From HEVC to VVC: the First Development Steps of a Practical Intra Video Encoder

Marko Viitanen, *Member, IEEE*, Joose Sainio, *Member, IEEE*, Alexandre Mercat, *Member, IEEE*, Ari Lemmetti, *Member, IEEE*, and Jarno Vanne, *Member, IEEE*

***Abstract*—Versatile Video Coding (VVC/H.266) is an emerging successor to the widespread High Efficiency Video Coding (HEVC/H.265) and is shown to double the coding efficiency for the same subjective visual quality. Nevertheless, VVC still adopts the similar hybrid video coding scheme as HEVC and thereby sets the scene for reusing many HEVC coding tools and techniques as is or with minor modifications. This paper explores the feasibility of developing a practical software VVC intra encoder from our open-source Kvazaar HEVC encoder. The outcome of this work is called uvg266 VVC intra encoder that is distributed under the same permissive 3-clause BSD license as Kvazaar. uvg266 inherits the optimized coding flow of Kvazaar and all upgradable Kvazaar intra coding tools, but it also introduces basic VVC intra coding tools not available in HEVC. To the best of our knowledge, this is the first work to describe the implementation details of upgrading an HEVC encoder to a VVC encoder. The rapid development time with promising coding performance make our proposal a viable approach over the encoder development from scratch.**

***Index Terms*—Code reuse, encoder implementation, High Efficiency Video Coding (HEVC), Versatile Video Coding (VVC), video codec.**

I. Introduction

OUR modern society is surrounded by a myriad of media applications where digital video is of the essence. This trend has resulted in snowballing growth of video data that has been estimated to account for 82% of all global IP traffic [1]. On top of that, the current growth rate of video shows no signs of deceleration thanks to an increasing plurality of new products and services that continuously seek for enhanced consumer experience.

Over the last three decades, ISO/IEC MPEG and ITU-T VCEG have released a series of international video coding standards to mitigate the phenomenal growth of video transmission and storage requirements. The latest standard, *Versatile Video Coding* (*VVC*/H.266), was ratified in 2020 [2] as the successor to the famous *High Efficiency Video Coding* (*HEVC*/*H.265*) [3]. VVC aims for 50% higher coding efficiency for the same subjective visual quality but the coding gain does not come without the added computational complexity as VVC is found to be from 7.4× up to 34.0× as complex as HEVC [4].

HEVC has been adopted for more than 2 billion devices and around 50% of broadcast and video streaming professionals are already using it [5]. Hence, it is likely that VVC will also gain a firm foothold this decade, though efficient encoder implementations will be key to its wider success.

Currently, there are two well-known open-source VVC encoders: *VVC test model* (*VTM*) [6] and *Fraunhofer Versatile Video Encoder* (*VVenC*) [7]. VTM is the VVC reference encoder that implements all normative VVC coding tools, but it is not designed for practical encoding. VVenC is an optimized implementation of VVC encoder for practical encoding, but it is currently unable to reach real-time coding speed [8].

Altogether, there are three conceivable approaches to developing a practical VVC encoder: 1) implementing an encoder from scratch; 2) optimize VTM as is the case with VVenC; or 3) converting a practical HEVC encoder into a VVC encoder. Starting the encoder from scratch allows for customized data structures but it tends to be the most time-consuming approach as a complete encoder can take more than hundred thousand lines of code. On the other hand, optimizing VTM easily requires design compromises that could result in suboptimal performance due to legacy of standardization process, e.g., no attention was paid on parallelization. This paper focuses on the third option. VVC can be considered a superset of HEVC, so reusing the optimized coding flow, tool, and techniques can improve productivity without performance compromises.

VVC encoders can be developed and benchmarked under the following four VVC *common test conditions* (*CTC*) [9]: *all intra* (*AI*), *low delay P* (*LDP*), *low delay B* (*LDB*), and *random access* (*RA*). AI coding is the simplest of these cases as it only includes spatial prediction and therefore all frames are independent of each other. LDP adds unidirectional temporal prediction and LDB bidirectional temporal prediction. Finally, RA allows encoding the frames in an arbitrary order. This paper only considers AI condition due to its simplicity. Efficient intra encoding tools are also vital in the other conditions.

This paper introduces a step-by-step workflow to upgrade an HEVC encoder to a VVC intra encoder. Kvazaar HEVC encoder [10] is used as a design entry point for this study. First, we make Kvazaar intra coding tools compatible with VVC and then we include a carefully selected set of new VVC tools based on their estimated performance impact and implementation effort. The outcome of this work is called uvg266 VVC intra

Manuscript submitted September 30th, 2021.

This work was supported in part by the AI for situational Awareness (AISA) project led by Nokia and funded by Business Finland.

M. Viitanen, J. Sainio, A. Mercat, A. Lemmetti, and J. Vanne are with Ultra Video Group, Tampere University, Tampere 33101, Finland. (e-mail: marko.viitanen@tuni.fi; joose.sainio@tuni.fi; alexandre.mercat@tuni.fi; ari.lemmetti@tuni.fi; jarno.vanne@tuni.fi).

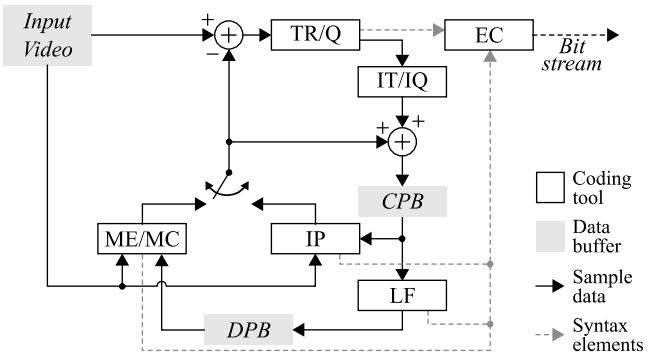

Fig. 1. Simplified block diagram of a VVC encoder.

encoder that will be distributed online under the 3-clause BSD license at:

https://github.com/ultravideo/uvg266.

To the best of our knowledge, this is the first paper to describe the implementation details when converting a HEVC encoder into a VVC encoder. This software case study can also be applied to hardware encoder implementations [11] and thereby deploy uvg266 in embedded consumer devices in the coming years.

The rest of this paper is structured as follow. Section II gives a comparative overview of the HEVC and VVC standards. Section III describes the existing VVC implementations. Section IV introduces our design methodology for HEVC to VVC encoder conversion and the implementation is detailed in Section V. Section VI describes the experimental setup and Section VII reports our results. Finally, Section VIII presents future work and Section IX concludes the paper.

