non-splitting CUs steadily increases with QP for all video sequences, approaching 90% at QP37. This indicates that as QP increases, the computation spent in trying all modes are mostly in vain because an increasing number of CUs remain non-split at *Depth0*. Similarly, for the CUs that are split at *Depth0* majority do not undergo split at *Depth1*, albeit a less predictable behavior. The same observation can be made about *Depth2*. It should be noted that $CU64 \times 64$ is at the root of coding tree and further splitting results in the evaluation of four 32×32 CUs, 16 16×16 CUs, and 64 8×8 CUs. Therefore, if split at a CU of 64×64 is skipped all subsequent evaluations are avoided. Therefore, in the mode decision process, correctly avoiding the evaluation of CU at different depths according to the QP value can bring significant time reduction.

To take the advantage of the observation in Fig. 4.7 a suitably mapping for all 52 QP values into four depths is needed. The entire QP range is dvided into four even sub-ranges with every 13 QP values mapped to the same termination depth as shown in Table 4.5. The mapping in Table 4.5 is universally applied to all video sequences. The RD performance of the proposed QP-based early termination scheme

is compared with the anchor encoder and presented in the first column of Table 4.6 (labeled *Baseline*). Comparison is made using Bjontegaard metric [1] over QP values of 22, 28, 32, and 37. The scheme works well for video sequences *PoznanHall2* and *Balloons*, where the contents are more static. However, the RD performance of the scheme is not sufficiently good for *Kendo* and *PoznanStreet* video sequences where the video contents are more dynamic. The bitrate in these cases increases by 11%. The discussion in the sequel presents a refinement scheme to improve the RD performance of video sequences.

4.5.3 Selective Coding Unit (CU) Split

The main reason for the RD performance loss in the simple QP-based early termination is the small number of available depths that cover the whole range of QP values. The coarse mapping in Table 4.5 cuts the CU tree too abruptly reducing the possibility of further CU splits to achieve a better RD performance at higher depths. To enhance the RD performance the use of three special types of CUs is introduced for which further split should be allowed.

$Inter2N \times 2N$ *CU*: A $2N \times 2N$ mode refers to the largest PU within a CU (see Fig. 4.3). When the coding selection of a $2N \times 2N$ PU is inter-prediction and the motion content is such that a $2N \times 2N$ mode selection is unable to capture all the motion information, a further split should be allowed. This split is likely to result in a better RD performance. In Table 4.6, the second column (labeled $Inter2N \times 2N$) shows the RD performance with respect to the *Baseline* when $Inter2N \times 2N$) CUs is allowed to split further. Compared with the anchor and *Baseline*, on average, this refinement technique has a bitrate increase of 3.5% with respect to the anchor, which is significant reduction from 6.2% for the *Baseline* case. The most dramatic effect

Table 4.6
Rate-distortion (RD) Performance for the Baseline QP-based Early Termination and Special CU Conditions for four MV-HEVC Multiview (3-view) Sequences Using Bjontegaard Metric over QP values of 22, 28, 32, and 37

	Baseline		**Inter**$2N \times 2N$				**Inter**$2N \times 2N$ **+ IntraMode**				**Inter**$2N \times 2N$ **+ IntraMode + Atypical_MV/DV**			
	DPSNR	DBR	DPSNR	DBR	ΔDPNSR	ΔDBR	DPSNR	DBR	ΔDPNSR	ΔDBR	DPSNR	DBR	ΔDPNSR	ΔDBR
Balloons	-0.15	+4.4%	-0.09	+2.7%	+0.06	-1.7%	-0.06	+1.8%	+0.03	-0.9%	-0.06	+1.8%	0	0
Kendo	-0.36	+11%	-0.20	+6.3%	+0.16	-4.7%	-0.17	+5.2%	+0.03	-1.1%	-0.15	+4.6%	+0.02	-0.8%
PoznanHall2	-0.06	+3.9%	-0.02	+1.8%	+0.04	-2.1%	-0.01	+1.3%	+0.01	-0.5%	-0.01	+1.2%	0	-0.1%
PoznanStreet	-0.13	+5.4%	-0.07	+3.1%	+0.05	-2.3%	-0.05	+2.3%	+0.02	+0.8%	-0.05	+2.1%	0	0.2%
(Average)	-0.18	+6.2%	-0.1	+3.5%	+0.08	-2.7%	-0.07	+2.7%	+0.02	-0.8%	-0.08	+2.4%	0	-0.3%

Table 4.7
Rate-distortion (RD) Performance for Four MV-HEVC Multiview (3-view) Video Sequences for the QP-based Early Termination and Selective CU Split (QPTerm) Scheme

Video Sequence	**Start QP**	**PSNR (dB)**		**Bitrate (KBits/Second)**		**Time (Second)**	
		Anchor	**QpTerm**	**Anchor**	**QpTerm**	**Anchor**	**QpTerm**
Balloons	22	44.85	44.84	4290	4292	2701	1644
(1024×768)	27	43.17	43.15	1715	1740	2271	856
	32	41.13	41.10	880	890	2090	737
	37	38.72	38.65	487	495	1958	367
Kendo	22	45.50	45.49	5114	5189	3408	1848
(1024×768)	27	43.91	43.89	2189	2271	2987	1035
	32	41.89	41.86	1090	1128	2755	925
	37	39.57	39.42	594	646	2608	522
PoznanHall2	22	44.46	44.45	13198	13198	8303	4813
(1920×1088)	27	43.42	43.41	1968	1978	5641	2280
	32	42.43	42.42	791	796	5120	2011
	37	41.10	41.06	406	411	4806	1116
PoznanStreet	22	43.23	43.22	31995	32067	7675	4931
(1920×1088)	27	40.94	40.92	7203	7318	5603	1958
	32	38.78	38.76	2519	2561	5059	1579
	37	36.66	36.59	1100	1141	4799	734

is in the most dynamic video *Kendo* video sequence where the reduction in DBR with respect to the *Baseline* case is 4.7%. With respect to the *Baseline* the average (DPSNR) is improved by 0.08 dB. For *Kendo* the improvement is 0.16 dB.

