Kvazaar HEVC Encoder for Efficient Intra Coding

Marko Viitanen, Ari Koivula, Ari Lemmetti, Jarno Vanne, and Timo D. Hämäläinen
Department of Pervasive Computing, Tampere University of Technology, Finland
{marko.viitanen, ari.koivula, ari.lemmetti, jarno.vanne, timo.d.hamalainen}@tut.fi

Abstract— **This paper presents an open-source Kvazaar encoder for HEVC intra coding. This academic software encoder has been developed from the scratch using *C* as an implementation language by prioritizing modularity, portability, and readability of the source code. Kvazaar implements almost the same intra coding functionality as HEVC reference encoder (HM) but its rewritten source code makes it significantly faster. In all-intra (AI) coding, a single-threaded *C* implementation of Kvazaar is 2.3 times faster than HM at a cost of 1.7% bit rate increase. The respective values with a high speed preset of Kvazaar are 10.6 and 8.8%. Compared to a single-threaded C++ implementation of x265, Kvazaar improves rate-distortion performance and increases encoding speed in both high-quality and high-speed test cases. Kvazaar has a particular edge in the high-speed test case where it almost halves the BD-rate loss and more than doubles the performance.**

Keywords—HEVC; Kvazaar HEVC encoder; intra coding; open-source implementation; rate-distortion-complexity

I. Introduction

The emerging international video coding standard *HEVC* (*High Efficiency Video Coding*) [1], [2] is the latest landmark in video compression. It has been developed by *Joint Collaborative Team on Video Coding* (*JCT-VC*) as a joint activity of *ITU-T Video Coding Experts Group* (*VCEG*) and *ISO/IEC Moving Picture Experts Group* (*MPEG*). HEVC is published as twin text by ITU, ISO and IEC as ITU-T H.265 | ISO/IEC 23008-2. The first edition of HEVC was finalized in January 2013 with three profiles: *Main*, *Main Still Picture*, and *Main 10*. This paper focuses on Main Profile under *all-intra* (*AI*) coding [3] configuration.

HEVC reduces the bit rate around 40% over the preceding state-of-the-art standard AVC [4] with the same objective quality but at about 1.4× encoding complexity when all coding tools are enabled [5]. The respective *rate-distortion-complexity* (*RDC*) values in the AI coding configuration are 23% and 3.2×. Therefore, efficient encoder implementations are crucial for the wide adoption of HEVC.

Up to this date, several HEVC encoders have been released but most of them are commercial products whose features and operating principles are kept confidential. Therefore, only open-source encoders are considered in this paper.

Among the existing open-source HEVC encoders [6]-[11], only *HEVC test model* (*HM*) [6], x265 [7], f265 [8], and our Kvazaar HEVC encoder [9] are under active development. HM as an HEVC reference codec is able to achieve the best coding efficiency among the existing HEVC encoders, but its object-based *C*++ implementation results in poor performance. Hence, it is targeted for research and conformance testing rather than practical encoding. The commercially funded x265 is the most well-known practical open-source HEVC encoder. It is based on HM C++ source code which has been enhanced by extensive assembly optimizations, multithreading, and techniques from the open-source x264 encoder. f265 is another industrial HEVC encoder. It is implemented in *C* with assembly optimizations. Although the source codes for these two commercially led projects are under open-source licenses, contributors to these projects must sign an agreement giving the companies copyright to their work. Requiring such agreements leaves room for non-commercial projects, like Kvazaar, that do not require signing separate agreements to participate. Kvazaar is an academic open-source HEVC encoder initiated and coordinated by us [12]. It is licensed under GNU GPLv2 license.

This is the first paper that focuses on Kvazaar by describing its intra coding functionality in detail. Kvazaar uses a reverse design approach compared with x265: it has been developed from the scratch using HM primarily as a reference for its encoding scheme and individual algorithm implementations. In addition, Kvazaar is developed in *C*. This more hardware-oriented approach eases source code acceleration, portability, and parallelization.

