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

Rafael Rodríguez-Sánchez[1], José Luis Martínez[2], Gerardo Fernández-Escribano[1], José Luis Sánchez[1], and José Manuel Claver[3]

[1] Instituto de Investigación en Informática de Albacete,
Universidad de Castilla-La Mancha, Avenida de España s/n, 02071, Albacete, Spain
{rrsanchez,gerardo,jsanchez}@dsi.uclm.es
[2] Architecture and Technology of Computing System Group,
Complutense University, Madrid, Spain
joseluis.martinez@fdi.ucm.es
[3]Departamento de Informática. Universidad de Valencia,
Avenida de Vicente Andrés Estelles, s/n, 46100 Burjassot, Valencia, Spain
jose.claver@uv.es

Abstract. H.264/AVC is the most recent predictive video compression standard to outperform other existing video coding standards by means of higher computational complexity. In recent years, heterogeneous computing has emerged as a cost-efficient solution for high-performance computing. In the literature, several algorithms have been proposed to accelerate video compression, but so far there have not been many solutions that deal with video codecs using heterogeneous systems. This paper proposes an algorithm to perform H.264/AVC inter prediction. The proposed algorithm performs the motion estimation, both with full-pixel and sub-pixel accuracy, using CUDA to assist the CPU, obtaining remarkable time reductions while maintaining rate-distortion performance.

Keywords: H.264/AVC, Heterogeneous computing, Motion Estimation, GPU.

1 Introduction

Most of the applications which have recently appeared in the multimedia community, such as digital TV, streaming, video conferencing or DVDs, require a video coding algorithm to meet their requirements and operate. The most recent video coding standard is H.264/AVC [1], which is able to outperform the video codecs of previous coding standards [2]. The compression gains are mainly related to the variable and smaller block size motion compensation, improved entropy coding, motion estimation with multiple reference frames, and smaller block transforms, among others. However, these new/improved video coding tools increase both the encoder and decoder complexity substantially.

Fortunately, heterogeneous computing has emerged as a real solution for high-performance computing [3]. There are many examples of this kind of systems, but possibly the systems composed of one or more Graphics Processing Units (GPUs) and one or more multi-core Central Processing Units (CPUs) are the most widely used. GPUs are small devices with hundred of similar processing cores organized to achieve higher performance. GPUs are highly parallel and are normally used as a coprocessor to assist the CPU for computing massive data. GPUs have a parallel architecture that focuses on executing many concurrent threads slowly, rather than executing a single thread very quickly. In order to assist programmers, the main GPU manufacturers provide them with different tools. For example, Nvidia® proposes a powerful GPU architecture called Compute Unified Device Architecture (CUDA) [4]. CUDA is basically a Single Instruction Multiple Data (SIMD) computing device.

Therefore, it is mandatory to explore new and efficient implementations of H.264/AVC video coding systems on different computing platforms in order to support these applications. At this point, this paper proposes an algorithm to perform the inter prediction carried out in H.264/AVC using CUDA to assist the CPU. The present approach performs the Motion Estimation (ME) with both full-pixel and sub-pixel accuracy over the GPU. It should be pointed out that the ME algorithm is one of the most computationally expensive tasks in an H.264/AVC encoder; it performs the same operations (Sum of Absolute Differences) over a large amount of data (over the search area). Therefore, ME fits well in the SIMD programming model. On the other hand, the ME algorithms implemented in the H.264/AVC reference software are sequential, where each MacroBlock (MB) is encoded based on its neighboring MBs. One of the major issues of our proposed algorithm is how to remove or mitigate these dependencies between MBs in order to minimize the Rate-Distortion (RD) penalties. Performance evaluation is carried out for High Definition (HD) video sequences. The results show a remarkably time reduction of up to 99% with a negligible RD penalty. Moreover, the proposed algorithm outperforms the fastest ME algorithms included in the H.264/AVC reference software as well as some of the approaches available in the literature in terms of coding efficiency and time savings.

The rest of the paper is organized as follows: Section 2 contains a brief overview of H.264/AVC and GPU programming; in Section 3 some related proposals are shown; Section 4 shows details about the approach presented in this paper; Section 5 describes the performance evaluation and, finally, conclusions are given in Section 6.

