HEVC In-Loop Filters GPU Parallelization in Embedded Systems

Diego F. de Souza, Aleksandar Ilic, Nuno Roma and Leonel Sousa
INESC-ID, IST, Universidade de Lisboa
Rua Alves Redol 9, 1000-029, Lisbon, Portugal
Email: {diego.souza,aleksandar.ilic,nuno.roma,leonel.sousa}@inesc-id.pt

***Abstract*—The added encoding efficiency and visual quality that is offered by the latest HEVC standard is mostly attained at the cost of a significant increase of the computational complexity at both the encoder and decoder. However, such added complexity greatly compromises the implementation of this standard in computational and energy constrained devices, including embedded systems, mobile and battery supplied devices. To circumvent this limitation, this paper proposes the exploitation of embedded GPU devices already equipping many state of the art SoCs to accelerate the HEVC in-loop filters (i.e. deblocking filter and sample adaptive offset). The presented approaches comprehensively exploit both fine and coarse-grained parallelization opportunities of these filters in an NVIDIA Tegra GPU.According to the conducted experimental evaluation, the proposed approach showed to be a remarkable strategy to satisfy the real-time requirements of the HEVC decoder, being able to filter each Ultra HD 4K intra frame in less than 20 ms (about 50 fps).**

I. Introduction

When compared with previous video standards (e.g., the H.264/MPEG-4 AVC), the state-of-the-art High Efficiency Video Coding (HEVC) standard [1] has shown to provide equivalent subjective visual quality, while achieving bit rate reductions as high as 50%. However, such coding efficiency comes at the cost of a substantial increase in the computational complexity of both the video encoder and decoder [2]. Such added cost is often regarded as an important limiting factor not only for desktop environments, but specially for embedded and portable systems with constrained computational resources (e.g. battery supplied smartphones, tablets, smartTVs, etc.).

To benefit from the offered bitrate reduction and improved visual quality, the HEVC encoder and decoder tightly couples the functionality of several modules, each with different computational complexities and execution profiles. In particular, the in-loop filters that are present both at the encoder and decoder sides, i.e., Deblocking Filter (DBF) and Sample Adaptive Offset (SAO), are responsible for reducing the block artifacts and for applying gradient based filtering, thus promoting the resulting visual quality and compression efficiency.

In particular, when regarded from the video compression performance perspective, the HEVC DBF module is responsible for an average bit rate reduction between 3.3% and 6% [3]. Conversely, the HEVC SAO filter has shown to provide bit rate reductions between 3.5% and 23.5% [4]. When evaluated from the computational load perspective, the conducted profiling of the decoder [2] on an ARM Cortex-A9 processor has shown that the HEVC in-loop filters are responsible for 19%–21% of the decoding time. As a consequence, reducing the HEVC decoding time via parallelization of the decoder modules represents an important research topic in modern video coding, namely the in-loop filters.

To overcome this problem, most System on Chips (SoCs) for embedded and mobile devices already include dedicated hardware for encoding and decoding of one or more video standards. For example, the NVIDIA Tegra® K1, Samsung Exynos 5 Octa and Allwiner A80 SoCs natively support H.264/MPEG-4 AVC decoding through hardware. However, the offered video encoder/decoder capabilities are usually highly restricted to specific standard profiles and video configurations (e.g., resolution, encoding tools and frame rate).

In this scenario, the work proposed herein represents a first step towards a software-based video decoding solution for embedded systems that is fully compliant with the HEVC standard. In particular, this work focuses on efficient parallelization of the HEVC in-loop filters on state-of-the-art SoCs with embedded Graphics Processing Unit (GPU). In fact, modern SoCs from several manufactures already encompass low-power GPUs, such as Tegra (NVIDIA), Mali (ARM), PowerVR (Imagination Technologies), Adreno (Qualcomm), etc. Besides their original graphic purposes, these embedded GPUs can also be used for general-purpose computations, due to their ability to efficiently exploit data-parallelism in different applications. Specifically, the NVIDIA's Tegra K1 GPU demarks from this vast set because it supports the Compute Unified Device Architecture (CUDA) [5], allowing a migration of existing CUDA algorithms from desktop to embedded platforms, by requiring a carefully redesign due to the limited capabilities of embedded GPUs.

Although some HEVC decoder modules (namely DBF [6] and Inverse Transform [7]) have been already implemented for desktop state-of-the-art GPUs, to the best of the authors' knowledge there is not yet any state-of-the-art approaches in the literature targeting the parallelization of all HEVC in-loop filters on GPU, specially in embedded environments. Therefore, the proposed parallel implementation of the HEVC in-loop filters that is herein presented improves the GPU DBF algorithm [6] and develops the parallel HEVC SAO module.

The proposed parallel approaches of the HEVC DBF and SAO modules allow to efficiently exploit the fine-grain GPU parallelism, by re-designing the execution pattern of these decoding modules. As a result, the proposed algorithms allow to achieve lower overall processing time, e.g., up to 20 ms for Ultra HD 4K (3840×2160) frames on the CUDA-enabled Jetson TK1 development board, which corresponds to

978-1-4673-7311-1/15/$31.00 ©2015 IEEE

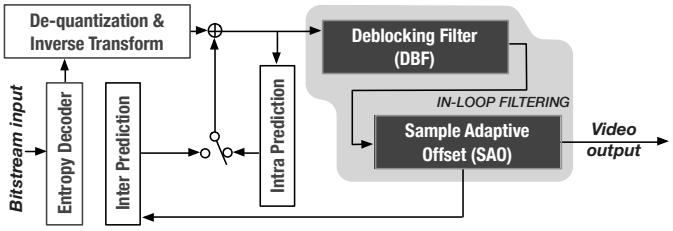


Fig. 1. Block diagram of the HEVC decoder.

a minimum frame rate of 50 frames per second (fps).

