COOPERATIVE CPU+GPU DEBLOCKING FILTER PARALLELIZATION FOR HIGH PERFORMANCE HEVC VIDEO CODECS

Diego F. de Souza, Nuno Roma, Leonel Sousa

IST / INESC-ID
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
difs@sips.inesc-id.pt, nuno.roma@inesc-id.pt, las@inesc-id.pt

ABSTRACT

Heterogeneous platforms integrating several CPU cores and GPU accelerators have established in several application domains, from desktop, server and mobile. To take full advantage of such platforms, video encoders/decoders have to exploit a broader design space, by cooperatively executing in all the available CPU and GPU cores. To attain such objective, three novel contributions that aim the exploitation of the maximum parallelism level in an HEVC deblocking filter are presented: *i*) a highly optimized CPU parallel implementation, which outperforms the current state of the art; *ii*) the first known GPU implementation of the HEVC deblocking filter; and *iii*) an hybrid and load-balanced CPU+GPU implementation, where all the available resources cooperatively execute, in order to maximize the attained performance. The obtained experimental results demonstrated the ability to achieve processing times as low as 0.8 ms and 0.5 ms to filter 1080p I-type and B-type frames, respectively, corresponding to speedup factors as high as 17 and 9.

Index Terms— Video coding, HEVC, deblocking filter, Graphics Processing Unit, parallel processing

1. INTRODUCTION

The recent proposed *High Efficient Video Coding* (HEVC) standard [1] is nowadays the state of the art on video compression [2]. When compared with previous standards, it has been demonstrated that HEVC encoders can achieve equivalent subjective visual quality as H.264/MPEG-4 AVC encoders, when using approximately 50% less bit rate [3]. However, such coding efficiency performance comes at cost of increasing the computational complexity of the video encoder and decoder [4]. According to the conducted decoder profiling [4], it has been shown that the HEVC deblocking filter is responsible for 14% of the time consumption in the random access configuration, for Full HD video sequences.

This work was supported by national funds through FCT – Fundação para a Ciência e a Tecnologia, under projects SFRH/BD/76285/2011, PTDC/EEI-ELC/3152/2012 and PEst-OE/EEI/LA0021/2013.

On the other hand, the deblocking filter is responsible for an average bit rate reduction of 1.3%–3.3%, although, more than 6% bit rate reduction can be achieved in certain sequences [5]. When compared to the H.264/MPEG-4 AVC deblocking filter, the complexity of this module has been significantly reduced in HEVC. The main factors are: it is performed in a grid of 8×8 samples, which are kept limited in the prediction and transform unit boundaries; there is no data dependencies between adjacent edges with same direction; the chroma blocks are only filtered when one of the adjacent blocks is intra predicted; all the vertical edges of the frame are performed independently and before the horizontal edges.

In this paper, a parallel algorithm is proposed for the HEVC deblocking filter, by exploiting independent regions of the frame and by reducing the overall memory accesses. It is also presented the first HEVC deblocking filter implementation using Graphics Processing Unit (GPU), with the NVIDIA Compute Unified Device Architecture (CUDA) [6] platform. The proposed algorithm is evaluated in three different scenarios: using only CPU cores, only the GPU and an load balance implementation, exploiting both the CPU cores and GPU.

The remainder of this paper is organized as follows. Section 2 describes the last proposed algorithms for the HEVC deblocking filter. In Section 3, the HEVC deblocking filter is briefly introduced. The proposed algorithm and consequent implementations are presented in Section 4. The experimental results and the addressed conclusions are shown in Sections 5 and 6, respectively.

2. BACKGROUND WORK AND STATE OF THE ART

In the past decade, GPUs have evolved from a fixed-function graphics pipeline to a programmable and general purpose parallel (GPP) processor, with computing power exceeding that of multi-core CPUs [7]. While the CPU is designed and optimized for sequential code performance, the GPU is specialized for compute-intensive highly parallel computation [8], like 3D rendering. In this way, the GPU has been designed to devote more transistors to data processing, rather than data caching and flow control.

978-1-4799-2893-4/14/$31.00 ©2014 IEEE

Along the past few years, several video encoding modules have been implemented in GPU devices. As an example, Wang et al. designed a HEVC motion estimation on a CPU + GPU platform [9]. In what concerns the deblocking filter module, all known implementations on GPU devices only consider the H.264/MPEG-4 AVC standard, and most of them achieve the claimed performance levels at the cost of alleviating some data dependencies in order to increase the level of parallelism [10] and [11]. To the best of the authors' knowledge, the presented contributions represent the first GPU implementation of the HEVC deblocking filter, in an heterogeneous and load-balanced CPU+GPU hybrid plataform.

