TOWARDS GPU HEVC INTRA DECODING: SEIZING FINE-GRAIN PARALLELISM

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

INESC-ID, IST, Universidade de Lisboa
Rua Alves Redol 9, 1000-029 Lisboa, Portugal
{diego.souza,aleksandar.ilic,nuno.roma,leonel.sousa}@inesc-id.pt

ABSTRACT

To satisfy the growing demands on real-time video decoders for high frame resolutions, novel GPU parallel algorithms are proposed herein for fully compliant HEVC de-quantization, inverse transform and intra prediction. The proposed algorithms are designed to fully exploit and leverage the fine grain parallelism within these computationally demanding and highly data dependent modules. Moreover, the proposed approaches allow the efficient utilization of the GPU computational resources, while carefully managing the data accesses in the complex GPU memory hierarchy. The experimental results show that the real-time processing is achieved for all tested sequences and the most demanding QP, while delivering average fps of 118.6, 89.2 and 49.7 for Full HD, 2160p and Ultra HD 4K sequences, respectively.

***Index Terms*—** HEVC, GPU, intra prediction, inverse transform, parallelization

1. INTRODUCTION

In an optimized High Efficiency Video Coding (HEVC) decoder, the De-quantization and Inverse Transform (DIT) module and the Intra Prediction (IP) module are responsible for 20-29% of the total decoding time [1]. This is mainly due to the several transform and block sizes that have to be considered for DIT and IP, together with 35 different IP modes [2]. Besides this large computational load, the execution of both modules must respect strict data dependencies imposed by the HEVC standard. These dependencies not only occur between the DIT and IP, but also between adjacent IP data blocks [3].

Reducing the HEVC decoding time via parallelization of decoder modules represents one of the most important topics in modern video coding. Although HEVC decoder implementations are already proposed for multi-core Central Processing Units (CPUs) [4] and Field-Programmable Gate Arrays (FPGAs) [5], the existing Graphics Processing Unit (GPU) approaches are limited to only a certain set of HEVC modules, namely motion estimation [6], deblocking filtering [7] and inverse transform [8].

This work was supported by national funds through Fundação para a Ciência e a Tecnologia – FCT, under projects PTDC/EEI-ELC/3152/2012, UID/CEC/50021/2013 and SFRH/BD/76285/2011.

In this paper, a set of efficient and fully compliant GPU parallel algorithms for the HEVC DIT and IP modules are proposed. In our previous work [8], the GPU DIT algorithm relies on a coarse-grain parallelization without synchronization between stages. In contrast, the GPU DIT algorithm proposed herein advances these contributions by providing a fine-grain parallelization of the DIT computational load, which allows achieving even higher performance on high-end GPUs.

Nevertheless, designing an efficient and fully compliant HEVC DIT and IP for GPU devices is far from being a trivial task, due to their inherent computational complexity and strict data dependencies. In fact, these difficulties were already noted at the CPU SIMD HEVC decoder implementation proposed in [4], where the HEVC IP module achieved one of the lowest speed-ups among all the other decoder modules. The authors in [4] also exploit multi-core parallel processing where the computational load is distributed among the CPU cores by combining the HEVC Wavefront Parallel Processing (WPP) scheme and frame-level parallelism. In contrast, the implementation proposed in this paper is fully compliant with both HEVC parallelization techniques (namely *Tiles* and WPP) [2] and represents one of the first attempts in the literature to port the execution of both DIT and IP modules on the highly data-parallel GPU architecture.

This paper is organized as follows: the HEVC DIT and IP principles are introduced in Section 2, the proposed parallel algorithms are presented in Section 3, while the experimental results and conclusions are shown in Sections 4 and 5.

