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:

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MMSys ’20, June 8–11, 2020, Istanbul, Turkey
© 2020 Copyright held by the owner/authors. Publication rights licensed to the Association for Computing Machinery.
ACM ISBN 978-1-4503-6845-2/20/06...$15.00
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
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 \times 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 unpredicated blocks and then bipredected blocks. The unpredicted 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 bipredected blocks. The two best motion vectors of the unpredictions, one for each of the reference picture lists ($L_0$ and $L_1$), are chosen to generate the bipredected 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 ($P_{\text{inter}}$) 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.

IPP outputs the best mode to the IPP that computes the intra prediction ($P_{\text{intra}}$) for the current PU. $P_{\text{intra}}$ is computed by accessing the Current Picture Buffer (CPB) that contains previously reconstructed blocks ($D_{\text{rec}}$) 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 $P_{\text{intra}}$ or $P_{\text{inter}}$ from the original CU samples. The transform ($T$) stage transforms $D$ from the spatial domain into transform domain coefficients ($\text{TCOEFFs}$) [24]. Kvazaar supports transform units (TUs) of size $32 \times 32$, $16 \times 16$, and $8 \times 8$. It implements integer Discrete Sine Transform (DST) for intra coded $4 \times 4$ 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 $D_{\text{rec}}$ is then yielded by adding $P_{\text{inter}}$ or $P_{\text{intra}}$ to $D'$ and the result is stored into CPB.
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 frames 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 reconstituted 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].

3 Rate-Distortion-Complexity Analysis
In our experiments, the vsrslow 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 vsrslow 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).

<table>
<thead>
<tr>
<th>Class</th>
<th>Format</th>
<th>Sequences</th>
<th># of frames</th>
<th>Frame rate</th>
<th>Kvazaar 2.0 vs. HM</th>
<th>Kvazaar 2.0 vs. x265</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BD-BR</td>
<td>Speedup 1 thread</td>
</tr>
<tr>
<td>A</td>
<td>2560×1600 (1600p)</td>
<td>PeopleOnStreet</td>
<td>150</td>
<td>30 fps</td>
<td>11.9%</td>
<td>3.9× 182.9×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Traffic</td>
<td>150</td>
<td>30 fps</td>
<td>-7.7% 8.8× 328.5×</td>
<td>-12.5% 3.8×</td>
</tr>
<tr>
<td>B</td>
<td>1920×1080 (1080p)</td>
<td>BasketballDrive</td>
<td>500</td>
<td>50 fps</td>
<td>17.9% 5.5× 171.1×</td>
<td>-3.6% 3.0×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BQTerrace</td>
<td>600</td>
<td>60 fps</td>
<td>-5.1% 11.2× 202.3×</td>
<td>-11.4% 3.3×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Cactus</td>
<td>500</td>
<td>50 fps</td>
<td>0.6% 6.3× 166.9×</td>
<td>-7.8% 2.9×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kimono</td>
<td>240</td>
<td>24 fps</td>
<td>16.6% 5.6× 178.0×</td>
<td>-5.0% 2.8×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ParkScene</td>
<td>240</td>
<td>24 fps</td>
<td>9.2% 6.8× 166.6×</td>
<td>-13.9% 3.1×</td>
</tr>
<tr>
<td>C</td>
<td>832×480 (WVGA)</td>
<td>BasketballDrill</td>
<td>500</td>
<td>50 fps</td>
<td>-0.4% 4.2× 58.4×</td>
<td>-21.2% 2.9×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BQMail</td>
<td>600</td>
<td>60 fps</td>
<td>12.9% 5.1× 61.6×</td>
<td>-11.7% 3.3×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PartyScene</td>
<td>500</td>
<td>50 fps</td>
<td>6.5% 4.4× 50.7×</td>
<td>-21.0% 3.4×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RaceHorses</td>
<td>300</td>
<td>30 fps</td>
<td>18.9% 3.5× 71.5×</td>
<td>6.8% 3.5×</td>
</tr>
<tr>
<td>D</td>
<td>416×240 (WQVGA)</td>
<td>BasketballPass</td>
<td>500</td>
<td>50 fps</td>
<td>15.0% 3.6× 17.0×</td>
<td>-3.3% 1.9×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BlowingBubbles</td>
<td>600</td>
<td>60 fps</td>
<td>9.8% 4.8× 21.1×</td>
<td>-17.3% 2.9×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BQSquare</td>
<td>500</td>
<td>50 fps</td>
<td>-3.4% 9.0× 39.3×</td>
<td>-20.5% 4.7×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RaceHorses</td>
<td>300</td>
<td>30 fps</td>
<td>20.4% 3.3× 20.6×</td>
<td>3.7% 2.2×</td>
</tr>
<tr>
<td>E</td>
<td>1280×720 (720p)</td>
<td>FourPeople</td>
<td>600</td>
<td>60 fps</td>
<td>-27.5% 12.9× 131.6×</td>
<td>-20.5% 2.4×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Johnny</td>
<td>600</td>
<td>60 fps</td>
<td>-30.2% 21.5× 218.4×</td>
<td>-16.4% 3.3×</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KristenAndSara</td>
<td>600</td>
<td>60 fps</td>
<td>-22.0% 14.5× 164.5×</td>
<td>-17.4% 2.9×</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td></td>
<td></td>
<td></td>
<td>2.4% 7.5× 125.0×</td>
<td>-10.7% 3.0×</td>
</tr>
</tbody>
</table>

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 1080p 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