Título artículo / Títol article:	H.264/AVC inter prediction on accelerator-based multi-core systems
Autores / Autors	Rodríguez Sánchez, Rafael ; Martínez, José Luis ; Fernández Escribano, Gerardo ;Sánchez, José L. ; Claver, José M.
Revista:	Multimedia Tools and Applications, October 2013, Volume 66, Issue 3
Versión / Versió:	Post-print del autor
Cita bibliográfica / Cita bibliogràfica (ISO 690):	RODRÍGUEZ-SÁNCHEZ, Rafael, et al. H. 264/AVC inter prediction on accelerator-based multi-core systems. *Multimedia tools and applications*, 2013, vol. 66, no 3, p. 361-381.
url Repositori UJI:	http://hdl.handle.net/10234/92611

Multimed Tools Appl
DOI 10.1007/s11042-012-1056-6

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

Rafael Rodríguez-Sánchez · José Luis Martínez · Gerardo Fernández-Escribano · José Luis Sánchez · José Manuel Claver

Abstract The AVC video coding standard adopts variable block sizes for inter frame coding to increase compression efficiency, among other new features. As a consequence of this, an AVC encoder has to employ a complex mode decision technique that requires high computational complexity. Several techniques aimed at accelerating the inter prediction process have been proposed in the literature in recent years. Recently, with the emergence of many-core processors or accelerators, a new way of supporting inter frame prediction has presented itself. In this paper, we present a step forward in the implementation of an AVC inter prediction algorithm in a graphics processing unit, using Compute Unified Device Architecture. The results show a negligible drop in rate distortion with a time reduction, on average, of over 98.8 % compared with full search and fast full search, and of over 80 % compared with UMHexagonS search.

Keywords AVC · Motion estimation · Inter prediction · Heterogeneous computing

R. Rodríguez-Sánchez (✉) · G. Fernández-Escribano · J. L. Sánchez
Instituto de Investigación en Informática de Albacete, Universidad de Castilla-La Mancha, Avenida de España s/n, 02071 Albacete, Spain
e-mail: rrsanchez@dsi.uclm.es

G. Fernández-Escribano
e-mail: gerardo@dsi.uclm.es

J. L. Sánchez
e-mail: jsanchez@dsi.uclm.es

J. L. Martínez
Architecture and Technology of Computing Systems Group, Complutense University, Ciudad Universitaria s/n, 28040 Madrid, Spain
e-mail: joseluis.martinez@fdi.ucm.es

J. M. Claver
Departamento de Informática, Universidad de Valencia, Avenida de la Universitat s/n, 46100 Burjassot, Valencia, Spain
e-mail: jose.claver@uv.es

Published online: 14 March 2012

1 Introduction

ISO/IEC 14496–10 | ITU-T Rec. H.264 or Advanced Video Coding (AVC) [8] is the newly-established video coding standard which represents the best state-of-the-art video compression and achieves much higher compression than the other previously existing video coding standards at the same quality. The main purpose of AVC is to offer a good quality standard able to considerably reduce the output bit rate of the encoded sequences, compared with previous standards, with a view to being used in a variety of applications such as DVD, video-streaming, HDTV, etc., while exhibiting a substantially improved definition of quality and image. AVC promises a significant advance compared with the commercial standards currently most in use (MPEG-2 and MPEG-4) [7].

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

One of these new features is AVC inter prediction, which supports motion compensation block sizes ranging from 16×16 to 4×4, with many options available between them. AVC inter prediction supports motion compensation block sizes ranging from 16×16, 16×8, 8×16 to 8×8, where each of the sub-divided regions is a MacroBlock (MB) partition. If the 8×8 mode is chosen, each of the four 8×8 block partitions within the MB may be further split in 4 ways: 8×8, 8×4, 4×8 or 4×4, which are known as MB sub-partitions. Then, the Motion Compensation (MC) and the Motion Estimation (ME) processes are carried out for each partition and sub-partition, with these processes taking up most of the time of the encoding algorithm. More specifically, ME can take up more than 90 % of the total encoding time. Inter prediction is implemented in the reference software by means of the tree-structured MC algorithm.

Therefore, it seems reasonable to accelerate ME in order to reduce the total encoding time. Different computing platforms can be used for this purpose. Interesting cases are the heterogeneous architectures. Examples of this type of platforms include Graphics Processing Units (GPUs), Cell Broadband Engines (Cell BEs), and Field-Programmable Gate Arrays (FPGAs). GPUs, for instance, consist of tens or hundreds of similar processing cores which are designed and organized with the goal of achieving higher performance [5]. The demand for these devices comes primarily from consumer applications, including multimedia and computer or console gaming.

The progress of GPUs is now the focus of a great deal of attention. They have changed from fixed pipelines to programmable pipelines, and the hardware design also includes multiple cores, bigger memory sizes and better interconnection networks that offer practical and acceptable solutions for speeding up both graphics and non-graphics applications. GPUs are highly parallel and are normally used as coprocessors to assist the CPU with massive data computations.

For example, NVIDIA® has developed a powerful GPU architecture called Compute Unified Device Architecture (CUDA) [15], which consists of a Single Instruction Multiple Data (SIMD) computing device. The main feature of these devices is a large number of processing elements integrated into a single chip at the expense of a significant reduction in cache memory. For instance, the architecture of the NVIDIA® GPUs is organized as a set of SIMD multiprocessors called Stream Multiprocessors (SM). Each SM has multiple processing elements and a set of resources shared by all cores. Each core executes the same instruction at the same clock cycle but on different data. GPUs also have an external DRAM memory which can be classified depending on its access mode.

The ME algorithm proposed in the AVC encoding algorithm fits well in the GPU philosophy, and offers a new challenge for GPUs. The main issue is how to efficiently distribute all the computations over the GPU. This paper proposes an algorithm to perform the integer-pixel and fractional-pixel ME developed for the AVC over a GPU. In order to evaluate the algorithm, both in terms of time and in terms of Rate Distortion (RD) performance, the algorithm is integrated in the AVC JM reference software, using CUDA. The algorithm obtains on average a speed up of over 86 compared with full search and fast full search, and of over 5 compared with UMHeagonS search.

The rest of the paper is organized as follows: Section 2 presents related proposals; the details of the implementation of our approach and its performance evaluation are presented in Sections 3 and 4, respectively; finally, the conclusions and future work are set out in Section 5.

2 Related work

In the AVC JM reference software, inter prediction is the most computationally-expensive task. It is carried out in two steps: MC and ME. In the literature, many approaches have been proposed in order to accelerate these processes. Most of them are based on estimate data using faster algorithms, aiming to determine which MB partitions are not suitable to be selected for encoding the current MB, based on its characteristics or determining stopping criteria for the MB mode selection algorithms. But, until now, there have not been many solutions which make use of GPUs to accelerate this highly complex algorithm, which is the major focus of this paper: to exploit the powerful GPU architecture to accelerate the AVC JM reference software ME algorithm.

