View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Ghent University Academic Bibliography

biblio.ugent.be

The UGent Institutional Repository is the electronic archiving and dissemination platform for all UGent research publications. Ghent University has implemented a mandate stipulating that all academic publications of UGent researchers should be deposited and archived in this repository. Except for items where current copyright restrictions apply, these papers are available in Open Access.

This item is the archived peer-reviewed author-version of:

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

Rafael Rodriguez–Sanchez, Jose Luis Martinez, Gerardo Fernandez–Escribano, Jose L. Sanchez, Jose M. Claver, Jan De Cock, Bart Pieters, and Rik Van de Walle

In: Computers & Electrical Engineering, 39 (8), 2623–2637, 2013.

http://www.sciencedirect.com/science/article/pii/S0045790613001419

To refer to or to cite this work, please use the citation to the published version:

Rodriguez–Sanchez, R., Martinez, J. L., Fernandez–Escribano, G., Sanchez, J. L., Claver, J. M., De Cock, J., Pieters, B., and Van de Walle, R. (2013). 3D High Definition video coding on a GPU–based heterogeneous system. *Computers & Electrical Engineering 39(8)* 2623–2637. http://dx.doi.org/10.1016/j.compeleceng.2013.05.009

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

Rafael Rodríguez–Sánchez, José Luis Martínez, Gerardo Fernández–Escribano, José L. Sánchez

Instituto de Investigación en Informática de Albacete, Universidad de Castilla-La Mancha, Avenida de España s/n, 02071 Albacete, Spain

José M. Claver

Departamento de Informática, Universidad de Valencia, Avenida de la Universitat s/n, 46100 Burjassot, Valencia, Spain

Jan De Cock, Bart Pieters, Rik Van de Walle

Multimedia Lab, Ghent University – IBBT, Gaston Crommenlaan 8 bus 201, B–9050 Ledeberg–Ghent, Belgium

Abstract

H.264/MVC is a standard for supporting the sensation of 3D, based on coding from 2 (stereo) to N views. H.264/MVC adopts many coding options inherited from single view H.264/AVC, and thus its complexity is even higher, mainly because the number of processing views is higher. In this manuscript, we aim at an efficient parallelization of the most computationally intensive video encoding module for stereo sequences. In particular, inter prediction and its collaborative execution on a heterogeneous platform. The proposal is based on an efficient dynamic load balancing algorithm and on breaking encoding dependencies. Experimental results demonstrate the proposed algorithm's ability to reduce the encoding time for different stereo high definition sequences. Speed–up values of up to 90x were obtained when compared with the reference encoder on the same platform. Moreover, the proposed algorithm also provides a more energy–efficient approach and hence requires less energy than the sequential reference algorithm.

Keywords:
H.264/AVC, Inter Prediction, Motion Estimation, Disparity Estimation, Heterogeneous Computing

1. Introduction

The H.264/MPEG-4 Advanced Video Coding (AVC) standard [1] and its successor, High Efficiency Video Coding (HEVC) [2], have demonstrated significant improvements in video compression capabilities in the past few years [1], and currently [2], for a wide range of bit–rates and resolutions [3]. Moreover, since developing the H.264/AVC standard, both the Joint Video Team (JVT) of the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) have also standardized an extension of it, which is referred to as Multi–view Video Coding (MVC) [4]. The Joint Collaborative Team on 3D Video (JCT-3V) is currently working on the 3D extension of HEVC, and on the corresponding 3DV-HTM software. Therefore, the one standard solution for multi–view / stereo video coding at the moment is the H.264/MVC implementation.

H.264/MVC can provide users with the impression of complete scene perception by simultaneously transmitting several views to the receivers. It can also give users vivid information about the scene structure. H.264/MVC considers not only the correlation in the temporal direction for motion-compensated prediction (MCP), but also in the view direction for disparity compensated prediction (DCP). MVC adopts many coding tools used for single view H.264/AVC, such as the variable block-size Motion Estimation (ME) algorithm (which will be described in Section 2), among others. This is a very complex algorithm and, as part of the whole inter prediction algorithm, it is applied many times in terms of different MacroBlock (MB) sizes and, in the particular case of multi–view, between different views. All these computations involve even higher encoder complexity than single view H.264/AVC, which makes it necessary to develop a method that can reduce the complexity of MVC with minimal loss in coding efficiency (bit–rate and quality). In this area, several approaches are available in the literature that aim to accelerate this process by using algorithmic and code optimizations. However, as far as the authors of this paper know, there are no approaches that make use of hardware platforms to achieve this goal.

On the hardware side, in the past few years new architectures have been introduced in high–performance computing. Examples of such architectures include systems composed of multi–core CPUs and Graphics Processing Units (GPUs). The heterogeneous nature of modern desktop systems imposes addi-

Email addresses: `rrsanchez@dsi.uclm.es` (Rafael Rodríguez–Sánchez), `joseluismm@dsi.uclm.es` (José Luis Martínez), `gerardo@dsi.uclm.es` (Gerardo Fernández–Escribano), `jsanchez@dsi.uclm.es` (José L. Sánchez), `jose.claver@uv.es` (José M. Claver), `jan.decock@ugent.be` (Jan De Cock), `bart.pieters@ugent.be` (Bart Pieters), `rik.vandewalle@ugent.be` (Rik Van de Walle)

tional challenges for module parallelization. Several programming frameworks, such as the CUDA (Compute Unified Device Architecture)[5] programming model, already address the programmability issues in CPU+GPU environments by offering the unification of the execution space for different heterogeneous devices. CUDA abstracts both Simple Instruction Multiple Data (SIMD) and task parallelism into thousands of simultaneous threads.

At this point, this paper proposes an algorithm that efficiently distributes the burden of complexity of the inter prediction algorithm executed in H.264/MVC Stereo High Profile on a GPU-based heterogeneous platform. This paper is an extension of the one previously presented in [6]. This extension includes a more accurate and complex way of obtaining the motion information, surpassing the Rate Distortion (RD) performance obtained in [6]. Moreover, it includes a more complete evaluation section, as the proposals are tested against two of the most well–known inter prediction algorithms implemented in the reference software: the full search and UMHexagonS algorithms [7].

The algorithm includes both pixel and sub-pixel accuracy, as well as temporal and inter-view predictions and different transform sizes. Although the algorithm described in this paper is evaluated for stereo (2 views), it could easily be extended to n–views, and it can also provide the capability of 3D perception. The idea of the proposed algorithm consists in starting with the smallest MB sub-partitions, and is able to build the entire tree-structured ME procedure from bottom to top. In addition, some structural references have to be broken or adapted when they are calculated in parallel. Performance evaluation is carried out for full High Definition (HD) stereo video sequences and adapted for a system composed of an NVidia GTX480 and an Intel Core i7 processor. The results show a noteworthy time reduction of up to 98% with only a negligible RD penalty. Moreover, this work also shows that the GPU-based H.264/MVC encoding algorithm consumes less energy than the baseline algorithms running on a CPU core. Additionally, the proposed algorithm is tested against full and UMHexagonS search algorithms. GPU–based platforms raise the power consumption but the execution time is much shorter, which leads to a more efficient use of energy.

There are many potential applications for the algorithms presented in this paper. Nowadays, GPUs are available in a wide variety of environments, ranging from small and cheap personal computers, to large and expensive supercomputers. GPU manufacturers provide different GPU solutions to satisfy the requirements of all these different environments. As a consequence, the solutions proposed in this paper can be used by personal computers when performing a video conference, by more powerful servers dedicated to video streaming or by large high performance computers in the video storage industry.

The rest of the paper is organized as follows. Section II outlines the technical background. In Section III some related work is presented. Section IV describes the approach presented in this paper. In Section V the proposal presented is evaluated. Finally, conclusions are given in Section VI.

2. Background

2.1. H.264/MVC

The H.264/MVC coding standard was recently finalized by JVT, and was developed as an extension of H.264/AVC. MVC mainly uses H.264/AVC while taking advantage of temporal and inter-view dependencies [8] which are based on Hierarchical Bidirectional Pictures prediction to exploit both temporal and inter-view correlations [9]. Moreover, MVC provides other techniques such as Disparity Estimation (DE), which is used in the process of inter-view coding.

H.264/MVC inherits many video coding techniques from H.264/AVC, such as multiple reference frames, weighted prediction, a de-blocking filter, variable block size and quarter-pixel precision for motion compensation, which allow this optimum performance to be achieved at the expense of an increase in the computational complexity of the encoder.