2 Technical Background

In the H.264/AVC standard [1] inter prediction is carried out by means of the process of variable block size ME, which is able to eliminate the temporal redundancy between two or more adjacent frames. This approach supports motion compensation block sizes ranging between 16x16, 16x8, 8x16 and 8x8, where each of the sub-divided regions is an MB partition. If the 8x8 mode is chosen, each of the four 8x8 block partitions within the MB may be further split in 4 ways: 8x8, 8x4, 4x8 or 4x4, which are known as sub-MB partitions. The ME is carried out for each of these partitions. Furthermore, Motion Vectors (MVs) from spatially adjacent blocks and from other MB partitions are used to initialize the search area for the current partition. These are known as Motion Vector predictors (MVp). In addition, to compute the rate term R in the Lagrangian cost the MVs of the neighboring MBs are required [5].

In terms of Multi-Core graphics processors, GPUs are small accelerator devices with hundreds of cores which are organized in several SIMD blocks. GPUs are characterized by a high level of parallelism and are usually used as a coprocessor to assist the CPU in computing massive data. For instance, the architecture of the Nvidia® GPUs consists of a set of SIMD multiprocessors called Stream Multiprocessors (SM). Each SM has up to 48 processing elements called cores and a set of resources shared by all cores, such as 32-bit registers, local shared / texture memory or caches. More detail about the Nvidia GPU architecture can be found in [4].

3 Previous Work

Most of the proposals available in the literature for accelerating the H.264/AVC encoding algorithm are sequential-based approaches, but so far there have not been many solutions which make use of Many-Core graphics hardware to accelerate this highly complex algorithm. At this point, the main objective of this paper is to combine powerful Multi-Core architectures to accelerate traditional video coding algorithms, such as H.264/AVC. In 2007, Lee et al. [6] presented a multi-pass and frame parallel algorithm to accelerate H.264/AVC ME using a GPU. They unroll and rearrange the multiple nested loops by using the multi-pass method. The multiple reference frames method is implemented at frame parallel level by the use of SIMD vector operations of the GPU. In 2008, Ryoo et al. in [7] and Chen and Hang [8] presented some optimization principles of a multithreaded GPU using CUDA. In [8] the algorithm is based on an efficient block-level parallel algorithm for the variable block size motion estimation in H.264/AVC. They decompose the H.264/AVC ME algorithm into 5 steps so that they can achieve highly parallel computation. The major failing of all these approaches is that they do not analyze the RD performance, they only show timing results; although the speedup and time reduction are acceptable, they are only valid if they keep the RD as close as possible to the sequential approach.

More recently, in 2010, Cheung et al. proposed a GPU implementation of the simplified Unsymmetrical Multi-Hexagon search (smpUMHexagonS) [9] ME algorithm, which is a fast ME technique implemented in the H.264/AVC reference software. 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 (0.4dB) depending on the sequence and the tile length.

Many-Core architectures have been also used for accelerating other modules of the H.264/AVC encoding algorithm, such as the Intra Prediction [10]. Based on a dependency analysis of intra-mode decision. they propose to encode the video blocks following the greedy order, leading to highly parallel RD cost computations. They obtain an speed of up to 80 with negligible RD penalty.

4 Proposed GPU-Based Motion Estimation Algorithm

This section describes the algorithm for implementing the inter prediction performed by the H.264/AVC encoding algorithm via a system composed of a CPU with the support of a GPU. The ME as part of the inter prediction process has been implemented in the JM v17.2 reference software [11], but this algorithm can be easily adapted to other H.264 implementation such as x264 [12]. The present approach is based on the *Full Search* (FS) ME algorithm implemented in the JM reference software. At this point, this paper proposes a modified GPU-based FS ME algorithm with quarter-pixel accuracy.

ME is tackled in two steps, full-pixel and sub-pixel accuracy ME; each one is performed following a highly-parallel procedure over the GPU. Firstly, the image to be processed (with full-pixel accuracy) is moved from CPU to GPU and it performs the full-pixel ME. Then, the GPU is able to generate the sub-pixel image from the reference image. Finally, the GPU performs the sub-pixel ME. In other words, once the frame is moved to the GPU, all the computations relating to the ME are carried out over the GPU. In this way, we avoid memory transfers between the CPU and GPU, which is the bottleneck in this kind of systems. The proposed GPU-based ME algorithm is performed concurrently for the complete image at the beginning of coding each P frame, where the inter prediction is applied. Figure 1 shows a simplified activity diagram of our parallel ME proposal.

