TOWARDS A LIVE SOFTWARE DECODER IMPLEMENTATION FOR THE UPCOMING VERSATILE VIDEO CODING (VVC) CODEC

Adam Wieckowski, Gabriel Hege, Christian Bartnik, Christian Lehmann, Christian Stoffers, Benjamin Bross, and Detlev Marpe

Fraunhofer HHI, Video Coding & Analytics, Berlin, Germany

ABSTRACT

Versatile Video Coding (VVC) is the emerging video coding standard to be finalized by the Joint Video Experts Team in July 2020. Compared to its predecessor, the High Efficiency Video Coding (HEVC) standard, VVC provides 50% bit-rate reduction at comparable visual quality for natural video content in high-definition (HD) and ultra high-definition (UHD) resolution. To achieve this, the standard incorporates more advanced and generalized algorithms, leading to an increase in computational complexity. This includes for example additional in-loop filters, decoder-side motion refinement and search as well as an increased number of transforms, which creates a grand challenge for implementers to achieve live decoding on general-purpose CPUs.

In this paper, the work on an efficient software decoder implementation for the upcoming VVC standard is described, including optimization of sample operations using single instruction multiple data (SIMD) instructions and parallelization approaches with multithreading. As a result, the presented decoder can perform live decoding of 10bit HD video at 60 frames per second (fps) and 10bit UHD video at 30fps on modern mobile consumer hardware, showcasing that VVC live decoding is possible already right before finalization of the standard.

Index Terms— VVC, software decoding, video coding, SIMD, AVX2, multi-threading.

1. INTRODUCTION

VVC [1][2] is the emerging new video coding standard jointly developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Pictures Experts Group (MPEG), and to be finalized in July 2020. VVC provides 50% bit-rate reduction over HEVC [3] standard and besides that, VVC is designed to efficiently cope with even higher resolution, dynamic range and color gamut as well as with a variety of different applications. This includes adaptive streaming use cases with reference picture resampling and temporal/spatial scalability, immersive applications using 360-degree video, and screen-content coding for gaming and screen sharing applications. The current state of standard development is reflected in the specification draft text [1]. Accompanying the draft text is the VVC test model (VTM) [4], which represents a reference software implementation of an encoder and decoder. Its main focus is for experiments with new technology and some coarse complexity analysis [5]. Being a reference implementation, it targets mainly correctness, completeness and readability and thus, it is not supposed to be used in production or consumer systems. For those reasons the software is not properly optimized for performance. It only has basic SIMD algorithm implementations, and does not support any parallelization features on the decoder side.

As the comparison with the HEVC test model (HM) [4] shows, the decoding complexity of the new standard is going to be about twice that of HEVC [6]. Based on the times reported in [4], for live play-back of UHD video content up to 40 Mbps, a decoder would on average need to be 30 times as fast as the reference software, for HD content up to 20 Mbps, 7× speed-up is required. Some previous work is available on VVC encoding complexity reduction [7-9], but to our knowledge there is no current work on software decoder complexity reduction.

In this paper, an optimized VVC software decoder implementation is presented. As the standard is still under development, the decoder is compatible with VVC draft 5 [1] and VTM-5.2 with further integration and development work ongoing, targeting compliance with the final standard shortly after finalization in summer 2020.

2. VERSATILE VIDEO CODING COMPLEXITY

A schematic visualization of a VVC decoder pipeline is shown in Fig. 1. It resembles the decoder schematics of an

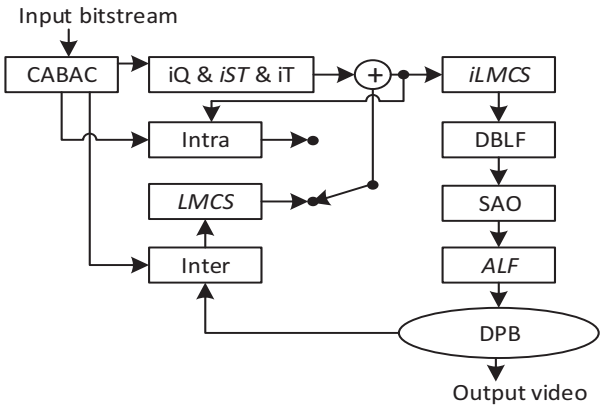

Fig. 1: Schematic of a Versatile Video Coding decoder [10][11]. New pipeline steps are marked cursive.

