Efficient HEVC Decoder for Heterogeneous CPU with GPU Systems

Biao Wang, Mauricio Alvarez-Mesa,
Chi Ching Chi and Ben Juurlink
AES, Technische Universität Berlin
Email: biaowang@win.tu-berlin.de,
{mauricio.alvarezmesa,
chi.c.chi, b.juurlink}@tu-berlin.de

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

Abstract—**The High Efficiency Video Coding (HEVC) standard provides higher compression efficiency than other video coding standards but at the cost of increased computational load, which makes it hard to achieve real-time encoding/decoding of high-resolution, high-quality video sequences. In this paper, we investigate how Graphics Processing Units (GPUs) can be employed to accelerate HEVC decoding. GPUs are known to provide massive processing capability for throughput computing kernels, but the HEVC entropy decoding kernel cannot be executed efficiently on GPUs. We therefore propose a complete HEVC decoding solution for heterogeneous CPU+GPU systems, in which the entropy decoder is executed on the CPU and the remaining kernels on the GPU. Furthermore, the decoder is pipelined such that the CPU and the GPU can decode different frames in parallel. The proposed CPU+GPU decoder achieves an average frame rate of 150 frames per second for Ultra HD 4K video sequences when four CPU cores are used with an NVIDIA GeForce Titan X GPU.**

I. Introduction

The High Efficiency Video Coding (HEVC) standard [1] has established itself as the state of the art in video coding, providing a 50% bitrate reduction with the same subjective quality when compared to H.264 [2]. Such a compression improvement, however, is achieved at the cost of an increase of the computation load for encoding and decoding the video sequences. Fortunately, the HEVC standard has been designed by taking into account parallel computing architectures.

In particular, in order to achieve higher performance, Single Instruction, Multiple Data (SIMD) instructions are usually exploited on CPU architectures. In [3], Chi et al. proposed a set of optimizations for the HEVC decoder by considering all major SIMD ISAs. At the end, the proposed parallel HEVC decoder is able to process up to 133 frames per second (FPS) and 37.8 FPS for Full HD and 2160p video sequences, respectively, on a single core Intel processor.

SIMD execution is also exploited on Graphics Processing Unit (GPU) devices, which are able to deliver higher performance levels than the CPUs in applications with massive parallelism and little execution divergence. In addition, GPUs are already present in most computing devices, from state-of-the-art desktop machines to embedded systems, including tablets and smartphones. Consequently, GPU devices represent an attractive opportunity to offload the most computational demanding HEVC procedures from the CPU. However, GPU acceleration of the HEVC decoder procedure is not a trivial task. First, each decoding procedure should be completely redesigned to potentiate the exploitation of the parallelism and to reduce the execution path divergence. Second, a careful design has to be made in order to exploit multiple levels of parallelism between CPU and GPU.

In a first approach to attain efficient heterogeneous HEVC decoder implementations, some modern GPUs integrated dedicated hardware structures to perform video decoding (e.g., NVIDIA's PureVideo [4]). However, most of such hardware-based HEVC decoders offer poor flexibility, mainly because of their hardware-specific codec support. Moreover, only a few specific NVIDIA GPUs provide HEVC decoding capabilities through hardware [5] (e.g., the state-of-the-art NVIDIA GeForce GTX Titan X that is used in this paper does not offer such capability). Regarding software decoding on GPUs, Ittian system presented an HEVC decoder [6] based on their Mali-T604 GPU. In the same trend, Strongene developed another decoder implementation for AMD GPUs [7]. Nevertheless, neither of them are free nor have scientific evaluation for their implementations.

In this paper, an efficient parallelization of the HEVC decoder for heterogeneous CPU+GPU platforms is presented. To attain such objective, most of the HEVC procedures had to be re-designed so that sequential entropy decoder is executed on the CPU and the remaining decoding kernels are migrated and further optimized to be executed on the GPU [8]–[12]. Furthermore, a pipeline decoding scheme has been implemented between the CPU and the GPU, where both devices execute their tasks in parallel. In accordance, the main contributions of this paper are: *i*) **optimized GPU algorithms** for inverse transform, motion compensation, intra-prediction, and the in-loop filters. *ii*) **a pipelined decoding scheme executed on the CPU+GPU platform**: tasks assigned to the CPU and GPU devices can be executed in parallel. The proposed heterogeneous decoder is able to outperform the **state-of-the-art** CPU-based HEVC decoder [3] when the number of CPU cores is relatively small. In particular, it achieves a speedup of 2.2× when two CPU cores are employed and 1.8× when four CPU cores are used to decode Ultra HD 4K sequences.