In the framework of video processing using GPUs, one of the pioneering approaches was developed by Kelly and Kokaram in 2004 [11]. In this work, the authors propose using computer graphics hardware for fast image interpolation. Basically, the authors implemented the well-known full search block-matching ME algorithm by using the OPENGL API. The results show a speedup of up to fourfold. However, the ME algorithm is only a part of the current video coding standards. In the same year, Chen et al. in [3], exploiting hyper-threading architectures, parallelized the AVC encoding algorithm using the OpenMP programming model for Intel architectures. The authors obtained speedups of up to fourfold. This implementation was not a GPU-based approach but was instead a hyper-threading one.

In 2006, Ho et al. in [6] presented an ME algorithm for AVC using GPUs based on a block-by-block approach and providing a mechanism which is able to adjust the arithmetic intensity to maximize performance on different GPUs.

In 2007, Lee et al. in [13] presented multi-pass and frame parallel algorithms to accelerate the AVC ME using a GPU. They unroll and rearrange the multiple nested loops involved in the ME algorithm by using the multi-pass method over the GPU.

In 2008, Chen and Hang in [2] proposed an implementation of the AVC ME algorithm using CUDA. The algorithm is based on an efficient block-level parallel algorithm for the variable block size ME in AVC. They decompose the AVC ME algorithm into 5 steps so that they can achieve highly parallel computation. Also in 2008, Kung et al. in [12] presented a GPU-based ME for 4×4 blocks. They rearrange the 4×4 block encoding order in order to overcome the dependencies between adjacent MBs. However, AVC defines more block sizes for ME.

In 2009, Schwalb et al. in [17] presented a small diamond search ME adapted to the programming model of modern GPUs. They reduce the execution time compared with the

Unsymmetrical Multi-Hexagon search (UMHexagonS) ME algorithm. The coding efficiency is lower than the one reported by UMHexagonS. In the same year, Momcilovic and Sousa in [14] proposed a scalable parallelization approach for AVC ME. They obtain the Motion Vectors (MVs) by applying efficient data reusing techniques and exploiting the power of the GPUs.

Finally in 2010, Cheung et al. in [4] proposed a GPU implementation of the simplified Unsymmetrical Multi-Hexagon search (smpUMHexagonS) ME algorithm. The authors divide the current frame into multiple tiles. Each tile is processed by a single GPU thread, and different tiles are processed by different independent threads concurrently on the GPU. They report significant bitrate increases (12 %) with a penalty in quality (0.4 dB) depending on the sequence and the tile length.

The present approach uses the same starting point as [2], but our work (as will be shown below) offers better performance than that obtained in [2]. This is mainly due to an improved usage of memory transfers (our algorithm performs as few memory transfers and allocations as possible, since memory transfers and allocations can degrade the algorithm´s performance), a considerable reduction in bank conflicts and an adequate definition and management of data structures. Moreover, in [2] there are some parts of the algorithm that are not parallelized, such as the seven different MB-partitions, which does not occur in this work. Therefore, the present work improves the AVC ME procedure, creating a more parallel and more efficient procedure. Moreover, the work presented in [2] does not offer RD results, because they only implement the ME algorithm and do not include it in any AVC encoder. Thus, the speedups are meaningless since the quality and bit rate are as important as the time reduction.

In a nutshell, there are not many approaches focused on AVC ME implementation in GPUs using CUDA, and most of them do not analyze RD performance. The time reduction, or the speedup, is a very important feature that can be achieved by using GPUs, but all the approaches must keep the RD as close as possible to the reference software, which is a sequential approach to be run on a traditional CPU. Moreover, there are some approaches that do not focus on video coding standards and only try to parallelize a part of them, such as ME. In the AVC video coding standard, ME is only a part of the whole encoding algorithm and, therefore, all the approaches should try to combine all of the coding tools in order to show how this affects the rest of the modules. At this point, the approach presented in this paper offers an implementation of the AVC reference software video-encoding algorithm using a GPU as a co-processor of the CPU. Our new approach is integrated in the AVC JM 17.2 reference software [9] and offers a high speed up with a negligible RD drop.

3 Proposed inter prediction on CUDA

The main challenge of this approach is to efficiently support the tree-structured MC algorithm executed in the AVC JM 17.2 reference software encoder [9] for P frames. The algorithm is divided into two main steps: integer-pixel ME and fractional-pixel ME.

Before starting with the execution of the GPU code, some data must be transferred to the GPU DRAM. The data which does not change during the execution is transferred once per sequence and is allocated to the GPU constant memory: frame dimensions, search range, search area dimensions, number of positions inside the search area, and distribution of positions within the search area, which has been changed to exploit locality in memory accesses. Figure 1.a shows the search area distribution used by the JM reference software,

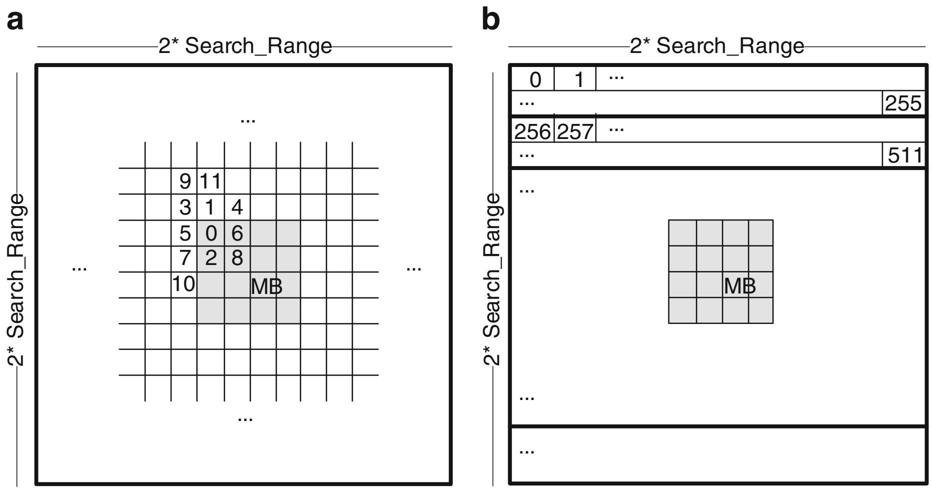


Fig. 1 Search area distribution, a) spiral pattern, b) custom pattern

which follows a spiral pattern and position 0 corresponds to the center of the search area. Figure 1b shows our custom search area distribution, which follows the raster scan order and position 0 corresponds to the top-left corner of the search area and the positions are distributed by rows. Position values shown in Fig. 1b are an example, and they depend on the Search_Range parameter.

At the beginning of coding each P frame, the search area MV predictors (MV$_p$s) are calculated and transferred to the GPU DRAM. The search area MV$_p$s are calculated using the motion information from the previous frame, unlike in a sequential execution in which the motion information of neighbor MBs can be used. We use as MV$_p$ the 16×16 MV of the MB located in the same position in the previous frame. The frame to be coded and the reference frame are also transferred to the GPU DRAM.

