Parallelization of Kvazaar HEVC Intra Encoder for Multi-core Processors

Ari Koivula, Marko Viitanen, Jarno Vanne, Timo D. Hämäläinen
Department of Pervasive Computing
Tampere University of Technology
Tampere, Finland

Laurent Fasnacht
EPFL SCI-STI-MM
Ecole Polytechnique Fédérale de Lausanne
Lausanne, Switzerland

Abstract—**This paper introduces key parallelization strategies of our Kvazaar HEVC intra encoder for multicore processors. The schemes implemented in Kvazaar are 1) tiles; 2) Wavefront Parallel Processing (WPP); and 3) picture-level parallel processing. Kvazaar is the only practical open-source HEVC encoder that supports all these schemes. In addition, its rate-distortion-complexity characteristics are superior to other public implementations in all-intra (AI) coding. Our experiments with high-quality encoder presets show that a C implementation of Kvazaar is 19% faster than the corresponding implementation of x265 for the same coding efficiency with 8 threads and 38% faster with 16 threads. With the high-speed presets, Kvazaar improves coding efficiency by 4.5% while being twice as fast as x265. The high-speed preset of Kvazaar obtains almost the same coding efficiency as the high-quality preset of f265 while being 24 times faster when 16 threads are used.**

Keywords—HEVC; intra coding; Kvazaar HEVC encoder; tiles; Wavefront Parallel Processing (WPP)

I. INTRODUCTION

HEVC (*High Efficiency Video Coding*) [1], [2] is the latest international video coding standard. 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. Its first edition was finalized in January 2013 with three profiles: *Main*, *Main Still Picture*, and *Main 10*. This paper addresses Main Profile under *all-intra* (*AI*) coding [3] configuration.

In AI coding, HEVC is reported to reduce the bitrate by 23% over the preceding state-of-the-art standard AVC [4] with the same objective quality but at about 3.2× encoding complexity [5]. Therefore, efficient HEVC intra encoder implementations are vital in practical video products and services whose resolutions are increasingly approaching 4K (2160p) and 8K (4320p) formats. Implementing a high-performance HEVC encoder with a reasonable power budget presumes its parallelization between multiple low-power processors or platforms. For software encoders considered in this paper, parallelization is conducted by multithreading an implementation on multi-core processors with data-level parallelization techniques where data processing flows are identical for each core.

In addition to slices used already in AVC, HEVC supports two new data-level parallelization schemes: 1) *tiles* [6]; and 2) *Wavefront Parallel Processing* (*WPP*) [7]. The implementation details of these schemes are kept strictly confidential in commercial HEVC encoders, so only open-source implementations are considered in this paper.

Currently, there exist four noteworthy open-source HEVC encoders: *HEVC test model* (*HM*) [8], x265 [9], f265 [10], and our Kvazaar [11]. HM as an HEVC reference codec achieves the highest coding efficiency and supports both tiles and WPP, but is slow and not multithreaded.

Among the practical open-source HEVC encoders, the commercially funded x265 has gained the highest popularity. It is based on HM C++ source code which has been extensively refined by assembly optimizations and techniques from the open-source x264 [12] encoder. For parallel processing, x265 implements WPP but not tiles. Parallelization of x265 has been examined in [13] but the further development of x265 has made those results obsolete.

Another commercially backed HEVC encoder is f265. It is implemented in *C* and optimized with assembly. The parallelization tools of f265 include tiles but not WPP.

Kvazaar is an academic open-source HEVC encoder [14]. It is developed by us [15] from scratch in *C* using HM as a reference for its encoding scheme and individual algorithm implementations. This hardware-oriented approach facilitates source code acceleration, portability, and parallelization. Kvazaar is the only one of these practical open-source encoders that supports both tiles and WPP.

This paper focuses mainly on Kvazaar and especially its parallelization schemes which are each able to obtain high resource utilization, low synchronization overhead, and almost linear parallel scalability. The rest of this paper is organized as follows. Section 2 presents the main features of Kvazaar HEVC intra encoder. The parallelization schemes supported by Kvazaar are introduced in detail in Section 3. Section 4 benchmarks the *rate-distortion-complexity* (*RDC*) characteristics of Kvazaar intra coding over those of HM, x265, and f265 as a function of the encoding threads (from 1 to 16). Section 5 concludes the paper.