The remaining sections are organized as follows. Section II presents the design of the parallel decoding scheme on CPU and GPU, and the performance results are discussed in Section III. Finally, the conclusions are drawn in Section IV.

II. Proposed CPU+GPU HEVC Decoding Scheme

To properly distribute the decoding tasks between the CPU and the GPU, a parallelism analysis for different kernels is performed. Then, the work flow of the offloaded kernels on the GPU is elaborated. Finally, the pipelined decoding scheme between the CPU and the GPU is presented.

A. Task Distribution between CPU and GPU

The HEVC decoding procedures can be divided into: Entropy Decoder (ED), Inverse Transform (IT), Intra Prediction (IP), Motion Compensation (MC), and In-Loop Filters (ILF) which in turn contains the Deblocking Filter (DBF) and the Sample Adaptive Offset (SAO) filter. However, not all of these decoding procedures present the same suitability for GPU processing measured in terms of the amount of offered parallelism and execution divergence. Table I presents a qualitative analysis of all decoding procedures, which are a major concern for efficient GPU execution. As it can be observed, most of the decoding procedures expose a high level of parallelism, allied with a low execution divergence. The only exception is the ED, which is divergent and dependent at bit level. As such, it should be executed by the CPU, while the remained procedures are offloaded onto the GPU.

TABLE I
Qualitative analysis of the HEVC decoding kernels in terms of Parallelism and Execution Divergence.

Type	HEVC decoding procedures					
	ED	IT	MC	IP	DBF	SAO
Parallelism	very low	high	high	medium	high	very high
Divergence	very high	low	low	medium	low	very low

B. Optimization of the Decoding Procedures for GPU Execution

Optimized CPU implementations such as the one presented in [3] performs the HEVC decoding procedures at block-level to exploit the data locality. The herein proposed GPU-based procedures, however, are performed at frame-level, which increases the parallelism for GPU execution.

Moreover, all targeted GPU kernels have been implemented using Compute Unified Device Architecture (CUDA), according to the following 3-level bottom-up organization: thread, thread block, and grid. A thread is an instance of one specific kernel and is executed in lock-step within a group of 32 threads (or *warps* in CUDA's terminology). A thread block consists of a set of threads whose size can be configured and it is usually specified as a multiple of warps. At the top level, a grid is a set of thread blocks. In accordance, all targeted kernels are mapped to one entire frame. Their thread block setups, however, varies depending on each procedure's implementation.

Figure 1 presents the execution order of decoding kernels within the proposed CPU+GPU decoder. When the entropy decoding is performed on the CPU, all GPU kernel inputs are collected and stored in CPU *Host Memory*. A Host to Device (*H2D*) memory transfer is followed after all kernel inputs are complete. When they are transferred to GPU *Global Memory*, the GPU kernels can be launched. For each GPU kernel, their thread block mapping is shown at the bottom. These decoding modules will be briefly introduced since their algorithm has been elaborated individually in [8]–[12]. For all target kernels, one common optimization is concerned with their ability to support video sequences with 10-bit depth, while previous approaches could only decode bitstreams with 8-bit depth.

As in [9], the warps of the IT GPU kernel are assigned according to the block partitioning obtained from the bitstream. The new optimizations that were herein introduced for the IT GPU kernel consist of: *i*) **better data packing**: the required data is stored in a 2 bytes word per 8×8 block, which includes the block sizes, transform flags, prediction type; and *ii*) **inter predicted blocks**: the new IT GPU kernel already supports the inverse transform of inter predicted blocks, which have not been considered for the intra decoder in [9].