Basically, variable block-size matching ME [3] and DE [8] are used to reduce the temporal and inter-view redundancies between frames. In this coding system, variable block-size ME is carried out using eight inter prediction modes (SKIP, Inter 16x16, Inter 16x8, Inter 8x16, Inter 8x8, Inter 8x4, Inter 4x8, and Inter 4x4), which are depicted in Figure 1. MVC determines which partitions deal with cost as the best MB partition. This results in a high complexity encoder. Therefore, it is necessary to develop a method that can reduce the execution time of MVC with minimal loss of image quality.

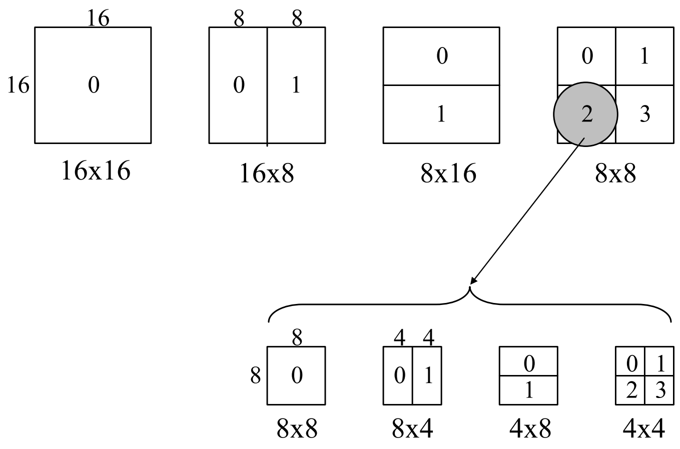


Figure 1: MB partitions.

2.2. Graphics Processing Units (GPUs)

Recent heterogeneous platforms include GPUs in order to achieve high-performance computing. Although GPUs come primarily from interactive applications such as multimedia and computer or console gaming, they can be used for running general purpose applications. In fact, GPUs have recently changed from being exclusively used in graphics applications to being used in what is called general purpose computing on GPU (GPGPU).

The GPU architecture offers a new challenge for engineering because the programming model must be adapted to the available hardware to obtain good performance and exploit the full potential of the GPU. This problem has been solved by GPU manufacturers, such as NVIDIA and AMD/ATI, by proposing new languages or even extensions for the most commonly-used high-level programming languages. NVIDIA GPUs were chosen since they can be programmed using CUDA C, which is a C–based high level programming language designed to maintain a low learning curve for programmers familiar with standard C. NVIDIA GPUs can be programmed using other high level programming languages such as OpenCL or Microsoft's C++ Accelerated Massive Parallelism (AMP) library (only for devices supporting DirectX 11), but CUDA is the best way to exploit the GPU's capabilities.

NVIDIA proposes the CUDA parallel computing architecture, which is a software platform for programming massively parallel high-performance computing applications on their company's powerful GPUs. NVIDIA has seized upon this opportunity to create a better programming model and to improve the shaders or stream processors. Usually, each stream processor on an NVIDIA GPU can manage many concurrent threads, and has its own FPUs (float point units), registers, and shared local memory [5]. At this point, the programmers do not write explicitly threaded code. A hardware thread manager handles threading automatically, which is an important feature of CUDA. Thus, the temporal and inter-view prediction algorithms proposed in the H.264/MVC encoder fit well in the GPU philosophy because they perform the same computations (computed costs) over different data (search area), and offer a new challenge for the GPUs. The main issue is how to efficiently distribute all the computations over the GPU resources and avoid sequential dependencies.

3. Related work

As far as the authors know, the algorithm presented in this paper is the first approach available in the literature that proposes a GPU-based algorithm for reducing the encoding time spent in MVC. It is true there are approaches which use heterogeneous computing to resolve single view inter prediction in H.264/AVC, and there are also approaches which deal with MVC inter-view prediction but by using faster (single core) algorithms. However, when using parallel architectures new challenges appear that need to be solved. The following paragraphs will provide an insight into the state–of–the–art from these two points of view.

Solutions for accelerating the H.264/AVC encoding algorithm by making use of Many-Core graphics hardware were firstly proposed in 2007 by *Lee et al.* [10]. They used a multi-pass and frame parallel algorithm to accelerate some ME tools available in an H.264/AVC encoder by using the OPENGL API. They unroll and rearrange the multiple nested loops by using a multi–pass method. Full-pixel accuracy ME is implemented using a two–pass method and sub–pixel ME is implemented using a six–pass method. The algorithm is implemented using a multi–pass method because the total number of instructions is higher than the GPU instruction limit. However, the algorithm does not support variable block size ME and it is not integrated in any H.264/AVC encoder.

In 2008, *Ryoo et al.* in [11] presented some optimization principles of a multi-threaded GPU using CUDA. In [12] *Chen and Hang* proposed an implementation of the H.264/AVC ME algorithm using CUDA. The algorithm is based on an efficient block-level parallel algorithm for the variable block size ME in H.264/AVC. They decompose the H.264/AVC ME algorithm into 5 steps so they can achieve highly parallel computation. However, they use many sequential kernels, thus reducing the parallel computations and increasing the memory transfers between the GPU and its DRAM memory. Moreover, they do not deal with the problem of Motion Vector Predictors (MVPs) in H.264/AVC and the algorithm is not included in any H.264/AVC encoder, so it is not possible to show any results for RD performance.

In 2010, *Cheung et al.* proposed a GPU implementation of the smpUMHexagonS ME algorithm implemented in JM 14.2 using CUDA [12]. This algorithm uses several techniques in order to save computation, including MVPs, different search patterns (cross, hexagon and diamond) and early–out termination. The authors divide the current frame into multiple tiles, each tile being processed by a single GPU thread, and different tiles are processed concurrently by different independent threads on the GPU. The number of tiles used affects the algorithm's performance, both in terms of execution time and in terms of RD performance. The greater the number of tiles used, the faster the algorithm is, and the greater the parallelism that can be achieved, but the RD performance is worse, and fewer MBs are predicted using real MVPs. They report significant bit–rate increments (12%) with a penalty in quality of up to 0.4 dB depending on the sequence and the tile length.

Studies on the reduction of the computational complexity of MVC have also appeared recently in the literature. As DE is different from ME in MVC, DE algorithms based on camera geometry and selective DE algorithms were proposed to reduce computational complexity in inter-view prediction. In 2007, *Lu et al.* in [13] proposed a DE technique to accelerate the disparity search by using an epipolar geometry. Epipolar geometry has been widely studied in computer vision and is the only geometry constraint between a stereo pair of images in a single scene. The proposed epipolar–geometry–based DE can greatly reduce the search region and effectively track large and irregular disparity, which is very common in multi–view scenarios. *Huo et al.* in [14] presented a scalable prediction structure for MVC in which inter–view prediction may be disabled if the inter–view redundancy can be almost eliminated by temporal or intra predictions. In this way, the time employed for DE may be saved by reducing encoder complexity. The authors use a hierarchical Group of Pictures (GOP) pattern and propose not to carry out the DE in one or more of the highest temporal layers of the hierarchical GOP pattern, since they observed that the percentage of temporal predictions increases with the increment of the temporal layer index. *Ding et al.* in [15] proposed a content–aware prediction algorithm for inter–view mode decisions. The proposed algorithm is able to reduce the unnecessary computational load by exploiting the correlation between the different views in MVC. The MB coding modes and their corresponding MV may be predicted by using the DE and the coding information of neighbouring views. Therefore, the computational complexity of ME can be greatly reduced since some MBs may be early identified as SKIP, INTRA or as a DE mode, and therefore it is not necessary to perform the ME.

More recently, in 2010, *Zeng et al.*[16] presented a fast mode decision algorithm called MET. First, for each MB the SKIP mode is evaluated. Then, if the encoding cost of the SKIP mode is below an adaptive threshold, the other modes do not need to be checked. The threshold is based on the mode correlation of adjacent MBs in the current view and in neighboring views. In 2012, *Liu et al.* in [17] presented a high-speed mode decision algorithm for the inter–view predictions of multi-view video sequences. Some candidate modes are disqualified from being checked to reduce the encoding cost calculations and an early stop mode decision is made by using multiple parameters related to the estimation of the final optimal mode. These parameters are: the MB residual and the temporal, spatial and inter–view correlations.

