Kvazaar: Open-Source HEVC/H.265 Encoder

Marko Viitanen, Ari Koivula, Ari Lemmetti, Arttu Ylä-Outinen, Jarno Vanne, Timo D. Hämäläinen
Department of Pervasive Computing
Tampere University of Technology, Finland

{marko.viitanen, ari.koivula, ari.lemmetti, arttu.yla-outinen, jarno.vanne, timo.d.hamalainen}@tut.fi

ABSTRACT

Kvazaar is an academic software video encoder for the emerging High Efficiency Video Coding (HEVC/H.265) standard. It provides students, academic professionals, and industry experts a free, cross-platform HEVC encoder for x86, x64, PowerPC, and ARM processors on Windows, Linux, and Mac. Kvazaar is being developed from scratch in C and optimized in Assembly under the LGPLv2.1 license. The development is being coordinated by Ultra Video Group at Tampere University of Technology (TUT) and the implementation work is carried out by an active community on GitHub. Developer friendly source code of Kvazaar makes joining easy for new developers. Currently, Kvazaar includes all essential coding tools of HEVC and its modular source code facilitates parallelization on multi and manycore processors as well as algorithm acceleration on hardware. Kvazaar is able to attain real-time HEVC coding speed up to 4K video on an Intel 14-core Xeon processor. Kvazaar is also supported by FFmpeg and Libav. These de-facto standard multimedia frameworks boost Kvazaar popularity and enable its joint usage with other well-known multimedia processing tools. Nowadays, Kvazaar is an integral part of teaching at TUT and it has got a key role in three Eureka Celtic-Plus projects in the fields of 4K TV broadcasting, virtual advertising, Video on Demand, and video surveillance.

Keywords

Open source; Video coding; Video encoder; High Efficiency Video Coding (HEVC); Kvazaar HEVC encoder

1. INTRODUCTION

Video traffic is reported to reach 80-90% of all global consumer Internet traffic by 2019 [1]. This is due to rapid proliferation of video applications together with universal expectations of better-quality video content and immersive user experience. The holistic growth of video inevitably sparks a need for better compression.

The latest international video coding standard, *High Efficiency Video Coding* (*HEVC*/*H.265*) [2], [3] is developed to address the increasing transmission and storage requirements of video. HEVC reduces the bit rate by almost 40% over the preceding state-of-the-art standard AVC [4] for the same objective quality but at about 40% encoding complexity overhead. Therefore, implementing a powerful HEVC encoder with a reasonable coding speed, cost, and power budget requires an efficient encoder implementation.

In software encoders, HEVC complexity can be primarily tackled by two techniques: multithreading through data-level parallelism and *single instruction multiple data* (*SIMD*) optimizations. Further speedup and lower power dissipation can be obtained by offloading the compute-intensive coding tools to hardware accelerators or implementing the entire encoder on hardware.

Currently, there exist a couple of open-source HEVC encoders [5]-[9]. The summary of these projects is given in Table 1. *HEVC reference encoder* (*HM*) [5] supports all HEVC coding tools and is able to achieve high coding efficiency. However, it is slow and single threaded, as it is targeted for research and conformance testing rather than practical encoding. The commercially funded x265 [6] is probably the most well-known practical open-source HEVC encoder at the moment. It is based on the C++ source code of HM which has been enhanced by extensive assembly optimizations, multithreading, and techniques from the open-source x264 encoder [10]. HomerHEVC [7] and f265 [8] are less feature complete encoders and both of them have only a single active developer.

Kvazaar [9] is an academic cross-platform HEVC encoder coordinated by Ultra Video Group [11] at *Tampere University of technology* (*TUT*). The development of Kvazaar was started by us in a private academic project in May 2012. The source code of Kvazaar was published on GitHub under GPLv2 in Jan 2014 and relicensed as LGPLv2 in Feb 2015. The latest version of its source code and issue tracker can be found on GitHub [9]. Kvazaar allows participation without any copyright transfers contrary to x265, f265, and HomerHEVC that require signing a *contributor licence agreement* (*CLA*) before accepting code into the project.