The remaining of this paper is organized as follows. Section II provides a brief overview on the basic functional principles behind the HEVC in-loop filters, while Section III revises the state-of-the-art approaches for HEVC filtering modules. The proposed algorithms and consequent parallel implementations are presented in Section IV. The obtained experimental results and the derived conclusions are presented in Sections V and VI, respectively.

II. HEVC Decompression

Figure 1 depicts a generic block diagram of the HEVC decoder. First, the input bitstream is decoded by the entropy decoder, in order to produce the coefficient data, as well as all other information needed to decompress the video sequence. The coefficient data is then de-quantized and inverse transformed, in order to obtain the residual data. The reconstructed block is computed by adding the residual data with the predicted block from either inter or intra prediction. To attenuate the blocking artifacts introduced by the block-based prediction and transform coding, the DBF is then applied at the boundaries of the reconstructed blocks. Finally, the mean sample distortion is further reduced with the application of the SAO module, where the final video output is produced.

Along the HEVC encoding procedure, each picture is partitioned into a grid of $L\times L$ pixel blocks (denoted as Coding Tree Units (CTUs)), where L is dynamically selected by the encoder procedure ($L\in\{16, 32, 64\}$). The CTUs are then grouped in slices or tiles, which are processed in raster scan order at the decoder. Each CTU is independently split in smaller blocks (called Coding Units (CUs)) according to a quadtree structure, from a maximum size of 64×64 pixels to a minimum size of 8×8 pixels. Additionally, each CU is further divided in Prediction Units (PUs) and Transform Units (TUs), corresponding to the prediction and to the residual blocks, respectively [8]. Inside each CTU, the CUs are decoded by following a z-scan order, as well as the PUs and the TUs within each CU.

For each video component (i.e., luma and both chromas), the same frame partitioning (CTU, CU, PU and TU) is applied. In particular, when the usual 4:2:0 chroma subsampling is adopted, the chroma blocks are four times smaller than the corresponding luma blocks, until the minimum size of 4×4 pixels.

A. Deblocking Filter

In the HEVC standard, the deblocking filter is only applied to the boundaries of the PU and TU, which rely on a 8×8 samples grid for both luma and chroma. For each boundary, a Boundary filtering Strength (BS) is evaluated, according to several conditions from the neighboring blocks. The resulting BS value varies between 0 and 2, where 0 means that no deblocking filter will be applied. Whenever one of the neighboring blocks is intra-predicted, the BS value is always set to 2. Moreover, only when the BS value is two, the chroma samples are filtered [3].

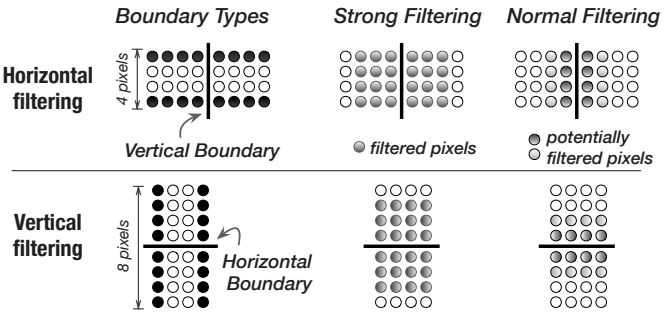


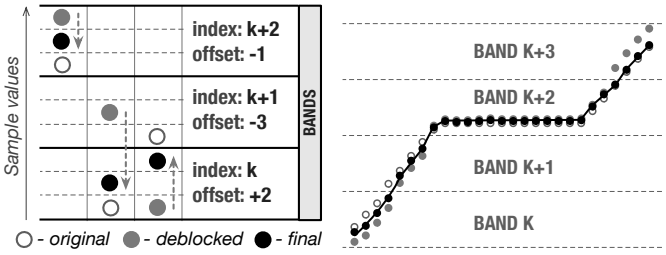
Fig. 2. Boundary types of the HEVC Deblocking Filter. Filtering decisions are made based on the pixel lines or columns dark-filled.

In what concerns luma boundaries, additional conditions are verified to determine whether the DBF should be applied. Each condition is verified for each set of 8×4 or 4×8 pixels, corresponding to the vertical and horizontal edges, respectively (see *Boundary Types* in Fig. 2). Accordingly, a set of pixels in the first and the last row (or column) are used to decide which filter is going to be applied, i.e., none, normal or strong (see dark-filled pixels in Fig. 2). In each side of the boundary, only up to four neighboring samples have to be considered and up to three may be modified. Taking the luma component as an example, the *strong* filtering is applied on three pixels in each side of the boundary, while at most two pixels may be filtered on each side of the boundary in the *normal* filtering, which depends on a set of DBF conditions (see *Strong Filtering* and *Normal Filtering* in Fig. 2). In contrast, the *normal* filtering is only applied on a single pixel in each side of the boundary for chroma samples. Finally, the HEVC standard specifies that all vertical edges from the frame are processed by the DBF before the filtering procedure of the horizontal edges [1].

B. Sample Adaptive Offset

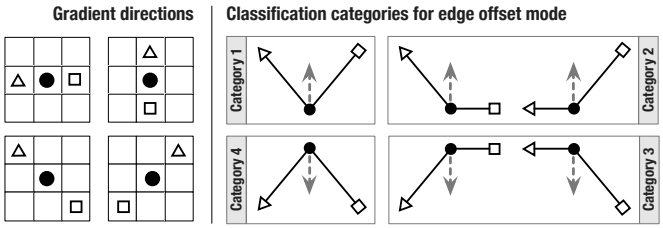
As depicted in Fig. 1, the reconstructed samples are processed by the SAO module soon after being filtered by the DBF module. In this case, the deblocked samples are subsequently modified by adding an offset value whose magnitude depends on a set of SAO parameters, namely, *Type*, four *Offset Values* and *Band Position/Edge Class*. These SAO parameters are encoded in the bitstream for each CTU and can have different values for the luma and the two chroma components of each CTU [4]. In particular, the *SAO Type* parameter signals the decoder which SAO filtering should be applied (none, band offset or edge offset).