II. KVAZAAR HEVC INTRA ENCODER

Kvazaar intra encoder supports HEVC Main profile for 8-bit 4:2:0 video with two presets: 1) RD1 for high-speed encoding; and 2) RD2 for high-quality encoding. The parameters of RD1 and RD2 are detailed in Table I.

Kvazaar implements a complete HEVC block partitioning structure [16] in which the pictures are partitioned into *coding tree units* (*CTUs*) of size $2N_{\text{MAX}} \times 2N_{\text{MAX}}$, where $N_{\text{MAX}} \in \{8, 16, 32\}$. 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. A CTU represents depth 0 of the implemented coding tree. At depth 1, CTBs of a CTU can be optionally divided into four equal-sized square *coding blocks* (*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 $N \in \{4, 8, 16, 32\}$. Respectively, the 4:2:0 color format limits the sizes of the chroma CBs to $N \times N$. For RD1 and RD2, $N_{\text{MAX}} = 32$ and $N_{\text{MIN}} = 4$, so $h_{\text{MAX}} = 4$.

For a CB of size $2N \times 2N$, Kvazaar intra coding supports *luma prediction blocks* (*PBs*) of size $2N \times 2N$. In addition, it supports luma PBs of size $N \times N$ when $N = 4$. All 35 *intra prediction* (*IP*) modes of HEVC are realized (DC, planar, and 33 angular modes) with each N.

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 implements *transform skip* that may replace transform of a 4×4 TB by bit shift and *transform split* which splits the intra prediction block into four blocks using the same IP mode.

Mode decision of Kvazaar is similar to that of HM. *Sum of Absolute Transformed Differences* (*SATD*) is used for *Rough Mode Decision* (*RMD*). However, contrary to HM, Kvazaar iteratively focuses the search on the best mode, reducing the number of angular modes. In order to not exclude modes that could work well with transform skip, Kvazaar also computes the *Sum of Absolute Differences* (*SAD*) for 4x4 *Prediction Units* (*PUs*) and, if substantially smaller than SATD, uses it as the metric instead. Next, for RD2, *rate-distortion optimization* (*RDO*) is performed on the best modes found in RMD. For RD1, the mode is chosen based on RMD only, although PU split decisions always use RDO.

Kvazaar estimates costs for context coded bits with two methods. The first method performs *context-adaptive binary arithmetic coding* (*CABAC*) for TCOEFFs only and counts the number of coded bits. The second method uses a table to estimate fractional bits according to current context state. To increase coding time with acceptable RD cost, the CABAC contexts are only updated between CTBs. Kvazaar intra coding scheme is described in more detail in [14]. Kvazaar also implements HEVC inter coding [11] which is, however, out of the scope of this paper.

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 PBs	64x64, 32×32, 16×16, 8×8, 4×4	
IP modes	DC, planar, 33 angular	
Sizes of luma TBs	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	
Mode decision metric	SATD, SAD	SSD
RDO	Light	Heavy
RDOQ	Enabled	

III. KVAZAAR PARALLELIZATION

Kvazaar implements three parallelization schemes: tiles, WPP, and picture-level parallel processing. Tiles and WPP divide a picture into multiple partitions that can be processed in parallel. The partitions are always composed of an integer number of CTBs and their start positions inside a picture are indicated in the bitstream by entry point offsets. Picture-level parallel processing can be utilized together with tiles and WPP.

Kvazaar parallelization is implemented using a thread pool with a single CTB as the smallest work unit. The CTBs are put in a queue in the order in which they would be processed in a single threaded case, and the free worker threads always select the first CTB with no dependencies for processing.

A. Tiles

Tiles segment the picture into the rectangular groups of CTBs. The number of tiles and the location of their boundaries can be specified either at picture level or at sequence level. Typically, tiles are of equal size.

However, prediction is not made across tile boundaries and *CABAC* probabilities are reset at the beginning of each tile, while *deblocking filter* (*DF*) and *sample adaptive offset* (*SAO*) can cross tile boundaries. Unlike with WPP, RD loss grows together with the number of tiles [17].