This paper gives an overview of Kvazaar including its main features, high-level functionality, and target applications. It serves as a Kvazaar synopsis for software programmers, video coding experts, and end-users in academia and industry.

2. OVERVIEW OF KVAZAAR

The development of Kvazaar was started from scratch in C while using HM as a clarification of the standard text and as a source of fundamental HEVC algorithms. Currently, Kvazaar supports HEVC Main, Main Still Picture, and Main 10 profiles for 8-bit 4:2:0 progressive video.

MM '16, October 15 - 19, 2016, Amsterdam, Netherlands

ACM 978-1-4503-3603-1/16/10…$15.00
DOI: http://dx.doi.org/10.1145/2964284.2973796

Table 1. Open-source HEVC encoders

Feature	HM [5]	x265 [6]	HomerHEVC [7]	f265 [8]	Kvazaar
License	BSD	GPL2 / Commercial	LGPLv2.1	BSD	LGPLv2.1
Coordinator	JCT-VC	MulticoreWare	Juan Casal	Vantrix	TUT
Contribution	Free	With CLA	With CLA	With CLA	Free
Core language	C++	C++	C	C	C
Lines of C/C++	56k	51k	21k	23k	22k
Lines of assembly	0	149k	0	4k	2k
% of comments	14 %	7 %	13 %	25 %	22 %
Contributors	133	102	2	3	15
Commits	4879	11458	243	67	2068
Last commit	Active	Active	2/2016	9/2014	Active

The main development goals of Kvazaar are: 1) Coding efficiency close to HM; 2) Easy portability to various platforms; 3) Real-time coding speed; 4) Optimized computation and memory resources; and 5) Well-documented source code. A priority list for these objectives is specified by a target application since working simultaneously for all of them needs compromises. In practice, predefined presets are used to facilitate selecting different speed and quality settings for Kvazaar.

2.1 Usage

Kvazaar accepts uncompressed video in YUV 4:2:0 format and outputs HEVC bitstream. It comes with own *command line interface* (*CLI*) that is documented in the readme file. Alternatively, Kvazaar can be compiled as a library for embedding into other applications. The C-language interface is defined in kvazaar.h and is also used by the Kvazaar CLI. So far, support for using Kvazaar as a library has been included into FFmpeg and Libav multimedia frameworks.

Kvazaar supports ten presets: *ultrafast*, *superfast*, *veryfast*, *faster*, *fast*, *medium*, *slow*, *slower*, *veryslow*, and *placebo*. The naming convention follows that of x264. The fastest preset, *ultrafast*, is targeted for low-cost real-time coding. It reduces encoding complexity by supporting the most essential coding tools only. On the other hand, the most computation-intensive preset, *placebo*, enables every available tool. It is mainly recommended for testing and offline coding for the best possible quality. In practise, *veryslow* preset tends to be a better selection for high-quality coding due to better balance between quality and speed. Other presets are defined to fit between these two ends with respect to coding speed.

2.2 Software Architecture

Figure 1 depicts the high-level architecture of Kvazaar. Each module is implemented using one or more compilation units as described in the readme file and Doxygen documentation.

The entry point for users is either *Kvazaar CLI* or the library *libkvazaar* that is used by 3rd party software such as FFmpeg and Libav. The core of Kvazaar is an *Encoder* module which controls the state of the encoder and implements the interfaces for libkvazaar. Efficient *Threading* module is crucial for Kvazaar. The implementation is a thread queue built on top of Posix threads. On Windows, Pthreads-w32 is used to provide native threading.

Compression, *Reconstruction*, and *HEVC bitstream* modules implement the HEVC coding tools that handle compression and output standard-compliant bitstream, while *Strategies* module provides platform-specific optimized implementations via dynamic dispatch. An optional *Visualization* module provides a graphical presentation of the state of the currently encoded pictures.

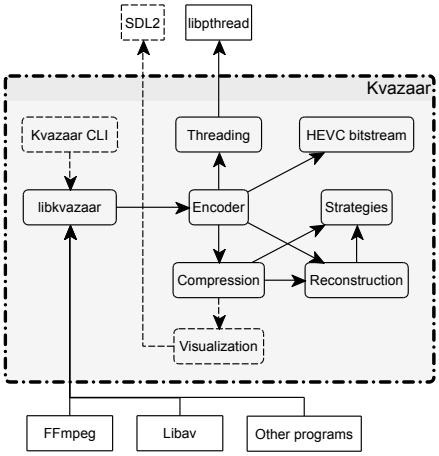

Figure 1. Module structure of Kvazaar.

