J Real-Time Image Proc (2016) 12:531–547
DOI 10.1007/s11554-015-0519-1

SPECIAL ISSUE PAPER

GPU-assisted HEVC intra decoder

Diego F. de Souza · Aleksandar Ilic · Nuno Roma · Leonel Sousa

Received: 22 January 2015 / Accepted: 24 June 2015 / Published online: 9 July 2015
© Springer-Verlag Berlin Heidelberg 2015

Abstract The added encoding efficiency and visual quality offered by the High Efficiency Video Coding (HEVC) standard is attained at the cost of a significant computational complexity of both the encoder and the decoder. In particular, the considerable amount of intra prediction modes that are now considered by this standard, together with the increased complexity of the adopted block coding tree structures using a larger diversity of transforms imposes demanding computational efforts that can hardly be satisfied by current general-purpose processors to attain hard real-time requirements. Furthermore, the strict data dependencies that are imposed make parallelization a difficult and hardly efficient option with conventional approaches. To circumvent this adversity, this paper exploits Graphics Processing Units (GPUs) to accelerate the intra decoding procedure in HEVC, encompassing the most demanding modules of the decoder (i.e., de-quantization, inverse transform, intra prediction, deblocking filter, and sample adaptive offset). The presented approaches comprehensively exploit both coarse and fine-grained parallelization opportunities in an integrated perspective by redesigning the execution pattern of the involved modules, while simultaneously coping with their inherent computational complexity and strict data dependencies. As a result, the proposed parallelization, which is fully compliant with the HEVC standard, has shown to be a remarkable viable approach, being capable of satisfying hard real-time requirements by processing each Ultra HD 4 K intra frame in less than 25 ms (about 40 fps).

Keywords HEVC · Intra decoder · GPU · Parallel processing

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 UID/CEC/50021/2013.

D. F. de Souza (✉) · A. Ilic · N. Roma · L. Sousa
INESC-ID, IST, Universidade de Lisboa, Rua Alves Redol 9, 1000-029 Lisbon, Portugal
e-mail: diego.souza@inesc-id.pt

A. Ilic
e-mail: aleksandar.ilic@inesc-id.pt

N. Roma
e-mail: nuno.roma@inesc-id.pt

L. Sousa
e-mail: leonel.sousa@inesc-id.pt

1 Introduction

In the past few years, the HEVC emerged as the new video compression standard [1]. When compared with the previous standards (e.g., H.264/MPEG-4 AVC), the HEVC has shown to provide equivalent subjective visual quality, while achieving bit rate reductions as high as 50 %. As a result of its high compression efficiency, HEVC is nowadays exploited even in areas beyond video coding (e.g., image compression [2]). However, such coding efficiency comes at the cost of a substantial increase in the computational complexity of both the encoder and the decoder [3].

These increased computational requirements are particularly observed at the level of the intra CODEC, which allow achieving a bitrate reduction of up to 36 % over the H.264/MPEG-4 AVC, while providing similar objective quality [4]. Such bitrate reduction and improved visual quality arise from a tightly coupled functionality of the several modules with different computational complexities and processing execution paradigms.

A generic block diagram of the HEVC decoder is presented in Fig. 1. The decoding procedure starts by applying

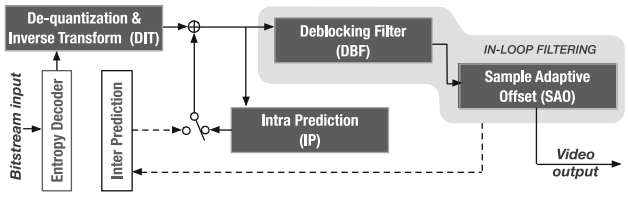


Fig. 1 Block diagram of the HEVC decoder

entropy decoding to the bitstream input, in order to obtain the coefficients data, as well as all other information needed to decompress the video sequence. The coefficients data are then de-quantized and inverse transformed by the De-quantization and Inverse Transform (DIT) module, in order to obtain the residual data. The reconstructed image blocks are produced by adding the residual data with the predicted image blocks from either inter or intra prediction modules. In intra decoding, which is the main focus of this paper, the predicted blocks are computed in the Intra Prediction (IP) module. Hence, the HEVC IP and DIT modules allow achieving higher compression rates by extensively exploiting the spatial correlation within a frame. Then, to attenuate eventual blocking artifacts introduced by the block-based prediction and transform coding, the Deblocking Filter (DBF) is applied at the boundaries of the reconstructed blocks. Finally, the mean sample distortion is reduced in the Sample Adaptive Offset (SAO) module, where the final video output is produced. Besides this functional diversity across the different modules at the HEVC intra decoder, a high computational complexity can also be individually observed at the level of each module:

- the DIT, responsible for recovering the pixel residues by dequantizing the entropy decoded coefficients and by applying the inverse integer transforms to such coefficients, reverting them into the pixel domain by considering up to four different block sizes and five diverse integer discrete transforms [5];
- the IP module, responsible for implementing the spatial prediction mechanism in the decoding side of the CODEC, by considering the already reconstructed neighboring pixels for the prediction of the current block, which is subsequently added with the residual data computed by the DIT [4]. Several block sizes have to be considered, as well as 35 different IP modes that are specified by the HEVC standard [4]. However, since each prediction mode takes into account the reconstructed pixel samples of the neighboring blocks, it must respect strict data dependencies imposed by the HEVC standard. These dependencies do not only occur between adjacent data blocks within the IP module, but also across the DIT and IP modules;
- the in-loop filters (including the DBF and SAO filter) are responsible for reducing the block artifacts and for applying a gradient-based filtering, thus promoting the resulting visual quality and encoding efficiency. To reduce the block artifacts on the reconstructed blocks, the DBF is applied on an 8×8 sample grid of the frame, where the vertical edges are processed first, and followed by the processing of the horizontal ones [6]. The sample distortion obtained by the previous modules is subsequently reduced by the SAO module according to a set of parameters selected by the encoder [7]. However, although the DBF and the SAO modules offer a certain amount of parallelization opportunities, both modules must be sequentially executed after DIT and IP.

As a result, reducing the HEVC decoding time via parallelization of the decoder modules has represented an important research topic in modern video coding.This is particularly significant for the above-referred HEVC intra decoding modules, which are responsible for about 40–49 % of the total intra decoding time, even when an optimized CPU HEVC decoder is considered [3], while the entropy decoder module is in charge for the remaining decoding time. Nevertheless, the importance of conceiving efficient Intra CODECs is already evidenced in several studies [2, 8, 9], where the Intra-only video encoders are used even beyond the original video coding purposes. All Intra encoders are also employed in state-of-the-art DSLR cameras, e.g., the Panasonic DMC-GH3 and DMC-GH4 [10], in order to achieve high frame rate videos and prevent loss of image quality because editing does not require re-encoding. Hence, several scientific works [11, 12] have already tackled hardware implementations of HEVC Intra encoders on Field-Programmable Gate Array (FPGA).

However, there is still a limited number of studies targeting efficient parallelizations of HEVC decoding modules on general-purpose architectures [13]. In the same extent, efficient Intra decoder solutions for modern data-parallel accelerators [such as Graphics Processing Unit (GPU)] are yet to be proposed, which is the main focus of this paper. In fact, modern GPU have evolved into programmable and powerful parallel accelerators, being already able to deliver a performance level that greatly exceeds the capabilities of multi-core CPUs [14]. However, this performance is mostly attainable for compute-intensive applications, with significant amount of data parallelism and regular memory access patterns. As a result, fully exploiting the GPU capabilities for a set of diverse and computationally complex HEVC intra decoding modules is far from being a trivial task. In particular, the HEVC IP decoding module is one of the hardest

modules to be accelerated by the GPU, due to its strict data dependencies and highly irregular memory accesses.

In order to tackle this problem, a set of efficient and fully compliant parallel algorithms of the HEVC inverse transform, de-quantization, intra prediction, deblocking filter, and sample adaptive offset modules suitable for GPU are proposed.The presented concretization of the algorithms efficiently exploits the fine-grain parallelism of the GPU architectures, by re-designing the execution pattern of these intra decoding modules while simultaneously coping with their inherent computational complexity and strict data dependencies. Hence, the proposed algorithms allow achieving a significant reduction in the overall processing time, while preserving the full functionality of the HEVC Intra decoding modules, as specified by the standard. To further reduce the processing time, this paper also investigates the possibility of overlapping memory transfers, between CPU and GPU, and GPU kernel executions by taking into account the required sequential processing order of the HEVC Intra decoder modules. As a result, the presented approach requires no more than 25 ms for decoding Ultra HD 4 K (3840×2160) intra frames on state-of-the-art GPU devices, which corresponds to a minimum frame rate of 40 frames per second (fps).

The remaining of this paper is organized as follows. Section 2 provides a brief overview on the basic functional principles behind the HEVC intra decoding modules, while Sect. 3 revises the state-of-the-art approaches for HEVC parallel decoding. The proposed algorithms and consequent parallel implementations are presented in Sect. 4. The obtained experimental results and the derived conclusions are presented in Sects. 5 and 6, respectively.

2 HEVC intra decoding

In the HEVC standard, each picture is partitioned in $L \times L$ pixel blocks called Coding Tree Unit (CTU), where L is selected by the encoder $(L \in \{16, 32, 64\})$. The CTU is grouped in slices or tiles of the frame and decoded in raster scan order. Each CTU is independently split in smaller blocks called Coding Unit (CU) using a quadtree structure. The dimension of the CU varies between a maximum size of 64×64 pixels to a minimum size of 8×8 pixels. Additionally, each CU is further divided in the Prediction Unit (PU) and Transform Unit (TU), which correspond to the predicted and the residual block data, respectively [1]. Inside each CTU, the CU is decoded by following a z-scan order, as well as the pPU and the TU within each CU.

The same frame partitioning (CTU, CU, PU, and TU) is applied to each component, i.e., luma and both chromas. 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.