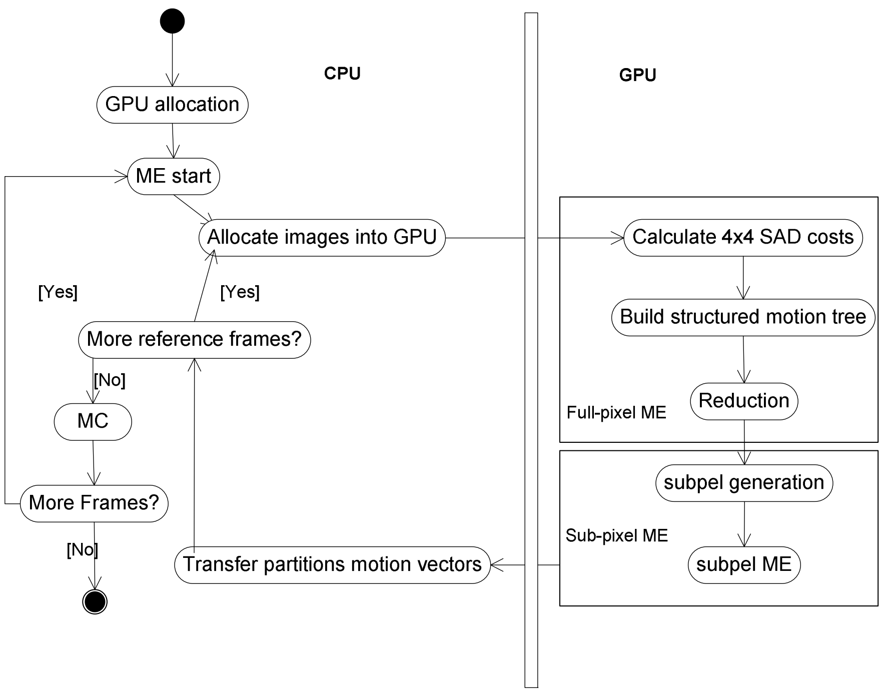

Fig. 1. Activity Diagram of Proposal

4.1 Full-Pixel Motion Estimation

The proposed Full-pixel ME process is divided into three steps. The goal of the first step is to obtain the Sum Absolute Differences (SAD) calculation between the current

MB (split into sixteen 4x4 partitions) and all MB positions in the reference frame inside the search area. Then, the second step, using the previous 4x4 block SAD calculations, is able to obtain the SAD costs for all other MB partitions. Finally, the last step reduces the SAD costs to one SAD cost for each one of the 41 MB partitions of each MB. This three-step algorithm is implemented using two GPU kernels.

In the first kernel, all threads from a thread block cooperate to copy its assigned MB and corresponding search area from texture memory to multiprocessor local shared memory. Shared memory is defined as an integer and it allows contiguous multiprocessor threads to read from contiguous memory banks without access conflicts in the memory banks. The SAD calculations are carried out in 4x4 blocks, therefore each MB is divided into sixteen 4x4 blocks for each search area position. These SAD costs are stored in registers to build the structured motion tree (*4x4 SAD sub-matrix* in Figure 2). The complete search area is computed by rows, one or more rows corresponding to a thread block, so contiguous search area positions for a certain MB are computed by the same thread block, with normally 256 positions for each thread block.

