Kvazaar 2.0: Fast and Efficient Open-Source HEVC Inter Encoder

Ari Lemmetti, Marko Viitanen, Alexandre Mercat, and Jarno Vanne
Ultra Video Group, Tampere University, Finland
{ari.lemmetti, marko.viitanen, alexandre.mercat, jarno.vanne}@tuni.fi

ABSTRACT

High Efficiency Video Coding (HEVC) is the key to economic video transmission and storage in the current multimedia applications but tackling its inherent computational complexity requires powerful video codec implementations. This paper presents Kvazaar 2.0 HEVC encoder that is the new release of our academic open-source software (github.com/ultravideo/kvazaar). Kvazaar 2.0 introduces novel inter coding functionality that is built on advanced rate-distortion optimization (RDO) scheme and speeded up with several early termination mechanisms, SIMD-optimized coding tools, and parallelization strategies. Our experimental results show that the proposed coding scheme makes Kvazaar 125 times as fast as the HEVC reference software HM on the Intel Xeon E5-2699 v4 22-core processor at the additional coding cost of only 2.4% on average. In constant quantization parameter (QP) coding, Kvazaar is also 3 times as fast as the respective preset of the well-known practical x265 HEVC encoder and is still able to attain 10.7% lower average bit rate than x265 for the same objective visual quality. These results indicate that Kvazaar has become one of the leading open-source HEVC encoders in practical high-efficiency video coding.

CCS CONCEPTS

• Computing methodologies ~ Computer graphics ~ Image compression • Computing methodologies ~ Parallel computing methodologies ~ Parallel algorithms ~ Vector / streaming algorithms

KEYWORDS

High Efficiency Video Coding (HEVC), Kvazaar HEVC encoder, open-source, rate-distortion optimization (RDO), inter coding

ACM Reference format:

Ari Lemmetti, Marko Viitanen, Alexandre Mercat, and Jarno Vanne. 2020. Kvazaar 2.0: Fast and Efficient Open-Source HEVC Inter Encoder. In *Proceedings of ACM Multimedia Systems Conference (MMSys'20). ACM, Istanbul, Turkey, 6 pages.* https://doi.org/10.1145/3339825.3394927

1 Introduction

According to Cisco, global IP video traffic will increase fourfold from 2017 and account for 82% of all IP traffic by 2022 [1]. This explosive growth is mainly driven by a proliferation of advanced multimedia devices, omnipresent connectivity, and popular video applications with immersive user experience.

The latest international video coding standard, *High Efficiency Video Coding* (*HEVC/H.265*) [2], is developed to mitigate the prevailing growth of video transmission and storage. It improves coding efficiency by almost 40% over the preceding *Advanced Video Coding* (*AVC/H.264*) [3] standard for the same objective visual quality but with around 40% complexity overhead [4]. Therefore, fostering HEVC deployment worldwide calls for open and extremely powerful implementations which are able to tackle the HEVC complexity and attain appealing coding speed, coding efficiency, and power budget.

Currently, there exist a couple of noteworthy open-source HEVC encoders [5]-[9] out of which only *HEVC reference software model* (*HM*) [5], x265 [6], and our Kvazaar [7] are under active academic research and development. HM supports all normative HEVC coding tools and is able to achieve the best coding efficiency among these encoders. However, huge computational complexity restricts its usage to research and conformance testing rather than practical encoding, despite numerous optimizations made to the original implementation [10]-[12]. The commercially funded x265 is probably the most well-known practical open-source HEVC encoder at the moment so it is used as a main reference for our work.

Kvazaar is an academic open-source HEVC software encoder developed by our Ultra Video Group at Tampere University. It is available online on GitHub at

https://github.com/ultravideo/kvazaar

under GNU LGPLv2.1 license. Our previous works [13]-[16] have thoroughly considered the first-generation version of Kvazaar that was shown to be the academic frontrunner in HEVC intra coding. This paper introduces the second-generation implementation of Kvazaar, a.k.a. Kvazaar 2.0, that has particularly been optimized for HEVC inter coding. Here, the main focus is on Kvazaar *veryslow* preset [7] that seeks high *rate-distortion* (*RD*) performance under *Random Access* (*RA*) coding configuration but without letting the coding time get out of hands. Special attention is paid to the overall software encoder architecture that implements