II. Comparison of VVC and HEVC Coding Tools

Fig. 1 depicts an overview of the VVC encoder architecture. Both VVC and HEVC encoding processes are based on the well-known block-based hybrid video coding scheme that is composed of the following five stages: *intra prediction* (*IP*), *motion estimation and compensation* (*ME/MC*) a.k.a. inter prediction, *forward/inverse transform and quantization* (*TR/Q*), *entropy coding* (*EC*), and *loop filtering* (*LF*). In the AI case, the ME/MC stage is omitted.

Table I summarizes the main coding tools of HEVC and VVC and whether the tool is implemented in uvg266. Generally speaking, VVC has adopted many new coding tools in each coding stage. Please refer to VVC algorithm description [12] and specification [13] by JVET for further information. The tools included in this work are detailed next.

A. Entropy Coding

HEVC and VVC utilize *Context Adaptive Binary Arithmetic Coding* (*CABAC*) as an entropy reduction tool. It compresses individual bits according to the context of the bit. VVC adds an extra field called *rate* for each context, changing the rate of state change when a bit is encoded. Thus, how fast the probability of the symbol is changed depends on the context, which makes more probable symbols more cost-effective as they can be coded with fractional bits based on the statistics.

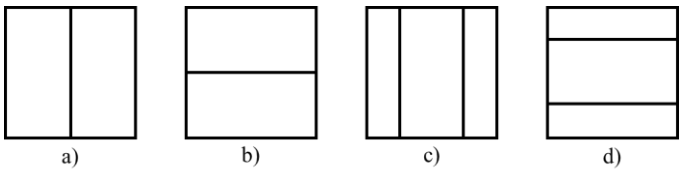

Fig. 2. VVC Binary and Ternary partition splits. a) Binary vertical. b) Binary horizontal. c) Ternary vertical. d) Ternary horizontal.

B. Block Structures

The basic block structure has not been changed from that of the HEVC and the same top-down block splitting mechanisms are still used. The largest block or the *coding tree unit* (*CTU*) has been increased up to 128×128 luma pixels and the quadtree splitting can be used in the same way as in HEVC. For the non-square block splitting after the quadtree, the previous *symmetric motion partitions* (*SMPs*) and *asymmetric motion partitions* (*AMPs*) have been replaced with the new binary and ternary splits, jointly called *multi type tree* (*MTT*), that are depicted in Fig. 2. Additionally, VVC optionally allows a separate block structure for chroma channels in intra slices with *chroma separate tree*.

C. Intra Prediction

The DC and planar prediction modes are the same as in HEVC. On the other hand, the amount of angular intra modes was roughly doubled from 33 to 65. The number of *most probable modes* (*MPM*) has been increased from 3 to 6 but the selection process is not significantly modified from that of HEVC. There are, however, potentially more bits to signal.

Position dependent intra prediction combination (*PDPC*) is a mandatory intra filter, which is applied for DC, Planar, and angular intra modes where the filter scale would be non-negative, without any signaling after the prediction.

D. Transform

The basic transform is largely the same between HEVC and VVC, except VVC uses *discrete cosine transform II* (*DCT2*) for all *coding units* (*CUs*) by default and the maximum size is increased to 64×64. Additionally, VVC introduces two additional DCT/*discrete sine transform* (*DST*) algorithms, namely DCT8 and DST7. The selection process for the different transforms is called *multiple transform selection* (*MTS*) with two available selection modes. *Implicit MTS* will select the transforms implicitly for certain blocks effectively adding zero processing and signaling cost for slight improvement. *Explicit MTS* utilizes a search to select between four MTS modes for each CU, applying DCT8 and DST7 horizontally and vertically in different combinations.

Transform skip is changed slightly from HEVC to VVC; transform shift is removed and the coefficient coding is done using separate contexts.

Finally, VVC introduces *Joint Coding of Chroma Residuals* (*JCCR*) that allows coding chroma residual with only one set of coefficients that are then used to reconstruct both of the chroma residuals. For intra CUs, the residuals can have the same values or one of the residuals can have half of the intensity.

TABLE I
MAIN CODING TOOLS OF HEVC, VVC, AND THEIR IMPLEMENTATION STATUS IN UVG266

	HEVC	VVC	uvg266
Block partitioning	▪ Coding Tree Unit (CTU) quadtree (QT) structure	▪ CTU quadtree structure with nested multi-type tree (QT+MTT)	QT
	▪ From 64×64 to 8×8 Coding Unit (CU) size	▪ From 128×128 to 4×4 CU size	64×64 to 4×4
		▪ Chroma separate tree	
		▪ Virtual pipeline data units (VPDUs)	×
Intra prediction	▪ DC, planar	▪ DC, planar	×
	▪ 33 directional prediction modes	▪ 65 directional prediction modes	×
		▪ Wide-angle prediction modes	
	▪ Linear interpolation	▪ 4-tap interpolation filters (IFs) using 2 sets of filters	×
		▪ Position-dependent prediction combination (PDPC)	×
		▪ Multiple reference lines (MRL)	
		▪ Matrix-based intra-picture prediction (MIP)	
		▪ Cross-component linear model (CCLM)	
		▪ Intra sub-partitions (ISP)	
Forward/inverse transform and quantization	▪ Square transform (up to 32×32)	▪ Square and rectangular transform (up to 64×64)	32×32 to 4×4
	▪ Discrete cosine and discrete sine transforms	▪ Multiple transform selection (MTS)	×
	▪ DCT2 and DST7	▪ DCT2, DST7, and DCT8	×
		▪ Non-separable secondary transform (LFNST)	
		▪ Adaptive chroma QP offset	×
	▪ Sign data hiding (SDH)	▪ SDH	×
		▪ Dependent quantization (DQ)	
		▪ Joint coding of chroma residuals (JCCR)	×
Entropy coding	▪ Context-adaptive binary arithmetic coding (CABAC)	▪ CABAC with multi-hypothesis probability estimates	×
	▪ Coefficient group	▪ Additional coefficient group size	×
	▪ Reverse diagonal, horizontal and vertical coefficient scan	▪ Reverse diagonal coefficient scan only	×
		▪ Improved probability model sections for absolute transform coefficient levels	×
Loop filtering	▪ Deblocking filter (DF)	▪ Luma mapping with chroma scaling (LMCS)	×
		▪ Deblocking boundary handling modifications	×
		▪ Deblocking long filter	×
		▪ Luma-adaptive deblocking	×
	▪ Sample adaptive offset (SAO)	▪ Sample adaptive offset (SAO)	×
		▪ Adaptive loop filter (ALF)	×
		▪ Cross-Component ALF (CC-ALF)	×
Parallelization	▪ Slices	▪ Slices	×
	▪ Tiles	▪ Tiles	
		▪ Subpictures	
	▪ WPP with CTU row delay of two CTUs	▪ WPP with CTU row delay of one CTU	×

E. Loop Filtering

Sample adaptive offset (*SAO*) has been included in VVC without a change. Deblocking filter has been updated with boundary handling and 4×4 grid. In addition to the deblocking and SAO, two new in-loop filters were added in VVC: *Adaptive Loop Filter* (*ALF*) and *Luma mapping with chroma scaling* (*LMCS*).