3.1 Integer-pixel motion estimation

The proposed algorithm for integer-pixel ME is divided into three steps which need to be executed sequentially following a highly-parallel procedure by using the GPU: calculate 4× 4 Sum of Absolute Differences (SAD) costs, build structured motion tree and perform a reduction. The steps are executed once per frame, obtaining the MVs for all MB partitions and sub-partitions with integer-pixel accuracy, in parallel.

The goal of the first step is to obtain the 4×4 SAD-Submatrix for each MB in a frame, as depicted in Fig. 2. For this purpose, a GPU thread is generated for each search area position ((2*Search_Range)^2 search area positions per MB) and 256 threads are grouped into a GPU thread block. Contiguous threads are mapped to contiguous search area positions as depicted in Figs. 1b and 2. The SAD is computed in 4×4 blocks; thus, each thread obtains 16 SAD costs resulting from dividing the MB into 4×4 blocks (b_0 to b_{15} in Fig. 2), so a thread obtains all 4×4 SAD costs associated to a certain MB candidate position in the search area. Therefore, each column of the 4×4 SAD-Submatrix depicted in Fig. 2 corresponds to one position checked inside the search area, which is calculated by the same GPU thread. Intermediate results for this kernel (4×4 SAD-Submatrix) are stored in GPU registers for faster memory access by the second step.

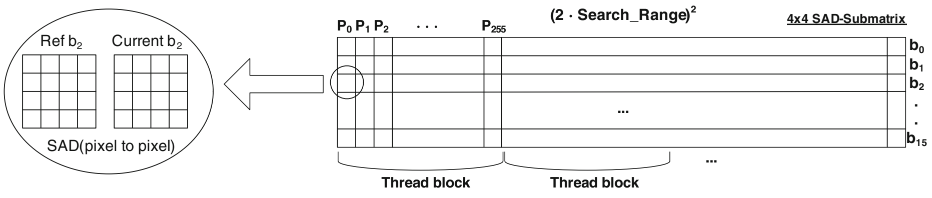


Fig. 2 Obtaining 4×4 SAD costs

The second step is performed by the same GPU kernel used to perform the first step. By using the 4×4-block SAD calculations the second step obtains the SAD costs for the different partitions and sub-partitions required in the tree-structured MC algorithm. Figure 3 shows how to build the SAD costs for a given position (Pst) for the different MB partitions and sub-partitions starting with the 4×4 SAD costs. Our algorithm only needs to add two 4×4 SAD costs to obtain one 8×4 SAD cost and so on. This procedure is possible because we use a common search area for all MB partitions and sub-partitions, unlike a sequential implementation which can calculate separate MV_ps and separate search areas for the different MB partitions and sub-partitions. Furthermore, to compute the final costs, the Lagrangian cost equation is used. The Lagrangian cost is defined as $SAD_{costs} + \lambda * vector_{bits}$, where $vector_{bits}$ is the number of bits required to encode the MV - MV_p and λ depends on the Quantification Parameter (QP) used to encode the sequences. Intermediate results are stored in multiprocessor local shared memory for the final step.

Finally, the same GPU kernel performs a partial reduction (third step) of the generated data. At this point, each thread block contains the motion information for all partitions and sub-partitions for 256 correlative search area positions and applying a binary reduction for each partition/sub-partition obtains the best MV for each partition/sub-partition of the associated positions. Figure 4 shows a generic binary reduction in which each thread involved in the reduction procedure (t_0 to t_{m-1}, where m is the number of threads involved in the reduction procedure) performs a reduction for each row in multiprocessor shared memory (b_0 to b_{N-1}, where N is the number of rows in multiprocessor shared memory involved in the reduction procedure). Note that m is half of the remaining positions in any of the eight iterations needed to reduce from 256 positions to 1. In order to complete the reduction process, eight iterations are needed, starting with 256 (2^8) Lagrangian costs per

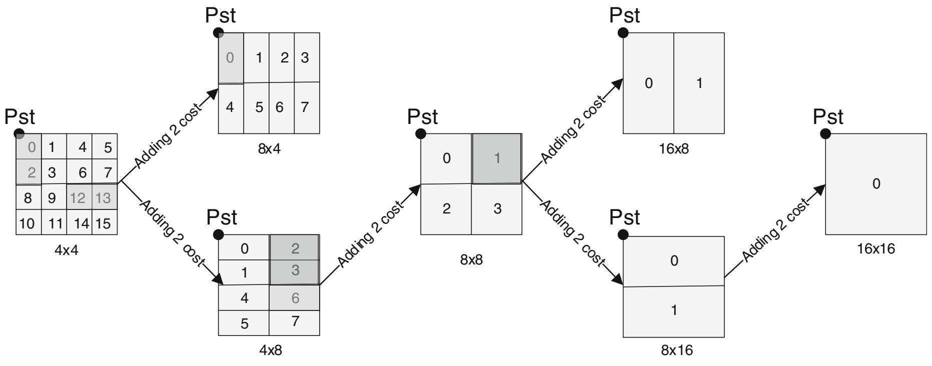


Fig. 3 SAD cost building for different MB partitions

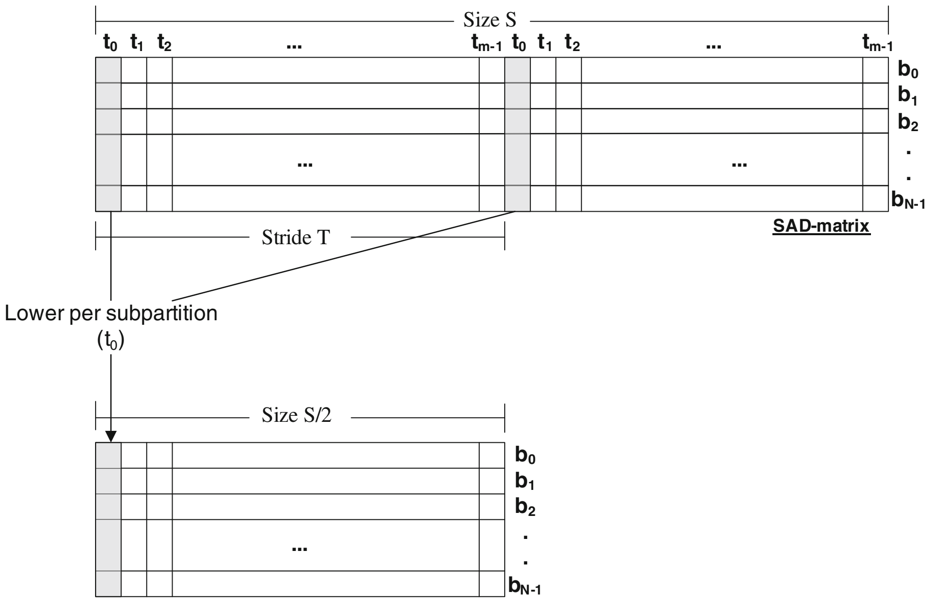

Size S
t0 t1 t2 ... tm-1 t0 t1 t2 ... tm-1
b0
b1
b2
bN-1
SAD-matrix
Stride T
Lower per subpartition (t0)
Size S/2

Fig. 4 Binary reduction scheme

partition/sub-partition and finishing with one Lagrangian cost per sub-partition, where the SAD-matrix size is reduced by half in each iteration. The reductions are performed with SAD-matrix sizes (S) of 256, 128, 64, 32, 16, 8, 4, and 2 using T strides, such that T = S/2, to obtain the best Lagrangian cost per block. These strides are chosen to avoid local shared memory bank conflicts. The code for the eight iterations is unrolled to avoid unnecessary loop climbs. Intermediate results are allocated to multiprocessor shared memory.