Regarding the band offset mode, the full amplitude of the pixel range is divided by 32 in order to define a set of *bands*. The filtering procedure for this mode consist of adding a offset value to all samples whose values belong to the same *band*. For example, in Fig. 3(a), the deblocked samples from *bands* with indexes k, $k+1$ and $k+2$ are added to offset values of $+2$, -3



(a) SAO Band Offset filtering.

(b) SAO Band Offset example.



(c) Gradient directions and categories for the SAO Edge Offset.

Fig. 3. SAO Band Offset and Edge Offset modes.

and -1, respectively, in order to push the final sample values towards the original ones. To reduce the complexity, in the HEVC standard, only four consecutive *bands* are considered for SAO band offset filtering. In this way, only the lowest *band* index needs to be stored in the bitstream, namely the *SAO Band Position* (k in Fig. 3(a)). For each processed band, a single offset value is provided in the respective *SAO Offset Value* parameter. In Fig. 3(b), an example of corrupted deblocked samples by the quantization errors are presented in gray-filled dots, where the horizontal and vertical axis denote sample spatial position and value, respectively. In this case, the final filtered samples (dark-filled dots in Fig. 3(b)) from *bands* k to $k+3$ are corrected with the SAO band offset filtering by moving towards to the original samples (white-filled dots).

For the edge offset SAO mode, the decoded CTU samples are classified into four categories according to the corresponding gradient direction, as specified in the *SAO Edge Class* parameter. Figure 3(c) depicts all four possible gradient directions and allowed SAO categories. Similarly to the band offset mode, the offset value assigned to each category is stored in the *SAO Offset Value* parameter. The *SAO Offset Value* is positive for categories 1 and 2 and negative for categories 3 and 4 (see arrows in Fig. 3(c)). Hence, whenever a pixel is classified in one of these categories, its deblocked sample is added to the corresponding *SAO Offset Value*. Whenever, the deblocked samples can not be classified in any of these categories the SAO edge offset filtering is not applied.

III. Related Work

Several video encoding and decoding modules have been implemented in high-end GPU devices along the past years. At the encoder side, most of such GPU-based implementations deal with the most computational demading motion estimation schemes (often also supporting relaxed dependencies), as proposed in [9], [10] for HEVC and in [11] for H.264/MPEG-4 AVC. However, very few efficient GPU approaches for HEVC decoding have been proposed and even less have considered the explotation of mobile GPUs. In fact, parallel implementations often pose difficult challenges at the decoder side, not only because the decoder should be able to support bitstreams produced by any encoder configuration, but also because the processing platform at the decoding device often imposes restrictive processing capabilities.

In [12], Chi et al. exploit Single Instruction, Multiple Data (SIMD) parallelism to implement HEVC decoder modules by specifically focusing on modern multi-core Central Processing Unit (CPU) architectures, including ARM processors. To maximize the attained performance, the authors in [12] divide the computational load among CPU cores, by relying on an alternative method based on the HEVC Wavefront Parallel Processing (WPP) [1]. At the end, an average frame rate of 35.5 and 77.8 fps for Full HD video sequences was achieved with an ARM Cortex-A9 and an ARM Cortex-A15, respectively. However, the time consumed per HEVC decoding modules was not provided for ARM processors by these authors [12], making a direct comparison difficult to be established. Furthermore, contrary to the multi-core implementation proposed in [12], the GPU parallel implementations proposed herein are fully compliant with the HEVC standard. Also, it could take advantage of both HEVC parallelization techniques, namely *Tiles* and WPP [8].

When looking at different approaches based on dedicated hardware often supported on Field-Programmable Gate Array (FPGA) technology, the HEVC in-loop modules have also been designed for the encoder [13] and decoder [14]. However, such implementations usually represent different compromises in terms of programmability, resources utilization and energy efficiency, preventing a fair comparison with high-performance computing platforms, like GPUs.

Despite the absence of existing approaches that tackle GPU implementations of the entire HEVC decoder, individual decoding modules (such as, deblocking filtering [6] and inverse quantization and transform [7]) have already been proposed by the authors targeting high performance GPU platforms. However, convenient changes and adaptations had to be developed in order to make it suitable for embedded GPUs. Moreover, a new GPU parallel algorithm of the SAO filtering module is proposed in this paper, which represents one of the first approaches in the literature to handle this HEVC module in embedded GPUs.

IV. Proposed Parallel Algorithms

The proposed implementation of the HEVC in-loop were devised to efficiently exploit the offered parallel processing capabilities already made available on mobile GPU architectures. They leverage the fine-grain parallelism of these computationally complex modules, while providing fully compliant HEVC decoding. The GPU execution is organized in groups of 32 parallel threads (warps), which are grouped in several Thread Blocks (ThBs). To maximize the attained performance, the proposed algorithms carefully maximize the number of active warps by targeting for embedded GPUs, while ensuring that all threads in a warp perform the same operation from the GPU code (kernel). Furthermore, the data accesses were carefully managed, in order to efficiently exploit the complex embedded GPU memory hierarchy (i.e., global, cache, shared and constant) [5].

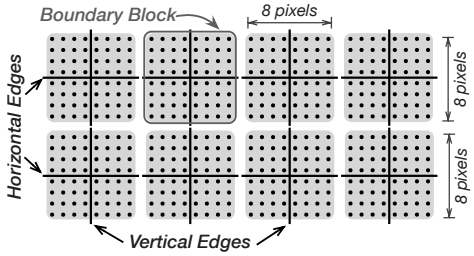


Fig. 4. Edge-level parallelism exploited by the proposed GPU deblocking filtering algorithm.