2. HEVC INTRA DECOMPRESSION

In the HEVC standard, a picture is partitioned in $L\times L$ pixel blocks called Coding Tree Units (CTUs), where L is selected by the encoder ($L\in\{16, 32, 64\}$). The CTUs are grouped in slices or tiles of the frame and decoded in raster scan order. Each CTU is independently split using a quadtree structure in blocks called Coding Units (CUs), between a maximum size of 64×64 and a minimum size of 8×8 pixels. Furthermore, each CU is divided in a Prediction Unit (PU) and a Transform Unit (TU), which correspond to the predicted and the residual block, respectively [2]. Inside a CTU, the CUs are decoded in z-scan order, as well as the PUs and TUs in a CU.

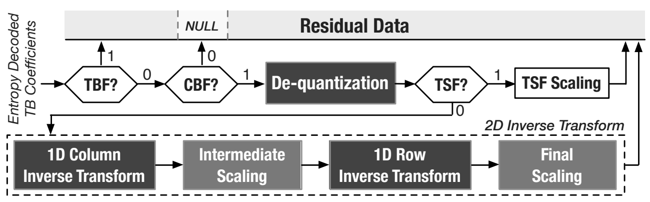

Fig. 1. The HEVC residual data decompressing flowchart.

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

Inside a CU, the TU is further split in smaller blocks (4×4, 8×8, 16×16 or 32×32) according to the quadtree structure. These blocks are named Transform Blocks (TBs) and each TU is composed by luma and chroma TBs.

According to the HEVC standard (see Fig. 1), DIT is applied on the *TB coefficients* 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). When TBF is set, DIT is bypassed, which means that the residual data directly corresponds to the TB coefficients. The CBF indicates if there is some residual data in the TB. When CBF is set, the de-quantization is applied to the TB coefficients. The De-quantization module implements the HEVC inverse scaling, which depends on the Quantization Parameter (QP) and on the adopted TB size.

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, which consists of 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 the *transform core*, which is chosen in respect to the TB size and prediction mode. In the HEVC standard, only integer transform cores are specified, where the 4×4 to 32×32 transform cores are based on the integer discrete cosine or sine transforms [9].

2.2. Intrapicture 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 PUs for an 8×8 CU.

Similarly to the TU, the PU is further divided in luma and chroma Prediction Blocks (PBs), where the intra prediction is applied to each PB. The final reconstructed block is obtained by adding the prediction from PB and the residual data from

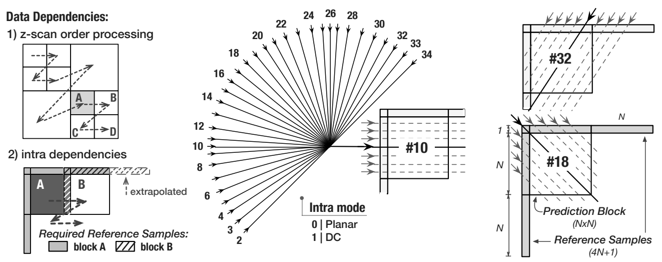

Fig. 2. Intra prediction dependencies and modes.

DIT. The PBs are processed in a strictly defined order specified by the HEVC standard [2]. As presented with dashed-line in Fig. 2, the data dependencies come from the *z-scan order processing* for different PUs. These *intra dependencies* occur because the previously reconstructed blocks (*reference samples*) are used as the input when predicting the next PBs.

Figure 2 presents an example of the required references samples for the 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 produced. 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 PB in CTU, one or more reference sample sets may not be available (see block B in Fig. 2). 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.

Upon all 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. 2, 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 [3]. In angular prediction, the interpolation is applied on the reference samples according to the specified direction [3] (e.g., #18 in Fig. 2).

Each PB is predicted using one of these intra modes, which can differ for luma and chroma PBs. When the TB is smaller than 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 TB. For the intra prediction, TU can not be larger than PU [3].

The HEVC standard also defines the I_PCM mode to reconstruct extremely rare blocks at the CU level [2]. In this case, the entropy decoder, IP and DIT are bypassed and the reconstructed block is obtained directly from the bitstream.