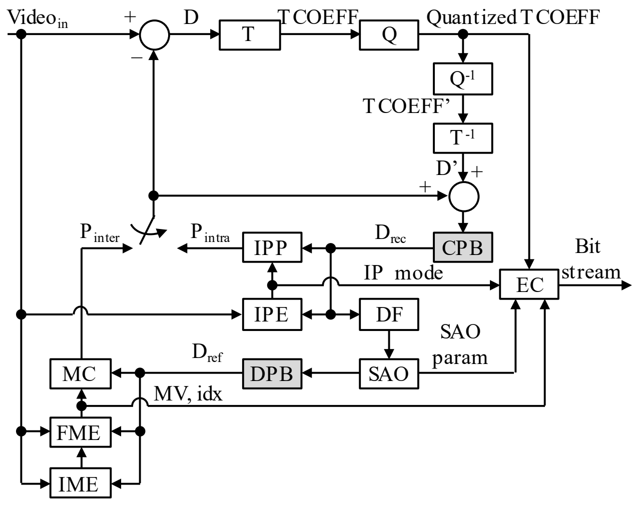

Figure 1: Block diagram of the Kvazaar HEVC encoder.

Table 1: Main coding options of the Kvazaar HEVC encoder

Feature	Available in Kvazaar	Experimental Setup
Profile	Main	
Internal bit depth	8, 10 (experimental)	8
Color format	4:2:0, 4:0:0	4:2:0
Presets	10 (ultrafast...placebo)	veryslow
Coding configuration	AI, LP, LB, RA	RA
GOP	Open, Closed	Open
GOP structure	Configurable	Hierarchical 16
Parallelization	Tiles, Slices, WPP, OWF	WPP, OWF
Slice type	I, P, B	
CU sizes	64, 32, 16, 8	
Motion partitions	SMP, AMP	SMP
Transform depth	Implicit and Intra 4	Implicit
Reference frames	1-15	Variable (1-5)
Sub-pixel MVs	¼-pel luma	
Merge candidates	5	
Temporal MVPs	L0	
Loop filters	Deblocking, SAO	
Additional	Biprediction, Transform skip, Sign bit hiding	

multithreaded RA coding scheme with RD-optimized mode decision, coding tools with *Advanced Vector Extensions 2* (*AVX2*) optimizations, and HEVC parallelization strategies.

The rest of this paper is organized as follows. Section 2 provides an overview of the Kvazaar 2.0 HEVC encoder architecture by particularly focusing on its RD-optimized inter coding scheme and related coding tools. Section 3 benchmarks the coding speed and efficiency of Kvazaar over those of x265 and HM in RA coding. Finally, Section 4 concludes the paper.

2 Kvazaar 2.0 HEVC Encoder

Fig. 1. depicts an overview of the Kvazaar 2.0 architecture. It supports HEVC Main profile [17] with ten presets (*ultrafast…placebo*) out of which *veryslow* preset with 8-bit 4:2:0 progressive videos is addressed in this proposal. Table 1 lists all essential coding parameters available in Kvazaar and a subset of parameters benchmarked in this work.

2.1 Rate-Distortion Optimization (RDO)

Kvazaar supports a multi-stage *rate-distortion optimization* (*RDO*) [18] scheme with *approximate* (fast) and *accurate* (complex) RDO stages. These stages minimize the overall coding cost by comparing the RD costs of each explored coding candidate prior to making a decision about the block structure and mode parameters.

The approximate RDO stage compares costs based on *Sum of Absolute Transformed Differences* (*SATD*) of the luma channel and associated bits of the coding mode. Typically, this stage is used to explore many different alternatives to choose from, and forward only the best ones to the accurate RDO stage.

The accurate RDO stage takes into account both luma and chroma distortion as well as residual quadtree bits. It uses *Sum of Squared Differences* (*SSD*) as distortion metric and reconstructs the predicted block partition before cost calculation and comparison.

2.2 Coding Tree Structure

Kvazaar implements a complete HEVC quadtree coding structure [19] in which each input frame is split into equally-sized square blocks called *Coding Tree Units* (*CTUs*). In Kvazaar, the size of a CTU is 64 × 64 pixels. In the 4:2:0 color format, a CTU contains one luma block, two chroma blocks, and associated syntax elements.