Finally, there are other approaches that can jointly accelerate both DE and ME [18, 19, 20]. In 2007, *Li et al.* in [18] presented a fast inter prediction algorithm for both ME and DE. First, the prediction type is selected depending on the reference frames. Regions with fast motion are best handled by inter–view prediction (DE). On the other hand, homogeneous and stationary regions are best handled by temporal prediction (ME). The reason is that regions with fast motion may be predicted using small block sizes and large MVs, which decrease the coding efficiency. Then, some unuseful search regions in the view direction are discarded from analysis, based on the displacement between the cameras which recorded the 3D scene. Finally, a fast mode decision is performed based on the prediction type previously determined, applying the ME or DE processes only to a subset of the available block modes. In 2009, *Shen et al.* in [19] proposed a fast DE and ME algorithm based on the correlation between the prediction/mode size and on motion homogeneity. MBs with homogeneous motion usually select temporal predictions with large block sizes, and MBs with complex motion usually select inter–view prediction or temporal prediction with small block sizes. The proposal uses the spatial properties of the motion field, which is generated by obtaining the MVs of all 4x4 partitions within a frame. In 2011, *Shen et al.* in [20] presented a low complexity mode decision algorithm to reduce the complexity of ME and DE in MVC. The proposed algorithm is based on four decision techniques: early SKIP mode decision, adaptive early–out termination, fast mode size decision, and selective intra coding in inter frames. The authors evaluate each technique separately, and as a final step evaluate the results of the entire algorithm.

As a conclusion for this section, the authors would like to mention that a fair comparison with the related papers is not possible for several reasons. Firstly, the number of views used in the related works affects the algorithm's performance; this paper proposes a stereo-based algorithm (2-views) while the other approaches use n-views, and thus the speed-ups are not comparable. Secondly, most of the MVC proposals presented in this section are implemented by using different reference software, called JMVM [21]. Nevertheless, the proposal presented in this paper exhaustively checks all the available ME and DE modes by using a highly parallel algorithm to exploit the GPUs capabilities, obtaining large time reductions whilst maintaining encoding efficiency in comparison with the reference software [22].

4. Proposed Algorithm

In this section, we describe our proposed GPU-based inter prediction algorithm, which is optimized for 3D HD video sequences. The proposed algorithm is integrated in the well-known H.264/AVC JM encoder, which offers support for MVC, version 17.2 [22], so a proper evaluation could be carried out.

During the whole encoding procedure, there is some data that does not change, such as frame sizes and the search area distribution. This information is located in GPU constant memory in order to take advantage of its cache. Furthermore, at the beginning of coding each frame, the frame itself and the reference frame are transferred to GPU texture memory to take advantage of its access modes.

In what follows, we describe the main parts of our proposed ME algorithm: full-pixel and sub-pixel inter prediction, inter-view prediction and a GPU–based 8x8 transform.

4.1. Full-Pixel inter prediction

For each MB in a frame, the reference full-pixel inter prediction sequentially obtains the cost for all positions checked inside the search area for all possible partitions/sub-partitions defined by the standard. Our main idea is to generate all this motion information at the beginning of coding each frame by using two GPU kernels.

The first GPU kernel uses 256 GPU threads per thread block. Each thread block takes as input a specific MB and the reference area for 256 positions of this MB. The MB and the reference area are stored in multiprocessor shared memory to reduce memory traffic and improve data locality at the same time. Shared memory is organized in a 4-byte data structure (integer values) in order to avoid shared memory bank conflicts since this memory is accessed in 4-byte words. Each GPU thread obtains the costs for the sixteen 4x4 blocks in which an MB can be divided for a specific position (Pst) and stores these costs in GPU registers for the cost generation step (see Figure 2).

Figure 2 shows how to obtain the motion information for a specific position (Pst) for all partitions/sub-partitions. In order to obtain the motion information for the eight 4x8 and for the eight 8x4 sub-partitions, it is only necessary to add two 4x4 SAD costs for each of them. For instance, by adding #0 and #2 SAD costs from the 4x4 sub-partition, the #0 SAD cost for the 8x4 sub-partition is obtained (see shaded boxes), and to obtain the four 8x8 partitions it is only necessary to add two 4x8 SAD costs, and so on. As was done for the first part of the algorithm, intermediate results are stored in multiprocessor shared memory using a 4-byte data structure, but in this case this structure is composed of two unsigned shorts (2 bytes each) which contain the cost and an associated position.

Finally, this first kernel performs a reduction of the previously generated information (Figure 3). This kernel has

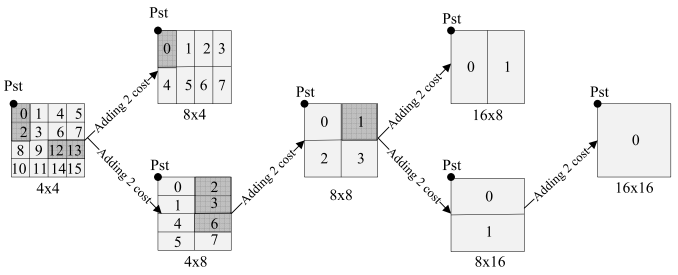
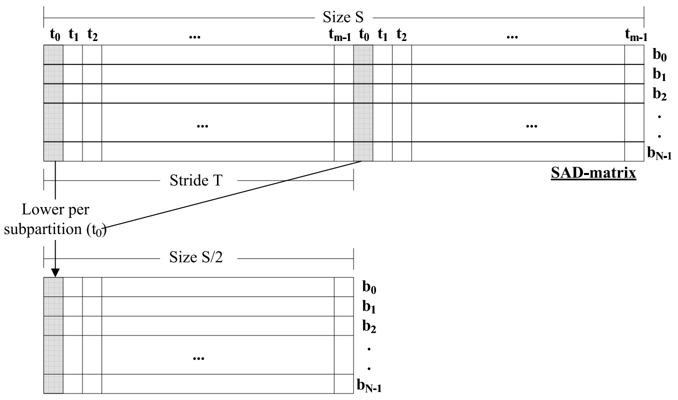


Figure 2: Building SAD costs.

generated the motion information of 256 positions for all partitions/sub-partitions and the aim of the reduction is to obtain the position with the lowest cost for each partition/sub-partition of the 256 positions. A binary reduction per partition/sub-partition is carried out (b_0 to b_{N-1} in Figure 3, where N is the number of partitions/sub-partitions), overwriting the information previously stored in shared memory. Initially, the matrix size (S) is 256 (2^8) positions, and by using the binary reduction in 8 iterations it is possible to obtain the best position of each partition/sub-partition. Note that in order to avoid shared memory bank conflicts where possible (shared memory is organized in 16/32 memory banks depending on the GPU used), the reduction is carried out using strides of half the remaining positions (128, 64, 32, 16, 8, 4, 2 and 1).



Size S
t0 t1 t2 ... tm-1 t0 t1 t2 ... tm-1
b0
b1
b2
bN-1
SAD-matrix
Stride T
Lower per subpartition (t0)
Size S/2

Figure 3: Proposed binary reduction.

An independent kernel (second kernel) performs the last reduction. Typically, in a search area there are more than 256 positions, e.g. using 32 as search range there are 4096 positions. Therefore, an extra kernel is required to perform the final reduction. The reduction procedure is the same as that explained for kernel #1 but using different data, e.g. using 32 as search range this reduction would be performed using 4096/256 = 16 elements per partition/sub-partition.

4.2. *Sub-Pixel inter prediction*

Before starting with sub-pixel inter prediction, the reference frame must be sub-sampled to quarter-pixel accuracy because it was transferred to GPU memory with full-pixel accuracy, i.e. each full-pixel is converted into sixteen sub-pixels (the frame size is multiplied by four in each direction, see Figure 4). These sixteen sub-pixels can be further classified into full-pixels, sub-pixels with half-pixel accuracy and sub-pixels with quarter-pixel accuracy.

Half-pixels are obtained by means of a 6-tap filter for which six full-pixels or six half-pixels are required. Quarter-pixels are obtained by means of a bilinear filter, for which two half-pixels are required. Note that there are dependencies in sub-pixel generation (half-pixels are required to generate other half-pixels and half-pixels are required to generate the quarter-pixels). Therefore, in order to avoid these dependencies, sub-pixel generation is performed using more than one GPU kernel. Three GPU kernels are configured with as many threads as there are full-pixels in a frame: the first one obtains two half-pixels, the second one obtains one half-pixel and the third one obtains all quarter-pixels.

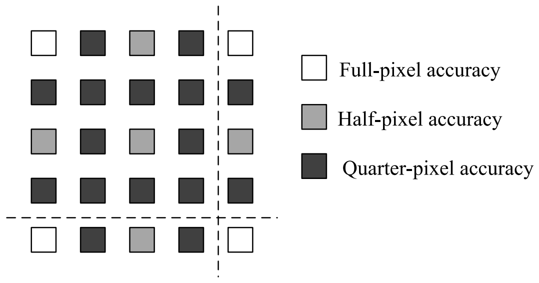


Figure 4: Sub-pixel generation.