3. PROPOSED PARALLEL ALGORITHMS

The proposed DIT and IP algorithms are 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 mod-

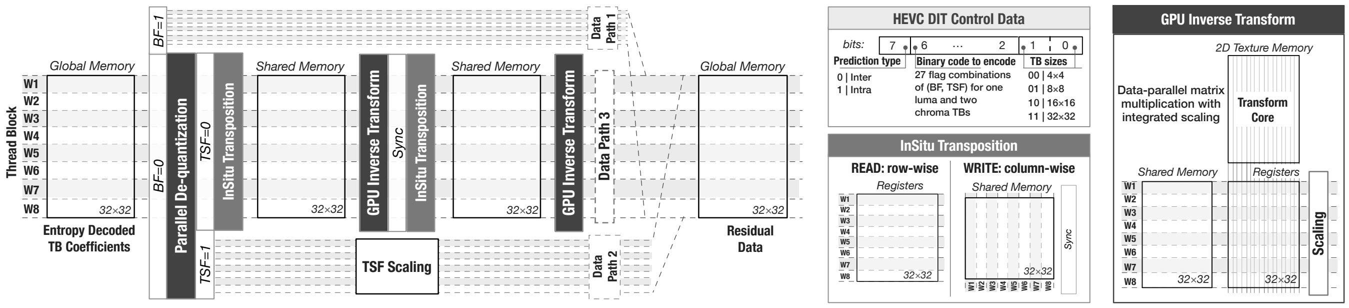

Fig. 3. Overall de-quantization and 2D inverse transform schematic in the GPU for one 32×32 luma TB.

ules, 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 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). Furthermore, the data accesses are carefully managed to efficiently use the complex GPU memory hierarchy, i.e., global, cache, shared, constant and texture memory.

3.1. GPU De-quantization and Inverse Transform

In contrast to [8], 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. 3 for the case of 32×32 TB.

In the proposed GPU DIT, a single ThB contains 8 warps (Wi), which perform the DIT computations on independent TB parts, e.g., eight 32×4 TB sub-blocks in Fig. 3. The overall procedure starts by asynchronously acquiring 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 different warps, after executing the *Data Path* 1, 2 or 3.

The *Data Path* is selected by the DIT flags, i.e., TBF, CBF and TSF (see Section 2). In the proposed implementation, TBF and CBF of a single TB are merged in one new flag, which is named Bypass Flag (BF) [8]. To reduce the communication overheads, a specific 8-bit *HEVC DIT Control Data* structure is designed to integrate all 27 BF and TSF combinations. In brief, for each TB, there are 3 possible flag combinations, i.e., $(\mathrm{BF}, \mathrm{TSF}) \in \{(1,*); (0,1); (0,0)\}$, giving a total of 27 ($3^3$) combinations for one luma and two chroma TBs. This information is binary encoded with five bits and stored in bit positions 2–6 of the *HEVC DIT Control Data*. Furthermore, bit positions 0 and 1 of this structure are reserved for the TB size information (see Fig. 3). The remaining bit position 7 is used to designate the prediction type (Intra or Inter).

The *HEVC DIT Control Data* is packed for each 4×4 block. This data is used by the warps to extract the BF and TSF values during the GPU kernel execution, i.e., to select the *Data Path* for each 32×4 TB sub-block in Fig. 3. 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. 3). In the *Parallel De-quantization*, each thread in a warp independently de-quantizes one TB coefficient.

If TSF is set, the warps asynchronously perform the *TSF Scaling* in the *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 rearranges the data to the correct form. As presented in Fig. 3, the first *InSitu Transposition* is applied to the de-quantized TB coefficients (in the GPU registers). Here, each warp rearranges 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*, the shared memory is always read in a row-wise manner by each warp. Then, the data-parallel matrix multiplication is performed between the read data and the selected *Transform Core* (stored in the GPU texture memory). Afterwards, the HEVC Intermediate *Scaling* is separately applied on the obtained results by each warp.