CTUs are recursively divided recursively into four equal-sized square *Coding Units* (*CUs*) until the maximum hierarchical depth of the quadtree is reached. The size of the luma blocks is specified as $2N\times2N$ and the 4:2:0 color format limits the size of the chroma blocks to $N\times N$, where $N \in \{32, 16, 8, 4\}$. This quadtree decomposition is determined with a depth-first search strategy in z-scanning order. During the search, accurate RD costs are assigned to all evaluated CUs and a split decision is made if it yields lower RD cost than a non-split alternative.

The complexity of the recursive search is reduced with the following two early termination mechanisms: 1) further search for sub-CUs in the quadtree can be bypassed after the accumulated cost of the computed sub-CUs exceeds that of the parent CU as evaluating the remaining sub-CUs has no effect on the split decision anymore; and 2) recursive search for further CU splits can be stopped after finding a CU without any non-zero coefficients since continuing the search is unlikely to yield any significant RD improvement. The latter mechanism is derived from [20].

2.3 Prediction Mode Decision

In HEVC, a CU defines a picture region that shares the same prediction mode that can be either *Intra*, *Inter,* or *Skip*. The Skip mode can be considered a special case of *Merge mode* in inter prediction.

CUs can be further split into rectangular-shaped *Prediction Units* (*PUs*) that share the identical prediction information. For a CU of size $2N\times2N$, Kvazaar supports the following partition modes with the PUs: two square partitions of size $2N\times 2N$ and $N\times N$, two *Symmetric Motion Partition* (*SMP*) modes of size $2N\times N$ and $N\times 2N$, and four *Asymmetric Motion Partition* (*AMP*) modes of size $2N\times nU$, $2N\times nD$, $nL\times 2N$, and $nR\times 2N$ [19], out of which $N\times N$ partition is only eligible for Intra in Kvazaar. The Skip CU is always coded with the $2N\times 2N$ partition.

For each explored CU node in the quadtree, Kvazaar evaluates the prediction and partition modes in the following order: 1) square Inter modes; 2) SMP and AMP Inter modes; and 3) Intra mode. The best prediction mode and partition are selected by the accurate RDO at this stage.

During the evaluation of a single CU, the complexity of the search is reduced with an aggressive early Skip mode decision. Compared with other corresponding complexity reduction techniques such as [21], the early Skip mode decision is already evaluated after a subset of the $2N\times 2N$ Inter mode search. If a Skip decision is taken, the CU is immediately set without considering any other modes at the current depth. As a consequence, the Skip CU also triggers the early termination of the recursive CU search due to zero residual.

2.4 Inter Prediction

In the inter prediction stage, motion parameters for merge mode are inherited implicitly from neighboring PUs. A list of merge candidates is constructed and all available candidates are used to generate the respective luma prediction blocks [22]. Each merge candidate undergoes the approximate RDO process to find the best merge mode parameters.

Early Skip mode decision is tried after merge candidate analysis. In Kvazaar, the best merge candidate is predicted, residual transformed, and quantized to find any coded residual. Luma and chroma channels are handled separately and the chroma is reconstructed only if luma has zero residual. If the best merge candidate has zero residual, Skip mode is selected, i.e., early Skip mode decision is taken and no other search is performed to find Inter or any other modes at this depth.

To find the optimal *Advanced Motion Vector Prediction* (*AMVP*) mode, motion parameters for Inter PUs are obtained with *motion estimation* (*ME*). It starts with *integer ME* (*IME*) that refers to previously coded reference frames. First, ME analyzes unipredicted blocks and then bipredicted blocks. The unipredicted blocks are explored with a *Test Zone* (*TZ*) search whose starting point may be (0, 0) vector, a merge candidate, or a scaled MV of a collocated reference block.