If the current frame contains both intra and inter predicted blocks (i.e., not an I frame) the MC kernel (see Fig. 1) is launched with the motion vector (*MV*), reference indexes (*REFIdx*), and the corresponding decoded pictures. In Fig. 1, the

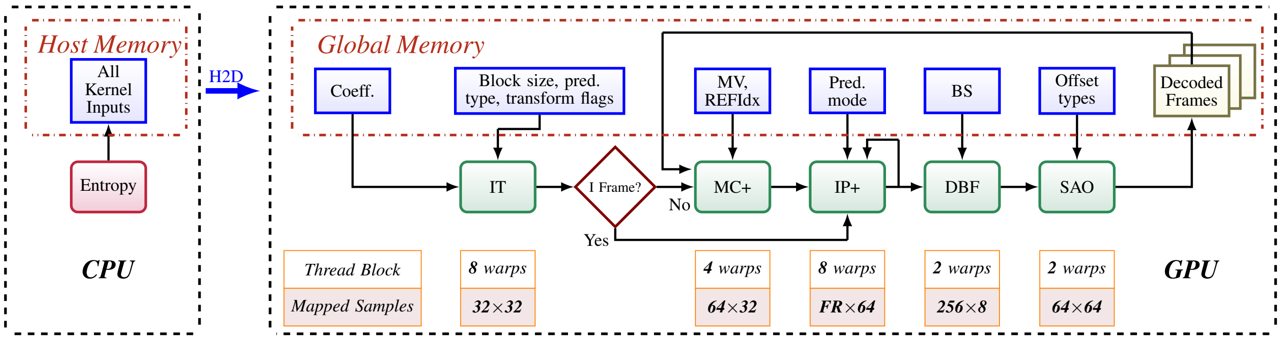


Fig. 1. Work flow overview of proposed CPU+GPU decoder for one specific frame. The entropy decoding module is assigned on the CPU and remaining kernels are offloaded on the GPU. Thread block level mapping is described at the bottom within GPU block.

MC kernel is followed with a "+" symbol to indicate that the reconstruction procedure (predicted samples + residual data) is done within the kernel. Herein, 4 warps (one thread block) are assigned to perform the inter prediction of a 32×64 samples of the frame (as in [10]). In addition, the new set of optimizations that are now proposed include: *i*) **8K video support**: since the maximum value of the motion vector in [10] is not enough to point to a reference area for 8K sequences, the motion vectors are now represented by a 64-bit word for each 8×8 block; and *ii*) **in-place reference frames**: since in-loop filters are performed in the GPU, all frames are decoded and stored in the GPU *Global Memory*, avoiding the memory transfers for reference frames between the CPU and the GPU.

The IP kernel is performed after MC due to the intrinsic data dependencies of the reconstructed neighboring blocks (see Fig. 1). If the current frame is encoded as I frame, with only intra predicted blocks, then the IP kernel is started right after the IT kernel. As it was proposed in [9], each warp performs the intra prediction of all blocks or sub-blocks in a 8-sample row of the frame. Similarly, the thread block of this kernel consists of 8 warps, which perform a frame row with a height of 64 samples (FR×64), thus accomplishing a wavefront approach for the whole frame. The herein considered optimizations over [9] correspond to: *i*) **better data packing**: 4 bytes per each 8×8 block of the frame, which store the prediction mode (*Pred. mode*) for the luma and chroma components; *ii*) **inter predicted neighboring**: additional conditions must be verified if neighboring blocks were inter predicted, while all blocks were intra predicted in [9].

After the IP kernel, the reconstruction frame is generated and DBF is applied to reduce the blocking artifact. It contains two filters: the horizontal filter and the vertical filter. These two filters can be independently processed for each 8×8 block, as shown in Fig. 2. For each sub-filter, two edges in the same direction can be processed at the same time. The **thread mapping** of the DBF has been optimized over [11] and [12], where an area of 256×8 samples is cooperatively processed by two warps within a thread block. When the horizontal filter starts, each warp maps to a set of 256×4 samples, where each thread maps to one horizontal edge of 8×4 samples. When the vertical filter starts, each warp maps to a set of 128×8 samples, where each thread maps to one vertical edge of 4×8 samples.

Finally, the SAO kernel is followed to complete the entire decoding procedure. Compared to [12], **vector processing**

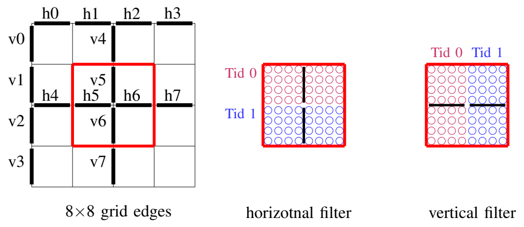


Fig. 2. Deblocking filter thread mappings, each two threads operate on an independent block of 8×8 samples.