Intra-predicted CU: An intra-predicted CU does not rely on motion information. Therefore, it is more likely that split in such CUs will further improve RD performance. In general, intra-prediction contributes far less to the execution time than inter-prediction and allowing intra-predicted CUs to split only increases the execution time slightly. The third Column (labeled $Inter2N \times 2N$ + *IntraMode*) in Table 4.6 shows the aggregated effect of further split of $Inter2N \times 2N$ and intra-predicted CUs. The ΔPSNR and ΔDPR values show the incremental improvement that results from the further splits of intra-predict CUs. On average, the DBR and PSNR improve by 0.8% and 0.02dB, respectively. The most significant improvement is, again, for video sequence *Kendo* where the DBR is reduced by 1.1% to 5.2%.

Atypical MV/DV CU: From experimentation with a range of video sequences, it is observed that some CUs with $Inter2N \times 2N$ as the best mode have very large MV/DV values. As RD evaluation in the ME/DE is a function of both MV/DV value and residual SAD, in a CU with dynamic content the cost of SAD dominates over the cost of MV/DV. In such a case, further split of the CU aides the RD evaluation to find better matching block with smaller overall residual SAD and MV/DV cost in the next depth level. As CU size reduces MVs/DVs become more accurate, and the magnitude of residual SAD decreases; so the likelihood of atypical MV/DV also reduces. To accommodate CU at all depth level, an atypical MV/DV distance is associated with every depth as a measurement of the likelihood of occurrence of atypical MV. This distance is selected to exponentially grow from *Depth0* to *Depth2* as shown in Fig. 4.8. The method of selection of atypical distances is inline with exponentially decreasing magnitude of SAD with the quad-splits of CUs.

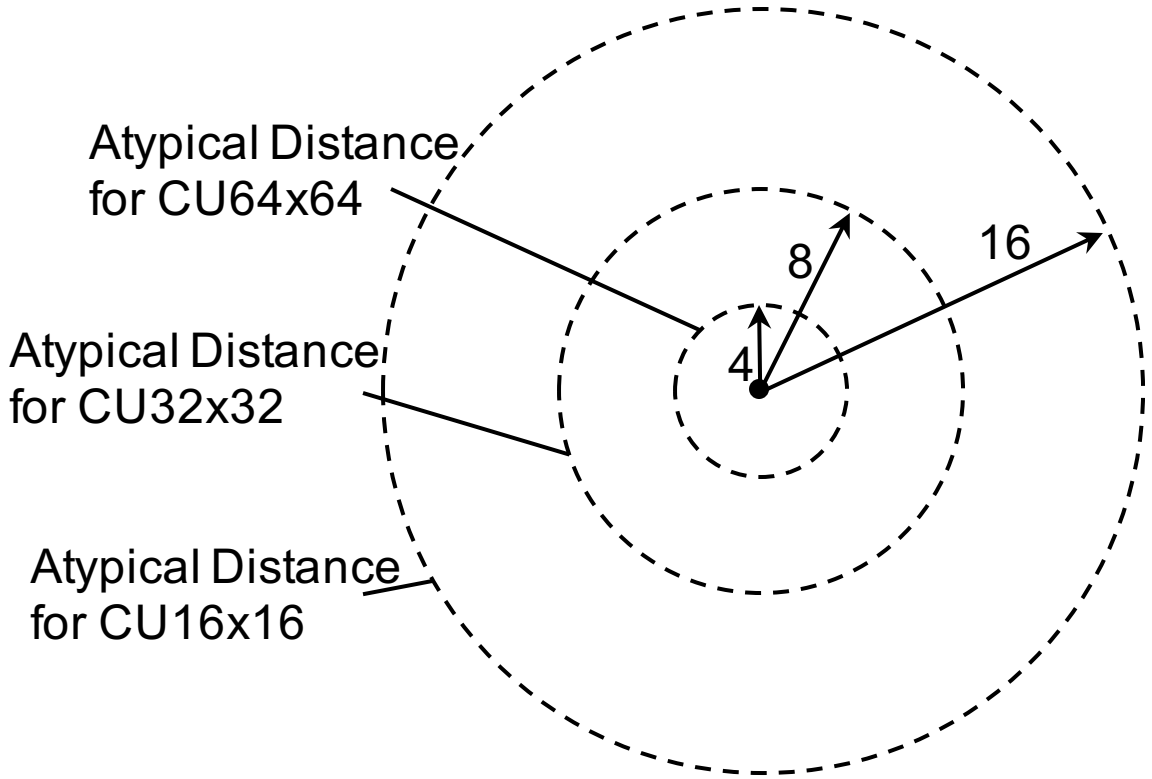


Figure 4.8: Atypical distances for three CU blocks

When a CU selects $Inter2N \times 2N$ as the best mode, but the value of its MV/DV is larger than atypical distance defined for its depth, further split is allowed. The RD performance using this technique is shown in the last column of Table 4.6 (labeled $Inter2N \times 2N + IntraMode + Atypical\_MV/DV$). As expected, this technique works well for dynamic video sequences such as *Kendo*, the DBR is reduced further by 0.8% and DPSNR is increased by 0.02 dB. On average, this technique improves the DBR performance by 0.3%.

In Table 4.7, the PSNR and bitrate performance of the QP-based early termination with selective CU split refinement (*QPTerm*) is compared against the anchor encoder at four QP values (22, 27, 32, 37). The data in the table show that *QPTerm* is an effective scheme in reducing the encoding execution time while maintaining a good RD performance across a range of QP values. At QP value of 37, the execution time

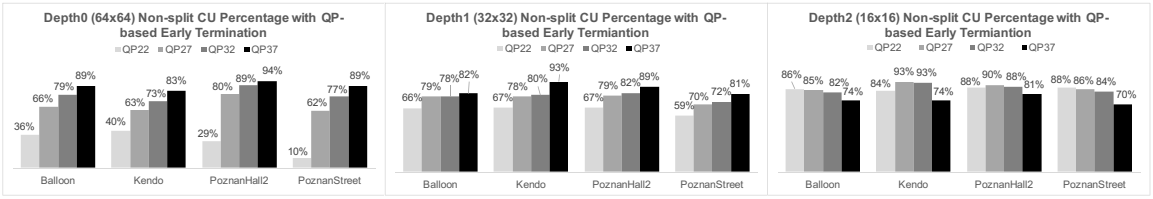


Figure 4.9: Percentage of non-splitting CU at *Depth0*, *Depth1*, *Depth2* for 3-view MV-HEVC test video sequences, *Balloons*, *Kendo*, *PoznanHall2*, and *PoznanStreet* for QP-based early termination.