After IME, *Motion Vectors* (*MVs*) are refined by *fractional ME* (*FME*) that operates at sub-pixel accuracy by interpolating and sampling values between the pixels. To limit computational complexity, RD costs are only computed for the nine half-pixel positions that are closest to the best integer MV at this stage. Then, the same is repeated around the best half-pixel position with the most adjacent quarter-pixel positions. No separate motion estimation or refining is performed to find optimal bipredicted blocks. The two best motion vectors of the unipredictions, one for each of the reference picture lists (*L0* and *L1*), are chosen to generate the bipredicted block that is averaged from the samples determined by the motion vectors. The best MV(s) and respective reference frame indexes (*idxs*) are finally output to *motion compensation* (*MC*). It produces inter prediction (P_{inter}) by using MVs and idxs to access pixels from a *decoded picture buffer* (*DPB*) which contains the previously *reconstructed reference pictures* (*Dref*).

IME and FME are performed using the approximate RDO with the exception that IME uses the *Sum of Absolute Differences* (*SAD*) in block matching, whereas the distortion metric in FME is SATD. At the end of the approximate RD cost estimation, the best alternative between the best merge mode and AMVP mode acquired by the motion estimation is selected. After all the PUs depending on the selected motion partition are parameterized, the accurate RD cost for the respective Inter CU is calculated.

2.5 Intra Prediction

The intra prediction stage is split into *Intra Picture Estimation* (*IPE*) and *Intra Picture Prediction* (*IPP*). IPE supports all the 35 Intra modes of HEVC: DC, planar, and the 33 angular modes [23]. First, it computes approximate RD costs for planar, DC, and the angular modes traversed by logarithmic search. Next, the accurate RDO process is performed for a shortlist of the best modes.

IPE outputs the best mode to the IPP that computes the *intra prediction* (P_{intra}) for the current PU. P_{intra} is computed by accessing the *Current Picture Buffer* (*CPB*) that contains previously reconstructed blocks (D_{rec}) of the current picture. The Kvazaar intra coding process is detailed in [15].

2.6 Transform and Quantization

After the prediction stage, a *prediction residual* (*D*) is computed by subtracting P_{intra} or P_{inter} from the original CU samples. The *transform* (*T*) stage transforms *D* from the spatial domain into *transform domain coefficients* (*TCOEFFs*) [24]. Kvazaar supports *transform units* (*TUs*) of size 32×32, 16×16, and 8×8. It implements integer *Discrete Sine Transform* (*DST*) for intra coded 4×4 luma blocks and integer *Discrete Cosine Transform* (*DCT*) for the other blocks. In Kvazaar, TU size of square Inter mode matches the CU size unless a split is implied. TUs are always split once for SMP/AMP blocks to adapt to possible discontinuities at the motion partition boundary.

The *quantization* (*Q*) stage maps TCOEFFs into *Quantized TCOEFFs*. Kvazaar also implements a non-normative *Rate-Distortion Optimized Quantization* (*RDOQ*) [25]. It improves coding efficiency by introducing small amounts of error to Quantized TCOEFFs if it results in better RD trade-off.

The decoding path of Kvazaar includes *Inverse Quantization* (Q^{-1}) and *Inverse Transform* (T^{-1}) stages to dequantize and convert *Quantized TCOEFFs* back to spatial domain (*D'*). The reconstructed block D_{rec} is then yielded by adding P_{inter} or P_{intra} to *D'* and the result is stored into CPB.

Table 2: Encoding parameters of Kvazaar, HM, and x265

Encoder	Options
Kvazaar 2.0	-q (qp) --input-res=(width)x(height) --input-bitdepth=8 --input-fps=(fps) --seek 0 -n (frames) -i (input) -o (output) --preset=veryslow --period=256 --threads=auto --no-info --hash=checksum
HM16.20	-c encoder_randomaccess_main.cfg -q (qp) -f (frames) -fs 0 -i (input) -wdt (width) -hgt (height) -b (output) --InputBitDepth=8 --FrameRate=(fps) --ReconFile=nul --SEIDecodedPictureHash=3 --ConformanceWindowMode=1
x265 v3.2 RC1	--preset=veryslow --tune=psnr --psnr --no-info --hash=3 -q (qp) -f (frames) --input (input) --input-res (width)x(height) --input-depth 8 --output-depth 8 --fps (fps) -o (output) --no-progress

Additionally, Kvazaar improves transform and quantization efficiencies by implementing *transform skip* and *sign bit hiding* coding tools specified by the standard.