2.1 De-quantization and inverse transform

As referred before, inside each CU, the TU is split in smaller blocks (4×4, 8×8, 16×16, or 32×32) according to a quadtree structure. These blocks are named as Transform Block (TB) and each TU is composed by luma and chroma TB.

According to the HEVC standard (see Fig. 2), the DIT is directly applied on the TB coefficients obtained from the entropy decoding, in order to obtain the TB Residual Data. For each TB, the overall procedure is controlled by three flags: the Transquant Bypass Flag (TBF), the Coded Block Flag (CBF), and the Transform Skip Flag (TSF). Whenever the TBF is set, DIT is bypassed, which means that the residual data directly correspond to the TB coefficients. In fact, the TBF is used by the encoder to achieve lossless CTU (or frame) reconstruction, where DIT, DBF, and SAO are bypassed. The CBF indicates when there is some residual data in the TB. When CBF is set, the de-quantization must be applied to the TB coefficients. The *De-quantization* module also implements the HEVC inverse scaling, which depends on the Quantization Parameter (QP) and on the adopted TB size. Finally, the TSF signals the decoder to skip the inverse transform and to apply the TSF Scaling (TSF $=$ 1). When TSF is not set, the 2D Inverse Transform is performed on the de-quantized data, implemented with two 1D decompositions (i.e., 1D column and 1D row inverse transforms). Each decomposition is followed by a specific scaling procedure to preserve the normalization property in the transform domain.

Each 1D decomposition is performed on a specific *transform coefficient array*, which is chosen in respect to the adopted TB size and prediction mode. In the HEVC standard, only integer transform coefficient arrays are specified, where the 4×4 to 32×32 transform coefficient arrays are based on the integer discrete cosine or sine transforms [5].

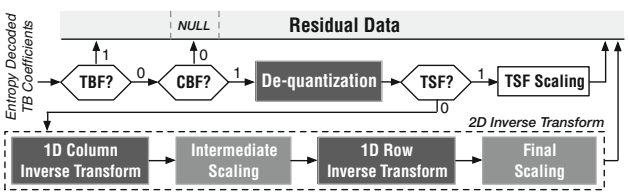


Fig. 2 The HEVC residual data decompressing flowchart

2.2 Intra prediction

When a CU is encoded using intra prediction, the PU has the same size as the CU. The only exception occurs for the smallest CU size in the bitstream, where the PU can be further partitioned in four blocks (e.g., four 4×4 PU for an 8×8 CU), which are processed in z-scan order.

Similarly to the TU, the PU is further divided in luma and chroma Prediction Block (PB), where the intra prediction is applied to each PB. The final reconstructed block is obtained by adding the prediction data from the PB and the residual data from DIT. The PB are processed in a strictly defined order, as specified by the HEVC standard [4]. The dashed-line in Fig. 3 represents the data dependencies arisen from the *z-scan processing order* for different PU. These *intra dependencies* occur because the previously reconstructed blocks (*reference samples*) are used as input for the next PB.

Figure 3 also presents an example of the required references samples for blocks A and B (see *intra dependencies*). Block A requires the reconstructed data from adjacent upper and left blocks, while block B can only be predicted after the reference samples from block A are computed. In general, for intra prediction of an $(N \times N)$ PB, $4N + 1$ reference samples are required from up and left adjacent blocks. However, depending on the relative position of the PB in the CTU, one or more reference sample sets may not be available (see block B in Fig. 3). In this case, the remaining samples are *extrapolated* to fill the whole set of $4N + 1$ samples, by repeating the value of the nearest available reference sample.

As soon as all the required $4N + 1$ reference samples are generated, the intra prediction of the current PB can start. For luma PB, a smoothing filtering is firstly applied to the reference samples, according to the PB size and prediction mode. As presented in Fig. 3, there are 35 different *Intra modes*: (i) mode 0 refers to planar intra prediction; (ii) mode 1 to DC prediction; and (iii) modes 2 to 34 to angular predictions [4]. In angular prediction, the interpolation is applied on the reference samples according to the specified direction [4] (e.g., #18 in Fig. 3).

Accordingly, each PB is predicted using one of those intra modes, which can differ for luma and chroma PB. When the TB is smaller than the PB, the intra prediction is performed at the TB level. In this case, each sub-block in the PB is predicted in z-scan order, where the size of each sub-block is defined by the TB. For intra prediction, TU can not be larger than PU [4].

The HEVC standard also defines the I_PCM mode to reconstruct extremely rare blocks at the CU level [1]. In this case, the entropy decoder, IP, and DIT are bypassed and the reconstructed block is obtained directly from the bitstream.

2.3 Deblocking filter

The HEVC deblocking filter is applied to the boundaries of the PU and TU, which rely on 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, the chroma samples are only filtered when the BS value is 2 [6].

For 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. 4). 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. 4). In each side of the boundary, only up to four neighboring samples have to be considered and up to three may be modified. For example, for luma component, the *strong* filtering is applied on three pixels in each side of the boundary, while in the *normal* filtering at most two pixels can be filtered on each side of the boundary, depending on a set of DBF conditions (see "*Strong Filtering*" and "*Normal Filtering*" in Fig. 4). In contrast, for chroma

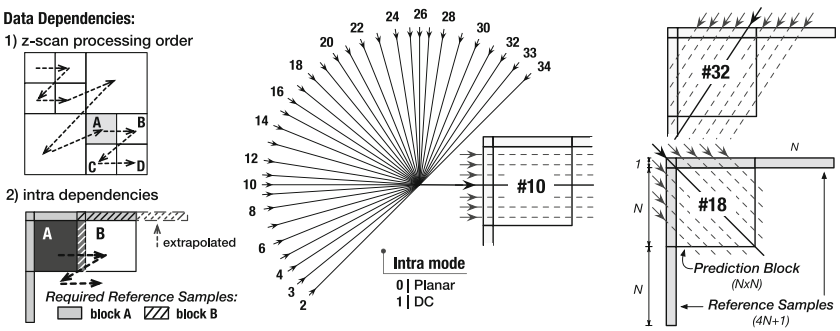


Fig. 3 Intra prediction dependencies and modes

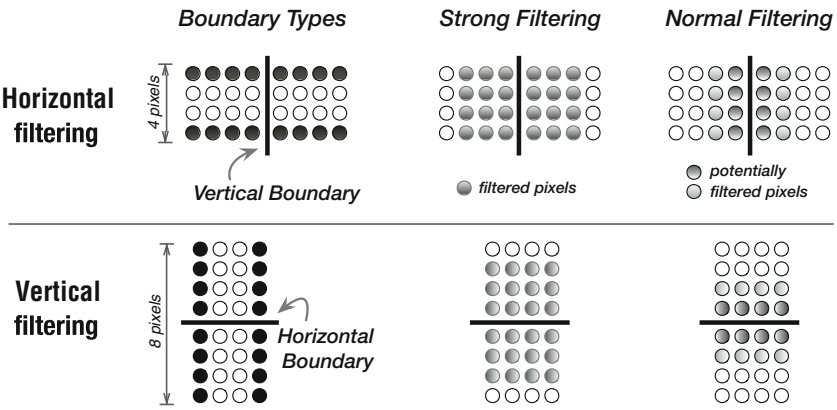


Fig. 4 Deblocking filter boundary and filtering types

samples, the *normal* filtering is only applied on a single pixel in each side of the boundary.

It is worth noting that in the DBF overall process (as defined by the standard), all vertical edges from the frame are filtered before the horizontal edges [15]. However, luma or chroma samples are not modified if one of the following conditions is true: (i) the samples were predicted as I_PCM and the *pcm_loop_filter_disabled_flag* is set; and (ii) the samples belong to a CTU where TBF is set (lossless mode).

2.4 Sample adaptive offset

The deblocked samples are subsequently modified in the SAO module by adding an offset value according to a set of SAO parameters, namely, *Type*, *Offset Values*, and *Band Position/Edge Class*. These SAO parameters are encoded in the bitstream for each CTU and can have different values for luma and both chroma components, even for the same CTU [7]. The *SAO Type* parameter signals the decoder which SAO filtering should be applied (none, band offset, or edge offset).

In the band offset mode, the full amplitude of the pixel range is divided by 32 to define a set of *bands*. From this set, only four consecutive bands are considered for SAO filtering, according to the information stored in the bitstream (i.e., the *SAO Band Position* parameter). For each specified band, a single offset value is provided in the respective *SAO Offset Value* parameter. Then, all samples whose values belong to these four bands have to be modified, such that the deblocked sample value is added with the corresponding *SAO Offset Value*.

In the edge offset mode, the CTU samples are classified into four categories, according to the gradient direction, which is specified in the *SAO Edge Class* parameter. Figure 5 depicts all four possible gradient directions and allowed SAO categories. Similarly to the band offset mode, the offset value for 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 arrow in Fig. 5). Hence, whenever a pixel is classified in one of these categories, its deblocked sample is added to the corresponding *SAO Offset Value*. In contrast, the SAO is not applied if the samples can not be classified in any of these categories.

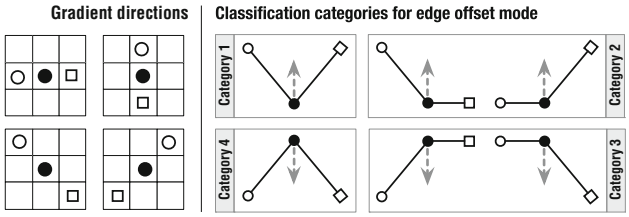


Fig. 5 SAO gradient directions and classification categories

As specified by the HEVC standard, the whole SAO process is bypassed for the lossless mode (TBF = 1) or when the samples are from an I_PCM predicted PU and *pcm_loop_filter_disabled_flag* is set.

3 Related work

In the past years, several video encoding and decoding modules have been implemented in GPU devices. Usually, most of the GPU video coding implementations deal with motion estimation schemes with relaxed dependencies, as proposed in [16, 17] for HEVC and in [18] for H.264/MPEG-4 AVC. However, efficient GPU approaches for HEVC decoding are yet to be derived, especially for intra decoding. In fact, at the decoder side, parallel implementations often pose difficult challenges, 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 highly restrictive processing capabilities.