2.3 Implementation

Kvazaar has three types of threads: an input-thread for reading the uncompressed images, worker threads for conducting the actual encoding, and the main thread for the remaining tasks. After receiving the input picture, the main thread assigns it to a picture level encoding.

Table 2 lists the essential coding parameters and coding tools of Kvazaar. The supported HEVC coding configurations are *all-intra* (*AI*), *random access* (*RA*), and *low-delay P* (*LP*). Kvazaar encodes pictures at the granularity of a *coding tree block* (*CTB*) of 64 × 64 pixels, according to the block diagram depicted in Figure 2. CTBs can be optionally divided into four equal-sized square *coding blocks* (*CBs*) and the division can be recursively continued until the minimum size of CBs is reached. The final quadtree structure is selected based on coding cost using a depth-first search with an early termination condition.

Within each CB, a *rate-distortion optimized* (*RDO*) prediction mode search is performed. Kvazaar offers all 35 *intra prediction* (*IP*) modes for choosing an *intra mode* (P_{intra}) and all square, *symmetric* (*SMP*), and *asymmetric* (*AMP*) motion partition modes for choosing an *inter mode* (P_{inter}). The motion parameters (MV, idx) are obtained through *motion estimation* (*ME*) that refers to previously coded pictures. ME includes *integer ME* (*IME*) stage and optional *fractional ME* (*FME*) stage that interpolates values between the pixels. *Motion compensation* (*MC*) produces P_{inter} by using motion parameters to access pixels from a *decoded picture buffer* (*DPB*) which contains the previously reconstructed *reference pictures* (D_{ref}).

After choosing a prediction mode for the CB, the *difference* (*D*) between the original picture and prediction is transformed (*T*)

Table 2. Coding parameters of Kvazaar

Feature	Kvazaar HEVC encoder
Profile	Main, Main Still Picture, Main 10
Internal bit depth	8 (10 if enabled when compiling)
Color format	4:2:0
Coding modes	Intra, Inter, Skip, Merge
Slice Types	I, P, B
Coding Configurations	AI, RA, LP
Coding Blocks	64×64, 32×32, 16×16, 8×8
Intra Prediction modes	DC, planar, 33 angular
Intra Prediction Blocks	32×32, 16×16, 8×8, 4×4
Inter Prediction Blocks	64×64, 32×32, 16×16, 8×8, 4×4, AMP, SMP
Transform Blocks	32×32, 16×16, 8×8, 4×4
Transform	Integer DCT (Integer DST for luma 4×4)
4x4 transform skip	Enabled
Transform split	64×64
Parallelization	Tiles, Wavefront, Picture-level
Rate Control	Available
RDO	Partial (includes RDOQ)

with *Discrete Cosine/Sine Transforms* (*DCT/DST*) to *transform domain coefficients* (*TCOEFFs*) and quantized (*Q*). It is possible to use *RDO quantization* (*RDOQ*) and sign hiding at this stage. After *inverse quantization* (Q^{-1}) and inverse *transformation* (T^{-1}), *loop filtering* (*LF*) is applied to the reconstructed pixels (D_{Rec}) stored in *coded picture buffer* (*CPB*). Deblocking is done after the CTU is finished, but *sample adaptive offset* (*SAO*) is only applied for a single row of CTUs at a time. The loop filtered pixels are stored in a DPB which is implemented as a single reference counted pixel buffer that can be read from the other picture-level encoders, even before the encoding is finished. All syntax elements are converted into a standard compliant HEVC bitstream by *entropy coding* (*EC*).

2.4 Parallelization

For parallel encoding, Kvazaar offers three schemes: 1) tiles; 2) *Wavefront Parallel Processing* (*WPP*); and 3) picture-level parallel processing. WPP and tiles divide a picture into multiple partitions that can be processed in parallel.