HEVC decoder [10], with a few additional steps. An input stream is entropy decoded through CABAC [12]. It provides information about the block partitioning, used predictors (inter-, intra-picture prediction) and the coded residual data. The latter is then dequantized (iQ) before secondary and primary inverse transformations (iST, iT) are applied to reconstruct the residual data. The reconstructed residual and prediction are added together and processed by a series of in-loop filters, including inverse luma mapping and chroma scaling (iLMCS), deblocking filter (DBLF), sample adaptive offset filter (SAO), and adaptive loop filter (ALF).

The added complexity over HEVC is caused both by the inclusion of new processing steps [13-16], but also through improved and more complex prediction algorithms, both in the intra as well as inter frame prediction. The intra-frame prediction includes position dependent prediction filtering as an additional step. For the inter prediction, there are two new tools allowing for decoder-sided prediction refinement of B-predicted blocks, either by solving a simplified optical flow equation (BDOF) or performing a small decoder sided refinement search (DMVR). While those algorithms include entirely new processing steps for the decoder, affine prediction allows the signaling of affine motion models, which requires motion compensation on 4x4 block basis, increasing the decoder workload (processing of larger blocks can generally be optimized more efficiently). The added in-loop filters also add new complexity, and new pipeline stages.

3. EXPERIMENTAL SETUP

The numbers reported in [4], and the inferred speed-up factors are based on Intel Xeon E5-2697A v4 processors. As this project is targeting mobile platforms, all results presented in this work were obtained on a laptop with an 8-core Intel Core-i9 9980HK processor (hyper threading disabled), 32GB RAM memory and 1TB of SSD storage. The laptop was running Ubuntu 18.04 with the software decoder compiled using GCC7.

The obtained speed-up results were measured by decoding the 5 HD (Class B) and 6 UHD (Classes A1 and A2) sequences from the common test conditions test-set [17], encoded in 10bit with constant QP in random access (RA) configuration for every second QP value between 29 and 41 using the reference encoder VTM-5.2 (resulting in 7 QP points per sequence). The profiling results were acquired using the *Intel VTune* software and the *time* utility.

The speed-ups described in this paper are expressed in terms of the percentage of the wall-clock time saved *TS* when enabling a specific feature, i.e.:

$$TS = 100\% \cdot (T_{ref} - T_{test}) / T_{ref} \quad (1)$$

4. SAMPLE OPERATIONS

In signal processing applications, such as video decoders, a common speed-up measure is to implement the sample processing algorithms using SIMD operations (single instruction multiple data) [18-20]. This allows executing the same calculation (single instruction) on multiple values (multiple data) simultaneously. Modern x86 processors implement SIMD instructions on 128-bit wide (up to SSE42), 256-bit wide (AVX2) and 512-bit wide registers (AVX512). In this paper we present the speed-up results obtained by implementing sample processing operations using either the SSE42 instruction set or its super-set AVX2. No results are presented for AVX512 as the extension is not yet broadly supported in consumer CPUs.

Fig. 2 shows the speed-ups obtained by implementing parts of specific tools or decoding stages. Most notably, SIMD implementation does not help parsing and motion information derivation (Mi-Der), because those are inherently serial processes which do not rely on sample operation, but rather logical metadata processing (like coding units). Similarly, LMCS does not profit from a SIMD implementation. The application of LMCS is a sample-wise mapping through a look-up-table – an operation which cannot be efficiently implemented using SIMD.

Surprisingly, the deblocking filter implementation also achieves very little speed-up, contrary to results reported in other papers [19]. In our particular implementation, most of the time required for the deblocking filter is used for the parameter derivation and not the actual sample operations, which were sped-up very effectively.