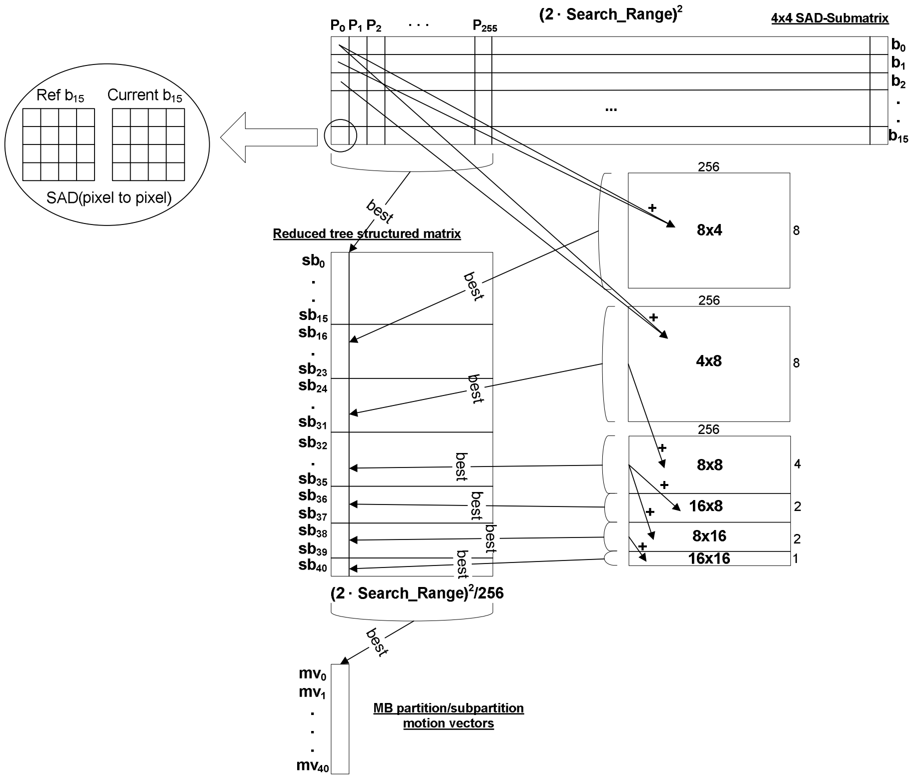

Fig. 2. Proposed Full-pixel ME Algorithm

In the same GPU kernel used to obtain the 4x4 SAD costs, the structured motion tree is obtained. Using the information previously stored in registers (4x4 SAD costs) our algorithm is able to obtain the SAD costs for higher partitions. As depicted in Figure 2, by adding two 4x4 SAD costs it is able to obtain the 4x8 and 8x4 SAD costs, by adding two 4x8s it is able to obtain the 8x8 SAD costs and so on. Intermediate results are stored in multiprocessor shared memory for faster memory accesses.

Finally, this kernel also carries out a first reduction due to the large amount of data generated. The reduction starts with 256 SAD costs per MB partition and finishes with 1 SAD cost per MB partition (*Reduced tree-structured matrix* in Figure 2). For this reduction procedure, a binary reduction has been implemented. Note that this kernel performs the reduction procedure over the 256 positions configured in a thread block; the final reduction is performed by an independent kernel. This last kernel obtains the best SAD cost for each one of the MB partitions in each MB, using the same binary reduction procedure as the previous kernel.

Our GPU-based algorithm is executed concurrently for a complete frame, but each MB coding depends on their neighbors in two ways: 1) to compute the Lagrangian cost and, 2) to locate the search area (MVp). These dependencies mean that the optimal MV may not be found, resulting in a bitrate increase and in a PSNR drop.

Therefore, the proposed algorithm also tries to mitigate the effect of MVp, which is one the biggest challenges of performing the ME process in parallel. The idea for solving these impairments consists of reusing the MV of the previous frame to adjust the MB search area. The MVp does not have a big impact on low resolution and/or low motion sequences, but the lack of MV predictors for higher resolutions (such as HD) and/or high motion sequences may result in a big impact on RD performance. After a large set of experiments, we conclude that the best way to estimate the motion is to use the MV from the higher partition (16x16) for the MB located in the same position in the reference frame, but with the constraint that the MVp cannot be higher than the search range to ensure that the MV (0,0) is inside the search area.

Furthermore, to compute the Lagrangian cost, the MVp is required. The Lagrangian cost is defined as $SAD_{cost} + \lambda * vector_{bits}$, where $vector_{bits}$ is the number of bits required to encode the $MV - MV_P$. Nevertheless, the MVp is required to obtain the final cost for all positions inside the search area, which is affected by the dependencies between neighboring MBs.