ALF is a computationally complex filter applied after Deblock and SAO with 7×7 (luma) or 5×5 (chroma) diamond filter using 25 classifications applied for each 4×4-pixel block in the frame.

LMCS maps the pixel values to spread them across the dynamic range for slight improvement of coding efficiency. The other part of the tool, chroma scaling, is the same basic process done for the chroma to compensate for the mapping. Chroma scaling can be independently disabled.

F. Quantization

The *quantization parameter* (*QP*) range has been extended from 51 to 63. The mapping between luma/chroma and QPs has been changed from a fixed table to a piecewise linear model, signaled in *sequence parameter set* (*SPS*).

III. OPEN-SOURCE VVC IMPLEMENTATIONS

VVC was developed alongside with the VTM [6] reference software. VTM is an open-source software containing both encoder and decoder with all normative coding tools. The focus of VTM is to serve as a common reference implementation. Therefore, it provides the best encoding efficiency but its unoptimized performance is far from real-time.

Fraunhofer HHI has released VTM based VVenC [7] on Sept. 2020 and switched to 3-clause BSD license on Dec. 2020. It improves the VTM encoding performance by defining a set of presets from *slower* to *faster* compression, selecting a set of tools matching the use case, optimizing the coding functions on algorithmic and instruction set level, and including concurrency making it 2× to 140× faster than VTM in RA condition while losing around 0% to 60% in BD-rate, respectively [7].

Finally, there are plans to release x266 [14] as the successor to the popular x265 HEVC encoder [15]. It can be assumed to be based on VTM, as x265 was based on the *HEVC test model* (*HM*) [16].

IV. DESIGN APPROACH

A VVC encoder can be defined as an encoder that is able to produce valid VVC bitstream, i.e., a bitstream decodable by the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCE.2022.3146016, IEEE Transactions on Consumer Electronics

reference decoder with the decoded frames having identical reconstruction in encoder and decoder. Since VVC includes a plurality coding tools which are not strictly necessary depending on the use case, it is important to consider the order of coding tools to implement.

Section IV-A presents the minimum requirements for a VVC encoder to produce a valid bitstream. However, the coding efficiency is severely limited, and any encoder aiming for practical usage should implement significant portion of the available coding tools. Section IV-B describes the criteria for the order of selecting tools for implementation and which tools were selected in this work.

A. Minimum VVC Implementation

The minimum VVC implementation only requires the following tools:

- **Headers:** binary coded header *network abstraction layer* (*NAL*) units for SPS, *Picture Parameter Set*, and the *slice* containing info on the used tools and video parameters.
- **CABAC:** the basic binary coding method, each bit is coded separately, with a context depending on the usage of the bit and each bit updating the context to allow following bits to be code cheaper.
- **Intra prediction:** a prediction based on the bordering pixels, must be done in order to get the residual pixel data. Simplest form is the DC mode containing an average of the bordering pixels and giving the same value prediction over the whole block.
- **Transform:** DCT and DST to transform the residual pixels to the frequency domain. A simple matrix multiplication is used with the precalculated matrix matching the block size and the output is the frequency domain coefficients.
- **Quantization:** quantize the frequency domain coefficients using pre-calculated tables.
- **Data output:** using CABAC, the binarized block data is pushed to the output by iterating over the coding blocks.

B. Selection Criteria for Coding Tools

Overall, there are two main factors for selecting the tools to be implemented: 1) the *rate-distortion-complexity* (*RDC*) performance of the tool and 2) the implementation effort. For 1), a way to estimate the RDC performance is to run VTM with the tool enabled and disabled. Overall, this will give a good estimate on how the tool might perform in practical encoder. 2) is largely influenced by how much the tool interacts with other tools, e.g., ALF is completely separate from rest of the encoding loop, whereas the MTT influences almost all other tools. This means that implementing some of the tools can be done at the same time with different teams whereas others require coordination to avoid overlap.

Table I tabulates the implementation status of the tools. The HEVC based tools were selected because they only needed to be upgraded instead of a brand-new implementation. For the VVC tools the extra intra modes were implemented since they do not require a lot of work after updating the prediction generation. MTS was selected due to its *rate-distortion* (*RD*) performance and JCCR because it largely concerns them same area of the encoder as MTS. ALF was selected due to the RD performance and its separation from the encoding loop. LMCS is another tool selected by being separate from the coding loop, mapping the pixels before intra predictions and back before loop filters while providing small RD improvement with small performance cost.

TABLE II
AVX2 OPTIMIZED PARTS OF KVAZAAR AND THEIR REUSABILITY IN UVG266

Optimized Function		Reuse in uvg266
Intra prediction	DC/Planar	100%
	Angular	5%
SAD and SATD cost calculation		100%
Transform	DCT2	100%
	DST7/DCT8	90%
Transform quantization		95%
Coefficient encoding		50%*
SAO		100%

*Estimate, not implemented yet.

V. PROPOSED IMPLEMENTATION

The design entry point for our work is the award winning Kvazaar HEVC encoder [17] due to its optimized coding flow, threading support, and recently announced permissive 3-clause BSD-license. VVC conversion is applied resulting in a new open-source proof-of-concept VVC encoder, uvg266, which is still able to utilize parts of the optimized coding flow and full threading of Kvazaar.

Most demanding coding tools of Kvazaar are optimized using AVX2. A non-negligible part of these optimizations can be reused for the VVC implementation without changes, as tabulated in Table II. This reduces the development time for optimized solution, even with the new tools. Since intra encoding is needed in all configurations, we start the conversion considering only the intra tools.

Section V-A considers tools that exist in HEVC and how they are updated to conform to VVC. Section V-B cover non-normative tools and Section V-C considers newly added tools that are implemented into uvg266.

A. Normative Coding Tools Updated from HEVC

1) CABAC

Although the structure and bit-pushing to CABAC are similar, changes from HEVC are significant enough that most of the CABAC engine must be rewritten. Context initialization values need to be replaced with the new arrays. The state is stored in two 16-bit variables, replacing the one 8-bit variable in HEVC. Additionally, the state change is now controlled by a *rate* variable defined for each context, which in HEVC used to be a constant global table. CABAC main state does not need to be changed.

2) Block Structure

VVC introduces the 128×128 CTU size, but 64×64 and 32×32 are still options. Since HEVC uses at maximum 64×64 CTUs, only 64×64 CTUs is consider as a first step in this conversation. Additionally, since intra CUs can be at maximum 64×64, the benefit of 128×128 CTUs are limited for intra

encoding. For the *quadtree* (*QT*) structure only the 4×4 CUs require change, since unlike HEVC where the 4×4 blocks are *Prediction Units* (*PUs*) whereas in VVC they are CUs. The main change is that the chroma *transform unit* (*TU*) is coded after all of the corresponding luma TUs instead of the first luma TU.

