Performance analysis of optimized versatile video coding software decoders on embedded platforms

Anup Saha, Wassim Hamidouche, Miguel Chavarrias, Fernando Pescador, Ibrahim Farhat

Journal of Real-Time Image Processing (2023) 20:120
https://doi.org/10.1007/s11554-023-01376-7

RESEARCH

Performance analysis of optimized versatile video coding software decoders on embedded platforms

Anup Saha[1] · Wassim Hamidouche[2,3] · Miguel Chavarrías[1] · Fernando Pescador[1] · Ibrahim Farhat[2]

Received: 9 May 2023 / Accepted: 4 October 2023 / Published online: 31 October 2023

Abstract

In recent years, the global demand for high-resolution videos and the emergence of new multimedia applications have created the need for a new video coding standard. Therefore, in July 2020, the versatile video coding (VVC) standard was released, providing up to 50% bit-rate savings for the same video quality compared to its predecessor high-efficiency video coding (HEVC). However, these bit-rate savings come at the cost of high computational complexity, particularly for live applications and on resource-constrained embedded devices. This paper evaluates two optimized VVC software decoders, named OpenVVC and Versatile Video deCoder (VVdeC), designed for low resources platforms. These decoders exploit optimization techniques such as data-level parallelism using single instruction multiple data (SIMD) instructions and functional-level parallelism using frame, tile, and slice-based parallelisms. Furthermore, a comparison of decoding runtime, energy, and memory consumption between the two decoders is presented while targeting two different resource-constraint embedded devices. The results showed that both decoders achieve real-time decoding of full high-definition (FHD) resolution on the first platform using 8 cores and high-definition (HD) real-time decoding for the second platform using only 4 cores with comparable results in terms of the average energy consumed: around 26 J and 15 J for the 8 cores and 4 cores platforms, respectively. Furthermore, OpenVVC showed better results regarding memory usage with a lower average maximum memory consumed during runtime than VVdeC.

1 Introduction

A new era of information and communication technologies is emerging, where video communication plays an essential role in internet traffic. In particular, the significant increase in video traffic, supported by emerging video formats and applications, has led to the development of a new video coding standard named versatile video coding (VVC)/H.266. The latter was standardized in July 2020 by the Joint Video Experts Team (JVET) of the ITU-T Video Coding Experts Group (VCEG) and the Motion Picture Experts Group (MPEG) of ISO/IEC JTC 1/SC 29 [1]. VVC enables bit-rate savings of up to 50% [15] with respect to the previous standard High Efficiency Video Coding (HEVC)/H.265 [2] for the same video quality. However, this achievement comes at the cost of 8× and 2× more complexity compared to HEVC for the reference encoder and decoder, respectively [3]. In this scenario, the main challenge is to develop real-time VVC codecs, either a hardware or software solution for video encoding or decoding, that consider resource-constrained consumer devices frequently used in consumer electronics based on embedded platforms.

This work was supported by both the Energy Efficient Enhanced Media Streaming (3EMS) project funded by the Brittany Region and TALENT project (PID2020-116417RB-C41), funded by the Spanish Ministerio de Ciencia y Innovación.

✉ Miguel Chavarrías
miguel.chavarrias@upm.es

Anup Saha
anup.saha@upm.es

Wassim Hamidouche
wassim.hamidouche@insa-rennes.fr;
wassim.hamidouche@tii.ae

Fernando Pescador
fernando.pescador@upm.es

Ibrahim Farhat
ibrahm.farhat@insa-rennes.fr

[1] CITSEM at Universidad Politécnica de Madrid, Madrid, Spain

[2] Univ. Rennes, INSA Rennes, CNRS, IETR—UMR, 6164 Rennes, France

[3] Technology Innovation Institute (TII), P.O.Box: 9639, Masdar City Abu Dhabi, UAE

Each coding standard is released with a reference software implementation available to the scientific community. These solutions incorporate all the standard features but offer minimal speed performance. For example, in the case of VVC, the reference software is VVC test model (VTM) [4]. Taking this as a starting point, research groups and companies develop their own real-time software and hardware solutions. These solutions mainly exploit the intrinsic parallelism of the algorithms, both at the data and functional levels, to enhance their performance in terms of speed and energy consumption. In the first case, some data operations included in the source code are optimized using instructions of type single instruction multiple data (SIMD) [5]. Here, vectorized operations perform mathematical operations with more than one operator using a single processor instruction. The other potential optimization route is to take advantage of the intrinsic parallelism of the independent processing of pictures [6] or smaller parts of the picture, such as slices [7] or tiles [8]. In the latter case, it is necessary that the coding is done by activating these normative tools that break dependencies between adjacent regions.

In this work, two open-source VVC decoders are evaluated and compared against each other. These solutions, named OpenVVC [9, 10] and Versatile Video deCoder (VVdeC) [11] decoders, are optimized using data and functional-level parallelism techniques. This paper evaluates their performance in decoding runtime, power consumption, and memory usage targeting two different embedded platforms. The results showed that both decoders achieved 15 to 34 frame per second (fps) for ultra-high-definition (UHD) sequences with quantization parameter (QP) 27 and 37 and achieved real-time decoding of full high-definition (FHD) and high definition (HD) sequences over the first target platform using 8 cores. Furthermore, 16 to 28 fps have been obtained for FHD sequences with QPs 27 and 37, and real-time decoding has been achieved for all HD sequences by OpenVVC and VVdeC when targeting the second embedded platform with 4 cores. Regarding energy consumption and maximum memory usage, the experimental results showed that VVdeC requires 2× more memory compared to OpenVVC on both target platforms. On the other hand, OpenVVC consumed the same energy as VVdeC on the embedded system-on-chip (ESoC)1 platform with 8 cores and around 1.25× VVdeC's energy consumption when targeting the ESoC2 embedded platform with 4 cores.

The remainder of the paper is structured as follows. First, Sect. 2 briefly introduces the VVC standard. Then, Sect. 3 describes the optimizations included in VVC decoders using specific parallelization techniques along with the state-of-the-art of VVC decoders, followed by a brief description of open-source VVC decoders in Sect. 4. Next, Sect. 5 details the proposed optimization techniques in the OpenVVC decoder. The results obtained and the comparison between the performance of OpenVVC and VVdeC are provided in Sect. 6. Finally, Sect. 7 concludes the paper.

2 Introduction to VVC

This section briefly describes the VVC decoder and related codec optimizations in scientific literature. Like its predecessors, VVC was designed based on a hybrid coding scheme using intra-/inter-prediction and transform coding. Figure 1 presents the block diagram of the VVC decoding process. Here, the encoded bit-stream is the input, the decoded video is the output, and blocks present the decoding stages.

2.1 Entropy decoding

Bit-stream decoding begins with entropy decoding relying on enhanced context adaptive binary arithmetic coding (CABAC) [12]. An updated multi-hypothesis probability estimation model was adapted, and the computed look-up table in HEVC was removed to enhance the accuracy. In addition, the coefficient coding has been improved by allowing 1×16, 2×8, 8×2, 2×4, 4×2, and 16×1 coefficients group size for TX block size.

2.2 Inverse quantization and transform

The spatial domain coefficients are retrieved from the frequency domain by inverse quantization and inverse transformation. VVC introduces multiple transform selection (MTS) [13] tool used to encode the residual inter- and intra-coding blocks. MTS allows three transforms of the rectangle blocks with the height and width of ≤ 64 for discrete cosine transform (DCT)-II, ≤ 32 for DCT-VIII and discrete sine transform (DST)-VII. Furthermore, the coefficients of high frequency are zeroed when the height and width are equal to 64 for DCT-II and 32 for DCT-VIII and DST-VII.

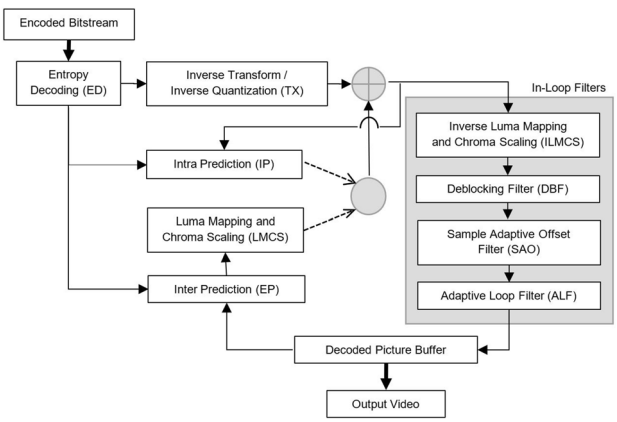

Fig. 1 Block diagram of a VVC decoder

2.3 Intra-prediction

In VVC intra-prediction module, 32 additional directional intra-prediction modes are added to those in HEVC. Moreover, the intra-prediction allows wide-angle intra-modes for rectangular blocks, improving prediction accuracy. In addition, the matrix-weighted intra-prediction tool is used as a new intra-prediction mode by taking the above and left neighboring lines of the prediction block. Further, VVC adapted the cross component linear model tool [14], which was applied to predict the chroma components from the luma components.