2.7 Loop Filtering

Kvazaar implements two sequential in-loop filters: *Deblocking Filter* (*DF*) [26] and *Sample-Adaptive Offset* (*SAO*) [27], which filter the distortion and visible block edges in order to improve the quality of reconstructed samples before they are stored in DPB. DF and SAO are applied at the CTU level.

2.8 Entropy Coding

Entropy Coding (*EC*) compresses binarized syntax elements such as quantized TCOEFFs, IP mode, MV, idx, SAO param, and others by *Context-Adaptive Binary Arithmetic Coding* (*CABAC*) [28].

Kvazaar supports two methods for coding cost estimation. The first method performs CABAC for TCOEFFs and related elements. The second one is a table-based estimation adopted from HM. It estimates fractional bits according to the current context state. To limit coding time with acceptable RD loss, Kvazaar updates contexts primarily at the CTU level during the actual bitstream generation.

2.9 Parallel Processing

In addition to slice and tile mechanisms [14], Kvazaar implements *Wavefront Parallel Processing* (*WPP*) [29] and *Overlapped Wavefront* (*OWF*) [30] parallelization strategies for efficient multithreading. When WPP is enabled, multiple CTU rows of a frame can be encoded in parallel. Processing of a CTU row can be started once the first two CTUs of the previous row have been encoded. To improve the ramping inefficiencies of WPP, the non-normative OWF technique overlaps the execution of consecutive frames [30]. When a CTU row is encoded and no more unprocessed rows are available in the current frame, a new CTU row can be taken from the next frame. Kvazaar only starts a WPP row from the next frames if there are some reconstructed reference pixels available for motion estimation in all directions. Kvazaar automatically sets the threads and OWF values for maximum CPU utilization.

At the data level, the most time-consuming coding tools, such as SAD, SATD, SSD, interpolation filters, IPP, and transforms, are optimized using Intel Intrinsics or x86-64 assembly language. Kvazaar includes a dynamic dispatch mechanism for SIMD-optimized functions, i.e., it automatically selects the best SIMD optimizations according to the instructions sets supported by the CPU. Optimizations for SSE2, SSE4.1, AVX, and AVX2 are supported, but most of them are for AVX2 [31].

Table 3: Major differences in the configurations

Enabled Features	Kvazaar	x265	HM
Max I-frame period	256	250	32
GOP	16	Adaptive	16
AMP	No	Yes	Yes
Transform depth	Implicit only	3	3
Scenecut detection	No	Yes	No
Lookahead	No	Yes	No

Table 4: Test Environment

Processor	Intel Xeon E5-2699 v4 (22 × 2.2Ghz)
Memory	64 GB DDR4-2400 (4 × 16)
Operating system	Microsoft Windows 10
Compiler	MS Visual Studio 2019

3 Rate-Distortion-Complexity Analysis

In our experiments, the *veryslow* preset of the proposed Kvazaar v2.0.0 (see experimental setup in Table 1) was benchmarked under RA coding configuration against the default configuration of HM16.20 [5] and the *veryslow* preset of x265 v3.2 RC1 [6] tuned for PSNR. The respective command line options are listed in Table 2.

The evaluated encoder configurations of HM, x265, and Kvazaar converge to a large extent, but there are six notable differences listed in Table 3.

- HM uses an intra frame period of 32 by default which is only a fraction of that of Kvazaar and x265.
- Kvazaar adopts a hierarchical *group of picture* (*GOP*) structure of 16 frames from HM [5], whereas x265 uses adaptive GOP structure.
- AMPs are disabled in Kvazaar only.
- HM and x265 utilize complete splitting of the residual quadtree whereas Kvazaar only splits a TU once for the implicit splits.
- Only x265 implements scene cut detection.
- Only x265 takes advantage of lookahead of upcoming frames.

3.1 Experimental Setup

The experiments were conducted on an Intel Xeon E5-2699 v4 22-core processor detailed in Table 4. It supports SIMD extensions up to AVX2 instruction set.

All eighteen 8-bit HEVC common test sequences [17] from classes *A* to *E* were executed. They differ in terms of number of frames, frame rate, and spatial resolution.