Sub-pixel inter prediction is performed in two steps; half-pixel refinement and quarter-pixel refinement. The algorithm for both steps is the same, but applied over different data. The best full-pixel Motion Vector (MV) (obtained by full-pixel inter prediction) of each partition/sub-partition becomes the center point for half-pixel refinement and the best half-pixel MV (obtained by half-pixel refinement) of each partition/sub-partition becomes the center point for quarter-pixel refinement (Figure 5). The best quarter-pixel MV of each partition/sub-partition is the required output of the proposed algorithm. Nine positions are evaluated in the half– and quarter–pixel refinement, since each full–pixel is surrounded by nine half–pixels and each half–pixel is surrounded by nine quarter–pixels.

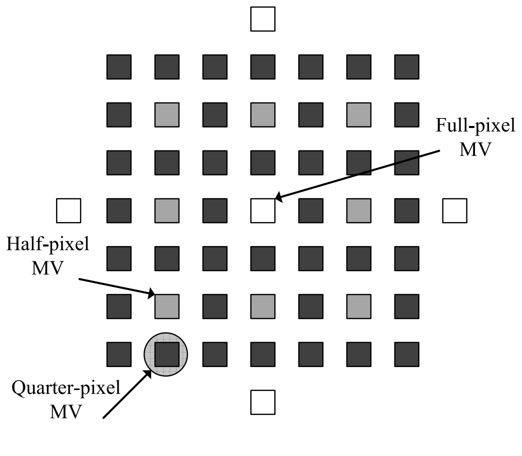


Figure 5: Sub-pixel MV refinement.

The algorithm for sub-pixel inter prediction is similar to the algorithm used for full-pixel inter prediction: we divide the MB into sixteen 4x4 blocks and each one takes as its starting point the appropriate MV (full-pixel MV or half-pixel MV of each partition/sub-partition). However, in this case, the cost of the 4x4 sub-partition cannot be used to build the cost of higher partitions/sub-partitions, and must be recalculated for each partition/sub-partition. All 4x4 blocks will take the same MV to perform the 16x16 partition and the final cost will be obtained using atomic GPU operations, or all 4x4 blocks will take different MVs to perform the 4x4 sub-partition and no extra operations will be needed.

4.3. Temporal and inter–view MVP calculation

So far, nothing has been said about the metric used to evaluate which is the best MV, just cost has been mentioned. However, from this point onward, this will be very important. The cost calculation is based on Equation 1, which is a standard equation and the one used in the JM reference encoder.

$$Cost = SAD_{cost} + \lambda * vector_{bits} \tag{1}$$

where SAD_{cost} is the metric used to calculate the differences (SAD cost for full-pixel inter prediction and Hadamard SAD cost for sub-pixel inter prediction), λ is an encoder parameter which depends on the quantization parameters (QP) used and $vector_{bits}$ is the number of bits required to encode the MV minus the MV predictor (MVP).

This equation is very important because it is a big challenge for parallel programming, since there are dependencies between adjacent MBs. The MVP is calculated using the motion information previously obtained for neighbouring MBs, as depicted in Figure 6. It shows an MB and its neighbouring MBs involved in the MVP calculation. If there is more than one partition in the neighbouring MBs (in Figure 6, the left and upper MBs are divided into more than one partition), the nearest partition to the top–left corner of the current MB is selected in order to calculate the MVP (see A and B partitions in Figure 6). The MVP is calculated as the median of the three selected partitions (A, B, and C partitions in Figure 6). Therefore, in order to obtain the cost for a certain position of each MB, the cost of neighbouring MBs must be calculated in advance.

At this point we need to eliminate or mitigate these dependencies. An initial solution could be not to use the MVP in Equation 1. Therefore, $vector_{bits}$ would be the number of bits required to encode the MV. However, this initial solution causes significant RD degradation, so it is necessary to find a procedure to predict the movement and mitigate these dependencies.

The proposed GPU-based inter prediction algorithm is executed concurrently for a complete frame, so the only way to predict the movement is by extracting some information from previously coded frames. As MVP we propose to use the MV of the MB located in the same position but in the previously coded frame. However, this proposal only partially solves the problem. The proposed algorithm in a 3D video encoder is applied in two different ways: by using a reference frame from

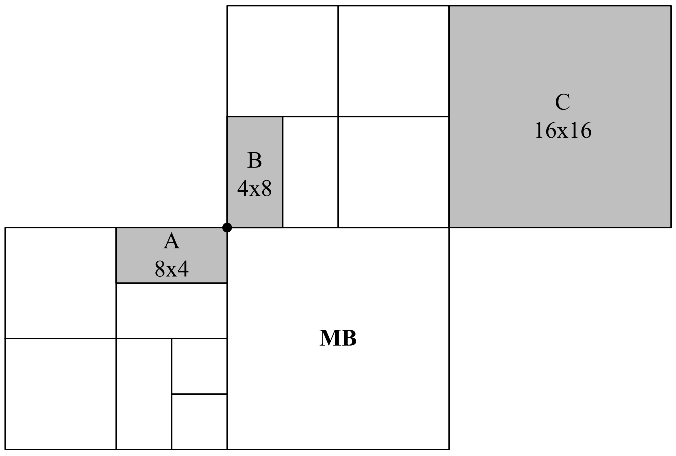


Figure 6: MV predictors.

the same view (temporal prediction) and by using a reference frame from a different view (inter-view prediction). This proposal solves the problem of RD degradation when the inter prediction algorithm is applied using a reference frame from the same view, but suffers from high RD degradation when the ME algorithm is applied using a reference frame from a different view. In other words, this MVP is able to predict the movement of a video sequence but is not able to predict the camera's distance when the inter–view prediction is applied. In the evaluation section, we refer to this proposal as Last Prediction MVP (LP–MVP). Figure 7 shows how the MVP calculation is carried out in this method (following the encoding order).

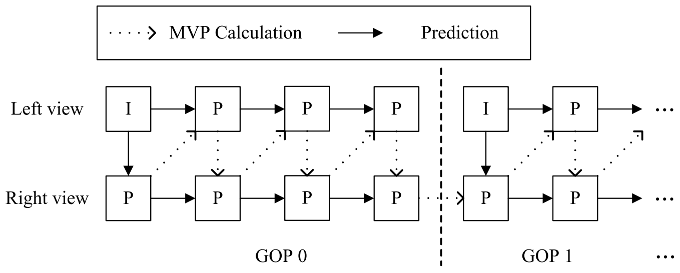


Figure 7: LP–MVP method.

Finally, due to the fact that there are two different ways of applying the proposed algorithm, we propose to update the MVP in two different ways. When using a reference frame from the same view, the MVP is calculated using the MV of the MB located in the same position but in the previously temporarily predicted frame. When using a reference frame from a different view the MVP is calculated using the MV of the MB located in the same position but in the previously inter-view predicted frame. In the evaluation section, we refer to this proposal as Same Kind of Prediction MVP (SKP–MVP). Figure 8 shows how the MVP calculation is carried out in this method. In comparison with the previously described method, the MVPs of the first frame in the right view of each GOP (inter–view prediction) are not calculated using the motion information of the last frame in the right view of the preceding GOP, but are calculated using the motion information of the previously inter–view predicted frame. Similarly, the MVPs of the first P frame in the left view of each GOP (temporal prediction) are calculated using the motion information of the previously temporarily predicted frame (last frame in the right view of the preceding GOP).

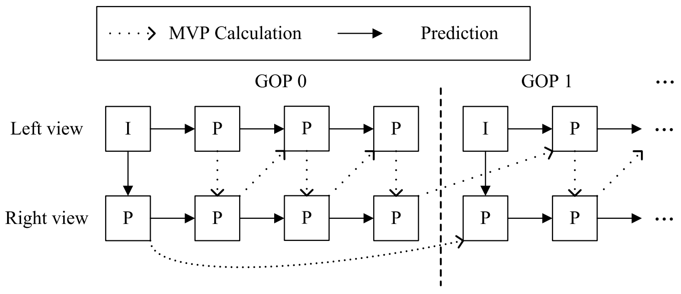


Figure 8: SKP–MVP method.

4.4. *GPU–based 8x8 transform*

As has been stated in the above section, the cost metric used for sub–pixel inter prediction is Hadamard SAD, which is a product–free transform. This metric is more complex than the SAD metric used for full–pixel inter prediction, but improves the encoding efficiency. This is because after the inter prediction module, the residual data is transformed and quantified.

Originally, the Hadamard SAD is calculated using a 4x4 transform, one for each of the sixteen 4x4 blocks into which an MB is divided. The cost metric for higher partitions is obtained by adding the output of these 4x4 transforms. However, by default, the profile used for coding 3D video sequences in the H.264/AVC JM reference software (Stereo High Profile) also allows the use of an 8x8 transform to obtain the encoding cost. The cost of the smallest partitions (4x4, 4x8, 8x4 and 8x8) is obtained by adding the output of 4x4 transforms, and the cost of the biggest partitions (8x8, 8x16, 16x8 and 16x16) is obtained by adding the output of 8x8 transforms. Note that the cost of the 8x8 partition can be obtained by using either 4x4 transforms or an 8x8 transform. The 8x8 transform is more complex than the 4x4 transform, and requires six times more add/subtract operations.

