VVC Complexity and Software Implementation Analysis

Frank Bossen, *Senior Member, IEEE*, Karsten Sühring, Adam Wieckowski, and Shan Liu, *Senior Member, IEEE*

(Invited Paper)

***Abstract*— A steady increase in available processing power continues to drive advances in video compression technology. The recently completed Versatile Video Coding (VVC) standard aims to double the compression efficiency of HEVC and deliver a same quality of video at half the bitrate. To achieve this goal, VVC includes several new methods that improve coding efficiency at the cost of increased complexity. This paper provides a complexity analysis of VVC and its VTM reference software. Whereas VVC is more complex than HEVC, it remains readily implementable in software on current generation processors. Performance of practical decoders are reported, showing that real-time decoding of 8K content is feasible. An encoder is also presented, showing that most of the compression gains of VVC over HEVC can be obtained at a small fraction of the resources needed by the VTM encoder under common test conditions.**

***Index Terms*— Video coding, Versatile video coding (VVC).**

I. Introduction

VERSATILE Video Coding (VVC) [1] is the most recent video compression specification jointly developed by ISO/IEC and ITU-T. One of its purposes is to provide a substantial improvement in compression efficiency over the previous HEVC standard: deliver a same quality of video at half the bitrate. It features several profiles. This paper focuses on the core "Main 10" profile, which supports sample bit depths of up to 10 bits and the 4:2:0 chroma sampling format. Some aspects of the "Main 10 4:4:4" profile are also discussed. It should be noted that VVC, unlike its predecessors, does not feature a separate profile for lower sample bit depths (e.g., 8 bits). VVC's improved compression efficiency comes at the cost of increased complexity. Complexity may come in various forms, including computational complexity, memory requirements (local and global), and memory bandwidth.

This paper aims to describe and quantify this complexity increase. It follows a similar structure to that of its sibling on HEVC [2]. Section II discusses design aspects of VVC and their impact on complexity. Section III provides an analysis of the VTM reference software. Section IV describes optimized VVC software decoders. Section V describes an optimized VVC software encoder. Finally, conclusions are drawn in Section VI.

II. Design Aspects

The following sections review complexity aspects of the different modules of the VVC standard. Emphases are put on comparing VVC to HEVC, and on software implementations. In the following, complexity is often expressed in multiply-accumulate (MAC) operations per sample. Such a MAC operation may typically multiply two 16-bit input values and accumulate the result in a 32-bit register. As a point of reference, a single processor core may compute in the order of 24 such MAC operations per cycle.[1] When considering a processor with 8 cores clocked at 3GHz and 4K-UHD video at 60 fps, this translates to about 750 MACs per sample. Most algorithms do not solely use MAC operations, and it should therefore not be assumed that a fixed budget of 750 MACs per sample is available. Where other types of operations are dominant, operation counts per sample are provided and each MAC is counted as two operations. Operations that rearrange data without modifying it (e.g., block transpose) are generally ignored here, even though they add overhead in practice.

A. Block Partitioning

VVC supports more flexible block partitioning than HEVC [3]. Indeed, binary, ternary, or quaternary splits may be recursively applied to partition a coding tree unit (CTU) into coding units (CU). The binary and ternary splits may be along either horizontal or vertical directions. A ternary vertical split decomposes an N × M unit into units of size N/4 × M, N/2 × M, and N/4 × M. It should be noted that the middle partition has width N/2 but may not be aligned to the N/2 × N/2 grid, which may lead to unaligned memory access requests. The quaternary split corresponds to the quadtree split also present in HEVC. Fig. 1 provides an example of a picture partitioned into CUs.

Manuscript received September 11, 2020; revised February 11, 2021; accepted March 25, 2021. Date of publication April 9, 2021; date of current version October 4, 2021. This article was recommended by Associate Editor J. Chen. *(Corresponding author: Karsten Sühring.)*

Frank Bossen is with Sharp Electronics of Canada Ltd., Mississauga, ON L4Z 1W9, Canada (e-mail: fbossen@sharpsec.com).

Karsten Sühring and Adam Wieckowski are with the Video Coding and Analytics Department, Fraunhofer Institute for Telecommunications–Heinrich Hertz Institute, 10587 Berlin, Germany (e-mail: karsten.suehring@hhi.fraunhofer.de).

Shan Liu is with Tencent America, Palo Alto, CA 94306 USA (e-mail: shanl@tencent.com).

Digital Object Identifier 10.1109/TCSVT.2021.3072204

[1] Assuming 3 execution ports, each 256 bits wide, and 2 instructions to execute a MAC: a 16 × 16-bit multiplication and a 32-bit add (e.g., vpmaddwd and vpadd on x86-64). 48 MACs can be computed every 2 cycles.

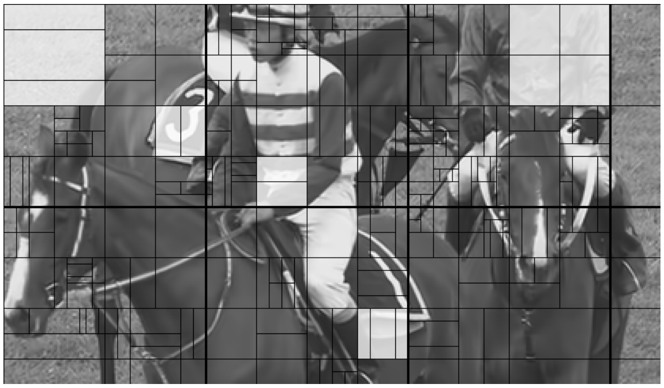

Fig. 1. Example of a picture partitioned into CUs, where VVC's five split patterns are highlighted (quaternary, horizontal binary, vertical binary, horizontal ternary, and vertical ternary). Thick lines represent CTU boundaries.

A CU generally comprises spatially collocated luma and chroma coding blocks (CB). However, a CU may comprise only luma or chroma CBs when luma and chroma coding trees diverge in intra-coded regions, as in the following cases: (a) an intra slice uses separate trees; or (b) use of local separate trees is triggered to satisfy minimum size constraints. To avoid processing small blocks the following constraints apply to CBs when using the 4:2:0 chroma sampling format: the minimum width is 4 samples for luma CBs and intra-coded chroma CBs, and 2 samples for inter-coded chroma CBs; the minimum height is 4 samples for luma CBs and 2 samples for chroma CBs; and the minimum area is 16 samples for luma CBs and intra-coded chroma CBs, and 8 samples for inter-coded chroma CBs.

It should be noted that VVC does not use the concept of prediction units (PU) as in HEVC. The size of a predicted block generally matches the size of the CB, except for intra-predicted blocks where a CB might be split into multiple blocks when either intra sub-partitions (ISP) are used, or the CB size exceeds the maximum transform size.

Given the more flexible partitioning, keeping track of which samples have already been processed to determine availability of data is not as straightforward as in HEVC. Also, separate accounting of availability is needed for luma and chroma components when their respective block partitions diverge.

The set of supported CTU sizes is changed from 16 × 16, 32 × 32, and 64 × 64 in HEVC, to 32 × 32, 64 × 64, and 128 × 128 in VVC. Whereas the maximum CTU size is increased from 64 × 64 to 128 × 128, splits are constrained such as to enable a decoder to operate on units of 64 × 64 samples. Thus, a CU is either a combination of 1, 2 or 4 blocks of 64 × 64 samples, or a subset that fits within a block of 64 × 64 samples. In both cases, 64 × 64 blocks or subsets thereof do not cross the 64 × 64 grid. The same applies to transforms.

Whereas VVC's more flexible block partitioning adds complexity in a decoder, the increase is not substantial given that no sample-based operations are involved. On the other hand, an encoder is faced with many more possibilities to choose from, and effective strategies for pruning the search space are needed (see Section V).

B. Intra-Picture Prediction

HEVC features 35 luma intra-picture prediction modes including planar, DC, and 33 angular modes. VVC builds on this framework: it extends the number of angular modes to 65 and enables prediction from one of multiple reference lines [4]. In VVC, one of two polyphase 4-tap filters is used for angular prediction, thereby doubling (from 2 to 4) the number of MACs compared to HEVC. The smoothing process that HEVC applies to DC and horizontal/vertical angular modes is generalized and extended to the planar and most angular modes in VVC. Furthermore, the spatial extent of smoothing may be larger and cover the entire predicted block. This accounts for another 2 or 3 MACs per sample, as a weighted average is computed between a predicted sample and one or two reference samples.

VVC also features a separate set of 60 prediction modes (up to 32 for a given block size), where a vector containing left and above reference samples is multiplied by a matrix to obtain a predicted block. To limit complexity, the matrix multiplication applies to a reduced resolution representation of the block. For example, to predict a 16 × 16 block, the left and above reference samples are downsampled by a factor 4:1 to form a single vector containing 8 values. This vector is then multiplied by a matrix of size 64 × 8 to obtain 64 values that are arranged into an array of size 8 × 8. This array is then upsampled by a factor 1:2 along each dimension to the target block size. In this example, the MAC count is 32 for downsampling, 512 for the matrix multiplication, and 384 for upsampling for a total of 928, or about 3.63 per sample.[2] The MAC per sample count peaks at 5.38 for blocks of size 4 × 8 and 8 × 4. The computational complexity is therefore quite similar to that of angular prediction (6 MACs per sample). As with other intra prediction modes, care was taken to limit the bit width of constants and intermediate values. As the matrices have no regular structure, all elements (5120 signed 8-bit values) need to be stored in read-only memory (ROM). This amount includes a reduction in half that arises from the fact that one half of the matrices are the transpose of the other half.

Intra-picture prediction of chroma samples differs from prediction of luma samples mainly in the following ways: (a) a 2-tap filter is used for angular prediction instead of a 4-tap one, (b) the matrix multiplication modes are not supported,[3] and (c) a cross-component linear model (CCLM) may be used to predict chroma samples from collocated luma samples. The CCLM mode has higher complexity than angular prediction modes, as luma samples are downsampled using a 2-dimensional 6-tap filter before applying a linear model to each sample. The linear model is derived from reference samples located in neighboring blocks. An alternate plus-shaped 5-tap downsampling filter may be used when the chroma samples in a 4:2:0 video source are vertically aligned

[2] Downsampling involves generating 8 samples using a 4-tap filter (32 MACs), and upsampling involves generating 192 samples using a 2-tap filter (384 MACs) to augment the 64 samples resulting from the matrix multiplication and form a 16 × 16 predicted block.