All these schemes have been implemented using a dynamic task graph and a thread pool, where the workers in the pool perform the tasks with no unmet dependencies. This scheme is illustrated with pictures of size 3×3 CTUs in Figure 3 where black arrows show the direction of dataflow and grey arrows depict a possible dependency graph. The input thread reads the uncompressed video and stores it in the buffer. The main thread creates new frame encoders and the task graphs required to encode the frame correctly. The workers in the thread queue perform encoding. The size of the work unit is a single CTB/LF. The main thread collects the resulting bitstreams and outputs the compressed video.

Experiments with Intel 8-core i7 [12] and 61-core Xeon Phi [13] processors show that Kvazaar scales almost linearly to the number of cores in the processor.

2.5 Optimizations

Kvazaar is compatible with x86, x64, PowerPC, and ARM processors on Windows, Linux, and Mac. It uses a dynamic dispatch mechanism for SIMD-optimized functions, i.e., the function is chosen based on instruction sets supported by the processor. Optimizations exist for SSE2, SSE4.1, AVX, AVX2, and AltiVec, but the majority of them are for AVX2. Most of the optimized functions are implemented in C using intrinsics, but a few assembly versions are also maintained in order to have an infrastructure for contributors using assembly directly.

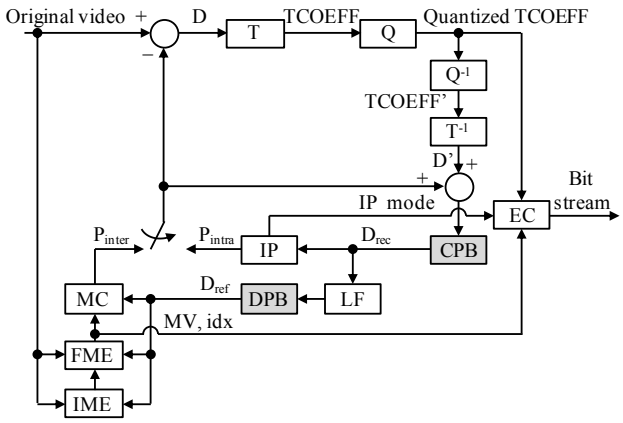

Figure 2. Block diagram of HEVC encoding in Kvazaar.

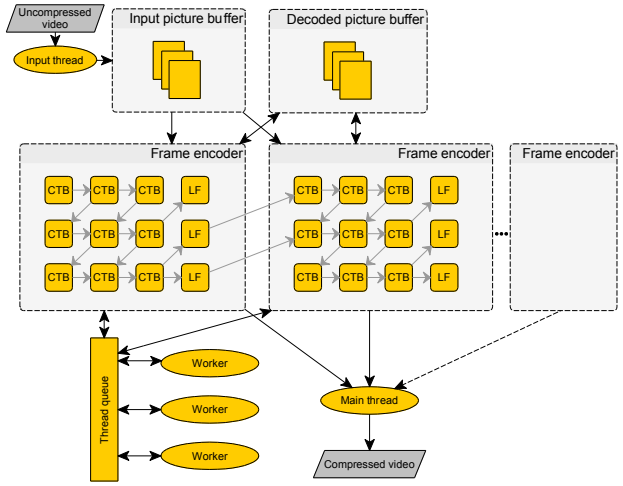

Figure 3. Diagram of Kvazaar parallelization architecture.

The cumulative speedup from the optimizations for the whole encoder is around 2.0× for all-intra coding, which is enough to encode Full HD resolution in real time on Intel 8-core i7 processor [14]. Kvazaar is also able to attain real-time HEVC coding speed up to 4K resolution on Intel 14-core Xeon processor.

Hardware-oriented source code eases acceleration of Kvazaar, e.g., on *field-programmable gate array* (*FPGA*) through high-level synthesis [15].

3. VISUALIZATION TOOL

The analysis of HEVC block structures and coding mode decisions is an integral part of the encoder development. However, the needed analysis tools are usually meant for offline use. In Kvazaar, a tool for visualizing the encoding process is integrated into the encoder. This allows real-time debugging of the encoding, which is more informative than the offline analysis since the information used for coding decisions is also available.