A. Deblocking Filter

As referred in Section II, the DBF module considers up to four samples within a 4×8 (or 8×4) pixel region, in order to filter up to three samples in each side of the considered boundary (as it was shown in Fig. 2). According to the HEVC standard [1], this procedure is firstly applied on all vertical edges in a frame, followed by the processing of the horizontal ones.

However, although this processing order fits well with CPU architectures, it might not deliver a suitable degree of parallelism when exploiting GPU hardware, specially embedded GPUs. First, inevitable synchronization between the two DBF stages may significantly degrade the overall GPU performance, since it is required either to launch separate GPU kernels or to synchronize the execution over the global memory. Second, this approach involves many superfluous data transfers and does not allow efficient utilization of the embedded GPU memory hierarchy. For example, upon the processing of the vertical edges, the data needs to be stored in the global memory, and retrieved again when filtering the horizontal edges. Additionally, the memory access pattern for the vertical filtering involves column-wise strided data accesses, which require several global memory transactions to fetch a single portion of horizontally filtered data.

To circumvent these limitations, the proposed DBF implementation exploits an improved strategy for efficient GPU parallelization, by relying on a different paradigm for the execution of the two DBF stages. As presented in Fig. 4, when two consecutive horizontal 4×8 filtering regions are considered, one can identify several non-ovelapping blocks that can be filtered in parallel [6]. These 8×8 pixel blocks, herein referred to as Boundary Blocks (BBs), allow performing both horizontal and vertical filtering on a small subset of locally stored and independent input data. Hence, all 8×8 BBs in a frame can be simultaneously executed without any need for synchronization, leading to an efficiently utilization of the GPU memory hierarchy.

As shown in Fig. 5, which depicts the assignments of the proposed DBF GPU algorithm, each ThB is assign to perform the deblocking filter on a row of 32 BBs (i.e., 256×8 luma pixels). Inside each ThB, four warps are in charge of carrying out the deblocking filter in a row of 8 BBs (i.e., a block of 64×8 luma pixels). In order to cope with the increased warp-level data requirements, the shared memory is used as an additional memory space to store the GPU register values. As a result, each warp has its own 64×8 memory space for storing the intermediate filtered samples.

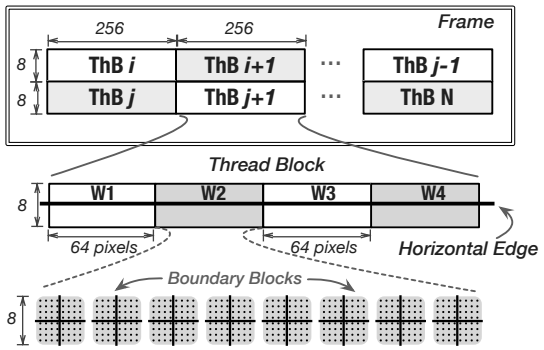


Fig. 5. Thread block assignments for one frame, consisting of four warps per ThB and eight BBs per warp (*Wi*).

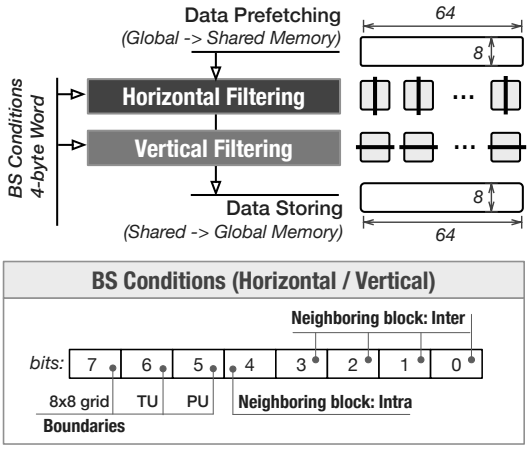


Fig. 6. Warp-level processing for the proposed GPU implementation and BS Conditions bit assignments.

Figure 6 represents the execution of the proposed data-parallel DBF algorithm at the level of a single warp. First, all threads in a warp simultaneously copy a 64×8 luma block from the global memory to its own portion of shared memory space (*Data Prefetching*). In this case, each thread copies one pixel at time, leading to one memory instruction to copy 32 luma pixels in a row. The consecutive 32 pixels in the same row are also copied with one memory instruction. Since this data is already coalesced, the GPU L2 cache can be maximally exploited.

Then, the deblocking filter is performed on the data that is already in the shared memory, where two threads are assigned to perform DBF on a single BB. For each BB, two edges are filtered at the same time, i.e., two horizontal filtering procedures are performed on vertical boundaries (*Horizontal Filtering*) before the two vertical filtering procedures are applied on the horizontal boundaries (*Vertical Filtering*). Finally, the data is stored back into the GPU global memory (*Data Storing*).

For both filtering steps, the input data that is required to calculate the BS conditions are packed by following an approach like to the one in [6] for each 4×4 edge. The BS conditions and the adopted bit encoding (in a byte) are shown in Fig. 6 (see *BS Conditions*). Herein, the *neighboring blocks* information is packed in bits 0 to 4, where bit 4 signals if one of the blocks are intra predicted and bits 0 to 3 contain

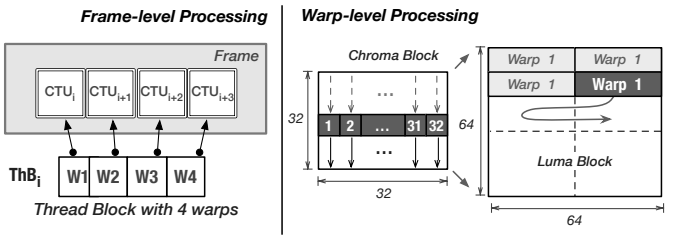


Fig. 7. Frame-level and warp-level processing of the proposed parallel SAO algorithm, where each ThB is responsible for four CTUs in a row and each warp (*Wi*) is assigned to one CTU.