4.2 Sub-pixel Accuracy Motion Estimation

In order to further improve compression, the H.264 /AVC standard assumes that the best match can be found at 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. H.264/AVC reference software supports quarter-pixel accuracy, which means that the image sizes are multiplied by four on 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. A GPU thread per 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, both of 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- 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 4x4 blocks and each one takes as its starting point the appropriate MV, i.e., all 4x4 blocks will take the same MV to perform the 16x16 partition and the final cost will be obtained using atomic GPU operations. On the other hand, all 4x4 blocks will take different MVs to perform the 4x4 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 main aspects to take into account. 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 4x4 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 H.264/AVC JM 17.2 [11] reference software used.

5 Performance Evaluation

In order to show the performance of the proposed algorithm, it was implemented in the H.264/AVC JM v17.2 reference software encoder. The parameters used in the H.264/AVC encoder configuration file were those included in the baseline profile of the mentioned reference software. However, 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, but higher number of reference frames can be used; RD-Optimization was disabled for the same reason as the NumberReferenceFrames parameter; the GOP pattern was set to one I frame followed by eleven P frames (I11P); the tests were carried out with popular sequences in 720p format (1280 x 720) and 1080p format (1920 x 1080); the frame rate parameter was set to 50 because sequences were sampled at 50Hz; the parameter FramesToBeEncoded was adjusted according to the sequence, in order to encode the full sequence; the Quantization Parameter (QP) called QPISlice and QPPSlice was varied among 28, 32, 36 and 40 according to [13].

The performance evaluation of our proposal for H.264/AVC based on the JM v17.2 encoder was carried out on a system composed of an Intel® Core™ i7 @930 running at 2.80 GHz, with 6GB DDR3 memory and the GPU Nvidia GTX480. The operating system was Ubuntu 10.4 with the Nvidia GPU driver 260.19 and CUDA SDK 3.2 was used.

5.1 Metrics

In order to evaluate the time saved by the proposed algorithm with respect to the reference H.264/AVC encoder, two metrics were used: Time Reduction (TR), which is based on Equation 1, and Speedup, which is based on Equation 2.

$$TR\,(\%) = \frac{T_{JM} - T_{FI}}{T_{FI}}\, x\, 100 \tag{1}$$

$$Speedup = \frac{T_{JM}}{T_{FI}} \tag{2}$$

where T_{JM} denotes the coding time used by the reference software, and T_{FI} is the time taken by the algorithm proposed in this paper. The times measured by T_{JM} and T_{FI} refer to the time employed to carry out the ME. T_{FI} also includes all the computational costs for the operations needed in order to prepare the information required by our proposal.

5.2 Results

Table 1 shows the RD performance and time reduction of our proposed GPU-based ME algorithm for 720p sequences against three of the most well-known ME algorithms implemented by the reference H.264/AVC encoder (Full search (FS), Unsymmetrical Multi-Hexagon search (UMHexagonS) and simplified Unsymmetrical Multi-Hexagon search (smpUMHexagonS) [14])). The results show that the RD performance obtained by our proposed ME algorithm is very similar to the sequential-based implementations. Our GPU-based ME algorithm compared with FS ME and smpUMHexagonS ME have a PSNR drop of up to 0.1 dB for a given bitrate, and a bitrate increase of up to 3.52% for a given PSNR. On the other hand, our algorithm has a PSNR increase of up to 0.199 dB for a given bitrate and a bitrate drop of up to 7.07% for a given bitrate, compared with UMHexagonS ME.

Table 1. RD Performance and TR of the proposed GPU-Based Algorithm. 720p sequences.

Sequence	Full Search			UMHexagonS			smpUMHexagonS		
	ME TR	ΔPSNR (dB)	ΔBitrate (%)	ME TR	ΔPSNR (dB)	ΔBitrate (%)	ME TR	ΔPSNR (dB)	ΔBitrate (%)
Dolphins	98.82	-0.072	2.59	83.01	0.199	-7.07	68.60	-0.039	1.35
Mobcal	99.06	-0.037	1.15	81.64	0.031	-1.12	74.46	-0.073	2.35
Parkrun	99.28	-0.043	1.40	86.31	-0.018	0.57	80.72	-0.049	1.59
Shields	98.94	-0.043	1.38	83.50	-0.038	1.00	73.95	-0.100	3.21
Stockholm	98.96	-0.040	1.35	82.67	-0.021	0.45	74.74	-0.097	3.52
Average	***99.01***	***-0.047***	***1.57***	***83.43***	***0.031***	***-1.23***	***74.50***	***-0.072***	***2.40***