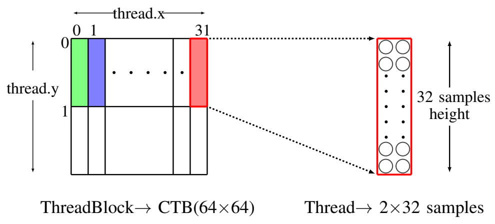


Fig. 3. SAO thread mappings, each thread block is mapped to a set of 64×64 samples and each thread operates on 2×32 samples.

operation is enabled by adopting a new thread mapping (see Fig. 3), where two warps are assigned for each thread block and each of them is responsible for 64×32 samples. Each thread is mapped to two samples (apart horizontally) and iterates its operations vertically 32 times. A vector operation with two elements is performed, making each thread able to process two samples in parallel.

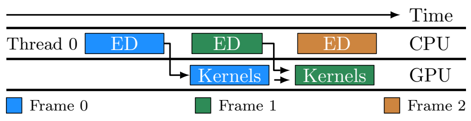
C. Pipelined Decoding on CPU and GPU

Figure 4 presents the proposed pipelined decoding scheme with two different CPU thread configurations: with a single CPU thread (Fig. 4a) and with two CPU threads (Fig. 4b). As can be noticed, the decoding tasks of the CPU and of the GPU are designed to be executed in parallel with a single or multiple CPU threads.

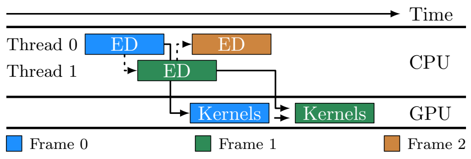
If a single CPU thread is used (see Fig. 4a), the *Thread 0* starts the entropy decoding of *Frame 0*, whose dependent frames are assumed to be completed. When the entropy decoding for *Frame 0* is concluded, the required kernel data is transferred to the GPU and the corresponding GPU kernels can be launched. After, the *Thread 0* starts the entropy decoding of *Frame 1*, since the kernel launches are asynchronous. Under this specific circumstance, the ED procedure of *Frame 1* will be overlapped with the GPU kernels of *Frame 0*. Although the kernels from *Frame 1* could start as soon as *Frame 1* ED procedure finishes, the kernels are stalled due to the motion compensation data dependencies, since, in this case, *Frame 0* is a reference frame of *Frame 1*. To ensure these data dependencies, the kernels of a specific frame are launched only if all its reference frames have been completely decoded. The overall process is repeated to *Frame 2* and so on. If two or more frames are independent from each other, their kernels can be executed concurrently according to the available GPU resources.

When multiple CPU threads are employed, the ED procedure of different frames can be performed in parallel (see Fig. 4b). The synchronization between the CPU decoding tasks is performed on Coding Tree Unit (CTU) line basis. In particular, a CPU thread starts entropy decoding a new CTU line of a frame only after the motion vectors of its co-located area on the previous frames have been entropy decoded.

In both configurations, kernels of a frame are queued in the same CUDA Stream, where each frame is assigned to its own CUDA Stream. This setup allows concurrently kernel execution of independent frames, where there is no data dependencies between them. No synchronization is needed between GPU kernels, since they are issued in order in a CUDA Stream.



(a) One CPU thread.



(b) Two CPU threads.

Fig. 4. Pipelined decoding scheme between CPU and GPU with multiple CPU Threads configurations.

Moreover, the GPU sets a flag in the CPU *Host Memory* as soon as a frame has been completely decoded through the *cudaStreamAddCallback* function. In this way, before launching the GPU kernels, all reference frames completion flags are checked, to ensure that the motion compensation kernel for the current frame can be started.

III. Experimental Evaluation

To evaluate the proposed CPU+GPU decoder approach, it was executed on a state-of-the-art heterogeneous platform equipped with an Intel Xeon E5-2699 CPU and an NVIDIA GeForce GTX Titan X GPU. The host CPU consists of 18 cores with both turbo boost and hyperthreading disabled. The Titan X GPU contains 24 stream multiprocessors (SMMs) and it is shipped with 12 gigabytes of GDDR5 memory. The two devices are connected via a PCIe 3.0× bus. The proposed decoder was compiled with GCC 4.8.4 using -O3 optimization level and was run on Kubuntu 14.04 distribution with Linux kernel 3.16. The CUDA kernels were developed with CUDA Toolkit 7.5 using version 352.63 of the graphics driver.