In the current state-of-the-art approaches, only rare attempts have tackled the efficient HEVC decoder implementations, but mainly for multi-core CPUs [13] and FPGA [19].

In [13], Chi et al. exploit Single Instruction, Multiple Data (SIMD) parallelization models at the level of HEVC decoder modules by specifically focusing on modern multi-core CPU architectures. To achieve better performance, the authors in [13] divide the load among CPU cores by relying on an alternative method based on the HEVC Wavefront Parallel Processing (WPP). In contrast, the GPU parallel implementations that are herein proposed are fully compliant with both HEVC parallelization techniques, namely *Tiles* and WPP [1].

Regarding real-time HEVC decoding, a FPGA implementation was developed in [19], which can decode Full HD video sequences at 30 frames per second (with an operating frequency of 110 MHz). However, such implementations represent different compromises in terms of energy efficiency, resources utilization, and programmability, preventing a fair comparison with high-performance computing platforms, like GPU.

Nevertheless, although there are no existing approaches that tackle GPU implementations for the entire HEVC decoder, several research works consider the GPU parallelization of individual decoding modules, namely, deblocking filtering [20] and inverse transform [21].

In what concerns the transform module, several algorithms were already proposed to alleviate the complexity at the encoder side, like the zero block detection [22], which was based on [23] to eliminate redundant computations. At the decoder side, the state-of-the-art GPU implementation of DIT [21] relies on a coarse-grain parallelization of DIT stages, which is more suitable for embedded GPU devices. However, the GPU DIT algorithm that is herein proposed advances these contributions by providing a fine-grain parallelization of the DIT computational load, thus allowing it to achieve a higher performance across a range of different GPU devices.

As previously referred, the HEVC IP is one of the most difficult modules to be efficiently parallelized on the GPU because of the reconstructed neighboring block dependencies and subsequently limited parallelization potential. In fact, these difficulties can also be observed in the multi-core CPU IP implementation proposed in [13], where this HEVC module achieves one of the lowest speed ups among all others modules.

To the best of the authors' knowledge, the herein proposed IP and SAO parallel modules represent one of the first approaches to handle these HEVC modules in the GPU. Furthermore, a novel approach is also proposed in this paper for GPU-accelerated real-time and fully compliant HEVC intra decoding across a set of modules, namely, DIT, IP, DBF, and SAO.

4 Proposed parallel algorithms

The proposed implementation of the considered HEVC intra decoding modules (i.e., DIT, IP, DBF, and SAO) is designed to efficiently exploit the capabilities of highly parallel GPU architectures. They leverage the fine-grain parallelism of these computationally complex and highly data-dependent 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 Block (ThB). To achieve high performance, the proposed algorithms maximize the number of active warps, while ensuring that all threads in a warp perform the same operation from the GPU code (kernel). Also, the data accesses are carefully managed, in order to efficiently use the complex GPU memory hierarchy, i.e., global, cache, shared, constant, and texture memory [24].

4.1 GPU de-quantization and inverse transform

In contrast to [21], the herein proposed parallel HEVC DIT algorithm relies on a single GPU kernel for all TB sizes (i.e., from 4×4 to 32×32). The general layout of the proposed parallel DIT is presented in Fig. 6 for the case of 32×32 TB.

In the proposed GPU DIT, a single ThB contains 8 warps (W*i*), which perform the DIT computations on independent TB parts (e.g., eight 32×4 TB sub-blocks in Fig. 6). The overall procedure starts by asynchronously reading the *entropy decoded TB coefficients* from the GPU global memory. Here, all 32 parallel threads in a warp fetch four rows of the 32×4 TB part. The parallel DIT output (*Residual Data*) is produced by the different warps, after executing the *Data Paths* 1, 2, or 3.

The considered *Data Path* is selected by the DIT flags, i.e., TBF, CBF, and TSF (see Sect. 2). In the proposed implementation, the TBF and CBF of a single TB are merged in one new flag, which is named Bypass Flag (BF) [21]. Furthermore, to reduce the communication overheads, a specific 8-bit *HEVC DIT Control Data* structure is designed, such that all the 27 BF and TSF combinations are integrated. In brief, for each TB, there are 3 possible flag combinations that can occur, i.e., (BF, TSF) $\in \{(1,*);(0,1);(0,0)\}$, giving a total of 27 ($3^3$) combinations for one luma and two chroma TB. This information is binary encoded with five bits and stored in bit positions 2–6 of the *HEVC DIT Control Data* structure. Furthermore, the bit positions 0 and 1 of this structure are reserved for

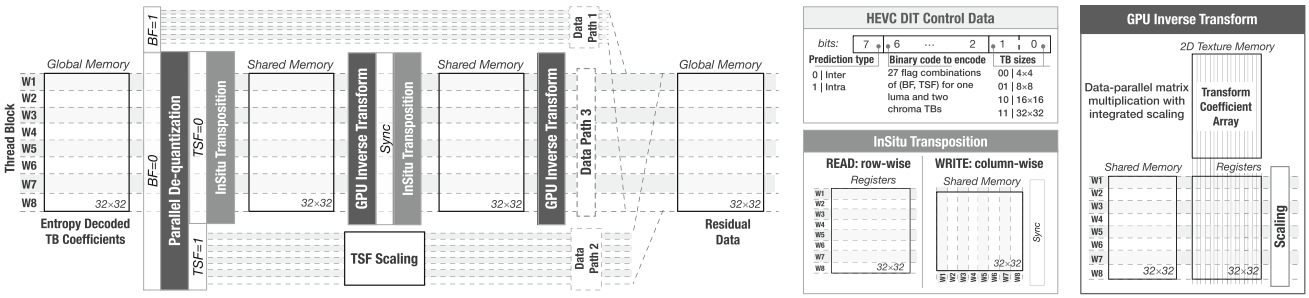


Fig. 6 Overall de-quantization and 2D inverse transform implementation in the GPU for one 32 × 32 luma TB

the TB size information (see Fig. 6). The remaining bit position (7) is used to designate the prediction type (i.e., Intra or Inter).

These *HEVC DIT Control Data* are packed for each 4 × 4 block of the frame. These data are used by the warps to extract the BF and the TSF values during the GPU kernel execution (i.e., to select the *Data Path* for each 32 × 4 TB sub-block in Fig. 6). Whenever the BF is set (*Data Path 1*), the fetched TB coefficients are directly forwarded to the residual data (i.e., TBF = 1 or CBF = 0). This execution path is also used to integrate the I_PCM mode, where the I_PCM data are stored as the TB coefficients on the CPU side. When the BF is unset, the *Parallel De-quantization* is performed before the TSF is evaluated (see Fig. 6). In the *Parallel De-quantization*, each thread in a warp independently de-quantizes one TB coefficient.

If the TSF is set, the warps asynchronously perform the *TSF Scaling* in *Data Path* 2. Otherwise (when TSF = 0), *Data Path* 3 is selected, where the parallel 2D GPU inverse transform is applied. In the proposed approach, the HEVC 1D Column and 1D Row Inverse Transforms are performed by the same *GPU Inverse Transform* procedure. This is achieved by applying the *InSitu Transposition*, where each warp re-arranges the data to fit the correct form. As presented in Fig. 6, the first *InSitu Transposition* is applied to the de-quantized TB coefficients (in the GPU registers). Here, each warp re-arranges the coefficients to a column-wise representation in the shared memory. To ensure the correctness of the HEVC 1D Column Inverse Transform, all warps must finish the *InSitu Transposition* (*Sync*) before the *GPU Inverse Transform*.

In the *GPU Inverse Transform* implementation, the shared memory is always read in a row-wise pattern by each warp. Then, the data-parallel matrix multiplication is performed over the read data and the selected *Transform Coefficient Array* (stored in the GPU texture memory). Afterward, the HEVC Intermediate *Scaling* is independently applied on the obtained results by each warp.

To proceed with the computation of the HEVC 1D Row Inverse Transform, all warps must finish the previous *GPU Inverse Transform* (*Sync*) and apply the second *InSitu Transposition*. The *Sync* point guarantees that the first 1D transform is finished in all warps before the results are written back to the shared memory. Then, the second *GPU Inverse Transform* is applied with the integrated HEVC Final *Scaling*. Finally, each warp finishes, by asynchronously writing the produced residual data in the GPU global memory.

Figure 7 presents an example of distinct warp assignments for different TB sizes (i.e., 4 × 4, 8 × 8, 16 × 16, and 32 × 32). When the GPU kernels start, all eight warps within a ThB start by obtaining the TB sizes stored in the *HEVC DIT Control Data* (for each 4 × 4 block of the 32 × 4 frame segment). According to the obtained TB size, the warps are assigned to different portions of the TB. Figure 7 also shows an example of how 4 ThB can be assigned for a 32 × 32 partitioned TB.

It is worth emphasizing that the proposed parallel DIT algorithm extensively exploits an efficient utilization of the GPU memory hierarchy, by organizing the data accesses as follows: (i) the *HEVC DIT Control Data*, the TB coefficients, and the residual data are stored in the global memory; (ii) the transform coefficient arrays are kept in the read-only 2D texture memory; (iii) the low latency shared memory is used to store the intermediate data for the 2D inverse transform; and (iv) all remaining data (such as frame size, QP, and scaling factors) are stored in the constant memory and broadcasted to each warp in one single memory transaction.

4.2 GPU intra prediction

Due to the strict dependencies between the reconstructed blocks, the proposed GPU parallel IP algorithm adheres to the wavefront execution paradigm. As it is shown in Fig. 8 (*Frame-level and Thread Block Processing*), a single ThB composed of eight warps is assigned to process 64 rows of luma pixels (32 for chroma), in an 64 × 64 CTU row. Hence, each warp computes 8 consecutive rows of luma pixels (or 4 for chroma).