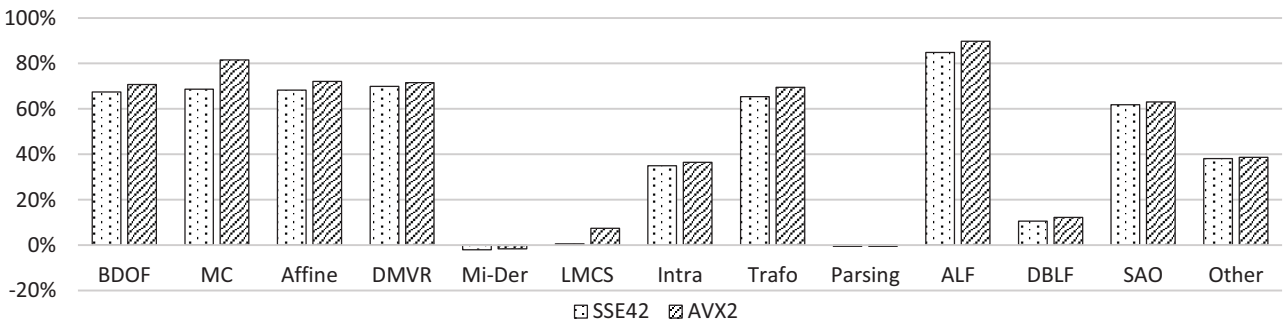

Fig. 2: Time-savings (averaged over all test sequences) obtained by implementing aspects using SSE42 and AVX2. DMVR only represents additional processing steps required by the tool, while the final prediction generation is counted as motion compensation MC. Affine describes both the affine parameter derivation and the final prediction signal generation.

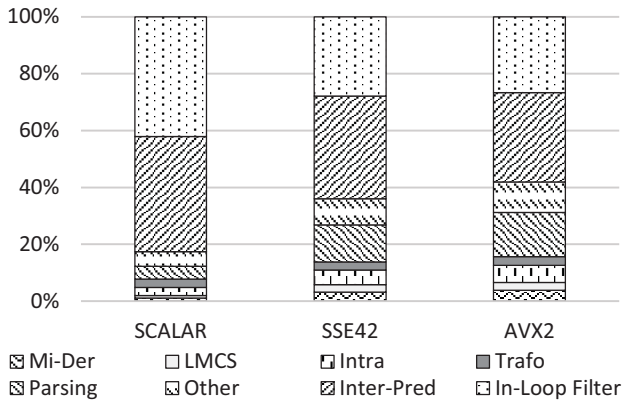

Fig. 3: Share of the run-time taken up by specific stages depending on the SIMD extension used (averaged over all test sequences).

Decoding aspects benefitting the most are motion compensation and inter-prediction tools, inverse transformation, SAO and ALF. The speed-ups are smaller than some state-of-the art [19], because the profiling measurements not only include the sample operations, but rather the timing of the first function call that can be associated with a specific processing stage, thus also containing some of the preparation logic associated with that stage.

In particular, for the inter prediction stage about 70% time saving is achieved when using SSE42. For the actual motion compensation interpolation filter (IF), more than 30% of the remaining time can be saved by enabling AVX2. Remarkably, the affine prediction can be sped-up by a similar factor as the non-affine prediction, although it is based on 4×4 samples block prediction, which is harder to efficiently optimize. To overcome this, a specific 4×4 6-tap and 4-tap IF kernel were designed (affine prediction does not sub-sample chroma sub-block size). The optimized implementation does not save the interim results in memory, but rather directly applies horizontal and vertical filtering to every read sample. Using a similar approach, with reduced interim results buffering, optimized 16×16 8-tap (luma) and 8×8 4-tap (chroma) filters were developed. DMVR splits larger blocks into 16×16 sub-blocks, making this specific block-sizes crucial for an optimized decoder. For the DMVR block sizes, only AVX2 non-buffering implementation is provided due to more space available with the larger SIMD register size.