To perform 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 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 integrated HEVC Final *Scaling*. Finally, each warp asynchronously writes the produced residual data in the GPU global memory.

Figure 4 presents an example of warp assignments for dif-

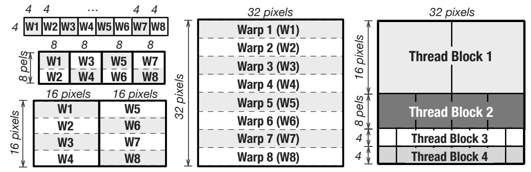

Fig. 4. Example of IT warp and ThB assignments.

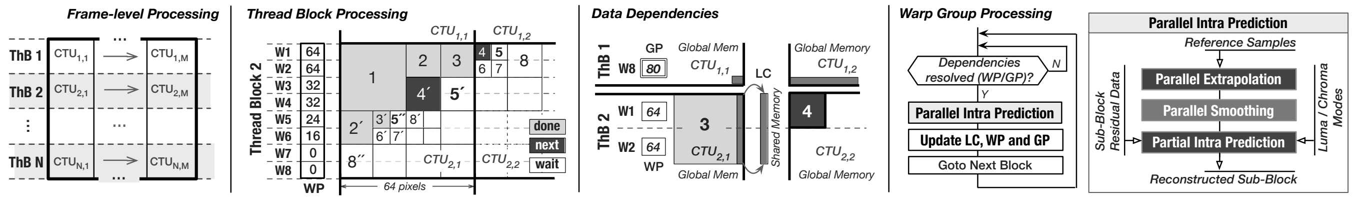

Fig. 5. GPU intra prediction warps assignment and framework with a wavefront approach.

ferent TB sizes, i.e., 4×4, 8×8, 16×16 and 32×32. In brief, when the GPU kernels starts, all eight warps within a ThB firstly obtain 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 4 also shows an example of how 4 ThBs can be assigned for a 32×32 partitioned TB.

It is worth emphasizing that the proposed parallel DIT algorithm allows an efficient utilization of the GPU memory hierarchy by organizing the data accesses as follows: *i*) the *HEVC DIT Control Data*, TB coefficients and residual data are stored in the global memory; *ii*) the transform cores 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) is stored in the constant memory and broadcasted in one warp memory transaction.

3.2. GPU Intra Prediction

Due to the strict dependencies on reconstructed blocks, the proposed GPU parallel IP algorithm adheres to the wavefront execution paradigm. As it is shown in Fig. 5 (*Frame-level and Thread Block Processing*), a single ThB with eight warps is assigned to process 64 pixel-rows of the luma component (32 for chroma), e.g., a row with 64×64 CTUs. Hence, each warp computes 8 consecutive luma pixel-rows (or 4 for chroma).

When processing a single N×N luma block (N≥8), each warp is responsible for the prediction and reconstruction of a corresponding N×8 sub-block. For chroma, the same warp operates on a corresponding (N/2)×4 sub-block. As it can be observed, this processing granularity is chosen in order to keep 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 acquired by reusing the data available in the global memory, i.e., bits 0 and 1 of the *HEVC IT Control Data*. This is because IP is performed after DIT, where PBs are processed at the TB level. When a single warp completes the assigned sub-block, it proceeds to the next N×8 luma sub-block in 8 pixel-row. However, the IP procedure is performed only if 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. 5 (*Thread Block Processing*), where the numbers assigned to the blocks represent their processing order. For example, only after the warps W1–W4 finish processing of block 1, can blocks 2 and 2' be performed in parallel.