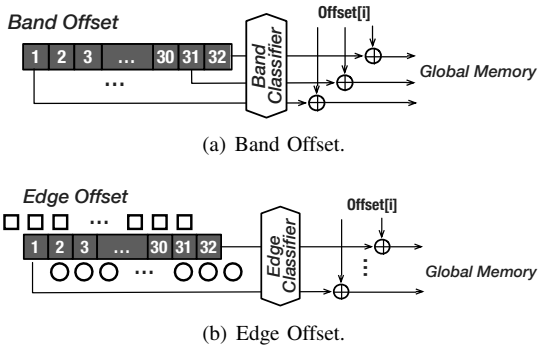


Fig. 9. Thread-level processing of the proposed SAO algorithm.

related inter prediction data of neighboring blocks (residual data, reference frames and motion vectors). The remaining bits are related with *boundaries* characteristics, where bits 5, 6 and 7 signal if the boundary is a PU edge, a TU edge and belongs to the 8×8 grid, respectively. This approach reduces the required memory transfers in the GPU, where the final BS value for each boundary can be quickly obtained with simple bitwise operations. Finally, each BB requires only four bytes to be processed, two bytes for *Horizontal Filtering* and two bytes for *Vertical Filtering*.

It is also worth nothing that the whole shared memory block that is assigned to a warp is used for both luma and chroma components. Hence, while such 64×8 space is used (in its entirety) to perform the DBF of the luma component, this same amount of space is equally divided for the two chroma components, i.e., a 32×8 chroma Cb and 32×8 chroma Cr blocks are retrieved (in parallel) in the *Data Prefetching* phase, and the DBF procedure is simultaneously performed on both chroma components.

B. Sample Adaptive Offset

In the proposed parallel implementation of the SAO algorithm, each ThB (with four warps) is responsible for applying the SAO filtering to 4 CTUs in a single frame row, where each warp is assigned to one CTU, as depicted in Fig. 7 (see *Frame-level Processing*). Hence, each warp executes (in parallel) out the SAO procedure for 32 pixels, i.e., one pixel line at a time.

The 32 pixels of each line of the 32×32 chroma block, are simultaneously processed, as shown in Fig. 7 (see *Warp-level Processing*). In contrast, when greater CTUs are considered (i.e., 64×64 luma block), the SAO is applied to each 64 pixels line, where the left-most set of 32 pixels is processed first, and then the right-most 32 pixels of the same row are processed.

In order to handle the complexity of the SAO procedure and to efficiently use the GPU memory hierarchy, a specific data structure was developed, i.e., the *SAO Control and Offset Data*, as depicted in Fig. 8. For each frame component, the proposed 4-byte structure allows storing the maximum size of each SAO parameter, as specified by the HEVC standard, while still providing an unused bit (0). As it is shown in Fig. 8, the SAO parameters are divided in *SAO Control* and *Offset Data* fields, where the *SAO Control* field contains the *SAO Type* and the *SAO Band/Class*. Here, bits 1 and 2 are used to store the three possible *SAO Types*, namely, 0: none, 1: band offset and 2: edge offset. Contrasting to the *SAO Class* field, which can only have four possible values (gradient directions), the *SAO Band* can contain up to 32 different values, in order to represent the considered SAO band position. Hence, to ensure the compliancy with the HEVC standard, the SAO class/band has to be stored in a five bits field, i.e. bits 3 to 7. Finally, the remaining bits are used to store the four *SAO Offsets* (bits 8 to 31), where each positive/negative offset occupies 6 bits, since its magnitude can be up to 31, according to the HEVC standard. Hence, the SAO parameters for each CTU are stored in a 12-byte word (4-bytes for the luma and each chroma).

Figure 9 also provides a general overview of the data-parallel *Thread-level processing* applied on a set of 32 pixels, in order to perform the computations from the *Band Offset* and *Edge Offset* SAO types. For the *Band Offset* (see Fig. 9(a)), each thread (from 1 to 32) performs the following set of operations: *i*) the data is firstly fetched from the global memory; *ii*) the sample pixel value is classified according to the *SAO Band* parameter in the *Band Classifier*; *iii*) the corresponding *Offset[i]* is added; and *iv*) the final value is stored in the global memory. A similar procedure is performed in the *Edge Offset* (see Fig. 9(b)), where each thread fetches 3 pixel samples according to the *SAO Class* parameter, then the *Edge Classifier* is applied to determine the pixel category and choose the corresponding *Offset[i]* value. In order to reduce the complexity and avoid branch divergence, a parallel version of the fast sample classifiers proposed in [4] was implemented.

V. Experimental Evaluation

To experimentally evaluate the efficiency of the proposed parallel algorithms of the HEVC in-loop filters in a mobile

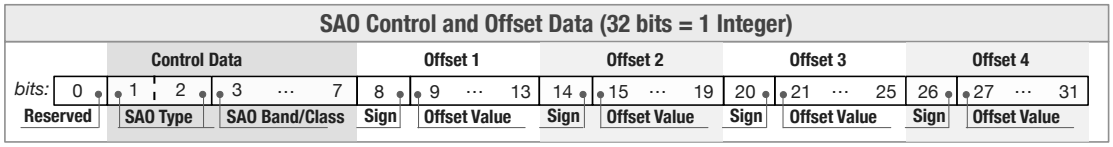


Fig. 8. Data structure used by the proposed parallel SAO implementation.

GPU, the recommended JCT-VC test conditions were adopted with an *All Intra* configuration [15]. The set of video bitstreams from the highest frame resolution (classes A, B and E) were considered, since they are the most computationally demanding. To further challenge the proposed GPU implementation, an additional set of Ultra HD 4K sequences [16] was also evaluated (class S).