The transformation stage, including inverse quantization, primary and secondary transformation is only SIMD optimized for the inverse quantization and primary transformation. The latter is implemented as a regular integer matrix multiplication. Exploiting the usual coefficient sparsity, it outperforms fast DCT-II implementations [6] for all but smallest blocks, with the added advantage of being generally applicable to every transformation type available in VVC.

Fig. 3 shows the share of the run-time taken up by a specific processing stage. The relative run-time share for parsing, other, and motion info derivation, increases. For further development, optimization of those non-sample based operations will be a crucial aspect.

Overall, time savings of 69% for SSE42 and 73% for AVX2 can be achieved on a modern processor.

5. PARALLELIZATION

While SIMD implementation can speed-up the actual processing of decoding stages, most modern processors provide another scalability mechanism: a CPU usually has multiple cores available which can execute entirely different code simultaneously (in contrast to SIMD scalability, which always executes the same code for a chunk of data).

The decoder is parallelized according to the decoding stages identified in Fig. 1. As each frame can be parsed independent of any other frame, it defines a first parallel task. Parsing might take more time than available to sustain a certain frame-rate. To prevent starvation of the worker threads multiple frames are parsed in parallel. This introduces a slight decoder delay, which can be configured by setting the size of the parsed frame buffer.

As soon as a parsed frame is available for reconstruction, a highly parallelized reconstruction process is started. It is divided into CTU-based and CTU-line-based tasks. To coordinate the tasks, each CTU is assigned a state according to the stage next to be executed for the CTU. After a stage has been executed, the CTU is assigned the next state which can be executed when the dependencies are resolved as described below.

- Motion derivation task can be executed for a CTU when the previous CTU in the same line and the upper-right CTU in the line above are in the next state.
- Inter prediction, residual calculation and DBLF parameter derivation can be directly executed for each CTU for which the motion information has been derived.
- Intra prediction for a CTU (and subsequent reconstruction) can be performed when the previous CTU in the same line and the upper-right CTU are in the next state.

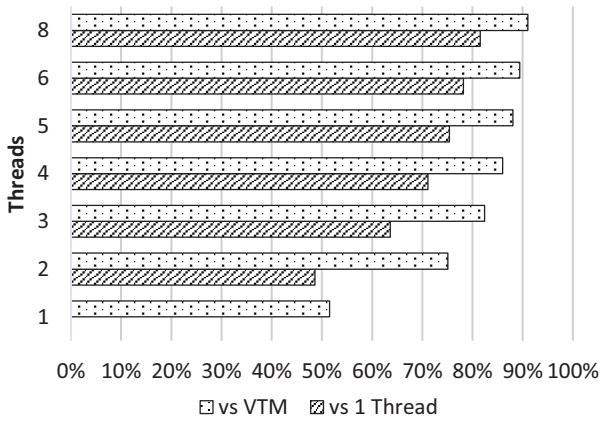

Fig. 4: Multi-threading based speed-up of the optimized decoder vs VTM and single-threaded execution (averaged over all test sequences).

- LMCS can be applied to a CTU if it is not a prediction source for a CTU still in the intra stage (i.e. the three adjacent CTUs in the line below are already finished with the intra prediction).
- Vertical-edge deblocking filter can be applied to a CTU when the CTU to the left is finished processing LMCS (as deblocking filter affects samples in the preceding CTU).
- Horizontal-edge deblocking filter can be applied when the upper, upper-right and the CTU to the right are all finished processing the vertical-edge deblocking filter.
- Due to implementation reasons, SAO requires a preparation step that can be applied to any line for which the line itself, and all CTUs in the line below it, are done processing the horizontal-edge deblocking filter.
- The actual SAO filtering can be applied to all CTUs in every line for which the preparation step has been completed.
- ALF filtering requires all eight adjacent CTUs to be finished with SAO stage processing.

For every CTU, a task worker is allocated. A thread pool scans the CTU task workers for tasks which have all dependencies resolved and executes all stages until subsequent dependencies are not satisfied. A frame is finished with the decoding after all CTUs are finished with the ALF processing stage.