Correspondingly, when processing a single N × N luma block (N ≥ 8), each warp is responsible for the prediction

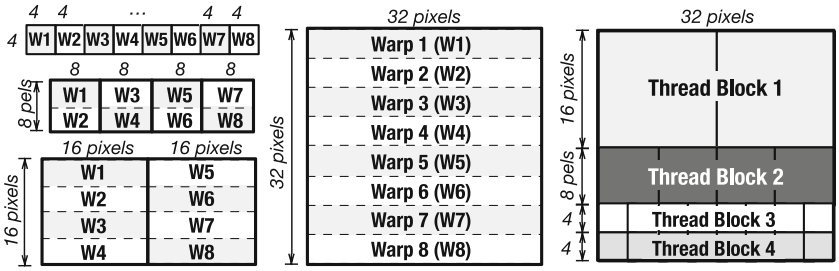


Fig. 7 Example of IT warp and ThB assignments

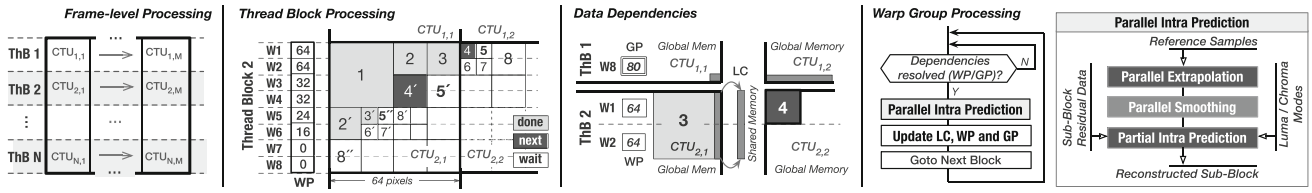


Fig. 8 GPU intra prediction warps assignment and framework with a wavefront approach

and reconstruction of a corresponding N × 8 sub-block. For the chroma component, the same warp operates on a corresponding (N/2) × 4 sub-block. Hence, the adoption of this processing granularity allows keeping the same number of active warps when processing luma and chroma components (4 × 4 is the minimum chroma block size). In addition, this granularity is also suitable when processing a partitioned 8 × 8 block, since four 4 × 4 blocks are sequentially processed in a z-scan order.

Within each warp, the block size (N) is determined from the data available in the global memory (i.e., bits 0 and 1 of the *HEVC IT Control Data*). This design option arises from the fact that the IP is performed after the DIT, where PB is processed at the TB level. Hence, when a single warp completes the assigned sub-block, it proceeds to the next N × 8 luma sub-block of the assigned 8 pixel row. However, this IP procedure can only be performed if all the data dependencies are satisfied (i.e., if the neighboring blocks are already processed). An example of the GPU IP execution in a ThB is presented in Fig. 8 (*Thread Block Processing*), where the numbers assigned to the blocks represent their processing order. For example, only after the warps W1–W4 finish processing block 1, blocks 2 and 2' can be processed in parallel.

To keep track of the reconstructed neighboring blocks, the proposed IP algorithm relies on a specifically developed warp-level data structure for each ThB, i.e., Warp Positioning (WP). The WP is updated in the shared memory, where each warp stores the frame x-axis position of its recently finished sub-block. In Fig. 8 (*Thread Block Processing*), the presented array of WP values reflects the position of each warp before blocks 4 and 4′ are processed. As it can be seen, W1 and W2 have WP = 64, since they were responsible for processing a 32 × 8 sub-block of block 1 and 16 × 8 sub-blocks of blocks 2 and 3 ($32 + 2 \cdot 16 = 64$).

The currently active warp determines the required neighboring blocks by accessing a dependency map in the GPU constant memory. This map contains all needed dependency information to process all blocks in a CTU. According to this information, the currently active warp queries the WP of the warps responsible for producing the neighboring blocks, i.e., to check if all data dependencies are satisfied, such that the currently active warp can proceed with its execution. In the case of the 8 × 8 block 4 from ThB 2, presented in Fig. 8 (*Data Dependencies*), W1 checks the status of block 3 from ThB 2, as well as the status of the reference samples produced in ThB 1. Hence, the processing of block 4 can only start if WP of both W1 and W2 from ThB 2 are set to 64 and WP of W8 from ThB 1 is set to 80 (see Sect. 2).

Since the warps from one ThB can not access the shared memory from the other ThB, the WP of the last warp from each ThB is stored in the GPU global memory, i.e., Global Positioning (GP). Thus, the data dependencies between ThB (64 × 64 CTU) are checked by assessing the GP of the previous ThB. In the example depicted in Fig. 8 (*Data Dependencies*), W1 inspects the GP value of ThB 1 when checking the dependencies between ThB.

As presented in the flow-graph of Fig. 8 (*Warp Group Processing*), each warp will proceed with the *Parallel Intra Prediction* only when all data dependencies are satisfied, i.e., the $4N + 1$ reference samples are produced. The *Parallel IP* starts by fetching the vertical reference samples, on the block left side. In order to improve the memory access, these samples are allocated as a Last Column (LC) array in the shared memory, where the last reconstructed pixel column in a ThB is stored. Thus, each warp is

responsible for updating the LC (for both luma and chroma components) with its reconstructed pixels. By doing so, the other warps can access this information with less and faster memory transactions. Nevertheless, the upper reference samples are brought from the global memory, since these data are already coalesced and can take advantage of the GPU L2 cache.

After fetching the *Reference Samples*, the *Parallel Extrapolation* procedure is performed to fill the whole $4N+1$ reference sample set. Here, the nearest available reference sample is broadcasted to all threads in a warp, where each thread simultaneously copies this value to a single unavailable reference sample position. For luma blocks, the *Parallel Smoothing* is then performed, where each thread is responsible for applying the HEVC smooth filtering to a single reference sample. To avoid the need for intrinsic synchronization points and excessive register consumption, each warp has a separate array of reference samples stored in the shared memory. Then, the *Partial Intra Prediction* is performed on each N $\times$ 8 sub-block, to produce the corresponding reconstructed sub-block.

The *Partial Intra Prediction* is performed in two steps: *Mode Prediction* and *Reconstruction*. In the *Mode Prediction* step, the predicted sub-block is produced by performing one of the three possible modes (Angular, DC, or Planar). The mode is selected according to the corresponding luma or chroma prediction mode, which is stored in a 2-byte word for each 4 $\times$ 4 block. In each byte, 6 bits are needed to encode 36 intra modes (including I_PCM), while the remaining bits are used to store the CTU TBF flag. The TBF flag is added to this structure, since both the considered intra modes and the TBF flag are required for performing the DBF and SAO modules. As a result, this 2-byte structure is also re-used in the proposed DBF and SAO parallel algorithms.

As it is presented in Fig. 9 for a 32 $\times$ 8 sub-block, the selected mode is simultaneously performed, where each thread in a warp is responsible for one pixel. In the *Angular Mode*, each thread interpolates the reference samples according to the given direction. In the *Planar Mode*, each thread applies a bilinear model according to the relative spatial position of the pixel to be predicted and the reference samples. In the *DC Mode*, all threads cooperatively compute the reference samples average to produce the predicted pixels. In the second step, the *Reconstruction* is performed in parallel, by adding the predicted sub-block with the corresponding residual sub-block. The obtained reconstructed sub-block is stored in the GPU global memory.

Finally, after the *Partial Intra Prediction* is finished, each warp updates its LC, WP, and GP values before proceeding to the next block. Then, the overall intra GPU decoding procedure is repeated until all warps reach the end of the frame. As mentioned before, in the I_PCM mode, the residual data are marked as the reconstructed sub-block and the *Mode Prediction* is bypassed.

4.3 GPU deblocking filter

As referred in Sect. 2, the DBF module considers up to four samples within a 4 $\times$ 8 (or 8 $\times$ 4) pixel region, in order to filter up to three samples in each side of the boundary. According to the HEVC standard [1], this procedure is firstly applied on all vertical edges in a frame, followed by the horizontal ones.

Although this processing order fits to the CPU architectures, it might not deliver a suitable degree of fine-grained parallelism when exploiting GPU hardware. 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 it does not allow efficient utilization of the GPU memory hierarchy. For example, upon the vertical edges are filtered, the data need to be stored in the global memory, and again retrieved 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

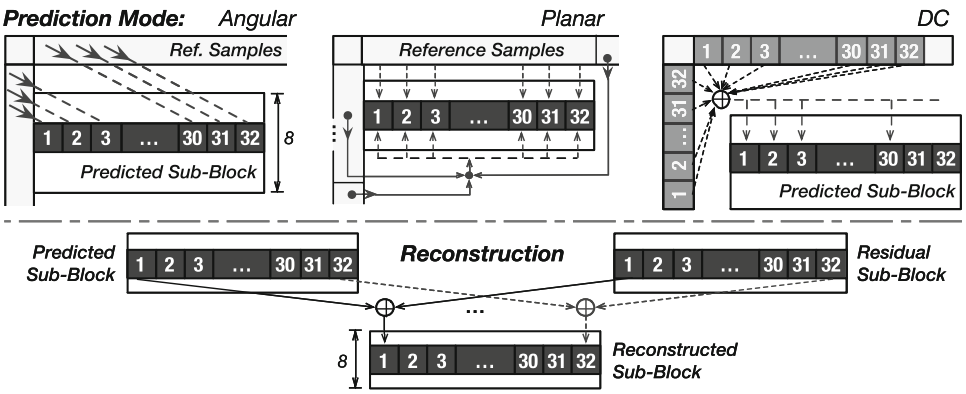


Fig. 9 Proposed GPU Partial Intra Prediction algorithms for the *Mode Prediction* and *Reconstruction* steps

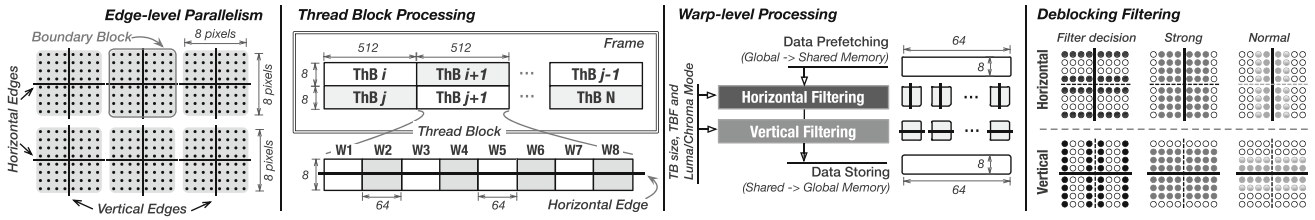


Fig. 10 Thread blocks/warps assignment and the functionality of the proposed GPU Deblocking Filter

transactions to fetch a single portion of horizontally filtered data.