Initially, the code of the 8x8 transform was not ported to GPU code, and the encoding cost of all partitions was obtained by adding the output of 4x4 transforms, introducing some drift errors. This initial solution is evaluated in the performance evaluation section (the SKP–MVP proposal includes this scenario).

Then, the code of the 8x8 transform was ported to GPU code. However, the parallelism for the biggest partitions (8x8, 8x16, 16x8 and 16x16) is greatly reduced since when using 4x4 transforms, sixteen GPU threads perform one 4x4 transform and now four threads have to perform one 8x8 transform (an MB is divided into four 8x8 blocks). As a consequence, there are fewer GPU threads obtaining the costs, and the GPU execution time is almost doubled. Moreover, one must remember that the 8x8 transform is more complex.

Fortunately, the 8x8 transform can be carried out in parallel by different GPU threads. In the sub–pixel GPU kernels, sixteen threads have been configured to handle each one of the sub–pixel positions. Therefore, the aim of this last improvement is that sixteen GPU threads should cooperate to obtain the cost of four 8x8 transforms.

The 8x8 transform can be parallelized, but there are some limitations that affect the final performance:

- Extra synchronization barriers. An 8x8 transform must be carried out in four steps, and each step is carried out by four GPU threads in parallel. The first one obtains the SAD coefficients to which the transform is going to be applied; the second one, in column fashion, performs the first part of the 8x8 transform (each thread processes two complete columns); the third one, in row fashion, performs the second part of the 8x8 transform (each thread processes two complete rows); and the final one adds the resulting coefficients after applying the 8x8 transform (the 64 coefficients are added to find the final cost). Three synchronization barriers are required since there are data dependencies between the four steps.

- Thread divergences. The final step, which was described above, only needs to be executed on one of the four GPU threads performing the 8x8 transform. This method is faster than introducing more synchronization barriers to obtain partial results (each thread adding the coefficients of two complete rows or columns of the 8x8 matrix obtained) and using atomic GPU operations to obtain the final cost.

- Extra shared memory. In order to perform the transform in parallel, different GPU threads need to communicate with each other, and the way to do this is through GPU shared memory. A data structure to store the partition information for all positions (the cost for a certain partition is calculated in parallel for all positions) is required. The size of this structure is 9 KB (1 KB per position), limiting the number of thread blocks that can be marked as active on each one of the available GPU processors.

- Algorithm complexity. The 8x8 transform requires six times more add/subtract operations, so it is six times more complex than the 4x4 transform. If we compare it with the previous proposal, one 8x8 transform is going to be executed instead of executing four 4x4 transforms. However, the complexity continues to be higher and 50% more add/subtract operations are required.

This final proposal is evaluated in the next section and is labeled as the SKPMVP method plus the GPU–based 8x8 transform (SKP-MVP + 8x8T).

5. Performance evaluation

In order to evaluate the proposed algorithm, it was implemented in the H.264/AVC JM 17.2 reference software encoder [22]. The parameters used for the evaluation were those included in the Stereo High Profile of the said reference software. Only some parameters were changed in the configuration file. The number of reference frames was set to 1 and RD-Optimization was disabled in order to keep the complexity as low as possible. Different values for these parameters were also tested, and similar conclusions were obtained. The tests were carried out using 3D full HD sequences (1080p, 1920 x 1080 pixels), with each view rated at 25 frames per second (25Hz). The QP was varied between 28, 32, 36 and 40, according to [23]. The search range was set to 32, which means 4096 search area positions. The GOP pattern was set as shown in Figure 9.

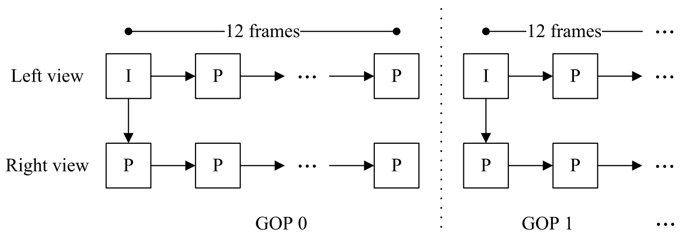


Figure 9: Configured GOP pattern.

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

5.1. System

To evaluate the performance of the proposed algorithm, the following development environment was used: the host machine used was an Intel Core i7 running at 2.80 GHz with 6GB of DDR3 memory. The GPU used was an NVIDIA GTX480 with an NVIDIA driver and CUDA support (260.19). The operating system for this scheme was Linux Ubuntu 10.4 x64 with GCC 4.4. Table 1 shows the main GPU features.

Table 1: GTX480 features

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

5.2. Metrics

The following metrics were used to evaluate the proposal:

5.2.1. RD function

In the definition of the RD function, the PSNR is the distortion for a given bit–rate. The average global PSNR is based on a standard Equation (Equation 2). The Luminance PSNR is multiplied by four, since the YUV input files are in the format 4:2:0, which is composed of four 8x8 blocks for the luminance component and only two 8x8 blocks for the chrominance components.

$$PSNR = \frac{4 * PSNR_Y + PSNR_U + PSNR_V}{6} \tag{2}$$

5.2.2. Time Reduction and Speed–up

This is to evaluate the time saved by the proposed algorithm. Time Reduction (TR) follows Equation 3 and speed–up is based on Equation 4.

$$TR(\%) = \frac{T_{JM} - T_{FI}}{T_{JM}} * 100 \tag{3}$$

$$Speed\text{–}up = \frac{T_{JM}}{T_{FI}} \tag{4}$$

T_{JM} denotes the coding time required by the H.264/AVC JM 17.2 reference software, and T_{FI} is the time taken by JM using the algorithm proposed in this paper. T_{FI} also includes all the computational costs for the operations needed in order to prepare the information required by our proposal.

5.2.3. ΔPSNR and Δbit–rate

The experiments were carried out on the test sequences using four QPs, namely, 28, 32, 36 and 40. The detailed procedures for calculating bit–rate and PSNR differences can be found in the work by Bjøntegaard [23], and make use of Bjøntegaard and Sullivan's common test conditions [24]. These procedures have been recommended by the JVT Test Model Ad Hoc Group. The YUV files used for comparing the PSNR results are the original YUV file at the input of the H.264/AVC JM 17.2 reference software and the one obtained after decoding the H.264/MVC video stream using the H.264/AVC JM 17.2 reference software decoder.

5.2.4. Power and Energy Consumption

It is known that current GPUs suffer from high power consumption requirements. Consequently, power and energy consumption become essential metrics in this kind of studies. With the aim of sampling the power consumed by the whole system including the Power Supply Unit (PSU), we developed a data logger device capable of collecting this data and transmitting it to a computer. This device analyses the magnetic field produced by an electric current flowing through a straight conductor and is capable of sampling and reconstructing the resulting wave, whatever its form, and processing it in order to obtain an average value. More details about the profiling system used can be found in [25].

5.3. Results

Table 2 shows the ΔPSNR and the Δbit–rate results obtained when coding five 3D 1080p sequences using the LP–MVP method, the SKP–MVP method and the SKP–MVP + 8x8T method, compared with the reference full search ME algorithm. These results are shown separately for the left view, for the right view and for both views.

As mentioned above, this ME proposal using the LP–MVP method obtains an acceptable RD degradation for the left view (all frames use temporal prediction). On average, the left view obtains a Δbit–rate of 2.16% and a ΔPSNR of −0.048 dB. However, it obtains unacceptable RD degradation for the right view (some frames use inter-view prediction and some frames use temporal prediction. On average, the right view obtains a Δbit–rate of 20.53% and a ΔPSNR of −0.514 dB. This RD degradation is due to the fact that the MVPs for inter-view predicted frames are based on the motion information obtained for a frame which uses temporal prediction, and there is one of this kind of frames per GOP. Note that the RD degradation produced in one frame is propagated to the frames for which it is the reference.

The ME proposal using the SKP–MVP method obtains an acceptable RD degradation for both views. On average, the left view obtains a Δbit–rate of 1.92% and a ΔPSNR of −0.043 dB and the right view obtains a Δbit–rate of 6.12% and a ΔPSNR of −0.147 dB. This drop in RD performance is negligible if the computational savings are taken into account (Table 3 and Table 4). If we compare the RD performance obtained by this SKP–MVP method with the LP–MVP method, the left view obtains slightly better RD results because the *MVP* for all frames using temporal prediction is always based on the motion information obtained for a frame which uses temporal prediction, and the right view obtains considerably better RD results because the *MVP* for inter-view predicted frames is based on previously coded inter-view predicted frames.