2.4 Inter-prediction

Inter-prediction takes advantage of the temporal redundancy of the video by removing the correlation in the temporal domain [15]. The motion compensation estimates the current coding unit samples according to the samples recorded in the decoded picture buffer. In addition, an 8-tap filter is used for luma samples to create motion-compensated prediction, and an 8-tap filter is used for chroma samples for interpolation. Furthermore, VVC achieved improved prediction accuracy using decoder-side motion vector refinement [16] and bi-directional optical flow prediction refinement [16].

2.5 Luma mapping with chroma scaling

Forward luma mapping with chroma scaling (LMCS) is a new tool introduced in VVC that comes after inter-prediction stage. It has two parts: luma mapping (LMP), used to modify predicted luma samples, and chroma scaling (CSP), used to modify chroma residues. LMP makes the most use of the range of luma code values and provides an efficient real-location process of luma code values in the coding domain. Therefore, CSP changes the value of the residual chroma samples in the chroma coding block to mitigate the defect that arises from the interaction between luma and corresponding chroma signals [17].

2.6 In-loop filters

VVC in-loop filters consist of inverse luma mapping chroma scaling (ILMCS) [18], deblocking filter (DBF), sample adaptive offset (SAO) filter, and adaptive loop filter (ALF). First, ILMCS is a new tool in VVC, which enhances decoding performance by inversely mapping the luma code to the reconstructed block. DBF and SAO in VVC are very similar to HEVC [19]. DBF is used to detect and filter the artifacts of pixels at the boundary of the block, and SAO is used to minimize the distortion of the sample over the pixels filtered by DBF. Furthermore, unlike HEVC, VVC includes ALF [18] to reduce the mean square error of the decoded pixels. Therefore, undesired artifacts obtained by the previous decoding modules, including blurring, blocking, flickering, ringing, and color shift, are mitigated using in-loop filters.

3 Optimized and real-time software decoders

Two levels of parallelism can be exploited to speed up the video decoding process: coarse-grained and fine-grained. In this section, we describe first some general concepts related to the parallelization methos used to accelerate video codecs. Later both coarse-grained and fine-grained parallelisms are presented.

3.1 Codec optimizations

Various parallelization methods have been used to accelerate video codecs on central processing unit (CPU), graphics processing unit (GPU), and the hybrid architectures. Yan et al. [20] accelerated a HEVC decoder by ×4 compared to HM 4.0 using SIMD technologies on an x86 processor. The authors in [21] and [22] proposed a GPU-based implementation of HEVC decoder that satisfied real-time requirements for the decoding of UHD 4k sequences. S. Gudumas et al. [7] discussed various optimization techniques to implement VVC using multiple CPU cores on heterogeneous platforms to achieve real-time decoding. Here, the decoding tasks were redesigned and parallelized with task parallelization based on load balancing and data parallelization at the coding tree unit (CTU) level. The authors of [23] presented a GPU-based motion compensation system to accelerate the VVC decoder that takes advantage of partitioning different coding unit (CU) threads and proper organization. Furthermore, Wieckowski et al. in [24] described an optimized VVC decoder that achieved real-time decoding on general-purpose CPUs. Here, SIMD operations-based optimization and multithreading-based optimization were adopted. The authors in [25] demonstrated an optimized real-time VVC decoder that takes advantage of SIMD instruction extensions on x86-based CPU. Moreover, the authors discussed the implementation of the frame, CTU, sub-CTU, and task-level parallelizations. An optimized VVC software decoder for mobile platforms is presented in [26]. This decoder is based on VTM−11.0 reference software, and achieves real-time decoding for HD video sequences using SIMD and multi-threading on ARM [19] based platform. Finally, in [40] the authors present an heterogenous CPU + GPU implementation of a VVC decoder where the ALF filtering was migrated to the GPU cores.

3.2 Coarse-grained parallelism

Frame-level parallelism: Simultaneous processing of multiple frames is performed in frame-level parallelism while dependencies of motion compensation are also satisfied. The range of the motion vector is, in this case, the deterministic factor [7]. Video sequences with large motion would imply large dependencies between frames, possibly creating a significant bottleneck for frame-level parallelism. Hence, sequences in the all-intra configuration are the most profited by frame-level parallelism since there are no motion compensation dependencies. Moreover, a single thread should allocate additional memory for global (i.e., picture) and local buffers in frame-level parallelism, requiring more memory than in sequential decoding.

Wave-front parallel processing: Wave-front parallel processing (WPP) allows virtual picture partitioning into CTU rows [7]. WPP removes the above right-coding intra-prediction dependency during the picture partitioning while the entropy engine is initialized at the start of each CTU line. Therefore, at the beginning of each CTU row, the CABAC context is reinitialized, and it depends on the data from first CTU of the previous row. As a result, the row decoding dependency slightly limits the parallelization efficiency of WPP.

Tiles parallelism: VVC supports tiles of rectangular shape consisting of CTUs [15]. An example of tile partition is shown in Fig. 2. Here, four tiles are labeled with A, B, C, and D. Tiles are separated by boundaries, eliminating the prediction dependencies. Therefore, for all the tiles, the entropy encoding step is reinitialized, which allows the decoding of tiles independently. It allows for decoding a picture concurrently using multiple threads. However, the in-loop filtering, when enabled, can only be carried out at the tile boundaries when pixels are reconstructed in the adjacent tiles.

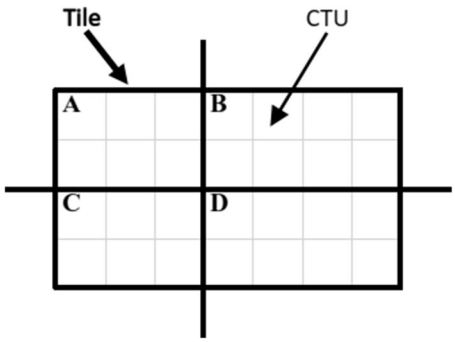

Fig. 2 Illustration of tile partitioning in VVC decoder

3.3 Fine-grained parallelism

Single instruction multiple data: SIMD is a data-level parallel processing technique that loads multiple data in a single register. The x86-based architectures offer streaming SIMD extensions (SSE) and advanced vector extensions (AVX)-based SIMD intrinsics. Embedded general purpose processor (EGPP)-based platforms support ARM Neon suite instead of SSE- and AVX-based SIMD intrinsic. ARM Neon is an advanced SIMD technology designed for mobile devices that support up to 128-bit register. Neon-based SIMD technology can be used in the following ways [27]: a) using Neon intrinsics, b) Neon-enabled libraries, c) compiler auto-vectorization, and d) hand-coded Neon assembler.

4 Open-source VVC decoders

A few open-source software decoders are available that comply with the VVC standard. First, the reference test model VVC mentioned above, or VTM. Second, VVdeC [11], an implementation proposed by the Fraunhofer Institute, is an optimized decoder based on VTM. It includes SIMD and multi-threads parallelization for optimal decoding performance. Finally, OpenVVC is a lightweight open-source software decoder available in [9]. OpenVVC decoder targets different operating systems and hardware architectures. Similarly to VVdeC, OpenVVC uses data and functional-level parallelism to optimize the decoding performance. For more details on VVC codecs, Sullivan [28] provides a complete list of available VVC encoder and decoder implementations.

4.1 Introduction to OpenVVC

OpenVVC is an open-source software VVC decoder written in C programming language. It is compiled as a cross-platform library, compatible with most-used operating systems, and optimized for x86 and ARM processors. The last version of the decoder is compatible with the VVC Main profile. In addition, the decoder was integrated with VLC [29], GPAC [30], and FFplay [31] video players. OpenVVC provides high decoding speed with a low memory footprint. It takes advantage of tile- and frame-based parallelization on multi-core CPU along with SIMD optimizations for accelerating the decoding process. The OpenVVC decoding process starts by parsing the sequence parameters. The reconstruction tasks, including inverse quantization and transform (TX), LMCS, inter-prediction (EP), and intra-prediction (IP) decoding blocks, are performed at the CU level. Then DBF is performed immediately after the reconstruction process is completed at the level of CTU. This approach helps optimize memory usage by avoiding massive storage of quantization parameter map and CU dimension required for the DBF

process. Finally, ALF is applied after the SAO at the level of CTU line before delivering the decoded frame as output.