The test video sequences used for the conducted evaluation consist of two sets: the recommended JCT-VC *Class B* test set [13] and five 4K (2160p) sequences from EBU UHD-1 [14] (i.e., *Lupo_confetti*, *fountain_lady*, *rain_fruits*, *studio_dancer* and *waterfall_pan*), referred herein as *Class U*. All video sequences were encoded with the four recommended Quantization Parameter (QP) values (22, 27, 32 and 37) using the following two configurations: *All Intra* and *Random Access*. Moreover, neither Tiles nor Wavefront Parallel Processing (WPP) are employed, which represents the worst case for the CPU decoder in terms of parallelism and data locality. Since the proposed decoder supports both Main and Main10 profiles, *Class B* and *Class U* sequences were encoded with 8 and 10 bit depths, respectively.

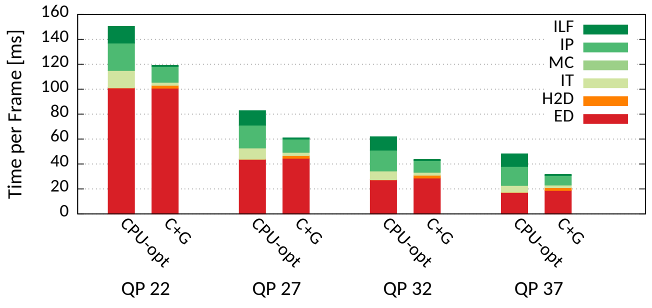
A. Sequential Execution Profile

In the first part of the evaluation, each GPU-based HEVC procedure is compared with a state-of-the-art HEVC decoder proposed in [3] (named as *CPU-opt*), by considering the overall frame processing time. Figure 5 presents the average decoding time per frame (in milliseconds) for both HEVC decoders, i.e., *CPU-opt* and the proposed CPU+GPU (*C+G*), when considering all QPs, configurations and classes. The decoding procedures are separated into six stages: *i*) Entropy Decoder (*ED*); *ii*) memory upload from the host to the device (*H2D*); *iii*) Inverse Transform (*IT*); *iv*) Motion Compensation (*MC*); *v*) Intra Prediction (*IP*); and *vi*) In-loop Filters (*ILF*).

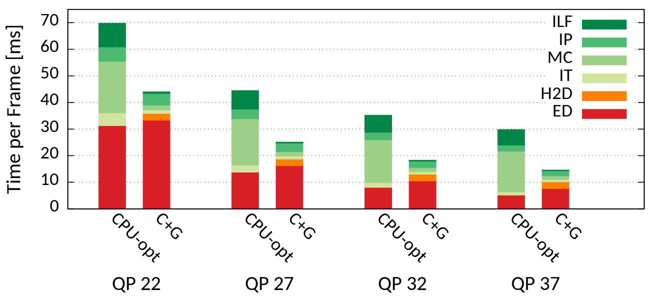
In general, with only one CPU core, the proposed *C+G* decoder outperforms the *CPU-opt* decoder in all considered cases. In particular, for *Class U* (see Fig. 5a and 5b) and QP=22 (worst case scenario), the proposed *C+G* decoder achieves 119.1 ms and 43.9 ms for *All Intra* and *Random Access* configuration, respectively, while the *CPU-opt* decoder executes in 150.3 ms and 69.8 ms. When higher QP values are considered, the average decoding time of both decoders are reduced. This is specially noted in the entropy decoder procedure. Nevertheless, all remaining decoding procedures of the proposed *C+G* approach are significantly reduced in all QP values when compared with *CPU-opt*. Furthermore, an average speedup of 1.4× for *All Intra* configuration and 1.8× for *Random Access* configuration are observed for the *C+G* decoder over *CPU-opt* for the *Class U* benchmark set. If the *ED* procedure is disregarded, the corresponding average speedups are as high as 2.9× and 5.1×. A higher performance is obtained for the *Random Access* configuration because the most time consuming module, *MC*, is significantly accelerated by the GPU kernel. Specifically, 21.5×, 1.2×, 11.0×, and 2.1× speedups are achieved for the *ILF*, *IP*, *MC*, and *IT* procedures, respectively, when processing the *Random Access* configuration of *Class U*.