To keep track of the reconstructed neighboring blocks, the proposed IP algorithm implements a specific warp-level 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. 5, the presented WPs reflect the position of each warp before the blocks 4 and 4' are performed. As it can be seen, W1 and W2 have WP=64, since they were responsible for processing of a 32×8 sub-block of block 1 and 16×8 sub-blocks of blocks 2 and 3 (32+2·16=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 for the blocks in a CTU. According to this information, the warp queries the WPs of the warps responsible for performing these neighboring blocks, in order to check the dependency status. In the case of 8×8 block 4 from ThB 2, presented in Fig. 5 (*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, block 4 can only start if WPs of both W1 and W2 from ThB 2 are set to 64 and WP of W8 from ThB 1 is set to 80 (see Section 2).

Since the warps from one ThB can not access the shared memory from other ThBs, 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 ThBs (64×64 CTUs) are checked by assessing GP of the previous ThB. Hence, in Fig. 5 (*Data Dependencies*), W1 inspects GP of ThB 1, when checking dependencies between ThBs.

As presented in Fig. 5 (*Warp Group Processing*), the warp proceeds 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 LC (for luma and chroma components) with its reconstructed pixels. As a result, other warps can access this information with less and faster memory transactions. Nevertheless, the upper reference samples are brought from the global memory, since this data is already coalesced and can take advantage of the GPU L2 cache.

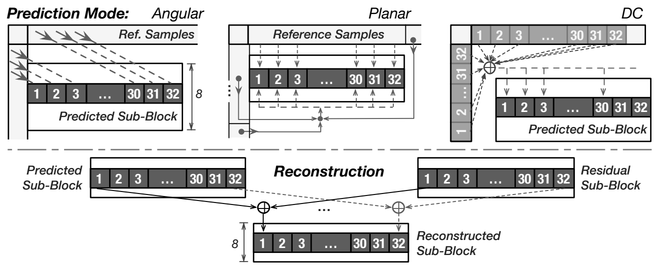

Fig. 6. Proposed GPU *Mode Prediction* and *Reconstruction*.

After fetching the *Reference Samples*, the *Parallel Extrapolation* is performed to fill the whole $4N+1$ reference sample set. Here, the nearest available reference sample is broadcast to all threads in a warp, where each thread simultaneously copies this value to a single unavailable reference sample position. Afterwards, for luma blocks, the *Parallel Smoothing* is performed, where each thread is responsible for applying the HEVC smooth filtering to a single reference sample. To avoid the need of 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 an N×8 sub-block, in order to produce the corresponding reconstructed sub-block.

The *Partial Intra Prediction* is performed in two steps: *Mode Prediction* and *Reconstruction*. In the *Mode Prediction*, 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 luma or chroma prediction mode, which is stored in a 2-byte word for each 4×4 block. As presented in Fig. 6 for a 32×8 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 between the pixel to be predicted and the reference samples. In the *DC Mode*, all threads cooperatively compute the reference sample average to produce the predicted pixels. In the second step, the parallel *Reconstruction* is performed by adding the predicted sub-block with the residual sub-block. The produced 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 before going to the next block. The overall process is repeated until all warps reach the end of the frame. As mentioned before, in the I_PCM mode, the residual data is marked as the reconstructed sub-block and the *Mode Prediction* is bypassed.

4. EXPERIMENTAL RESULTS

To experimentally evaluate the efficiency of the proposed GPU algorithms for DIT and IP, the JCT-VC test conditions were adopted with *All Intra* configuration [10]. The video bitstreams from the highest frame resolution classes A and B were considered, since they represent the most demanding setups. To further challenge the proposed algorithms, a set of Ultra HD 4K sequences [11] was also evaluated (class S).