However, the proposed DBF implementation provides a better opportunity for efficient GPU parallelization, by relying on a different paradigm for the execution of the DBF stages. As presented in Fig. 10 (see *Edge-level Parallelism*), when two consecutive horizontal 4 × 8 filtering regions are considered, one can identify several non-ovelapping blocks that can be filtered in parallel [20]. These 8 × 8 pixel blocks, herein referred to as Boundary Block (BB), allow performing both horizontal and vertical filtering on a small subset of locally stored and independent input data. Hence, all 8 × 8 BB in a frame can be simultaneously processed without any synchronization points and by efficiently using the GPU memory hierarchy.

As it is shown in Fig. 10 (*Thread Block Processing*), in the proposed DBF GPU algorithm, each ThB is assigned to perform the deblocking filter on a row of 64 BB (i.e., 512 × 8 luma pixels). Inside each individual ThB, eight warps are in charge of carrying out the deblocking filter in a row of 8 BB (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.

Figure 10 (*Warp-level Processing*) describes how the data-parallel DBF algorithm is performed at the level of a single warp. First, all threads in a warp simultaneously copy a 64 × 8 luma block from global memory to its own portion of shared memory space (*Data Prefetching*). Then, the deblocking filter is performed on this data in the shared memory, where two threads are assigned to perform the DBF on a single BB. In this procedure, two BB 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*). For both steps, the input data (i.e., TB sizes, TBF, and intra modes) are re-used from the HEVC DIT Control Data and IP mode. Finally, the data are stored back to the global memory (*Data Storing*).

In the proposed algorithm, the whole shared memory that is assigned to a warp is used for both luma and chroma components. Hence, a single 64 × 8 space is used in its entirety to perform the DBF for the luma component. On the other hand, this same amount of space is equally divided for chroma components, i.e., a 32 × 8 chroma U and 32 × 8 chroma V blocks are retrieved in parallel in the *Data Prefetching* phase, and the DBF procedure is simultaneously performed on both chroma components.

Furthermore, considering that a PU boundary always matches a TU boundary in the HEVC intra prediction, only TU boundaries have to be filtered. Hence, for each filtering process (*Horizontal* and *Vertical Filtering*), the TB size information from the HEVC DIT Control Data is used to determine the TU boundary according to the relative position of the corresponding pixels within a frame, i.e., their relative position must be a multiple of the TB size. In addition, since the BS value is always set to 2 for intra prediction, it does not need to be calculated. During the *Deblocking Filtering* procedure, the DBF decisions are verified according to a set of pixels from the first and the last row (or column) from both 4 × 8 (or 8 × 4) regions within the currently considered BB (see the dark-filed pixels in "*Filter decision*" part of Fig. 10). Here, whenever the *Strong* filter is selected, up to three pixels in each side of the boundary are filtered (and up to two for the *Normal* filtering).

For samples predicted with the I_PCM mode, the luma and chroma intra modes are considered to disable the DBF according to the *pcm_loop_filter_disabled_flag*, which is stored in the GPU constant memory. Furthermore, the DBF is bypassed for samples from a lossless-encoded CTU, where TBF is equal to one. In order to reduce the number of accesses to the global memory, the TBF and the intra modes are packed in the same byte word and decoded by the GPU with bitwise operations.

4.4 GPU sample adaptive offset

In the proposed parallel implementation of the SAO algorithm, each ThB (composed by eight warps) is responsible for performing the SAO procedure for 8 CTU in a single row, where each warp is assigned to one CTU, as depicted

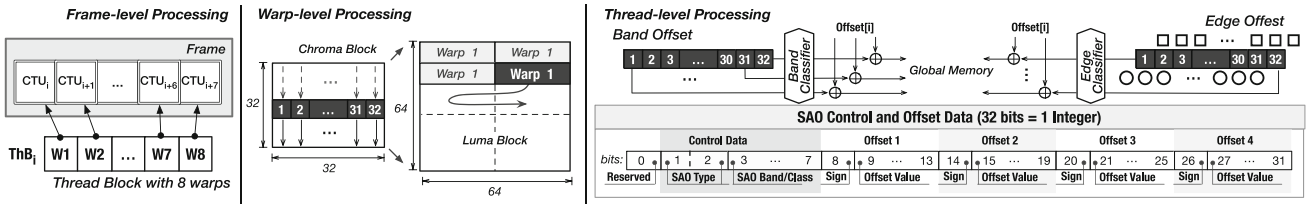


Fig. 11 Different processing levels in the proposed GPU Sample Adaptive Offset filtering implementation.

in Fig. 11 (*Frame-level Processing*). In this case, each warp carries out the SAO procedure for 32 pixels in parallel, i.e., one CTU line at a time. Hence, the 32 pixels of each line of the 32 × 32 chroma CTU are simultaneously processed, as shown in Fig. 11 (*Warp-level Processing*). On the other hand, when processing each row of each 64 × 64 luma block, the SAO filter is first performed on the left-most set of 32 pixels, and then on the right-most set of 32 pixels within the same row.

In order to handle the computational complexity of the SAO procedure and to efficiently use the GPU memory hierarchy, a specific data structure was defined and denoted as *SAO Control and Offset Data*, as depicted in Fig. 11. For each frame component, the proposed 4-byte data structure allows storing the maximum size of each SAO parameter, as specified by the HEVC standard. As it is shown in Fig. 11, the SAO parameters are divided in *SAO Control* and *Offset Data* fields, where the *SAO Control* field contains the information corresponding to the *SAO Type* and to the *SAO Band/Class*. Here, bits 1 and 2 are used to encode the three possible *SAO Types*, namely, 0: none, 1: band offset and 2: edge offset. In contrast to the *SAO Class* field, which can only have four possible values (gradient directions), the *SAO Band* field can contain up to 32 different values according to the HEVC standard, in order to represent the SAO band position. In accordance, the compliancy with the HEVC standard is ensured, by encoding the SAO class/band in a five bits field (i.e., bits 3 to 7). Finally, the remaining bits are used to encode the four *SAO Offsets* (bits 8 to 31), where each offset is represented with 6 bits, since their absolute values can be up to 31, according to the HEVC standard. As a result, for each CTU, the SAO parameters are accommodated in a 12-byte word (4-bytes word for each luma/chroma components).

Figure 11 provides a general overview of the data-parallel *Thread-level Processing* applied on each set of 32 pixels, in order to perform the computations from the *Band Offset* and *Edge Offset* SAO filters. For the *Band Offset* filtering, each thread (from 1 to 32) performs the following set of operations: (i) the data are firstly fetched from the global memory; (ii) the 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* filtering, 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.

Similarly to the adopted GPU DBF algorithm, the proposed SAO parallel approach also fetches the intra prediction mode corresponding to each 4 × 4 block in a CTU to avoid filtering pixel samples predicted with I_PCM mode when the *pcm_loop_filter_disabled_flag* is set. Moreover, the lossless mode (TBF = 1) is already integrated in the SAO parameters, by setting the *SAO Type* equal to zero.

4.5 Integration of the proposed GPU intra decoder

In order to ensure a fully compliant and real-time HEVC intra decoding, all previously proposed parallel algorithms for the different HEVC modules are closely integrated into a collaborative CPU+GPU decoding environment. Due to its highly sequential and irregular nature, the entropy decoder is the only HEVC module that is performed on the CPU, while all remaining HEVC modules are implemented on the GPU, by relying on the proposed parallelization approaches (see Fig. 12).

The collaborative CPU + GPU HEVC intra decoder starts by acquiring the bitstream portion that corresponds to the first frame, which is entropy decoded on the CPU. Then, the entropy decoded data are sent to the GPU and used as the input data for GPU intra decoding (see "*Input Data* (*CPU* → *GPU*)" in Fig. 12). Afterward, the herein proposed parallel modules are performed in the GPU, i.e., DIT, IP, DBF, and SAO. Once the frame is decompressed, it is sent to the CPU for visual playback (see "*Output Data* (*GPU* → *CPU*)" in Fig. 12). However, while the memory transfers and the GPU intra decoding kernels are performed for the first frame, the CPU continues its processing, i.e., entropy decoding of the bitstream portion that corresponds to the next frame. As depicted in Fig. 12, this pipelined procedure is applied for the decoding of all subsequent frames; while the GPU decodes the current frame (f), the CPU entropy decoder processes the next frame ($f + 1$).

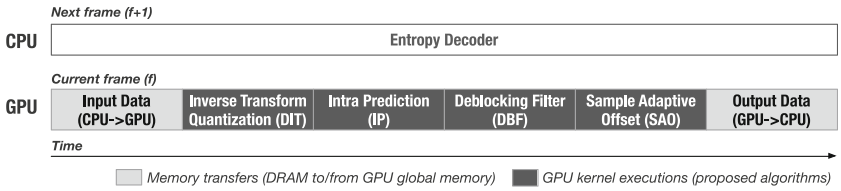
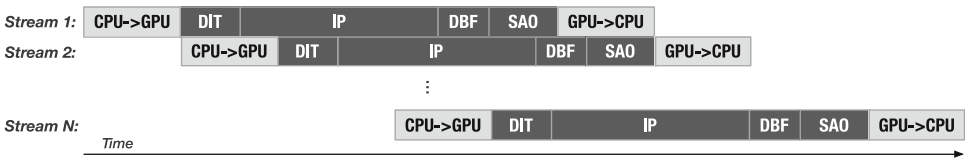
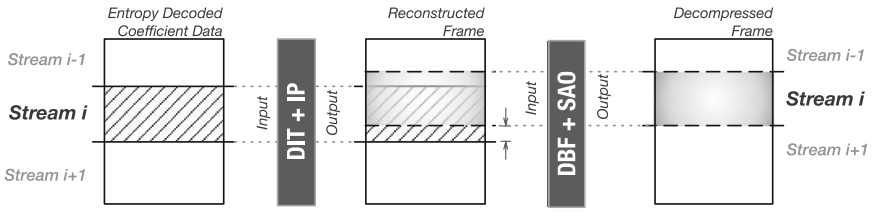


Fig. 12 Proposed HEVC intra decoder



(a) Single frame decoding with several CUDA streams.