4.2 Introduction to versatile video decoder

VVdeC [11] is an open-source software VVC decoder optimized for x86 architectures and developed by Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute (HHI). Unlike OpenVVC, VVdeC has been developed from VTM reference software [32]. It supports the VVC Main 10 profile, enabling to decode all conformance VVC bitstreams [33]. In addition, VVdeC comes with SIMD optimizations and multi-threading parallelization for ARM and x86 architectures. The parallelization of VVdeC decoding begins by parsing multiple frames concurrently. Therefore, in the reconstruction process, tasks are split based on CTU lines and CTUs. Here, a stage is given to each CTU for tracking the following stage and process tasks in parallel after the dependencies are satisfied. It allows task coordination where a task worker is assigned to each CTU. A thread pool schedules the task workers assigning the available tasks. VVdeC has achieved decoding time reduction up to 90% [24] with respect to VTM.

5 Decoder optimizations

In previous work [34], VVdeC V0.2 was optimized for ARM architectures using Neon-based SIMD parallelisms. The source code is openly available in [35]. Moreover, the optimization of VVdeC V1.3 is similar. Therefore, this section mainly focuses on implementing frames, tiles, and Neon-based SIMD parallelisms in OpenVVC over EGPP-based platforms.

5.1 Frames and tiles parallelization in OpenVVC

In frame-level parallelism of OpenVVC, a main thread is used to parse the picture parameter set (PSP), sequence parameter set (SPS), picture/slice header and schedule decoding threads with a thread pool. Then the main thread provides the data and updates the internal structure of the available threads in the thread-pool for decoding the frame. Therefore, motion compensation synchronization between threads is performed for sequences with inter-coding configuration after starting the decoding process. In fact, this latter is the most challenging step in frame-level parallelism, where the available thread has to wait for motion compensation before starting the pixel processing. When the pixels are ready, the available thread is able to perform the decoding process. This process is applied at the CTU line level since OpenVVC performs decoding and in-loop filtering at CTU line basis. Once the decoding process is completed, the decoding threads signal their availability to the main thread and return to the thread-pool.

On the other hand, tiles level parallelism is applied at a portion of a frame. In fact, every frame is decomposed into rectangular regions of the picture containing multiple CTUs [15]. The main challenge of tile level parallelism is that tiles could have different runtime complexities. Therefore, the time required to finish one frame is the time to finish the longest tile. In this case, at a certain processing time, some threads are free, without a task, waiting to finish processing the current frame. For more details about this issue, a quality-driven dynamic frame partitioning for efficient parallel processing is explained by Amestoy et al. in [6]. A dynamic tile and rectangular slice partitioning solution enables the best partitioning configuration that minimizes the trade-off between multi-thread encoding time and quality loss. This is performed by taking into account both spatial texture of the content and encoding times of previously encoded frames. Experiments prove that the proposed solution, compared to uniform partitioning, significantly decreases multi-thread encoding time, with slightly better quality.

The proposed solution integrated into OpenVVC aims to efficiently activate all threads at all times. To do so, it applies a thread pipelining technique that overlooks frames and focuses only on tiles. Figure 3 illustrates tile pipelining. The tile partitioning forms a 2×2 grid. They are labeled A, B, C, and D for the first frame and A', B', C', and D' for the second frame and delimited by thicker black lines. Prediction dependencies across tile boundaries are broken and entropy encoding state is reinitialized for each tile. These restrictions ensure that tiles are independently decoded, allowing several threads to decode simultaneously the same picture. As it can be observed, regardless of the tile position, as soon as a thread is available from the thread pool,

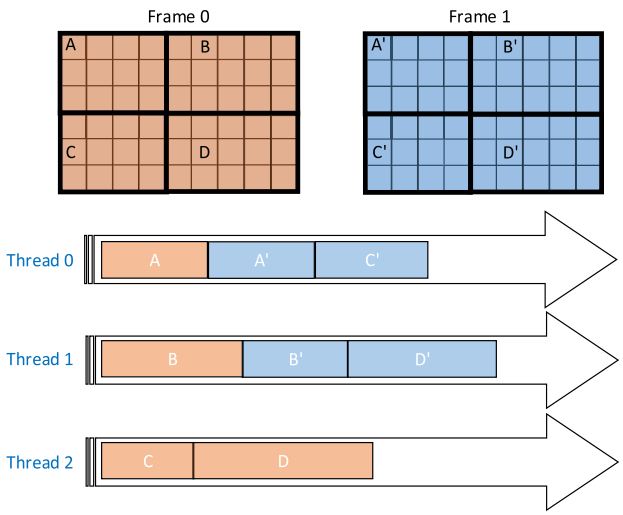

Fig. 3 Illustration of tile pipelining in OpenVVC decoder

the tile is processed. Thread 2, for example, does not work on any tiles of the second frame since it takes the entire time working on tile D of the first frame. This fact does not restraint threads 0 and 1 from working on the tiles of the second frame. However, adopting this technique creates a type of combination between frame and tile parallelism, as a result, dependencies between frames for inter-prediction and motion compensation should be taken into account. This latter is handled by OpenVVC. Moreover, since OpenVVC processes tiles independently of their frame affiliation, tile size, and load optimization at the encoder side do not actually impact the performance of OpenVVC. At the end, tiles are pipelined regardless of their size or load without waiting to finish processing the current frame.

5.2 SIMD optimization in OpenVVC

In this study, ARM Neon-based SIMD optimizations were adapted to accelerate OpenVVC targeting EGPP-based platforms. First, the x86 architecture-based SIMD intrinsics used in OpenVCC are replaced by the ARM Neon-based SIMD intrinsics. Therefore, additional adjustment was adapted due to the fact that Neon-based intrinsics are not as powerful and complete as compared to SSE or AVX intrinsics. In particular, for some cases, one SIMD instruction for x86-based was replaced with multiple Neon-based SIMD instructions. For instance, two Neon intrinsics vmull_s16 for multiply operation and vpaddq_s32 for add operation are needed to replace madd_epi16.

ESoCs used in this study support up to 128-bit SIMD registers. A 128-bit register can be loaded with 16 8-bit, 8 16-bit, 4 32-bit, or 2 64-bit data. This fact allows concurrent data processing to achieve a theoretical speedup of up to ×16 on 8-bit data. In this study, Neon-based SIMD instructions are used to optimize the high computational demanding VVC decoder modules presented in Table 1 by adapting the SIMD [36] library. Here, DST-VII, DCT-II, DCT-VIII, inter-component transform and low-frequency non-separable transform (LFNST) module of TX block of VVC decoder was accelerated using SIMD registers. TX involves several matrix operations including matrix multiplication for the inverse transformations. These matrix operations were tackled using SIMD intrinsics based on logical and mathematical operations: vand, veor, vadd, and vmul. The most benefiting modules of EP block by SIMD parallelization are luma 8-tap filters, chroma 4-tap filters, bi-directional optical flow, decoder side motion vector refinement, and prediction refinement with optical flow. These functions contain various mathematical and clipping operations which were handled by vadd, vsub, vmin, and vmax instructions. In addition, loading and storing data in larger SIMD registers helped to accelerate EP, because the prediction information of the pixels is needed multiple times in different EP functions. Then the pixel prediction inside the picture of IP block was effectively managed by storing masks, clipped, and offset value using SIMD intrinsics. Further, the edge and band filter of SAO use vceq, vadd, and vsub instructions to handle mathematical operations. Finally, ALF filters are parallelized by concurrently storing filter parameters using shuffle intrinsic. Moreover, it exploits the full capacity of SIMD register of 128 bit using load and store intrinsic instructions.

Table 1 Main functions optimized with SIMD

VVC Block	Module
TX	DST-VII, DCT-II, DCT-VIII
	Inter-component transform
	Low-frequency non-separable transform
EP	Luma 8-tap filters
	Chroma 4-tap filters
	Bi-directional optical flow
	Decoder side motion vector refinement
	Prediction refinement with optical flow
IP	DC, Planar
	Cross-component linear model
	Matrix-based intra-prediction module
In-loop filters	Edge and band filter of SAO
	ALF 7×7 diamond shape filters for the luma component
	ALF 5×5 diamond shape filters for the chroma component
	Block classification of ALF

6 Experimental results

In this section, the experimental setup, test bench used in this study, and the experimental results obtained are presented for two open-source optimized decoders VVdeC V1.3 and OpenVVC V1.0 on two EGPP-based embedded platforms.