Class	Sequence	QP	HM 15.0	Proposed	
				G780	K40c
S 3840×2160	DucksTakeOff	22	273.67	22.62	29.53
		27	239.78	21.37	27.36
		32	186.19	15.22	20.02
		37	172.20	12.11	15.90
	CrowdRun	22	245.77	21.58	28.11
	InToTree	22	228.26	21.26	28.08
	OldTownCross	22	262.87	22.33	29.34
	ParkJoy	22	235.60	21.55	28.27
A 2560×1600	Traffic	22	114.64	13.56	17.60
	PeopleOnStreet	22	111.90	12.69	16.57
	Nebuta	22	92.40	9.14	11.93
	SteamLocomotive	22	79.78	9.46	12.29
B 1920×1080	Kimono	22	45.69	7.25	9.55
	ParkScene	22	58.52	9.19	12.10
	Cactus	22	58.88	8.95	11.80
	BQTerrace	22	56.36	8.06	10.77
	BasketballDrive	22	52.46	8.69	11.47

Table 1. Average frame processing time (in milliseconds) of only the HEVC DIT and IP modules.

To fully exploit the GPU architecture, the proposed algorithms were implemented with CUDA [12] and integrated in the reference HM 15.0 HEVC decoder. Hence, the HEVC DIT and IP modules, including I_PCM mode, are handled by the proposed GPU algorithms. All other HEVC modules, such as entropy decoder and in-loop filters, are executed on the CPU with the original HM. In the GPU execution, CUDA Streams [12] are used to overlap data transfers and kernels, where each CUDA stream is in charge for a set of CTU rows.

Table 1 presents the experimentally obtained average frame processing time (considering only the joint DIT and IP modules) for each considered test sequence. The efficiency of the proposed algorithms was evaluated in two state-of-the-art NVIDIA GPUs with CUDA 6.5, i.e., Tesla K40c @ 876 MHz (K40c) and GeForce GTX 780 Ti @ 1046 MHz (G780). The presented results of the proposed GPU algorithms include the kernel execution time and the time to transfer the required data to/from the GPU. The reported time of the original HM 15.0 was obtained on a single core of the Intel® Core™ i7-4770K CPU @ 3.50GHz. HM 15.0 execution time was chosen as baseline comparison since it is the most used HEVC decoder in the literature and there are no approaches for DIT and IP on the GPU that can be used for comparison. Moreover, a direct comparison with the related works can not be performed on a fair-basis, since Chi et al. [4] only consider random access configuration and our previous work [8] requires residual data transfers from GPU to CPU. However, in the work presented herein, these data transfers are not required due to the integrated nature of the IP and DIT modules.

As it can be observed in Table 1 for the *DucksTakeOff* sequence, the processing time decreases with the increase of the QP for both original HM 15.0 and for the herein proposed algorithms. As expected, the highest processing time corresponds to the higher bitrate (i.e., QP=22). In contrast,

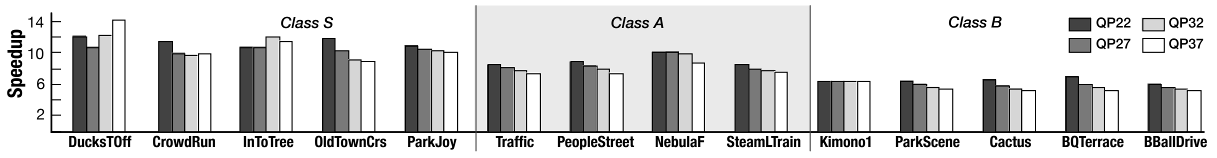

Fig. 7. Obtained speed-up for the G780 GPU implementation over the HM 15.0 reference software.

when using larger QPs (lower bitrates), a shorter execution time is attained. This is because the encoder tends to choose larger TUs and PUs with Planar or DC intra modes, in order to achieve bitrate savings, yielding a coarser grained frame partitioning and reducing the total number of blocks. Hence, on the decoder side, the GPU IP algorithm for this bitstream deals with less dependencies and performs the computationally less demanding modes (Planar or DC). For these reasons, only the results for the most demanding configuration (QP=22) are shown in Table 1 for the other tested sequences.