Finally, the ME proposal using the SKP–MVP + 8x8T method improves upon the RD results when using only the SKP–MVP method, since it is able to obtain more accurate motion information. On average, both views obtain a Δbit–rate of 2.23% and a ΔPSNR of −0.053 dB, which is better than the results obtained when using only the SKP–MVP method (Δbit–rate of 3.40% and a ΔPSNR of −0.082 dB).

Table 3 and Table 4 show the timing results of our proposed H.264/AVC JM encoders when coding five 3D 1080p sequences using the SKP–MVP method and using the SKP–MVP + 8x8T method, compared with the reference full search ME algorithm. Note that the LP–MVP method is not included since the timing results are almost the same as the ones obtained when using the SKP–MVP method. The timing results are shown for the proposed GPU code (ME column) and for the complete H.264/AVC encoder (Encoder column). Table 3 shows the TR(%) results and Table 4 shows the speed–up results.

The ME proposal using the LP–MVP method or the SKP–MVP method shows significant improvements, obtaining a speed–up of nearly 10x on average, which means a time reduction of 89.74% for the complete H.264/AVC JM encoder, and a speed–up of over 67x on average, which means a time reduction of 98.52% for the proposed ME algorithm.

Table 3: TR(%) results for the proposed encoders compared with the full search algorithm

	SKP–MVP		SKP–MVP + 8x8T	
Sequence	ME	Encoder	ME	Encoder
Beergarden	98.90	92.12	98.41	91.92
Cafe	98.12	87.43	97.28	86.66
CarPark	98.67	90.56	98.07	90.03
Hall	98.19	87.92	97.37	87.16
Street	98.69	90.67	98.09	90.09
Average	98.52	89.74	97.84	89.17

Table 4: Speed–up results for the proposed encoders compared with the full search algorithm

	SKP–MVP		SKP–MVP + 8x8T	
Sequence	ME	Encoder	ME	Encoder
Beergarden	91.60	12.70	63.00	12.37
Cafe	53.35	7.96	36.74	7.50
CarPark	75.36	10.59	51.87	1.03
Hall	55.23	8.28	38.02	7.79
Street	76.28	10.72	52.36	10.09
Average	67.43	9.75	46.40	9.23

Finally, the ME proposal using the SKP–MVP + 8x8T method obtains slightly worse results than the ones obtained for the LP–MVP and SKP–MVP methods, but shows significant improvements when compared with the reference H.264/AVC JM encoder and reports better RD results than the ones reported for the LP–MVP and SKP–MVP methods (see Table 2). As mentioned in the algorithm description section, the 8x8 transform is six times more complex than the 4x4 transform and requires more synchronization barriers, thus lowering the algorithm's performance. This proposal obtains a speed–up of over 9x on average, which means a time reduction of 89.17% for the complete H.264/AVC JM encoder, and obtains a speed–up of over 46x on average, which means a time reduction of 97.84% for the proposed ME algorithm.

Table 5 shows the ΔPSNR and the Δbit–rate results obtained when coding five 3D 1080p sequences using the LP–MVP method, the SKP–MVP method and the SKP–MVP + 8x8T method compared with the reference UMHexagonS ME algorithm. The UMHexagonS ME algorithm is a *fast* ME algorithm implemented in the JM reference encoder. It is based on not exploring all available positions within the search area following a hexagonal search pattern. Therefore, it introduces certain RD degradations when compared with full search ME, but it is considerably faster. These results are shown separately for the left view, for the right view and for both views.

Experimental results show that the three proposed encoders

Table 2: RD performance compared with the full search algorithm

			LP–MVP		SKP–MVP		SKP–MVP + 8x8T	
Sequence	#frames	view	Δbit–rate%	ΔPSNR(dB)	Δbit–rate%	ΔPSNR(dB)	Δbit–rate%	ΔPSNR(dB)
Beergarden	150	left	1.71	-0.057	1.55	-0.051	0.82	-0.026
		right	51.04	-1.351	3.66	-0.105	1.60	-0.045
		both	20.21	-0.592	2.32	-0.070	1.11	-0.033
Cafe	200	left	2.38	-0.053	2.16	-0.048	1.15	-0.026
		right	19.78	-0.500	3.90	-0.094	2.43	-0.059
		both	9.55	-0.250	2.82	-0.071	1.64	-0.042
CarPark	250	left	0.90	-0.022	0.86	-0.021	0.49	-0.012
		right	21.09	-0.495	14.42	-0.351	12.39	-0.299
		both	7.66	-0.188	5.40	-0.135	4.47	-0.111
Hall	200	left	3.88	-0.067	3.42	-0.060	2.10	-0.037
		right	6.39	-0.117	4.33	-0.079	2.95	-0.054
		both	5.01	-0.095	3.83	-0.073	2.50	-0.047
Street	250	left	1.91	-0.042	1.63	-0.036	1.30	-0.028
		right	4.35	-0.108	4.30	-0.106	1.82	-0.046
		both	2.80	-0.065	2.64	-0.061	1.42	-0.033
Average		left	2.16	-0.048	1.92	-0.043	1.17	-0.026
		right	20.53	-0.514	6.12	-0.147	4.24	-0.101
		both	9.05	-0.238	3.40	-0.082	2.23	-0.053

Table 5: RD performance compared with the the UMHexagonS algorithm

			LP–MVP		SKP–MVP		SKP–MVP + 8x8T	
Sequence	#frames	view	Δbit–rate%	ΔPSNR(dB)	Δbit–rate%	ΔPSNR(dB)	Δbit–rate%	ΔPSNR(dB)
Beergarden	150	left	-0.20	0.006	-0.36	0.011	-1.07	0.035
		right	21.09	-0.638	-16.67	0.575	-18.31	0.634
		both	8.80	-0.263	-7.38	0.243	-8.47	0.280
Cafe	200	left	-2.02	0.046	-2.23	0.051	-3.19	0.072
		right	-2.67	0.065	-15.69	0.465	-16.88	0.502
		both	-2.55	0.062	-8.60	0.219	-9.65	0.246
CarPark	250	left	-0.41	0.010	-0.45	0.011	-0.81	0.020
		right	-6.52	0.169	-11.61	0.318	-13.21	0.366
		both	-2.91	0.072	-4.94	0.124	-5.77	0.146
Hall	200	left	-4.60	0.092	-5.01	0.100	-6.22	0.123
		right	-10.34	0.227	-12.03	0.265	-13.18	0.290
		both	-7.58	0.156	-8.60	0.178	-9.77	0.204
Street	250	left	-2.13	0.048	-2.40	0.055	-2.72	0.062
		right	-5.07	0.125	-5.13	0.126	-7.33	0.187
		both	-3.38	0.087	-3.54	0.091	-4.67	0.119
Average		left	-1.87	0.040	-2.09	0.046	-2.80	0.062
		right	-0.70	-0.010	-12.23	0.350	-13.78	0.396
		both	-1.52	0.023	-6.61	0.171	-7.67	0.199

surpass the RD performance obtained when compared with the UMHexagonS ME algorithm for all tested sequences, except for the right view of the Beergarden sequence using the LP–MVP method. This behaviour is due to this video sequence having a static background, which is the worst scenario for the LP–MVP method, since the MVPs for inter-view predicted frames are based on the motion information obtained for a frame which uses temporal prediction, thus providing a highly inaccurate MVP.

Table 6 and Table 7 show the timing results of our proposed H.264/AVC JM encoders when coding five 3D 1080p sequences using the SKP–MVP method and using the SKP–MVP + 8x8T method, compared with the reference UMHexagonS ME algorithm. The timing results are shown for the proposed GPU code (ME column) and for the complete H.264/AVC encoder (Encoder column). Table 6 shows the TR(%) results and Table 7 shows the speed–up results. As mentioned above, the UMHexagonS ME algorithm is a *fast* ME algorithm and, as expected, the TR and the speed–up results obtained are smaller than the ones obtained when compared with the full search ME algorithm.