(b) Data flow within single CUDA stream.

Fig. 13 Asynchronous CUDA stream processing in the proposed GPU HEVC intra decoder

In order to fully exploit the GPU computational capabilities for real-time HEVC intra decoding, an additional level of parallel processing is considered herein, where different portions of the current frame are simultaneously decompressed. In fact, although the strict data dependencies imposed by the IP module do not allow an efficient parallel processing across different portions of the frame, the parallel execution can be extended to the other GPU intra decoding modules, i.e., DIT, DBF, and SAO. By adopting such a pipelined processing scheme, the DIT can be implemented in parallel on different parts of the frame, before the IP is performed. Furthermore, the DBF and the SAO modules can also simultaneously process different portions of the reconstructed frame.

In particular, this frame-level parallelism can be further exploited in NVIDIA GPU, by relying on Compute Unified Device Architecture (CUDA) streams [24], which provide the means to overlap memory transfers and GPU kernels execution (across different streams). As it is presented in Fig. 13a, different entropy decoded portions of the current frame are assigned for parallel processing with several CUDA streams. As a result, the memory transfers in both directions (i.e., input CPU → GPU, and output GPU → CPU) from a single stream can be overlapped with the intra decoding performed in other streams. In modern NVIDIA GPU devices, it is also possible to overlap the kernel computations across different CUDA streams (see Fig. 13a).

However, when decoding is simultaneously performed on different portions of a single frame, a special attention must be paid to preserve the compliancy with the HEVC standard, i.e., the one already attained at the level of individual intra decoding modules. In fact, due to the strict data dependencies in the IP module (see Sect. 2), the parallel processing within a single frame can only be efficiently exploited at the level of CTU rows. Hence, each CUDA stream must be assigned with a set of consecutive CTU rows.

Nonetheless, this frame partitioning involves additional data dependencies among different CUDA streams in the DBF module, since the DBF must be applied on the horizontal edges in the top and bottom borders of the assigned set of CTU rows. As a result, to properly filter these top and bottom horizontal edges, it is required to obtain 8 additional pixel rows (i.e., 4 from the above stream and 4 from the below stream). These data dependencies also occur in the SAO filtering module for the edge offset type, since the gradient direction may require a surrounding pixel row computed by another stream, i.e., at most 2 pixel rows for both top and bottom borders of the assigned frame portion.

In order to efficiently cope with these data dependencies, the herein proposed GPU stream-level processing scheme is organized such that the overall volume of assigned CTU rows is kept constant, but it is shifted up after the DIT and IP modules are processed. This procedure guarantees that all data dependencies are resolved for both

luma and chroma components in the DBF and SAO modules. This is illustrated in Fig. 13b: before the DBF and the SAO modules are executed, the assigned portion of the reconstructed frame in *Stream i* is shifted up when compared to the portion of the frame initially assigned to the DIT and the IP modules.

It is worth noting that the cross-stream dependencies for the DBF and the SAO modules can also be solved by shifting down the assigned region. However, the proposed data reassignment procedure allows to take advantage of the IP data dependencies, since the IP of *Stream* $i-1$ is always finished before the IP of *Stream i*. Hence, the required data for the in-loop filtering of the top horizontal border are already computed, while the same can not be guaranteed for *Stream* $i+1$.

5 Experimental evaluation

To experimentally evaluate the efficiency of the proposed GPU HEVC intra decoding modules, the recommended JCT-VC test conditions were adopted by considering the *All Intra* configuration [25]. The test also considered the video bitstreams from the highest frame resolution classes A and B, since they are the most computationally demanding. To further challenge the proposed GPU algorithms, an additional set of Ultra HD 4K sequences was also evaluated (class S) [26].

To fully exploit the targeted NVIDIA GPU architectures, the proposed algorithms were implemented with CUDA programming model version 6.5 [24] and integrated in the reference HM 15.0 HEVC decoder [27]. Accordingly, the HEVC de-quantization, inverse transform, intra prediction, deblocking filter, and sample adaptive offset are completely handled by the proposed GPU parallel modules.[1] As it was previously referred, only the HEVC entropy decoder is executed on the CPU, since it is the first decoding stage and due to its sequential and irregular nature.The HM 15.0 decoder was chosen for the baseline comparison reference, since it is the most commonly used HEVC implementation in the literature. Moreover, there are no other state-of-the-art approaches implementing the considered HEVC intra decoding modules on the GPU, which would be required for fair comparison.

Table 1 presents the experimentally obtained average times for processing a frame on each proposed GPU parallel HEVC decoding modules across a set of representative video sequences from different classes, and encoded with different QP values (i.e., 22, 27, 32, and 37). The efficiency of the proposed GPU parallelizations was evaluated in a state-of-the-art NVIDIA GPU with CUDA 6.5 [i.e., GeForce GTX 780 Ti @ 1046 MHz (G780)], using a single CUDA stream. The average frame processing times per decoding module of the original reference software HM 15.0 were also assessed on a single core of the Intel® Core™ i7-4770K CPU @ 3.50 GHz. The ratios corresponding to the average processing time of each module, measured as a percentage of the whole processing time for all QP when executed in the proposed GPU and in the benchmarked HM 15.0 CPU, are also presented in Table 1. Additionally, the obtained average speedups per Intra decoding module are also provided in Table 1, where the ratios between the CPU processing time over the GPU counterpart per module are also averaged over all QP.

As it can be observed in Table 1, the required time to perform the GPU memory transfers for a set of input and output data (i.e., "*CPU* → *GPU*" and "*GPU* → *CPU*," respectively) does not significantly vary within a single class of video sequences. This is mainly because the data to be sent to the GPU (as well as the amount of decompressed frame data to be transferred from the GPU) do not significantly depend on the QP value. In fact, the amount of input/output data mainly depends on the frame resolution, thus increasing the transfer times with the growth of the frame resolution. Such transfer times vary between 1.2–4.6 % of the total decoding time. As expected, for the HM 15.0 CPU benchmark, these memory transfers are not required.

Similarly to the data transfer times, the average frame processing times per module obtained with the proposed parallel GPU HEVC decoder and the original HM 15.0 CPU decoder also vary across different classes (e.g., the highest resolution results in the highest per-module processing time). However, for the same video resolution (classes A, B, or S), the share of each individual module, which can be assessed by the average processing time in percentage in Table 1, is different when comparing both approaches. Since the proposed GPU algorithms are fully compliant with the HEVC standard and the computational complexity of their serial versions corresponds to the HM 15.0 implementation, this variation mainly arises from the efficiency of the proposed parallelization strategies, as well as from the ability of different device architectures (GPU vs. CPU) to cope with the inherent execution characteristics of different Intra decoding modules.

Furthermore, for all tested video sequences and QP values, the IP in the proposed approach represents the most time-consuming module (up to 83.4 %), due to its strict data dependencies, computational complexity, and lower parallelism degree. However, even though the IP module is responsible for the higher processing time among all the other modules, the proposed approach allows achieving

[1] The presented GPU-based HEVC intra decoder is available upon request by e-mail to the corresponding authors.

Table 1 Average frame processing time (in milliseconds), ratio over the whole processing time (in percentages), and the average speedup per HEVC intra decoding module for the proposed approach on the G780 GPU with a single CUDA stream, as well as for the original HM 15.0 on the Intel® Core™ i7-4770K CPU

Class	Sequence	QP	Proposed GPU time (ms)						HM 15.0 CPU time (ms)			
			CPU → GPU	DIT	IP	DBF	SAO	GPU → CPU	DIT	IP	DBF	SAO
S 3840 × 2160	ParkJoy	22	0.25	3.97	18.15	0.77	0.52	0.98	109.67	133.25	62.64	16.64
		27	0.25	4.29	15.25	0.87	0.57	0.98	106.70	86.09	51.01	19.62
		32	0.25	4.22	13.83	0.96	0.47	0.98	106.67	77.70	50.33	15.90
		37	0.25	4.03	12.82	1.00	0.42	0.99	105.01	73.45	50.11	12.24
Average processing (%)			1.2	19.1	68.7	4.2	2.3	4.6	40.2	33.9	19.9	6.0
Average speedup			–	26.0×	6.1×	60.9×	32.2×	–	–	–	–	–
A 2560 × 1600	Traffic	22	0.17	1.67	12.40	0.51	0.37	0.49	56.12	70.05	35.05	11.04
		27	0.17	1.53	11.65	0.54	0.36	0.49	55.18	62.81	33.92	10.88
		32	0.17	1.42	11.11	0.58	0.32	0.49	54.29	55.95	32.51	8.79
		37	0.17	1.32	10.10	0.58	0.27	0.49	**53.67**	48.86	29.62	7.25
Average processing (%)			1.2	10.4	78.9	3.9	2.3	3.4	35.2	37.8	21.0	6.0
Average speedup			–	37.1×	5.2×	59.7×	28.7×	–	–	–	–	–
B 1920 × 1080	ParkScene	22	0.13	0.84	8.83	0.26	0.21	0.25	27.33	36.18	16.40	4.92
		27	0.14	0.75	8.40	0.28	0.21	0.25	27.25	32.84	16.17	5.15
		32	0.13	0.68	7.87	0.30	0.21	0.25	26.33	28.12	15.17	4.06
		37	0.13	0.63	7.09	0.30	0.17	0.25	26.34	23.29	13.39	2.81
Average processing (%)			1.4	7.5	83.4	3.0	2.1	2.6	35.4	39.1	20.0	5.5
Average speedup			–	37.4×	3.7×	53.6×	20.9×	–	–	–	–	–

speedups as high as 6.1× over the IP HM 15.0 CPU implementation.

