Signal Processing: Image Communication 62 (2018) 93–105

Contents lists available at ScienceDirect

Signal Processing: Image Communication

journal homepage: www.elsevier.com/locate/image

Highly parallel HEVC decoding for heterogeneous systems with CPU and GPU☆

Biao Wang [a,*], Diego Felix de Souza [b], Mauricio Alvarez-Mesa [c], Chi Ching Chi [c], Ben Juurlink [a], Aleksandar Ilić [b], Nuno Roma [b], Leonel Sousa [b]

[a] AES, Technische Universität Berlin, Berlin, Germany
[b] INESC-ID Lisboa, IST, Universidade de Lisboa, Lisbon, Portugal
[c] Spin Digital Video Technologies GmbH, Berlin, Germany

A B S T R A C T

The High Efficiency Video Coding HEVC standard provides a higher compression efficiency than other video coding standards but at the cost of an increased computational load, which makes hard to achieve real-time encoding/decoding for ultra high-resolution and high-quality video sequences. Graphics Processing Units GPU are known to provide massive processing capability for highly parallel and regular computing kernels, but not all HEVC decoding procedures are suited for GPU execution. Furthermore, if HEVC decoding is accelerated by GPUs, energy efficiency is another concern for heterogeneous CPU + GPU decoding. In this paper, a highly parallel HEVC decoder for heterogeneous CPU + GPU system is proposed. It exploits available parallelism in HEVC decoding on the CPU, GPU, and between the CPU and GPU devices simultaneously. On top of that, different workload balancing schemes can be selected according to the devoted CPU and GPU computing resources. Furthermore, an energy optimized solution is proposed by tuning GPU clock rates. Results show that the proposed decoder achieves better performance than the state-of-the-art CPU decoder, and the best performance among the workload balancing schemes depends on the available CPU and GPU computing resources. In particular, with an NVIDIA Titan X Maxwell GPU and an Intel Xeon E5-2699v3 CPU, the proposed decoder delivers 167 frames per second (fps) for Ultra HD 4K videos, when four CPU cores are used. Compared to the state-of-the-art CPU decoder using four CPU cores, the proposed decoder gains a speedup factor of 2.2×. When decoding performance is bounded by the CPU, a system wise energy reduction up to 36% is achieved by using fixed (and lower) GPU clocks, compared to the default dynamic clock settings on the GPU.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

The HEVC [1] standard represents the current state of the art in video coding technology. It provides 50% bitrate reduction with the same subjective quality when compared to H.264/MPEG-4 AVC (H.264) [2]. However, such improvement in bitrate compression is achieved at the cost of an increase in the computational requirements. Furthermore, the main applications of HEVC are delivery of Ultra High Definition (UHD) videos, including 4K and 8K. Emerging video quality enhancements on those UHD videos, such as High Dynamic Range (HDR) [3], Wide Color Gamut (WCG) [4], and High Frame Rate (HFR) [5], add even more computing requirements. Fortunately, HEVC has been designed with parallelism in mind. Coding tools such as Wavefront Parallel Processing (WPP) [6] and Tiles [7] have been added in order to take advantage of parallel architectures. Parallel processing for HEVC decoding has been analyzed and implemented in several homogeneous architectures. For example, the state-of-the-state CPU decoder [8] exploiting SIMD instructions and advanced multi-threading is able to decode 4K UHD video on contemporary desktop CPUs.

In addition to CPUs, modern computer systems often include Graphics Processing Units (GPUs), resulting into a class of heterogeneous architectures. Such heterogeneous CPU+GPU systems can potentially provide the computing capability needed for the next generation of UHD HEVC decoding. In order to extract the maximum performance, HEVC decoding has to be mapped appropriately onto such heterogeneous architectures. First, the decoding sub-modules need to be distributed properly between the CPU and GPU according to their computing characteristics. Second, the assigned decoding tasks on both the CPU and GPU sides have to be parallelized and optimized. Besides, the decoding operations between the CPU and GPU requires efficient communication and pipeline consideration. Finally, multiple load balancing schemes are desired when the available computing resource changes on the CPU and GPU devices.

☆ Extension of Conference Paper: "Efficient HEVC decoder for heterogeneous CPU with GPU systems," 2016 IEEE 18th International Workshop on Multimedia Signal Processing (MMSP), Montreal, QC, 2016, pp. 1–6. The paper is extended with (i) additional workload balancing scheme (ii) integrated energy measurement module for CPU and GPU devices. (iii) energy optimized decoding for heterogeneous system by setting the GPU at fixed clock rates.

* Corresponding author.
E-mail address: biaowang@win.tu-berlin.de (B. Wang).

https://doi.org/10.1016/j.image.2017.12.009
Received 2 August 2017; Received in revised form 16 December 2017; Accepted 21 December 2017
Available online 30 December 2017
0923-5965/© 2017 Elsevier B.V. All rights reserved.

In this paper, a highly parallel design of the HEVC decoding for heterogeneous CPU+GPU systems is proposed. The HEVC procedures have been redesigned so that the sequential entropy decoding stage is executed on the CPU, while the remaining parallel kernels are offloaded onto the GPU. In addition to the data parallelism exploited on the GPU, the available wavefront parallelism for the CPU task is also exploited. Furthermore, the decoding tasks on the CPU and GPU have been designed to execute in a pipelined fashion, with an efficient one-direction data transfer. On top of the parallel design, different workload balancing strategies have been developed, in order to deliver the best performance according to the exploited set of computation resources. Finally, an energy measurement solution has been integrated within the heterogeneous CPU+GPU decoder, with which the energy efficiency of the proposed decoder is evaluated and analyzed. To summarize, the contributions of this paper are the following.

- A highly parallel HEVC decoder for heterogeneous CPU+GPU systems is proposed, where multiple levels of parallelism are exploited simultaneously. On the CPU, it exploits both the intra- and inter-frame parallelism. On the GPU, it allows concurrent kernel execution, in addition to the data-level parallelism within a frame. Between the CPU and GPU devices, pipelining is also exploited at the frame level.
- On top of the proposed design, different workload balancing schemes are implemented, in order to find the most efficient workload distribution depending on the available CPU and GPU computing resources. In particular, with an NVIDIA Titan X Maxwell GPU and an Intel Xeon E5-2699v3 CPU, average frame rates of 167 frames per second (fps) and 60 fps are achieved for 4K and 8K videos, respectively.
- An energy efficiency analysis is performed for the proposed CPU+GPU decoder with the integrated energy measurement module. Compared to the default clock settings of the GPU, the energy efficiency of the heterogeneous decoding can be further optimized by tuning GPU clocks, with a system wise energy reduction up to 36%.

This paper is organized as follows. Section 2 discusses the related work. Section 3 provides a parallelism analysis for the HEVC decoding. Section 4 elaborates on the proposed decoding design. Section 5 describes the energy measurement module for the CPU and GPU devices. In Section 6, the performance and energy efficiency results of the proposed heterogeneous CPU+GPU decoding are presented and analyzed. Finally, the conclusions are drawn in Section 7.

2. Related work

This section provides a review of HEVC decoding implementations on different architectures, such as CPUs, GPUs, and dedicated hardwares. Furthermore, a brief review of energy optimized GPU computing and video decoding is presented.

On the general-purpose CPU processor, the open-source HEVC Test Model (HM) [9] is often used as a baseline. However, HM was developed mainly for validation of the HEVC standard, being not optimized for real-time decoding. In contrast, an optimized decoder with Single Instruction, Multiple Data (SIMD) and multi-threading was developed in [10]. On an Intel i7-2600 3.4 GHz quad-core CPU, the optimized decoder delivers 40–75 fps for 4K videos. Another SIMD and multi-threaded decoder with additional memory optimizations was proposed in [8]. This decoder delivers 134.9 fps on an Intel i7-4770S 3.1 GHz quad-core CPU for 4K videos.

Regarding software-based GPU acceleration for video decoding, most of previous work targets only single HEVC decoding modules, such as Inverse Transform (IT) in [11,12], Motion Compensation (MC) in [13], Intra Prediction (IP) in [14], Deblocking Filter (DBF) in [15,16], and in-loop filters in [17]. In particular, Souza et al. [18] presented a set of optimized GPU kernels, where they optimized and integrated individual HEVC modules. The set of GPU kernels, however, did not cover all HEVC decoding modules, i.e., the Entropy Decoding (ED) is excluded. Experimental results show these GPU-based kernels (i.e. excluding ED) deliver a frame rate of 145 fps for the 4K videos using an NVIDIA TITAN X Maxwell GPU.

Apart from the above software approaches, hardware implementations of HEVC decoding have been proposed as well. Abeydeera et al. [19] presented an HEVC decoder based on Field-Programmable Gate Array (FPGA). With a Xilinx Zynq 7045 FPGA, their decoder delivers 30 fps for 4K videos. Tikekar et al. [20] implemented an Application-Specific Integrated Circuit (ASIC) in 40 nm CMOS technology with a set of architectural optimizations. Their ASIC decoder is also able to decode at 30 fps for 4K videos. In addition, modern commercial GPUs often provide dedicated hardware accelerators for video decoding, such as NVIDIA's PureVideo [21], Intel's Quick Sync Video [22], and AMD's Unified Video Decoder [23]. Most of the hardware-based HEVC accelerators, however, are limited to specific architectures and further constrain their support to certain HEVC profiles. For example, NVIDIA adds complete HEVC hardware acceleration until GM206 architectures, and constrains its decoding capability to HEVC Main profile up to Level 5.1 [24]. In contrast, the set of software-based solutions that are adopted by this paper can provide HEVC real-time decoding capabilities for nowadays heterogeneous systems, even when the considered GPUs are not equipped with HEVC hardware acceleration.

When considering energy optimized GPU computing/video decoding, Mei et al. [25] exploited the impact of up-to-date GPU Dynamic Voltage and Frequency Scaling (DVFS) [26] techniques on the application performance, power consumption, and energy conservation. Their results showed that the energy saving not only depends on GPU architectures but also characteristic of GPU applications. For video decoding application, two approaches were exploited in [27] for achieving low-power and high-efficiency real-time video decoding on different CPU architectures. Results showed that the "exploiting slack" approach is more power efficient than the "race to idle" strategy on all evaluated CPUs. However, both of the above studies investigated energy optimization strategies only on homogeneous architectures, either on CPUs or on GPUs.