Table 6: TR(%) results for the proposed encoders compared with the UMHexagonS algorithm

	SKP–MVP		SKP–MVP + 8x8T	
Sequence	ME	Encoder	ME	Encoder
Beergarden	78.55	34.18	68.82	32.33
Cafe	73.86	29.65	62.03	25.32
CarPark	75.03	30.06	63.72	26.16
Hall	75.29	31.11	64.12	26.86
Street	75.95	31.11	64.96	26.82
Average	75.74	31.23	64.73	27.50

Table 7: Speed–up results for the proposed encoders compared with the UMHexagonS algorithm

	SKP–MVP		SKP–MVP + 8x8T	
Sequence	ME	Encoder	ME	Encoder
Beergarden	4.66	1.52	3.21	1.48
Cafe	3.82	1.42	2.63	1.34
CarPark	4.01	1.43	2.76	1.35
Hall	4.05	1.45	2.79	1.37
Street	4.16	1.45	2.85	1.37
Average	4.12	1.45	2.84	1.38

The ME proposal using the SKP–MVP method continues showing significant improvements, obtaining a speed–up of nearly 1.5x on average, which means a time reduction of 31.23% for the complete H.264/AVC JM encoder, and a speed–up of over 4x on average, which means a time reduction of 75.74% for the proposed ME algorithm. The ME proposal using the SKP–MVP + 8x8T method obtains a speed–up of nearly 1.4x on average, which means a time reduction of 27.50% for the complete H.264/AVC JM encoder, and a speed–up of over 2.8x on average, which means a time reduction of 64.73% for the proposed ME algorithm.

Table 8 shows the average power consumption, the execution time and the total energy consumed when coding one GOP (24 frames, 12 frames per view) for the complete test computer when coding five 3D 1080p sequences. The first main column shows them for the reference H.264/AVC JM encoder using the full search ME algorithm (without any modification), the second main column shows them for the H.264/AVC JM encoder using the SKP–MVP method, and the third one shows them for the H.264/AVC JM encoder using the SKP–MVP + 8x8T method. Additionally, there is an extra result for the GPU–based encoders, which shows the ratio between the energy consumed by the reference H.264/AVC JM encoder and the proposed H.264/AVC JM encoders. On average, the energy consumption for the GPU-based encoder using the SKP–MVP method is 9.61 times better than for the reference encoder, and the ratio using the SKP–MVP + 8x8T method is 8.91. This small drop in energy savings is due to the fact that the proposal using the GPU–based 8x8 transform is more complex than the one which uses the 4x4 transform, and the GPU kernels consume more time (see Figure 10 for a graphical analysis). Note that the average power consumption of the proposed encoders (248W and 250W) is about 30% greater than when using the reference encoder (184W). This increase is not much if you consider that when the GPU is in execution the power consumption is over 350W (almost doubling the power consumption of the reference encoder), and the reason is that the GPU is in execution about 12%-16% of the total average encoding time.

Table 9 shows the average power consumption, the execution time and the total energy consumed when coding one GOP (24 frames, 12 frames per view) for the complete test computer when coding five 3D 1080p sequences. The first main column shows them for the reference H.264/AVC JM encoder using the UMHexagonS ME algorithm (without any modification), the second main column shows them for the H.264/AVC JM encoder using the SKP–MVP method, and the third one shows them for the H.264/AVC JM encoder using the SKP–MVP + 8x8T method. As in the previous table, there is an extra result for the GPU–based encoders which shows the ratio between the energy consumed by the reference H.264/AVC JM encoder and the proposed H.264/AVC JM encoders. This table shows that the average power consumption when using the UMHexagonS ME algorithm is very similar to the one obtained when using the full search ME algorithm (see Table 8), but the execution time is considerably shorter. Therefore, the total energy consumed by this reference algorithm is significantly smaller. In fact, the GPU–based encoders are faster but consume more energy than the reference algorithm and there are no significant energy savings or energy penalties.

Figure 10 shows an extract from the power consumption over time (5 seconds) for the complete test computer, when coding the Beergarden sequence for the reference H.264/AVC JM encoder using the full search ME algorithm, for the pro-

Table 8: Energy consumption for coding a GOP compared with the full search algorithm

Sequence	Reference encoder			SKP–MVP				SKP–MVP + 8x8T			
	Power (Watts)	Time (seconds)	Energy (Joules)	Power (Watts)	Time (seconds)	Energy (Joules)	Ratio	Power (Watts)	Time (seconds)	Energy (Joules)	Ratio
Beergarden	184.59	613.68	113,279.19	246.69	40.57	10,008.21	11.32	245.24	44.46	10,903.37	10.39
Cafe	184.13	427.17	78,654.81	247.04	40.63	10,037.23	7.84	247.13	43.54	10,760.04	7.31
CarPark	184.65	531.32	98,108.24	248.66	40.98	10,190.09	9.63	248.06	43.58	10,810.45	9.08
Hall	183.82	482.67	88,724.40	249.83	40.30	10,068.15	8.81	250.64	42.76	10,717.37	8.28
Street	183.62	583.45	107,133.09	250.96	40.91	10,266.77	10.43	251.69	43.39	10,920.83	9.81
Average	184.16	527.66	97,179.95	248.64	40.68	10,114.09	9.61	250.35	43.55	10,902.74	8.91

Table 9: Energy consumption for coding a GOP compared with the UMHexagonS algorithm

Sequence	Reference encoder			SKP–MVP				SKP–MVP + 8x8T			
	Power (Watts)	Time (seconds)	Energy (Joules)	Power (Watts)	Time (seconds)	Energy (Joules)	Ratio	Power (Watts)	Time (seconds)	Energy (Joules)	Ratio
Beergarden	184.92	60.29	11,148.83	246.69	40.57	10,008.21	1.11	245.24	44.46	10,903.37	1.02
Cafe	182.30	57.44	10,471.31	247.04	40.63	10,037.23	1.04	247.13	43.54	10,760.04	0.97
CarPark	183.93	56.32	10,358.94	248.66	40.98	10,190.09	1.02	248.06	43.58	10,810.45	0.96
Hall	184.77	54.81	10,127.24	249.83	40.30	10,068.15	1.01	250.64	42.76	10,717.37	0.94
Street	184.83	56.38	10,420.72	250.96	40.91	10,266.77	1.02	251.69	43.39	10,920.83	0.95
Average	184.15	57.05	10,505.76	248.64	40.68	10,114.09	1.04	250.35	43.55	10,902.74	0.96

posed GPU–based H.264/AVC JM encoder using the SKP–MVP method, and for the proposed GPU–based H.264/AVC JM encoder using the SKP–MVP + 8x8T method. Note that the power consumption over time when using the reference UMHexagonS ME algorithm is not shown in this graph because it is similar to the one shown when using the reference full search ME algorithm, but the execution time is shorter (Table 8 and Table 9). When the encoder process begins all encoders consume the same power (around 180–185 Watts), but when the GPU starts working the power consumption of the proposed encoders increases. They have a power consumption of around 335-350 Watts (see power consumption peaks in Figure 10). Moreover, if we focus on the first power consumption peak of the GPU–based encoders, it is clear that it is longer when using the 8x8 transform since it is more complex and the GPU consumes more time to perform it. Finally, we should mention that the power consumption for the CPU code in the proposed encoders is around 235 Watts, which is higher than for the reference execution (180–185 Watts) because the GPU is always active, waiting for new kernels.

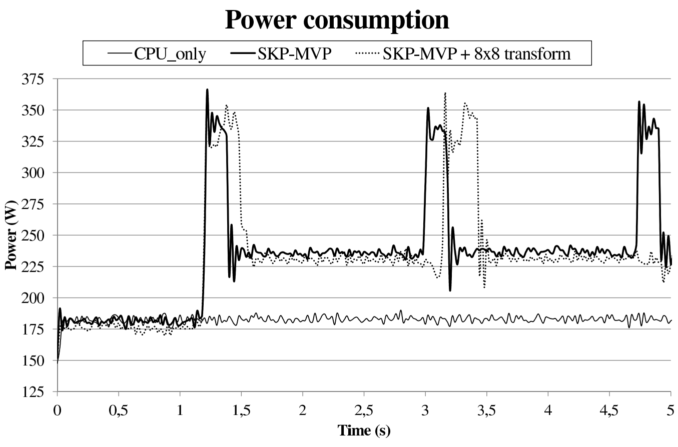


Figure 10: Power consumption.

6. Conclusions

This paper presents two GPU–based inter prediction approaches for H.264/MVC. The methods presented try to mitigate sequential dependencies when coding adjacent MBs in parallel in a 3D scene while using a simplified way to obtain the motion information, and a more accurate but more complex method to obtain the motion information. Both methods are tested against two of the most well–known ME algorithms implemented in the reference H.264/AVC JM encoder.

The proposal using the simplified way of obtaining the motion information obtains the best timing and energy saving results, but obtains slightly worse RD results than the ones obtained by the proposal using the most accurate approach. It obtains a speed–up of nearly 10x for the complete H.264/MVC encoder when compared with the full search ME algorithm and a speed–up of nearly 1.5x when compared with the UMHexagonS ME algorithm. The energy consumption of the proposed H.264/MVC encoder is 9.61 times better than that of the reference H.264/AVC JM encoder when using full search ME and the energy efficiency is maintained when compared with the execution of the UMHexagonS ME algorithm.