Fig. 4 shows the achieved speed-ups, both vs. VTM as well as versus the optimized decoder running without parallelization. The optimized decoder already shows significant times-savings of about 50% over the reference software. This can be attributed to more optimal and complete SIMD implementations as well as reduced structural overhead.

The decoding scales fairly well, achieving 80% run-time reduction when using all 8 threads available on the test platform. While this could be more optimal, mobile platforms tend to reduce CPU clocks under sustained heavy load.

Compared to VTM, 50-90% percent of the decoding time can be reduced when using the optimized implementation – depending on the number of processing threads. Fig. 5 shows what this means in terms of achievable decoding performance measured in FPS. For HD signals, 60fps live decoding can be achieved using only 2 or 3 CPU cores. For UHD signals, 30fps decoding can be achieved for a variety of bit-rates with as little as 4 CPU cores utilized. Using all of the CPU power, up to 50fps live decoding is currently possible.

6. DISCUSSION AND OUTLOOK

The presented results already demonstrate the standard's viability for application on modern consumer grade hardware. Still, VVC shows the greatest potentials with higher resolutions [4], and was designed for use with novel immersive applications. UHD streaming at 60fps is becoming available and for the content providers to be able to utilize the benefits of VVC in such applications, the end-user needs to be able to consume such content on a variety of devices.

The described test environment is a high grade consumer device. While it is to be expected that with time high performance will be available even wider across the market, the main application of a software video decoder is to support early adopters and legacy hardware further on.

In immersive and other novel applications, even larger resolutions are possible. Also, such applications require additional computing power to support application level processing, in addition to the actual video decoding.

Considering those facts, it can be concluded that further reduction of VVC decoder implementation complexity is still required. As the standard is nearing completion, it is now down to implementers to implement the draft in the most efficient manner possible.

7. CONCLUSION

This paper presents recent advances in VVC software live decoder implementation. This work shows the viability of the new standard. We discuss parallelization potentials – both low level using SIMD operations and high level at the application level, as allowed by the pipeline stages defined by the standard.

Our preliminary implementation based on draft version 5 [1] and compatible with VTM-5.2 [4] can achieve live decoding of 10 bit HD sequences way beyond 60fps, and 10 bit UHD sequences at 30fps and more on modern consumer laptops.

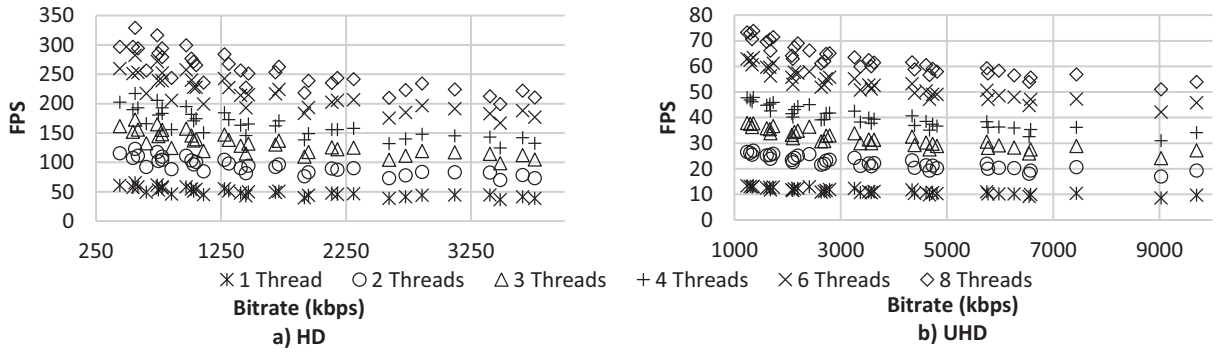

Fig. 5: Achievable decoding speed (in FPS) for a) HD and b) UHD signals depending on the stream bitrate and number of threads used for decoding.

7. REFERENCES

[1] B. Bross, J. Chen, and J. Liu, *Versatile Video Coding (Draft 5)*, document JVET-N1001, Joint Video Experts Team, Mar. 2019.