Compared to the software-based approaches, in this paper a complete HEVC decoder for heterogeneous system consisting of CPU and GPU devices is presented. We exploit available parallelism on the CPU, GPU and between the CPU and GPU devices simultaneously. Furthermore, different workload distributions between the CPU and GPU devices are implemented, and hence the proposed decoder can achieve the best performance under different computing resource configurations. Finally, we analyze the energy efficiency of HEVC decoding on heterogeneous architectures. By tuning clocks of the more power hungry GPU device, a system wise energy consumption is reduced by up to 36%, when compared to the default GPU clock settings.

3. Parallelism analysis for the HEVC decoding

This section starts with the discussion of the parallelization opportunities within the HEVC decoding that were exploited in the proposed design. Afterwards, an analysis of the parallelism within all decoding tasks is performed by considering GPU architectures.

3.1. Parallel decoding in the HEVC standard

There are two forms of parallelism available in the HEVC decoding: intra- and inter-frame parallelism. The intra-frame parallelism is available when WPP [6] is enabled at the encoder side. WPP allows multiple threads to decode several lines of Coding Tree Units (CTUs) in parallel, as shown in Fig. 1. Each decoding thread processes CTUs in the same row from left to right. Due to data dependencies, each CTU can only be decoded if its top right CTU is decoded, which leaves a distance of two CTUs between neighboring threads. To fulfill this dependency, WPP

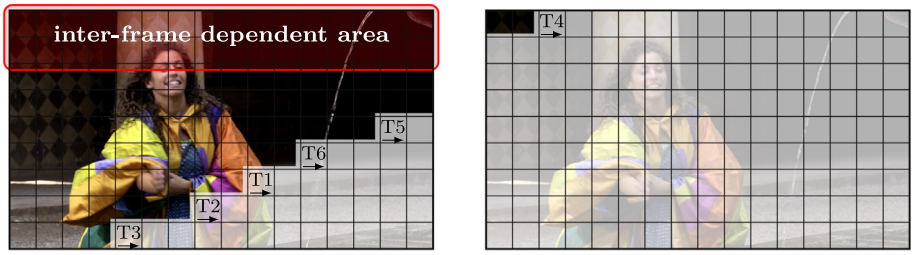


Fig. 1. Intra- and inter-frame parallelism exploited in HEVC decoding. Each cell in the grid of a frame represents a CTU.

Table 1
Qualitative analysis of the HEVC decoding stages in terms of data parallelism and branch divergence.

Decoding stage features	HEVC decoding stages					
	ED	IT	MC	IP	DBF	SAO
Data parallelism	very low	high	high	medium	high	very high
Branch divergence	very high	low	low	medium	low	very low

suffers from a low thread utilization at the start and the end of each frame, when only a single frame's decoding task is considered.

Such inefficiency can be relieved by also exploiting inter-frame parallelism when multiple *frames-in-flight (FiF)* are available, where CTUs from different frames can be decoded in parallel. As it is shown in Fig. 1, the decoding thread (T4) no longer remains idle as the workload from the next frame can be scheduled. In addition, the decoding task for CTUs in the next frame does not have to wait for the completion of the reference frame, but it can start as long as its dependent area in the previous frame is decoded. This strategy that exploits the inter-frame parallelism and relieves the WPP inefficiency was firstly proposed in [28], termed as the *Overlapped Wavefront (OWF)* approach.

In addition to WPP, slices and tiles are the other two parallel coding tools in HEVC that can increase the intra-frame parallelism. By dividing a frame into multiple independent slices/tiles, the decoding task for each slice/tile can be processed in parallel. Comparing all methods, WPP (OWF) has been proven the most efficient way to exploit the parallelism in the HEVC decoding, as evaluated in [28]. When WPP, tiles, and slices are all disabled, only inter-frame parallelism can be exploited.

3.2. Suitability of GPU acceleration for HEVC decoding

HEVC decoding can be divided into six steps: Entropy Decoding (ED), Inverse Transform (IT), Motion Compensation (MC), Intra-Prediction (IP), Deblocking Filter (DBF), and Sample Adaptive Offset (SAO) filter. However, not all of these decoding kernels are suitable for GPU architectures. Only kernels that exhibit a high degree of data level parallelism and a low degree of branch divergence can lead efficient GPU execution.

Table 1 presents a qualitative analysis for the HEVC decoding kernels, when they are performed at the frame level. In particular, the ED exposes little data level parallelism and is highly divergent due to its bit-level dependency in the decoding path. The IT can be performed independently for each transform block in a frame, where thousands of transform blocks are available. Such independent block processing can also be applied for the decoding procedures of MC, DBF, and SAO. However, the IP cannot be applied in parallel for all blocks within a frame, due to its block-level data dependency. For one block's prediction, depending on its prediction mode, the samples of other blocks from the top-right, top, top-left, left, and bottom-left directions might need to be predicted first, as exemplified by one 4×4 block's prediction in Fig. 2. Hence, the number of blocks that can be predicted in parallel in IP is reduced. Meanwhile, the IP has a total of 35 prediction modes, while other kernels, except ED, in general exhibit a low execution divergence.

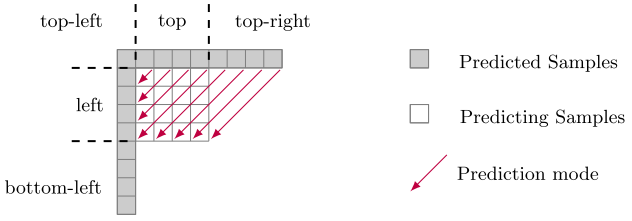


Fig. 2. The potential dependent samples in HEVC intra prediction, exemplified by a 4×4 block with one prediction mode.

4. Proposed decoding design for heterogeneous systems with CPU and GPU

In this section, a general design for parallel HEVC decoding on heterogeneous platforms is presented first. After that, different workload balancing schemes on top of the proposed design is elaborated. With them, a more balanced workload distribution can be achieved for different input sequences, according to the available computing resources on the CPU and GPU devices.

4.1. HEVC decoding task distribution for heterogeneous CPU+GPU systems

Based on the decoding procedure analysis in Section 3.2, a purely task-based workload distribution between CPU and GPU is proposed, as shown in Fig. 3. For every frame, the ED task is executed on the CPU, due to its sequential and irregular processing pattern, while the remaining decoding procedures are offloaded onto the GPU. The tasks targeted for the GPU are sometimes referred to together as *reconstruction* kernels, since they are responsible for reconstructing the frames.

Among the reconstruction kernels, the IP has a medium level of data parallelism and branch divergence, which can be executed either on the CPU or the GPU. Executing IP on the CPU, however, will introduce two extra data transfers between the CPU and the GPU, which are a well-known source of bottleneck for heterogeneous CPU+GPU computing. Due to data dependency, the reconstructed samples derived from the IT and MC on the GPU have to be firstly transferred back to the CPU, as the input for the IP. After the IP is processed on the CPU, the reconstructed samples from the intra-predicted blocks need to be uploaded to the GPU again, as the input for the DBF performed on the GPU. In contrast, we assign the IP on the GPU to reduce the data dependency between the CPU and GPU devices. As a result, the data transfer between the CPU and the GPU is minimized to once only, as shown in Fig. 3.

In our baseline multi-core CPU decoder [8], all decoding procedures are applied at block-level in order to exploit data locality. In the

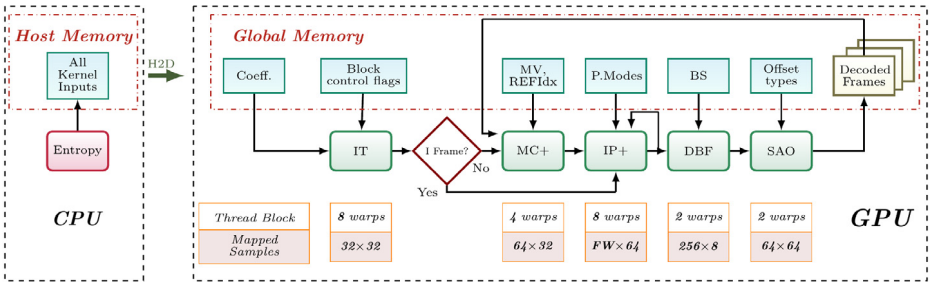


Fig. 3. Work flow overview of task based partition for CPU+GPU decoding on one specific frame. The entropy decoding module is assigned on the CPU and the remaining kernels are offloaded onto the GPU. Thread block level mapping is presented at the bottom, within the GPU block.

herein proposed CPU+GPU decoding, however, reconstruction kernels are applied at frame-level, in order to increase the data parallelism for GPU execution. Hence, from a high-level perspective, three steps are performed based on the baseline decoder to achieve the workload distribution in Fig. 3. First, the ED is decoupled from the decoding loop that fuses all decoding procedures. Second, the reconstruction kernels are changed from the block-level processing to the frame-level processing. Third, all reconstruction kernels are parallelized for GPU execution.

After this redesign for heterogeneous CPU+GPU processing, the decoding task for a single frame is performed as follows. First, while the ED is executed on the CPU, the input data for the reconstruction kernels is collected at the frame level. The collected data includes the coefficients (Coeff.) and block control flags for IT, the motion vector (MV) and the reference index (RefIdx) for MC, the prediction modes (P. Modes) for IP, the boundary strength (BS) for DBF, and the offset types for SAO, as shown on the top in Fig. 3. After an entire frame is processed by the ED, the collected data is transferred from the Host to Device (labeled as *H2D*). As soon as such data is transferred to GPU *Global Memory*, the reconstruction kernels are launched in the following order to fulfill the HEVC standard specifications: IT, MC, IP, DBF, and SAO. Along with prediction kernels (i.e., MC and IP), the suffix "+" indicates that the reconstruction output (predicted samples + residual data) is computed within the GPU kernel. After all GPU kernels have been executed, the decoding task for one frame is complete. The decoded frames can remain in the GPU global memory as the reference frame for the MC, which is also performed on the GPU. In this way, the data dependency of MC is addressed completely on the GPU, and the decoded frames do not have to be transferred back to the CPU, as shown in Fig. 3.

4.1.1. Parallel decoding on the CPU and GPU devices

On the CPU side, when WPP and multiple FiFs are available, the ED task exploits both intra- and inter-frame parallelism with the OWF approach, as shown in Fig. 1. For the entropy decoding of one frame, multiple threads are allowed to process the frame in parallel, each corresponding to one row of CTUs. Meanwhile, the ED task can start across multiple frames. The motion vector prediction that integrated within the ED stage can start as long as its reference area (instead of the complete frame) is ready. When the CTU rows from the same frame and other frames are both available, the ED processes the CTU lines that come from the same frame first, in order to minimize the frame-level decoding latency.