The proposal using the accurate approach obtains a speed–up of over 9x for the complete H.264/MVC encoder when compared with the full search ME algorithm and a speed–up of nearly 1.4x when compared with the UMHexagonS ME algorithm. The energy consumption of the proposed H.264/MVC encoder is 8.91 times better than that of the reference H.264/AVC JM encoder when using full search ME and the energy efficiency is maintained when compared with the execution of the UMHexagonS ME algorithm.

Finally, these proposals obtain an acceptable RD degradation when compared with the execution of the full search ME algorithm, and an RD improvement when compared with the execution the UMHexagonS ME algorithm.

Acknowledgments

This work was supported by the Spanish MEC and MICINN funds, under the grants TIN2009-14475-C04-03 and TIN2012-38341-C04-04.

References

[1] ISO/IEC International Standard 14496-10:2005. Information technology Coding of audio-visual objects Part 10: Advanced Video Coding; 2005.

[2] Ohm J-R, Sullivan G J, Schwarz H, Tan T K, Wiegand T. Comparison of the Coding Efficiency of Video Coding Standards - Including High Efficiency Video Coding (HEVC). IEEE Transactions on Circuits and Systems for Video Technology 2012; 22(12): 1669-1684.

[3] Wiegand T, Sullivan G J, Bjøntegaard G, Luthra A. Overview of the H.264/AVC Video Coding Standard. IEEE Transactions on Circuits and Systems for Video Technology 2003; 13(7): 560-576.

[4] ISO/IEC International Standard 14496-10:2007. Advanced Video Coding for Generic Audiovisual Services (MVC extension); 2007.

[5] NVIDIA. NVIDIA CUDA Compute Unified Device Architecture-Programming Guide Version 3.2; 2010.

[6] Rodríguez–Sánchez R, Martínez J L, Fernández–Escribano G, Sánchez J L, Claver J M. A Fast GPU-Based Motion Estimation Algorithm for HD 3D Video Coding. In: 10th IEEE International Symposium on Parallel and Distributed Processing and Aplications; July 2012. 166-173.

[7] Richardson I E G. The H.264 Advanced Video Compression Standard, 2nd Edition. John Wiley and Sons 2010.

[8] Vetro A, Wiegand T, Sullivan G J. Overview of the Stereo and Multiview Video Coding Extensions of the H.264/AVC Standard. Proceedings of the IEEE 2011; 99(4): 626-642.

[9] Merkle P, Smolic A, Muller K, Wiegand T. Efficient prediction structure for multiview video coding. IEEE Transaction Circuits System for Video Technology 2007; 17(11): 1461-1473.

[10] Lee C-Y, Lin Y-C, Wu C-L, Chang C-H, Tsao Y-M, Chien S-Y. Multi-Pass and Frame Parallel Algorithms of Motion Estimation in H.264/AVC for Generic GPU. In: Proceedings of the IEEE International Conference on Multimedia and Expo; 2007. 1603-1606.

[11] Ryoo S, Rodrigues C, Baghsorkhi S, Stone S, Kirk D. Hwu W-M. Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming; February 2008. 73-82.

[12] Chen W-N, Hang H-M. H.264/AVC motion estimation implementation on Compute Unified Device Architecture (CUDA). In: Proceedings of IEEE International Conference on Multimedia and Expo(ICME); June 2008. 679-700.

[13] Lu J B, Cai H, Lou J G, Li J. An epipolar geometry-based fast disparity estimation algorithm for multiview image and video coding. IEEE Transactions on Circuit and System for Video Technology 2007; 17(6): 737-750.

[14] Huo J Y, Chang Y L, Li M, Ma Y Z. Scalable prediction structure for multiview video coding. In: Proceedings of IEEE International Symposium on Circuits and Systems; May 2009. 2593-2596.

[15] Ding L, Tsung P, Chien S, Chen W, Chen L. Content-aware prediction algorithm with inter-view mode decision for multiview video coding. IEEE Transactions on Multimedia 2008; 10(8): 1553-1563.

[16] Zeng H, Ma K K, Cai C. Mode-correlation-based early termination mode decision for multi-view video coding. In: 17th IEEE International Conference on Image Processing; September 2010: 34053408.

[17] Liu X, Yang L T, Sohn K. High-speed inter-view frame mode decision procedure for multi-view video coding. Future Generation Computer Systems 2012; 28(6): 947956.

[18] Li X M, Zhao D B, Ji X Y, Wang Q, Gao Q. A fast inter frame prediction algorithm for multi-view video coding. In: Proceedings of IEEE International Conference on Image Processing; September 2007: 417-420.

[19] Shen L, Liu Z, Liu S, Zhang Z, An P. Selective disparity estimation and variable size motion estimation based on motion homogeneity for multiview coding. IEEE Transaction on Broadcast 2009; 55(4): 761-766.

[20] Shen L, Liu Z, An P, Ma R, Zhang Z. Low-Complexity Mode Decision for MVC. IEEE Transactions on Circuits and Systems for Video Technology 2011; 21(6): 837843.

[21] AHG on Multiview Video Coding, ISO/IEC JTC1/SC29/WG11 Docs: N6501 N7829; 2007.

[22] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Reference Software to Committee Draft: JVT-F100 JM17.2; 2011. http://iphome.hhi.de/suehring/tml/.

[23] Bjøntegaard G. Calculation of Average PSNR Differences between RD-Curves. In: the 13th VCEG-M33 Meeting; April 2001.

[24] Sullivan G, Bjøntegaard G. Recommended Simulation Common Conditions for H.26L Coding Efficiency Experiments on Low-Resolution Progressive-Scan Source Material. In: ITU-T VCEG Doc. VCEG-N81; September 2001.

[25] Rodríguez-Sánchez R, Martínez J L, Fernández-Escribano G, Sánchez J L, Claver J. M, Díaz P. Optimizing H.264/AVC Inter Prediction on a GPU-based Framework. In: Technical Report DIAB–11–01–2, Department of Computing Systems, University of Castilla-La Mancha; 2011.

Rafael Rodríguez–Sánchez obtained his M.S. and Ph.D. degrees in Computer Science from the University of Castilla-La Mancha, Spain, in 2010 and 2013, respectively. Since 2008 he has been working at the Institute for Research in Informatics, UCLM, RAAP group. His research interests include video coding, parallel programming and heterogeneous computing.

José Luis Martínez received his Ph.D. degree in Computer Science from the UCLM, Spain, in 2009. In 2010, he joined the department of DACyA at the Complutense University in Madrid, where he was assistant professor. In 2011 he came back to UCLM. His research interests include parallel video processing. He has also been a visiting researcher in the USA and the UK.

Gerardo Fernández–Escribano received his M.Sc. degree in Computer Engineering and Ph.D. degree from the University of Castilla-La Mancha, Albacete, Spain, in 2003 and 2007, respectively. In 2004, he joined the Department of Computer Engineering at the University of Castilla-La Mancha, where he is currently an Associate Ph.D. Professor at the School of Industrial Engineering.

José L. Sánchez received his PhD degree from the Technical University of Valencia, Spain, in 1998. Since November 1986 he has been a member of the Computer Systems Department at the University of Castilla-La Mancha. He is currently an associate professor of computer architecture and technology. His research interests include multicomputer systems, on- and off-chip networks, parallel programming, and heterogeneous computing.

José M. Claver received his M.Sc. degree in Physics (1984) from the University of Valencia, and a Ph.D. degree in Computer Science from the Technical University of Valencia (Cum Laude, 1998), Spain. He is Associate Professor in the Department of Computer Science, University of Valencia. His research interests include computer architecture, parallel computing, computer networks, embedded systems, and video coding.

Jan De Cock obtained his M.S. and Ph.D. degrees in Engineering from Ghent University, Belgium, in 2004 and 2009, respectively. Since 2004 he has been working at the Multimedia Lab, Ghent University, iMinds. In 2010, he obtained a postdoctoral fellowship from the Research Foundation Flanders (FWO). His research interests include high-efficiency (scalable) video coding and transcoding, and multimedia applications.

Bart Pieters received his B.Sc. and M.Sc. degrees in Informatics from Ghent University, Belgium, in 2004 and 2006, respectively. At the end of 2006, he joined the Multimedia Lab of the Department of Electronics and Information Systems at Ghent UniversityIBBT (Belgium) as a researcher and in 2012, he finished his Ph.D. on parallel video coding on the GPU. After working as a postdoctoral researcher for a year, he started his own company working on GPU technology in 2013.

Rik Van de Walle received his M.Sc. and PhD degrees in Engineering from Ghent University, Belgium, in 1994 and 1998, respectively. After a visiting scholarship at the University of Arizona (Tucson, USA), he returned to Ghent University, where he became professor of multimedia systems and applications, and head of the Multimedia Lab.