3) Intra Prediction

DC and planar predictions require no changes and the AVX2 kernels can be used as is. However, the angular AVX2 optimizations have to be rewritten almost completely due to new algorithm in VVC. Updating the MPM selection is a straightforward task to implement. The PDPC is also straightforward to implement since it is essentially just an extra step in the prediction generation.

4) Transform

Since the DST7 is only used for 4×4 luma CUs in HEVC, the 4×4 DCT2 also exists in HEVC, thus updating to VVC consists in setting the 4×4 luma transform to DCT2. The 64×64 transforms are optional, and we are yet to implement them.

5) Loop Filtering

Since SAO is identical to HEVC, it does not require any changes. For deblocking, the 4×4 grid requires some changes since Kvazaar performs deblocking on CTU basis. The right and bottom edges of the CTU are deblocked when the following CTUs are processed to fully utilize the *wavefront parallel processing* (*WPP*). Because the filtering length for larger blocks exceeds four pixels in VVC, the CTU edge and the last inner 4×4 CU edges must be deblocked by the following CTUs.

6) Quantization

Increasing the maximum QP value is effortless, since the underlying formula was not changed. Similarly, the implementation of the chroma QP scaling only require implementing the piecewise linear model. The quantization is optimized with AVX2 kernels in Kvazaar, and they can be included in uvg266 with minor changes.

B. Non-Normative Coding Tools Updated from HEVC

1) Intra Search

With the angular modes increasing from 33 to 65, the search itself needed only small changes. For the angular intra modes, the first version omitted the extra angular modes added in VVC, where the HEVC modes had to be mapped to correct VVC modes in the bitstream writing. Supporting all the intra angular modes needs minor changes in the search loops. Costs are calculated the same way as in HEVC, so the AVX2-optimized kernel can be used as is.

2) Rate-Distortion Optimized Quantization (RDOQ)

Since the CABAC contexts and bit coding have changed, the RDOQ must be adapted to the VVC by addressing the changed binary output in all the places where bit costs are calculated. This includes changing the CABAC contexts and adding new costs for bits that did not exist in HEVC.

C. New VVC Coding Tools

1) Adaptive Loop Filter (ALF)

Although ALF requires more lines of code than most other tools, it is completely separated from rest of the encoding loop, thus the implementation is straightforward. ALF filtering is done for the whole frame and requires analysis data from the complete frame, which prevents using WPP efficiently. Therefore, the final bitstream generation has to be moved after the whole frame is processed. To allow use of WPP and keep bit estimation accurate for the *rate distortion optimization* (*RDO*) process, the bitstream generation is simulated during the search process.

TABLE III
TEST SEQUENCES

Class	Format	Sequence	Frame count	Frame rate
A1	3840×2160 (2160p)	Campfire	300	30 fps
		FoodMarket4	300	60 fps
	4096×2160	Tango2	294	60 fps
A2	3840×2160 (2160p)	CatRobot	300	60 fps
		DaylightRoad2	300	60 fps
		ParkRunning3	300	50 fps
B	1920×1080 (1080p)	BasketballDrive	500	50 fps
		BQTerrace	600	60 fps
		Cactus	500	50 fps
		MarketPlace	600	60 fps
		RitualDance	600	60 fps
C	832×480 (480p)	BasketballDrill	500	50 fps
		BQMall	600	60 fps
		PartyScene	500	50 fps
		RaceHorses	300	30 fps
D	416×240 (240p)	BasketballPass	500	50 fps
		BlowingBubbles	500	50 fps
		BQSquare	600	60 fps
		RaceHorses	300	30 fps
E	1280×720 (720p)	FourPeople	600	60 fps
		Johnny	600	60 fps
		KristenAndSara	600	60 fps
UVG Dataset	3840×2160 (2160p)	Beauty	600	120 fps
		Bosphorus	600	120 fps
		HoneyBee	600	120 fps
		Jockey	600	120 fps
		Lips	600	120 fps
		ReadySetGo	600	120 fps
		ShakeNDry	300	120 fps
		YachtRide	600	120 fps

2) Luma Mapping with Chroma Scaling (LMCS)

Unlike other in-loop filters, LMCS directly influences the IP, thus implementing it is more challenging than the other in-loop filters. However, the implementation of Luma Mapping that is the first part of the tool is straightforward for AI condition. In intra coding, the pixels are always in mapped domain after the parameters are constructed at the start of the frame and pixels mapped, and only right before LF stage they are reverse mapped. Chroma Scaling, which is the second part, is applied to pixels before TR/Q stage and they are scaled back after IQ/IT stage.

3) Multitype Transform Selection (MTS)

Kvazaar features AVX2 optimized kernels for the DCT2 and 4×4 DST7 transforms. Fortunately, the kernels are designed in such fashion that providing new coefficients allows using the kernels for all of the MTS transforms. The MTS search is implemented to replace the transform depth search since it is removed in VVC. Finally, moving the transform skip coding to separate context requires some work but overall, it is pretty straightforward, since it is mostly similar to normal transform coefficient coding.

TABLE IV
PROFILING PLATFORM FOR COMPLEXITY ANALYSIS

Processor	General Purpose Processor 18-core (4.20GHz)
Cache	24.75 MB
SSD	1TB NVMe PCIe Gen 3
RAM	64GB DDR4 3200 MHz
Compiler	GCC 7.2.0
Kernel version	64-bit Linux 4.13.0

TABLE V
ENCODING PARAMETERS

Encoder	Options
Kvazaar	--preset veryslow -p1 --wpp --threads=<Threads>
uvg266	--preset veryslow -p1 --wpp --owf=<Threads> --mts=intra --alf=full --threads=<Threads>
HM	-c encoder_intra_main.cfg --TemporalSubsampleRatio=8
VVenC	-c randomaccess_slower.cfg --IntraPeriod=1 --GOPSize=1 --TemporalSubsampleRatio=8 --WaveFrontSynchro=1 --WppBitEqual=1 -t <Threads> --MaxParallelFrames=1 --AMaxBT=0
VTM	-c encoder_intra_vtm.cfg

4) Joint Coding of Chroma Residuals (JCCR)

Although Kvazaar is designed to handle all the channels separately, adding the consideration for JCCR is straightforward since only the generation of the combined residual needs to be added and the RD-cost calculation can be mostly facilitated by existing functions.

VI. EXPERIMENTAL SETUP

All our experiments were automated using uvgVenctester [18].