Table 2 shows the RD performance and time reduction of our proposed algorithm for 1080p sequences. The RD performance conclusions are similar to those obtained for 720p format. Our GPU-based algorithm obtains slightly worse results than the FS ME algorithm and the smpUMHexagonS ME algorithm (the bitrate increases and the

PSNR drops) and it obtains slightly better results than UMHexagonS ME (the bitrate drops and the PSNR increases). However, for both resolutions our proposal obtains considerable time reductions. The average ME time reduction comparing with FS ME is better than 99% (Speedup over 100), the average ME time reduction comparing with UMHexagonS ME is better than 81% (Speedup over 5) and finally, the average ME time reduction comparing with smpUMHexagonS ME is better than 74.5% (Speedup close to 4). At this point, the negligible RD penalty is an acceptable solution because of the very high time reductions achieved.

Table 2. RD Performance and TR of the proposed GPU-Based Algorithm. 1080p sequences.

Sequence	Full Search			UMHexagonS			smpUMHexagonS		
	ME TR	ΔPSNR (dB)	ΔBitrate (%)	ME TR	ΔPSNR (dB)	ΔBitrate (%)	ME TR	ΔPSNR (dB)	ΔBitrate (%)
Crowd	99.15	-0.118	3.86	83.14	0.035	-1.25	78.57	-0.103	3.32
Ducks	99.31	-0.037	1.20	84.19	-0.002	-0.12	81.18	-0.036	1.05
IntoTree	99.13	-0.074	3.66	84.19	0.026	-1.18	77.56	-0.113	5.00
OldTown	98.92	-0.048	2.31	80.15	-0.017	0.36	71.17	-0.102	3.78
ParkJoy	99.25	-0.077	2.33	85.10	0.017	-0.61	79.38	-0.087	2.61
Average	***99.15***	***-0.071***	***2.67***	***81.89***	***0.012***	***-0.56***	***77.57***	***-0.088***	***3.15***

Figure 3 shows the RD graphic (PSNR against bitrate) results for the reference algorithms and the proposed approach, using different 1080p sequences. In general, the PSNR against bitrate curves are quite similar to those achieved by the reference algorithm while our proposal is always much faster than the reference implementations. Due to space limitations only 1080p sequences are shown. Similar RD results are obtained for 720p sequences.

5.3 Comparison with Other Known Results

In this section, a comparative performance evaluation in terms of the RD performance and execution time is presented. We compare the results of the proposed algorithm with those shown in one of the papers available in the literature with the most promising results. In [9], the authors proposed 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 3 shows the RD results for their algorithm as well as our RD results using the same encoding conditions. We have employed the same 720p sequences sampled at 60Hz, 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. In 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.

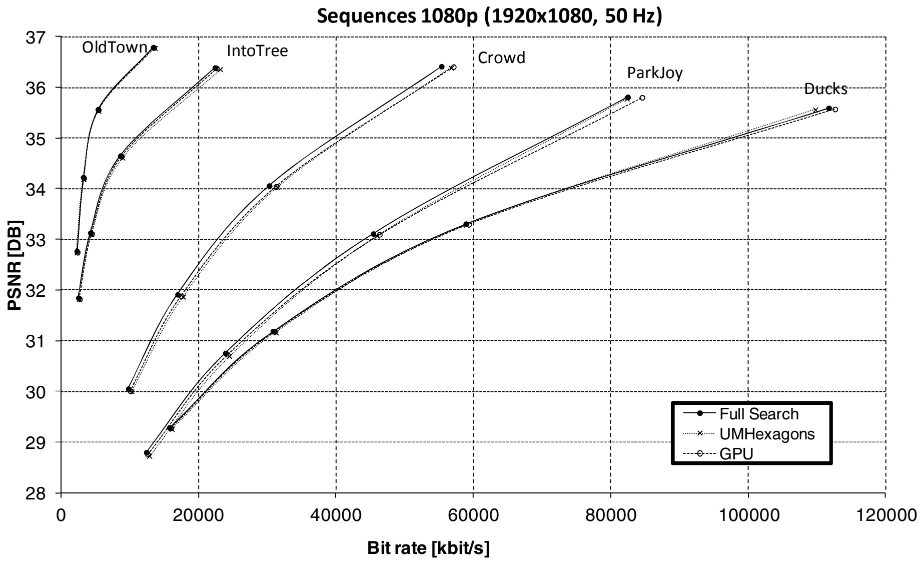

Fig. 3. RD results comparing the performance of the proposal and the H.264 reference. 1080p Sequences.