As expected, the average execution time also varies across different frame resolutions, where class B achieves the lowest execution time. Nonetheless, the average time also varies for the same class and QP, which is determined by the video content, i.e., intra modes, TU and PU sizes chosen by the encoder. However, the impact of these encoder decisions is attenuated by the parallelism obtained in the GPU algorithms. This can be observed in Table 1, where the proposed algorithms achieve significantly lower processing time when compared with HM 15.0. On average, across all sequences for QP=22, the proposed algorithms outperform the HM 15.0 CPU execution for about 6.8× on K40c and 8.9× on G780. Furthermore, G780 delivers better execution times when compared to K40c, due to its higher clock frequency.

As it can be seen in Table 1, all proposed GPU implementations can handle real-time processing for all video sequences, i.e., at least 30 frames per second (fps) is achieved in both G780 and K40c. More specifically, for G780 and the most demanding QP=22, the proposed GPU implementations allow achieving the average fps of 49.7 for class S, 89.2 for class A and 118.6 for class B.

For all tested video bitstreams, Fig. 7 presents the speedups obtained on G780 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 (i.e., larger wavefront in IP). In general, the speedup increases when the QP is decreased. However, depending on the video content, the higher speedups can be obtained even for the other QPs. For example, *DucksTakeOff* and *InToTree* sequences achieve the highest speedups of 14.2× and 12× for QPs 37 and 32, respectively.

5. CONCLUSIONS

In this paper, efficient GPU parallel algorithms were proposed for fully compliant HEVC de-quantization, inverse transform and intra prediction. By leveraging the fine grain parallelism of these computationally complex and highly data dependent modules, the proposed algorithms efficiently exploit the capabilities of modern GPU architectures. To achieve high performance, the proposed parallelization approaches aim at maximizing the number of active warps, while carefully managing the use of the complex GPU memory hierarchy. The efficiency of the proposed algorithms was assessed on state-of-the-art GPU devices for an extensive range of computationally demanding frame resolutions (1080p, 1600p and 2160p). The experimental results show that the real-time processing was achieved for all tested sequences and for the most demanding QP setup, providing an average of 118.6 fps for Full HD sequences. For Ultra HD 4K sequences, the proposed algorithms also deliver the maximum speedup of 14.2× over the reference CPU serial execution.

6. REFERENCES

[1] F. Bossen, B. Bross, K. Suhring, and D. Flynn, “HEVC complexity and implementation analysis,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1685–1696, 2012.

[2] G. J. Sullivan et al., “Overview of the high efficiency video coding (HEVC) standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, 2012.

[3] J. Lainema, F. Bossen, W.-J. Han, J. Min, and K. Ugur, “Intra coding of the HEVC standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1792–1801, 2012.

[4] C. C. Chi et al., “SIMD acceleration for HEVC decoding,” *IEEE Trans. Circuits Syst. Video Technol.*, p. 14, 2014.

[5] D. Engelhardt et al., “FPGA implementation of a Full HD real-time HEVC main profile decoder,” *IEEE Trans. Consum. Electron.*, vol. 60, no. 3, pp. 476–484, 2014.

[6] G. Cebrián-Márquez et al., “Accelerating HEVC using heterogeneous platforms,” *Journal of Supercomputing*, 2014.

[7] D. F. de Souza, N. Roma, and L. Sousa, “Cooperative CPU+GPU deblocking filter parallelization for high performance HEVC video codecs,” in *Proceedings of the IEEE ICASSP*, 2014, pp. 4993–4997.

[8] D. F. de Souza, N. Roma, and L. Sousa, “OpenCL parallelization of the HEVC de-quantization and inverse transform for heterogeneous platforms,” in *Proceedings of EUSIPCO*, 2014.

[9] M. Budagavi et al., “Core transform design in the high efficiency video coding (HEVC) standard,” *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 6, pp. 1029–1041, 2013.

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

[11] L. Haglund, “The SVT high definition multi format test set,” Tech. Rep., Sveriges Television AB (SVT), Sweden, 2006.

[12] NVIDIA, *CUDA™ Programming Guide*, v6.5, 2014.