At the end of this kernel, each thread block launched for execution returns the best 41 Lagrangian costs (all MB partitions of the tree-structured MC algorithm) associated to 256 search area positions, and the resulting data is allocated to global GPU memory.

An independent GPU kernel performs the final reduction following the same binary reduction explained above. At this point, in GPU global memory there are still $(2{*}\mathrm{Search\_Range})^2/256$ Lagrangian costs for each partition/sub-partition. This last kernel performs a final reduction per sub-partition to obtain the best Lagrangian cost for each partition/sub-partition of each MB in a frame. Final results are located in the GPU global memory. When the GPU kernels have been completed, the final results are transferred to CPU main memory.

3.2 Fractional-pixel motion estimation

In order to further improve compression, the AVC standard assumes that the best match can be found in a region offset from the current MB (search area) by an integer number of pixels. However, for many MBs a better match can be obtained by searching a region interpolated to sub-pixel accuracy; for this case, a new prediction pixel is created by means of an interpolation of its neighbor. The AVC reference software supports quarter-pixel accuracy, which means that the image sizes are multiplied by four in each dimension or, in other words, one pixel is converted into sixteen sub-pixels. One of these sub-pixels is the pixel with full-pixel accuracy; three of them are the sub-pixels with half-pixel accuracy and the other twelve pixels are the sub-pixels with quarter-pixel accuracy.

As mentioned at the beginning of this section, the images are located in GPU DRAM with full-pixel accuracy. So, we need firstly to extend the reference images to sub-pixel accuracy. The sub-pixels with half-pixel accuracy are obtained by means of a 6-tap filter and the sub-pixels with quarter-pixel accuracy are obtained by a bilinear filter. One GPU thread per integer-pixel is generated and it applies both filters to obtain the fifteen sub-pixels.

The sub-pixel accuracy ME is performed in two steps: the first one is the half pixel refinement and the second one is the quarter pixel refinement, which are performed for all partitions. The best matching obtained for full-pixel accuracy becomes the center point for half-pixel refinement, and the best matching for half-pixel refinement becomes the center for quarter-pixel refinement. The algorithm for half-pixel and quarter-pixel refinement is the same, but applied over different data.

The algorithm for sub-pixel ME is similar to the algorithm used for full-pixel ME: we divide the MB into sixteen 4×4 blocks and each one takes as its starting point the appropriate MV, i.e., all 4×4 blocks will take the same MV to perform the 16×16 partition and the final cost will be obtained using atomic GPU operations. On the other hand, all 4×4 blocks will take different MVs to perform the 4×4 partition and no extra operations will be needed. The same reduction procedure used for full-pixel accuracy ME is used to obtain the best MV. However, there are two important things to take into account here. First, we cannot reuse the motion information from the smallest partition to obtain the Lagrangian cost of the higher partition because each partition has a different starting point (full-pixel MV or half-pixel MV). We have to recalculate the 4×4 cost for each partition. Second, the metric to compute the Lagrangian cost is the Hadamard SAD instead of SAD, as configured for the baseline profile in the AVC JM 17.2 [9] reference software used. Hadamard SAD is more complex than SAD, but it gives better results.

4 Performance evaluation

In order to test the performance of the algorithm proposed in this paper, it was integrated in the AVC JM 17.2 reference software encoder [9]. The parameters used in the AVC encoder configuration file were those included in the baseline profile of this reference software. Only some parameters were changed in the configuration file:

- The number of reference frames was set to 1 in order to keep the complexity as low as possible, because higher values imply excessive time consumption. An analysis using more reference frames is possible but the conclusions obtained will be the same regardless of the number of reference frames used, because the algorithm is executed as many times as reference frames configured.
- RD-Optimization was disabled for the same reason as the NumberReferenceFrames parameter.
- The GOP pattern was set to I(11)P.
- The tests were carried out with popular sequences in VGA format (640×480), 720p format (1280×720) and 1080p format (1920×1080), so the SourceWidth and Source-Height parameters were changed accordingly.
- Depending on the length of the sequence, the parameter FramesToBeEncoded was adjusted in order to encode the full sequence in each case.

- The Quantization Parameter (QP) called QPISlice and QPPSlice was varied between 28, 32, 36 and 40, according to [1, 10, 18].
- The search range was set to 32, which means 4096 positions inside the search area per MB partition and sub-partition.
- The frame rate parameter was set to 30 for VGA format (30 Hz) and 50 for 720p and 1080p formats (50 Hz).
- By default the SearchMode for testing was full search, but we also tested our approach using fast full search and Unsymmetrical Multi-Hexagon Search (UMHexagonS) mode as reference. More information about these ME algorithms can be found in [16].

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

4.1 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 6 GB 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.

4.2 Metrics

In order to evaluate the performance of our proposal the following metrics were used:

- RD function. In the definition of the RD function, the PSNR is the distortion for a given bit rate. The averaged global PSNR is based on Eq. 1. The Luminance PSNR is multiplied by four, since the YUV input files are in the format 4:2:0, which is composed of four 8×8 blocks for the luminance component and only two 8×8 blocks for the chrominance components.

$$\overline{PSNR} = \frac{4xPSNR_Y + PSNR_U + PSNR_V}{6} \tag{1}$$

Table 1 GPU main features

characteristic	GTX480
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	1.4GHZ

- TR and Speedup. This is to evaluate the time saved by the proposed algorithm. Time Reduction (TR) is based on Eq. 2 and Speedup is based on Eq. 3.

$$TR(\%) = \frac{T_{FI} - T_{JM}}{T_{JM}} \times 100 \tag{2}$$

$$Speedup = \frac{T_{JM}}{T_{FI}} \tag{3}$$

T_{JM} denotes the coding time used by the AVC JM 17.2 reference software, and T_{FI} is the time taken by 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.

- ΔPSNR and ΔBitrate. The experiments were carried out on the test sequences using four quantization parameters, namely, QP=28, 32, 36 and 40. The detailed procedures for calculating bitrate and PSNR differences can be found in the work by Bjøntegaard [1], and make use of Bjøntegaard and Sullivan's common test rule [18]. These procedures have been recommended by the JVT Test Model Ad Hoc Group [10]. The YUV files used for comparing the PSNR results are the original YUV file at the input of the AVC JM 17.2 reference software and the one obtained after decoding the AVC video stream using the AVC JM 17.2 reference software decoder.
- Frames per second (FPS). FPS shows the frames coded in a second by the different parts of the AVC encoder.