[2] J. Chen, Y. Ye, S. Kim, *Algorithm description for Versatile Video Coding and Test Model 5 (VTM 5)*, document JVET-N1002, Joint Video Experts Team, Mar. 2019.

[3] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1649–1668, 2012.

[4] F. Bossen, X. Li, A. Norkin, K. Sühring, *JVET AHG report: Test model software development (AHG3)*, document JVET-O0003, Joint Video Experts Team, Jul. 2019.

[5] W.-J. Chen et al, *JVET AHG reort: Tool reporting procedure (AHG13)*, document JVET-O0013, Jul 2019.

[6] F. Bossen, B. Bross, K. Sühring, and David Flynn, "HEVC Complexity and Implementation Analysis", IEEE Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1685–1696, 2012.

[7] A. Tissier, A. Mercat, T. Amestoy, W. Hamidouche, J. Vanne and D. Menard, "Complexity Reduction Opportunities in the Future VVC Intra Encoder," 2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP), Kuala Lumpur, Malaysia, 2019, pp. 1-6.

[8] M. Aklouf, M. Leny, F. Dufaux and M. Kieffer, "Low Complexity Versatile Video Coding (VVC) for Low Bitrate Applications," 2019 8th European Workshop on Visual Information Processing (EUVIP), Roma, Italy, 2019, pp. 22-27.

[9] H. Gao, S. Esenlik, Z. Zhao, E. Steinbach and J. Chen, "Low Complexity Decoder Side Motion Vector Refinement for VVC," 2019 Picture Coding Symposium (PCS), Ningbo, China, 2019, pp. 1-5.

[10] M. Alvarez-Mesa, C. C. Chi, B. Juurlink, V. George, and T. Schierl, "Parallel video decoding in the emerging HEVC standard", in 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Kyoto, Japan, 2012, pp. 1545-154.

[11] S. Gudumasu, S. Bandyopadhyay, A. Srivastava, and Y. He, *[AHG16] VVC software decoder and performance analysis*, document JVET-Q0211, Joint Video Experts Team, Jan. 2020.

[12] D. Marpe, G. Blättermann, G. Heising, and T. Wiegand, "Video Compression Using Context-Based Adaptive Arithmetic Coding," in 2001 IEEE International Conference on Image Processing (ICIP), Thessaloniki, Grece, 2001.

[13] B. Bross et al., "General Video Coding Technology in Responses to the Joint Call for Proposals on Video Compression With Capability Beyond HEVC," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 5, pp. 1226-1240, May 2020.

[14] J. Chen, M. Karczewicz, Y. Huang, K. Choi, J. Ohm and G. J. Sullivan, "The Joint Exploration Model (JEM) for Video Compression With Capability Beyond HEVC," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 5, pp. 1208-1225, May 2020.

[15] J. Pfaff et al., "Video Compression Using Generalized Binary Partitioning, Trellis Coded Quantization, Perceptually Optimized Encoding, and Advanced Prediction and Transform Coding," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 30, no. 5, pp. 1281-1295, May 2020.

[16] N. Sidaty, P. Cabarat, W. Hamidouche, D. Menard and O. Deforges, "Performance and Computational Complexity of the Future Video Coding," 2018 IEEE International Workshop on Signal Processing Systems (SiPS), Cape Town, 2018, pp. 31-36.

[17] 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, Joint Video Expert Team, Mar. 2019.

[18] R. B. Lee, "Accelerating multimedia with enhanced microprocessors", IEEE Micro, vol. 15, no. 2, pp. 22-32, Apr. 1995.

[19] C. C. Chi, M. Alvarez-Mesa, B. Bross, B. Juurlink, and T. Schierl, "SIMD Acceleration for HEVC Decoding," IEEE Trans. Circuits Syst. Video Technol., vol. 25, no. 5, pp. 841–855, 2015.

[20] B. Bross et al., "HEVC performance and complexity for 4K video," in 2013 IEEE Third International Conference on Consumer Electronics, Berlin (ICCE-Berlin), IFA Fairground, Berlin, Germany, Sep. 2013 - Sep. 2013, pp. 44–47.