[3] This restriction is lifted when processing data in the 4:4:4 format, where all components may use a same prediction mode.

with the even luma sample rows. Filtering and downsampling are not required when using the 4:4:4 chroma format.

It should be noted that VVC does not support constrained intra prediction, where only samples from neighboring intra-predicted blocks may be used for intra prediction. The omission of this feature simplifies the padding process of reference sample arrays.

To avoid writing narrow blocks of data to memory, the minimum width of an intra prediction is 4 samples for all color components. Therefore, when the ISP mode is used and a CB is split into narrow blocks, the split does not apply for prediction purposes, but does for transforms.

Unlike in HEVC, intra block copy (IBC) is featured in the core profile of VVC [5]. A block vector is signaled to indicate the relative position of a reference block within the current picture to be copied and used as prediction. A separate data buffer is typically maintained to store data to be referenced. The need for a separate buffer arises from the fact that loop filters are not applied to this reference data. The size of the reference buffer is limited to 16384 samples in VVC to keep implementation cost in check (for hardware in particular).

VVC supports significantly more intra prediction modes than HEVC. Evaluating all modes is generally not practical in an optimized encoder.

C. Inter-Picture Prediction

VVC and HEVC share a same core inter-picture prediction method: an 8-tap filter is used to generate a motion-compensated prediction, where the smallest luma block size is 8 × 8 for bi-predicted blocks, and 8 × 4 / 4 × 8 otherwise. Predictions may be weighted according to weights that are customizable on a per slice or picture basis, or to weights selected for each CU from a fixed set. Such prediction process requires 48 MACs per sample for bi-predicted N × 8 luma blocks (2 × 15 for horizontal filtering, 2 × 8 for vertical filtering, and 2 × 1 for the weighted average), and 62 MACs per sample for bi-predicted N × 4 luma blocks (2 × 22 for horizontal filtering, 2 × 8 for vertical filtering, and 2 × 1 for the weighted average).

VVC further defines an affine mode where two or three control motion vectors are provided for a CU. A motion vector is then derived for each 4 × 4 block within the CU by interpolating between the control ones. Given this smaller block size and to reduce complexity, a shorter 6-tap filter is applied in this mode, requiring 41 MACs per sample (2 × 13.5 for horizontal filtering,[4] 2 × 6 for vertical filtering, and 2 × 1 for the weighted average). To avoid increasing memory bandwidth, a fallback mode applies when the difference between motion vectors of adjacent 4 × 4 blocks within the CU is larger than a threshold. In this mode, a single motion vector is used for the entire CU such as to limit the size of the area in the reference picture required to predict the CU. When the fallback mode is not triggered, prediction refinement with optical flow (PROF) may be further applied to affine blocks. PROF computes horizontal and vertical sample gradients and multiplies them by per-sample horizontal and vertical motion vector adjustments, at a cost of about 48 operations per sample.[5]

Decoder-side Motion Vector Refinement (DMVR) can be used for bi-predicted CUs that don't use the affine mode, are coded with merge/skip, and for which the two reference pictures are equidistant from the current picture but in opposite temporal directions. DMVR performs a local motion search to increase the similarity between the two predictions. It operates on blocks of either 8 × 16, 16 × 8 or 16 × 16 samples, where larger CUs are split into smaller blocks of such size. Initially, motion compensation is performed using a bilinear filter to generate two (N+4) × (M+4) blocks (at a cost of 15.6 MACs per sample) within which a ±2 sample search is conducted. The sum of absolute differences (SAD) metric is used to determine the best match, where every other line in the block is skipped such as to reduce complexity. The computational cost is 12.5 absolute differences per sample (a 5 × 5 search window and half the samples are considered), or 37.5 operations per sample. The SADs are further used to determine a fractional motion vector refinement. The regular 8-tap interpolation filter is then applied using the refined motion vector, at a cost of 48 MACs per sample in the worst case (when DMVR operates on 16 × 8 blocks). To avoid increasing memory bandwidth, the (N+7) × (M+7) array fetched in consideration of the unrefined motion vector is extended using padding before generating a prediction using the refined motion vector. Bidirectional optical flow (BDOF) may be further applied. BDOF computes horizontal and vertical sample gradients within the CU. It further calculates a fractional motion refinement for each 4 × 4 block in the CU based on sums of gradients and prediction differences within 6 × 6 windows. A significant number of operations are added: about 41 per sample. To reduce the average number of operations, the DMVR search and BDOF processing are skipped if the difference between the two predictions after bilinear interpolation is smaller than a threshold. It should be noted that the refined motion vectors from the DMVR search are used for motion compensation (including chroma) and stored for reference by future pictures. However, they are not used for motion vector prediction and motion discontinuity determination in deblocking.

Intra- and inter-picture predictions may be combined (CIIP) in VVC. In such case, the planar prediction mode is always used for the intra prediction. Predictions are weighted according to the presence of neighboring intra-coded CUs. It should be noted that in this CIIP mode, the size of the intra prediction may not match the size of the transform (e.g., if the maximum transform size is set to 32 × 32, for a CU of size 64 × 8, the intra predicted block is of size 64 × 8, and there are two transform blocks of size 32 × 8).

[4] When predicting a 4 × 4 block, a horizontal 6-tap filter is applied to a 9 × 9 block to obtain a 4 × 9 block. The number of MACs for this operation is $4 \cdot 9 \cdot 6 = 216$, or 13.5 per sample in the target 4 × 4 block.

[5] For a 4 × 4 block, the number of operations can be broken down to: (a) 32 shift and 32 subtraction operations to compute sample gradients (4 operations per sample); (b) 2 MACs, 2 shifts and 4 min/max operations per sample to compute motion vector adjustments; and (c) 2 MACs and 2 min/max operations per sample to compute the product of gradients and adjustments. The number of operations is doubled in the bi-prediction case.

TABLE I
ESTIMATED WORST-CASE NUMBER OF OPERATIONS PER LUMA SAMPLE FOR BI-PREDICTED INTER CUs

Component	Core 16 × 4	Affine+PROF 8 × 8	DMVR+BDOF 16 × 8
Luma	124	130	214
Chroma 4:2:0	30	24	24
Chroma 4:4:4	96	84	84

VVC also supports resolution changes within a sequence, where a picture may be predicted from another picture with a difference size using reference picture resampling (RPR). The resampling process is integrated into the motion compensation process, which becomes a bit more involved since the phase of the interpolation filter may be different for each sample. However, the number of MACs is not increased in this case. There are restrictions on the downsampling ratio (2:1) to limit the size of the reference sample array and on the upsampling ratio (1:8).

Two inter-picture predictions may also be blended using one of 64 geometric patterns such as to more faithfully track object boundaries. This blending mode cannot be used in combination with affine, DMVR, BDOF, and weighted prediction, and does not have a significant impact on decoder complexity.

In VVC, as in HEVC, whenever the motion compensation process references a sample outside of picture boundaries, the closest sample inside the picture is used instead. It is not uncommon to extend an entire reference picture with padding to implement this process. In VVC, such an approach is not practical, and it is preferable to do padding on the fly, i.e., at the time a block is predicted. Indeed, with features such as wraparound motion compensation and subpictures, it is not possible to have a single padded reference picture that is suitable for all CTUs.

In VVC, motion vectors are represented in units of 1/16$^{\text{th}}$ of a luma sample and require 18 bits of storage per component (32-bit representations are typically used in software). For temporal storage, an 8 × 8 granularity is used in VVC instead of 16 × 16 in HEVC. To limit the memory amount and bandwidth, motion vector components are compressed to 10-bit floating point values for reference by subsequent pictures.

Inter-picture prediction of chroma components is simpler than luma for several reasons: (a) a 4-tap filter is used for interpolation; (b) DMVR reuses the refinement derived for luma; and (c) PROF and BDOF do not apply. It should be noted that when the affine mode is used, a single averaged motion vector is used for 4 × 4 chroma blocks (such averaging is not required for the 4:4:4 format).

Table I summarizes the estimated number of operations required to perform various inter prediction methods in the worst case. The combination of DMVR and BDOF almost doubles the number of operations for the luma component compared to the core case (HEVC-style prediction). It should be noted that for the core case, the worst-case complexity of VVC is higher than HEVC, in terms of computational complexity and of memory bandwidth. In VVC, every CU in a picture can be of size 16 × 4 and have unique bi-predictive motion parameters, whereas in HEVC, a PU of size 16 × 4 is always paired with another PU of size 16 × 12 inside a CU.

TABLE II
NUMBER OF MACs PER SAMPLE FOR VARIOUS TRANSFORM SIZES

Type	4 × 4	8 × 8	16 × 16	32 × 32	64 × 64
DCT	8	12	18	29	21.84
DCT+LFNST	16	18	16.5	18.88	16.57
DST	8	16	32	24	n/a

Motion vector prediction is also more complex in VVC than in HEVC. The number of merge candidates is increased to 6 and a history of recent motion vectors is maintained. The number of ways to code motion data is also increased: VVC introduces adaptive motion vector resolution (AMVR), symmetric motion vector difference (SMVD), and merge with motion vector difference (MMVD). Whereas these additional methods do not significantly impact a decoder, an encoder may have to test many more options for optimal performance.

D. Transforms and Quantization

VVC's transform and quantization design is based on that of HEVC. The DCT and DST approximations defined in HEVC are also featured in VVC. However, many more transforms are supported in VVC, including rectangular transforms, one-dimensional transforms, DSTs of size 8, 16, and 32, and DCTs of size 2 and 64. A secondary low-frequency non-separable transform (LFNST) may be applied on top of the DCT [6]. This secondary transform adds up to 8 MACs per sample and requires a fair amount of ROM (8K bytes). When the LFNST is used, only a few coefficients representing low spatial frequencies may be nonzero (8 or 16 depending on the transform size).