The remainder of this paper is organized as follows. Section II gives rationale for our Kvazaar HEVC encoder development. The intra coding scheme in Kvazaar is presented in detail in Section III. Section IV compares the RDC characteristics of Kvazaar with those of HM and x265 in AI coding configuration. Section V concludes the paper.

II. Kvazaar HEVC Encoder

Kvazaar HEVC encoder is being developed in an academic open-source project towards the following primary goals:

1) Coding efficiency close to HM
2) Modular structure to ease its parallelization and portability
3) Excellent source code readability and documentation

To obtain these objectives with the highest possible encoding speed and with minimal computation/memory resources, Kvazaar is developed from the scratch with *C* language. This unique approach uses the object-based C++ implementation of HM as a reference for its encoding scheme and individual algorithm implementations, but it adopts completely new data and function call tree structures.

The source codes and issue tracker for Kvazaar can be found on its GitHub page [9]. The supported platforms are x86, x64, and PowerPC on Windows, Linux, and Mac.

III. Intra Coding in Kvazaar

Kvazaar supports HEVC Main profile for AI coding of 8-bit video with 4:2:0 chroma sampling. The latest version of Kvazaar includes two presets: 1) RD1 for high-speed encoding; and 2) RD2 for high-quality encoding. The parameters of these presets are detailed in Table I.

Kvazaar implements a complete HEVC block partitioning structure [13] in which the pictures are partitioned into *coding tree units* (*CTUs*). In 4:2:0 color format, each CTU consists of one square *coding tree block* (*CTB*) of luma samples, two quarter CTBs of chroma samples, and associated syntax elements. In Kvazaar, the size of the luma CTB can be defined as $2N_{\text{MAX}} \times 2N_{\text{MAX}}$, where $N_{\text{MAX}} \in \{8, 16, 32\}$.

As specified by HEVC, the implemented coding tree divides CTBs into smaller *coding blocks* (*CBs*) according to a quadtree structure. A CTU represents a root node (depth 0) of the quadtree. At depth 1, CTBs of a CTU can be optionally divided into four equal-sized square CBs. The division can be recursively continued until the maximum hierarchical depth (h_{MAX}) of the quadtree is reached.

The leaf nodes of the quadtree are called *coding units* (*CUs*). For each CU, the size of its luma CB can be defined as $2N \times 2N$, where $N \leq N_{\text{MAX}}$ and $N \in \{4, 8, 16, 32\}$, so $N_{\text{MIN}} = 4$ and $h_{\text{MAX}} = 4$. Respectively, the 4:2:0 color format limits the sizes of the chroma CBs to $N \times N$.

Kvazaar encodes pictures at CU level according to the block diagram depicted in Fig. 1. The CUs of the coding tree are processed depth-first with a greedy search algorithm and the search is stopped early if a block has no residual after quantization. The $N \times N$ block size is treated as an additional level in the prediction tree. Full reconstruction is done for both luma and chroma to compare the non-split CU to its split counterpart. Each CU acts as a root for trees of *prediction units* (*PUs*) and *transform units* (*TUs*).

A. Intra Prediction

For a CU of size 2N × 2N, Kvazaar intra picture estimation (IPE) supports luma prediction blocks (PBs) of size $2N \times 2N$. In addition, it supports luma PBs of size $N \times N$ when $N = 4$. IPE implements all 35 *intra prediction* (*IP*) modes of HEVC (DC, planar, and 33 angular modes) with each N. Chroma prediction modes are supported, but luma prediction mode is always used.

IPE outputs the best IP mode to an *intra picture prediction* (*IPP*) stage. IPP computes *intra prediction* (P_{intra}) for the processed PU by accessing a *current picture buffer* (*CPB*) that contains previously reconstructed blocks of the current picture (D_{Rec}). A *prediction residual* (D) is computed by subtracting P_{intra} from the original CU.