4.3 Results

Tables 2, 3 and 4 show that the RD performance is practically the same for our GPU-based approach as for three of the most well-known ME algorithms implemented by the reference AVC encoder (full search, fast full search and UMHexagonS search [16]). The penalty obtained by our algorithm compared with the full search and fast full search algorithms is allowed due to the TRs obtained, since most of the operations are executed in parallel. This

Table 2 RD Performance of the proposed GPU-based algorithm for VGA sequences

Sequence	Full Search		Fast Full Search		UMHexagonS Search	
	ΔPSNR (dB)	ΔBitrate (%)	ΔPSNR (dB)	ΔBitrate (%)	ΔPSNR (dB)	ΔBitrate (%)
Fun fair	−0.026	0.82	−0.032	1.02	0.045	−1.40
Harp	−0.068	2.25	−0.079	2.64	0.001	−0.02
Mobile	−0.082	2.36	−0.094	2.72	−0.019	0.49
Parade	−0.072	1.97	−0.083	2.29	−0.024	0.60
Sgi-ant	−0.082	2.33	−0.082	2.32	−0.016	0.49
Soft-football	−0.039	1.22	−0.039	1.21	0.080	−2.72
Tempete	−0.068	1.99	−0.081	2.35	−0.014	0.27
Waterfall	−0.072	2.66	−0.104	3.88	−0.026	0.98
mean	−0.064	1.95	−0.074	2.30	0.003	−0.16

Table 3 RD Performance of the proposed GPU-based algorithm for 720p sequences

Sequence	Full Search		Fast Full Search		UMHexagonS Search	
	ΔPSNR (dB)	ΔBitrate (%)	ΔPSNR (dB)	ΔBitrate (%)	ΔPSNR (dB)	ΔBitrate (%)
Coral	−0.027	1.13	−0.043	1.79	0.054	−2.08
Dolphins	−0.072	2.59	−0.087	3.11	0.199	−7.07
Mobile	−0.037	1.15	−0.068	2.16	0.031	−1.12
Nile	−0.104	4.03	−0.183	7.27	0.009	−0.31
Parkrun	−0.043	1.40	−0.045	1.48	−0.018	0.57
Shields	−0.043	1.38	−0.108	3.48	−0.038	1.00
Stockholm	−0.040	1.35	−0.105	3.87	−0.021	0.45
mean	−0.052	1.86	−0.091	3.31	0.031	−1.22

RD penalty is a consequence of some algorithm issues, such as the custom MV_ps or the search area distribution. As the ME for a given MB is calculated concurrently together with all the MBs of the current frame, the center point of the search area may not be adjusted by taking into account the information of the neighbor MBs because this information is not accessible. However, the algorithm described in this paper may determine sub-optimal MV_ps by using previous frames. On the other hand, our algorithm outperforms the coding efficiency of the UMHeaxagonS search algorithm.

Figure 5 shows the RD graphic results for the reference and the proposed approaches, for different sequences in VGA format and Fig. 6 shows them for 1080p format, from a value of 28 to 40 for QP, comparing them with the full search and UMHexagonS search algorithms. As can be seen from the figure, the PSNR vs. bit rate obtained with the proposed encoder, based on our algorithm, deviates slightly from the results obtained when applying the sequential reference encoders. Due to space limitations only a sub-set of the complete set of sequences is shown.

Table 5 shows the speedup, TR and its associated FPS for the ME process against three of the most well-known ME algorithms implemented by the reference AVC encoder (full

Table 4 RD Performance of the proposed GPU-based algorithm for 1080p sequences

Sequence	Full Search		Fast Full Search		UMHexagonS Search	
	ΔPSNR (dB)	ΔBitrate (%)	ΔPSNR (dB)	ΔBitrate (%)	ΔPSNR (dB)	ΔBitrate (%)
Blue sky	−0.064	2.08	−0.056	1.80	0.168	−5.19
Crowd	−0.118	3.86	−0.130	4.26	0.035	−1.25
Ducks	−0.037	1.20	−0.047	1.52	−0.002	−0.12
Into Tree	−0.074	3.66	−0.124	5.48	0.026	−1.18
ParkJoy	−0.077	2.33	−0.083	2.52	0.017	−0.61
Pedestrian	−0.051	2.27	−0.065	2.92	0.168	−6.79
Riverbed	0	−0.03	−0.004	0.15	0.024	−0.91
Tractor	−0.127	4.49	−0.127	4.47	0.557	−17.50
mean	−0.069	2.48	−0.080	2.89	0.124	−4.19

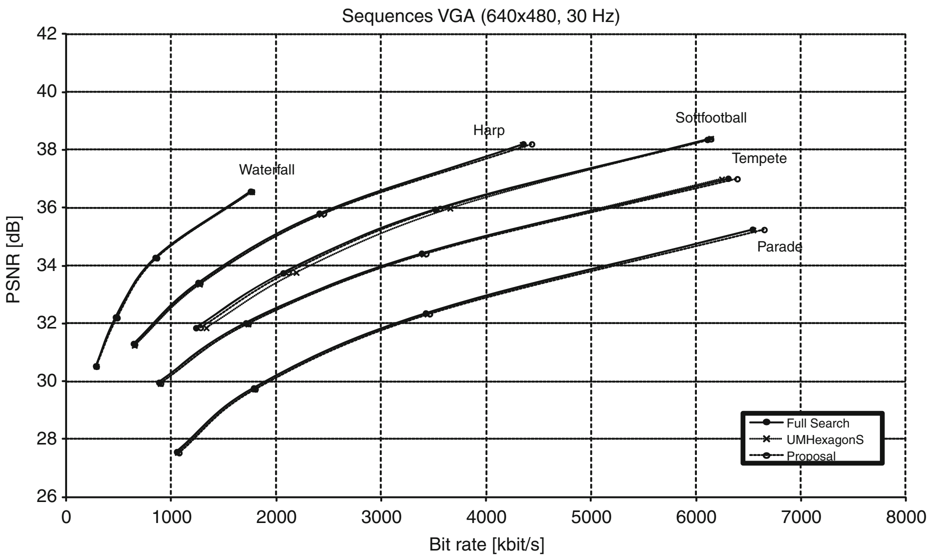


Fig. 5 RD results comparing the performance of the proposed and the AVC JM reference software encoder for VGA sequences

search, fast full search and UMHexagonS search [16]) for VGA format, Table 6 shows them for 720p format and Table 7 shows them for 1080p format. The first main column shows the differences between the GPU approach compared with the full search algorithm, in the second main column compared with the fast full search algorithm and in the third main column compared with the UMHexagonS search algorithm. Finally, the fourth main column