To support larger DSTs and DCTs without substantially increasing the number of MACs, coefficients representing high frequencies may only take value 0 for the larger transform sizes. For example, for a 32 × 32 DST, only the upper-left 16 × 16 region of the transformed block can take nonzero values. Table II summarizes the number of MACs per sample needed to compute inverse transforms of various sizes. The largest amount (32 MACs) is found for the 16 × 16 DST. It is not far off from the 32 × 32 DCT (29 MACs), which is the largest amount that applies to HEVC. In practice the 16 × 16 DST may run faster than the 32 × 32 DCT given its more regular structure (straight matrix multiply). It can also be noted that the DCT+LFNST case requires fewer operations than the DCT case for large transform sizes. This situation arises from the limitation on the number of nonzero coefficients when the LFNST is used: many inverse column transforms may be skipped. Whereas the MAC counts provided here consider the worst case, decoders are generally able to further reduce the amount of computation given that most transform coefficients tend to be equal to 0.

The quantization process is mostly the same as in HEVC, with one notable difference: the addition of trellis-coded quantization (TCQ), also referred to as dependent quantization in the VVC specification. For each transform coefficient, one of two quantizers is selected based on a state machine driven by the parity of previous coefficients in a scan order. This addition doesn't have much impact on the decoder, but it requires more resources in an encoder as rate-distortion optimized quantization is implemented using a trellis algorithm, where a trellis needs to be constructed for adequately selecting one of the two quantizers for every coefficient.

E. Entropy Coding

VVC uses a different arithmetic coding engine than AVC and HEVC. VVC's engine has better compression performance (resulting in about 1% bitrate reduction), but at the cost of higher complexity. Two probability estimates are maintained for each context, and the average of the two estimates is then used to determine subinterval ranges. During the development of VVC, the throughput of the engines of HEVC and VVC were evaluated in optimized software implementations. The throughput of the VVC engine was reported to be about 7% lower [6].

The number of contexts that VVC defines is significantly larger than HEVC (365 vs 154). The memory cost for context state storage was not deemed critical, and there was no sustained effort during the development of VVC to keep the number of contexts very low. More than half of VVC's contexts relate to coding of transform coefficients. The context derivation process is more complex in VVC than in HEVC, in particular for coding transform coefficients [8]. For example, to determine the context for a significance flag, the sum of five neighboring levels is computed and a threshold is applied. A second transform coefficient coding method is also provided for blocks where no transform is applied.

The use of TCQ tends to increase the number of coded bins, and VVC bitstreams tend to have more bins to process given a same number of bits [9]. To limit the number of context-coded bins for coding transform coefficients, a counter is maintained within a TU, and coding switches to bypass mode when a threshold equal to 1.75 times the TU area is reached. The number of coefficients for which the bypass mode applies is generally small but can reach 50 percent at high bitrates.

F. Loop Filters

VVC includes the deblocking and SAO loop filters that are present in HEVC, as well as two additional filters: Luma Mapping (LM) and Adaptive Loop Filter (ALF) [10]. SAO in VVC is identical to that in HEVC. It is thus not further discussed here.

LM defines a piecewise linear mapping function to adaptively modulate quantization according to luminance. This function is applied to each individual luma sample prior to adding a transform residual, and its inverse is applied after adding such residual. It should be noted that this function is also applied when the residual is zero. The complexity of this filter does therefore not vary much with bitrate. Whereas a single MAC operation per sample is required for both forward and inverse mapping, the determination of the segment index can be more complex. The number of segments is set to 16, which limits the size of lookup tables. When applying the forward mapping, the segment index can trivially be determined by extracting the four most significant bits of the luma sample value. For the inverse mapping, restrictions are placed on the parameters of the function such that determining the segment index involves comparing the luma sample value with one of 32 thresholds, where the threshold is selected based on the five most significant bits of the luma sample value.

VVC's deblocking filter is similar to HEVC's but differs in a couple of key aspects: (a) it is applied along edges lying on a 4×4 grid instead of an 8×8 grid, and (b) longer filters may be applied when large transforms are present. It maintains a feature present in HEVC: for each filtering direction, all edges can be filtered in parallel. The long filter may modify up to 7 samples on either side of an edge, where a weighted average of 14 samples is first computed. Each of the 14 samples is then weighted with this average. The computational cost of this filter is thus relatively low: 6 MACs per sample. However, given the smaller grid, the number boundary strength computations may be doubled, as well as the number of on/off filtering decisions based on local gradients.

ALF includes three components: a symmetric luma filter with a 7×7 diamond shape (25 taps, 12 coefficients), a symmetric chroma filter with a 5×5 diamond shape (13 taps, 6 coefficients), and a cross-component filter (CC-ALF) with a 3×4 diamond shape (8 taps, 7 coefficients). Given the symmetries and a unity constraint on the filter gain, ALF requires 12 MACs per sample for the luma filter, 6 MACs per sample for the chroma filter, and 8 MACs per sample for the cross-component filter. However, these MAC counts do not reflect the full complexity of the filter. The input to a MAC is given by the sum of two clipped differences (except for CC-ALF). Therefore, for each MAC operation there are 7 additional operations (2 subtractions, 4 min/max operations, and 1 addition). The complexity of luma ALF is thus closer to 108 operations per sample (12 coefficients with 9 operations per coefficient, MAC included). Furthermore, a different filter may be selected for each 4×4 luma block. The selection is based on local gradients of various orientations. This adaptivity requires about 25 operations per sample. Thus, the luma portion of the filter contributes to most of the complexity of ALF.

The number of required line buffers typically increases with each added filter and its spatial extent. To reduce the amount of line buffers, VVC defines a line buffer boundary for each CTU row. The number of lines that require buffering is limited to 4 lines above the CTU boundary for the luma component and 2 lines for the chroma component (regardless of the chroma format). Whenever a loop filter would refer to a sample on the other side of a line buffer boundary, a padding or mirroring process applies to avoid referencing such sample.

From the discussion above, it is evident that loop filters are major contributors to VVC complexity in a decoder. From an encoder perspective, one challenge is that multiple passes over a picture may be needed to derive suitable parameters for ALF.

G. High-Level Parallelism

HEVC introduced high-level parallelism tools such as tiles and wavefront parallel processing (WPP) to facilitate distributed processing. Such features are also supported by VVC. Additionally, the concept of subpictures is introduced, where a picture can be split into multiple regions that are independently coded over the duration of a sequence. Note that tiles are not fully independent since inter-picture prediction is not restricted to using data from collocated tiles in previous pictures. As in most HEVC profiles, high-level parallelism features are not mandated to be used by an encoder, and a decoder may thus not rely on their presence.

H. Miscellaneous

The nominal decoded picture buffer (DPB) size is increased from 6 in HEVC to 8 in VVC, leading to an approximately 33% larger memory requirement. This change was driven by the desire to support larger picture hierarchies for improved coding efficiency.

VVC supports scalability and extended chroma formats in dedicated profiles such as “Multilayer Main 10” and “Main 10 4:4:4”. Scalability support requires little adaptation at the signal processing level since RPR is already supported in the “Main 10” profile. For extended chroma formats, an adaptive color transform (ACT) for RGB content coding and a palette mode are added, but the signal processing is generally unchanged, and observations related to chroma remain valid unless otherwise noted.

I. Summary

Most aspects of the VVC design are more complex than their HEVC counterparts. The increase in complexity comes in two flavors: an increase in the number of modes that need to be implemented, and an increase in the computational complexity of the modes. On the inter-prediction front, there is no significant increase in memory bandwidth, but the worst-case computational complexity is substantially increased with the combination of DMVR and BDOF. This increased complexity may not be obvious on average, as these modes are enabled only when multiple conditions are met. The addition of ALF further contributes to the overall complexity. There is also an increase in ROM requirements as trained matrices are used for transforms and intra prediction.

A VVC encoder is faced with many more modes to choose from, and effective heuristics to reduce the search space are critical.

III. VTM Reference Software

The reference software for VVC is named VVC test model (VTM) [11]. It was originally derived from the HEVC test model (HM) [12]. Whereas much of the block level functionality was rewritten to support VVC’s partitioning structure and new coding tools, VTM still shares some code with HM, e.g., in HLS, CABAC parsing, and motion search. The software uses the more modern C++ 11 version and utilizes the C++ standard library more frequently.

The purpose of the reference implementation is to provide a framework for coding experiments and tool evaluation, and an example to guide implementors. Thus, data structures and code are generally not optimized for speed.

During the standardization process, most coding experiments were performed using common test conditions (CTC) [20], which were also used to obtain the results presented in this section. Compared to the HEVC common test conditions, an emphasis on higher image resolutions is added. Sequence class A contains six 4K-UHD sequences (3840 × 2160), class B five HD sequences (1920 × 1080), class C four SD sequences (832 × 480), and class D four quarter-SD sequences (416 × 240). Class E features video conferencing content and class F screen content, for which additional screen content coding tools are enabled. Most sequences are 5 or 10 seconds long (with different frame rates). Test configurations include “all intra” (AI) with intra pictures only, “random access” (RA) using a 16-frame hierarchical B picture GOP[6] and a random-access point every second, and “low delay” (LD) which uses the same picture order for coding as for display using B pictures (LDB) and optionally only P pictures (LDP). For all sequences, 10 bits per sample are input to the encoder. For some sequences, this involves a conversion from 8- to 10-bit before encoding.

Run times are significantly increased compared to HM, in particular for the AI configuration, where encoding a single 4K-UHD frame can take an hour. Thus, for AI only every 8th picture is coded. This reduces the total run time while showing similar coding results.

A different approach was taken for RA: each random-access segment (about 1s long) can be encoded independently in a parallel fashion. This reduces the run time to that of the slowest segment. The resulting bitstreams can then be combined into a single one [13]. In the worst case under CTC, encoding a 65-frame segment can take 2.5 days. Such parallelization is not possible for LD, because there are no random-access points except for the initial frame. The maximum resolution is thus limited to HD in this case.