B. Transform and Quantization

A *transform* (T) stage converts spatial domain D into *transform domain coefficients* (*TCOEFFs*) after which TCOEFFs are quantized in a *quantization* (Q) stage. Kvazaar supports square-shaped *transform blocks* (*TBs*) of size 4 × 4, 8 × 8, 16 × 16, and 32 × 32 pixels. It implements integer *discrete sine transform* (*DST*) for intra-coded 4 × 4 luma TBs and integer *discrete cosine transform* (*DCT*) for the other TBs. Multiple TBs inside a single CU can be arranged in a quadtree structure whose maximum depth is three. Kvazaar also supports *transform skip* (*TS*) that may replace transform of a 4 × 4 TB by bit shift.

The decoding path uses *inverse quantization* (Q^{-1}) and *inverse transform* (T^{-1}) stages to dequantize and convert *quantized TCOEFFs* back to spatial domain D (D'). D_{Rec} is then yielded by adding P_{intra} to D'.

C. Loop Filtering

A *loop filtering* (*LF*) stage filters the distortions and visible CU/PU/TU borders from the picture. The LF stage of Kvazaar contains two sequential in-loop filters: *deblocking filter* (*DF*) and *sample-adaptive offset* (*SAO*), in that order. Two SAO types are adopted: *edge offset* and *band offset*.

DF and SAO operate at CTU level. In DF, filtering the pixels of the right and bottom edges of the CTU is dependent on its neighboring CTUs queued later in the coding order. Kvazaar avoids this dependency by selecting *SAO parameters* (*SAO param*) before DF has processed these pixels. This approach speeds up SAO with only a slight RD loss.

TABLE I. Coding Options of Kvazaar HEVC Intra Encoder

Feature	Kvazaar HEVC intra encoder	
	RD1	RD2
Profile	Main	
Internal bit depth	8	
Color format	4:2:0	
Coding modes	Intra	
Sizes of luma CBs	64×64, 32×32, 16×16, 8×8	
Sizes of luma TBs	32×32, 16×16, 8×8, 4×4	
Sizes of luma PBs	32×32, 16×16, 8×8, 4×4	
IP modes	DC, planar, 33 angular	
Mode decision metric	SATD, SAD	SSD
RDO	Light	Heavy
RDOQ	Enabled	
Transform	integer DCT (integer DST for luma 4×4)	
4x4 transform skip	Enabled	
Loop filtering	DF, SAO	
Transform split	Disabled	

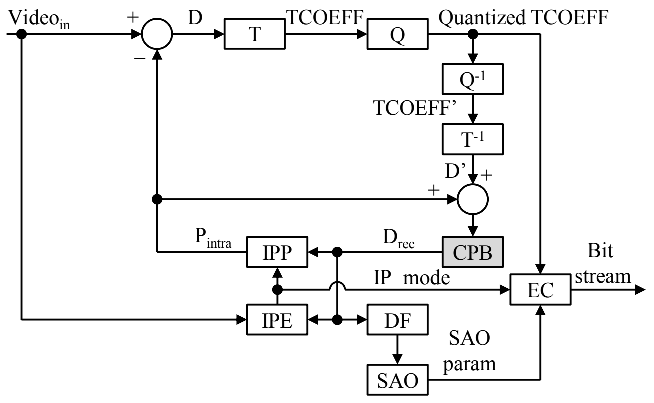

Fig. 1. Block diagram of Kvazaar HEVC intra encoder.

D. Entropy Coding

An *entropy coding* (*EC*) stage converts quantized TCOEFFs, IP mode, SAO param, and other syntax elements to binary codewords which are multiplexed together to a bit stream. The used EC technique is *context-adaptive binary arithmetic coding* (*CABAC*).

Kvazaar estimates bit costs for context coded bits with two methods. The first method performs CABAC coding for TCOEFFs only and counts the number of coded bits. The second method uses a table adopted from *RD optimized quantization* (*RDOQ*) to estimate fractional bits according to current context state.