6.1 Experimental setup

This study focusses on low-cost mobile embedded heterogeneous platforms. Therefore, two ESoC platforms, ESoC1 [37] and ESoC2 [38] have been used. ESoC1 processor consists of 8 EGPP cores running with a maximum clock speed of 2.26 GHz and 512 embedded GPU (EGPU) cores running with a maximum clock speed of 1.37 GHz. In addition, ESoC1 has 8MB of L2 cache memory, 4MB of L3 cache memory, and 32MB 256 bit random access memory with 137 GB/s speed. ESoC2 has 4 EGPP cores and 128 EGPU cores running with a maximum clock speed of 1.48GHz and 0.92 GHz, respectively. Moreover, it has 2MB L2 cache memory and 4MB 64 bit random access memory with

25.6 GB/s speed. This work is only based on EGPP cores, and GPU cores could be used in future works to further speedup the decoder. In both platforms, a gcc compiler version 7.5 with -O3 flag activated and Cmake version 3.16.5 have been used alongside an Ubuntu 18.04 operating system. Thus, this work comprehensively covers the design of applications based on this type of platforms including both, an embedded platform with limited performance, adjusted to hardware-constrained use cases, but also an embedded platform with a high computing power profile.

6.2 Test video sequences

Table 2 presents the different features of the 15 JVET common test sequences [39] used in this study. The following sequences, grouped by resolution classes, have been encoded by the VTM−11.0 reference software with 10-bit random access and 4×3 tile configuration at two QP 27 and 37. As it can be seen, only A- and B-class sequences have been included. Lower resolution class sequences were discarded as in most cases the performance obtained with them would be above the real time, following the trends proportional to the results observed with the chosen test set.

6.3 Results and analyses

Since this study focusses on analyzing the decoding performance over embedded platforms, the average energy consumption and the maximum memory usage have been also measured. To do so, two open-source optimized VVC decoders: VVdeC and OpenVVC over the two already mentioned platforms have been used.

6.3.1 Decoding performance

First, the decoding performance of OpenVVC has been studied for five combinations of frame–tile parallelization by taking advantage of the eight physical cores integrated in the ESoC1 architecture:

- 8-frame and 0-tile per frame in parallel (f8/t0) (only frame parallelism without tiles parallelism).
- 1-frame and 8-tile per frame in parallel (f1/t8).
- 2-frame and 4-tile per frame in parallel (f2/t4).
- 4-frame and 2-tile per frame in parallel (f4/t2).
- 2-frame and 8-tile per frame in parallel (f2/t8).

The default OpenVVC configuration chooses the best combinations of frame–tile parallelization. However, different configurations were presented in the study to present the performance differently for different configurations. This study has been performed to do a fair comparison between the default (best) configuration of OpenVVC and the default configuration of VVdeC frame–tile parallelization.

Figure 4 shows the average decoding performance in frames per second (fps) for HD and FHD test sequences with QP27 and QP37 on ESoC1 using OpenVVC decoder.

It can be seen from Fig. 4 that the least performing configuration among all configurations with tiles parallelism is f1/t8 and the best performing configuration is f4/t2 for all QPs, HD, and FHD sequences. f4/t2 configuration has achieved in average ×1.4 and ×1.3 fps compared to the f1/t8 configuration for FHD and HD sequences, respectively. These results mainly illustrate the gain brought considering both frame and tile parallelism compared to only frame parallelism which is constrained by the inter coding dependency. Furthermore, the configuration with only frame

Table 2 Features of the considered VVC test sequences

Class	Sequence	Resolution	# Frames	Bitdepth	Framerate
A1	Tango2	3840×2160	294	10	60
	FoodMarket4	3840×2160	300	10	60
	Campfire	3840×2160	300	10	30
A2	CatRobot1	3840×2160	300	10	60
	DaylightRoad2	3840×2160	300	10	60
	ParkRunning3	3840×2160	300	10	50
B	MarketPlace	1920×1080	300	10	60
	RitualDance	1920×1080	300	10	60
	Cactus	1920×1080	300	10	50
	BasketballDrive	1920×1080	300	10	50
	BQTerrace	1920×1080	300	10	60
	ArenaOfValor	1920×1080	300	10	60
E	FourPeople	1280×720	300	10	60
	Johnny	1280×720	300	10	60
	KristenAndSara	1280×720	300	10	60

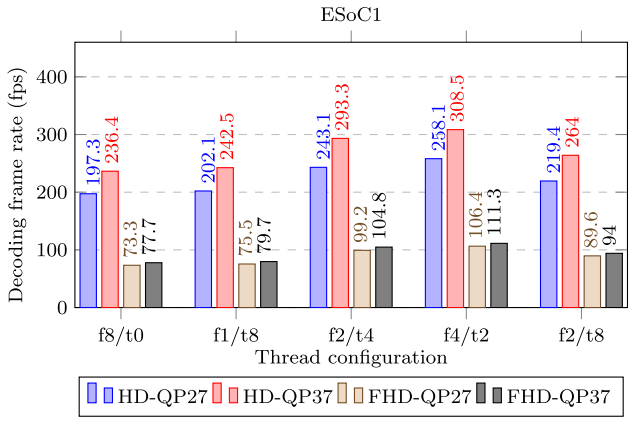

Fig. 4 Average decoding performance (fps) of OpenVVC for QP 27 and 37 sequences on the ESoC1 platform

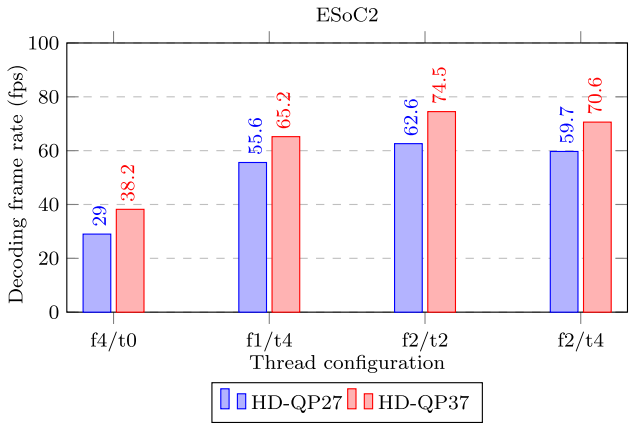

Fig. 5 Average decoding performance (fps) of the OpenVVC decoder for QPs 27 and 37 sequences on the ESoC2 platform

parallelism without tiles parallelism, f8/t0 obtained less fps than the frame and tiles parallelism-based least performing configuration, f1/t8 for all QPs, HD, and FHD sequences.

Figure 5 presents the average decoding performance in fps obtained over ESoC2 for HD sequences using both QPs. ESoC2 has four physical cores. For this reason, only four combinations of frame–title parallelization have been studied. Here, all results that have not reached the real-time decoding target have been omitted.

- 4-frame and 0-tile per frame in parallel (f4/t0) (only frame parallelism without tiles parallelism).
- 1-frame and 4-tile per frame in parallel (f1/t4).
- 2-frame and 2-tile per frame in parallel (f2/t2).
- 2-frame and 4-tile per frame in parallel (f2/t4).

In this case, and as shown in Fig. 5, the f2/t2 configuration has achieved 62.6 fps for QP27 and 74.5 fps for QP37, which is higher on average by 8.1 fps and 3.4 fps than f1/t4 and f2/t4 configurations for HD sequences, respectively. Moreover, the only frame parallelism-based configuration, f4/t0 obtained average 40% less fps than the low-performing frame–tile parallelism-based configuration f1/t4. Therefore, in the following parts of the article, only frame and tile parallelism-based configurations are presented.

The decoding performance and speedup of VVdeC and f4/t2 configuration of OpenVVC decoders for QPs 27 and 37 over ESoC1 are presented in Table 3 for all video sequences considered. It can be seen that the decoding speed obtained by both VVdeC and OpenVVC decoders is close to real-time for the UHD sequences and achieves real-time for all the FHD and HD sequences over ESoC1 using 8 cores. Therefore, the experiment for the FHD and HD sequences is presented in the following part of this study, which achieves real-time over ESoC1. Moreover, it can be seen that VVdeC has obtained slightly better fps than OpenVVC on single core configuration for all sequences at different quantification parameters QPs. However, speedup shows that OpenVVC has provided better parallelism compared to VVdeC when the number of threads has increased, which compensates for the first limitation.