Whereas previous test models did not include platform specific optimizations, VTM utilizes optimized ×86 SIMD code for distortion measurement, interpolation filters, affine motion estimation, ALF, and IBC hash maps to reduce the run times. The same functions are also provided in generic C++ code form. For a fairer run time comparison, some of the SIMD code was ported back into HM.

Table IV shows relative run times for class A sequences when disabling the SIMD code. The impact on AI encoding times is rather low (10% or less), but decoding times are up to 42% higher. In the RA case, the encoder run time almost doubles for Tango2 and FoodMarket4. The decoder run times may also increase by up to 95%. Thus, SIMD code provides a significant run time benefit when processing 4K-UHD sequences.

Table III lists available fast encoding configuration options and the CTC configurations that they are enabled in. In contrast to HM, most fast options are enabled to reduce the required encoding time for experiments. Fast tool options usually reduce the number of modes or partitions that are evaluated. They are tested with the specific CTC configurations and show low coding performance impact therein. For configurations

[6] In VTM version 11.0, it was increased to a 32-frame hierarchy.

TABLE III

VTM ENCODER FAST ENCODING MODE CONFIGURATION PARAMETERS

Configuration parameter	CTC	Description
ASR	RA	Adaptive motion search range by POC difference
ECU	none	Early CU termination: no splitting of skipped CUs
ESD	none	Early skip detection
FEN	all	SAD subsampling in large blocks, reduced iteration for bi-drectional motion vector refinement
FDM	all	Fast encoder decisions for merge mode
FastMrg	all	Fast methods for inter merge
AMaxBT	all	Adaptive maximum BT size
ContentBasedFastQtbt	none	Fast algorithm for a gradient-based selection of directional splits.
PBIntraFast	all	Fast assertion if the intra mode is probable
FastUDIUseMPMEnabled	all	Adapt intra direction search, accounting for MPM
FastLocalDualTreeMode	all	Intra coding speedup introducted with local dual tree mode
FastMIP	AI	Fast encoder search for MIP
ISPFast	AI	Fast encoder methods for ISP
TransformSkipFast	all	Reduced testing of the transform-skipping mode decision for chroma
BcwFast	RA, LD	Fast encoding of BCW
FastLFNST	AI	Fast encoding of LFNST

The CTC column lists the CTC configurations, if any, in which each parameter is enabled.

TABLE IV

RUN TIMES WHEN DISABLING SIMD OPTIMIZATIONS

	All Intra		Random Access	
Sequence	Encoder	Decoder	Encoder	Decoder
Tango2	110%	140%	197%	194%
FoodMarket4	107%	142%	197%	187%
Campfire	106%	140%	156%	155%
CatRobot	106%	135%	190%	177%
DaylightRoad2	105%	135%	186%	188%
ParkRunning3	103%	123%	163%	177%

TABLE V

RELATIVE RUN TIME OF VTM COMPARED TO HM

		All Intra		Random Access		Low Delay	
	Sequence	Enc	Dec	Enc	Dec	Enc	Dec
A	Tango2	1863%	162%	712%	153%		
	FoodMarket4	1564%	163%	614%	153%		
	Campfire	2713%	174%	1334%	161%		
	CatRobot	2493%	170%	792%	163%		
	DaylightRoad2	2880%	161%	921%	165%		
	ParkRunning3	3338%	186%	1088%	189%		
B	MarketPlace	3047%	166%	818%	154%	778%	151%
	RitualDance	2313%	173%	761%	155%	768%	145%
	Cactus	3624%	181%	922%	148%	807%	144%
	BasketballDrive	3052%	181%	1012%	156%	951%	149%
	BQTerrace	3115%	165%	880%	157%	672%	126%

that are significantly different from CTC, disabling some fast modes may achieve better coding performance.

Early versions of the VTM encoder, based on NextSoftware [12], supported two multithreaded modes of operation: wavefront-based and split-based [15]. However, given the lack of use these modes in CTC, they were not maintained and were partially removed over time. Thus, VTM multithreading is not further considered here.

Table V shows relative run times of VTM 10.0 compared to HM 16.23 for class A and B sequences. The numbers are averaged over all CTC rate points, with quantization parameter (QP) values 22, 27, 32 and 37. The encoder run time increase is usually much larger for higher bitrates and can reach a factor 40 in AI for QP 22. For RA and LD, the largest increase is about 15 times. Whereas the encoder run time increase is significant, the decoder run time increases much less with a maximum factor around 2, and averages between 1.5 and 1.8.

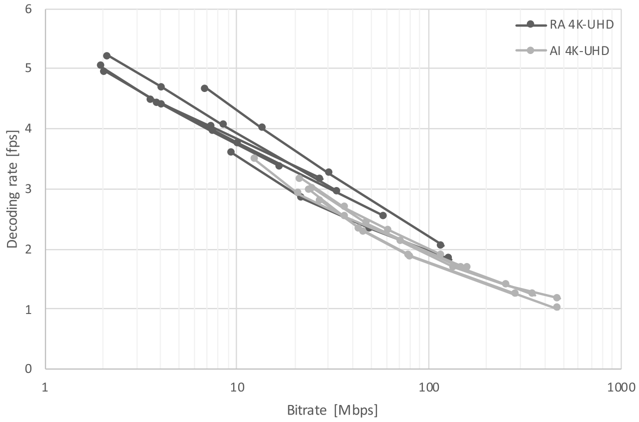

Fig. 2. VTM decoding rates in frames per second as a function of bitrate for 4K-UHD content in AI and RA configurations.

A. VTM Decoder

Fig. 2 shows VTM decoding speed in frames per second (fps) as a function of bitrate for class A sequences. Run times were measured on an Intel Xeon W-2140B processor. Whereas its base frequency is 3.2 GHz, the frequency observed during the experiment was about 3.9-4.0 GHz. The diagram shows that decoding speed depends mainly on bitrate, and to a lesser extent on sequence content and encoder configuration.

With decoding frame rates between 1 and 2 fps, decoding of 60 fps 4K-UHD content may take around 50 times longer than real time for low QP values in AI. In the RA configuration, decoding is faster but may still take around 30 times longer than real-time decoding. VTM can achieve real-time decoding for class D (quarter SD) and almost does for class C (SD).

These numbers illustrate well that significant optimization is required to achieve real-time decoding speeds for 4K-UHD resolution in software. Table VI and Table VII show exemplary profiling runs for selected class A sequences at QP values 22 and 37. The two sequences were chosen because they exhibit different characteristics that have an impact on the run time distribution: ParkRunning has finer detail and more localized motion than FoodMarket.

The tables show selected function blocks that consume significant amounts of time. Names indented to the right indicate that these blocks are part of the preceding function, e.g., Deblocking, ALF, and SAO are sub-functions of Loop Filters. Loop filtering requires a significant amount of time (31-46%) in both RA and AI configurations. Although deblocking always shows higher numbers than ALF, it should be noted that ALF uses optimized SIMD code and deblocking doesn't.

CABAC parsing has a much higher impact in AI, with up to almost 35% run time in ParkRunning. This can be explained by a higher bitrate resulting from the small spatial structures that lead the selection of smaller block partitions. The higher amount of residual coding in AI can also be observed in

TABLE VI

TIME SPENT IN VTM DECODER FUNCTIONS FOR RA CLASS A SEQUENCES

	FoodMarket4		ParkRunning 3	
	QP 22	QP 37	QP 22	QP37
Loop filters	38.4%	33.3%	36.6%	40.7%
Deblocking	18.7%	20.2%	20.4%	20.7%
ALF	15.6%	9.0%	11.7%	16.1%
SAO	1.2%	0.1%	2.4%	0.4%
Slice Decoding	59.8%	64.7%	61.1%	57.5%
CABAC CTU tree parsing	9.2%	7.2%	19.7%	9.0%
Inter reconstruction	41.1%	48.5%	32.8%	39.8%
DMVR	10.2%	26.9%	7.4%	21.9%
Inter prediction affine	9.9%	6.1%	8.6%	5.8%
Inter block	2.6%	1.0%	2.5%	1.0%
Inv transform + dequant	9.5%	4.2%	6.1%	1.9%
Intra reconstruction	4.2%	3.1%	3.5%	3.4%

TABLE VII

TIME SPENT IN VTM DECODER FUNCTIONS FOR AI CLASS A SEQUENCES

	FoodMarket4		ParkRunning 3	
	QP 22	QP 37	QP 22	QP37
Loop filters	40.4%	45.8%	31.7%	43.0%
Deblocking	26.3%	26.7%	19.4%	27.4%
ALF	11.1%	14.2%	7.6%	9.5%
SAO	3.6%	1.7%	3.1%	3.3%
Slice Decoding	56.9%	51.8%	64.7%	54.6%
CABAC CTU tree parsing	20.9%	11.3%	34.8%	19.1%
Intra reconstruction	33.8%	37.5%	27.9%	32.2%
Inv transform	14.2%	17.4%	9.9%	12.1%
dequant	2.0%	1.7%	2.5%	1.8%
InitIntraPattern	4.5%	3.6%	4.5%	4.6%
IntraPredAngular	2.5%	3.1%	1.7%	2.8%

the amounts of time (up to 17%) spent in inverse transform functions.

For RA a significant amount of time is spent on DMVR (up to 27%), but the percentage varies according to sequence characteristics and QP value. As shown in Fig. 7, the fraction of luma samples where DMVR is used varies with bitrates, and hence QP values.

B. VTM Encoder

The VTM encoder was instrumented with time measurement functions for the different modules. Such measured times are reported in Fig. 3. The diagram shows the average number of seconds consumed by each module for encoding one frame. The average is computed across all class A sequences using the RA configuration.

It can be seen that run times vary significantly over the range of tested QP values, generally showing the highest processing time for QP value 22. In particular, evaluations of intra and merge modes display the biggest increase for low QP values. This can be explained by the way that most search heuristics operate: modes are checked in a predefined order until the search algorithm terminates, typically when the residual coding cost is deemed low enough. With low QP values, the number of bits spend for residual coding increases, which leads to searching more partitions and more modes. As more intra, merge, and other modes are evaluated, more time is also spent in the quantization module, because it is used to compute costs associated with each mode. For higher QP values, most time is spent in affine search, followed by merge and intra.