To reduce encoding time, the contexts are only updated at CTU level during the actual bitstream generation. Even though the contexts become increasingly outdated as the encoding approaches the end of the CTU, the associated RD loss is still considered acceptable for Kvazaar.

E. Mode Decision

Kvazaar uses a two-step strategy to find the best IP mode. The first step tests DC and planar modes and conducts a half-interval mode search on the directional modes. The cost function used for all block sizes is SATD. For 4 × 4 blocks, SAD is also used to retain potential modes for transform skip. A normalized SAD cost augmented by the coding cost of transform skip is compared to the SATD cost of the mode and better cost is selected. The final decision whether to use transform skip is done later during reconstruction.

At the second step, the modes are sorted according to their cost. For RD1, the mode with the best cost is selected. For RD2, 8 best modes with 4 × 4 and 8 × 8 blocks and 3 best modes with 16 × 16 and 32 × 32 blocks are passed on to a slower reconstructing search adopted from HM. The slower search codes the residual to get a better estimate of the final RD cost.

A separate chroma mode search is not done in RD1. Instead, chroma is taken into account during RD2 mode search and luma mode is always used for chroma.

IV. Rate-Distortion-Complexity Analysis

The RDC characteristics of Kvazaar version 0.3.5 are evaluated by benchmarking its single-threaded *C* implementation (RD1 and RD2) against the latest versions of HM and x265, i.e., HM 16.0 [6], [14] and x265 version 1.3+372-96609efaa877. The parameters used for each encoder are listed in Table II.

Our experiments rely on the default configuration of HM 16.0 which is used as an anchor for the obtained results. The evaluated presets of x265 are *placebo* and *ultrafast*. These boundary cases represent the slowest and fastest presets of x265, respectively. For a fair comparison, the experiments are conducted with a single-threaded C++ implementation of x265 by disabling its assembly optimizations and parallelization options. x265 is selected for the comparison due to its cutting-edge status among the open-source HEVC encoders.

TABLE II. Encoder Parameters Used for Testing

Encoder	Parameters
HM	-c encoder_intra_main.cfg -f (frames) -wdt (width) -hgt (height) -fr (fps) -q (qp)
Kvazaar	--input-res (resolution) -n (frames) --cpuid 0 -p 1 -q (qp) --rd (preset)
x265	-I 1 --no-scenecut --ipratio 1 -p (preset) -f (frames) -q (qp) --input-res (resolution) --fps (fps) --no-asm --tune psnr --psnr --log-level debug --no-progress --threads 1 --frame-threads 1

A. Setup for Rate-Distortion Analysis

The RD comparisons between Kvazaar, HM, and x265 are based on the sequence-specific bit rate differences for an identical PSNR_{AVG} value that is a weighted average of luma (PSNR_Y) and chroma (PSNR_U and PSNR_V) PSNR components [15]. All involved test sequences are in 4:2:0 color format, for which PSNR_{AVG} values per picture are computed as

$$\text{PSNR}_{\text{AVG}} = (6 \times \text{PSNR}_Y + \text{PSNR}_U + \text{PSNR}_V)/8, \qquad (1)$$

where PSNR_Y, PSNR_U, and PSNR_V components are obtained by averaging their picture-specific values.

The sequence-specific bit rate differences have been compared in terms of the *Bjøntegaard delta bit rate* (*BD-rate*) [16]. For each scheme, the RD curves for the BD-rate computations have been interpolated through experimentally specified RD points that represent the QPs of 22, 27, 32, and 37. Here, the BD-rate calculations are based on the piecewise cubic interpolation [17].

B. Setup for Complexity Analysis

The analysis relies on Intel VTune Amplifier XE profiler, which is able to report estimated cycle counts for each encoder function. Our profiling environment is composed of 3.0 GHz Intel Core 2 Duo E8400 processors running 64-bit Windows 7 Enterprise SP 1. Only a single core per processor has been used. The benchmarked encoders have been compiled with Microsoft Visual Studio 12.0.30723.00 Update 3 as 64 bit release version using the build scripts included in each project.