In what concerns the GPP platforms, Yan et al., in [12], proposed a HEVC decoder by exploiting Single Instruction, Multiple Data (SIMD) parallelism in each decoder module, in order to increase the achieved performance. For the HEVC deblocking filter, Streaming SIMD Extensions 2 (SSE2) arithmetic functions of x86 processors are applied to achieve an speedup to 5 for 1080p video sequences. In [13], Kotra et al. proposed three implementations for the HEVC deblocking filter. The first divides the frame horizontally (for vertical filtering) and vertically (for horizontal filtering). The other implementations combine the vertical and horizontal filtering in a single pass, while the frame is vertically divided among the CPU cores. When executed in a 6-core CPU platform, the best implementation of the proposed algorithms can obtain an average speedup of 5 for 1080p video sequences.

Hardware HEVC deblocking filter implementations using Field-Programmable Gate Arrays (FPGA) were also proposed in [14] and [15]. For a 1080p video resolution, the proposed architectures can filter one frame in 1.1 ms and 33.9 ms, respectively. However, such hardware implementations represent different compromises, in terms of energy efficiency, preventing a fair comparison with high-performance computing platforms, like GPUs.

3. HEVC DEBLOCKING FILTER

As defined in the HEVC standard, each frame is divided in Coding Three Units (CTU) of 64×64, 32×32 or 16×16 samples. Each CTU can be further divided in smaller blocks, called Coding Units (CU), by using a quadtree structure. Each CU is divided in the Prediction Unit (PU) and the Transform Unit (TU), which can be further split in smaller blocks.

The HEVC deblocking filter is applied to the boundaries of the PU and TU, which rely on a 8×8 samples grid. For each boundary, a Boundary filtering Strength (BS) is evaluated, according to eight conditions of the neighboring blocks. The range of the BS value is defined between 0 and 2, where 0 means that no deblocking filter will be applied. The chroma blocks will be only filtered if the BS value is equal to 2 [5].

For luma boundaries, with BS greater than 0, additional conditions are checked to determine whether the deblocking filter should be applied or not. Each condition is verified from a set of 8×4 or 4×8 pixels, for the vertical and horizontal edges, respectively, as it is marked in Figure 1. The outer pixel lines or columns, marked with solid lines in Figure 1, are subsequently used to decide which filter is going to be applied (none, normal or strongest). In each side of the boundary, only up to four neighboring have to be considered [1].

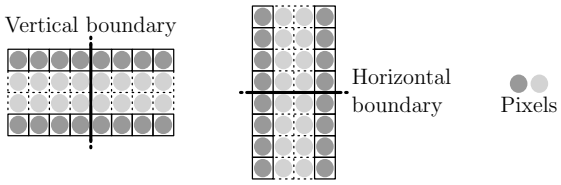

Fig. 1. Block unit where the filter may be turned on, for vertical and horizontal boundaries. Deblocking decisions are based on the pixel lines or columns marked with solid lines.

4. PARALLEL CPU+GPU IMPLEMENTATION

In the implementations that are presented herein, the eight BS conditions are evaluated by the CPU as soon as the entropy decoded data is acquired. Each condition is stored in one single bit, which can be 0 for a false condition and 1 when the condition is true. The eight BS conditions, with the respective bit positions, are described in Table 1. Then, the boundary conditions are stored in a two byte word, one byte for the horizontal filtering and another for the vertical filtering. This approach reduces the memory access by the CPU and the required memory transfers to the GPU. The final BS value for each boundary can be quickly obtained with bitwise operations, either in the CPU or in the GPU.

Since only up to four samples are needed and up to three samples can be filtered in each side of a boundary in the 8×8 pixel grid, there are several non-ovelapping blocks that can be filtered in parallel. Those *Independent Frame Regions* (IFRs) and their relative position in the 8×8 pixel grid are depicted in Figure 2. However, inside each of those regions, the hori-

Bit	BS conditions
7	Boundary in the 8×8 grid
6	TU boundary
5	PU boundary
4	One of the blocks is using Intra prediction
3	One of the blocks has residual data
2	Different reference frames *or* number of motion vectors are used in Inter prediction for the adjacent blocks
1	Absolute differences between the corresponding spatial components of the motion vectors of the blocks are greater than 1 (in units of integer pixels) for 1 reference frame
0	Absolute differences between the corresponding spatial components of the motion vectors of the blocks are greater than 1 (in units of integer pixels) for 2 reference frames

Table 1. Boundary filtering strength conditions and respective bit position in a byte.