In Table 4, the decoding performance of the decoders VVdeC and OpenVVC (f2/t2 configuration) for all the FHD and HD sequences with QPs 27 and 37 on the ESoC2 platform is shown. Here, both VVdeC and OpenVVC decoders have achieved real-time for HD sequences over ESoC2 using 4 cores. Therefore, in the next part of this study, the results are presented for HD sequences over ESoC2.

The average decoding performance with respect to the number of cores is presented in Fig. 6. Here, the decoding frame rates have been recorded for the OpenVVC and VVdeC decoders over ESoC1 and ESoC2. For both QPs, the average results in fps of OpenVVC and VVdeC are similar for one to four cores over ESoC2. Moreover, VVdeC has reached ×1.08 fps with respect to OpenVVC on ESoC1 and has reached the saturation point with 7 cores for HD sequences. However, for FHD sequences, the performance results of OpenVVC and VVdeC are comparable to ESoC1. The performance results follow the same pattern for both considered QPs. The impact of decoding sequences without tiles versus those with tiles, in which case this feature is deactivated in the decoder, has been compared and results in an average performance loss of 8%.

6.3.2 Memory usage

Memory usage is one of the most limiting factors and a likely bottleneck for video decoding over resource-constrained embedded hardware. This part of the study presents

Table 3 Decoding performance (fps) for the test sequences considered in QP27 (top) and QP37 (bottom) on the ESoC1 platform with 1, 2, 4, 6 and 8 cores

Seq.:QP	VVdeC (fps)/speedup					OpenVVC (fps)/speedup				
# cores	1	2	4	6	8	1	2	4	6	8
Tango2:27	3.3	6.3/1.9	12.4/3.8	18.1/5.6	22.8/7.0	3.1	6.1/2.0	12.1/3.9	17.9/5.8	22.7/7.4
FoodMarket4:27	3.3	5.9/1.8	12.4/3.8	18.2/5.6	23.0/7.0	3.0	6.1/2.0	12.0/4.0	17.6/5.8	22.6/7.5
Campfire:27	3.7	6.8/1.9	13.6/3.7	19.5/5.3	24.0/6.5	3.5	7.5/2.1	14.7/4.2	21.7/6.2	27.3/7.8
Average	3.4	6.4/1.9	12.8/3.8	18.6/5.5	23.3/6.8	3.2	6.6/2.0	12.9/4.0	19.0/6.0	24.2/7.6
CatRobot1:27	3.3	6.4/1.9	12.7/3.8	18.5/5.5	23.3/7.0	3.2	6.3/2.0	12.4/3.9	18.1/5.7	22.9/7.2
DaylightRoad2:27	3.1	6.0/1.9	11.7/3.8	17.2/5.5	21.5/7.0	2.8	5.7/2.0	11.2/4.0	16.4/5.9	20.9/7.5
ParkRunning3:27	2.4	4.3/1.8	8.5/3.6	12.4/5.2	15.7/6.7	2.1	4.1/2.0	8.1/3.9	11.9/5.7	15.1/7.3
Average	2.9	5.6/1.9	11.0/3.7	16.0/5.5	20.2/6.9	2.7	5.3/2.0	10.5/3.9	15.5/5.8	19.7/7.3
MarketPlace:27	12.5	23.8/1.9	46.0/3.7	66.5/5.3	82.7/6.6	11.1	21.8/2.0	43.2/3.9	63.3/5.7	79.6/7.2
RitualDance:27	13.9	26.3/1.9	51.3/3.7	73.2/5.2	91.0/6.5	13.0	25.2/1.9	49.8/3.8	73.7/5.7	90.4/7.0
Cactus:27	16.6	30.7/1.8	59.0/3.6	85.1/5.1	103.0/6.2	15.4	29.6/1.9	57.9/3.8	82.6/5.4	94.3/6.1
BasketballDrive:27	12.2	22.5/1.8	43.7/3.6	63.5/5.2	77.5/6.4	11.1	21.5/1.9	42.8/3.9	63.2/5.7	79.6/7.2
BQTerrace:27	12.9	24.3/1.9	47.4/3.7	68.7/5.3	82.6/6.4	11.7	23.1/2.0	45.7/3.9	67.4/5.8	85.2/7.3
ArenaOfValor:27	15.5	28.5/1.8	55.0/3.6	79.2/5.1	96.8/6.3	14.0	27.3/1.9	53.8/3.8	78.9/5.6	96.8/6.9
Average	13.9	26.0/1.9	50.4/3.6	72.7/5.2	88.9/6.4	12.7	24.7/1.9	48.9/3.8	71.5/5.6	87.6/6.9
FourPeople:27	52.2	98.7/1.9	185.5/3.6	257.1/4.9	271.0/5.2	45.8	88.8/1.9	175.4/3.8	252.1/5.5	285.7/6.2
Johnny:27	54.3	104.4/1.9	196.7/3.6	272.2/5.0	288.2/5.3	48.9	92.6/1.9	180.7/3.7	238.1/4.9	252.1/5.2
KristenAndSara:27	52.0	97.3/1.9	184.0/3.5	257.5/5.0	294.4/5.7	47.6	89.6/1.9	177.5/3.7	250.0/5.3	272.7/5.7
Average	52.8	100.1/1.9	188.8/3.6	262.3/5.0	284.5/5.4	47.5	90.3/1.9	177.9/3.7	246.7/5.2	270.2/5.7
Tango2:37	4.1	8.2/2.0	16.0/3.9	23.2/5.6	29.7/7.2	4.1	8.2/2.0	15.1/3.7	23.7/5.8	30.0/7.3
FoodMarket4:37	4.0	7.9/2.0	15.8/3.9	22.9/5.7	29.2/7.2	3.9	7.8/2.0	15.3/3.9	22.5/5.7	28.2/7.2
Campfire:37	4.8	9.0/1.9	17.8/3.7	25.8/5.4	31.9/6.7	4.7	9.7/2.0	19.0/4.0	28.1/5.9	34.3/7.2
Average	4.3	8.4/1.9	16.5/3.8	24.0/5.6	30.3/7.0	4.3	8.6/2.0	16.5/3.9	24.8/5.8	30.8/7.2
CatRobot1:37	4.1	8.2/2.0	16.1/3.9	23.3/5.7	29.4/7.1	4.2	8.3/2.0	16.2/3.9	23.8/5.7	30.0/7.2
DaylightRoad2:37	4.1	8.0/2.0	16.0/3.9	23.4/5.7	29.6/7.2	3.8	7.8/2.0	15.4/4.0	22.6/5.9	28.5/7.4
ParkRunning3:37	3.0	5.8/1.9	11.5/3.8	16.9/5.6	21.3/7.1	2.8	5.5/2.0	10.8/3.9	16.1/5.8	20.6/7.4
Average	3.8	7.3/2.0	14.5/3.9	21.2/5.6	26.8/7.1	3.6	7.2/2.0	14.2/3.9	20.8/5.8	26.4/7.3
MarketPlace:37	16.9	33.2/2.0	64.3/3.8	92.6/5.5	109.9/6.5	16.0	31.0/1.9	61.2/3.8	89.8/5.6	107.1/6.7
RitualDance:37	18.0	34.9/1.9	67.2/3.7	97.0/5.4	119.8/6.6	16.6	32.4/1.9	64.7/3.9	95.8/5.8	117.2/7.0
Cactus:37	21.7	40.3/1.9	77.3/3.6	111.0/5.1	133.0/6.1	20.5	39.5/1.9	76.9/3.8	111.1/5.4	124.0/6.0
BasketballDrive:37	14.7	28.1/1.9	55.0/3.7	79.6/5.4	98.2/6.7	13.7	26.6/1.9	52.9/3.9	77.5/5.6	97.1/7.1
BQTerrace:37	15.8	30.6/1.9	59.7/3.8	87.4/5.5	104.1/6.6	15.5	30.3/2.0	60.1/3.9	88.2/5.7	109.1/7.0
ArenaOfValor:37	19.9	38.5/1.9	74.4/3.7	106.6/5.4	128.7/6.5	19.1	37.9/2.0	74.6/3.9	108.3/5.7	130.4/6.8
Average	17.8	34.3/1.9	66.3/3.7	95.7/5.4	115.6/6.5	16.9	33.0/1.9	65.1/3.8	95.1/5.6	114.2/6.7
FourPeople:37	59.2	116.7/2.0	219.1/3.7	299.4/5.1	314.5/5.3	55.5	108.3/2.0	211.3/3.8	306.1/5.5	344.8/6.2
Johnny:37	64.2	123.2/1.9	231.5/3.6	319.5/5.0	355.0/5.5	57.8	113.2/2.0	219.0/3.8	297.0/5.1	319.1/5.5
KristenAndSara:37	60.0	115.4/1.9	217.5/3.6	302.1/5.0	338.2/5.6	52.9	103.8/2.0	204.1/3.9	294.1/5.6	322.6/6.1
Average	61.1	118.4/1.9	222.7/3.6	307.0/5.0	335.9/5.5	55.4	108.4/2.0	211.4/3.8	299.1/5.4	328.9/5.9