C. RDC Analysis

Table III tabulates the RDC characteristics of Kvazaar and x265 over HM 16.0. The test sequences adopted are from HEVC common test conditions (classes *A*-*F*) [18].

The high-quality RD2 preset of Kvazaar attains an average speedup of 2.3× (1.8-4.3×) over HM with BD-rate loss of 1.7% (0.7-4.1%). Compared with the placebo preset of x265, the BD-rate of RD2 is around 0.1% ahead and it is still almost 1.3× faster.

The average BD-rate loss of the high-speed RD1 preset of Kvazaar is 8.8% (6.2-11.5%) over HM, but it is also more than 10 times faster. Compared to x265, RD1 outperforms ultrafast preset of x265 without compromises. In this case, Kvazaar almost halves the BD-rate overhead from 15.0% to 8.8% and more than doubles the speedup from 4.8× to 10.6×.

TABLE III. RDC VALUES OF KVAZAAR AND X265 ENCODERS OVER HM 16.0 IN ALL-INTRA CODING CONFIGURATION

Class	Format	Sequence	# of	Frame	Kvazaar (RD2)		x265 (placebo)		Kvazaar (RD1)		x265 (ultrafast)	
			frames	rate	BD-rate	Speedup	BD-rate	Speedup	BD-rate	Speedup	BD-rate	Speedup
A	2560×1600 (1600p)	Traffic	150	30 fps	1.5%	2.2×	1.5%	1.9×	7.2%	9.8×	13.6%	4.6×
		PeopleOnStreet	150	30 fps	1.3%	2.2×	1.9%	1.9×	8.3%	10.0×	13.5%	4.7×
B	1920×1080 (1080p)	Kimono	240	24 fps	0.7%	2.4×	1.2%	1.9×	6.9%	10.5×	24.1%	4.4×
		ParkScene	240	24 fps	0.7%	2.2×	1.3%	1.8×	6.2%	9.8×	12.3%	4.7×
		Cactus	500	50 fps	1.4%	2.2×	1.7%	1.8×	8.2%	10.3×	14.3%	4.7×
		BQTerrace	600	60 fps	1.2%	2.3×	0.5%	1.9×	7.0%	10.6×	12.4%	4.9×
		BasketballDrive	500	50 fps	1.2%	2.4×	2.2%	2.0×	8.9%	10.8×	17.7%	4.7×
C	832×480 (WVGA)	RaceHorses	300	30 fps	1.1%	2.0×	1.7%	1.7×	7.1%	9.5×	11.4%	4.7×
		BQMall	600	60 fps	1.5%	2.1×	2.1%	1.9×	9.4%	9.9×	11.3%	4.8×
		PartyScene	500	50 fps	1.3%	1.8×	1.4%	1.7×	7.8%	9.1×	8.1%	5.1×
		BasketballDrill	500	50 fps	2.3%	2.1×	1.9%	1.8×	9.9%	9.8×	13.3%	4.7×
D	416×240 (WQVGA)	RaceHorses	300	30 fps	1.3%	1.9×	1.9%	1.7×	8.7%	9.1×	10.6%	4.7×
		BQSquare	600	60 fps	1.6%	1.9×	1.4%	1.7×	8.2%	9.6×	8.6%	5.1×
		BlowingBubbles	500	50 fps	1.6%	1.8×	1.6%	1.7×	8.7%	9.0×	8.2%	5.1×
		BasketballPass	500	50 fps	1.4%	2.0×	2.0%	1.8×	9.5%	9.3×	11.2%	4.6×
E	1280×720 (720p)	FourPeople	600	60 fps	1.6%	2.6×	2.5%	1.9×	9.1%	11.2×	13.6%	4.6×
		Johnny	600	60 fps	2.4%	3.2×	2.7%	2.0×	11.5%	13.2×	21.6%	4.5×
		KristenAndSara	600	60 fps	2.0%	3.0×	2.4%	2.0×	11.3%	12.6×	16.9%	4.6×
F	WVGA	BasketballDrillText	500	50 fps	2.4%	2.1×	1.7%	1.8×	10.0%	9.7×	12.9%	4.8×
	1024×768	ChinaSpeed	500	30 fps	2.6%	2.2×	2.5%	1.8×	10.1%	10.5×	25.2%	4.8×
	720p	SlideEditing	300	30 fps	1.6%	2.2×	2.1%	1.8×	8.5%	11.1×	29.1%	5.2×
		SlideShow	500	20 fps	4.1%	4.3×	2.6%	2.2×	10.8%	17.5×	20.6%	4.6×
Average					1.7%	2.3×	1.8%	1.8×	8.8%	10.6×	15.0%	4.8×