Table 3. RD comparison with *Cheung et al.* results[9]

	Sequence							
	Crew		City		Harbor		Night	
Number of Tiles	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 4 shows the execution time for the experiments carried out to fill the previous table. Table 4 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 [9] 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 4, which shows the ratio between the average execution time obtained by their implementation for a certain encoder configuration and the average execution time by 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 [9]; however we expect a higher execution time than the other tile configuration since they use less GPU threads.

Table 4. Execution time comparison with *Cheung et al.* results[9]

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

6 Conclusions

This paper proposes an algorithm to perform the inter prediction carried out in H.264/AVC using a system composed of a CPU with the support of a GPU. The proposed algorithm performs the ME over a GPU by means of an efficient distribution of complexity and management of GPU resources. The algorithm includes the ME with full-pixel and sub-pixel accuracy, as well as sub-pixel interpolation. Furthermore, the proposed approach is further adapted to avoid coding dependencies between MBs, which is one of the major issues when the ME is carried out in parallel. The results show that the proposed algorithm achieves almost the same coding efficiency but outperforms all the ME algorithms available in the JM reference software in terms of time. The performance is also compared with the state-of-the-art approach available in the literate, achieving a faster execution time together with better coding efficiency.

Acknowledgements. 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 and PII2I09-0045-9916.

We would like to thank *N.M. Cheung*, *O. C. Au* and the rest of the authors of the paper [9] for their invaluable support in improving this manuscript.

References

1. ISO/IEC International Standard 14496-10:2005, Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding
2. Wiegand, T., Sullivan, G.J., Bjontegaard, G., Luthra, A.: Overview of the H.264/AVC Video Coding Standard. IEEE Transaction on Circuits and System for Video Technology 13(7), 560–576 (2003)
3. Feng, W.-C., Manocha, D.: High-performance computing using accelerators. Parallel Computing 33(10-11), 645–647 (2007)
4. Nvidia, Nvidia CUDA Compute Unified Device Architecture-Programming Guide, Version 3.2 (August 2010)
5. Wiegand, T., Girod, B.: Lagrange multiplier selection in hybrid video coder control. In: Proc. IEEE International Conference on Image Processing, ICIP (2001)
6. 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 IEEE International Conference on Multimedia and Expo 2007 (ICME 2007), Beijing (China), pp. 1603–1606 (July 2007)
7. 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, Salt Lake City (USA), pp. 73–82 (February 2008)
8. 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 2008 (ICME 2008), Hannover (Germany), pp. 679–700 (June 2008)
9. Cheung, N.-M., Fan, X., Au, O.C., Kung, M.-C.: Video Coding on Multi-Core Graphics Processors. IEEE Signal Processing Magazine 27(2), 79–89 (2010)
10. Cheung, N.-M., Au, O.C., Kung, M.-C., Wong, P.H.W., Liu, C.H.: Highly Parallel Rate-Distortion Optimized Intra Mode Decision On Multi-Core Graphics Processors. IEEE Transactions on Circuit and System for Video Technology 19(11), 1692–1703 (2009)
11. Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Reference Software to Committee Draft. JVT-F100 JM17.2 (2010)
12. X.264 reference software, `http://www.videolan.org/developers/x264.html`
13. Sullivan, G., Bjøntegaard, G.: Recommended Simulation Common Conditions for H.26L Coding Efficiency Experiments on Low-Resolution Progressive-Scan Source Material, ITU-T VCEG, Doc. VCEG-N81 (2001)
14. Richardson, I.E.G.: Video codec design, 2nd edn. John Willey & Sons LTD. (2003)