Table 4.8
Rate-distortion (RD) and Execution Time Comparison with Anchor Using Bjontegaard metric [1] over QP values of 22, 28, 32, and 37, for eight MV-HEVC Multiview (3-view) Video Sequences

Video Sequence	**DPSNR**	**DBR**	**Speedup**		
			min (QP=22)	**max (QP=37)**	**avg**
Balloons	-0.06	1.8%	1.8	6.0	3.5
PoznanHall2	-0.01	1.2%	1.7	4.3	2.7
Newspaper	-0.03	1.0%	1.9	6.9	3.9
Kendo	-0.15	4.6%	1.8	5.0	3.2
PoznanStreet	-0.05	2.1%	1.6	6.5	3.5
Dancer	-0.08	2.6%	1.5	4.6	2.7
GTFly	-0.06	2.4%	1.6	4.9	2.9
Shark	-0.18	4.6%	1.4	2.6	4.3
Average	-0.08	2.5%	1.7	5.3	3.1

is reduced by factor 3 to 4 with insignificant change to the RD performance. Fig. 4.9 shows the percentages of non-splitting CUs after application of *QPTerm* scheme. For *Depth0*, the non-splitting percentages at all four QP values are slightly higher than those of anchor encoder in Fig. 4.7, specially for *Kendo* video sequence. This increase in the non-split rate explains the reason for slightly inferior RD performance for the proposed scheme. Trends for *Depth1*, are similar but with slightly higher percentage. For *Depth2* the non-split percentages are generally even higher.

Table 4.8 shows a summary of comparative result of the combined predicate and early termination scheme (*QPTerm*) with the anchor encoder using the Bjontegaard [1] metric for eight MV-HEVC multiview (3-view) test sequences. A speedup factor of 1.7 to 5.3 is achieved with an average of 3.1 (or 68% time reduction) over the range

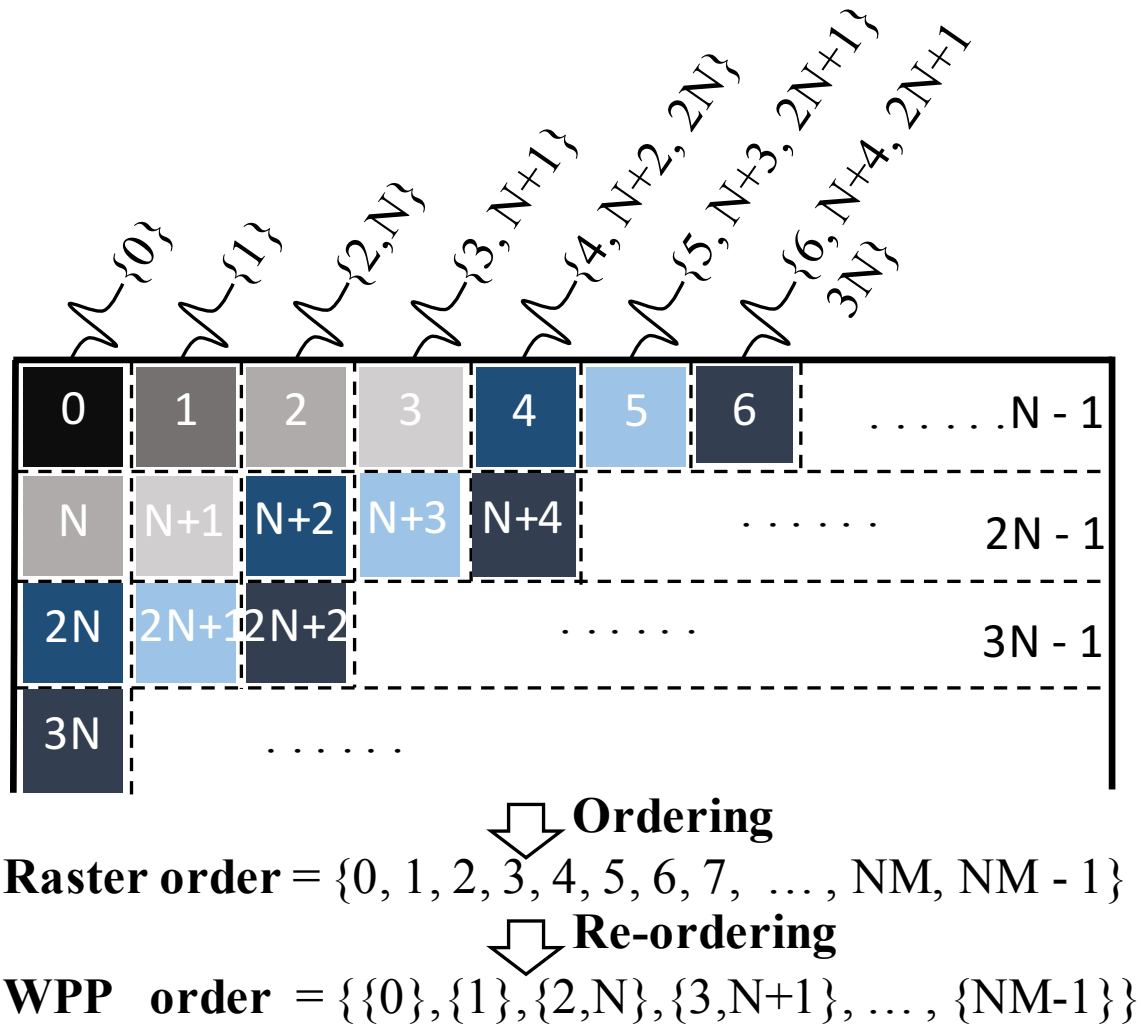


Figure 4.10: Wave-front parallel processing

of QPs. For majority of the video sequences the increase in the bitrate (DBR) is less than 3%, or the degradation in PSNR (DPSNR) is less than 0.08 dB. One notable exception is *Kendo* video sequence featuring a highly dynamic scene with an uneven background illumination, causing the proposed algorithm to be less effective.