On the GPU side, all reconstruction kernels have been parallelized using Compute Unified Device Architecture (CUDA) [29]. In CUDA, the threads are organized in three bottom-up levels: *thread*, *thread block*, and *grid*. Moreover, the threads are executed in groups of 32 threads, termed as *warps*. Hence, the thread block size is usually configured as multiple warps to avoid thread waste. Herein, all kernels are applied on the frame basis, and the thread mapping at the thread block level is summarized per kernel at the bottom of Fig. 3. The selected thread block configurations are derived either by tuning thread block sizes (such as MC and IP, as presented in [18]), or by further optimizing data mapping of the thread block (such as DBF and SAO, as presented in [30]).

For the IT, 8 warps are configured for processing a block of 32×32 samples. When there are multiple transform blocks within the mapped thread block, the warps are assigned according to the transform block partition. The thread block for MC is composed of 4 warps, and they are assigned to perform the inter prediction of a block consisting of 64×32 samples. In MC, the on-chip shared memory is used to buffer the reference samples that will be further used, thus reducing the required memory bandwidth to the global memory. The IP kernel is performed after the MC due to the intrinsic data dependencies on its neighboring predicted samples. In total 8 warps are allocated for one thread block in IP, and they are responsible for an area in a frame width with a height of 64 samples ($FW \times 64$), thus accomplishing a wavefront approach for the whole frame. For the in-loop filters, the DBF and SAO, each thread block contains 2 warps, but they are assigned to a block of 256×8 and 64×64 samples, respectively. The more detailed parallelization strategies for the IT, MC, IP, and the in-loop filters (i.e. DBF and SAO) have been elaborated in [11,13,31], and [30], respectively.

For the decoding tasks on the GPU, besides the frame-level data parallelism exploited by CUDA kernels, inter-frame parallelism is also exploited when multiple FiFs are configured. Fig. 4 presents an example with two independent FiFs. For each frame, its corresponding GPU kernels are issued in the same *CUDA stream*: a sequence of GPU operations that execute in issue order. In the proposed design, one CUDA stream is created per each frame and all GPU operations are issued asynchronously, which allows a concurrent execution on the GPU for different CUDA streams [32]. Two types of concurrency are exploited on the GPU. First, the host to device memory copy (*H2D*) is performed by the copy engine on the GPU, which can be overlapped with the kernel execution from other frames. Second, if the GPU has idle computing resources when executing one given kernel, the kernels from other streams can be concurrently executed. For example, the execution of IP is overlapped with one other kernel for most of the time, since its limited amount of parallelism leads to a low utilization of the GPU resources. Kernel concurrency is also observed in the execution of SAO (from stream 1) and DBF (from stream 2), but for another reason. Both SAO and DBF expose massive parallelism but they are lightweight for a powerful GPU, and hence can be concurrently executed.

4.1.2. Pipelined decoding between the CPU and the GPU

Besides the parallelism exploited on the CPU and GPU devices, pipelining is exploited as well in the proposed design. Fig. 5 presents an example of pipelined execution between the CPU and the GPU when multiple FiFs and WPP are available. In total three threads are configured, together with three FiF, each labeled with a different color that represents the associated frame buffer. For each frame, the task assigned to the CPU is labeled in the form *Frame No.: ED*, while the reconstruction kernels assigned to the GPU are labeled as *Frame No.: Rec*. For the sake of easier explanation, it is assumed that every two frames (Frame 1 and 2, 3 and 4, etc.) can be decoded independently. Moreover, the first frame in the independent frame pair is assumed as the one (and the only one) reference frame of the second frame. Hence, the MC of Frame 2 shall wait until Frame 1 is completely decoded, Frame 4 shall wait for Frame 3, and so on.

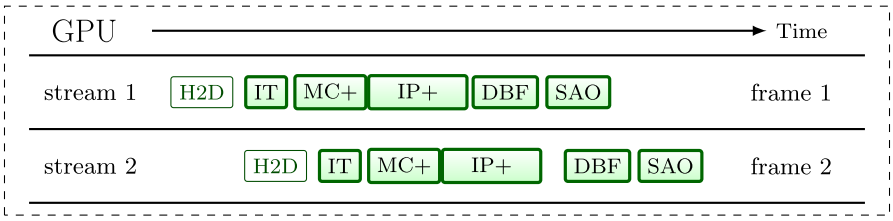


Fig. 4. Parallel decoding on the GPU with two independent frames in flight (and hence two CUDA streams), assuming that the considered GPU has enough resources to execute multiple kernels concurrently.

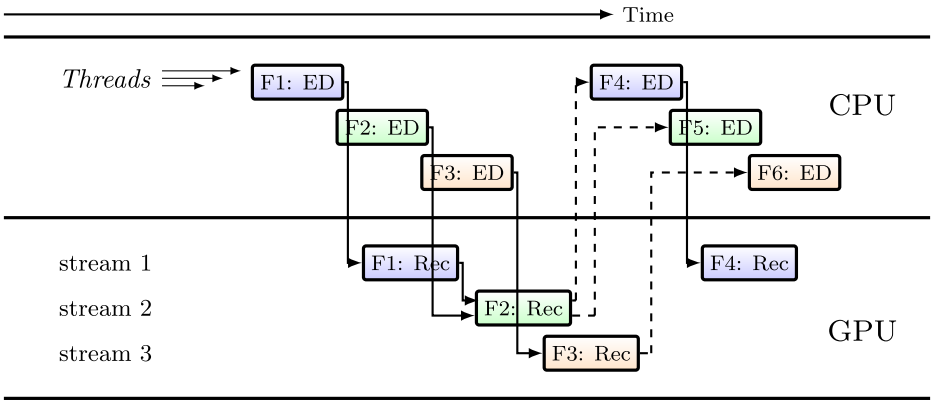


Fig. 5. Pipelined decoding between the CPU and the GPU with three frames in flight and three threads, assuming that the considered GPU has enough resources to execute multiple kernels concurrently.

The entire decoding process starts on the CPU, with entropy decoding of Frame 1 (*F1: ED*). Since the decoding task within the same frame has a higher priority, each thread on the CPU takes a row of CTUs in Frame 1 and decodes the frame in a wavefront scheme. When they are approaching the end of the frame, the CTU rows of Frame 2 are scheduled for these CPU threads. Hence, the ED tasks at the end of Frame 1 and the beginning of Frame 2 are decoded in parallel. If the WPP is disabled, then the configured three threads will spread over the available FiF, and hence the decoding of Frame 2 and Frame 3 will start sooner. After the CPU accomplishes all the entropy decoding of Frame 1, the reconstruction kernel inputs are transferred to the GPU side, and hence the GPU kernels of Frame 1 can be executed. Meanwhile, the ED task of Frame 2 is also processed on the CPU side in a wavefront approach. When the CPU complete the decoding of Frame 2, however, due to the motion compensation data dependency, GPU kernels cannot start until Frame 1 is completed decoded. Therefore, no concurrent GPU execution is observed between Frame 1 and 2. However, the GPU kernels on Frame 2 and 3 are independent of each other, and hence concurrent execution is exploited between them. When Frame 2 is completely decoded on the GPU, the frame buffers for Frame 2 and its reference frame (Frame 1) are freed. The freed frame buffers can accommodate new frames (Frame 4, 5), and the overall process is repeated.

The synchronization between the CPU and the GPU is performed as follows. When the decoding task on the CPU is completed for a given frame, a flag is set for this frame's GPU decoding task. GPU kernels will not be scheduled without this flag set. Furthermore, all reference frames of the current frame are checked before launching its GPU kernels, to assure that its motion compensation dependency is fulfilled. Finally, after all GPU kernel launches of one frame, the callback function *cudaStreamAddCallback* is appended in the same CUDA stream. As soon as all kernels are complete, this callback function is activated by the CUDA runtime, informing the CPU to start decoding a new frame.

4.2. Different workload balancing schemes

Depending on the ratio of computational power between the CPU and the GPU (e.g., the number of CPU cores/the number of GPU cores), different workload distribution must be employed in order to achieve better performance. If more GPU than CPU computing resources are available, it is better to submit all frames to the GPU for reconstruction kernels' execution. However, if more CPU than GPU computing resources are available, the GPU might not be able to process all the frames at the desired rate and can become the bottleneck. In this case, it is better to send fewer frames for reconstruction to the GPU and reconstruct more in the CPU.

The presented decoding scheme in Section 4.1, termed as scheme I afterwards, divides the workload between the CPU and the GPU based only on the decoding procedures. For a given video with lightweight entropy decoding workload, this task-based distribution can lead to workload imbalance when a high number of CPU cores are employed. To mitigate this problem, fewer frames shall be sent to the GPU for the reconstruction tasks. Instead, these reconstruction tasks are executed on the CPU, and hence a better workload balancing between the CPU and GPU devices is achieved.

However, one pending issue is the selection of frames that do not offload the reconstruction kernels onto the GPU anymore. One option to accomplish the new frame distribution is based on reference and non-reference frames. A reference frame is used by other frames as the input for motion compensation, while a non-reference frame is not used by any other frames. Fig. 6 presents an example of reference and non-reference frames in a Group Of Pictures (GOP) with a size of 8 frames. The numbers labeled within the frames represent their displaying order, and the frame-level dependencies between these frames are indicated by the arrows. For example, frame 4 can only be decoded after the completion of frame 0 and 8.

In the newly proposed workload distribution scheme, termed as scheme II afterwards, all decoding tasks for the reference frames are preserved on the CPU, and the corresponding GPU kernels are disabled. Meanwhile, the CPU decodes these reference frames at the CTU line level, in order to exploit both inter- and intra-parallelism, as presented in Fig. 1. The workload distribution for non-reference frames remains the same as presented in Fig. 3, i.e., ED is assigned on the CPU and other kernels are assigned on the GPU. In this way, no memory transfer

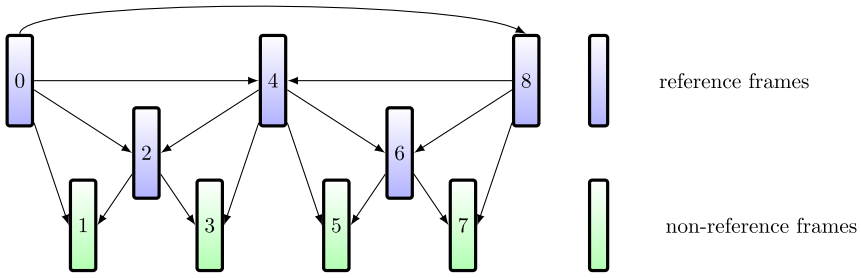


Fig. 6. Inter-frame dependency in a group of pictures (GOP) with a size of 8 frames.