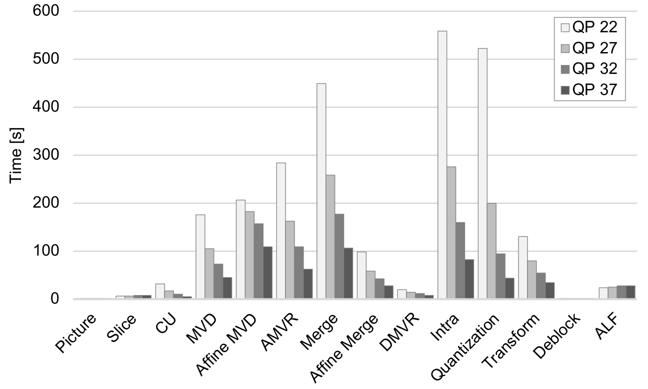

Fig. 3. Average per-frame VTM encoder run times across 4K-UHD CTC sequences for different modules and QP values.

Compared to the decoder, loop filters have a rather low impact on run time. The deblocking filter does not require a parameter search and follows the same rules as at the decoder, so its time impact is hardly measurable. The amount of time spent on ALF parameter estimation shows little variation over the range of QP values.

IV. OPTIMIZED SOFTWARE DECODERS

Multiple reports of optimized software decoders were made during the development of VVC [16]–[19]. They indicate that real-time decoders are achievable on the current generation of CPUs. There are three keys to achieve real-time decoding performance, in particular for higher resolution sequences: multithreading, adequate data structures and program flow, and efficient use of vector instructions (SIMD). Whereas the VTM software features SIMD code for several functions, including for example ALF, such code may not be optimal. Also, only a small fraction of the code uses 256-bit wide vector instructions such as AVX2. Multithreading can be added to the VTM software, but it isn't sufficient by itself to achieve real-time performance. The reasons for this insufficiency are inefficient data structures and program flow. Most, if not all, real-time implementations are written from scratch with lean data structures and improved program flow.

In the following, performance of real-time decoders is reported for several test cases: 4K-UHD AI and RA, 8K-UHD RA, and HD 4:4:4 RA, where bitstreams are generated according to [20] and [21] (with the sample bit depth is set to 10) using VTM version 10.0, unless noted otherwise. The decoder presented in [18] was used for the 4K-UHD and 8K-UHD cases, and the decoder described in [19] for the HD 4:4:4 cases.

A. 4K-UHD AI and RA

Two configurations are considered for 4K-UHD content: AI and RA. The AI configuration is interesting for two reasons: (a) VVC can be used for encoding single pictures (still images) and it reflects that use case, and (b) it gives insight into the complexity of entropy coding which tends to dominate

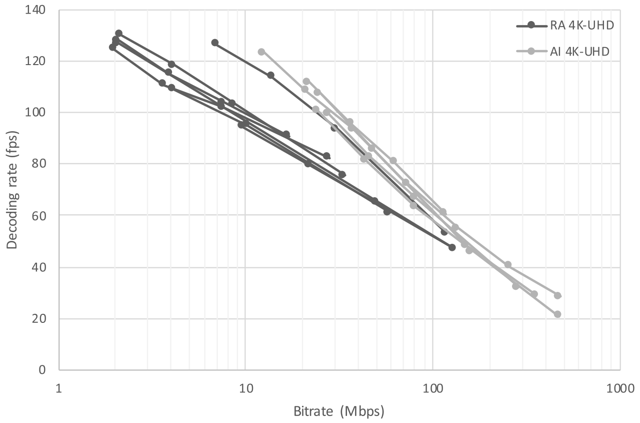

Fig. 4. Decoding rates in frames per second for an optimized decoder as a function of bitrate for 4K-UHD content in AI and RA configurations.

in this configuration as the bitrate is typically much higher. The RA configuration is generally used for broadcast and streaming applications. It aims to achieve the highest possible compression performance, in particular by allowing reordering of pictures and creating a prediction hierarchy.

Fig. 4 shows the decoding rates in frames per second achieved with an optimized decoder. The processor used in this experiment is an Intel Xeon W-2140B with 8 cores, and simultaneous multithreading is disabled (as in Fig. 2 for the VTM). The decoder aims to use all available CPU resources. The observed processor frequency during decoding is about 3.6-3.7 GHz. The decoding rate depends mainly on the bitrate, and rates of 60 frames per second or more are attained for bitrates up to about 60 Mbps. When comparing these results to those in Fig. 2, the average speedup over VTM is about 26× for RA and 32× for AI. These factors are substantially higher than what could result from applying multithreading to the VTM (i.e., 8× with 8 cores). It can be noted that one RA curve stands out in Fig. 4 and matches more closely the AI curves; it represents the Campfire sequence. The reason for its different behavior can be attributed to the high percentage of samples using intra prediction (e.g., 25% at QP 37) and low percentage of samples using bi-prediction (e.g., 43% at QP 37).

In practice, decoders often operate at a fixed frame rate, outputting frames at regular intervals, instead of decoding all frames as quickly as possible. Reflecting this common behavior, Fig. 5 shows the number of cores that are utilized when operating at 60 frames per second. About 7 cores are utilized on average at 40 Mbps, which is the maximum bitrate specified by Level 5.1 [1].

Table VIII and Table IX provide profiling results obtained for the AI and RA configurations with the QP value set to 27. The 6 sequences are Tango2 (TA), FoodMarket4 (FM), Campfire (CF), CatRobot (CR), DaylightRoad2 (DR), and ParkRunning3 (PR) [20].

As expected, entropy decoding (parsing) takes a significant chunk of decoding time in the AI case: about a third on average. Deblocking and ALF are also major contributors (about 20% each). Finally, intra prediction itself accounts for about a tenth of decoding time. There are significant variations across sequences which can be attributed to differences in bitrates.

In the RA case, inter-picture prediction is the largest contributor to decoding time (more than a third). Deblocking and ALF are also major contributors (about 15-20% each). As in the AI case, variations are mainly caused by differences in

TABLE VIII
4K-UHD All Intra Profiling Data

Seq	TA	FM	CF	CR	DR	PR	Avg
FPS	93.5	80.4	48.3	66.9	63.1	40.2	65.4
Bitrate	37.5	62.5	151.5	80.3	81.4	258.0	111.9
Parse	16.0	25.0	48.1	26.2	27.0	49.1	31.9
Intra	11.3	11.1	9.3	12.7	12.6	9.9	11.2
QT	7.0	7.1	4.5	5.8	6.4	6.1	6.2
MVP	0.6	0.5	0.4	0.5	0.4	0.3	0.4
LM	2.0	2.1	0.9	1.9	1.6	0.8	1.5
DB	23.4	20.1	13.4	19.3	18.7	12.1	17.8
SAO	1.8	1.5	1.1	1.6	1.4	1.1	1.4
ALF	29.7	24.3	14.3	20.4	20.7	12.6	20.3
Misc	8.2	8.5	7.9	11.7	11.1	7.9	9.2

FPS: Number frames decoded per second; Bitrate: Bitrate at 60 fps in Mbps; All remaining numbers are percentages of time spend in various processing units: QT: quantization and transforms; MVP: motion vector prediction; DB: deblocking

TABLE IX
4K-UHD Random Access Profiling Data

Seq	TA	FM	CF	CR	DR	PR	Avg
FPS	102.2	104.0	92.8	102.1	97.2	65.1	93.9
Bitrate	7.6	7.6	30.7	8.7	10.3	50.0	19.1
Parse	3.7	4.0	14.7	4.1	4.5	14.5	8.2
Intra	1.4	0.9	6.5	1.0	1.2	1.4	1.9
Inter	38.9	42.2	22.4	45.5	40.9	34.7	35.9
QT	1.2	1.3	3.1	1.0	0.8	1.9	1.7
MVP	1.6	1.7	1.4	1.8	1.6	1.8	1.6
LM	2.7	2.6	2.2	2.6	2.5	1.8	2.5
DB	16.4	15.1	18.6	15.1	15.2	13.4	16.1
SAO	1.5	1.6	1.2	1.7	1.5	1.3	1.5
ALF	25.0	22.8	22.7	19.3	24.0	19.6	22.2
Misc	8.0	8.4	8.1	8.6	8.7	10.1	8.5

FPS: Number frames decoded per second; Bitrate: Bitrate at 60 fps in Mbps; All remaining numbers are percentages of time spend in various processing units: QT: quantization and transforms; MVP: motion vector prediction; DB: deblocking

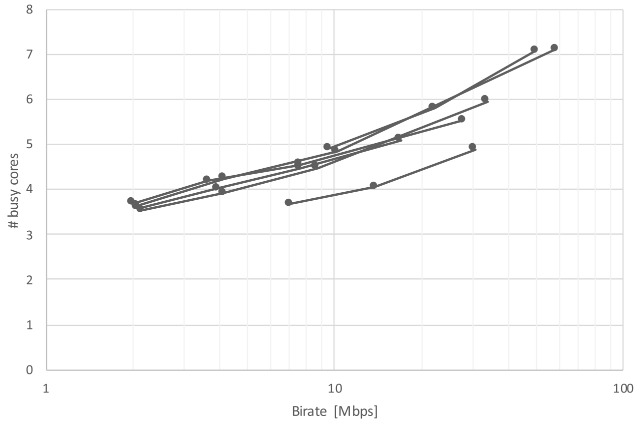

Fig. 5. Average number of busy processor cores as a function of bitrate for 4K-UHD content in RA configurations.