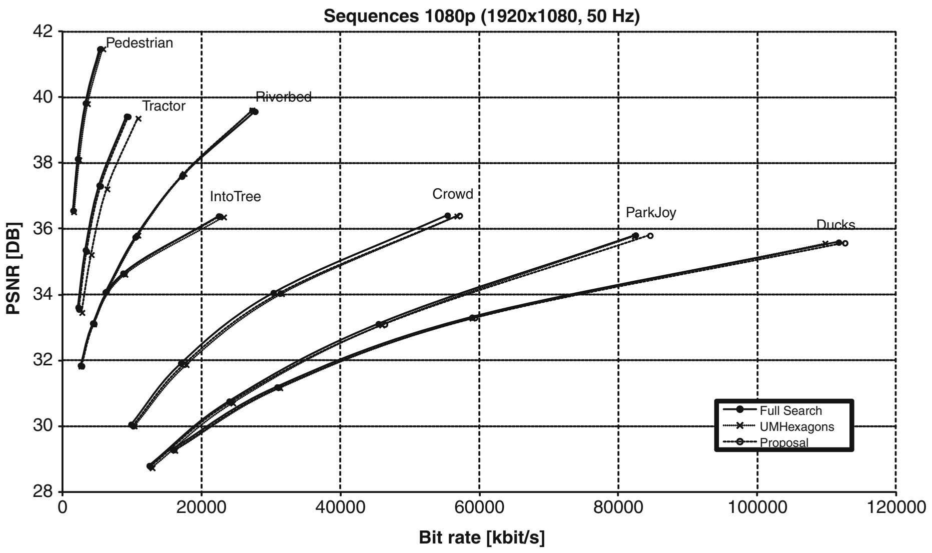


Fig. 6 RD results comparing the performance of the proposed and the AVC JM reference software encoder for 1080p sequences

Table 5 Time Reduction of the proposed ME for VGA sequences

Sequence	Full Search		Fast Full Search		UMHexagonS		FPS
	TR%	Speed up	TR%	Speed up	TR%	Speed up	
Fun fair	99.33	60.01	98.30	58.76	79.50	4.88	19.46
Harp	98.28	58.22	98.39	62.20	74.67	3.95	19.58
Mobile	98.53	67.84	98.28	58.06	78.25	4.60	19.45
Parade	98.55	69.04	98.27	57.74	76.87	4.32	19.47
Sgi-ant	97.66	42.74	98.30	58.92	64.24	2.80	19.79
Softfootball	98.77	81.43	98.34	60.10	85.03	6.68	19.23
Tempete	98.47	65.40	98.30	58.90	77.74	4.49	19.47
Waterfall	98.32	59.53	98.33	59.87	71.40	3.50	19.54
mean	98.34	60.42	98.33	59.71	75.25	4.04	19.50

shows the FPS for the ME process obtained by our proposed algorithm. The GPU approach considerably reduces the time requirements compared with all ME algorithms analyzed. On average, the algorithm obtains a speedup of over 52.1 (TR over 98 %) for all video content compared with the full search and fast full search algorithms, and it obtains a speedup of over 3.6 (TR over 72 %) for all video content compared with the UMHexagonS algorithm.

Table 8 shows the speedup, TR and its associated FPS, focusing exclusively on the parallelized process of the ME on the GPU (ME is composed of our parallel algorithm as well as of other parts that we do not port to the GPU, i.e. skip mode, ME structure initialization and MV_ps calculation) for VGA format, Table 9 shows them for 720p format, and Table 10 shows them for 1080p format. The first main column shows the differences between the GPU approach compared with the full search algorithm, the second main column shows the comparison with the fast full search algorithm and the third main column shows the comparison with the UMHexagonS search algorithm. Finally, the fourth main column shows the FPS obtained for the parallelized process of the ME. As expected after the analysis of Tables 5, 6 and

Table 6 Time Reduction of the proposed ME for 720p sequences

Sequence	Full Search		Fast Full Search		UMHexagonS		FPS
	TR%	Speed up	TR%	Speed up	TR%	Speed up	
Coral	97.20	35.77	98.42	63.26	62.36	2.66	6.57
Dolphins	98.03	50.88	98.40	62.66	75.41	4.07	6.49
Mobile	98.44	64.05	98.35	60.44	73.84	3.82	6.53
Nile	97.62	41.94	98.41	62.73	64.01	2.78	6.57
Parkrun	98.78	82.12	98.29	58.41	79.63	4.90	6.48
Shields	98.23	56.63	98.33	59.84	76.15	4.19	6.54
Stockholm	98.27	57.93	98.35	60.67	75.14	4.02	6.55
mean	98.08	52.18	98.36	61.10	72.36	3.62	6.54

Table 7 Time Reduction of the proposed ME for 1080p sequences

Sequence	Full Search		Fast Full Search		UMHexagonS		FPS
	TR%	Speed up	TR%	Speed up	TR%	Speed up	
Blue sky	97.72	43.84	98.42	63.44	72.35	3.62	2.89
Crowd	98.56	69.39	98.33	60.02	75.50	4.08	2.86
Ducks	98.84	85.87	98.31	59.17	76.64	4.28	2.86
Into Tree	98.53	67.99	98.43	63.53	71.86	3.55	2.87
ParkJoy	98.74	79.19	98.35	60.64	77.96	4.54	2.86
Pedestrian	97.64	42.42	98.41	62.92	71.30	3.48	2.90
Riverbed	98.86	87.87	98.38	61.69	83.12	5.92	2.83
Tractor	98.02	50.60	98.35	60.47	78.70	4.69	2.84
mean	98.36	61.11	98.37	61.44	75.93	4.15	2.86

7, the GPU approach achieves the best results focusing exclusively on the parallelized process of the ME. On average, the algorithm obtains a speedup of over 86.7 (TR over 98.80 %) for all video content compared with the full search and fast full search algorithms, and it obtains a speedup of over 5.1 (TR over 80.4 %) for all video content compared with the UMHexagonS algorithm.

Finally, Tables 11, 12 and 13 show the average FPS for the whole AVC encoding process obtained after encoding all sequences from a value of 28 to 40 for QP, depending on the sequence format. Also, they show the required time to encode a two-hour video (216000 frames for a VGA format at 30 Hz and 360000 frames for a 720p and 1080p format at 50 Hz), where the solution based on GPU achieves a considerable time reduction.

4.4 Comparison with other known results

In this section, we undertake a comparative analysis of our proposal with some of the most recent and prominent approaches in video coding using GPUs. In many cases, a comparative

Table 8 Time Reduction of the proposed GPU-based algorithm for VGA sequences

Sequence	Full Search		Fast Full Search		UMHexagonS		FPS
	TR%	Speed up	TR%	Speed up	TR%	Speed up	
Fun fair	98.99	99.37	98.97	97.18	86.12	7.21	32.48
Harp	98.96	95.80	99.02	102.39	82.27	5.64	32.50
Mobile	99.11	112.59	98.96	96.20	85.20	6.76	32.52
Parade	99.13	114.47	98.95	95.59	84.12	6.30	32.54
Sgi-ant	98.47	65.30	99.02	102.02	67.75	3.10	32.63
Softfootball	99.26	135.63	98.99	99.82	90.24	10.25	32.23
Tempete	99.08	108.46	98.97	97.55	84.77	6.57	32.54
Waterfall	98.98	98.45	98.99	99.11	79.72	4.93	32.62
mean	99.00	100.00	98.99	98.68	82.52	5.72	32.51