4.6 High Level Parallel Processing

HEVC video coding level (VCL) based tools provide support for picture, tile and wavefront parallel processing (WPP) as part of the standard [6] [5].

The picture level parallel processing [5] for MV-HEVC, and the way to overcome the serious problem of workload imbalance when processing frames in parallel in a MVC system was extensively studies in our previous work [76]. If designed correctly parallel processing of frames can hugely speed up the performance with no impact on the RD performance.

In addition to the speedup, the parallel processing of independent tiles in a frame where tiles are transported in different packets are also suitable in a lossy transmission environment [5]. However, the RD performance loss increases with the number of tiles, due to the breaking of dependencies along tile boundaries. The loss in the RD performance is specially significant in the MV-HEVC environment due to significantly larger number of dependencies. Therefore, tile parallel processing is not considered in this work.

The WPP is performed at the CTU level where multiple non-neighboring CTUs with no coding dependencies, (except for the CABAC context variables at the end of each CTU row [5]), can be processed in parallel as shown in Fig. 4.10. CTUs in Fig. 4.10 are identified by their raster-scan order (left-to-right and top-to-bottom). In the figure the CTUs with same color coding belong to the same *wavefront* and can be processed in parallel, rolling from the top-left to bottom-right corner. Due to 2-step delay in the processing of CTUs in a row with respect to its previous row, for a given number of parallel processors (12 cores in our experimentation), it will take certain number of ramp up coding steps (24 steps for 12 cores) before all the processors are

fully utilized. The parallelization inefficiencies in ramp up at top-left corner and the ramp down at bottom-right increase with the number of processors, and more so for the lower resolution video sequences.

The loss in the RD performance that would result from the conventional CABAC initialization at the starting point of each CTU row is, to some extent, mitigated by propagating the content of the partially adapted CABAC context variables from the encoded second CTU of the preceding CTU row to the first CTU of the current CTU row (as shown from CTU 1 to CTU N in Fig.4.10) and a reset at the end of each row [82]. However, our results show that the impact of CABAC dependencies on the coding efficiency in MV-HEVC is more severe than in the case of HEVC.

One common method, for WPP is the allocation of one processor for each CTU row [21] [82]. An alternate implementation of WPP is provided where the impact of parallelization inefficiencies due to ramp up and ramp down can be reduced. In the proposed implementation all the CTUs for the WPP are placed in the set of lists in advance. Each list corresponds to one step in coding wavefront as seen in Fig. 4.10 ({0}, {1}, {2, N}, {3, N+1}, {4, N+2, 2N}, ..., N-1, 2N-3, 3N-5, ...}, ..., {NM-1}). The CTUs within one list can be processed in any order by any processor, as soon one processor becomes available. However, to maintain the coding dependency requirement for the WPP in HEVC, the CTUs from one list can be processed only if there are no more CTUs left to be processed from the previous list. The advantage of this scheme is that CTUs with the identical raster-scan order from multiple views from the enhancement layers that have no coding dependencies, (View_1 and View_2 in Fig. 4.2), can be placed in the same list and processed in parallel. This will increase processor utilization of WPP.

It should be noted that in our implementation 12 processor cores all use the same set of two GPUs for *Scalefast* search for ME/DE. In our previous work [76] showed that this

Table 4.9
Rate-distortion and Execution Time Comparison of Wavefront Parallel Processing with Anchor Using Bjontegaard Metric [1] for eight MV-HEVC Multiview (3-view) Video Sequences

Video Sequence	DPSNR	DBR	Speedup		
			mix	max	avg
Balloons	-0.12	4.3%	3.0	8.8	5.1
PoznanHall2	-0.06	4.8%	3.7	10.0	5.6
Newspaper	-0.09	2.8%	2.7	9.2	5.2
Kendo	-0.22	7.1%	2.9	9.4	5.3
PoznanStreet	-0.07	3.4%	3.2	11.3	6.1
Dancer	-0.10	3.4%	2.6	9.6	5.3
GTFly	-0.08	3.6%	2.9	10.0	5.5
Shark	-0.22	5.5%	3.1	11.4	5.8
Average	-0.12	4.4%	3.0	9.7	5.4

limits the performance of the multicore parallel processing where massively parallel ME/DE search algorithm on the GPU constitute majority of the coding workload. However, both predicate and early termination schemes significantly reduce the need for *Scalefast* search on the GPU pair.

4.6.1 Results

Table 4.9 presents the comparative results of rate-distortion and execution time speedup of WWP with respect to the anchor encoder using the Bjontegaard metric [1]. As can be seen, the speedup gain of the encoder with WPP is 3.0 to 9.7, with an average of 5.4, when compared with the anchor encoder. This speedup comes at the average cost of 0.04 dB degradation in PSNR or 1.9% increase in the bitrate (due to partially adapted CABAC context variable) compared to the case with no WPP in Table 4.7. The Execution time performance of higher resolution videos is slightly better due to longer parallel wavefronts.

4.7 Conclusion

This chapter introduced several optimization techniques at different levels of coding abstraction for MV-HEVC. In the parallelization of ME, the use of inter-pixel similarity for integer ME/DE and SIMD for sub-pixel and bi-directional ME/DE is demonstrated, with an insignificant amount of loss in the RD performance. In the proposed early termination scheme the QP is used to dynamically control the partitioning depth in the encoding of a CTU. The advantage of the proposed optimization algorithm is its adaptability to the QP parameter and characteristics of the video content. The chapter also proposed a WPP implementation that utilizes the computing resource in an efficient way and is, therefore, less sensitive to ramp up and ramp down in the wavefront. The algorithm performance show less than 3.1% average bitrate increase compared with the anchor encoder with maximum speedup gain of 5.3 without WPP for the set of eight MV-HEVC multiview video sequences tried. The corresponding values with WWP are 5% for bitrate increase and 9.7 for the speedup.