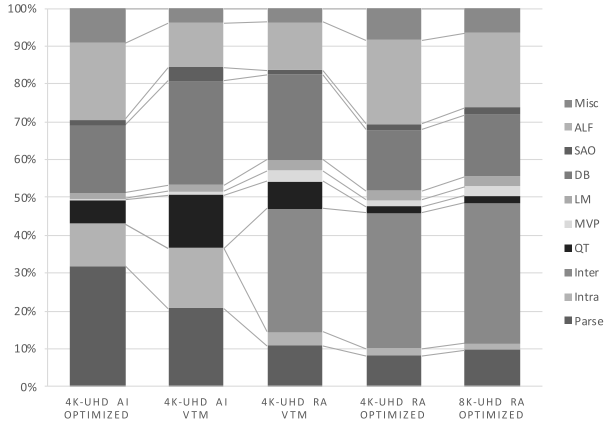

Fig. 6. Average decoding time distribution for 4K-UHD content under AI and RA conditions for optimized decoder and VTM, and 8K-UHD content under RA conditions for optimized decoder.

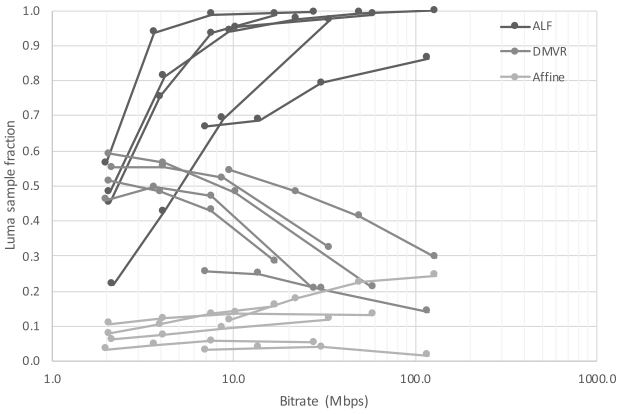

Fig. 7. Fraction of luma samples for which ALF, DMVR, and Affine are used as a function of bitrate. Each curve represents one 4K-UHD sequences.

bitrate. Fig. 6 shows the time distribution in graphical form along with the same statistics for VTM. It can be noted that the optimized decoder spends a smaller fraction of time than VTM on intra prediction, transforms and quantization, and deblocking, which indicates a more significant speedup from optimization for these modules.

Fig. 7 shows how frequently high-complexity modes are utilized as function of bitrate. ALF usage tends to grow with bitrate and reaches 95% or more for most sequences. DMVR usage generally hovers around 50% at low bitrates and tends to decrease with increasing bitrates. Affine usage tends to be lower (5-15%), and it remains fairly constant across bitrates, with a trend towards higher usage at higher bitrates.

B. 8K-UHD RA

Whereas 8K-UHD content (e.g., 7680 × 4320) is not included in the VVC common test conditions, interest in such high-resolution content is growing, including for immersive applications requiring a very wide field of view (e.g., 360 degrees). To test the ability to decode such high-resolution data in real time, several 8K-UHD sequences (60 fps) were encoded with VTM version 10.0 at a target bitrate of about 60 Mbps (an adequate QP value was selected for each sequence). The characteristics of the bitstreams are summarized in Table X. Substantial variations in the use of intra modes, DMVR, and ALF can be observed.

TABLE X

8K-UHD RANDOM ACCESS BITSTREAM CHARACTERISTICS

	Seq 1	Seq 2	Seq 3	Seq 4	Seq 5	Seq 6	Avg
Bitrate	52.2	56.7	63.1	54.1	63.2	55.8	57.5
Intra	2.7	25.7	4.7	14.4	5.3	6.0	9.8
Inter	97.3	74.3	95.3	85.6	94.7	94.0	90.2
Affine	3.2	4.3	4.4	5.2	7.7	8.2	5.5
PROF	2.6	3.9	4.2	4.8	7.3	6.1	4.8
DMVR	32.9	13.3	44.1	30.0	39.6	23.8	30.6
BDOF	7.8	5.0	43.0	23.9	14.7	7.8	17.0
ALF	60.4	100	97.4	83.2	96.9	99.8	89.6

All numbers are percentages of luma samples using a given mode, except bitrate which is expressed in Mbps.

TABLE XI

8K-UHD RANDOM ACCESS PROFILING DATA

	Seq 1	Seq 2	Seq 3	Seq 4	Seq 5	Seq 6	Avg
FPS	145	131	111	132	119	127	128
Load	16.0	18.3	21.9	17.6	21.0	19.5	19.1
Parse	9.7	9.7	9.2	10.1	9.7	9.7	9.7
Intra	0.2	4.0	1.2	2.4	1.1	1.5	1.8
Inter	42.1	26.4	39.4	36.4	38.6	39.3	37.0
QT	1.0	3.7	1.3	2.1	2.2	1.6	2.0
MVP	2.7	1.9	3.1	2.4	2.3	2.9	2.5
LM	3.2	2.4	2.5	2.6	2.4	2.5	2.6
DB	15.2	19.0	16.2	17.1	14.8	15.2	16.2
SAO	2.6	1.9	1.7	2.0	2.0	2.1	2.0
ALF	16.3	24.8	18.0	18.3	20.5	19.4	19.6
Misc	7.1	6.3	7.4	6.5	6.5	5.9	6.6

FPS: Number frames decoded per second; Load: average number of busy cores when processing 60 frames per second; All remaining numbers are percentages of time spend in various processing units: QT: quantization and transforms; MVP: motion vector prediction; DB: deblocking

To probe the limits of decoding performance, an AMD EPYC 7R32 processor is used. It features 48 cores and a maximum clock rate of 3.3 GHz (simultaneous multithreading is disabled). Given these vast resources, real-time decoding targets are substantially exceeded: using an evolution of the decoder presented in [18], between 110 and 145 frames are processed per second even though computing resources are not fully utilized. Profiling data is provided in Table XI. As in the 4K-UHD case, the main contributors to run time are inter-picture prediction, deblocking, ALF, and parsing. Together they account for about 85% of runtime. Between 16 and 22 processor cores are utilized to process 8K-UHD bitstreams when operating at 60 frames per second, and 19 on average.

C. HD 4:4:4 RA

Unlike HEVC and AVC, VVC supports high color fidelity formats such as 4:4:4 and 4:2:2 in its first version to address growing market needs. The decoding time overhead resulting from processing YUV 4:4:4 content relative to 4:2:0 content is analyzed using an optimized software decoder.

Table XII summarizes decoding speeds measured for YUV 4:4:4 content relative to the same content in 4:2:0 format, as well as profiling results obtained with the QP value set to 27. Six test sequences are considered: Traffic (TF),

TABLE XII
YUV 4:4:4 RANDOM ACCESS PROFILING DATA

Seq	TF	KM	LC	RF	VV	BC	Avg
FPS420	67.4	123.2	187.3	121.6	105.1	227.2	138.63
FPS444	47.7	79.6	137.9	96.2	75.5	143.2	96.68
Bitrate	1.6x	1.7x	1.1x	1.1x	1.2x	3.5x	1.7x
Speed	71%	65%	74%	79%	72%	63%	71%
Parse	10.1	7.9	4.4	6.2	5.5	6.8	6.82
QT	3.0	3.8	2.1	2.1	3.0	2.5	2.75
Intra	3.3	4.4	3.3	2.5	5.1	2.6	3.53
Inter	40.1	43.0	54.1	54.5	48.2	51.9	48.63
MVP	3.2	2.8	3.7	3.3	3.3	2.9	3.20
DB	19.3	15.4	10.7	13.6	15.4	12.9	14.55
SAO	14.4	13.6	16.8	12.8	13.9	15.3	14.47
ALF	2.1	5.3	1.2	1.7	2.6	0.9	2.30
Misc	4.5	3.6	3.7	3.4	3.1	4.2	3.75

FPS: Number frames decoded per second; Bitrate: Bitrate of YUV 4:4:4 encoding relative to YUV 4:2:0; Speed: Speed of YUV 4:4:4 decoding relative to YUV 4:2:0; All remaining numbers are percentages of time spend in various processing units: QT: quantization and transforms; MVP: motion vector prediction; DB: deblocking

Kimono1 (KM), EBULupoCandlelight (LC), EBURainFruits (RF), VenueVu (VV), and BirdsInCage (BC) [21]. The processor used in the experiment is an Intel Core i7-9700 with 8 cores clocked at 3.0 GHz (frequency scaling is disabled and simultaneous multithreading unsupported).

The observed decoding speed for YUV 4:4:4 is lower than for 4:2:0 by 29% on average. In other words, it takes only about 40% more time to decode a 4:4:4 picture than a 4:2:0 one, even though the number of processed samples is doubled. As in the 4:2:0 case, inter-picture prediction consumes the largest portion of the total decoding time (about 50%) in the 4:4:4 case. In-loop filters also contribute significantly (about 30%). The run time overhead for processing 4:4:4 content varies across sequences. Such variations can be attributed to differences in bitrates between 4:4:4 and 4:2:0 bitstreams, which range from 1.1× to 3.5× (BirdsInCage), but also to other factors, such as possibly different mode selections. In general, it is observed that when the bitrate increase is less than 60%, the decoding speed difference between 4:2:0 and 4:4:4 is close to the reported average (71%). Fig. 8 provides an example of a decoding time distribution comparison between 4:2:0 and 4:4:4 for one sequence. In this example, the bitrate difference between 4:4:4 and 4:2:0 is small (less than 10% for a same QP value). Decoding time differences are not uniform across modules. For example, differences between 4:2:0 and 4:4:4 are less pronounced for parsing than for inter prediction. When the QP value is lowered and the bitrate increases, the time consumed by entropy decoding and in-loop filters increases more noticeably for 4:4:4 than for 4:2:0.

D. Discussion

Whereas no direct comparison to HEVC is available, it can be reasonably said that a VVC decoder requires about 1.5-2.0× the computing resources of an HEVC decoder. In [2] it was reported that a single-threaded HEVC decoder processed about 100 frames per second of HD content encoded with QP set to 27. The performance reported here is also about 100 frames per second, but the resolution is 4 times higher and the load is distributed across 8 cores. There is thus roughly a ratio of 2 overall. It should be noted that in the HEVC case, the bit depth was lower (8 bits) and the processor different, and this ratio is therefore approximative. The relative time spent in each module is not substantially different from HEVC, although there is an increase for loop filters, mainly because of the addition of ALF. Whereas the results presented in this paper were obtained for 10-bit video, a software decoder may achieve higher frame rates when processing 8-bit video for two reasons: (a) code specialized for 8-bit processing can achieve a higher degree of SIMD parallelism; and (b) storage can be reduced from two bytes per sample to one, thereby reducing memory bandwidth and cache pressure. A frame rate increase of about 15% was reported for a real-time VVC decoder that incorporates code and storage optimized for multiple bit depths [19].