To fully exploit the targeted GPU architecture, the proposed algorithms were implemented with the CUDA framework [5] and integrated in the reference HM 15.0 HEVC decoder [17]. Accordingly, the proposed parallel implementations of the HEVC DBF and SAO modules were handled by the GPU, while other decoder modules were executed by the CPU.

To conduct this evaluation, NVIDIA Jetson TK1 CUDA-enabled development board (with CUDA 6.5) was chosen. This board provides a Tegra® K1 SoC, containing a NVIDIA Kepler GK20a GPU @ 852 MHz – one Streaming Multiprocessor (SMX) with 192 CUDA cores – and the 4-Plus-1™ quad-core ARM® Cortex-A15 CPU @ 2.32 GHz.

Since the CPU and GPU share the same memory space, the input data required to perform the HEVC in-loop filters is directly obtained from the SoC main memory through CUDA zero copy instruction. Due to the limited compute capability of this embedded GPU, the GPU kernel configurations (i.e. number of warps per ThB, shared memory usage, registers consumption, etc.) must be carefully chosen to maximize the number of active warps in the SMX. To attain the maximum performance, the kernel configurations of the proposed approaches were experimentally determined by evaluating trade-offs between register spilling and number of active warps.

Table I presents the experimentally obtained average processing times for each proposed GPU implementation of the HEVC in-loop filters. In this evaluation, all the adopted video sequences from A, B, E and S classes were encoded with the lowest and the highest considered Quantization Parameters (QPs) (i.e., 22 and 37) since they are the most and least computationally demanding configurations. As it was expected, the average frame processing time obtained with the proposed GPU HEVC parallel modules varies across different classes (e.g., the processing of the highest resolution frames results in the highest per-module time). Moreover, for all tested video sequences and QP parameters, the DBF represents the most time consuming in-loop filter, due to its higher computational load and to the fact that the parallel SAO module exploits a higher amount of data parallelism, thus resulting in a significantly lower processing time. However, in all tested configurations, the real-time in-loop filtering (less than 12 ms) was achieved on GK20a with an average power consumption of 8.4 W at the level of the overall development board.

TABLE I. RESULTING AVERAGE PROCESSING TIME (IN MILLISECONDS) PER HEVC MODULES IN THE GK20A AND K20C GPUS.

Class	QP	GK20a		K20c
		SAO	DBF	DBF [6]
S 3840×2160	22	7.06	10.25	2.99
	37	6.01	11.67	3.01
A 2560×1600	22	3.83	5.54	1.71
	37	3.48	6.13	1.72
B 1920×1080	22	2.16	2.85	1.08
	37	1.84	3.25	1.09
E 1280×720	22	0.99	1.42	0.71
	37	0.92	1.60	0.71

TABLE II. AVERAGE FRAME PROCESSING TIME (IN MILLISECONDS) TO EXECUTE THE DBF AND SAO WITH THE ORIGINAL HM 15.0 (SINGLE-CORE OF CORTEX-A15) AND WITH THE PROPOSED PARALLEL MODULES IN GK20A MOBILE GPU.

Class	Sequence	QP	HM 15.0	GK20a
S 3840×2160	CrowdRun	22	295.70	18.19
		27	275.85	18.95
		32	261.29	19.28
		37	238.91	18.83
	DucksTakeOff	22	272.27	16.73
		27	305.60	18.36
		32	221.88	18.08
		37	176.89	16.83
	InToTree	22	248.05	16.99
		27	226.49	17.81
		32	201.12	18.25
		37	166.71	16.79
	OldTownCross	22	273.54	16.57
		27	277.64	18.05
		32	222.47	18.41
		37	194.25	15.57
	ParkJoy	22	275.95	18.08
		27	256.27	18.56
		32	236.03	18.41
		37	229.22	18.52
A 2560×1600	Traffic	22	163.92	9.82
	PeopleOnStreet	22	161.25	9.74
	Nebuta	22	86.00	8.89
	SteamLocomotive	22	102.35	9.00
B 1920×1080	Kimono	22	62.04	4.98
	ParkScene	22	80.68	5.09
	Cactus	22	75.73	5.06
	BQTerrace	22	75.90	4.80
	BasketballDrive	22	77.07	5.11
E 1280×720	FourPeople	22	37.16	2.46
	Johnny	22	29.67	2.34
	KristenAndSara	22	32.04	2.43

The last column in Table I refers to the processing times obtained with an alternative HEVC DBF parallel implementation, as proposed in [6], evaluated on an NVIDIA Tesla K20c @ 706 MHz GPU (including 13 SMXs) with CUDA version 5.5. As expected, the higher computational capabilities of K20c resulted in achieving even lower processing times. However, this high performance was achieved at a the cost of significantly higher power consumption, i.e., 88 W when only the K20c GPU is considered. As a result, the herein proposed approaches for the HEVC in-loop filtering provide up to 70% energy savings on considered embedded device.

Although these two DBF implementations allow achieving real-time processing, it is worth to emphasize that their direct comparison can hardly be performed on a fair basis, since these approaches target GPU devices with different purposes and capabilities (e.g. the number and architecture of the SMXs). Moreover, in the considered NVIDIA Jetson TK1 development board the energy savings are expected to be even higher if power consumption of only the GPU GK20a is considered. However, in contrast to K20c, the NVIDIA Jetson TK1 does not provide the precise power consumption monitoring facilities to obtain these power measurements. Hence, the power consumption of the overall development board was obtained with an external power meter.