Table 2
Workload distribution in decoding scheme I and II.

Frame types	Scheme I		Scheme II	
	Entropy	Others	Entropy	Others
Non-reference frames	CPU	GPU	CPU	GPU
Reference frames				CPU

from the GPU to the CPU is required because the dependency between the reference frames on the CPU are self-contained, and dependency between the reference frames and non-reference frames can be addressed by transferring the decoded reference frames from the CPU to the GPU. The main differences in the considered workload distributions of the decoding scheme I and II are summarized in Table 2. The proposed decoding scheme II applies for all input sequences using hierarchical GOP structures, which is a common choice when encoding videos for consumer applications.

In addition to avoiding extra direction of memory copy (from the GPU to the CPU), the decoding scheme II also brings other benefits. First, the inter-frame parallelism exploited by the GPU is usually limited by the frame-level motion compensation dependencies. However, under the new decoding scheme, such dependency will not occur, since non-reference frames are independent of each other and can be processed in parallel by the GPU. Moreover, the tasks on the GPU are synchronized at the frame level, while the tasks on the CPU are synchronized at the CTU line level. Hence, the GPU can start a new decoding task only when the entire reference frame is completed. In contrast, the finer synchronization granularity on the CPU allows the decoding task to start without the completion of the entire reference frame, thus improving overall performance scalability.

5. Energy measurement for heterogeneous CPU+GPU decoding

In order to analyze the energy efficiency of heterogeneous CPU+GPU decoding, an energy measurement module is developed and integrated within the proposed decoder. The energy measurement module consists of two parts: the one used for measuring energy of Intel CPUs using the Running Average Power Limit (RAPL) [33] interface, and the other for measuring NVIDIA GPUs using the NVIDIA Management Library (NVML) [34].

5.1. Energy measurement of Intel CPUs

Since Sandy Bridge microarchitecture, RAPL interface is implemented to monitor and control the power consumptions of Intel CPUs. Its internal circuitry can estimate current energy usage based on a model driven by hardware counters, temperature, and leakage information [35]. The results of this power model have been validated with high accuracy [36] and are available to users via a set of Machine Specific Registers (MSRs). For fine grained report and control, RAPL interface provides sensors that allow measuring energy of the CPU-level components, referred to as a RAPL domain. In total there are four RAPL domains, namely, *package*, *pp0*, *pp1*, and *DRAM*. Package domain reports power consumption of the whole CPU package, pp0 and pp1 domains respectively refer to the power consumed by the core and uncore devices, and DRAM domain provides the power consumption of memory controller. These domains, however, are not always available. The domain availability depends on the processor models [37]. The server processor used in this paper (i.e. Haswell-EP Xeon E5-2699v3), for instance, only supports the package and DRAM domains [38].

For package domain, the energy usage can be read from the MSR register MSR_PKG_ENERGY_STATUS, and the energy usage of DRAM domain is read from the MSR_DRAM_ENERGY_STATUS register. These two registers are read-only and the energy value stored in them is updated every 1 ms [37]. The raw energy values from these two registers are counted in *energy units*, which are defined in register MSR_RAPL_POWER_UNIT.

The energy of CPU is measured by putting two reads for the package and DRAM domains at the beginning and the end of the decoding process. The consumed energy for each domain is then obtained by subtracting the value from the two reads, with overflow taken into account. The subtracted energy values for package and DRAM domain are then multiplied by their corresponding energy units, and finally added together.

5.2. Energy measurement of NVIDIA GPUs

The NVML library provides C-based Application Programming Interfaces (APIs) for monitoring and managing various states of NVIDIA GPUs [34]. These states include power, clocks of memory and Streaming Multiprocessors (SMs), performance state, temperature, fan speed, etc.

In contrast to RAPL interface, the NVML library does not provide a direct interface to read the energy usage of GPUs. To address this issue, the energy of the GPU device is estimated by the multiplication of power and execution time. A power sampling thread is forked at the beginning and joined at the end of the decoding process. It reads the current power consumption by *nvmlDeviceGetPowerUsage* API at a frequency of 62.5 Hz, which is the maximum power measurement frequency according to [39]. Then, the sampled power values are averaged and multiplied by the execution time. In order to understand the power management of NVIDIA GPUs better, we also query the performance state and clocks of memory and SM within the sample thread.

In addition to APIs to query state of GPUs, NVML also provides APIs to modify the settings of GPU execution, such as the clocks of graphics and memory. These APIs provide a way to limit the GPU power consumption by changing its operating clocks. The GPU power includes static and dynamic components. The static power is due to current sources and to leakage current when a transistor is nominally off. The dynamic power conventionally accounts for the majority of the total power, and can be determined by Eq. (1):

$$P_{dynamic} = aCV^2 f \tag{1}$$

where a represents the activity factor, C denotes the total capacitance, V is the supply voltage, and f stands for the operating frequency [40]. The higher clock rates of graphics and memory allow GPU to consume more power, and vice versa.

The default power management approach of NVIDIA GPUs is auto boost mode with DVFS, namely, changing the clock/voltage dynamically during the applications' runtime. This strategy, however, might not be the optimal choice of power management in the scenario of heterogeneous CPU+GPU HEVC decoding. To exploit the optimization opportunities of energy efficiency, the GPU clock setting utility is implemented and integrated within the decoder, with which the GPU can perform at the specified clocks. The clock setting utility is achieved in two steps. First, the auto boost mode needs to be disabled by calling *nvmlDeviceSetAutoBoostedClocksEnabled* with NVML_FEATURE_DISABLED as the parameter. Second, the memory and graphic clocks can be set by calling *nvmlDeviceSetApplicationsClocks*, with the desired memory and graphic clocks.

6. Experimental results

To evaluate the performance and energy efficiency of the proposed CPU+GPU decoder, it was executed on a system equipped with an Intel Xeon CPU and an NVIDIA GTX Titan X Maxwell GPU. The host CPU Xeon E5-2699v3 integrates 18 physical cores and has a Thermal Design Power (TDP) of 145 Watt (W). It was configured with both turbo boost and hyperthreading disabled. The device GPU GTX TITAN X has 3072 CUDA cores that work between 1 to 1.2 GHz when auto boost is enabled. It has a power limit of 250 W and is configured with auto boost enabled unless stated otherwise. The host and the device are connected via a PCIe bus 3.0 × 16. Table 3 summarizes the specifications of the test platform. The proposed CPU+GPU decoder was compiled with GCC 4.8.4 compiler with −O3 optimization level and ran on Kubuntu 14.04 Linux distribution using kernel 3.16. GPU kernels were developed using CUDA Toolkit 7.5, with graphic driver version 352.63.

The proposed heterogeneous decoder fully supports the HEVC Main10 profile [41]. Five 4K sequences from EBU UHD-1 sequence set [42] and two 8K sequences from NHK [43] were encoded with four distinct QP values. Their corresponding bitrates are presented in Table 4. Each 4K sequence consists of 500 frames with a GOP size of 8 frames, while the 8K sequences are 3600 frames each and with a GOP size of 16 frames. Both 4K and 8K videos were encoded with random access 10-bit configuration under 4:2:0 chroma sub-sampling format, with WPP enabled. For a given set of videos (e.g., belonging to the same QP or the same resolution, such as 4K and 8K), the frame rate was measured as the total number of frames of the test video sequences divided by the corresponding decoding time. Unless otherwise stated, the results that will be presented below are based on this set of videos (encoded with random access configuration, GOP size 8, rate control off).

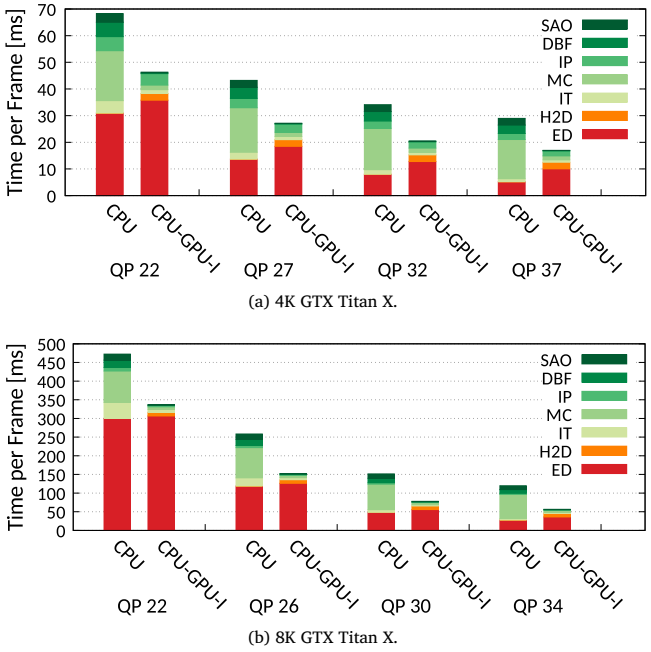
The experimental results are presented in two sub-sections, with performance results presented first and then the energy efficiency evaluation. Moreover, the proposed CPU+GPU decoder was compared against the CPU decoder in [8], since it presents complete HEVC decoding performance and represents the state-of-the-art software decoder.

6.1. Performance results

A comprehensive performance evaluation has been conducted for the proposed decoding schemes. Firstly, the single-threaded CPU+GPU decoding performance is presented to evaluate the impact of the GPU kernel acceleration. Afterwards, the multi-threaded CPU+GPU decoding performance is evaluated, followed by an evaluation of the potential peak performance and a bottleneck analysis.

6.1.1. CPU+GPU decoding time profiling

The decoding time breakdown per frame of the baseline *CPU* decoder and of the proposed CPU+GPU decoding scheme I (*CPU–GPU-I*) are presented in Fig. 7. Only a single CPU core is employed in both decoders, and they are compared against each other across different QP values. Their decoding time is divided into seven stages: *ED*, *H2D*, *IT*, *MC*, *IP*, *DBF*, and *SAO*.



(a) 4K GTX Titan X.

(b) 8K GTX Titan X.

Fig. 7. Decoding time breakdown for 4K and 8K per QP value, where *CPU* stands for the state-of-the-art CPU decoder and *CPU–GPU-I* the CPU+GPU decoding scheme I, both with a single CPU core.