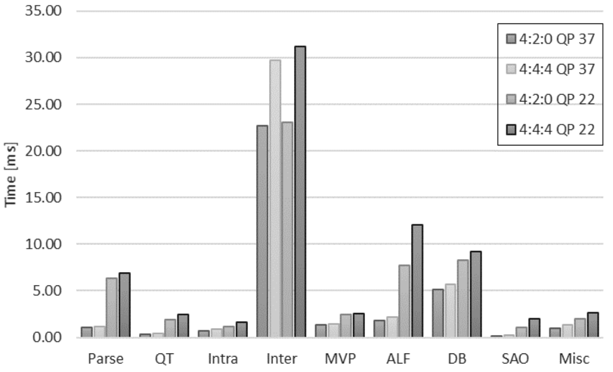

Fig. 8. Decoding time distribution for EBURainFruits sequence under Random Access condition.

V. OPTIMIZED SOFTWARE ENCODERS

As discussed previously, whereas the algorithms defining the VVC standard are more complex than those included in the HEVC specification, it is not the main factor contributing to the VVC encoding complexity. Many tools introduced in the new standard can be enabled or disabled independently for each CU, creating a very large search space. For this reason, contrary to the decoder, implementation efficiency is not the main focus of encoder optimization. Smart search algorithms and early termination strategies must be utilized to reduce encoding complexity, while minimizing any resulting bitrate increase. Both approaches need to be combined to achieve good quality encoding in a manageable amount of time.

In the following section, aspects of efficient search space traversal will be discussed, starting with partitioning and continuing with encoding tools. Afterwards, instrumentation of the more efficient VVenC encoder [23], [24] is compared to the VTM, and possible trade-offs are discussed.

A. Partitioning Complexity

The quadtree with binary- and ternary-tree partitioning is one of the main innovations in VVC, and significantly contributes to compression improvements over HEVC. The algorithmic complexity of the normative CU splitting process is relatively low. The added complexity is put almost entirely on the encoder side, which needs to find an effective split configuration for each CTU.

In HEVC, a CTU can be recursively split only in a quadtree fashion, with a maximum CTU size of 64 × 64. When increasing the maximum CTU size to 128 × 128, the search space is increased by the power of 4 (divided by 4 to account for the reduction of the number of CTUs). As defined in VVC's split semantics, multitype trees (MTT) comprising binary and ternary splits are rooted at the leaves of the quadtree (QT) structure. Because of various size- and position-dependent split restrictions, it is not trivial to quantify their exact impact on the search space beyond the quadtree splits. In the following, the runtime to bitrate reduction trade-offs achievable by varying the allowed partitioning depth are discussed.

In [25], it was estimated that 97% of the VTM encoding runtime could be saved in the AI configuration, if the selected partitioning could be directly inferred rather than determined through search. The VTM encoder already contains search strategies based on dynamic programming and heuristics to improve run times [26]. These enable VTM, in its default configuration, to run over 7× faster than if a full partitioning search was performed.

With those fast algorithms in place, the partitioning complexity can be efficiently controlled by varying the maximally allowed depth of the multitype trees. A thorough analysis of possible trade-off points was performed in [23]. It showed that the flexible partitioning structure provides considerable gains, even without allowing recursion in the MTT. In VTM, without such recursion, over 70% runtime reduction is possible for a BD-rate loss of around 4.5%. No state-of-the-art fast partitioning algorithm can achieve such a trade-off when applied on top of already available fast methods [26].

The methods described in [26] were designed with consideration for the default configuration of VTM, i.e., with multiple recursion levels in both the QT and MTT structures. When applied to a lower complexity configuration, such as those discussed below, the 7× speedup may be reduced to less than 3×. As discussed in [23], some methods can be adapted to a lower-complexity operating point, also because the incline of the Pareto set at such point allows for more aggressive strategies.

B. Tool Complexity

During the VVC standardization process, the performance of individual tools was continuously monitored and reported [27]. Whereas this tracking was helpful to identify potential regressions, it was conducted around the slow working point of VTM CTC, where some of the performance effects are masked by extensive searches conducted for other tools.

In [23], a "tool on" analysis based on a partitioning-only configuration was performed for several VVC tools using VTM and an early version of an optimized VVC encoder. Interestingly, some of the tools that add significant complexity in a decoder (e.g., BDOF and DMVR) are the most efficient for the encoder. Whereas these tools add complexity, as measured in runtime, their implicit nature (i.e., no explicit signaling per CU) ensures that they do not increase the encoding search space. A similar effect can be observed for the selection between multiple transform types, where the implicit mode provides a very attractive trade-off.

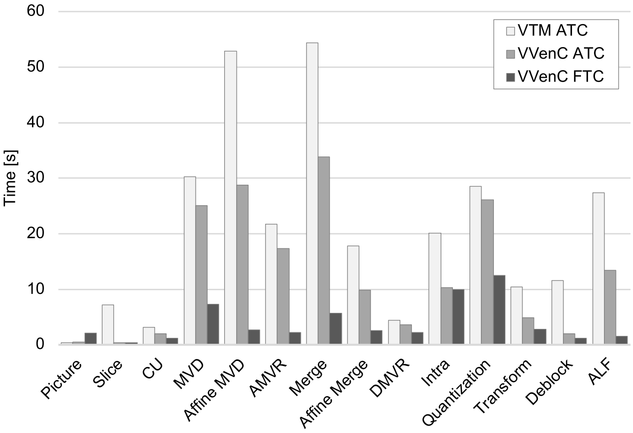

Fig. 9. Average per-frame encoding time distribution for 4K content under random access conditions in VTM and the optimized VVenC encoder in ATC and FTC configurations.

In [27], ALF appears to provide substantial gains with only a small encoding time overhead. However, a "tool-on" test shows a much larger runtime increase. Since ALF is an in-loop filter, the encoding time overhead is additive and not multiplicative to the CU-level tools. When applied to a much faster configuration, the relative overhead increases. To mitigate this effect, only one clipping value (instead of four) may be considered for each ALF coefficient to reduce the search space, and hence the encoding time. Overall, the analysis in [23] shows that keeping the search space manageable is at least as important for an optimized encoder as efficient implementation.

In the VTM reference encoder, the implementation of tools and their associated search algorithms were designed to showcase maximal compression gains. Many of the achieved trade-offs are not practical for a more efficient encoder. Adapting the algorithms of the high-gain high-overhead tools is possible with slight losses to achieve alternative trade-offs, as discussed in [23].

C. Profiling Results

Compared to VTM, the optimized encoder implementation in [23] features additional SIMD code, mostly ported from [17], as well as structural improvements. In Fig 9, profiling data from the optimized encoder is compared to the VTM for two configurations: a first one with search algorithms identical to VTM for the maximal common set of tools between the two encoders (ATC), and a second one with search algorithm adaptions aimed at trade-offs with lower runtime overhead (FTC). As shown in Table XIII, both configurations provide very similar results in terms of bitrate savings, with FTC running around four times faster than ATC. Compared to Fig. 3, a runtime increase for the deblocking filter is apparent. It is caused by the consideration for deblocking during the CU-level mode search. All other categories show lower runtime.

TABLE XIII

BD-Rate Results and Encoding Times of VTM, VVenC, and HM Encoders, Relative to VTM CTC Anchor

	Y [%]	U [%]	V [%]	EncT [%]
VTM 10.0 CTC	0.00	0.00	0.00	100.00
VTM 10.0 ATC	5.68	6.14	6.68	24.14
VVenC ATC	5.68	6.14	6.68	14.75
VVenC FTC (Medium)	5.62	1.24	2.47	3.80
VVenC FTC MT2	5.83	0.73	2.36	2.08
VVenC FTC MT4	5.83	0.73	2.36	1.28
VVenC FTC MT6	5.83	0.73	2.36	1.17
VVenC Slow	0.38	-4.83	-4.08	11.34
VVenC Faster	40.21	58.16	56.47	1.54
HM 16.22 CTC	62.66	79.57	80.93	11.27

All BD-rate and encoding time numbers are averaged across HD and UHD sequences from the JVET common test conditions test set. The MT results represent additional speedups when using multithreading with the specified number of threads.

In Fig 9, the differences between VTM and VVenC for the ATC configuration illustrate the improvements provided by the optimized implementation, as the two encoders yield equivalent results. The differences between the two VVenC configurations indicate where the runtime can be efficiently reduced within one implementation. The increase in the "Picture" category is caused by the inclusion of a pre-processing filter [28]. It efficiently compensates for some of the coding loss caused by algorithmic adaptations. The optimized implementation provides most time savings in the filtering, intra search, and affine search stages. The FTC configuration mostly reduces the search space in the inter-mode search and ALF stages. The fallback intra mode seems to be required to a similar extent in both configurations. Because of the reduced number of tested modes, time spent for quantization (TCQ in particular) and transforms is also reduced.

D. Coding Efficiency

Changes to encoder configurations and search algorithms result in coding efficiency differences. Table XIII shows the impact of various configurations on BD-rate and encoding time. The VVenC encoder runs around 40% faster than VTM in the ATC configuration (the largest common set of tools between the two encoders using the same search algorithm), which can be attributed to optimized implementation.

With a reduced complexity configuration and adapted search algorithms, VVenC can achieve very similar coding efficiency running almost four times faster in the FTC configuration, which is over 26× faster than the VTM reference encoder, or around 85× faster when multithreading is enabled.

Results for two additional configurations are also provided. In the Slow configuration, the encoder achieves compression performance very close to VTM, while running almost 9× faster. This is as fast as HM version 16.22, but with about a 38% BD-rate gain over it. In the Faster configuration, the encoder runs 60× faster than VTM, but large BD-rate losses, in excess of 40%, are observed. Nonetheless, the configuration is around 7× faster than the HM, while still achieving around 15% BD-rate gain over it.