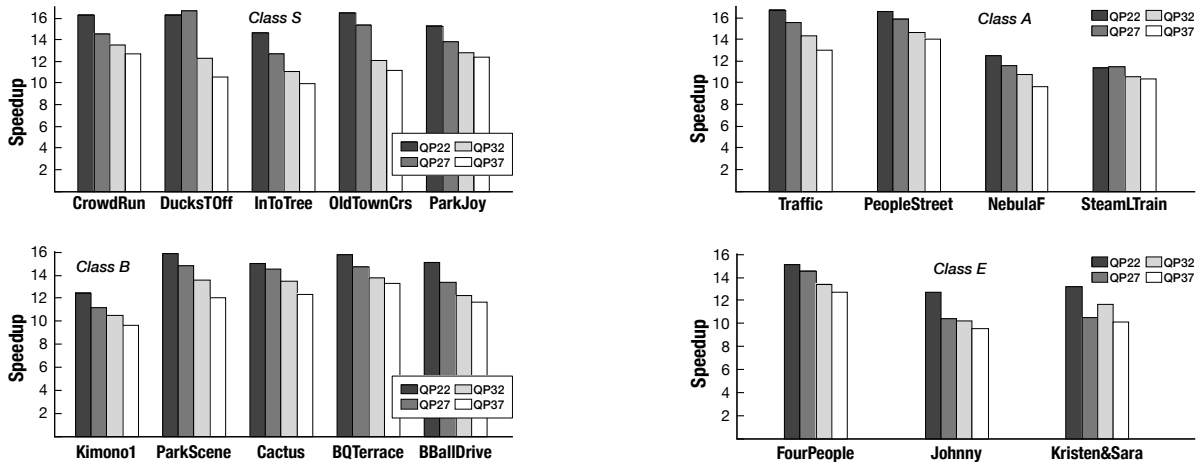


Fig. 10. Average speedup for all tested classes and QPs obtained with the proposed GPU HEVC in-loop decoding modules (DBF + SAO) on GK20a over the HM 15.0 on a single-core of the ARM Cortex-A15.

Another interesting phenomena is also worth noting in the considered set of video sequences when comparing the DBF processing time across different QPs: within a single class, the processing time slightly increases with the QP value. Such increment for larger QPs (lower bitrates) might be explained by the fact that the encoder tends to favor bitrate over distortion, in order to achieve higher bitrate savings. This yields to an increased presence of block artifacts, which are more visible for larger QPs, thus increasing the computation demand for the HEVC DBF module on the decoder side. On the other hand, the SAO module reveals to be more computational demanding for lower QPs, due to the increased details for higher spatial sample frequencies, i.e., more visual details obtained from higher bitrates.

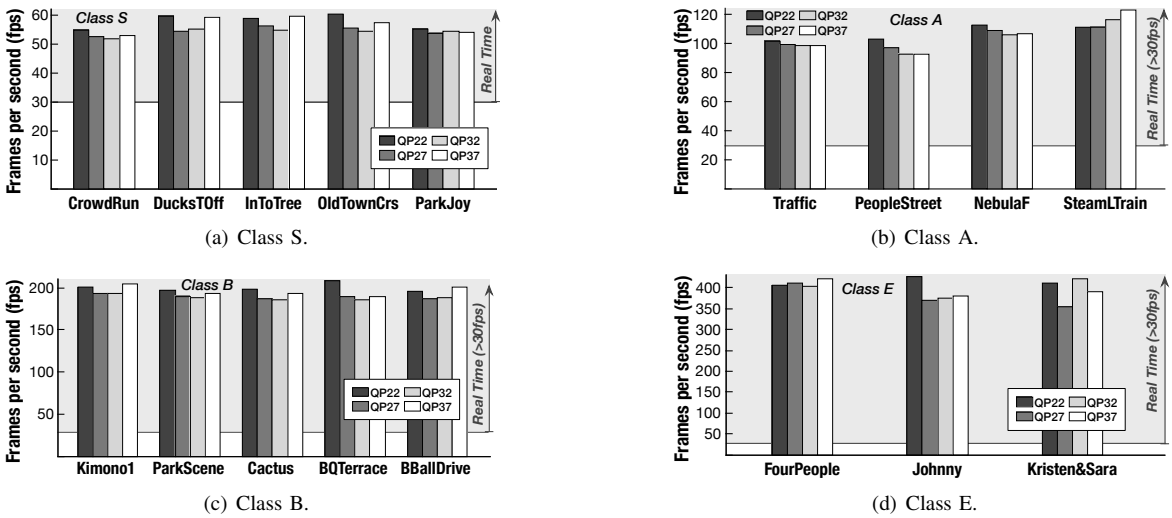
In order to further evaluate the efficiency of the proposed GPU HEVC in-loop filters (DBF+SAO), Table II presents the experimentally obtained average frame processing times for each considered test sequence. The reported time for the original HM 15.0 implementation when only DBF and SAO are considered was obtained on a single core of the ARM Cortex-A15 processor. The HM 15.0 in-loop filters implementation was chosen for the baseline comparison reference, since it is the most commonly used implementation in the literature. Moreover, there are no other state-of-the-art approaches in the literature implementing the considered HEVC in-loop decoding modules on desktop or embedded GPUs for a direct comparison.

As it can be observed in Table II, the overall processing time for the original HM 15.0 in-loop filtering significantly increases with the decrease of QP for all considered S class sequences. On the other hand, the processing time corresponding to the herein proposed parallel algorithms only slightly varies with the QP. This can be justified by the fact that when the processing time from one module increases, the time required by the other module decreases as it was shown in Table I. Accordingly, only the results corresponding to the more demanding evaluation (QP=22) are shown in Table II for the tested set of sequences from other classes, i.e., classes A, B and C.

The results presented in Table II also show the variation of the processing time according to the encoded data, i.e., video content. As it can be observed, the average processing time varies for the same class and QP in both the original HM 15.0 and the proposed GPU implementations. However, in the proposed GPU parallelizations, the impact of these encoder decisions is significantly attenuated. Moreover, the proposed algorithms achieve significantly lower processing times when compared with the HM 15.0. On average, across all sequences and QPs, the proposed algorithms outperform the HM 15.0 CPU execution for about 13× on the GK20a.