Chapter 5

Conclusion and Future Work

This dissertation presents highly efficient algorithms for video encoding on parallel computing platforms. To reduce encoding time while maintaining comparable quality, the problem is tackled in a novel parallel-computing perspective. Three levels of parallelism are investigated and corresponding parallel algorithms are proposed.

1. At the low parallel level (data-parallel), massively parallel algorithms for pixel level processing (motion estimation and disparity estimation) are implemented on massively parallel architecture. This brings over 100 times speed when compared against sequentialized full search ME/DE and over 8 times speedup when compared against the state-of-the-art sequentialized fast search ME/DE, with nearly same rate-distortion performance.

   In addition, SIMD instruction is adopted for small region ME/DE where the massively parallel shows lack of efficiency. Depending on the processor architecture, the speedup as a result of using SIMD instruction varies and is greater than 4.

2. At the higher level (task level), multi-core processors are employed to simultaneously process multiple coding units in a wave-front fashion to reduce impact on coding performance. Due to the use of massively parallel algorithm at the lower level, the improvement at this level ties to the number of MPA hardwares and the observed speedup is 4.

3. For multi-view coding where a large number of video sequences is available, a single computing node can be easily saturated with the parallel algorithms proposed for lower and higher levels. To enable acceleration across multiples views, a cluster implementation by exploiting the GOP-level parallelism is proposed and aims for load-balancing and maximizing resource usage. When evaluated on two popular prediction structures IBP and IPP, the speedup for 8-view encoding are 8 and 12, respectively.

5.1 Future Work

For the near future, parallel processors with higher number of cores (rather than higher frequency) are likely to remain as the dominant computing hardware. For this reason, parallel algorithms are still the first choice when it comes to improve computational performance. Three possible future research directions regarding parallel encoder optimization are given as follow.

1. The data level parallelism presented in this work is in the ME/DE process. There are other encoder modules possessing data level parallelism such as discrete cosine transform and entropy coding. In these procedures, the amount of parallelism is unlikely to be high enough for efficient massively parallelization. However, with proper SIMD instruction implementation, obtaining a speed up

in order of 10s is still promising and worth investigation.

2. All current parallel algorithms are designed separately for multi-core CPUs and massively parallel GPUs. However, there is no software or hardware limitation to merge the two computing hardwares to process a single task. In a simplest example, CPUs can be allocated to compute motion estimation in a quarter of the search region while GPU processes the remaining three quarter area. The impact on coding performance of simultaneous execution on heterogeneous devices is undetermined and requires careful analysis. A possible shortcoming is the large overhead involved in using heterogeneous devices but may be improved in the future as computer technology advances.

3. Recently there is an upheaval in the field of machine learning. It will be interested to see how machine learning can be adopted for video compression. Many existing works have proven that the coding modes of different coding units within a frame are correlated. This fact can be exploited to selectively skip future mode evaluations (and thus reduce execution time) based on previous mode decision knowledge. By learning the modes for a particular block pattern, it might be possible to skip mode evaluation when the same pattern reoccurs during the encoding of another video sequence. In addition, many algorithms in machine learning are highly parallel and amenable to massively parallel architecture.

References

[1] G. Bjontegaard, "Calculation of average PSNR differences between RD-curves," in *ITU - Telecommunications Standardization Sector Video Coding Expert Group (VCEG) 13th meeting, ITU-T Study Group 16, Question 6 (ITU-T SG16 Q.6), Doc No. VCEG-M33*, (Austin, TX), Apr. 2001.

[2] ITU-T and I. JTC, "SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS, Infrastructure of audiovisual services - Coding of moving video, ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), version 04, 2013."

[3] ITU-T and I. JTC, "Advanced Video Coding for Generic Audiovisual Services, ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), Version 16, 2012."

[4] Heinrich-Hertz-Institute (HHI), "H.265/MPEG HEVC Multiview Coding Reference Software (HTM) 16.2," 2014.

[5] V. Sze, M. Budagavi, and G. J. S. (Eds.), *High Effieicency Video Coding Algorithms and Architectures*. Springer, 2014.

[6] ITU-T and I. JTC, "SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS, Infrastructure of audiovisual services - Coding of moving video, ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), ITU-T H.265(V3), 2015,"

[7] P. Merkle, A. Smolic, K. Müller, and T. Wiegand, "Efficient prediction structures for multiview video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, pp. 1461–1473, Nov. 2007.

[8] C. Zhu, X. Lin, and L. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, pp. 349 –355, May 2002.

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

[10] L. Xie, L. Huang, and B. Chen, "UMHexagonS search algorithm for fast motion estimation," in *IEEE International Conference on Computer Research and Development (ICCRD)*, vol. 1, (Shanghai, China), pp. 483–487, Mar. 2011.

[11] A. M. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," in *Visual Communications and Image Processing*, (San Jose, CA), pp. 1069–1079, Jan. 2002.

[12] R. Rodriguez, J. Martinez, G.Fernandez-Escribano, J. Claver, and J. Sanchez, "Accelerating H.264 inter prediction in a GPU by using CUDA," in *Digest of Technical Papers, International Conference on Consumer Electronics (ICCE)*, (Las Vegas, NV), pp. 463–464, Jan. 2010.

[13] W. Chen and H. Hang, "H.264/AVC motion estimation implementation on Compute Unified Device Architecture (CUDA)," in *IEEE International Conference on Multimedia and Expo*, (Hannover, Germany), pp. 697–700, Jun. 2008.

[14] R. Cheng, Y. Eryan, and T. Liu, "Speeding up motion estimation algorithms on CUDA technology," in *Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, (Shanghai, China), pp. 93–96, Sept. 2010.

[15] C. Lu and H. Hang, "Multiview encoder parallelized fast search realization on NVIDIA CUDA," in *IEEE International Conference on Visual Communications and Image Processing (VCIP)*, (Tainan, Taiwan), pp. 1–4, Nov. 2011.