Nevertheless, all other intra decoding modules (i.e., DIT, DBF, and SAO) exploit a higher amount of data parallelism, thus providing a significantly lower processing time and, consequently, requiring between 12.6–25.6 % of the total decompressing time. Among these modules, the higher computational demands of the DIT module result in a higher processing time, when compared to both the DBF and SAO modules. However, due to the parallelization approach proposed for the DIT module, a speedup as high as 37.4× is achieved over the CPU counterpart DIT module.

In what concerns the HM 15.0 CPU per-module processing times, the IP is also the most time-consuming for higher bitrates (or QP = 22). However, since the parallelism is not exploited by the benchmarked implementation of the DIT module, this module is the most computationally demanding for low bitrates (or QP = 37). Actually, for the HM 15.0 CPU decoder, the IP and DIT modules account between 73.0 and 74.5 % of the total decoding time.

An interesting phenomenon is also worth noting in the considered set of sequences, when comparing the IP decoding module and the rest of the implemented modules, in what concerns their processing time across different QP. Within a single class, the IP module processing time significantly varies with the QP value: it decreases with the increase of QP. On the other hand, the processing times corresponding to the DIT, DBF, and SAO modules only slightly vary with the change of the QP value. The observed decrease in the IP processing time for larger QPs (lower bitrates) might be explained by the fact that the encoder tends to favor the resulting bitrate over the attained visual quality, in order to achieve higher bitrate savings. Hence, it mostly selects larger TUs and PUs encoded with Planar or DC intra modes. This yields a coarse-grained frame partitioning, reducing the total number of blocks to be processed. As a consequence, the IP algorithm on the decoder side has to deal with less data dependencies and performs the computationally less demanding modes (Planar or DC).

In contrast to the IP decoding module, the DBF processing time, in the GPU-based approach, slightly increases with the QP value, since the presence of blocking artifacts is more visible for larger QP values. Nevertheless, this module sustains the highest obtained speedups among all HEVC Intra decoding modules, which can be even 60.9× faster than the DBF CPU implementation.

In order to further evaluate the efficiency and scalability of the proposed GPU HEVC parallel modules at the level of the whole HEVC intra decoder, an additional high-performance NVIDIA GPU device was considered, i.e., Tesla K40c @ 745 MHz (K40c). The K40c GPU was

chosen since it has the same architecture and number of GPU cores than G780, but with lower core frequency and two copy engines. Moreover, concurrent CUDA streams [24] were used to overlap data transfers and kernels in this second experiment, where each CUDA stream is responsible for a set of CTU rows. By relying on the CUDA streams, the two copy engines of the K40c GPU can overlap not only data transfers and GPU kernels, but also perform memory transfers from and to the GPU simultaneously (i.e., "*CPU* → *GPU*" and "*GPU* → *CPU*" from different CUDA streams). Table 2 presents the experimentally obtained average frame processing times for each considered test sequence. The presented results include all kernel execution times and the time to transfer the required data to/from the GPU.

Table 2 Average frame processing time (in milliseconds) obtained with the original HM 15.0 and with the proposed HEVC intra decoding modules using multiple CUDA streams in the considered GPU devices

Class	Sequence	QP	HM 15.0	Proposed	
				G780	K40c
S 3840 × 2160	CrowdRun	22	332.15	23.27	29.25
		27	280.37	22.23	26.66
		32	260.26	21.13	25.22
		37	240.89	20.12	23.53
	DucksTakeOff	22	355.57	24.43	30.67
		27	331.16	23.44	29.44
		32	252.40	16.45	20.58
		37	222.53	13.09	16.29
	InToTree	22	299.96	23.18	29.12
		27	256.72	21.37	24.82
		32	224.15	17.40	20.32
		37	194.78	14.70	16.99
	OldTownCross	22	343.12	24.10	30.25
		27	296.44	23.01	28.11
		32	222.68	17.90	21.76
		37	191.03	15.57	18.73
	ParkJoy	22	319.09	23.44	29.42
		27	269.14	22.00	25.79
		32	251.88	20.78	23.80
		37	237.88	19.36	22.28
A 2560 × 1600	Traffic	22	162.29	14.68	18.77
	PeopleOnStreet	22	158.71	13.69	17.70
	Nebuta	22	114.30	10.23	13.07
	SteamLocomotive	22	111.13	10.22	13.12
B 1920 × 1080	Kimono	22	60.25	7.38	9.52
	ParkScene	22	81.02	9.77	12.57
	Cactus	22	79.87	9.39	12.14
	BQTerrace	22	77.61	8.59	11.06
	BasketballDrive	22	72.20	9.18	11.83

As it can be observed in Table 2, the overall processing time decreases with the increase of QP for both the original HM 15.0 decoder and the herein proposed parallel implementation in all presented sequences from the S class. As expected, the highest processing time corresponds to the higher bitrate (i.e., QP = 22). Consequently, only the results for the more demanding parametrization (QP = 22) are shown in Table 2 for the tested set of sequences from other classes.

As before, the average execution time varies across different frame resolutions, where class B achieves the lowest execution time. Nonetheless, the average processing time also varies for the same class and QP value, which is determined by the specific video content, i.e., mainly because of the chosen intra modes, TU and PU sizes by the encoder. However, the impact of these encoder decisions in the resulting processing time is mitigated by the amount of parallelism that is exploited by the GPU algorithms. This can be observed in Table 2, where the proposed algorithms achieve significantly lower processing times when compared with the sequential HM 15.0 decoder. On average, and across all sequences for QP = 22, the proposed algorithms outperform the HM 15.0 CPU execution for about 8.77× on K40c and 11.2× on G780. Furthermore, the G780 GPU delivers faster execution times when compared to the K40c GPU, mainly due to its higher clock frequency. However, even though the K40c GPU has lower computational capabilities than the G780 GPU, the processing times of the proposed intra decoder on K40c is 26 % slower on average due to the two copy engines provided by the K40c.

For all tested video bitstreams, Fig. 14 presents the speedups obtained on the G780 GPU with the proposed algorithms when compared to the single-core HM 15.0 execution. As it can be seen, the higher speedups are

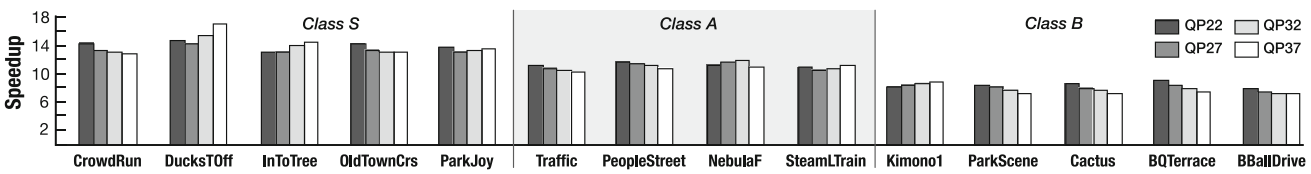
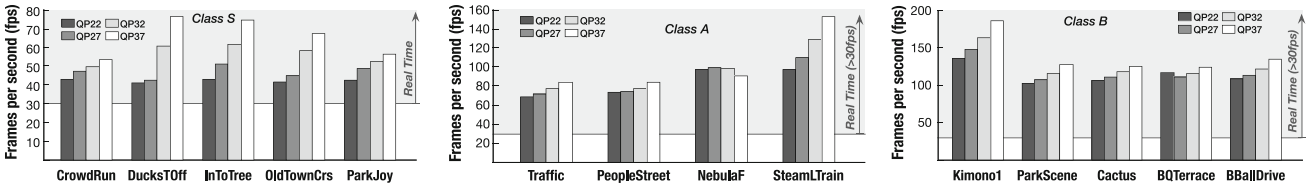


Fig. 14 Obtained speedup with the proposed GPU intra decoding modules (on G780) over the HM 15.0 reference software



(a) Class S. **(b)** Class A. **(c)** Class B.

Fig. 15 Average frame rate obtained with the proposed GPU HEVC intra decoding modules on G780 GPU

obtained for the higher frame resolution sequences, due to the increased amount of computational load.In general, the obtained speedup is balanced among the considered QP values for a single sequence, as well as among the considered classes (i.e., on average 8×, 11× and 13.8× for classes B, A, and S, respectively). Among all tested sequences, the *DucksTakeOff* sequence achieves the highest speedup of 17× for QP = 37.

To evaluate the resulting real-time processing capabilities of the proposed HEVC GPU parallel modules, Fig. 15 shows the obtained average frame rate on the G780 GPU for different QP parameters across a set of video sequences from different classes, namely, from classes S, A, and B (see Fig. 15a–c, respectively). As it can be observed, the proposed GPU implementation is able to offer real-time execution 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 52.9 fps for class S, 92.9 FPS for class A and 124.9 FPS for class B. Hence, the obtained experimental results demonstrate the feasibility of accelerating the intra decoding modules by using GPU devices and the achievement of real-time processing.

6 Conclusions

To circumvent the added complexity of the intra decoding procedure defined by HEVC, this paper proposed an efficient parallelization of the most important modules of the intra decoder using GPU accelerators, including de-quantization, inverse transform, intra prediction, deblocking filter, and SAO. To attain such objective, the presented implementation extensively exploits both fine and coarse-grained parallelization in an integrated perspective, by redesigning the execution pattern of the involved modules, while simultaneously coping with their inherent computational complexity and strict data dependencies. According to the presented experimental evaluation, the proposed parallel implementation provides a significant reduction of the overall processing time (only 25 ms for Ultra HD 4 K intra frames, i.e., 40 fps), thus allowing the satisfaction of the hard real-time requirements imposed by the decoding procedure.