Figure 10 presents the speedups obtained on GK20a for all tested video bitstreams with the proposed algorithms when compared to the single-core HM 15.0 execution. As it can be seen, the higher speedups are obtained for the higher frame-resolution sequences, due to the increased amount of computational load. In general, the obtained speedup is higher for lower QP values (most computational demanding), since the processing time of the proposed GPU parallel implementation slightly varies over the QP value, as it was shown in Table II. Among the considered classes, an average speedup of 12×, 13.3×, 13× and 13.5× were obtained for classes E, B, A and S, respectively, where a maximum speedup of 16.7× was achieved (*Traffic* sequence with QP=22).

In Fig. 11, it is depicted the obtained average frame rate on GK20a for different QP parameters across a set of video sequences from different classes, namely, from classes S, A, B and E (see Fig. 11(a), 11(b), 11(c) and 11(d), respectively). As it can be observed, the proposed GPU implementation can handle real-time processing for all video sequences, i.e., a frame rate of at least 30 fps is always achieved. In particular, the proposed GPU algorithms allow achieving an average frame-rate of 51.9 fps for class S, 92.8 fps for class A, 185.5 fps for class B and 355 fps for class E, thus demonstrating the feasibility of effectively accelerating the in-loop decoding modules by using low-power GPUs from embedded devices in the overall decoding pipeline. In this scenario, an optimized CPU implementation of the HEVC decoder could handle the other video decoder modules while offloading the execution of HEVC in-loop filters to the GPU.



(a) Class S.

(b) Class A.

(c) Class B.

(d) Class E.

Fig. 11. Average frame rate obtained with the proposed GPU HEVC in-loop decoding modules (DBF + SAO) on GK20a.

VI. Conclusion

To circumvent the added complexity of the decoding procedure defined by the HEVC standard, this paper proposed an efficient parallelization of the in-loop filtering modules (DBF and SAO) of the HEVC decoder by adopting low-power GPU accelerators of embedded systems. To attain such objective, the presented approaches extensively exploit both fine and coarse-grained parallelization techniques in an integrated perspective, by re-designing the execution pattern of the involved modules, while simultaneously coping with their inherent computational complexity targeting embedded GPUs. According to the presented experimental evaluation, the proposed parallel algorithms provide a significant reduction in the overall processing time (to less than 20 ms for Ultra HD 4K intra frames, i.e. 50 fps) over a single-core implementation, thus contributing for a software-based HEVC decoder solution in nowadays embedded systems, like tablets and smartphones.

Acknowledgment

This work was supported by national funds through FCT (Fundação para a Ciência e a Tecnologia), under projects PTDC/EEI-ELC/3152/2012 and UID/CEC/50021/2013. Diego F. de Souza also acknowledges FCT for the Ph.D. scholarship SFRH/BD/76285/2011.

References

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

[2] 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, Dec 2012.

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

[4] C.-M. Fu, E. Alshina, A. Alshin, Y.-W. Huang, C.-Y. Chen, C.-Y. Tsai, C.-W. Hsu, S.-M. Lei, J.-H. Park, and W.-J. Han, "Sample adaptive offset in the HEVC standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1755–1764, Dec 2012.

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

[6] D. F. de Souza, N. Roma, and L. Sousa, "Cooperative CPU+GPU deblocking filter parallelization for high performance HEVC video codecs," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, May 2014, pp. 4993–4997.

[7] ——, "OpenCL parallelization of the HEVC de-quantization and inverse transform for heterogeneous platforms," in *Signal Processing Conference (EUSIPCO), 2014 Proceedings of the 22nd European*, Sept 2014, pp. 755–759.

[8] 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, Dec 2012.

[9] G. Cebrián-Márquez, J. L. Hernández-Losada, J. L. Martínez, P. Cuenca, M. Tang, and J. Wen, "Accelerating HEVC using heterogeneous platforms," *The Journal of Supercomputing*, pp. 1–16, 2014.

[10] S. Radicke, J.-U. Hahn, Q. Wang, and C. Grecos, "Bi-predictive motion estimation for HEVC on a graphics processing unit (GPU)," *Consumer Electronics, IEEE Transactions on*, vol. 60, no. 4, pp. 728–736, Nov 2014.

[11] S. Momcilovic, A. Ilic, N. Roma, and L. Sousa, "Dynamic load balancing for real-time video encoding on heterogeneous CPU+GPU systems," *Multimedia, IEEE Transactions on*, vol. 16, no. 1, pp. 108–121, Jan. 2014.

[12] C. C. Chi, M. Alvarez-Mesa, B. Bross, B. Juurlink, and T. Schierl, "SIMD acceleration for HEVC decoding," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[13] F. Rediess, R. Conceição, B. Zatt, M. Porto, and L. Agostini, "Sample adaptive offset filter hardware design for HEVC encoder," in *Visual Communications and Image Processing Conference, 2014 IEEE*, Dec 2014, pp. 299–302.

[14] J. Zhu, D. Zhou, G. He, and S. Goto, "A combined SAO and de-blocking filter architecture for HEVC video decoder," in *Image Processing (ICIP), 2013 20th IEEE International Conference on*, Sept 2013, pp. 1967–1971.

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

[16] L. Haglund, "The SVT high definition multi format test set," Sveriges Television AB (SVT), Sweden, Tech. Rep., 2006. [Online]. Available: ftp://vqeg.its.bldrdoc.gov/HDTV/SVT_MultiFormat/2160p50_CgrLevels_Master_SVTdec05_/

[17] JCT-VC. (2014) Subversion repository for the HEVC test model version HM 15.0. [Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-15.0/