[16] R. R. Sanchez, J. Martinez, G. F. Escribano, J. Sanchez, and J. Claver, "A Fast GPU-Based Motion Estimation Algorithm for HD 3D Video Coding," in *Proceedings of IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA),*, (Leganes, Spain), pp. 166–173, July 2012.

[17] R. Rodriguez-Sanchez, J. Martinez, G. Fernandez-Escribano, J. Claver, and J. Sanchez, "Reducing complexity in H.264/AVC motion estimation by using a GPU," in *IEEE 13th International Workshop on Multimedia Signal Processing (MMSP)*, (Hangzhou, China), pp. 1–6, Oct. 2011.

[18] K. Choi, S. H. Park, and E. S. Jang, "Coding tree pruning based CU early termination," in *document JCTVC-F092, JCT-VC*, (Torino, Italy), Jul. 2011.

[19] R. H. Gweon and Y. L. Lee, "Early termination of CU encoding to reduce HEVC complexity," in *document JCTVC-F045, JCT-VC*, (Torino, Italy), Jul. 2011.

[20] J. Yang, J. Kim, K. Won, H. Lee, and B. Jeon, "Early SKIP detection for HEVC," in *document JCTVC-G543, JCT-VC*, (Geneva, Switzerland), Nov. 2011.

[21] W. Xiao, B. Li, J. Xu, G. Shi, and F. Wu, "HEVC encoding optimization using multicore CPUs and GPUs," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, pp. 1830–1843, Nov. 2015.

[22] L. Zhao, L. Zhang, S. Ma, and D. Zhao, "Fast mode decision algorithm for intra prediction in (hevc)," in *IEEE Visual Communications and Image Processing (VCIP)*, pp. 1–4, Nov. 2011.

[23] Y. Pang, L. Sun, J. Wen, F. Zhang, W. Hu, W. Feng, and S. Yang, "A framework for heuristic scheduling for parallel processing on multicore architecture: A case study with multiview video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, pp. 1658–1666, Nov. 2009.

[24] P. Carballeira, J. Cabrera, A. Ortega, F. Jaureguizar, and N. Garcia, "A framework for the analysis and optimization of encoding latency for multiview video," *IEEE Journal of Selected Topics in Signal Processing*, vol. 6, pp. 583–596, Sept. 2012.

[25] M. Flierl and B. Girod, "Multiview video compression," *IEEE Signal Processing Magazine*, vol. 24, pp. 66 –76, Nov. 2007.

[26] A. Vetro, T. Wiegand, and 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, pp. 626 –642, Apr. 2011.

[27] Heinrich-Hertz-Institute (HHI), "H.264/MPEG AVC Multiview Coding Reference Software (JMVC) 8.5," 2011.

[28] Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG 16 WP3 and ISO/IEC JTC 1/SC 29/WG 11, "HEVC Test Model (HM) 16. Reference Software," 2015.

[29] L. Ding, W. Chen, P. Tsung, T. Chuang, P. Hsiao, Y. Chen, H. Chiu, S. Chien, and L. Chen, "A 212 MPixels/s 4096 × 2160p multiview video encoder chip for 3D/quad full HDTV applications," *IEEE Journal of Solid-State Circuits*, vol. 45, pp. 46–58, Jan. 2010.

[30] S. B. B. Zatt, M. Shafique and J. Henkel, "Multi-level pipelined parallel hardware architecture for high throughput motion and disparity estimation in multiview video coding," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, (Grenoble, France), pp. 1–6, Mar. 2011.

[31] P. Merkle, K. Müller, A. Smolic, and T. Wiegand, "Efficient compression of multi-view video exploiting inter-view dependencies based on H.264/MPEG4-AVC," in *IEEE International Conference on Multimedia and Expo*, (Toronto, ONT, Canada), pp. 1717–1720, July 2006.

[32] H. Schwarz, D. Marpe, and T. Wiegand, "Analysis of hierarchical B pictures and MCTF," in *IEEE International Conference on Multimedia and Expo*, (Toronto, ONT, Canada), pp. 1929–1932, July 2006.

[33] P. Merkle, A. Smolic, K. Müller, and T. Wiegand, "Efficient prediction structures for multiview video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, pp. 1461–1473, Nov. 2007.

[34] M. Shami and A. Hemani, "Classification of massively parallel computer architectures," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) PhD Forum, 2012*, pp. 344–351, May 2012.

[35] B. Pieters, C.-F. Hollemeersch, P. L. J. De Cock, W. D. Neve, and R. V. D. Walle, "Parallel deblocking filtering in MPEG-4 AVC/H.264 on massively parallel architectures," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, pp. 96–100, Jan. 2011.

[36] J. Ohm, G. Sullivan, T. T. H. Schwarz, and T. Wiegand, "Comparison of the coding efficiency of video coding standards – including high efficiency video coding (HEVC)," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1669–1684, Dec. 2012.

[37] G. Lee, Y. Chen, M. Mattavelli, and E. Jang, "Algorithm/architecture co-exploration of visual computing on emergent platforms: Overview and future prospects," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, pp. 1576–1587, Nov. 2009.

[38] D. Finchelstein, V. Sze, and A. Chandrakasan, "Multicore processing and efficient on-chip caching for H.264 and future video decoders," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, pp. 1704–1713, Nov. 2009.

[39] M. Sniedovich, *Dynamic Programming: Foundations and Principles*. Taylor & Francis, 2010.

[40] NVIDIA, "NVIDIA Compute Unified Device Architecture (CUDA) Programming Guide 5.0," 2013.

[41] C. Jiang and S. Nooshabadi, "GPU accelerated motion and disparity estimations for multiview coding," in *IEEE International Conference on Image Processing (ICIP)*, (Melbourne, Australia), pp. 2106–2110, Sept. 2013.

[42] NVIDIA, "CUDA Basic Linear Algebra Subprograms (CUBLAS) library," 2013.