the maximum memory (in MB) consumed by OpenVVC and VVdeC over ESoC1 and ESoC2 for the FHD/HD sequences with two QPs (27, 37). In Fig. 7, the average maximum memory usage for different OpenVVC configurations and VVdeC is shown. Here, for both FHD and HD sequences, f1/t8 and f2/t8 configurations have used the least and the most memory, respectively. This behavior is expected that with the increase of the number of frames decoded in parallel, the memory usage increases since the large part memory of the decoder is related to the decoded frame. However, VVdeC requires in average ×2.1 more memory for HD sequences and ×2.7 more memory for FHD sequences than the OpenVVC f2/t8 configuration over ESoC1. In addition, the scenario is the same over ESoC2 platform where f2/t4 configuration requires in average a maximum memory of 59.7MB for HD sequences. Furthermore, VVdeC requires

Table 4 Decoding performance (in fps) for the considered HD and FHD test sequences at QP27 (top) and QP37 (bottom) on the ESoC2 platform with 1, 2, 3, and 4 cores

Seq.:QP	VVdeC (fps)				OpenVVC (fps)			
# cores	1	2	3	4	1	2	3	4
MarketPlace:27	4.6	8.8	12.9	16.5	4.3	8.2	12.0	15.5
RitualDance:27	5.4	10.0	14.6	18.7	5.0	9.7	14.1	18.2
Cactus:27	6.4	11.6	17.0	21.5	5.8	11.3	16.4	21.0
BasketballDrive:27	4.6	8.6	12.5	16.2	4.3	8.5	12.4	16.1
BQTerrace:27	4.8	9.0	13.2	17.1	4.6	8.9	13.0	16.7
ArenaOfValor:27	6.0	10.9	15.9	20.1	5.5	10.7	15.7	20.0
Average	5.3	9.8	14.3	18.4	4.9	9.6	13.9	17.9
FourPeople:27	21.3	39.0	56.4	69.4	18.3	35.4	50.9	64.4
Johnny:27	21.9	40.4	57.9	71.4	18.1	34.7	50.5	63.2
KristenAndSara:27	20.7	38.1	55.1	68.3	18.0	34.8	50.3	64.0
Average	21.3	39.2	56.5	69.7	18.1	34.9	50.6	63.8
MarketPlace:37	6.3	12.1	17.6	22.5	5.9	11.5	16.9	21.7
RitualDance:37	7.0	13.1	17.7	24.5	6.4	12.4	18.0	23.1
Cactus:37	8.2	15.4	22.5	28.4	7.7	14.9	21.4	27.6
BasketballDrive:37	5.6	10.7	15.7	20.3	5.2	10.3	14.9	19.2
BQTerrace:37	5.9	11.4	16.6	21.4	5.8	11.4	16.6	21.4
ArenaOfValor:37	7.8	14.7	21.4	26.1	7.5	14.5	21.0	26.7
Average	6.8	12.9	18.6	23.9	6.4	12.5	18.1	23.3
FourPeople:37	24.8	46.2	65.7	80.9	21.8	42.1	61.0	77.3
Johnny:37	25.6	48.1	68.6	83.6	21.9	42.3	60.9	76.9
KristenAndSara:37	23.8	44.5	63.8	77.9	20.6	39.6	57.0	72.6
Average	24.7	46.2	66.0	80.8	21.4	41.3	59.6	75.6

on average ×2.4 more memory for HD sequences than the OpenVVC f2/t4 configuration on ESoC2 platform. It can be concluded from the results that OpenVVC requires notably less memory compared to VVdeC. That fact is attributed to both its carefully designed local structure (as presented in [10], Section III-C) and the efficient management of the picture buffer pool, also outlined in [10], Section III-B. Therefore, OpenVVC provides a great advantage and is suitable for resource-constrained embedded platforms.

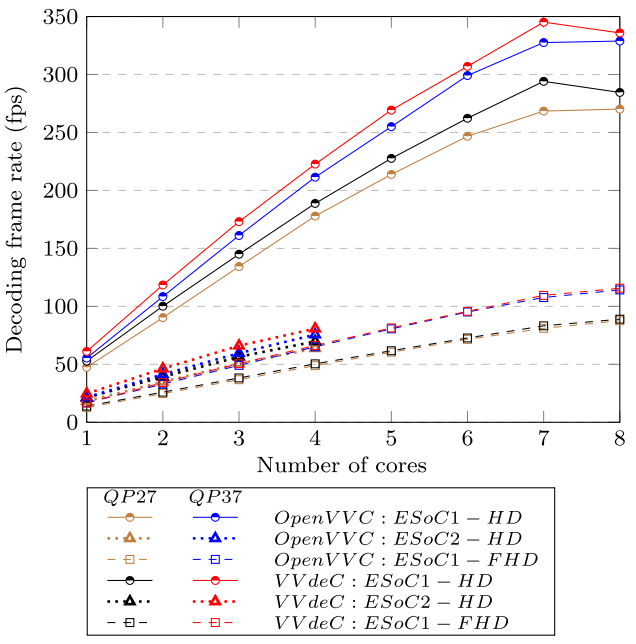

Fig. 6 Average decoding performance (in fps) of OpenVVC, in brown QP 27 and blue QP 37, and VVdeC, in black QP27 and red QP37, for 1 to 8 cores

6.3.3 Energy consumption

Energy consumption is another important factor for video processing operation over embedded platforms. The characterization of the impact of the software on hardware is essential to obtain models that allow the identification of the optimal working points of video decoders [41]. In this study, the energy consumption was calculated as follows: 1) the power consumption is taken (in mW) after decoding each frame using the built-in power monitor of both ESoC, 2) the average power consumption of the entire sequence is multiplied by the total time in seconds spent decoding the sequence. The average energy consumption in J with different configurations of OpenVVC and VVdeC decoders is shown in Fig. 8. Here, OpenVVC and VVdeC have consumed comparable average energy in ESoC1 for all configurations. VVdeC has consumed on average ×1.17 higher energy for the HD sequences and ×1.04 higher energy for

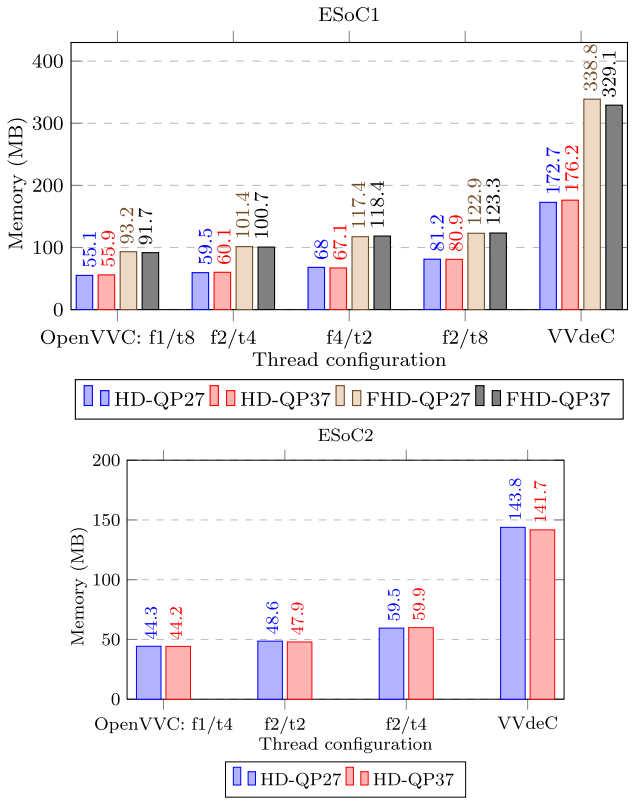

Fig. 7 Average maximum memory (in MB) used for QPs 27 and 37 sequences over ESoC1 (top) and ESoC2 (bottom)