On the other hand, the proposed *C+G* decoder also introduces two intrinsic overheads. First, the memory copy from CPU to GPU (*H2D*), which is required to transfer all the data under processing. Second, the *ED* procedure also includes a time overhead that is required to collect and pack the data for the GPU kernels. Such increased average processing time for *ED* and *H2D* in the *Random Access* configuration is 3× larger than in *All Intra* mode, since the *MC* data is not necessary for the latter configuration. These two overheads are negligible with lower QP values, since the overall average decoding time is high. However, the performance degradation is noticeable for videos sequences with higher QPs, due to the smaller workload. For example, when QP=22, the overheads are responsible for 1.6% and 10.3% of the total decoding time for *All Intra* and *Random Access* configuration, respectively. While, for QP=37, the overheads of the respective configurations account for 12.0% and 33.9% of the total decoding time.

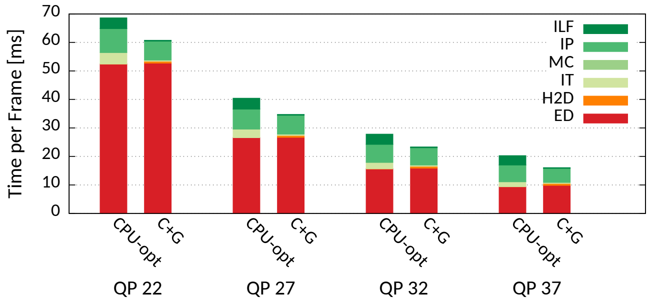
In what concerns *Class B* (see Fig. 5c and 5d), the ratio of the average *ED* processing time over the total decoding time is higher than the same ratio for the *Class U*. For example, even for the *Random Access* configuration, the *ED* procedure is responsible for up to 46.1% of the decoding time (on average), while this fraction is around 32.7% in *Class U*. As a consequence, the overall speedups of the *C+G* decoder over the *CPU-opt* are significantly reduced. Specifically, an average speedup of 1.4× is achieved for the *Random Access* configuration for the proposed decoder over the *CPU-opt* decoder.



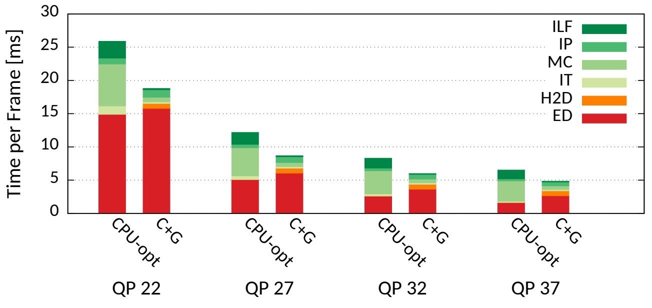
(a) Class U (3840×2160) – *All Intra* configuration.



(b) Class U (3840×2160) – *Random Access* configuration.



(c) Class B (1920×1080) – *All Intra* configuration.



(d) Class B (1920×1080) – *Random Access* configuration.

Fig. 5. Average decoding time per frame by the *CPU-opt* CPU decoder and by the proposed *C+G* decoder with different QP values for *Class U* and *Class B*.

B. Multi-core and CUDA Streams Performance Evaluation

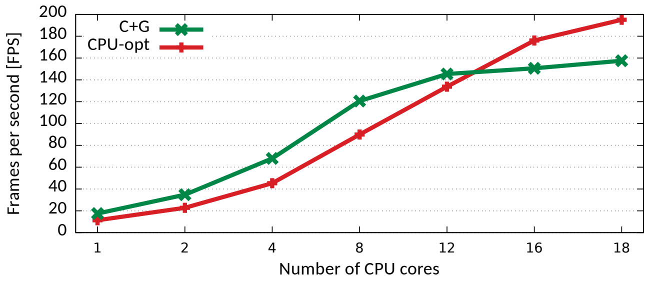
The overall performance of both the proposed *C+G* decoder and the *CPU-opt* state-of-the-art decoder are presented in Fig. 6. For both decoders, 32 frames can be simultaneously processed in parallel provided that there are enough CPU cores available and the frame dependencies are satisfied (e.g., all reference frames are decoded). For the *C+G* decoder, 32 CUDA Streams are employed since each frame has its own CUDA Stream. Each obtained frame rate value is an average that considers all QP values and tested sequences, when separated by class, configuration and number of CPU cores.