[43] M. A. Mesa, A. Ramirez, A. Azevedo, C. Meenderinck, B. Juurlink, and M. Valero, "Scalability of macroblock-level parallelism for H.264 decoding," in *International Conference on Parallel and Distributed Systems (ICPADS)*, (Shenzhen, China), pp. 236–243, Dec. 2009.

[44] Mitsubishi Electric Research Laboratories (MERL), "MVC test sequences: Ballroom, Vassar, and Exit," 2005.

[45] KDDI R&D Laboratories Inc, "MVC test sequences: Race1," 2011.

[46] G. J. Sullivan, J.Ohm, H. Woo-Jin, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1649–1668, Dec. 2012.

[47] X. Wang, L. Song, M. Chen, and J. Yang, "Paralleling variable block size motion estimation of HEVC on CPU plus GPU platform," in *IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, (San Jose. CA), pp. 1–5, July 2013.

[48] C.-Y. Chen, J.-C. Wang, J.-F. Wang, and Y.-H. Hu, "Motion entropy feature and its applications to event-based segmentation of sports video," *EURASIP Journalof Advance Signal Process*, vol. 2008, pp. 152:1–152:8, Jan. 2008.

[49] E. Li, E. Li, X. Tong, J. Li, Y. Chen, T. Wang, P. Wang, W. Hu, Y. Du, Y. Zhang, and Y.-K. Chen, "Accelerating video-mining applications using many small, general-purpose cores," *IEEE Micro*, vol. 28, pp. 8–21, Sept. 2008.

[50] C. Jiang and S. Nooshabadi, "A scalable massively parallel motion and disparity estimation scheme for multiview video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, pp. 346–359, Feb. 2016.

[51] A. Margara and G. Cugola, "High-performance publish-subscribe matching using parallel hardware," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 126–135, Jan 2014.

[52] G. Dal, W. Kosters, and F. Takes, "Fast diameter computation of large sparse graphs using GPUs," in *22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, (Turin, Italy), pp. 632–639, Feb. 2014.

[53] G. Jiang, Z. Li, F. Wang, and S. Wei, "A high-utilization scheduling scheme of stream programs on clustered VLIW stream architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, pp. 840–850, Apr. 2014.

[54] S. Zezza, S. Nooshabadi, M. Martina, and G. Masera, "Efficient implementation techniques for ML based error correction for JPEG2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, pp. 591–596, Apr. 2009.

[55] M. Dyer, S. Nooshabadi, and D. Taubman, "Design and analysis of system on a chip encoder for JPEG2000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, pp. 215–225, Feb. 2009.

[56] A. Gupta, S. Nooshabadi, D. Taubman, and M. Dyer, "Realizing low-cost high-throughput general-purpose block encoder for JPEG20000," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, pp. 843–858, July 2006.

[57] M. Dyer, S. Nooshabadi, and A. Gupta, "Concurrency techniques for arithmetic coding in JPEG2000," *IEEE Transactions on Circuits an Systems Part I*, vol. 53, pp. 1203–1213, June 2006.

[58] H. Franke, J. Xenidis, B. C., B. Bass, S. S. Woodward, J. D. Brown, and C. L. Johnson, "Introduction to the wire-speed processor and architecture," *IBM Journal of Research and Development*, vol. 54, pp. 3:1–3:11, January 2010.

[59] Y. Lee, R. Avizienis, A. Bishara, R. Xia, D. Lockhart, C. Batten, and K. Asanović, "Exploring the tradeoffs between programmability and efficiency in data-parallel accelerators," *SIGARCH Comput. Archit. News*, vol. 39, pp. 129–140, June 2011.

[60] A. Rafique, G. Constantinides, and N. Kapre, "Communication optimization of iterative sparse matrix-vector multiply on GPUs and FPGAs," *IEEE Transactions on Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2014.

[61] K. Wakabayashi, T. Takenaka, and H. Inoue, "Mapping complex algorithm into FPGA with high level synthesis reconfigurable chips with high level synthesis compared with CPU, GPGPU," in *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 282–284, Jan 2014.

[62] KHRONSOS Group, "The open standard for parallel programming of heterogeneous systems," 2014.

[63] Y. Yang, P. Xiang, M. Mantor, and H. Zhou, "CPU-assisted GPGPU on fused CPU-GPU architectures," in *2012 IEEE 18th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–12, Feb 2012.

[64] C. Wittenbrink, E. Kilgariff, and A. Prabhu, "Fermi GF100 GPU architecture," *IEEE Micro*, vol. 31, pp. 50–59, March 2011.

[65] M. Showerman, J. Enos, C. Steffen, S. Treichler, W. Gropp, and W.-M. Hwu, "EcoG: A power-efficient GPU cluster architecture for scientific computing," *Computing in Science Engineering*, vol. 13, pp. 83–87, March 2011.

[66] ITU-T and I. JTC, "SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS, Infrastructure of audiovisual services - Coding of moving video, ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), Version 3, 2015."

[67] A. Tsai, K. Bharanitharan, J. Wang, and K. Lee, "Effective search point reduction algorithm and its VLSI design for HDTV H.264/AVC variable block size motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 981–988, July 2012.

[68] G. Jo, J. Nah, J. Lee, J.Kim, and J. Lee, "Accelerating LINPACK with MPI-OpenCL on clusters of multi-GPU nodes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, pp. 1814–1825, July 2015.

[69] G. Bernabe, J. Cuenca, and D. Gimenez, "Optimizing a 3D-FWT code in a heterogeneous cluster of multicore CPUs and manycore GPUs," in *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, (Porto de Galinhas, Brazil), pp. 97–104, Oct. 2013.

[70] Y. Zhang and F. Mueller, "Autogeneration and autotuning of 3D stencil codes on homogeneous and heterogeneous GPU clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, pp. 417–427, Mar. 2013.

[71] F. Ries, T. De Marco, and R. Guerrieri, "Triangular matrix inversion on heterogeneous multicore systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, pp. 177–184, Jan. 2012.

[72] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 688–703, July 2003.

[73] G. Sullivan, J. Boyce, Y. Chen, J.-R. Ohm, C. Segall, and A. Vetro, "Standardized extensions of high efficiency video coding (HEVC)," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, pp. 1001–1016, Dec. 2013.