The visualizer is built using *Simple DirectMedia Layer* (*SDL*) multi-platform graphics library version 2. Illustrated components are 1) block partitioning; 2) intra prediction; 3) inter prediction; and 4) block reconstruction [16]. Figure 4 depicts two snapshots of the visualization. The block reconstruction is used as the background layer and the other layers can be toggled on and off. This way, Kvazaar developers can focus on visualization elements of particular interest.

Figure 4. Kvazaar visualizer output.

Multi-threaded processing is also visualized, including WPP, tiles, and picture-level parallel processing. The block borders and motion between consecutive pictures are marked with distinct colors to make identification of the individual pictures easier.

4. USE IN EDUCATION

Kvazaar is used as an exercise application for teaching embedded system development in the system design course at TUT. The exercise work is to implement a live streaming Kvazaar on a *System-on-Chip FPGA* (*SoC-FPGA*) which consists of a dual-core ARM processor, programmable logic, Full HD camera, and gigabit Ethernet. The design tasks include Kvazaar source code profiling for identifying the critical functions for FPGA acceleration and implementing the respective hardware/software partitioning to meet the performance goals. Deploying research results immediately into teaching resembles the development work in a company due to which the student feedback of the course has been very positive.

5. CONCLUSION AND FUTURE WORK

This paper presented a cross-platform academic Kvazaar encoder for the newest international video coding standard HEVC. Kvazaar is already included in the popular FFmpeg and Libav multimedia frameworks, deployed in teaching at TUT, and selected as a core component in three European multinational R&D projects where Kvazaar is being used in application development for 4K TV broadcasting, virtual advertising, Video on Demand, video surveillance, and video conferencing.

In the future, the objective is to make Kvazaar a leading open-source HEVC encoder globally in terms of performance, versatility, portability, and popularity among end-users and developers in the open-source community. HEVC Format Range Extensions will be added in the portfolio of Kvazaar coding tools to enable: 1) Format range extensions up to 12-bit 4:4:4 8K@120 fps Super Hi-Vision video; 2) Scalable HEVC coding to guarantee video streaming in resource-constrained environments; and 3) Multiview/3D coding to produce 3D, free viewpoint, and 360-degree panoramic videos. Foreseen future applications include virtual/augmented reality, free-viewpoint TV, telerobotics, telemedicine, and self-driving vehicles.

6. ACKNOWLEDGMENTS

This work was supported in part by the three European Celtic-Plus Projects: 4KREPROSYS, H2B2VS, and VIRTUOSE and by the Academy of Finland, decision no 301820. The authors would also like to thank all contributors of Kvazaar open-source project [9].

7. REFERENCES

[1] Cisco, *Cisco Visual Networking Index: Forecast and Methodology*, 2014-2019, May 2015.

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

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

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

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

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

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

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

[10] *x264* [Online]. Available: http://www.videolan.org/developers/x264.html

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

[12] 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.

[13] A. Koivula, M. Viitanen, A. Lemmetti, J. Vanne, and T. D. Hämäläinen, "Performance evaluation of Kvazaar HEVC intra encoder on Xeon Phi many-core processor," *in Proc. IEEE Global Conf. Signal Information Process.,* Orlando, Florida, USA, Dec. 2015.

[14] A. Lemmetti, A. Koivula, M. Viitanen, J. Vanne, and T. D. Hämäläinen, "AVX2–Optimized Kvazaar HEVC Intra Encoder," *Accepted to IEEE Int. Conf. Image Processing*, Phoenix, Arizona, USA, Sept. 2016.

[15] P. Sjövall, J. Virtanen, J. Vanne, and T. D. Hämäläinen, "High-level synthesis design flow for HEVC intra encoder on SoC-FPGA," *in Proc. Euromicro Symp. Digit. Syst. Des.*, Funchal, Madeira, Portugal, Aug. 2015.

[16] M. Viitanen, A. Koivula, J. Vanne, and T. D. Hämäläinen, "Live Demonstration: Run-time Visualization of Kvazaar HEVC Intra Encoder," *in Proc. IEEE Int. Symp. Circuits Syst.*, Montreal, Canada, May 2016.