E. Multithreading

The use of multithreading can further reduce encoding times. Table XIII shows the speedup achieved for the medium-complexity configuration when using 2, 4, and 6 threads. The basic multithreading included in VVenC is CTU-line based. Because of the number of CTU rows is generally not a multiple of the number of threads, the workload assigned to each thread may be different. The workload also varies as a function of picture resolution. Therefore, the combined results in Table XIII do not show the actual speedup numbers for a given resolution. For 4K-UHD, 6 threads provide a speedup of about 4×, and for HD, 4 threads provide a speedup of about 2.5×. Speedups saturate quickly when increasing the number of threads above these values.

A few algorithmic issues still need to be resolved to fully unlock the potential of multithreading. Most notably, the application of ALF seems problematic. ALF coefficients are typically derived by considering the whole picture after SAO processing. Therefore, ALF parameter selection cannot be included into the main CTU encoding loop. Whereas it is possible to include some of the ALF workload into the CTU-line parallel tasks, a major part of it is delayed until after encoding the whole picture. This introduces additional encoding latency and precludes encoding two dependent pictures in parallel. In the future, approximate models for ALF filter estimation or alternative techniques may be developed to mitigate this problem in VVenC.

F. Summary

As discussed in this section, encoding complexity is not something inherent to the standard, but rather to the encoder. The presented encoder can achieve most of VTM's compression gains over the HEVC reference software with 85× lower encoding time, showcasing an attractive trade-off. VVC provides many opportunities for an encoder to find an optimal encoding. It lies in the hands of the implementers to find search algorithms that do this efficiently within the restrictions of the given application.

VI. Conclusion

VVC was shown to be more complex than HEVC. Nevertheless, it can be implemented in software on current generation CPUs. An optimized decoder was shown to process 8K bitstreams in real time. The performance of an optimized encoder was also discussed, demonstrating that substantially better run time vs compression trade-offs can be achieved than with VTM under common test conditions. VVC is young, and it is expected that encoders will further improve in years to come.

References

[1] B. Bross, J. Chen, S. Liu, and Y.-K. Wang, *Versatile Video Coding (Draft 10)*, document JVET-S2001, 19th JVET Meeting, Online Meeting, Jun. 2020.

[2] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "High Efficiency Video Coding complexity and implementation analysis," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1685–1696, Dec. 2012.

[3] I.-K. Kim, J. Min, T. Lee, W.-J. Han, and J. Park, "Block partitioning structure in the VVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1697–1706, Dec. 2012.

[4] J. Pfaff *et al.*, "Intra prediction and mode coding in VVC," *IEEE Trans. Circuits Syst. Video Technol.*, to be published.

[5] T. Nguyen *et al.*, "Overview of the screen content support in VVC: Applications, coding tools, and performance," *IEEE Trans. Circuits Syst. Video Technol.*, to be published.

[6] X. Zhao *et al.*, "Transform coding in the VVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, to be published.

[7] F. Bossen, *CE5 on Arithmetic Coding: Experiments 5.1.1, 5.1.2, 5.1.3, 5.1.4, 5.1.5, 5.1.6, 5.1.7, 5.1.8, 5.1.10, 5.1.11, 5.1.12, 5.1.13, 5.2, and More*, document JVET-M0453, 13th JVET Meeting, Marrakech, Morocco, Jan. 2019.

[8] H. Schwarz *et al.*, "Quantization and entropy coding in the versatile video coding (VVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/9399502

[9] M. Zhou, *AHG16: A Study of Bin to Bit Ratio for VTM7.0*, document JVET-Q0102, 17th JVET meeting, Brussels, Belgium, Jan. 2020.

[10] M. Karczewicz *et al.*, "VVC in-loop filters," *IEEE Trans. Circuits Syst. Video Technol.*, to be published. [Online]. Available: https://ieeexplore.ieee.org/document/9399506

[11] *VTM VVC Reference Software*. Accessed: Mar. 31, 2021. [Online]. Available: https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM

[12] *HM HEVC Reference Software*. Accessed: Mar. 24, 2021. [Online]. Available: https://vcgit.hhi.fraunhofer.de/jvet/HM

[13] X. Ma, H. Chen, and H. Yang, *Simplification of the Common Test Condition for Fast Simulation*, document JVET-B0036, 2nd JVET Meeting, San Diego, CA, USA, Feb. 2016.

[14] A. Wieckowski *et al.*, *NextSoftware: An Alternative Implementation the Joint Exploration Model (JEM)*, document JVET-H0084, 8th JVET Meeting, Macao, São Lourenço, Oct. 2017.

[15] A. Wieckowski *et al.*, *Thread Parallel Encoding*, document JVET-J0036, 10th JVET Meeting, San Diego, CA, USA, Apr. 2018.

[16] J. Raj Arumugam *et al.*, *AHG16: Early Implementation of VVC Software Player and Demonstration on Mobile device*, document JVET-P0307, 16th JVET Meeting, Geneva, Switzerland, Oct. 2019.

[17] A. Wieckowski *et al.*, "Towards a live software decoder implementation for the upcoming versatile video coding (VVC) codec," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2020, pp. 3124–3128.

[18] F. Bossen, *AHG16: Performance of a Reasonably Fast VVC Software Decoder*, document JVET-S0224, 19th JVET Meeting, Online Meeting, Jun. 2020.

[19] B. Zhu *et al.*, "A software decoder implementation for H.266/VVC video coding standard," 2020, *arXiv:2012.02832*. [Online]. Available: http://arxiv.org/abs/2012.02832

[20] F. Bossen, J. Boyce, X. Li, V. Seregin, and K. Sühring, *JVET Common Test Conditions and Software Reference Configurations for SDR Video*, document JVET-N1010, 14th JVET Meeting, Mar. 2019.

[21] Y.-H. Chao, Y.-C. Sun, J. Xu, and X. Xu, *JVET Common Test Conditions and Software Reference Configurations for Non-4:2:0 Colour Formats*, document JVET-R2013, 18th JVET Meeting, Apr. 2020.

[22] F. Bossen, X. Li, and K. Suehring, *JVET AHG Report: Test Model Software Development (AHG3)*, document JVET-S0003, Joint Video Experts Team (JVET), Jun. 2020.

[23] J. Brandenburg *et al.*, "Towards fast and efficient VVC encoding," in *Proc. IEEE 22nd Int. Workshop Multimedia Signal Process. (MMSP)*, Sep. 2020, pp. 1–6.

[24] *Fraunhofer Versatile Video Encoder VVenC Software*. Accessed: Mar. 22, 2021. [Online]. Available: doi: https://github.com/fraunhoferhhi/vvenc

[25] A. Tissier, A. Mercat, T. Amestoy, W. Hamidouche, J. Vanne, and D. Menard, "Complexity reduction opportunities in the future VVC intra encoder," in *Proc. IEEE 21st Int. Workshop Multimedia Signal Process. (MMSP)*, Sep. 2019, pp. 1–6.

[26] A. Wieckowski, J. Ma, H. Schwarz, D. Marpe, and T. Wiegand, "Fast partitioning decision strategies for the upcoming versatile video coding (VVC) standard," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Sep. 2019, pp. 4130–4134.

[27] W.-J. Chen *et al.*, *JVET AHG Report: Tool Reporting Procedure (AHG13)*, document JVET-Q0013, Joint Video Experts Team (JVET), Jan. 2020.

[28] J. Enhorn, R. Sjöberg, and P. Wennersten, "A temporal pre-filter for video coding based on bilateral filtering," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2020, pp. 1161–1165.

Frank Bossen (Senior Member, IEEE) received the Ingénieur EPF degree in computer science and the Ph.D. degree in communication systems from the École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, in 1996 and 1999, respectively.

He has been active in video coding standardization since 1995, and contributed to the MPEG-4, H.264/AVC, H.265/HEVC, AV1, and H.266/VVC specifications. He has held a number of positions with IBM, Sony, GE, and NTT DOCOMO in Japan, Switzerland, and USA. He is currently a Principal Research Engineer with Sharp Electronics of Canada Ltd. in the Greater Toronto Area.

Dr. Bossen has been appointed as an Editor of the VVC reference software and VVC conformance specifications.

Karsten Sühring received the Dipl.-Inf. degree in applied computer science from the University of Applied Sciences, Berlin, Germany, in 2001.

He is currently with the Image and Video Coding Group, Fraunhofer Institute for Telecommunication–Heinrich Hertz Institute, Berlin, where he has been engaged in video coding standardization activities. His current research interests include coding and transmission of video and audio content, and software design and optimization.

Mr. Sühring has been a Co-Chair of the JVET ad hoc group on software development since May 2018. He is one of the coordinators for the VTM Reference Software for VVC. He has been an Editor of the reference software of H.264/AVC and H.265/HEVC.

Adam Wieckowski received the M.Sc. degree in computer engineering from the Technical University of Berlin, Berlin, Germany, in 2014.

In 2016, he joined the Fraunhofer Institute for Telecommunications–Heinrich Hertz Institute, Berlin, as a Research Assistant. He worked on the development of the software, which later became the test model for VVC Development. He contributed several technical contributions during the standardization of VVC. Since 2019, he has been a Project Manager coordinating the technical development of decoder and encoder solutions for the VVC standard.

Shan Liu (Senior Member, IEEE) received the B.Eng. degree in electronic engineering from Tsinghua University, and the M.S. and Ph.D. degrees in electrical engineering from the University of Southern California.

She was formerly with MediaTek, MERL, Sony Electronics, and Sony Computer Entertainment America (now Sony Interactive Entertainment). She is currently a Distinguished Scientist with Tencent. She has made numerous technical contributions to international standards, such as H.265/HEVC, H.266/VVC, OMAF, DASH, and PCC. Her research interests include audio-visual, high volume, immersive and emerging media compression, intelligence, transport, and systems. She served as a Co-Editor for HEVC SCC and VVC.