A. Encoder and Coding Conditions

To evaluate the implementation presented in this work, the proposed uvg266 encoder was first compared with Kvazaar 2.0 (git hash 54dc87d) [10] that was the latest available release during our experiments. In addition, uvg266 was benchmarked against HM 16.23 [16]. In the second step, the performance of uvg266 was evaluated against the existing VVC encoding solutions VTM 13.2 [6] and VVenC 1.0.0 [19].

The experiments were carried out under the VVC CTC with the exception of all 10-bit sequences converted to 8-bit because Kvazaar and uvg266 do not support 10-bit internal bit depth. The VTM and VVenC encoders were configured to their default internal 10-bit mode with 8-bit inputs whereas HM, Kvazaar and uvg266 use 8-bit internal. At this early stage of development, uvg266 only supports the internal bit depth of 8 bits but will be extended to support higher bit depths in a forthcoming release.

Encoders adopted the AI condition with the base QP values of 22, 27, 32, and 37 from the VVC CTC [9]. Under the AI condition, all frames were encoded as I-frames in display order without any QP offsets. For the AI condition, the VVC CTC [9] requires only every eighth frame to be encoded. For a fair comparison, the same temporal subsampling ratio was used with HM, Kvazaar, uvg266, and VVenC.

B. Test Sequences

Table III details our test set that features a broad range of sequence parameters (spatial resolution, frame rate, and bit depth). It includes all 22 natural full-length 8-bit and 10-bit YUV420 test sequences defined in the VVC CTC (*classes A–E*) [9]. In addition, it was extended with eight 2160p 120fps sequences from our UVG dataset [20].

TABLE VI
VVC BASELINE IMPLEMENTATION OF UVG266 COMPARED WITH SINGLE-THREADED KVAZAAR

Class	BD-rate			Speedup		
	PSNR	SSIM	VMAF	1 thr	8 thr	36 thr
HEVC constrained uvg266						
A1	0.08%	-0.13%	-3.59%	0.43×	3.39×	8.51×
A2	-0.79%	-0.62%	-5.01%	0.44×	3.46×	8.74×
B	-0.90%	-0.17%	-3.08%	0.43×	3.43×	8.70×
C	0.02%	1.59%	-3.09%	0.43×	3.43×	8.51×
D	0.82%	2.10%	-2.44%	0.43×	3.34×	8.04×
E	0.74%	2.19%	-1.75%	0.43×	3.40×	8.52×
UVG	-1.11%	-0.99%	-2.58%	0.43×	3.41×	8.59×
Avg.	**-0.33%**	**0.35%**	**-2.97%**	**0.43×**	**3.41×**	**8.53×**
uvg266 VVC baseline						
A1	-2.28%	-2.51%	-6.82%	0.40×	3.13×	7.83×
A2	-2.59%	-2.45%	-7.23%	0.40×	3.19×	8.10×
B	-2.48%	-1.83%	-5.28%	0.40×	3.18×	8.07×
C	-1.89%	-0.53%	-5.45%	0.41×	3.22×	8.02×
D	-1.17%	-0.08%	-5.20%	0.41×	3.13×	7.52×
E	-2.51%	-1.21%	-5.12%	0.41×	3.17×	7.95×
UVG	-2.04%	-1.89%	-4.09%	0.40×	3.12×	7.89×
Avg.	**-2.10%**	**-1.51%**	**-5.31%**	**0.40×**	**3.16×**	**7.91×**

C. Quality Metrics

The RD performances are reported in terms of *BD-rates* [21], [22], measured with three different objective quality metrics: *peak signal-to-noise ratio* (*PSNR*), *Structural SIMilarity* (*SSIM*) [23], and *Video Multimethod Assessment Fusion* (*VMAF*) [24]. Unlike PSNR and SSIM, VMAF is primarily intended for measuring quality of videos. Because of the temporal subsampling, the VMAF results presented in this paper should not be compared with VMAF results obtained without temporal subsampling.

D. Complexity Profiling Setup

Table IV details the profiling platform used for the testing. HM, VTM, and Kvazaar were benchmarked only with single threaded configuration whereas VVenC and uvg266 were additionally tested with 8 and 36 threads. Table V tabulates the parameters used for each encoder in addition to input, resolution, and QP. The CPU used for the testing has 18 cores with hyperthreading, but 36-thread results are not expected to scale much over the physical core count.

VII. PERFORMANCE EVALUATION

The performance evaluation is carried out in the following four steps.

1) The VVC baseline implementation is compared with the original Kvazaar HEVC encoder.
2) The impact of each added coding tool is independently evaluated in relation to the VVC baseline implementation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCE.2022.3146016, IEEE Transactions on Consumer Electronics

TABLE VII
UVG266 IMPLEMENTED TOOLS COMPARED WITH SINGLE-THREADED UVG266 VVC BASELINE

Class	BD-rate			Speedup			BD-rate			Speedup		
	PSNR	SSIM	VMAF	1 thr	8 thr	36 thr	PSNR	SSIM	VMAF	1 thr	8 thr	36 thr
	ALF						Implicit MTS					
A1	-3.12%	-3.51%	0.44%	0.55×	4.14×	9.63×	-1.38%	-1.77%	0.43%	0.98×	7.72×	19.44×
A2	-1.25%	-1.40%	-1.03%	0.63×	4.74×	10.71×	-1.03%	-1.28%	1.08%	0.99×	7.82×	19.90×
B	-3.41%	-4.34%	-2.73%	0.59×	4.61×	11.57×	-0.90%	-1.35%	1.13%	0.99×	7.82×	19.90×
C	-1.51%	-2.34%	-1.51%	0.50×	3.87×	9.51×	-0.66%	-0.65%	3.31%	0.99×	7.76×	19.43×
D	-0.60%	-0.51%	-0.19%	0.30×	2.32×	5.57×	-0.42%	-0.51%	3.56%	0.99×	7.56×	18.37×
E	-3.72%	-5.05%	-2.31%	0.44×	3.41×	8.53×	-2.24%	-2.07%	1.19%	0.99×	7.76×	19.56×
UVG	-3.70%	-4.52%	1.85%	0.57×	4.43×	10.68×	-0.60%	-1.06%	1.46%	0.98×	7.73×	19.59×
Avg.	**-2.64%**	**-3.31%**	**-0.48%**	**0.52×**	**4.00×**	**9.67×**	**-0.92%**	**-1.17%**	**1.76%**	**0.98×**	**7.74×**	**19.47×**
	Explicit MTS						LMCS					
A1	-1.93%	-1.78%	-1.67%	0.37×	2.93×	7.33×	0.52%	1.46%	-2.89%	0.97×	7.79×	18.09×
A2	-1.73%	-0.95%	-2.10%	0.37×	2.98×	7.48×	0.47%	-0.20%	2.90%	0.99×	7.91×	19.37×
B	-0.99%	-0.28%	-1.53%	0.35×	2.85×	7.17×	-0.12%	-0.13%	0.63%	0.96×	7.69×	19.12×
C	-0.92%	0.25%	-1.86%	0.35×	2.84×	7.04×	0.51%	0.62%	-1.15%	0.94×	7.41×	18.49×
D	-0.93%	0.59%	-1.96%	0.36×	2.77×	6.76×	0.36%	0.30%	0.14%	0.95×	7.17×	17.71×
E	-2.43%	-1.19%	-2.26%	0.35×	2.76×	6.95×	0.93%	0.99%	0.11%	0.92×	7.30×	17.44×
UVG	-0.87%	0.18%	-0.25%	0.37×	2.93×	7.36×	1.68%	2.05%	0.86%	0.95×	7.62×	17.89×
Avg.	**-1.25%**	**-0.28%**	**-1.43%**	**0.36×**	**2.87×**	**7.17×**	**0.73%**	**0.87%**	**0.21%**	**0.95×**	**7.56×**	**18.27×**
	JCCR											
A1	0.11%	0.11%	0.25%	0.91×	7.17×	18.18×						
A2	0.01%	-0.04%	0.00%	0.92×	7.24×	18.48×						
B	0.09%	0.07%	0.20%	0.93×	7.41×	18.87×						
C	0.10%	0.14%	0.20%	0.94×	7.37×	18.36×						
D	0.13%	0.15%	0.02%	0.94×	7.22×	17.53×						
E	0.19%	0.19%	0.24%	0.93×	7.30×	18.46×						
UVG	0.08%	0.04%	0.08%	0.92×	7.27×	18.52×						
Avg.	**0.10%**	**0.09%**	**0.13%**	**0.93×**	**7.29×**	**18.38×**						