The coding efficiencies of the evaluated encoders were compared in terms of average *Bjøntegaard Delta Bit Rate* (*BD-BR*) [32] using four base *Quantization Parameter* (*QP*) values: 22,

Table 5: Coding speed and efficiency of Kvazaar 2.0 over x265 and HM (smaller BD-BR is better).

					Kvazaar 2.0 vs. HM			Kvazaar 2.0 vs. x265		FPS	
Class	Format	Sequences	# of frames	Frame rate	BD-BR	Speedup 1 thread	Speedup multithread	BD-BR	Speedup multithread	x265	Kvz
A	2560×1600 (1600p)	PeopleOnStreet	150	30 fps	11.9%	3.9×	182.9×	-0.4%	2.5×	0.9	2.4
		Traffic	150	30 fps	-7.7%	8.8×	328.5×	-12.5%	3.8×	1.8	7.3
B	1920×1080 (1080p)	BasketballDrive	500	50 fps	17.9%	5.5×	171.1×	-3.6%	3.0×	1.7	5.3
		BQTerrace	600	60 fps	-5.1%	11.2×	202.3×	-11.4%	3.3×	2.5	8.7
		Cactus	500	50 fps	0.6%	6.3×	166.9×	-7.8%	2.9×	2.1	6.2
		Kimono	240	24 fps	16.6%	5.6×	178.0×	-5.0%	2.8×	2.1	6.1
		ParkScene	240	24 fps	9.2%	6.8×	166.6×	-13.9%	3.1×	2.1	6.6
C	832×480 (WVGA)	BasketballDrill	500	50 fps	-0.4%	4.2×	58.4×	-21.2%	2.9×	3.3	9.7
		BQMall	600	60 fps	12.9%	5.1×	61.6×	-11.7%	3.3×	3.5	11.6
		PartyScene	500	50 fps	6.5%	4.4×	50.7×	-21.0%	3.4×	2.5	8.4
		RaceHorses	300	30 fps	18.9%	3.5×	71.5×	6.8%	3.5×	2.5	8.9
D	416×240 (WQVGA)	BasketballPass	500	50 fps	15.0%	3.6×	17.0×	-3.3%	1.9×	5.4	10.6
		BlowingBubbles	500	50 fps	9.8%	4.8×	21.1×	-17.3%	2.9×	5.6	17.0
		BQSquare	600	60 fps	-3.4%	9.0×	39.3×	-20.5%	4.7×	6.8	35.7
		RaceHorses	300	30 fps	20.4%	3.3×	20.6×	3.7%	2.2×	5.0	11.4
E	1280×720 (720p)	FourPeople	600	60 fps	-27.5%	12.9×	131.6×	-20.5%	2.4×	6.1	14.9
		Johnny	600	60 fps	-30.2%	21.5×	218.4×	-16.4%	3.3×	7.4	26.2
		KristenAndSara	600	60 fps	-22.0%	14.5×	164.5×	-17.4%	2.9×	6.1	18.3
				Average	**2.4%**	**7.5×**	**125.0×**	**-10.7%**	**3.0×**		

27, 32, and 37 with *Peak Signal-to-Noise Ratio* (*PSNR*) as the image quality metric. All tests were run with a constant QP coding scheme where individual frames may have different QP offsets depending on the GOP structures. The encoders were configured to output checksum hashes in the bitstream. All custom *Supplemental Enhancement Information* (*SEI*) messages were omitted for a fair BD-BR comparison.

The coding speeds of the encoders were measured separately for each four QPs and their average was reported. For a reliable comparison, only one encoder instance was run at a time. The benchmarked encoders were allowed to utilize all optimizations, instruction set extensions, and multithreading options enabled by default. Apart from that, the coding speed of a single-threaded Kvazaar configuration was measured without any SIMD optimizations for a more straightforward comparison against HM that is inherently single-threaded and less extensively SIMD-optimized.

3.2 Experimental Results

Table 5 reports the RD characteristics and coding speed of Kvazaar v2.0.0 over HM16.20 and x265 v3.2 RC1.