Table 9 Time reduction of the proposed GPU-based algorithm for 720p sequences

Sequence	Full Search		Fast Full Search		UMHexagonS		FPS
	TR%	Speed up	TR%	Speed up	TR%	Speed up	
Coral	98.30	58.99	99.05	104.92	71.80	3.55	10.99
Dolphins	98.82	85.01	99.05	104.87	83.18	5.93	10.96
Mobile	99.06	106.84	99.00	100.71	81.78	5.49	10.98
Nile	98.56	69.48	99.04	104.27	73.36	3.75	11.02
Parkrun	99.28	138.04	98.98	97.89	86.36	7.33	10.97
Shields	98.94	94.35	98.99	99.65	83.61	6.10	11.00
Stockholm	98.96	96.37	99.01	101.01	82.80	5.82	11.00
mean	98.85	86.75	99.02	101.84	80.41	5.10	10.99

evaluation is not possible because the algorithm is completely different. In [12] the authors only focus on 4×4 blocks for ME and in [14] the authors utilize different metrics to obtain the candidate MVs, namely SAD costs instead of Hadamard SAD costs for sub-pixel accuracy ME, which is less complex than the one we use. In other cases, the encoding conditions are completely different and the comparison is not fair. In [17] the authors use JM9.0 and they do not use Bjøntegaard and Sullivan's common test rule [18] for RD performance analysis.

However, although the authors of [2] only implement the ME algorithm for variable block sizes and do not include it in any AVC encoder (they cannot analyze the RD performance), they report that their algorithm running for CIF sequences and using 16 as search range is executed at 31.54 fps. Our algorithm using the same encoding conditions is executed at 177.5 fps. Note that our GPU is 2.6 times faster than the GPU used in their work.

In [4], the authors propose a GPU-based implementation of the well-know smpUMHexagonS ME algorithm. They partition each frame into multiple tiles, where each tile contains one or more MBs and each tile is processed by a single GPU thread. Table 14 shows the RD

Table 10 Time reduction of the proposed GPU-based algorithm for 1080p sequences

Sequence	Full Search		Fast Full Search		UMHexagonS		FPS
	TR%	Speed up	TR%	Speed up	TR%	Speed up	
Blue sky	98.64	73.60	99.06	106.73	80.71	5.18	4.90
Crowd	99.15	117.21	99.01	101.32	83.27	5.98	4.87
Ducks	99.31	145.90	99.00	100.30	84.25	6.35	4.88
Into Tree	99.13	114.36	99.07	107.02	80.33	5.08	4.87
ParkJoy	99.25	133.87	99.02	102.38	85.18	6.75	4.87
Pedestrian	98.59	70.81	99.05	105.36	79.80	4.95	4.90
Riverbed	99.33	149.19	99.04	104.64	89.06	9.14	4.84
Tractor	98.84	85.95	99.03	102.79	85.86	7.07	4.87
mean	99.03	103.03	99.04	103.77	83.56	6.08	4.88

Table 11 Two-hour sequence encoding time for VGA sequences

Feature	JM 17.2 Full Search	JM 17.2 Fast Full Search	JM 17.2 UMHexagonS Search	Proposal
fps	0.31	0.32	3.08	6.27
Time (Hours)	193.55	187.50	19.48	9.57

results for their algorithm as well as our RD results using the same encoding conditions. We have employed the same 720p sequences sampled at 60 Hz, selecting 64 as the search range and all pictures are encoded as P-frames except the initial I-frame. The comparison is achieved when comparing our results against the reference smpUMHexagonS (implemented in JM) with their results against smpUMHexagonS too. With their implementation, they obtain more degradation as many tiles are used due to the dependencies between neighboring MBs. However, we mitigate the degradation in our approach. Our algorithm outperforms the RD performance obtained by their fastest configurations (90 or more tiles); our algorithm has lower bitrate increments and lower PSNR losses than their algorithm for all video sequences.

Table 15 shows the execution time for the experiments carried out to fill the previous Table 14. Table 15 also shows the average execution time for each configuration. Note that the peak performance for our GPU is 1350 GFlops and the peak performance for the GPU used in [4] is 345.6 GFlops, which means that our GPU is 3.9 times more powerful. For this reason and for a fair comparison, we have included the column labeled as Index in Table 15, which shows the ratio between the average execution time obtained by their implementation for a certain encoder configuration and the average execution time for our implementation using the same encoder configuration. Higher values than 3.9 for this index mean that our algorithm is faster than their algorithm. In conclusion, our algorithm is as fast as their best configuration (index of 3.85) and it outperforms the execution time for the other configurations (higher index than 3.9). The execution time using 3 and 12 tiles is not specified in [4]; however, we expect a higher execution time than the other tile configuration since they use less GPU threads.

5 Conclusions and future work

In this paper an algorithm that concurrently executes the inter prediction performed in the AVC JM 17.2 reference software encoder is presented. Our approach is based on an efficient parallel implementation of the algorithm and its data structures involved in the ME and MC. Exploiting current GPU computational capability provides us with another way to accelerate inter prediction in traditional video codecs, with a view to developing real-time video encoders.

Table 12 Two-hour sequence encoding time for 720p sequences

Feature	JM 17.2 Full Search	JM 17.2 Fast Full Search	JM 17.2 UMHexagonS Search	Proposal
fps	0.12	0.10	1.10	2.10
Time (Hours)	833.33	1000	90.91	47.62

Table 13 Two-hour sequence encoding time for 1080p sequences

Feature	JM 17.2 Full Search	JM 17.2 Fast Full Search	JM 17.2 UMHexagonS Search	Proposal
fps	0.04	0.04	0.43	0.86
Time (Hours)	2500	2500	232.56	116.28

Table 14 RD comparison with Cheung et al. results [4]

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

Table 15 Execution time comparison with Cheung et al. results [4]

Number of Tiles	Sequence				Average GPU time (ms)	Index
	Crew GPU Time (ms)	City GPU Time (ms)	Harbor GPU Time (ms)	Night GPU Time (ms)		
3,600	835.05	927.32	1,248.95	1,688.50	1,174.95	3.85
900	959.16	1,005.55	1,341.45	1,975.95	1,320.53	4.33
225	2,169.25	2,108.71	2,763.79	4,175.44	2,804.30	9.19
90	4,373.63	4,165.28	5,318.38	6,920.73	5,194.51	17.02
12	Unknown					
3						
Our algorithm	305.09	306.16	304.96	304.71	305.23	

Real-time encoding is not achieved in this work. However, future algorithm adaptations for Tesla Data Center Solutions composed of more than one GPU and powerful CPUs could be the starting point to achieve it. Moreover, in an AVC encoder there are other modules that could be good candidates to be adapted to the programming model of modern GPUs, e.g. Intra Prediction.

The algorithm's good performance is mainly due to an optimized usage of memory transfers, a significant reduction in shared memory bank conflicts and an adequate definition and management of data structures.