3) The most efficient coding tool set is evaluated against the original HEVC encoder.
4) Finally, the proposed solution is compared with the existing VVC encoding solutions.

The result tables presented in this section include the averages for each class and across all sequences. Negative BD-rate values imply better coding efficiency and speedups higher than 1.0× faster coding speed than the anchor.

A. VVC Baseline Implementation

The VVC baseline implementation of uvg266 is evaluated with two configurations: *HEVC constrained uvg266* and *uvg266 VVC baseline*. The former aims to use limited set of encoding tools of HEVC but encoding a compliant VVC bitstream. The angular modes are limited to the 33 found in HEVC in this configuration. The second configuration does not have this limitation and uses the full 65 angular modes in addition to DC and Planar.

Table VI presents the results of the two previous configurations compared with single-threaded Kvazaar used as an anchor. The *HEVC constrained uvg266* results show that the current implementation is around half the speed of Kvazaar while providing a similar BD-rate performance. VMAF is an exception showing 3% improved BD-rate. Using all angular modes improves in BD-rate the coding efficiency of 1.8%, 1.8%, and 2.4% in PSNR, SSIM, and VMAF, respectively, with around 7% reduction in coding speed. The results only include a comparison to single threaded Kvazaar, since uvg266 fully inherits the scaling capabilities of Kvazaar, as shown by these results and the performance of Kvazaar [25].

B. New VVC Tools

Table VII tabulates the performance results of each individual encoding tool implemented in uvg266 compared with the *uvg266 VVC baseline* configuration used as an anchor.

TABLE VIII
UVG266 WITH ALF AND MTS COMPARED WITH SINGLE-THREADED KVAZAAR

Class	BD-rate			Speedup		
	PSNR	SSIM	VMAF	1 thr	8 thr	36 thr
A1	-6.98%	-7.59%	-8.23%	0.11×	0.86×	2.05×
A2	-5.40%	-4.83%	-9.89%	0.12×	0.96×	2.25×
B	-6.63%	-6.48%	-9.44%	0.12×	0.91×	2.28×
C	-4.19%	-2.69%	-8.67%	0.11×	0.85×	2.09×
D	-2.57%	0.02%	-7.66%	0.08×	0.61×	1.48×
E	-8.13%	-7.27%	-9.27%	0.10×	0.76×	1.92×
UVG	-6.17%	-5.90%	-3.73%	0.11×	0.90×	2.20×
Avg.	**-5.70%**	**-4.98%**	**-7.48%**	**0.11×**	**0.84×**	**2.07×**

TABLE IX
UVG266 WITH ALF AND MTS COMPARED WITH SINGLE-THREADED HM

Class	BD-rate			Speedup		
	PSNR	SSIM	VMAF	1 thr	8 thr	36 thr
A1	-6.08%	-6.63%	-10.39%	0.60×	4.66×	11.08×
A2	-3.79%	-3.40%	-13.78%	0.52×	4.04×	9.47×
B	-5.20%	-5.10%	-10.12%	0.48×	3.78×	9.48×
C	-2.64%	-0.63%	-8.35%	0.36×	2.86×	7.06×
D	-1.74%	0.81%	-7.88%	0.24×	1.86×	4.48×
E	-6.55%	-5.64%	-8.90%	0.52×	4.00×	10.06×
UVG	-5.49%	-5.43%	-5.98%	0.58×	4.59×	11.27×
Avg.	**-4.56%**	**-3.84%**	**-8.75%**	**0.48×**	**3.75×**	**9.18×**

TABLE X
UVG266 WITH MTS AND ALF COMPARED WITH VTM AND VVENC

Class	BD-rate			Speedup		
	PSNR	SSIM	VMAF	1 thr	8 thr	36 thr
Single-threaded VTM anchor						
A1	32.93%	41.32%	36.97%	12.50×	96.97×	230.24×
A2	28.54%	33.85%	24.96%	14.80×	115.94×	272.33×
B	22.22%	26.45%	20.76%	13.98×	110.42×	277.25×
C	25.81%	31.15%	21.08%	13.48×	107.28×	264.67×
D	19.02%	24.04%	16.74%	9.88×	77.01×	185.80×
E	25.27%	29.56%	25.80%	12.61×	96.27×	242.11×
UVG	23.68%	26.23%	27.92%	14.43×	113.96×	279.54×
Avg.	**24.67%**	**29.23%**	**24.72%**	**13.10×**	**102.55×**	**255.28×**
Multi-threaded VVenC anchor						
A1	32.25%	39.03%	29.14%	9.83×	12.46×	25.88×
A2	27.74%	31.82%	22.50%	12.04×	14.44×	28.42×
B	22.43%	27.36%	21.32%	11.79×	20.47×	50.02×
C	25.60%	31.76%	21.36%	12.43×	38.61×	93.01×
D	18.92%	25.37%	18.97%	8.71×	45.56×	106.51×
E	25.36%	30.50%	25.64%	10.86×	28.02×	69.30×
UVG	25.57%	28.25%	26.97%	10.46×	12.50×	23.83×
Avg.	**25.03%**	**29.85%**	**23.85%**	**10.59×**	**24.58×**	**53.65×**

ALF doubles the encoding time for 2.6%, 3.3%, and 0.5% BD-rate improvement with PSNR, SSIM, and VMAF, respectively. Additionally, since the current implementation of ALF needs to perform search for the full frame after initial reconstruction, which requires increasing the number of parallel frames, which add memory overhead of almost 1MB/CTU or around 1.5GB for a single 2160p frame. Similarly, the concurrency scaling decreases slightly because of the increased number of parallel frames.