For both 4K and 8K videos, the *CPU–GPU-I* implementation outperforms the baseline *CPU* decoder across all QP values. When compared to *CPU*, the reconstruction kernels represented with green bars shrink dramatically in the *CPU–GPU-I* implementation. This reduction of the decoding time is achieved even in the presence of two unavoidable overheads in the *CPU–GPU-I* decoder. First, the *H2D* time penalty occurs due to the required data transfer between the CPU and the GPU. Second, the *ED* part grows because it also includes the time to collect the inputs for the GPU kernels. Moreover, a larger speedup factor is achieved at higher QP values, where the reconstruction kernels in the *CPU* decoder account a higher fraction of the total decoding time. Overall, the fraction of execution time for the reconstruction kernels in 4K is 67% and in 8K is 51%. Although 4K has a higher fraction of reconstruction kernels than 8K, a same (total) speedup of 1.6× is achieved for both of them at the applications level. Due to the 4× more data volume per frame, the 8K setup has a higher acceleration factor of 8.4× for the reconstruction kernels, while for the 4K the acceleration factor is reduced to 4.9×.

6.1.2. Parallel CPU+GPU decoding performance

The proposed decoding schemes allow parallel decoding with multiple CPU cores, allied with the CPU+GPU pipelining, as presented in Section 4. Fig. 8 depicts the overall performance of the proposed decoding schemes when executing on multiple CPU cores with the Titan GPU. The performance of the baseline *CPU* decoder is also included for comparison purposes.

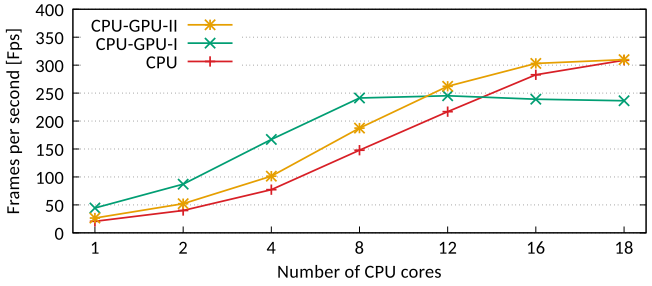
In general, the performance of all considered decoders improves by increasing the number of CPU cores. When a greater number of CPU cores is used, however, the performance of the proposed CPU+GPU decoding scheme I stops scaling. In particular, for 4K sequences (Fig. 8a), the *CPU–GPU-I* implementation saturates from 8 cores. This is justified by the fact that most decoding computations have been migrated to the GPU. As a result, the increased number of CPU cores can hardly be efficiently exploited by this decoding scheme I, despite of being faster than the *CPU*-only implementation. The performance of the baseline *CPU* decoder, on the other hand, scales continuously. As a consequence, the *CPU–GPU-I* implementation is eventually outperformed by the *CPU*

Table 3
Summary of the test platform hardware specifications.

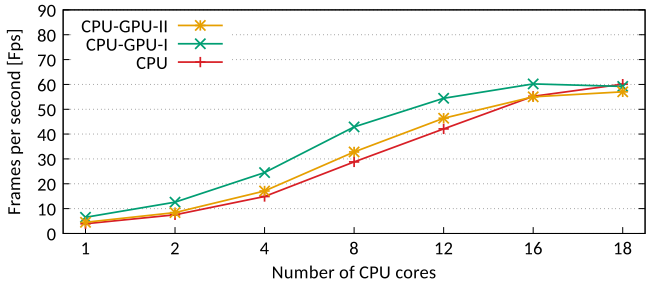
CPU: Intel Xeon E5-2699v3 (Haswell)						**Host Memory**	
Cores	Clock	$L1/core (I/D)	$L2/core	$L3	TDP	Size	Bandwidth
18	2.3 GHz	32 KB/32 KB	256 KB	45 MB	145 W	32 GB	68 GBps
GPU: NVIDIA GTX TITAN X (Maxwell)						**Device Memory**	
Cores	Clock	Compute capability		$L2	TDP	Size	Bandwidth
3072	1(1.2) GHz	5.2		3 MB	250 W	12 GB	336 GBps
Connection bus: PCIe 3.0 × 16							

Table 4
Bitrates in megabit per second [Mbps] of the main encoded video sequences with random access configuration.

Random access 4K, 10-bit, 4:2:0						Random Access 8K, 10-bit, 4:2:0		
QP	Fountain Lady	Lupo Confetti	Rain Fruit	Studio Dancer	Waterfall Pan	QP	Helicopter	Berlin
22	51.1	52.2	28.0	41.5	64.0	22	1164.5	250.1
27	23.3	18.5	11.7	11.7	25.6	26	341.9	140.4
32	10.7	9.5	5.9	6.0	10.3	30	95.5	86.4
37	5.0	5.5	3.2	3.3	4.2	34	39.7	52.1



(a) CPU vs. CPU+GPU decoding, 4K videos with all QP values considered.



(b) CPU vs. CPU+GPU decoding, 8K videos with all QP values considered.

Fig. 8. Performance of the proposed CPU+GPU decoding scheme I, scheme II, and the baseline CPU decoder for 4K and 8K videos, with all QP values considered.

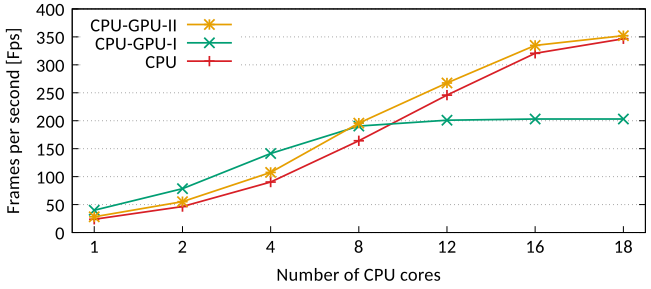


Fig. 9. CPU vs. CPU+GPU decoding, 4K videos encoded with low-delay P (IPPP) configuration.

Table 5
Decoding workload distribution in two task partitions of CPU+GPU decoding for 4K and 8K videos, with all QP values considered.

Videos	4K		8K	
Workload fraction	CPU vs. GPU		CPU vs. GPU	
Decoding scheme I	33%	67%	49%	51%
Decoding scheme II	71%	29%	80%	20%

decoder when more than 12 CPU cores are employed. Nevertheless, when only 4 CPU cores are used, which is one of the most common configurations in desktop PCs, the *CPU*-only implementation achieves 77 fps, while *CPU–GPU-I* achieves a performance of 167 fps, resulting into a speedup of 2.2× at the application level.

To address the GPU overloading issue when a high number of CPU cores are employed, the decoding scheme II offloads less workload onto the GPUs. Table 5 presents the workload distribution between the CPU and the GPU under the two proposed decoding schemes for 4K and 8K videos, when considering all QP values. The presented percentage is obtained by including the execution time of the entropy decoder and the remaining kernels using the baseline CPU decoder executing on a single CPU core.

For 4K videos, only 29% of workload is offloaded onto the GPU in decoding scheme II, while in scheme I the corresponding workload is 67%. As a result, the performance of scheme II is significantly improved at high number of CPU cores for 4K videos (see Fig. 8a). For example, *CPU–GPU-II* achieves 303 fps with 16 CPU cores, while *CPU–GPU-I* only attains 239 fps. Hence, by selecting appropriate decoding schemes, the proposed decoder is able to stride the workload balance between CPU and GPU according to their available computational resources. For 8K sequences (see Fig. 8b), the decoding scheme I outperforms scheme II even with more CPU cores because the workload distribution is more balanced between the CPU and the GPU for scheme I, with 49% vs. 51%, respectively. Compared to 4K, the heavier workload on CPU requires more CPU cores to match the computational capability of the GPU, and thus the performance of *CPU–GPU-I* scales well, even when 16 CPU cores are used. It outperforms *CPU* across all core configurations except 18, where both *CPU* and *CPU–GPU-I* achieve 60 fps.

6.1.3. *Decoding performance on videos with more encoding configurations*

The previously presented results only consider videos configured in random access mode (GOP size 8 and rate control off). To evaluate the performance of proposed decoders for a wider range of encoding modes, the five considered 4K sequences were further encoded with three more encoding configurations: the low-delay P (IPPP) encoding mode, the random access with rate control turned on, and the random access encoding mode with various GOP sizes.

Fig. 9 presents the obtained performance of the proposed decoders when applied on the IPPP videos. Compared to the results of the random

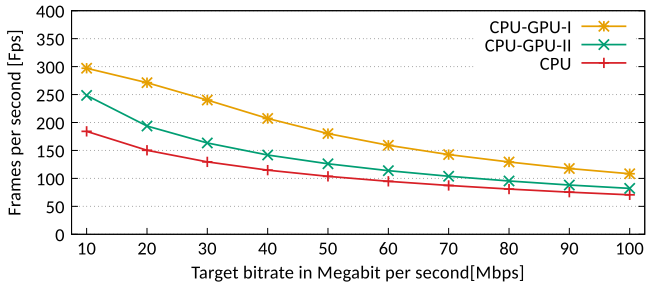


Fig. 10. CPU vs. CPU+GPU decoding, 4K videos encoded with random access mode and rate control turned on. All three decoders use 8 CPU cores.

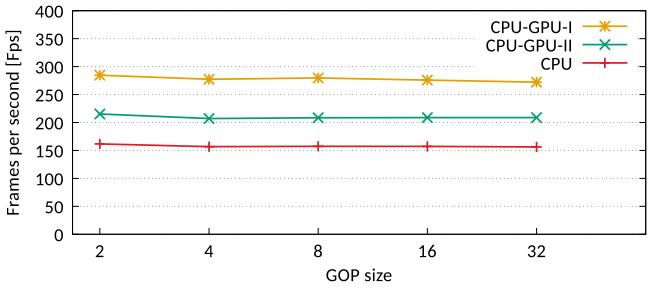


Fig. 12. CPU vs. CPU+GPU decoding, 4K videos encoded with random access mode and with GOP size of 2, 4, 8, 16, 32. All three decoders use 8 CPU cores.

access mode in Fig. 8a, the performance scalability of the three decoders is rather similar. However, the acceleration effect of CPU–GPU-I is lower than that in Fig. 8a, mainly because the workload for GPUs in the IPPP video encoding mode is lighter. Kernels targeted for GPUs account for 57% of the overall decoding time when using a single core CPU, while in the random access configuration the corresponding fraction reaches 67%, as presented in Table 5.

The performance of the proposed decoders for random access videos encoded with rate control turned on is presented in Fig. 10. All three decoders are executed using eight CPU cores. Compared to the CPU-only decoder, the proposed decoding schemes CPU–GPU-I and CPU–GPU-II both deliver higher frame rate when covering the whole range of the bitrate. Furthermore, it can be observed that CPU–GPU-I achieves better decoding performance than CPU–GPU-II, since reconstruction kernels of all frames are accelerated in CPU–GPU-I, while CPU–GPU-II only exploits GPU-acceleration for non-reference pictures.