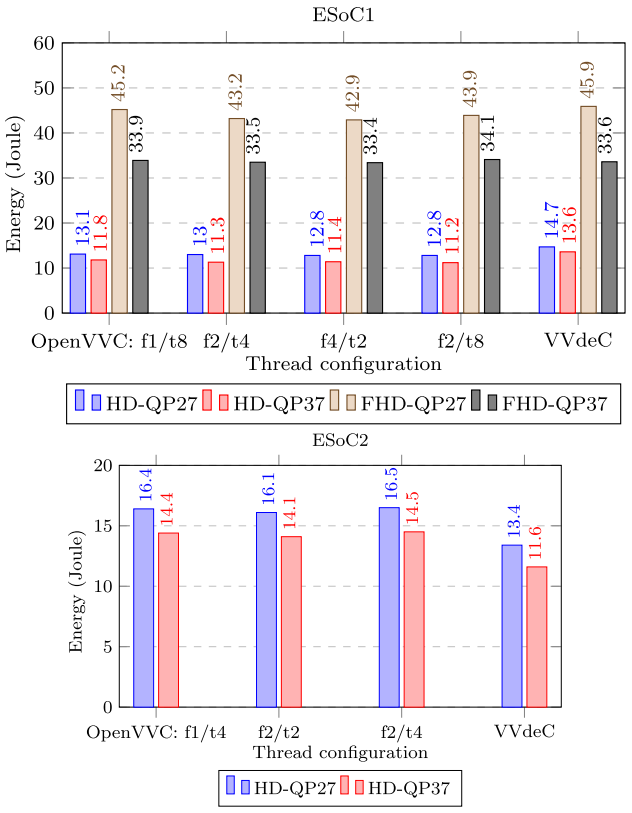

Fig. 8 Average energy (J) consumed for QP 27 and 37 sequences on ESoC1 (top) and ESoC2 (bottom) platforms

the FHD sequences with respect to the f4/t2 configuration of OpenVVC consumption on ESoC1 platform. Similar to the implementation over ESoC1, the f2/t2 configuration has used the least amount of average energy over ESoC2. Furthermore, the average energy consumption of OpenVVC is slightly higher than VVdeC consumption in comparison to ESoC2.

6.3.4 Comparison between OpenVVC and VVdeC decoders

Both open-source optimized video decoders OpenVVC and VVdeC have reached real-time for FHD and HD sequences over ESoC1 using 8 cores. In addition, both solutions present results close to real-time performance for UHD sequences on ESoC1 platform. Furthermore, the OpenVVC and VVdeC decoders achieved an average of 22 fps for QP27 and 28.5 fps for QP37 using 8 cores. Tables 5 and 6 show the average performance (in fps) of OpenVVC and VVdeC using different number of threads on ESoC1 and ESoC2. OpenVVC introduces slightly more runtime complexity compared to VVdeC: 3% for UHD, 5% for FHD and 12% for HD sequences in both platforms.

To summarize, there are three important parameters to take into consideration to select a video decoder for an embedded platform with limited hardware resources: the performance (fps), the energy consumed to decode a video, and the memory used. The performance of the decoders compared in this paper (VVdeC and OpenVVC) is very similar and only a small improvement is achieved in VVdeC while the number of cores remains low. The energy consumed to decode a sequence is also very similar between both decodes. Finally, OpenVVC consumes less memory than VVdeC with a factor greater than ×2.11. This significant reduction makes the OpenVVC decoder a suitable option to implement a VVC conformant video decoder in a multi-core platform with limited resources.

7 Conclusion

This paper presents two open-source VVC decoders: OpenVVC and VVdeC, optimized for low-cost resource-constrained embedded platforms. Here, OpenVVC and VVdeC have been optimized at the level of data processing using SIMD operations. In addition, tile- and frame-based parallelizations have been implemented in OpenVVC. Both decoders have achieved 15 to 34 fps for UHD sequences with QP 27 and 37, and achieved real-time decoding for all configurations of FHD and HD sequences over ESoC1 using

Table 5 Average performance (fps) of OpenVVC and VVdeC decoders on ESoC1 platform with 1 and 8 cores

	VVdeC (fps)		OpenVVC (fps)		VVdeC/OpenVVC	
# cores	1	8	1	8	1	8
UHD:QP27	3.17	21.72	2.94	21.95	108%	99%
UHD:QP37	4.02	28.52	3.93	28.59	102%	100%
FHD:QP27	13.94	88.94	12.72	87.64	110%	101%
FHD:QP37	16.66	115.62	16.92	114.15	98%	101%
HD:QP27	52.81	284.53	47.45	270.18	111%	105%
HD:QP37	61.14	335.90	55.39	328.85	110%	102%

Table 6 Average performance (fps) of OpenVVC and VVdeC decoders on ESoC2 platform with 1 and 4 cores

	VVdeC (fps)		OpenVVC (fps)		VVdeC/OpenVVC	
# cores	1	4	1	4	1	4
UHD:QP27	1.20	4.29	1.15	4.20	104%	102%
UHD:QP37	1.51	5.49	1.49	5.43	101%	101%
FHD:QP27	5.29	18.36	4.93	17.92	107%	102%
FHD:QP37	6.81	23.85	6.43	23.31	106%	102%
HD:QP27	21.31	69.74	18.12	63.83	118%	109%
HD:QP37	24.72	80.78	21.43	75.63	115%	107%

8 cores. Furthermore, 16 to 28 fps have been obtained for FHD sequences for QPs 27 and 37, and real-time decoding has been obtained for all HD sequences by OpenVVC and VVdeC on ESoC2 using 4 cores. Furthermore, the experimental results for the two most important factors of the embedded platform: the average energy consumption and maximum memory usage of both decoders were presented for ESoC1 and ESoC2. VVdeC has consumed on average ×2.74 and ×2.96 memory compared to the OpenVVC f4/t2 configuration on ESoC1 and the f2/t2 configuration on ESoC2, respectively. For average energy usages, VVdeC consumed on average ×1.11 energy with respect to the OpenVVC f4/t2 configuration on ESoC1 and ×0.83 energy with respect to the OpenVVC f2/t2 configuration on ESoC2.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availability The data that support the findings of this study are available from the first author and corresponding author, upon reasonable request.

References

1. *Versatile video coding,* Recommendation ITU-T H.266 and ISO/IEC 23090-3 (VVC), ITU-T and ISO/IEC JTC 1, Jul. (2020)
2. *High efficiency video coding,* Recommendation ITU-T H.265; (2013)
3. Feldmann, C.: "Versatile video coding hits major milestone," *Bitmovin*. Accessed on: May 27, (2022). [Online]. https://bitmovin.com/compression-standards-vvc-2020
4. *VVC test model*. Accessed on: May 15, (2022), [Online]. https://mpeg.chiariglione.org/standards/mpeg-i/versatile-video-coding/
5. Chi, C.C., Alvarez-Mesa, M., Bross, B., Juurlink, B., Schierl, T.: "SIMD acceleration for HEVC decoding''. IEEE Trans Circ. Syst. Video Technol. **25**(5), 841–855 (2015). https://doi.org/10.1109/TCSVT.2014.2364413
6. Amestoy, T., Hamidouche, W., Bergeron, C., Menard, D.: "Quality-driven dynamic VVC frame partitioning for efficient parallel processing," in *2020 IEEE Int. Conf. Image Process.*, Abu Dhabi, United Arab Emirates, pp. 3129-3133, (2020). https://doi.org/10.1109/ICIP40778.2020.9190928
7. Gudumasu, S., Bandyopadhyay, S., He, Y.: "Software-based versatile video coding decoder parallelization," in *Proc. 11th ACM Multimedia Syst. Conf.*, New York, NY, USA, pp. 202-212, (2020). https://doi.org/10.1145/3339825.3391871
8. Koziri, M., Papadopoulos, P. K., Tziritas, N., Dadaliaris, A., Loukopoulos, T., Khan, S. U., Xu, C-Z.: "Adaptive tile parallelization for fast video encoding in HEVC," in *2016 IEEE Int. Conf. Internet of Things* and *IEEE Green Computing and Communications* and *IEEE Cyber, Physical and Social Computing* and *IEEE Smart Data*, pp. 738-743, (2016). https://doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2016.156