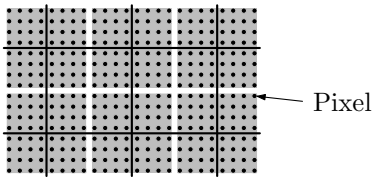

Fig. 2. IFRs are identified with gray squares, the boundaries of the 8×8 frame grid to be filtered are marked as solid lines.

zontal filtering of the vertical boundaries must be performed before the vertical filtering of the horizontal boundaries.

The chroma components are also filtered at the block boundaries of a 8×8 pixel grid. In this way, the IFRs are the same, but the BS conditions must be read according to the chroma subsampling format.

Then, each IFR (composed by four boundaries) is filtered by one thread, either at the CPU or at the GPU. To calculate the BS value for the chroma component, the BS conditions data of different IFRs may be used. However, since the memory content is not updated, the memory is not written, there is no dependency among IFRs.

In the CPU implementation, the IFRs, which are independently filtered, are equally distributed among the CPU cores, by using the POSIX Threads (Pthreads) API. This approach does not require any synchronization point, unlike the algorithms proposed in [13].

For the deblocking filter implementation in the GPU, each GPU thread is responsible for filtering only one IFR. Accordingly, each GPU thread block, composed by 32 GPU threads (1 warp), will filter 32 IFRs in a row. The GPU thread blocks are configured in a 2D grid, whose size will be:

$$\text{Grid.x} = \left\lceil \frac{\text{Frame Width}}{8 * 32} \right\rceil ; \text{Grid.y} = \left\lceil \frac{\text{Frame Height}}{8} \right\rceil . \quad (1)$$

All the GPU global memory accesses are performed at the warp level. However, whenever the GPU threads inside one warp require a memory access using strided memory addresses, the total memory access becomes serialized. To circumvent this, all the used memory addresses were properly aligned, so that one single memory access is required for the whole warp processing. On the other hand, the GPU global memory accesses can be reduced if the frame and the BS conditions data are accommodated in the global memory of the GPU in the form of vectors stored in a raster scan order and the warps are restricted to one single row of IFRs. The GPU shared memory (cache memory) is also used to store temporary pixel samples, in order to increase its efficiency.

In the GPU deblocking filter implementation, the memory transfers (from host to device and from device to host) are the most time consuming procedures. Nevertheless, multiple CUDA streams can be used to asynchronously overlap the memory transfers and the GPU kernel. With this approach, the GPU thread blocks are distributed among the defined CUDA streams. Figure 3 illustrates how the overall

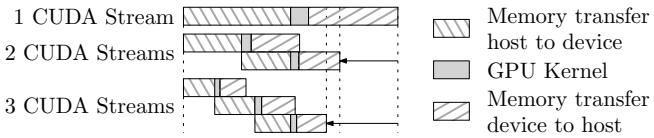

Fig. 3. Examples of asynchronous CUDA streams processing.

processing time can be reduced with CUDA streams. Since the GPU kernel can be completely overlapped by the memory transfers, the total processing time in the GPU implementation will only depend of the frame resolution.In the implementation described in this paper, the best number of concurrent CUDA streams was experimentally obtained, for each frame resolution.

To further increase the processing performance, a load balancing scheme for a collaborative processing using both the CPU and the GPU was devised. In the proposed collaborative implementation, the processing time of both executions for the current frame is monitored and used to predict the frame partitioning that will be applied in the following frame, in order to obtain the same processing time for both executions. This load balancing control, that chooses the best number of IFRs for the GPU and CPU, is executed separately for each slice type.

5. EXPERIMENTAL RESULTS

To assess and experimentally evaluate the processing throughput of the proposed CPU and GPU implementations, the common test conditions and configurations defined in [16] were adopted to consider video bit streams of all sequences from classes A, B and E (the greatest frame resolutions). An additional set of sequences with a 3840×2160 resolution (Ultra HD 4K) from the SVT High Definition Multi Format Test Set, composed by *CrowdRun*, *ParkJoy* and *DucksTakeOff*, was also evaluated. Those sequences will be referred to as class S (see Table 2).