By default, all previous 4K bitstreams with random access configurations are encoded with a GOP size of 8. If each frame is assumed to be decoded in a fixed time slot, defined as a cycle, Fig. 11 depicts that at least five cycles are required to complete a GOP with a size of 8 frames, since the GPU operates at frame level and some of the frames have to be serially processed, e.g., $0 \rightarrow 8 \rightarrow 4$. To evaluate the performance impact when GOP size changes, the five 4K considered videos were encoded with GOP sizes of 2, 4, 8, 16, and 32, using a QP value of 32.

Fig. 12 presents the decoding performance of proposed decoders using eight CPU cores when changing the GOP size configuration. In general, the decoding performance of the proposed decoders remain constant across different GOP sizes. Naturally, with a smaller GOP size, the number of cycles that is required to resolve the frame-level dependency inside a GOP decreases. For a given number of frames, however, the frame-level dependencies between GOPs increases. Taking the GOP size 2 as an example (which is not shown in Fig. 11), frame-level dependency across GOPs exists between frames $2 \rightarrow 4 \rightarrow 6 \rightarrow 8$ when considering four GOPs. As a result, changing the GOP size lays little influence on the decoding performance.

6.1.4. Performance gap to potential peak performance and bottleneck analysis

Taking into account that interconnection networks can be easily bandwidth-bound for parallel processing [44], the peak performance of the proposed CPU+GPU decoder is potentially limited by the host to device data transfer. Since the transferred data size for the kernel inputs and the decoded frames can be calculated, the potential peak performance of the proposed decoding schemes can be quantified based on the available bandwidth between the CPU and the GPU. Assuming **(i)** the peak bandwidth between the CPU and the GPU is BW_{peak} bytes per second, **(ii)** the amount of data that is transferred from the CPU to the GPU is $Size_{frame}$ bytes per frame, and **(iii)** the required time for transferring the data of one frame is δt, then the potential peak frame rate FPS_{peak} can be derived as:

$$FPS_{peak} = \frac{1}{\delta t} = \frac{BW_{peak}}{Size_{frame}} \tag{2}$$

BW_{peak} is fixed for each specific connection between the CPU and the GPU. For PCIe bus 3.0×16, the theoretical peak bandwidth is 16 GB/s [45]. In practice, however, a maximum bandwidth of 12 GB/s is eventually achieved using benchmarking [46]. On the other hand, the $Size_{frame}$ value depends on the resolution of the input videos, the kernel input data structures, and the selected decoding schemes.

Table 6 summarizes the data volume that is transferred per frame under the proposed decoding schemes I and II for the considered 4K (3840×2160) and 8K (7680×4320) video sequences. The transferred data size is independent of bitrates because the kernel input data structure is static for a given frame size. The peak performance for scheme I is straightforwardly estimated with Eq. (2), while scheme II needs to take into account the distribution of the reference and non-reference frames. For both 4K and 8K videos, the reference and non-reference frames are evenly distributed (50% each) for the chosen encoding configurations. Therefore, the data transferred per frame is the average of the reference and non-reference frame sizes. According to Table 6, the proposed decoder can achieve 75% of the peak performance for 4K (with scheme II) and 59% for 8K (with scheme I).

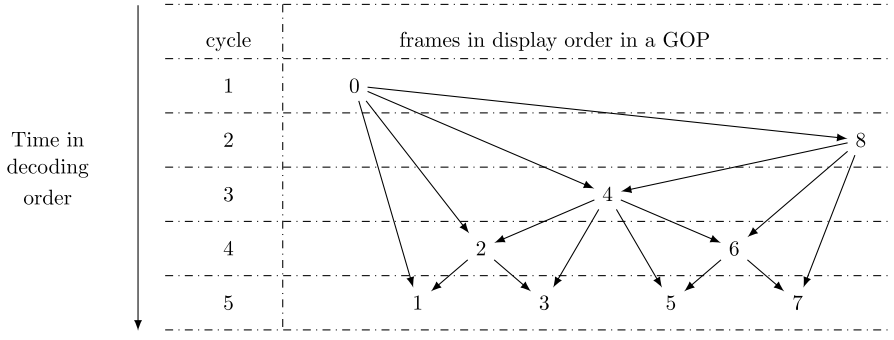
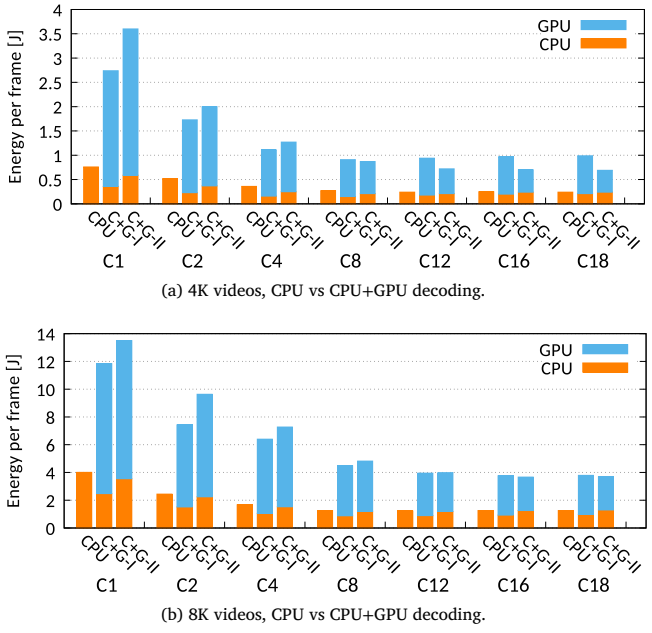


Fig. 11. Decoding order that addresses the inter-frame dependency in a GOP with a size of 8 frames.

Table 6
Transferred data size per frame in megabytes [MB] and the corresponding peak frame rate of 4K and 8K videos for the decoding schemes I and II.

	4K, 10-bit, 4:2:0			8K, 10-bit, 4:2:0		
	Scheme I	Scheme II		Scheme I	Scheme II	
	Each frame	Reference	Non-Ref.	Each frame	Reference	Non-Ref.
$Size_{frame}$	29.43 MB	28.30 MB	29.43 MB	117.70 MB	106.28 MB	117.70 MB
FPS_{peak}	408 fps	415 fps		102 fps	107 fps	



(a) 4K videos, CPU vs CPU+GPU decoding.

(b) 8K videos, CPU vs CPU+GPU decoding.

Fig. 13. Energy efficiency results of CPU only and CPU+GPU decoding, compared with energy per frame in Joules.

When a high number of CPU cores are used, the main bottlenecks of the GPU execution are the intra-prediction and motion compensation procedures. The low degree of wavefront parallelism in intra-prediction leads to a low resource utilization on GPU. The inter-frame dependency introduced by the motion compensation reduces the frame-level parallelism, which compromises the concurrency on the kernel execution from different CUDA streams. When CPU+GPU decoding scheme I is executed without an individual GPU kernel (but with all other GPU kernels on), switching off intra-prediction leads to the most significant performance gain, with a speedup factor of 1.38× compared to the decoding scheme I with all GPU kernels on. The second most significant speedup factor 1.27× comes from disabling motion compensation.

6.2. Energy efficiency results

With the help of energy measure module presented in Section 5, we firstly compare the energy efficiency of the baseline CPU decoder and the proposed CPU+GPU decoder. Afterwards, we show how to use the clock setting utility to exploit energy optimization opportunities.

6.2.1. Energy analysis of CPU+GPU decoding

Fig. 13 illustrates the average energy consumed per frame of the proposed decoding schemes when using a different number of CPU cores (C1–C18). The energy consumption of baseline CPU decoder is also presented for comparison purpose. For all decoding configurations (CPU baseline, decoding scheme I and II), the energy consumption per frame is reduced with the increase of CPU cores, but stops reducing approximately beyond 12 cores. In particular, the baseline CPU decoder consumes the least energy at 12 cores, with 0.24 J for 4K and 1.22 J for 8K. The proposed decoding scheme I consumes significantly more energy than the CPU baseline. Although its energy consumption is reduced on the CPU side, due to offloaded decoding workload, the energy consumed on GPU is higher than the reduction. The optimal CPU core configuration consuming the least energy for scheme I is 8 for 4K and 16 for 8K, with an energy of 0.92 J and 3.76 J, respectively. When compared to decoding scheme I, the decoding scheme II's energy consumption on the GPU is reduced significantly at high CPU cores for 4K but not for 8K. Such a difference occurs mainly due to the decoding time, as for 4K decoding scheme II has a significant better performance than scheme I while for 8K not.

The GPU energy efficiency for HEVC decoding is compromised by its high power consumption and the constrained performance due to the PCIe bus bandwidth. The energy per frame of GPU can be derived with power (joule per second) divided by performance (frames per second). This approach is used because the performance in heterogeneous CPU+GPU decoding is potentially bounded by the data transfer. The least GPU energy per frame is obtained with the maximum frame rate and the minimum power consumption. The maximum potential frame rate of the proposed decoder has been identified in Table 6, and the minimum power consumption of the GPU can be estimated by measuring the power when executing a benchmark with extremely lightweight workload. We design a benchmark with only one thread assigned in each SM, and each thread performs a single addition to represent the lightweight workload. By running this benchmark repeatedly for five seconds, the power consumption queried by the *nvmlDeviceGetPowerUsage* API keeps at 90 W. This measured power corresponds to a minimum energy per frame of 0.22 J (90 W/408 fps) for 4K and 0.88 J (90 W/102 fps) for 8K under decoding scheme I. In contrast, even with the decoding workload, the respective minimum energy of the CPU using the CPU only decoder is merely 0.24 J for 4K and 1.22 J for 8K.

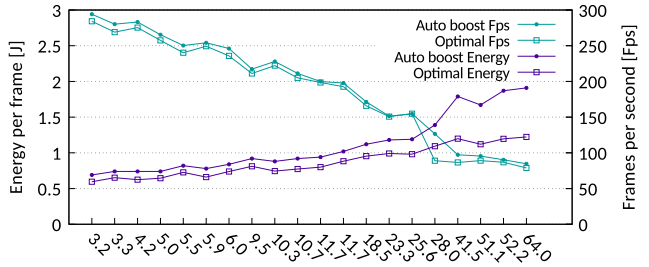
6.2.2. Energy optimized decoding by tuning GPU clocks

The previous results show that the high GPU power consumption hinders the energy efficiency of the proposed decoder. Hence, the GPU power reduction shall be our prime concern. On the other hand, the proposed decoder sometimes provides a higher frame rate than required by a target application, when the auto boost mode is enabled on the GPU. Decoding scheme I, for instance, provides 167 fps with 4 CPU cores for 4K videos, while in practice 120 fps is sufficient for envisioned UHD HFR applications [47]. In such cases, it is possible to trade the performance with power so that eventually a lower energy per frame is achieved. We exploit the power reduction opportunities on the more power hungry GPU device by setting its memory and graphics clocks to different clock rates.