Results show that implicit MTS has negligible effect to the encoding speed, and it improves BD-rate by 1.0% and 1.2% when measured with PSNR and SSIM but degrades it by 1.6% with VMAF. Conversely, explicit MTS improves BD-rate with all metrics, but it also cuts the encoding speed to nearly a third without affecting the scaling.

LMCS degrades BD-rate with PSNR and SSIM because it is not yet RD optimized. Nevertheless, the encoding complexity increase is only around 5% and the VMAF BD-rate improvement in *Class A1* shows that there is potential in the tool once it is RD optimized. However, LMCS also introduces a small bottleneck due to the mapping, which slightly decreases the thread scaling.

Like LMCS, JCCR is not yet RD optimized and degrades BD-rates by about 0.1% with around 7% increase in complexity and no decrease in scaling.

C. Proposed uvg266 Configuration

From the experiments of the previous section, the best encoding efficiency is reached when uvg266 enables ALF and explicit MTS tools. Table VIII tabulates the comparison between this proposed configuration and single-threaded Kvazaar used as an anchor. The results show that uvg266 improves the PSNR, SSIM, and VMAF BD-rates by 5.7%, 5.0%, and 7.5%, respectively. On the other hand, the encoding speed is reduced to about tenth. However, since the newly added tools are yet to be optimized, there is much room to improve the complexity of uvg266, e.g., the complexity of the new ALF filter can be significantly reduced with an optimized implementation.

TABLE XI
UVG266, VVENC, AND VTM ENCODING SPEED IN FPS

Class	uvg266			VVenC			VTM
	1 thr	8 thr	36 thr	1 thr	8 thr	36 thr	1 thr
A1	0.018	0.140	0.332	0.002	0.013	0.015	0.002
A2	0.014	0.110	0.258	0.001	0.008	0.009	0.001
B	0.051	0.407	1.021	0.004	0.020	0.021	0.004
C	0.170	1.353	3.338	0.014	0.035	0.036	0.013
D	0.425	3.308	7.974	0.049	0.074	0.076	0.043
E	0.146	1.113	2.802	0.014	0.041	0.042	0.012
UVG	0.017	0.137	0.337	0.002	0.012	0.015	0.001

Additionally, uvg266 is evaluated against HM and the results are shown in Table IX; HM being used as an anchor. These results show that uvg266 outperform HM even with fairly limited number of the VVC tools, especially when measured with VMAF. Moreover, uvg266 is up to 10× faster than HM using multi-threading.

D. Comparison with Existing VVC Encoders

Table X compares the proposed configuration of uvg266 with single-threaded VTM and multi-threaded VVenC used as an anchor. Moreover, Table XI tabulates the absolute encoding speed in *frames per second* (*FPS*) for each encoder and threading.

Since uvg266 does not yet implement all of the existing VVC tools, it is expected that both VTM and VVenC outperform it. In fact, they both have around 25%, 30%, and 24% better BD-rate measured with PSNR, SSIM, and VMAF, respectively. Nevertheless, uvg266 is over ten times faster with single-threaded operation and parallelizes better than VVenC. Based on these results, it is interesting to notice that developing a new VVC encoder based on a practical HEVC encoder allows to produce a highly parallel solution even in the early stages of development.

VIII. FUTURE WORK

In the future, we plan to continue developing uvg266 toward a complete VVC encoding solution that achieves real-time coding performance. This development can be roughly split into following five stages: 1) implementing the rest of the intra encoding tools, 2) implementing the inter tools, 3) implementing the MTT structure, 4) vectorization of new tools, and 5) algorithmic optimization and complexity reduction, e.g., using machine learning. 1) is currently underway and we have started exploring the inter tools to start 2). Furthermore, we have started formulated an initial plan for 3). There are two main points to adding the support for MTT: dealing with non-square CUs and managing the recursive splitting. The first step would be to implement a single layer of MTT splitting to deal with the non-square CUs, which can be then extended for the full MTT support. Since the implementation is still in an early stage, 4) and 5) are not yet a high priority but we will start from tools such as ALF and MTS that have a high complexity overhead. Finally, the development of 3) could be directly combine with 5) to use machine learning for predicting the structure, as was done with Kvazaar in [25].

Additionally, there is opportunities for RD optimization, especially with LMCS and JCCR, but most other tools should also have options.

IX. Conclusion

This paper introduced the conversion process of the practical HEVC encoder Kvazaar into the practical VVC intra encoder called uvg266, which is distributed under the permissive 3-clause BSD license. The purpose of this paper is to show that converting an existing HEVC encoder to VVC standard is a valid approach and allows to reuse part of the coding structure, optimization, and parallelism of the HEVC implementation. To conform to the VVC standard, coding tools can be categorized according to their implementation: 1) the HEVC tools that need to be entirely rewritten, such as CABAC; 2) the HEVC tools that can be upgraded with limited effort; and 3) the new VVC tools that need to be implemented from scratch.

To conclude, our success with the software implementation shows promise for more generic reuse of the encoder structure when moving between standards. Moreover, implementing an encoder in hardware with either hardware description language or high-level synthesis can also benefit from this approach by reusing encoder blocks similarly to a software encoder. Overall, this work aims to close the gap between implementation of HEVC and VVC encoding standard and accelerate the deployment of real-time VVC solutions on consumer electronics.

References

[1] Cisco, Cisco Visual Networking Index: Forecast and Trends, 2017-2022, Dec. 2018.

[2] ITU, "New 'Versatile Video Coding' standard to enable next-generation video compression," Sept. 2020, [Online]. Available: https://www.itu.int/en/mediacentre/Pages/pr13-2020-New-Versatile-Video-coding-standard-video-compression.aspx.

[3] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, Dec. 2012, pp. 1649–1668.

[4] A. Mercat, A. Mäkinen, J. Sainio, A. Lemmetti, M. Viitanen, and J. Vanne, "Comparative rate-distortion-complexity analysis of VVC and HEVC video codecs," IEEE Access, vol. 9, 2021, pp. 67813–67828.

[5] G. Sullivan, "Deployment status of the HEVC standard," document JVET-V0020, Apr. 2021.

[6] "VVC Reference Software Version 13.2," [Online]. Available: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/tree/VTM-13.2.