The results show a considerable time reduction of over 98.8 % compared with full search (speedup of over 86) and of over 98.9 % compared with fast full search (speedup of over 98) for all kinds of video content, with negligible RD drop. We should also mention that our algorithm outperforms the encoding efficiency of UMHexagonS search while obtaining a time reduction of over 82 % (speedup of over 5) for all kinds of video content.

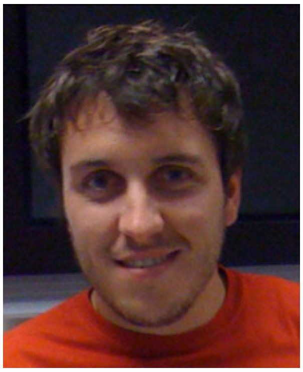
Acknowledgments This work was supported by the Spanish MEC and MICINN, as well as European Commission FEDER funds, under Grants CSD2006-00046 and TIN2009-14475-C04. It was also partly supported by The Council of Science and Technology of Castilla-La Mancha under Grants PEII09-0037-2328, PII2I09-0045-9916, and PCC08-0078-9856.

References

1. G. Bjøntegaard, "Calculation of Average PSNR Differences between RD-Curves", presented at the 13th VCEG-M33 Meeting, Austin (Texas), USA, April 2001.
2. Chen W-N, Hang H-M (2008) H.264/AVC motion estimation implementation on Compute Unified Device Architecture (CUDA). In Proc. of IEEE International Conference on Multimedia and Expo, ICME, pp. 697–700, Hannover, Germany, June
3. Chen YK, Tian X, Ge S, Girkar M (2004) Towards efficient multi-level threading of H.264 encoder on Intel hyper-threading architectures. In Proc. of the 18th International Parallel and Distributed Processing Symposium, pp. 63–72, Santa Fe(New Mexico), USA, April
4. Cheung N-M, Fan X, Au OC, Kung M-C (2010) Video coding on multicore graphics processors. IEEE Signal Process Mag 27(2):79–89
5. Feng W-C, Manocha D (2007) High-performance computing using accelerators. Parallel Computing 33 (10–11):645–647
6. Ho C-W, Au OC, Gary Chan S-H, Yip S-K, Wong H-M (2006) Motion Estimation for H.264/AVC using Programmable Graphics Hardware. In Proc. of IEEE International Conference on Multimedia and Expo, ICME, pp. 2049–2052, Toronto, Canada, July
7. ISO/IEC 14486–2 PDAM1: (1999) Infomation technology- generic coding of audio-visual objects- part 2: visual
8. ISO/IEC International Standard 14496–10: (2003) Information technology – coding of audio - visual objects – part 10: advanced video coding
9. Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Reference Software to Committee Draft. JVT-F100 JM17.2, 2010. Available on-line at http://iphome.hhi.de/suehring/tml/.
10. JVT Test Model Ad Hoc Group (2003) Evaluation sheet for motion estimation, draft version 4, February
11. Kelly F, Kokaram A (2004) Fast Image Interpolation for Motion Estimation using Graphics Hardware. In Proc. of IS&T/SPIE Electronic Imaging - Real-Time Imaging VIII, vol. 5297, pp. 184–194, San Jose (California), USA, January 2004
12. Kung M-C, Au OC, Wong PHW, Liu CH (2008) Block based parallel motion estimation using programmable graphics hardware. In Proc. of International Conference on Audio, Language and Image Processing, ICALIP, pp. 599–603, Shanghai, China, July
13. Lee C-Y, Lin Y-C, Wu C-L, Chang C-H, Tsao Y-M, Chien S-Y (2007) Multi-pass and frame parallel algorithms of motion estimation in H.264/AVC for Generic GPU. in Proc. of IEEE International Conference on Multimedia and Expo, ICME, pp. 1603–1606, Beijing, China, July
14. Momcilovic S, Sousa L (2009) Development and evaluation of scalable video motion estimators on GPU. in Proc. of IEEE Workshop on Signal Processing Systems, SiPS , pp. 291–296, October

15. NVIDA (2010) NVIDIA CUDA Compute Unified Device Architecture-Programming Guide, Version 3.2, August
16. Richardson IEG (2003) Video codec design. John Willey & Sons LTD, 2 edition
17. Schwalb M, Ewerth R, Freisleben B (2009) Fast motion estimation on graphics hardware for H.264 video encoding. in IEEE Transaction on Multimedia, vol. II, no. 1, pp. 1–10, January
18. Sullivan G, Bjøntegaard G (2001) Recommended simulation common conditions for H.26L coding efficiency experiments on low-resolution progressive-scan source material. ITU-T VCEG, Doc. VCEG-N81. September

Rafael Rodríguez-Sánchez received his M.Sc. degree in Computer Science in 2010 and his B.Sc. degree from the University of Castilla-La Mancha in 2008. He is completing his Ph.D in the Instituto de Investigación en Informática (I3A) in Albacete, Spain. His research interests include video coding and parallel processing. He has 6 publications as author in these areas in refereed conference proceedings.



José Luis Martínez (M'07) received his M.Sc. degree in Computer Science in 2007 and the Ph.D. degree from the University of Castilla-La Mancha in 2009. In 2005, he joined the Department of Computer Engineering at the University of Castilla-La Mancha. His research interests include Distributed Video Coding (DVC), video transcoding, quality metrics, Scalable Video Coding (SVC), parallel programming, GPUs among others. He has also been a visiting researcher at the Florida Atlantic University, (USA) and CCSR, at the University of Surrey (UK). He has over 25 publications in these areas in international refereed journals and conference proceedings. He is a member of the IEEE.

Gerardo Fernández-Escribano (M’05) received his M.Sc. degree in Computer Engineering, in 2003, and the Ph.D. degree from the University of Castilla-La Mancha, Spain, in 2007. In 2004, he joined the Department of Computer Engineering at the University of Castilla-La Mancha. His research interests include multimedia standards, video transcoding, video compression, video transmission, and machine learning mechanisms. He has also been a visiting researcher at the Florida Atlantic University, Boca Raton (USA), and at the Friedrich Alexander Universität, Erlangen-Nuremberg (Germany).

Jose L. Sánchez received the PhD degree from the Technical University of Valencia, Spain, in 1998. Since November 1986 he is 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, quality of service in high-speed networks, interconnection networks, heterogeneous computing, parallel algorithms and simulation.

José M. Claver (M'00) received the M.Sc. degree in physics from the University of Valencia, Burjassot, Spain, 1984, and the Ph.D. degree in computer science from the Technical University of Valencia, Valencia, Spain, in 1998. From 1985 to 1990, he was with the Electronics and Computer Architecture Department, University of Castilla-La Mancha, Albacete, Spain. From 1991 to 2007, he was with the Department of Computer Science, Jaume I University, Castellón, Spain. Since 2007, he has been an Associate Professor at the Computing department of the University of Valencia. He has taught undergraduate courses on computer architecture and embedded systems and graduate courses on high-speed networks, advanced computer architecture, and parallel computing. His research interests include computer architecture, parallel computing, high-speed QoS networks, network protocols, embedded systems, and reconfigurable computing. He is the author or coauthor of more than 40 research publications on these subjects. He is an IEEE member.