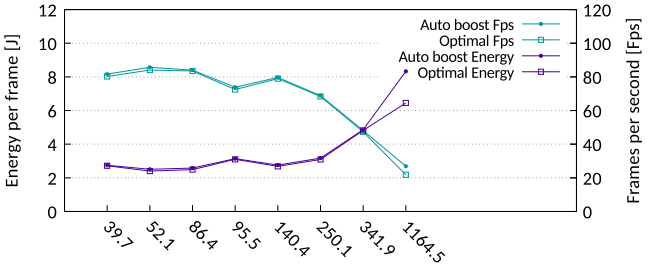
With NVML APIs, the GPU clocks of memory and graphics have to be set together, and each memory clock associates with a set of allowed graphics clocks. Table 7 presents the available values of memory and graphics clocks of the used Titan X GPU, in which the selected clock rates are labeled in boldface. The first two memory clocks associate with a same set of 64 graphics clocks. Additional 21 lower graphics clocks are available for memory clock 810 MHz. The lowest memory clock 405 MHz has the smallest number of graphics clocks, with only six values available. In particular, tests show that when the graphics clock is set larger than 1215 MHz, the queried graphics frequency is always 1215 MHz. Based on this test, we pick up seven graphics clocks (with a step

Table 7
Allowed clocks for memory and graphics of NVIDIA GTX Titan X in [MHz]. The selected clocks are in **boldface**.

Memory	Graphics (*64 clocks* for 3505 for 3304, *85 clocks* for 810, and *6 clocks* for 405)
3505	1392 1380 1367 1354 1342 1329 1316 1304 1291 1278 1266 1253 1240 1228 **1215** 1202 1190 1177 1164 1152 1139 1126 **1114** 1101 1088 1076 1063 1050 1038 1025 **1013** 1002 987 975 962 949 937 924 **911** 899 886 873 861 848 835 823 **810** 797 785 772 759 747 734 721 **709** 696 683 671
3304	658 645 633 620 **608** 595. Allowed *64 clocks*.
810	*64 clocks above plus* 582 570 557 544 532 519 507 494 482 469 457 444 432 419 407 405 324 270 202 162 135. Allowed *85 clocks*. Selected:**1215 1114 1013 911 810 709 608**
405	**405 324 270 202 162 135**. Allowed *6 clocks*.



(a) CPU+GPU decoding scheme I with 4 CPU cores for 4K.



(b) CPU+GPU decoding scheme I with 16 CPU cores for 8K.

Fig. 14. Energy efficiency when changing GPU memory and graphics clocks, compared to the default auto boost mode.

of 100 MHz) that cover most of allowed graphics clock ranges for the first three memory clocks. For the lowest memory clock, all associated six graphics clocks are selected. In this way, a total of 27 different clock settings are chosen to exploit the energy optimization space.

Two particular decoding configurations are selected for GPU clock tuning experiments, because their performance are close to standard frame rates, such as 60 fps and 120 fps [47]. By average, the first configuration (4K, decoding scheme I, 4 CPU cores) has a frame rate 167 fps and the second (8K, decoding scheme I, 16 CPU cores) has a frame rate 60 fps, as shown in Fig. 8. With GPU clocks set at different fixed rates, Fig. 14 illustrates the energy consumption per frame as well as the corresponding frame rates under the *optimal* mode (i.e. the clock settings leading to the least energy consumption) and the *auto boost* mode with the two configurations.

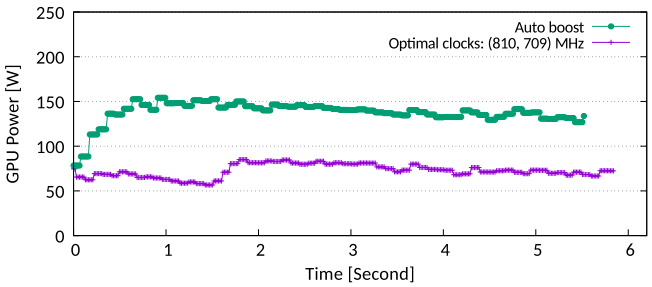
For 4K videos (see Fig. 14a), an obvious gap of energy is observed between the auto boost and optimal modes, thus showing the energy can be further optimized by using fixed clock rates. The energy reduction is mainly achieved by the reduced power consumption on GPU using lower operating clocks. Although reducing clock rates decreases the GPU performance, it has an insignificant impact on the overall decoding performance, since the heterogeneous decoding is CPU-bound with a configuration of only four CPU cores. For example, when compared to auto boost mode, an energy reduction of 36% is achieved at 64.0 Mbps by setting memory and graphics clocks to 810 MHz and 709 MHz, respectively. By contrast, in the auto boost mode, the GPU memory clock works at 3304 MHz and the graphic clock varies from 1001 MHz to 1202 MHz. The decoding performance at 64.0 Mbps, on the other hand, is merely reduced from 84.7 fps in auto boost mode to 78.9 fps in optimal mode.

In general, the energy reduction in high-bitrate videos (from 28.0 to 64.0 Mbps) is more obvious than in the low-bitrate videos (from 3.2 to 25.6 Mbps), for two reasons. First, the optimal clocks for high bitrate videos can be set at lower rates (memory: 810 MHz; graphics: 709 or 810 MHz) than that of the low bitrate videos (memory: 3304 MHz; graphics: 911 or 1013 MHz), thus resulting into a more significant power reduction. Second, the decoding time for high-bitrate videos are longer than low-bitrate videos. In fact, all low-bitrate videos are decoded at higher frame rates than 120 fps, while the high-bitrate videos are above 60 fps only. It is worth noting that even in optimal mode with lower clock rates, the decoding performance of low-bitrate and high-bitrate videos still exceeds the standard frame rates of 120 fps and 60 fps, respectively.

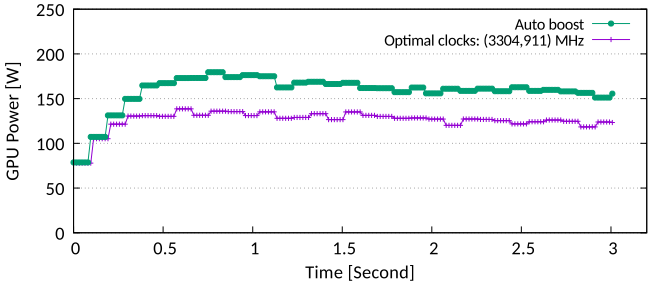
For 8K videos, because 16 cores are employed most of the input sequences' decoding is GPU-bound, and thus boosting GPU to high clocks is the optimal choice for overall performance. Therefore, the optimal clock settings with the minimum energy are mostly aligned with the auto boost mode, except the video with the highest bitrate at 1164.5 Mbps. The frame rate of this video is below 60 fps and represents an extraordinary encoding configuration. Entropy decoding of this video accounts for 66.1% and therefore the heterogeneous decoding is still CPU-bound. By setting both memory and graphics clocks to 810 MHz, an energy reduction of 23% is achieved for this video's decoding.

To visualize the power reduction in optimal mode, the real-time GPU power profiling of two representative 4K videos in high-bitrate and low-bitrate categories are depicted in Fig. 15a and b, respectively. The GPU power is obtained using a power sampling thread querying power consumption at the frequency of 62.5 Hz. For the high-bitrate sequence (Fig. 15a), there is a clear ramp-up phase for the power consumption at the beginning in auto boost mode. Initially, the GPU waits for kernel execution and thus the power stays at the lowest level (78 W). With more and more reconstruction kernels are prepared by the CPU, the GPU increases its power to adapt the heavier workload, and saturates at around 150 W. At the end, an average power of 138 W is consumed. In contrast, the power consumption under the optimal mode is constrained below 85 W, and with an average power of 72 W only. Although auto boost achieves a slightly shorter decoding time, its power consumption compromises its energy efficiency to a great extent.

For the low-bitrate video (Fig. 15b), a higher average power consumption in auto boost mode (158 W) is observed than that of the high-bitrate video (138 W), because the decoding of the low-bitrate video is less CPU-bound due to its lower fraction of entropy decoding (see Fig. 7). In this case, the overall decoding performance is more sensitive to the GPU performance, and therefore its optimal clocks (memory: 3304 MHz; graphics: 911 MHz) are higher than that of the high-bitrate video (memory: 810 MHz; graphics: 709 MHz). As a result, the average power consumption in optimal mode (126 W) is much higher than that of the high-bitrate video (72 W). The average power reduction from the auto boost mode to the optimal mode, on the other hand, shrinks to 32 W



(a) 4K WaterfallPan QP 22, 64.0 Mbps, CPU+GPU scheme I with 4 CPU cores.



(b) 4K WaterfallPan QP 27, 25.6 Mbps, CPU+GPU scheme I with 4 CPU cores.

Fig. 15. GPU power consumption under clock settings with minimal energy consumption and default auto boost configuration for two 4K videos, where a high-bitrate video is presented in Fig. 15(a) and a low-bitrate video in Fig. 15(b).

(158 W – 126 W) only, compared to 66 W (138 W – 72 W) in high-bitrate video. The less significant power saving as well as the shorter decoding time for low-bitrate videos results into less energy reduction.

7. Conclusions and future work

A highly parallel design for the HEVC decoding on heterogeneous architectures consisting of the CPU and GPU devices has been presented. It allows exploiting multiple levels of parallelism on the CPU, GPU, and between the CPU and GPU devices simultaneously, for achieving maximum performance. On top of that, different workload balancing schemes were proposed, in order to exploit the best performance depending on the employed CPU and GPU computing resources. In addition, we implemented an energy measurement approach for the heterogeneous CPU+GPU decoder with the RAPL interface and NVML library. Furthermore, the performance and energy efficiency of the proposed decoder were evaluated on a workstation desktop and compared to the state-of-the-art CPU decoder.