The proposed deblocking filter implementation was integrated in the HEVC Test Model (HM) version 11.0 [17], which was also used as one of the benchmarks, for baseline comparison purposes. The hardware setup includes an Intel® Core™ i7-4770K CPU @ 3.50GHz, using Pthreads library to distribute the load between the four CPU cores. In the GPU side, it was used an NVIDIA Tesla K20c @ 706 MHz, with CUDA version 5.5.

Table 2 depicts the average execution time for the HEVC deblocking filter for each considered resolution and quantization (QP) parameters. The gathered experimental data was divided in I and B frames, from *All Intra* (AI) and *Random Access* (RA) configurations, respectively. The *Low Delay* configuration was not taken into account, since the corresponding results for B frames have a similar timing as in the RA configuration, as well as the I frames in the RA and AI configurations. Accordingly, this setup avoids a misleading analysis

Class	Resolution	QP	I Frames (All Intra Configuration)				B Frames (Random Access Configuration)			
			HM 11.0	CPU Only	GPU Only	CPU+GPU	HM 11.0	CPU Only	GPU Only	CPU+GPU
S	3840x2160	22	70.97	4.67	2.99	2.04	45.67	3.47	2.99	1.92
		37	54.69	4.95	3.01	2.11	18.78	2.10	2.98	1.59
A	2560x1600	22	28.30	2.21	1.71	1.15	15.96	1.46	1.70	1.01
		37	27.66	2.54	1.72	1.23	9.01	0.96	1.70	0.86
B	1920x1080	22	15.13	1.23	1.08	0.81	7.16	0.72	1.07	0.67
		37	12.80	1.35	1.09	0.83	3.78	0.48	1.06	0.47
E	1280x720	22	6.09	0.69	0.71	0.58	1.55	0.27	0.67	0.26
		37	5.09	0.66	0.71	0.58	1.08	0.23	0.66	0.22

Table 2. The average time in milliseconds for the HEVC deblocking filter of tested decoders.

between I and B frames.

As expected, the average execution time increases with the frame resolution. Nevertheless, since the number of filtering operations is lower in B frames, the "CPU Only" implementation is faster for those frames, although presenting a significant variability from frame to frame. On the other hand, the timing for the "GPU Only" implementation is almost constant for a specific resolution and it is independent from the QP, slice type or video content, since it mainly depends on the amount of data transfers between the host and the device.

When comparing the three implementations, it is observed that the load balanced "CPU+GPU" implementation provides the best performance. The considered scheme to share the load indirectly takes into account the frame resolution, QP, slice type and video content, in order to find the best load balancing among the CPU and the GPU. As result, this implementation can provide the minimal time, regardless the sequence and coding configuration.

When compared with the current state of the art, it is noticed that despite addressing different application constraints (like power consumption) the dedicated hardware implementations presented in [14] and [15] are characterized by a processing time as high as 1.1 ms and 33.9 ms, respectively, for the 1080p frame resolution. This corresponds to significantly lower performance than the "CPU+GPU" implementation that is herein proposed.

To evaluate the proposed implementations in terms of scalability, Figure 4 depicts the observed speedup by considering the HM benchmark as reference. The presented speedup range (gray bars) considers all the values that were obtained for all the sequences in each class, as well as the recommended QP set [16]. The lines show the average speedup value, among all the video classes, by using the *dash-dot*, *dashed* and *solid* lines to represent the "CPU Only", "GPU Only" and "CPU+GPU", respectively.

As it can be observed, the overall speedup for all considered implementations increases with the frame resolution, since the number of IFRs (parallelism level) increase with the frame resolution. Regardless the video class and frame type, the conducted optimizations in the "CPU Only" implementation provides an average speedup as high as 13.5 by using four CPU cores (efficiency $= 13.5/4 = 3.38$), which overcomes the last proposed multi-core HEVC deblocking filter

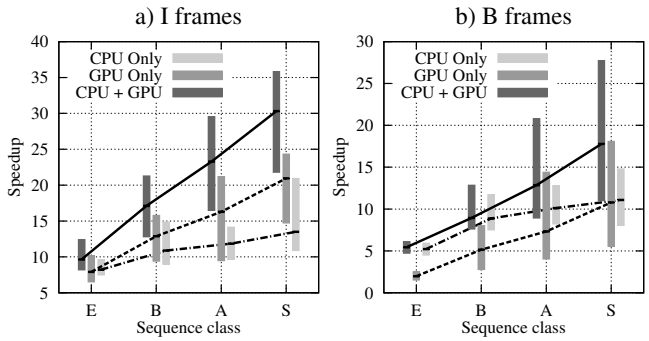

Fig. 4. Speedup range for the proposed implementations against the HM 11.0 with one core.

design [13], which only obtained a maximum speedup of 5 by using six CPU cores (efficiency $= 5/6 = 0.83$) and the same HM reference benchmark. The "GPU Only" implementation outperforms the "CPU Only" mainly for I frames, since the filtering time is larger than the memory transfer time between the host and the device. This does not happen in B frames, where the "CPU Only" has better performance. The "CPU+GPU" implementation outperforms any of the other implementations in all configurations.