According to our results, both practical encoders, Kvazaar and x265, suffer from inflated average BD-BR of +2.4% and +14.6% over HM, respectively. However, HM lags far behind in coding speed. Single-threaded scalar Kvazaar is 7.5× as fast as HM on average and the gap increases up to 125× on our 22-core processor with vectorization enabled. The speedup obtained with multithreading is directly proportional to the resolution, and especially to the number of CTU rows. Kvazaar is 256× and 177× as fast as HM on average for the sequences in classes *A* and *B*, respectively.

Furthermore, Kvazaar is 3× times as fast as x265 with a BD-BR improvement of -10.7% on average. In contrast to the fluctuating BD-BR, the speedup of Kvazaar is stable over x265 across the whole test set. Both practical encoders perform frame-level parallel processing and WPP to maximize the CPU utilization and spawn up to 44 threads, one thread per logical core.

These results show that Kvazaar is currently a very competitive encoder in terms of rate-distortion-complexity tradeoff in high-efficiency coding, particularly when speed aspect is taken into account. Using average coding speeds with Class B sequences as an example, encoding a two-hour 1080p60 movie on our 22-core processor (see Table 4) would approximately take 147 days with HM, more than 2 days with x265, and only 18 hours with Kvazaar.

4 Conclusion

This paper introduced Kvazaar 2.0 HEVC encoder that represents the second generation of our academic open-source software. It is available online at github.com/ultravideo/kvazaar. Kvazaar 2.0 is particularly designed and optimized for RD-optimized HEVC inter coding. It includes a novel scheme for efficient and high-speed prediction mode decision with optimized inter coding tools. In constant QP coding, the proposed parallel implementation of Kvazaar is 3× as fast as the respective *veryslow* preset of the x265 encoder with average BD-BR improvement of -10.7%. The first-generation implementation of Kvazaar was already shown to be the frontrunner in HEVC intra coding and the results of this work justify that Kvazaar 2.0 is currently one of the leading solutions among all practical open-source HEVC encoders.

Thanks to Kvazaar 2.0, the entire multimedia community can hereafter deploy an academic, cutting-edge alternative for HEVC inter coding in various multimedia applications. Kvazaar is already supported by the popular FFmpeg and Libav multimedia frameworks that facilitate the deployment of Kvazaar with other multimedia processing tools.

ACKNOWLEDGMENTS
This work was supported in part by the European ECSEL project PRYSTINE (under the grant agreement 783190) and the Academy of Finland (decision no. 301820).

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

[2] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, “Overview of the High Efficiency Video Coding (HEVC) standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1649-1668.

[3] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, “Overview of the H.264/AVC video coding standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, July 2003, pp. 560-576.

[4] J. Vanne, M. Viitanen, T. D. Hämäläinen, and A. Hallapuro, “Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1885-1898.

[5] JCT-VC. *HEVC reference software*. [Online]. Available: https://hevc.hhi.fraunhofer.de/. Accessed: Apr. 14, 2020.

[6] MulticoreWare Inc. *x265 HEVC Encoder / H.265 Video Codec*. [Online]. Available: https://bitbucket.org/multicoreware/x265/downloads/. Accessed: Apr. 14, 2020.

[7] Ultra Video Group. *Kvazaar open-source HEVC Encoder*. [Online]. Available: https://github.com/ultravideo/kvazaar. Accessed: Apr. 14, 2020.

[8] Turingcodec.org. *Turing codec*. [Online]. Available: http://turingcodec.org/. Accessed: Apr. 14, 2020.

[9] 01.org. *Scalable Video Technology for HEVC Encoder (SVT-HEVC Encoder).* [Online]. Available: https://github.com/OpenVisualCloud/SVT-HEVC. Accessed: Apr. 14, 2020.

[10] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, “Complexity control of high efficiency video encoders for power-constrained devices,” *IEEE Trans. Consum. Electron.*, vol. 57, no. 4, Nov. 2011, pp. 1866-1874.

[11] Y. Li, Z. Liu, X. Ji, and D. Wang, “CNN based CU partition mode decision algorithm for HEVC inter coding,” *in Proc. IEEE Int. Conf. Image Processing*, Athens, Greece, Oct. 2018, pp. 993-997.

[12] Y. Zhang, G. Wang, R. Tian, M. Xu, and C. C. J. Kuo, “Texture-classification accelerated CNN scheme for fast intra CU partition in HEVC,” *in Proc. Data Compression Conf.*, Snowbird, Utah, USA, Mar. 2019, pp. 241-249.