[74] X.Wang, L. Song, M. Chen, and J. Yang, "Paralleling variable block size motion estimation of HEVC on multi-core CPU plus GPU platform," in *IEEE International Conference on Image Processing (ICIP)*, (Tain City, Taiwan), pp. 1836–1839, Sept. 2013.

[75] G. Tech, Y. Chen, K. Muller, J.-R. Ohm, A. Vetro, and Y.-K. Wang, "Overview of the multiview and 3D extensions of high efficiency video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 26, pp. 35–49, Jan. 2016.

[76] C. Jiang and S. Nooshabadi, "Parallel multiview video coding exploiting group of pictures level parallelism," *IEEE Transactions on Parallel and Distributed Systems*, Early Access 2015.

[77] NVIDIA, "NVIDIA Compute Unified Device Architecture (CUDA) C Programming Guide 7.5," Sep. 2015.

[78] B. F. Wu, H. Y. Peng, and T. L. Yu, "Efficient hierarchical motion estimation algorithm and its vlsi architecture," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, pp. 1385–1398, Oct 2008.

[79] I. D. Zone, "Integer Intrinsic Intrinsics for Intel® Streaming SIMD Extensions 2 (Intel® SSE2)," 2014.

[80] F. Bossen, "Common HM test conditions and software reference configurations," Feb. 2012.

[81] D. Rusanovsky, K. Müller, and A. Vetro, "Common Test Conditions of 3DV Core Experiments," Jan. 2013.

[82] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel scalability and efficiency of hevc parallelization approaches," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, pp. 1827–1838, Dec 2012.

[83] J. Watts and S. Taylor, "A practical approach to dynamic load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, pp. 235–248, Mar 1998.

[84] V. D. Kim and Y. Chen, "Image processing on multicore x86 architectures," *IEEE Signal Processing Magazine*, vol. 27, pp. 97–107, March 2010.

[85] E. H. Jose L. Nunez-Yanez, A. Nabina and G. Vafiadis, "Cogeneration of fast motion estimation processors and algorithms for advanced video coding," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, pp. 437–448, March 2012.

[86] T. Mladenov, S. Nooshabadi, and K. Kim, "Implementation and evaluation of Raptor codes on embedded systems," *IEEE Transactions on Computers*, vol. 60, pp. 1678–1691, Dec. 2011.

[87] R. Rodriguez, J. Martinez, G. Fernandez-Escribano, J. Claver, and J. Sanchez, "Accelerating h.264 inter prediction in a gpu by using cuda," in *Proceedings of IEEE International Conference on Consumer Electronics (ICCE)*, (Las Vegas, NV), pp. 463–464, Jan. 2010.

[88] S. D. X. Tang and C. Cai, "An analysis of TZSearch algorithm in JMVC," in *Proceedings of 2010 International Conference on Green Circuits and Systems (ICGCS),*, (Shanghai, China), pp. 516–520, June 2010.

[89] H. J. C.D. Hong and S. Parameswaran, "Multi-ASIP based parallel and scalable implementation of motion estimation kernel for high definition videos," in *IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia)*, (Taipei, Taiwan), pp. 56–65, Oct. 2011.

[90] S. Zezza, S. Nooshabadi, and G. Masera, "A feasible VLSI engine for SISO for joint source channel codes," in *IEEE International Conference on Image Processing (ICIP-2009)*, (Cairo, Egypt), Nov. 2009.

[91] D. R. Butenhof, *Programming with POSIX Threads*. Addison-Wesley, 2000.

[92] NVIDIA, "FERMI Compute Architecture White Paper v1.1," 2009.

[93] NVIDIA, "NVIDIA Tesla C2075 Companion Processor," 2011.

Appendix A

Letters of Permission

A.1 Permission Letters for Chapter 2

Title: A Scalable Massively Parallel Motion and Disparity Estimation Scheme for Multiview Video Coding
Author: Caoyang Jiang
Publication: Circuits and Systems for Video Technology, IEEE Transactions on
Publisher: IEEE
Date: Feb. 2016

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

A.2 Permission Letter for Chapter 3

Title: Parallel Multiview Video Coding Exploiting Group of Pictures Level Parallelism
Author: Caoyang Jiang
Publication: Parallel and Distributed Systems, IEEE Transactions on
Publisher: IEEE
Date: 1 Aug. 2016

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK CLOSE WINDOW

A.3 Permission Letters for Chapter 4

Title: Massively Efficient Motion Estimation by Exploiting Inter-Pixel Similarities

Conference Proceedings: Data Compression Conference (DCC), 2016

Author: Caoyang Jiang

Publisher: IEEE

Date: March 2016

Thesis / Dissertation Reuse

The IEEE does not require individuals working on a thesis to obtain a formal reuse license, however, you may print out this statement to be used as a permission grant:

Requirements to be followed when using any portion (e.g., figure, graph, table, or textual material) of an IEEE copyrighted paper in a thesis:

1) In the case of textual material (e.g., using short quotes or referring to the work within these papers) users must give full credit to the original source (author, paper, publication) followed by the IEEE copyright line © 2011 IEEE.
2) In the case of illustrations or tabular material, we require that the copyright line © [Year of original publication] IEEE appear prominently with each reprinted figure and/or table.
3) If a substantial portion of the original paper is to be used, and if you are not the senior author, also obtain the senior author's approval.

Requirements to be followed when using an entire IEEE copyrighted paper in a thesis:

1) The following IEEE copyright/ credit notice should be placed prominently in the references: © [year of original publication] IEEE. Reprinted, with permission, from [author names, paper title, IEEE publication title, and month/year of publication]
2) Only the accepted version of an IEEE copyrighted paper can be used when posting the paper or your thesis on-line.
3) In placing the thesis on the author's university website, please display the following message in a prominent place on the website: In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of [university/educational entity's name goes here]'s products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to http://www.ieee.org/publications_standards/publications/rights/rights_link.html to learn how to obtain a License from RightsLink.

If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

BACK ... CLOSE WINDOW