As expected, and independently of the considered class and configuration, the performance of both decoders rise when the number of CPU cores increases (see Fig. 6). Nevertheless, there are saturation points where both approaches no longer achieve better performance with a higher number of CPU cores. These saturation points are due to two main factors: *i*) restrictions of the available resources and *ii*) the entropy decoder ratio over the overall decoding time. In fact, although the *CPU-opt* decoder can theoretically process up to 18 frames in parallel with 18 CPU cores, the *C+G* decoder is limited by the amount of available resources and the number of concurrency of the GPU kernels. Moreover, the ratio of the entropy decoder processing time over the overall decoding time has a direct effect in the workload balance between CPU and GPU. For example, in the *Random Access* configuration of *Class U*, where the entropy decoder ratio is 32.7% with one CPU core, the saturation point of the *C+G* decoder is achieved with only 8 CPU cores (see Fig. 6b). On the other hand, since the entropy decoder ratio is higher for the *All Intra* configuration, the proposed decoder saturates around 12 CPU cores, where the GPU resources become the limiting factor.

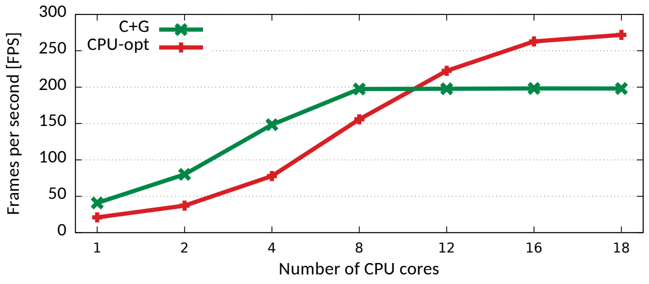
Although the *C+G* decoder saturates earlier in the *Random Access* configuration, the performance gain over the *CPU-opt* decoder is more significant for a lower number of CPU cores. This effect can be visualized when less than 8 CPU cores are used, where the gap between the two decoders is larger for *Random Access* than for *All Intra* configuration, as shown in Fig. 6a and 6b. For example, the *C+G* decoder delivers 80 FPS with 2 CPU cores and 148 FPS with 4 CPU cores, while the corresponding performance values of the *CPU-opt* decoder are only 37 FPS and 78 FPS, respectively. When executing with the same performance, the proposed *C+G* decoder requires a lower number of CPU cores, e.g., to achieve 80 FPS and 150 FPS, 4 and 8 cores are needed for the *CPU-opt* decoder.

For *Class B*, the proposed decoder outperforms the *CPU-opt* decoder in all tested configurations, because of a higher fraction of the sequential decoding part (in other words, a higher entropy decoder ratio). Only when a higher number of CPU cores is used does the performance of both decoders converge.

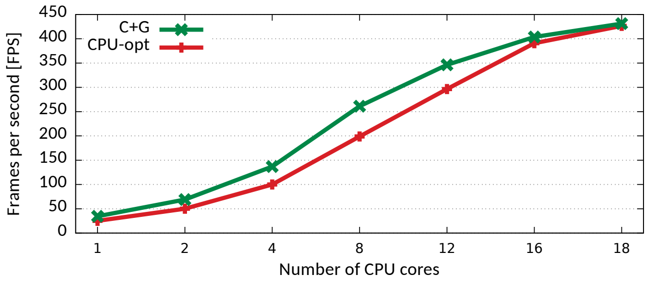
Speedups of 1.2× and 1.4× are obtained with 18 CPU cores for the *CPU-opt* decoder over the *C+G* decoder in *All Intra* and *Random Access* configuration, respectively. However, since most current desktop systems are equipped with less than 8 CPU cores, this decoder implementation can hardly be exploited by commonly used heterogeneous platforms. As such, since the proposed *C+G* decoding scheme always outperforms the state-of-the-art *CPU-opt* decoder for a lower number of