V. CONCLUSIONS

This paper introduced an open-source Kvazaar HEVC encoder that implements almost the same intra coding functionality as HM but its rewritten source code makes it significantly faster. On average, the high-quality and high-speed presets of a single-threaded *C* implementation of Kvazaar are 2.3× and 10.6× faster than HM at a cost of BD-rate loss of 1.7% and 8.8%, respectively. Kvazaar also outperforms a single-threaded C++ implementation of x265. In high-quality test case, Kvazaar improves BD-rate by 0.1% and is almost 1.3× faster than x265. The respective values are widened to 6.2% and 2.2× in the high-speed test case.

Kvazaar has been implemented by prioritizing modularity, portability, and readability of the source code. The selected design approach also makes Kvazaar a feasible implementation for hardware acceleration, parallelization, and mapping to different platforms.

ACKNOWLEDGMENT

This work was supported in part by the European Celtic-Plus Project H2B2VS. The authors would also like to thank all contributors of Kvazaar open-source project [9].

REFERENCES

[1] *High Efficiency Video Coding*, document ITU-T Rec. H.265 and ISO/IEC 23008-2 (HEVC), ITU-T and ISO/IEC, Apr. 2013.

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

[4] *Advanced Video Coding for Generic Audiovisual Services*, document ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), ITU-T and ISO/IEC, Mar. 2009.

[5] 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.

[6] *Joint Collaborative Team on Video Coding Reference Software, ver. HM 16.0* [Online]. Available: http://hevc.hhi.fraunhofer.de/

[7] *x265* [Online]. Available: http://x265.org/

[8] *f265* [Online]. Available: http://f265.org/

[9] *Kvazaar HEVC encoder* [Online]. Available: https://github.com/ultravideo/kvazaar

[10] *ces265* [Online]. Available: http://sourceforge.net/projects/ces265/

[11] *HomerHEVC encoder* [Online]. Available: https://github.com/jcasal-homer/HomerHEVC

[12] *Ultra video group* [Online]. Available: http://ultravideo.cs.tut.fi/

[13] I. K. Kim, J. Min, T. Lee, W. J. 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.

[14] K. McCann, C. Rosewarne, B. Bross, M. Naccari, K. Sharman, and G. Sullivan, "High Efficiency Video Coding (HEVC) Test Model 16 (HM16) encoder description," *Document JCTVC-R1002*, Sapporo, Japan, Jul. 2014.

[15] J. R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand," Comparison of the coding efficiency of video coding standards – including High Efficiency Video Coding (HEVC)," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1669-1684.

[16] G. Bjøntegaard, "Calculation of average PSNR differences between RD curves," *document VCEG-M33*, Austin, TX, USA, Apr. 2001, pp. 1-4.

[17] J. Wang, X. Yu, and D. He, "On BD-rate calculation," *Document JCTVC-F270*, Torino, Italy, Jul. 2011.

[18] F. Bossen, "Common test conditions and software reference configurations," *Document JCTVC-J1100*, Stockholm, Sweden, Jul. 2012.