9. *OpenVVC software repository,* Accessed on: Mar. 19, (2022). [Online]. https://github.com/OpenVVC/OpenVVC
10. Amestoy, T. , Cabarat, P., Gautier, G., Hamidouche, W., Menard, D.: "OpenVVC: a lightweight software decoder for the versatile video coding standard," *arXiv preprint*, arXiv:2205.12217, (2022)
11. *Fraunhofer HHI VVdeC software repository,* Accessed on: Mar. 6, (2022). [Online]. https://github.com/fraunhoferhhi/vvdec
12. Karwowski, D.: Precise probability estimation of symbols in VVC CABAC entropy encoder. IEEE Access **9**, 65361–65368 (2021). https://doi.org/10.1109/ACCESS.2021.3075875
13. Zhao, X., Kim, S.-H., Zhao, Y., Egilmez, H.E., Koo, M., Liu, S., Lainema, J., Karczewicz, M.: Transform coding in the VVC standard. IEEE Trans. Circuits Syste. Video Technol. **31**(10), 3878–3890 (2021). https://doi.org/10.1109/TCSVT.2021.3087706
14. Ghaznavi-Youvalari, R., Lainema, J.: "Joint cross-component linear model for chroma intra prediction," in *IEEE 22nd Int. Workshop Multimedia Signal Process.*, pp. 1-5, (2020). https://doi.org/10.1109/MMSP48831.2020.9287167
15. Bross, B., Wang, Y.-K., Ye, Y., Liu, S., Chen, J., Sullivan, G.J., Ohm, J.-R.: Overview of the versatile video coding (VVC) standard and its applications. IEEE Trans. Circ. Syst. Video Technol. **31**(10), 3736–3764 (2021). https://doi.org/10.1109/TCSVT.2021.3101953
16. Yang, H., Chen, H., Chen, J., Esenlik, S., Sethuraman, S., Xiu, X., Alshina, E., Luo, J.: Subblock-based motion derivation and inter prediction refinement in the versatile video coding standard. IEEE Trans. Circ. Syst. Video Technol. **31**(10), 3862–3877 (2021). https://doi.org/10.1109/TCSVT.2021.3100744
17. Lu, T. , Pu, F., Yin, P., McCarthy, S., Husak, W., Chen, T., Francois, E., Chevance, C., Hiron, F., Chen, J., Liao, R.-L., Ye, Y., Luo, J.: "Luma mapping with chroma scaling in versatile video coding," in *2020 Data Compression Conf.*, Snowbird, UT, USA, pp. 193-202, (2020). https://doi.org/10.1109/DCC47342.2020.00027
18. Karczewicz, M., Hu, N., Taquet, J., Chen, C.-Y., Misra, K., Andersson, K., Yin, P., Lu, T., François, E., Chenet, J.: VVC in-loop filters. IEEE Trans. Circ. Syst. Video Technol. **31**(10), 3907–3925 (2021). https://doi.org/10.1109/TCSVT.2021.3072297
19. Saha, A., Chavarrías, M., Pescador, F., Groba, Á.M., Chassaigne, K., Cebrián, P.L.: Complexity analysis of a versatile video coding decoder over embedded systems and general purpose processors. Sensors **21**, 3320 (2021). https://doi.org/10.3390/s21103320
20. Yan, L., Duan, Y., Sun, J., Guo, Z.: "Implementation of HEVC decoder on x86 processors with SIMD optimization," in *2012 Visual Commun. Image Process.*, pp. 1-6, (2012). https://doi.org/10.1109/VCIP.2012.6410845
21. de Souza, D. F., Ilic, A., Roma, N., Sousa, L.: "HEVC in-loop filters GPU parallelization in embedded systems," in *2015 Int. Conf. Embedded Computer Systems: Architectures, Modeling, and Simulation*, pp. 123-130, (2015). https://doi.org/10.1109/SAMOS.2015.7363667
22. de Souza, D.F., Ilic, A., Roma, N., Sousa, L.: GPU-assisted HEVC intra decoder. J. Real-Time Image Process. **12**(2), 531–547 (2016). https://doi.org/10.1007/s11554-015-0519-1
23. Han, X., Wang, S., Ma, S., Gao, W.: "Optimization of motion compensation based on GPU and CPU for VVC decoding," in *2020 IEEE Int. Conf. Image Process.*, pp. 1196-1200, (2020). https://doi.org/10.1109/ICIP40778.2020.9190708
24. Wieckowski, A., Hege, G., Bartnik, C., Lehmann, C., Stoffers, C., Bross, B., Marpe, D.: "Towards a live software decoder implementation for the upcoming versatile video coding (VVC) codec," in *2020 IEEE Int. Conf. Image Process.*, pp. 3124-3128, (2020). https://doi.org/10.1109/ICIP40778.2020.9191199
25. Zhu, B., Liu, S., Liu, Y., Luo, Y., Ye, J., Xu, H., Huang, Y., Jiao, H., Xu, X., Zhang, X., Gu, C.: "A real-time H.266/VVC software decoder," in *2021 IEEE Int. Conf. Multimedia Expo*, pp. 1-6, (2021). https://doi.org/10.1109/ICME51207.2021.9428470
26. Li, Y., Liu, S., Chen, Y., Zheng, Y., Chen, S., Zhu, B., Lou, J.: "An optimized H.266/VVC software decoder on mobile platform," in *2021 Picture Coding Symp.* , pp. 1-5, (2021). https://doi.org/10.1109/PCS50896.2021.9477484
27. *ARM Developer, Neon*. Accessed on: May 03, 2022. Online]. https://developer.arm.com/architectures/instruction-sets/simd-isas/neon
28. Sullivan, G.: "Deployment status of the VVC standard," ISO/IEC JTC1/SC29/WG11 JVET document Y0021 (JVET-Y0021), Teleconference, January, (2022)
29. *VLC media player: VideoLAN, a project and a non-profit organization,* Accessed on: May 23, (2022). [Online]. https://www.videolan.org/
30. *GPAC: multimedia open source project,* Accessed on: May 23, (2022). [Online]. https://gpac.wp.imt.fr/
31. *Ffmpeg: a complete, cross-platform solution to record, convert and stream audio and video,* Accessed on: May 23, (2022). [Online]. https://ffmpeg.org/
32. *VVC VTM reference software repository,* Accessed on: Mar. 14, (2022). [Online]. https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM
33. Wieckowski, A., Lehmann, C., Bross, B., Marpe, D., Biatek, T., Raulet, M., Le Feuvre, J.: "A complete end to end open source toolchain for the versatile video coding (VVC) Standard," in *2021 Proc. 29th ACM Int. Conf. Multimedia, Association Computing Machinery*, New York, NY, USA, 3795-3798. https://doi.org/10.1145/3474085.3478320
34. Saha, A., Chavarrías, M., Aranda, V., Garrido, M.J., Pescador, F.: "Implementation of a real-time versatile video coding decoder based on VVdeC over an embedded multi-core platform''. IEEE Trans. Consumer Electron. (2022). https://doi.org/10.1109/TCE.2022.3202512
35. *Fraunhofer HHI VVdeC software repository, Releases vvdec-0.2.0.0,* Accessed on: Sep. 12, (2022). [Online]. https://github.com/fraunhoferhhi/vvdec/releases/tag/v0.2.0.0
36. Nemerson, E.: "Transitioning SSE/AVX code to NEON with SIMDe," Accessed on: Jun. 7, (2022). [Online]. https://simd-everywhere.github.io/blog/2020/06/22/transitioning-to-arm-with-simde.html
37. *NVIDIA Jetson AGX Xavier developer kit, user guide,* DA_09403_003, December 17, (2019). [Online]. https://developer.nvidia.com/jetson-agx-xavier-developer-kit-user-guide
38. *NVIDIA Jetson Nano developer kit, user guide,* DA_09402_004, January 15, (2020). [Online]. https://developer.nvidia.com/embedded/dlc/Jetson_Nano_Developer_Kit_User_Guide
39. Bossen, F., Boyce, J., Li, X., Seregin, V., Sühring, K.: "JVET Common Test Conditions and Software Reference Configurations for SDR Video,'' Document JVET-N1010. JVET of ITU-T, Geneva (2019)
40. Saha, A., Roma, N., Chavarrías, M., Dias, T., Pescador, F., Aranda. V.: "GPU-based parallelisation of a versatile video coding adaptive loop filter in resource-constrained heterogeneous embedded platform." J. Real-Time Image Proc. 20, 43 (2023). https://doi.org/10.1007/s11554-023-01300-z
41. Le Gonidec, O., Chavarrias, M., Saha, A., Rosa, G., Pescador, F.: "Energy Efficient Versatile Video Coding Decoder Using Lightweight Regression Models". 26th Euromicro Conference Series on Digital System Design (DSD) and 49th Euromicro Conference Series on Software Engineering and Advanced Applications (SEAA), Durres, Albania, 2023. Pending publication