[13] M. Viitanen, A. Koivula, A. Lemmetti, J. Vanne, and T. D. Hämäläinen, “Kvazaar HEVC encoder for efficient intra coding,” *in Proc. IEEE Int. Symp. Circuits Syst.*, Lisbon, Portugal, May 2015, pp. 1662-1665.

[14] A. Koivula, M. Viitanen, J. Vanne, T. D. Hämäläinen, and L. Fasnacht, “Parallelization of Kvazaar HEVC intra encoder for multi-core processors,” *in Proc. IEEE Workshop Signal Process. Syst.*, Hangzhou, China, Oct. 2015, pp. 1-6.

[15] M. Viitanen, A. Koivula, A. Lemmetti, A. Ylä-Outinen, J. Vanne, and T. D. Hämäläinen, “Kvazaar: open-source HEVC/H.265 encoder,” *in Proc. ACM Int. Conf. Multimedia*, Amsterdam, The Netherlands, Oct. 2016, pp. 1179-1182.

[16] A. Mercat, A. Lemmetti, M. Viitanen, and J. Vanne, “Acceleration of Kvazaar HEVC intra encoder with machine learning,” *in Proc. IEEE Int. Conf. Image Processing*, Taipei, Taiwan, Sep. 2019, pp. 2676-2680.

[17] F. Bossen, “Common test conditions and software reference configurations,” *document JCTVC-L1100*, Geneva, Switzerland, Jan. 2013.

[18] G. J. Sullivan and T. Wiegand, “Rate-distortion optimization for video compression,” *IEEE Signal Processing Mag.*, vol. 15, no. 6, Nov. 1998, pp. 74-90.

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

[20] R. H. Gweon and Y. L. Lee, “Early termination of CU encoding to reduce HEVC complexity,” *IEICE Trans. Fundam. Electro., Commun. and Comput Sci.*, vol. 95, no 7, Sep. 2012, pp. 1215-1218.

[21] J. Kim, J. Yang, K. Won, and B. Jeon, “Early determination of mode decision for HEVC,” *in Proc. Picture Coding Symp.*, Krakow, May 2012, pp. 449-452.

[22] P. Helle et al., “Block merging for quadtree-based partitioning in HEVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1720-1731.

[23] J. Lainema, F. Bossen, W. Han, J. Min, and K. Ugur, “Intra coding of the HEVC standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1792-1801.

[24] M. Budagavi, A. Fuldseth, G. Bjøntegaard, V. Sze, and M. Sadafale, “Core transform design in the High Efficiency Video Coding (HEVC) standard,” *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 6, Dec. 2013, pp. 1029-1041.

[25] J. Stankowski, C. Korzeniewski, M. Domański, and T. Grajek, “Rate-distortion optimized quantization in HEVC: Performance limitations,” *in Proc. Picture Coding Symp.*, Cairns, Queensland, Australia, May-June 2015, pp. 85-89.

[26] A. Norkin et al., “HEVC deblocking filter,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1746-1754.

[27] C. Fu et al., “Sample adaptive offset in the HEVC standard,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1755-1764.

[28] V. Sze and M. Budagavi, “High throughput CABAC entropy coding in HEVC,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1778-1791.

[29] G. Clare, F Henry, and S. Pateux, “Wavefront parallel processing for HEVC encoding and decoding,” *document JCTVC-F274*, Torino, Italy, July 2011.

[30] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, V. George, and T. Schierl, “Improving the parallelization efficiency of HEVC decoding,” *in Proc. IEEE Int. Conf. Image Processing*, Orlando, Florida, USA, Sept.-Oct. 2012, pp. 213-216.

[31] A. Lemmetti, A. Koivula, M. Viitanen, J. Vanne, and T. D. Hämäläinen, “AVX2–optimized Kvazaar HEVC intra encoder,” *in Proc. IEEE Int. Conf. Image Processing*, Phoenix, Arizona, USA, Sep. 2016.

[32] G. Bjøntegaard, “Calculation of average PSNR differences between RD-curves,” *document VCEG-M33*, Austin, Texas, USA, Apr. 2001.