[7] A. Wieckowski, J. Brandenburg, T. Hinz, C. Bartnik, V. George, G. Hege, C. Helmrich, A. Henkel, C. Lehmann, C. Stoffers, I. Zupancic, B. Bross, and D. Marpe, "Vvenc: An Open And Optimized Vvc Encoder Implementation," in Proc. IEEE Int. Conf. Multimedia Expo Workshops, Shenzhen, China, Jul. 2021.

[8] J. Brandenburg, A. Wieckowski, T. Hinz, and B. Bross, "VVenC Fraunhofer Versatile Video Encoder," 2020, [Online]. Available: https://www.digitalmedia.fraunhofer.de/content/dam/dcinema/en/documents/ibc2020/VVenC_Versatile-Video-Encoder_Paper.pdf.

[9] F. Bossen, J. Boyce, K. Suehring, X. Li, and V. Seregin, "VTM common test conditions and software reference configurations for SDR video," document JVET-T2010, Teleconference, Oct. 2020.

[10] Ultra Video Group, "Kvazaar open-source HEVC encoder," [Online]. Available: https://github.com/ultravideo/kvazaar.

[11] P. Sjövall, A. Lemmetti, J. Vanne, S. Lahti, and T. D. Hämäläinen, "High-level synthesis implementation of an embedded real-time HEVC intra encoder on FPGA for media applications," Transactions on Design Automation of Electronic Systems, to be published.

[12] J. Chen, Y. Ye, and S. Kim, "Algorithm description for versatile video coding and test model 10 (VTM 10)," document JVET-S2002, Teleconference, Jul. 2020.

[13] J. Chen, Y. Ye, and S. Kim, "Versatile video coding editorial refinements on draft 10," document JVET-T2001, Teleconference, Oct. 2020.

[14] MulticoreWare, Inc., "Leading next-gen video technologies with development of open source x266 (VVC) encoding and x266 consortium," [Online]. Available: https://multicorewareinc.com/video/#x266.

[15] MulticoreWare, Inc., "x265 HEVC encoder / H.265 video codec," [Online]. Available: https://bitbucket.org/multicoreware/x265/downloads.

[16] "HEVC Reference Software Version 16.23," [Online]. Available: https://vcgit.hhi.fraunhofer.de/jct-vc/HM/-/tags/HM-16.23.

[17] A. Lemmetti, M. Viitanen, A. Mercat, and J. Vanne, "Kvazaar 2.0: fast and efficient open-source HEVC inter encoder," in Proc. ACM Multimedia Syst. Conf., Istanbul, Turkey, Jun. 2020.

[18] J. Sainio, A. Mercat, and J. Vanne, "uvgVenctester: open-source test automation framework for comprehensive video encoder benchmarking," in Proc. ACM Multimedia Syst. Conf., Istanbul, Turkey, Jun. 2021.

[19] "Fraunhofer Versatile Video Encoder (VVenC) v1.0.0," [Online]. Available: https://github.com/fraunhoferhhi/vvenc/tree/v1.0.0.

[20] A. Mercat, M. Viitanen, and J. Vanne, "UVG dataset: 50/120fps 4K sequences for video codec analysis and development," in Proc. ACM Multimedia Syst. Conf., Istanbul, Turkey, May 2020, pp. 297–302.

[21] G. Bjøntegaard, "Improvements of the BD-PSNR model," document VCEG-AI11, Berlin, Germany, Jul. 2008.

[22] ITU-T and ISO/IEC JTC 1, "Working practices using objective metrics for evaluation of video coding efficiency experiments," document ITU-T HSTP-VID-WPOM and ISO/IEC DTR 23002-8, 2020.

[23] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," IEEE Trans. Image Process., vol. 13, no. 4, Apr. 2004, pp. 600–612.

[24] Z. Li, A. Aaron, I. Katsavounidis, A. Moorthy, and M. Manohara, "Toward a practical perceptual video quality metric," Jun. 2016, [Online]. Available: http://techblog.netflix.com/2016/06/toward-practical-perceptual-video.html.

[25] A. Mercat, A. Lemmetti, M. Viitanen, and J. Vanne, "Acceleration of Kvazaar HEVC intra encoder with machine learning," in Proc. IEEE Int. Conf. Image Processing, Taipei, Taiwan, Sept. 2019.

Marko Viitanen (Member, IEEE) received the M.Sc. degree in information technology from the Tampere University of Technology, Tampere, Finland in 2017. He is currently working towards his Ph.D. degree at Tampere University (TAU), where he is working as a Doctoral Researcher as a part of Ultra Video Group (UVG).

His research activities include HEVC/VVC video coding, 360/VR video capturing and compression, as well as customized transmission systems.

Joose Sainio (Member, IEEE) received the M.Sc. degree in information technology from the Tampere University of Technology, Tampere, Finland, in 2018. He is currently pursuing the Ph.D. degree with UVG.

He has been a part of UVG, since 2016. His research interests include HEVC/VVC video coding, in particular enabling real-time encoding. He has experience in both hardware acceleration and more traditional optimization methods. Additionally, he has some familiarity with perceptual video coding and rate control.

Alexandre Mercat (Member, IEEE) received the M.Sc. and Ph.D. degrees in electrical and computer engineering from the Institut National des Sciences Appliquées (INSA) of Rennes, Rennes, France, in 2015 and 2018, respectively.

He has been a Postdoctoral Researcher with Computing Sciences, Tampere University (TAU), Tampere, Finland, since 2018. His research interests include implementation of image and signal processing applications in many-core embedded systems, real-time implementations of the new generation video coding standards, complexity-aware video coding, machine learning, approximate computing, power consumption, and digital systems design.

Ari Lemmetti (Member, IEEE) received the M.Sc. degree in information technology from Tampere University, Tampere, Finland, in 2021.

He is a Researcher at Tampere University, Tampere, Finland, and a member of Ultra Video Group since 2014. His research interests include HEVC and VVC video compression, rate-distortion-complexity optimization, and parallel computing.

Jarno Vanne (Member, IEEE) received the M.Sc. degree in information technology and the Ph.D. degree in computing and electrical engineering from the Tampere University of Technology (TUT), Tampere, Finland, in 2002 and 2011, respectively.

He is currently an Associate Professor with the Unit of Computing Sciences, Tampere University, Tampere. He is also the founder and a leader of the Ultra Video Group that is also the leading academic video coding group in Finland. He has been the project manager for 22 international/national research projects. He is the author of over 80 peer-reviewed scientific publications. His research interests include HEVC/VVC video coding, ML-powered video coding, immersive 3D/360 media processing for extended reality (XR), volumetric video capture and coding, vision-based environment perception in autonomous vehicles and drones, hybrid human–machine vision, remote machine control over 5G, telepresence, hardware accelerated video coding, video annotation, and virtual traffic simulation environments.