The obtained experimental results show that the offloaded kernels are accelerated significantly by the GPU device, with a factor of 4.9× for 4K and 8.4× for 8K. Moreover, the proposed CPU+GPU decoder provides application-level acceleration when compared to the state-of-the-art CPU decoder. In particular, when a low number of CPU cores are used, it is better to offload the reconstruction kernels of all frames. For example, the proposed decoder with four CPU cores under this task-based workload partition achieves 167 fps for Ultra HD 4K videos, suggesting a speedup of 2.2× at the application level. When a higher number of CPU cores are employed, only the reconstruction kernels of the non-reference frames are offloaded, in order to achieve a better workload balance. This new decoding scheme delivers 303 fps for 4K when 16 CPU cores are used, in contrast to 239 fps under the task-based partition. Overall, to achieve a better performance, the selection of the proposed decoding schemes depends on the ratio of the CPU and GPU computing resource, as well as the workload distribution within the input videos. Finally, we show that energy optimization can be applied by setting fixed and lower GPU clocks when the CPU+GPU decoding performance is bounded by the CPU, cases often observed in high bitrate videos. In particular, an energy reduction up to 36% is achieved when compared to the auto boost mode. Energy wise, however, GPU architecture is not as efficient as the CPU for HEVC decoding, due to its high power consumption and the constrained performance from the PCIe data transfer.

Under the current implementation, the proposed decoding schemes and GPU clocks are selected statically for a given hardware and video configuration. In future, a dynamic workload allocator can be developed as the next step to deal with the input variations.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 688759 (LPGPU2). This work was partially supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) under project number UID/CEC/50021/2013.

References

[1] G.J. Sullivan, J. Ohm, W. Han, T. Wiegand, Overview of the high efficiency video coding (HEVC) standard, IEEE Trans. Circuits Syst. Video Technol. 22 (12) (2012) 1649–1668.
[2] J. Ohm, G.J. Sullivan, H. Schwarz, Thiow Keng Tan, T. Wiegand, Comparison of the coding efficiency of video coding standards–including high efficiency video coding (HEVC), IEEE Trans. Circuits Syst. Video Technol. 22 (12) (2012) 1669–1684.
[3] E. Reinhard, G. Ward, S. Pattanaik, P. Debevec, High Dynamic Range Imaging: Acquisition, Display, and Image-Based Lighting (The Morgan Kaufmann Series in Computer Graphics), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
[4] K. Masaoka, T. Yamashita, Y. Nishida, M. Sugawara, Color management for wide-color-gamut UHDTV production, SMPTE Motion Imaging J. 124 (3) (2015) 19–27.
[5] L. Wilcox, R. Allison, J. Helliker, B. Dunk, R. Anthony, Evidence that viewers prefer higher frame-rate film, ACM Trans. Appl. Percept. (TAP) 12 (4) (2015) 15:1–15:12.
[6] F. Henry, S. Pateux, Wavefront Parallel Processing. Technical Report JCTVC-E196, March 2011.
[7] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, M. Zhou, An overview of tiles in HEVC, IEEE J. Sel. Top. Sign. Proces. 7 (6) (2013) 969–977.
[8] C.C. Chi, M. Alvarez-Mesa, B. Bross, B. Juurlink, T. Schierl, SIMD acceleration for HEVC decoding, IEEE Trans. Circuits Syst. Video Technol. 25 (5) (2015) 841–855.
[9] Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T and ISO/IEC. HM-16.7 Reference software, 2015, https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.7.
[10] Y. Duan, J. Sun, L. Yan, K. Chen, Z. Guo, Novel efficient HEVC decoding solution on general-purpose processors, IEEE Trans. Multimed. 16 (7) (2014) 1915–1928.
[11] D.F. de Souza, N. Roma, L. Sousa, Opencl parallelization of the HEVC de-quantization and inverse transform for heterogeneous platforms, in: 2014 Proceedings of the 22nd European Signal Processing Conference (EUSIPCO), IEEE, 2014, pp. 755–759.
[12] L. He, S. Goto, A high parallel way for processing IQ/IT part of HEVC decoder based on GPU, in: 2014 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), IEEE, 2014, pp. 211–215.
[13] D.F. de Souza, A. Ilic, N. Roma, L. Sousa, GPU acceleration of the HEVC decoder inter prediction module, in: 2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP), IEEE, 2015, pp. 1245–1249.
[14] D.F. de Souza, A. Ilic, N. Roma, L. Sousa, Towards GPU HEVC intra decoding: Seizing fine-grain parallelism, in: 2015 IEEE International Conference on Multimedia and Expo (ICME), June 2015, pp. 1–6.
[15] A.F. Eldeken, R.M. Dansereau, M.M. Fouad, G.I. Salama, High throughput parallel scheme for HEVC deblocking filter, in: 2015 IEEE International Conference on Image Processing (ICIP), September 2015, pp. 1538–1542.
[16] W. Jiang, H. Mei, F. Lu, H. Jin, L. Yang, B. Luo, Y. Chi, A novel parallel deblocking filtering strategy for HEVC/H.265 based on GPU, Concurr. Comput.: Pract. Exper. 28 (16) (2016) 4264–4276. CPE-15-0134. R1.
[17] D.F. de Souza, A. Ilic, N. Roma, L. Sousa, HEVC in-loop filters GPU parallelization in embedded systems, in: 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), IEEE, 2015, pp. 123–130.
[18] D.F. de Souza, A. Ilic, N. Roma, L. Sousa, Ghevc: An efficient hevc decoder for graphics processing units, IEEE Trans. Multimed. 19 (3) (2017) 459–474.
[19] M. Abeydeera, M. Karunaratne, G. Karunaratne, K. De Silva, A. Pasqual, 4K real-time HEVC decoder on an FPGA, IEEE Trans. Circuits Syst. Video Technol. 26 (1) (2016) 236–249.
[20] M. Tikekar, C.T. Huang, C. Juvekar, V. Sze, A.P. Chandrakasan, A 249-Mpixel/s HEVC video-decoder chip for 4K ultra-HD applications, IEEE J. Solid-State Circuits 49 (1) (2014) 61–72.

[21] NVIDIA. NVIDIA PureVideo HD Technology, July 2007, http://www.nvidia.com/page/purevideo_hd.html.
[22] Intel. Intel Quick Sync Video, 2017, http://www.intel.com/content/www/us/en/architecture-and-technology/quick-sync-video/quick-sync-video-general.html.
[23] AMD. White Paper, AMD Unified Video Decoder (UVD), July 2007, https://www.amd.com/Documents/UVD3_whitepaper.pdf.
[24] NVIDIA. NVDEC Support Matrix, 2016, https://developer.nvidia.com/video-encode-decode-gpu-support-matrix#Encoder.
[25] X. Mei, Q. Wang, X. Chu, A survey and measurement study of GPU DVFS on energy conservation, Digit. Commun. Netw. 3 (2) (2017) 89–100.
[26] G. Semeraro, G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, M.L. Scott, Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling, in: Proceedings Eighth International Symposium on High Performance Computer Architecture, February 2002, pp. 29–40.
[27] Ching Chi Chi, Mauricio Alvarez-Mesa, Juurlink Ben, Low-power high-efficiency video decoding using general-purpose processors, ACM Trans. Archit. Code Optim. 11 (4) (2015). 56:1–56:25.
[28] C.C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, T. Schierl, Parallel scalability and efficiency of HEVC parallelization approaches, IEEE Trans. Circuits Syst. Video Technol. 22 (12) (2012) 1827–1838.
[29] NVIDIA. NVIDIA CUDA C Programming Guide v7.5, September 2015, http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf.
[30] B. Wang, D.F. de Souza, M. Alvarez-Mesa, C.C. Chi, B. Juurlink, A. Ilic, N. Roma, L. Sousa, GPU parallelization of HEVC in-loop filters, Int. J. Parallel Program. (2017) 1–21.
[31] D.F. de Souza, A. Ilic, N. Roma, L. Sousa, GPU-assisted HEVC intra decoder, J. Real-Time Image Process. 12 (2) (2016) 531–547. Springer.
[32] NVIDIA. CUDA Concurrency & Streams, 2011, https://developer.nvidia.com/gpu-computing-webinars.
[33] Srinivas Pandruvada, Running average power limit, 2014, https://01.org/blogs/2014/running-average-power-limit----rapl.
[34] NVIDIA. NVML API Reference Guide, 2015, http://docs.nvidia.com/deploy/nvml-api/index.html.
[35] V.M. Weaver, M. Johnson, K. Kasichayanula, J. Ralph, P. Luszczek, D. Terpstra, S. Moore, Measuring energy and power with PAPI, in: 2012 41st International Conference on Parallel Processing Workshops, September 2012, pp. 262–268.
[36] E. Rotem, A. Naveh, A. Ananthakrishnan, E. Weissmann, D. Rajwan, Power-management architecture of the intel microarchitecture code-named sandy bridge, IEEE Micro 32 (2) (2012) 20–27.
[37] Intel. Intel 64 and IA-32 Architectures Software Developer's Manual. Volume 3B: System Programming Guide, Part 2, September 2016, http://www.intel.de/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf.
[38] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, R. Geyer, An energy efficiency feature survey of the intel Haswell processor, in: 2015 IEEE International Parallel and Distributed Processing Symposium Workshop, May 2015, pp. 896–904.
[39] K. Kasichayanula, D. Terpstra, P. Luszczek, S. Tomov, S. Moore, G.D. Peterson, power aware computing on GPUs, in: 2012 Symposium on Application Accelerators in High Performance Computing, July 2012, pp. 64–73.
[40] R. Gonzalez, B.M. Gordon, M.A. Horowitz, Supply and threshold voltage scaling for low power CMOS, IEEE J. Solid-State Circuits 32 (8) (1997) 1210–1216.
[41] JCT-VC. High Efficient Video Coding (HEVC). ITU-T Recommendation H.265 and ISO/IEC 23008-2, ITU-T and ISO/IEC JTC1, April 2013.
[42] European Broadcasting Union. EBU UHD-1 Test Sequences, 2012, http://tech.ebu.ch/testsequences/uhd-1.
[43] Japan Broadcasting Corporation. 8K content sequences, 2015, http://www.nhk-mt.co.jp/kouseisai/sales/8k.html.
[44] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, K. Yelick, The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, December 2006.
[45] Jason Lawley, Understanding Performance of PCI Express Systems White Paper, October 2014, http://www.xilinx.com/support/documentation/white_papers/wp350.pdf.
[46] NVIDIA. Bandwidth Test Sample Benchmark Under the CUDA Toolkit v7.5, September 2015.
[47] ITU Radio communication Sector. RECOMMENDATION ITU-R BT.2020: Parameter values for ultra-high definition television systems for production and international programme exchange. ITU-T Recommendation Broadcasting service (television), August 2012.