References

1. Sullivan, G.J., Ohm, J., Han, W.-J., Wiegand, T.: Overview of the high efficiency video coding (HEVC) standard. IEEE Trans. Circuits Syst. Video Technol. **22**(12), 1649–1668 (2012). doi:10.1109/TCSVT.2012.2221191. ISSN 1051–8215
2. Nguyen, T., Marpe, D.: Performance analysis of HEVC-based intra coding for still image compression. Pict Coding Symp. **2012**, 233–236 (2012). doi:10.1109/PCS.2012.6213335
3. Bossen, F., Bross, B., Suhring, K., Flynn, D.: HEVC complexity and implementation analysis. IEEE Trans. Circuits Syst. Video Technol. **22**(12), 1685–1696 (2012). doi:10.1109/TCSVT.2012.2221255. **ISSN 1051–8215**
4. Lainema, J., Bossen, F., Han, W.-J., Min, J., Ugur, K.: Intra coding of the HEVC standard. IEEE Trans. Circuits Syst. Video Technol. **22**(12), 1792–1801 (2012). doi:10.1109/TCSVT.2012.2221525. **ISSN 1051–8215**
5. Budagavi, M., Fuldseth, A., Bjøntegaard, G., Sze, V., Sadafale, M.: Core transform design in the high efficiency video coding (HEVC) standard. IEEE J. Sel. Top. Signal Process. **7**(6), 1029–1041 (2013). doi:10.1109/JSTSP.2013.2270429. **ISSN 1932–4553**
6. Norkin, A., Bjøntegaard, G., Fuldseth, A., Narroschke, M., Ikeda, M., Andersson, K., Zhou, M., Van der Auwera, G.: HEVC deblockingfilter. IEEE Trans. Circuits Syst. Video Technol. **22**(12), 1746–1754 (2012). doi:10.1109/TCSVT.2012.2223053. **ISSN 1051–8215**
7. Fu, C.-M., Alshina, E., Alshin, A., Huang, Y.-W., Chen, C.-Y., Tsai, C.-Y., Hsu, C.-W., Lei, S.-M., Park, J.-H., Han, W.-J.: Sample adaptive offset in the HEVC standard. IEEE Trans. Circuits Syst. Video Technol. **22**(12), 1755–1764 (2012). doi:10.1109/TCSVT.2012.2221529. **ISSN 1051–8215**
8. Santos, L., López, S., Callicoó, G.M., López, J.F., Sarmiento, R.: Performance evaluation of the H.264/AVC video coding standard for lossy hyperspectral image compression. IEEE J. Sel. Top. Appl. Earth. Obs. Remote Sens. **5**(2), 451–461 (2012). doi:10.1109/JSTARS.2011.2173906. **ISSN 1939–1404**
9. Sanchez, V., Bartrina-Rapesta, J.: Lossless compression of medical images based on HEVC intra coding. In: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 6622–6626 (2014). doi:10.1109/ICASSP.2014.6854881
10. Panasonic UK & Ireland. LUMIX G-DMC-GH4 Camera. http://www.panasonic.com/uk/consumer/cameras-camcorders/lumix-g-compact-system-cameras/dmc-gh4.html. Accessed 11 June 2015
11. Khan, M.U.K., Shafique, M., Henkel, J.: Software architecture of high efficiency video coding for many-core systems with power-efficient workload balancing. In: Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014, pp. 1–6. European Design and Automation Association, Leuven, Belgium (2014). doi:10.7873/DATE.2014.232
12. Abramowski, A., Pastuszak, G.: A novel intra prediction architecture for the hardware HEVC encoder. In: 2013 Euromicro Conference on Digital System Design (DSD), pp. 429–436 (2013). doi:10.1109/DSD.2013.54

13. Chi, C.C., Alvarez-Mesa, M., Bross, B., Juurlink, B., Schierl, T.: SIMD acceleration for HEVC decoding. IEEE Trans. Circuits Syst. Video Technol. **PP**(99), 1–1 (2014). doi:10.1109/TCSVT.2014.2364413. **ISSN 1051–8215**
14. Kirk, D.B., Hwu, W.W.: Programming Massively Parallel Processors: A Hands-on Approach, 2nd edn. Morgan Kaufmann Publishers Inc., Waltham (2013)
15. 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)
16. Cebrián-Márquez, G., Hernández-Losada, J.L., Martínez, J.L., Cuenca, P., Tang, M., Wen, J.: Accelerating HEVC using heterogeneous platforms. J. Supercomput. (2014). doi:10.1007/s11227-014-1313-8. **ISSN 0920-8542**
17. Xiao, W., Wu, F., Xu, J., Shi, G.: Fast HEVC encoding with GPU assisted reference picture selection. In Advances in Multimedia Information Processing–PCM 2013. In: Lecture Notes in Computer Science, vol. 8294, pp. 233–244. Springer, Berlin (2013). doi:10.1007/978-3-319-03731-8_22. ISBN 978-3-319-03730-1
18. Momcilovic, S., Ilic, A., Roma, N., Sousa, L.: Dynamic load balancing for real-time video encoding on heterogeneous CPU + GPU systems. IEEE Trans. Multimed **16**(1), 108–121 (2014). doi:10.1109/TMM.2013.2284892. **ISSN 1520–9210**
19. Engelhardt, D., Möller, J., Hahlbeck, J., Stabernack, B.: FPGA implementation of a Full HD real-time HEVC main profile decoder. IEEE Trans. Consum. Electron. **60**(3), 476–484 (2014). doi:10.1109/TCE.2014.6937333. **ISSN 0098–3063**
20. de Souza, D.F., Roma, N., Sousa, L.: Cooperative CPU + GPU deblocking filter parallelization for high performance HEVC video codecs. In: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 4993–4997 (2014). doi:10.1109/ICASSP.2014.6854552
21. de Souza, D.F., Roma, N., Sousa, L.: OpenCL parallelization of the HEVC de-quantization and inverse transform for heterogeneous platforms. In: 2014 Proceedings of the 22nd European Signal Processing Conference (EUSIPCO), pp. 755–759 (2014)
22. Lee, K., Lee, H.-J., Kim, J., Choi, Y.: A novel algorithm for zero block detection in high efficiency video coding. IEEE J. Sel. Top. Signal Process. **7**(6), 1124–1134 (2013). doi:10.1109/JSTSP.2013.2272772. **ISSN 1932–4553**
23. Sousa, L.A.: General method for eliminating redundant computations in video coding. Electron Lett. **36**(4), 306–307 (2000). doi:10.1049/el:20000272. **ISSN 0013–5194**
24. NVIDIA: CUDA™ Programming Guide. NVIDIA. v6.5 (2014)
25. Bossen, F.: Common test conditions and software reference configurations. Doc. JCTVC-L1100 of JCT-VC (2013)
26. Haglund, L.: The SVT high definition multi format test set. Technical report, Sveriges Television AB (SVT), Sweden (2006). ftp://vqeg.its.bldrdoc.gov/HDTV/SVT_MultiFormat/2160p50_CgrLevels_Master_SVTdec05_/. Accessed 11 June 2015
27. JCT-VC: Subversion repository for the HEVC test model version HM 15.0 (2014). https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-15.0/

Diego Felix de Souza is currently pursuing his Ph.D. thesis at the Instituto Superior Técnico (IST), Universidade de Lisboa, Portugal and is a member of the SiPS research group at INESC-ID, under the supervision of Prof. Leonel Sousa and Prof. Nuno Roma. He obtained his MSc degree in Electrical Engineering in 2010 from the Universidade Federal do Rio de Janeiro, Brazil. His current research interests are mainly focused on efficient GPU parallelization techniques for video coding applications, mainly based on the HEVC standard.

Aleksandar Ilic received his Ph.D. degree in Electrical and Computer Engineering from Instituto Superior Tecnico (IST), Universidade de Lisboa, Portugal, in 2014. He is currently an Assistant Professor in the Department of Electrical and Computer Engineering of IST and a Senior Researcher of the Signal Processing Systems Group (SiPS), Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). His research interests include high-performance and energy-efficient computing and modeling on parallel heterogenous systems.

Nuno Roma received the Ph.D. degree in Electrical and Computer Engineering from Instituto Superior Técnico (IST), Universidade Técnica de Lisboa, Lisbon, Portugal, in 2008. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering of IST and a Senior Researcher of the Signal Processing Systems Group (SiPS) of Instituto de Engenharia de Sistemas e Computadores R&D (INESC-ID). His research interests include specialized computer architectures for digital signal processing, parallel processing and high-performance computing. He has contributed to more than 50 papers to journals and international conferences. Dr. Roma is a Senior Member of the IEEE Circuits and Systems Society and a Member of ACM.

Leonel Sousa received a Ph.D. degree in Electrical and Computer Engineering from the Instituto Superior Tecnico (IST), Universidade de Lisboa (UL), Lisbon, Portugal, in 1996, where he is currently Full Professor. He is also a Senior Researcher with the R&D Instituto de Engenharia de Sistemas e Computadores (INESC-ID). His research interests include VLSI architectures, computer architectures, parallel computing, computer arithmetic, and signal processing systems. He has contributed to more than 200 papers in journals and international conferences, for which he got several awards—such as, DASIP'13 Best Paper Award, SAMOS'11 'Stamatis Vassiliadis' Best Paper Award, DASIP'10 Best Poster Award, and the Honorable Mention Award UTL/Santander Totta for the quality of the publications in 2009. He has contributed to the organization of several international conferences, namely as program chair and as general and topic chair, and has given keynotes in some of them. He has edited two special issues of international journals, and he is currently Associate Editor of the IEEE Transactions on Circuits and Systems for Video Technology and of the Springer journals JES and JRTIP. He is Fellow of the IET, a Senior Member of both IEEE and ACM, and Member of IFIP WG10.3 on concurrent systems.