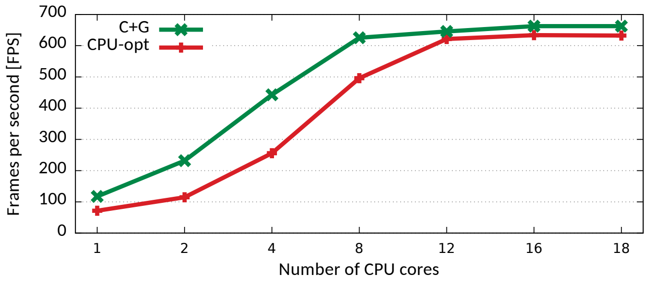
(a) Class U (3840×2160) – *All Intra* configuration.



(b) Class U (3840×2160) – *Random Access* configuration.



(c) Class B (1920×1080) – *All Intra* configuration.



(d) Class B (1920×1080) – *Random Access* configuration.

Fig. 6. Average frame rate obtained with the proposed *C+G* decoder and with the state-of-the-art *CPU-opt* decoder for *Class U* and *Class B*, when considering a variable number of CPU cores.

CPU cores, the proposed *C+G* decoder implementation is the most suited for nowadays heterogeneous platforms.

IV. Conclusions

In this paper, an efficient collaborative CPU+GPU decoding scheme is proposed for the HEVC standard. In particular, a task partition is proposed, where the entropy decoder procedure is preserved on the CPU side and all the remaining procedures are offloaded onto the GPU. In addition, frame-level parallelism is exploited between CPU and GPU, where CPU multi-core decoding is effectively implemented within the proposed heterogeneous CPU+GPU decoder.

The proposed decoder is evaluated on a work station desktop, where it is compared to a state-of-the-art CPU decoder [3]. The proposed decoding scheme provides both acceleration and offloading capabilities when CPU multi-core decoding is enabled, which can be employed in most of consumer level desktops. When considering Ultra HD 4K videos, the proposed CPU+GPU decoder achieves 80 FPS with only 2 cores and 150 FPS with 4 cores for *Random Access* configuration, which makes 4K@60p and 4K@120p possible with low CPU resources.

Acknowledgment

This work was supported by national funds through FCT (Fundação para a Ciência e a Tecnologia), under projects PTDC/EEI-ELC/3152/2012 and UID/CEC/50021/2013. It has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 688759 (LPGPU2). Diego F. de Souza also acknowledges FCT for the Ph.D. scholarship SFRH/BD/76285/2011.

References

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

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

[3] C. C. Chi, M. Alvarez-Mesa, B. Bross, B. Juurlink, and T. Schierl, "SIMD acceleration for HEVC decoding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 5, pp. 841–855, May 2015.

[4] NVIDIA, "NVIDIA PureVideo HD Technology," http://www.nvidia.com/page/purevideo_hd.html, Jul. 2007.

[5] ——, "NVIDIA Video Decoder (NVCUVID) Interface," https://developer.nvidia.com/nvidia-decoder, Nov. 2015.

[6] Ittiam system, "GPU Compute accelerated HEVC decoder," http://www.ittiam.com/products/software-ips/video/h265-hevc, Dec. 2014.

[7] Strongene, "Strongene OpenCL H.265/HEVC Decoder for Windows," http://www.strongene.com/en/downloads/downloadCenter.jsp, Jul. 2014.

[8] D. F. de Souza, N. Roma, and L. Sousa, "OpenCL parallelization of the HEVC de-quantization and inverse transform for heterogeneous platforms," in *Signal Processing Conference (EUSIPCO), 2014 Proceedings of the 22nd European*, Sept. 2014, pp. 755–759.

[9] D. F. de Souza, A. Ilic, N. Roma, and L. Sousa, "GPU-assisted HEVC intra decoder," *Journal of Real-Time Image Processing*, 2015.

[10] ——, "GPU acceleration of the HEVC decoder inter prediction module," in *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, Dec. 2015, pp. 1245–1249.

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

[12] D. F. de Souza, A. Ilic, N. Roma, and L. Sousa, "HEVC in-loop filters GPU parallelization in embedded systems," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, July 2015, pp. 123–130.

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

[14] "European broadcasting union," http://tech.ebu.ch/testsequences/uhd-1.