6. CONCLUSION

Three efficient parallelizations of the HEVC deblocking filter to be executed on heterogeneous platforms composed by multiple CPUs and GPU accelerators were presented in this paper. The conducted CPU parallelization outperforms the current state of the art, by adopting a new approach for the BS calculation with minimal memory accesses and by exploiting independent frame regions to implement the filtering. The same strategy was also applied in the implementation of this module in GPU devices. To the best of the authors' knowledge, the proposed GPU parallelization of the HEVC deblocking filter is one of the first in the literature. By simultaneously exploiting the CPU and GPU computational resources, the proposed load-balanced implementation ("CPU+GPU") was able to achieve speedup values as high as 35.8, obtained for the Ultra HD 4K (3840×2160 pixels) video sequence *DucksTakeOff*, corresponding to an overall processing time that is less than 1.88 ms.

7. REFERENCES

[1] ITU-T Recommendation H.265 and ISO/IEC 23008-2, ITU-T and ISO/IEC JTC 1, *High Efficient Video Coding (HEVC)*, Apr. 2013.

[2] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1649–1668, 2012.

[3] J. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards – including high efficiency video coding (HEVC)," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1669–1684, 2012.

[4] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC complexity and implementation analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1685–1696, 2012.

[5] A. Norkin, G. Bjøntegaard, A. Fuldseth, M. Narroschke, M. Ikeda, K. Andersson, M. Zhou, and G. V. der Auwera, "HEVC deblocking filter," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1746–1754, 2012.

[6] NVIDIA Corporation, *NVIDIA® CUDA™ Compute Unified Device Architecture Programming Guide*, version 1.0: Jun. 2007 (and subsequent editions).

[7] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A unified graphics and computing architecture," *Micro, IEEE*, vol. 28, no. 2, pp. 39–55, 2008.

[8] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach*, Morgan Kaufmann Publishers Inc., Waltham, MA, USA, 2nd edition, 2013.

[9] X. Wang, L. Song, M. Chen, and J. Yang, "Paralleling variable block size motion estimation of HEVC on multi-core CPU plus GPU platform," in *Image Processing (ICIP), 2013 20th IEEE International Conference on*, 2013, pp. 1836–1839.

[10] J. Li, O. C. Au, L. Fang, L. Sun, W. Sun, and D. Soysa, "A parallel deblocking filter based on H.264/AVC video coding standard," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, 2013, pp. 233–236.

[11] H. Su, J. Chai, M. Wen, J. Ren, and C. Zhang, "Parallelization design of irregular algorithms of video processing on GPUs," in *Multimedia and Expo (ICME), 2012 IEEE International Conference on*, 2012, pp. 997–1002.

[12] L. Yan, Y. Duan, J. Sun, and Z. Guo, "Implementation of HEVC decoder on x86 processors with SIMD optimization," in *Visual Communications and Image Processing (VCIP), 2012 IEEE*, 2012, pp. 1–6.

[13] A. M. Kotra, M. Raulet, and O. Deforges, "Comparison of different parallel implementations for deblocking filter of HEVC," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, 2013, pp. 2721–2725.

[14] W. Shen, Q. Shang, S. Shen, Y. Fan, and X. Zeng, "A high-throughput VLSI architecture for deblocking filter in HEVC," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, 2013, pp. 673–676.

[15] E. Ozcan, Y. Adibelli, and I. Hamzaoglu, "A high performance deblocking filter hardware for high efficiency video coding," *Consumer Electronics, IEEE Transactions on*, vol. 59, no. 3, pp. 714–720, 2013.

[16] F. Bossen, "Common test conditions and software reference configurations," Doc. JCTVC-L1100 of JCT-VC, Geneva, Switzerland, Jan, 2013.

[17] JCT-VC, "Subversion repository for the HEVC test model version HM 11.0," 2013.