Currently, Kvazaar does not perform either SAO or DF over tile boundaries. This design decision causes additional loss in RD performance, but makes encoding of tiles equivalent to encoding of frames. Tiles are processed in parallel, including SAO and DF, and the bitstreams are concatenated once all tiles in a picture have been finished.

B. Wavefront Parallel Processing (WPP)

In WPP, the size of each partition equals a single CTB row. Contrary to tiles, the prediction dependencies between the partitions are maintained. In addition, CABAC probabilities for the first CTB of the processed row are propagated from the second CTB of the previous row. Allowing prediction dependencies and CABAC probabilities cross the boundaries of the partitions gives significant RD gain over tiles [17].

For WPP, the maximum number of available partitions equals the number of CTB rows. However, wavefront dependencies prevent all partitions from being started simultaneously. The first CTB row of the picture is started immediately, but the latter rows cannot be started until two CTBs have been processed in the former row. This introduces parallelization inefficiency for WPP at the start and end of each picture. The same start condition holds for all CTBs in a row, i.e., the processing of the former row has to be two CTBs ahead of the latter through the whole row. Since the processing time varies between CTBs, the start condition makes the optimal processing order of CTBs less predictable. Kvazaar compensates this by tracking spatial dependencies of CTBs and assigning them to worker threads accordingly. A worker thread can select any CTB that has no unsatisfied dependencies.

In Kvazaar, DF is performed on a per CTB basis, whereas SAO is done for a whole row after the encoding of all CTBs in the row and the row below it. The bitstreams are concatenated once the whole picture is finished.

C. *Picture-level parallel processing*

In AI configuration, every thread could simply work on a different picture but it would be inefficient with large resolutions and a large number of threads. In Kvazaar, picture-level parallelization is integrated with WPP and tiles. Since there are no dependencies between pictures, the scheduler is free to select jobs from the next picture if necessary.

Kvazaar selects the number of parallel processed pictures based on their dimensions and the number of threads. The larger and wider the picture, the more WPP threads can process simultaneously. The number of parallel pictures is derived from the largest number of threads per picture so that all threads can work simultaneously for at least 85% of the picture. This constraint reduces the number of parallel pictures without sacrificing coding speed. For tiles, the number of pictures is selected so that there are four times as many tiles as threads.

IV. Experimental Results

The RDC characteristics of Kvazaar v0.4.0 are evaluated by benchmarking its RD1 (Kvazaar$_{rd1}$) and RD2 (Kvazaar$_{rd2}$) presets against HM 16.0 [18], x265 v1.4, and f265 v0.2. Release versions of Kvazaar, HM, and x265 were compiled with Visual Studio 2013 using the build files included in each project. A pre-built Windows binary for f265 v0.2 was obtained from the project website.

The command line parameters used in our encoder tests are listed in Table II. For a fair comparison, the experiments were conducted with a pure *C* implementation of Kvazaar, *C*++ implementation of x265, and *C* implementation of f265 by disabling their assembly optimizations.

Our experiments rely on the default AI configuration of HM 16.0 which is used as an anchor for the results. The evaluated presets of x265 are placebo (x265$_{placebo}$) and ultrafast (x265$_{ultrafast}$) that represent the slowest and fastest presets of x265, respectively. The presets selected for f265 are quality-10 (f265$_{q10}$) and quality-50 (f265$_{q50}$) which are the second fastest and slowest presets, respectively. The fastest preset of f265 is omitted due to its greater than 50% BD-rate loss over HM.

TABLE II. Encoder Command Line Parameters Used

Encoder	**Parameters**
f265	-w (res) -a (input) (output)
	-p "qp=(qp) key-frame-spacing=1 quality=(quality)"
tiles	-p "tiles=(xt),(yt) nb-workers=(threads),0 mt-mode=1"
HM	-c encoder_intra_main.cfg -f (frames) -fr (fps) -q (qp)
	-wdt (width) -hgt (height) --SEIDecodedPictureHash=3
Kvazaar	--input-res=(res) -n (frames) --cpuid=0 -p 1 -q (qp)
	--rd=(preset) --threads=(threads)
wpp	--wpp
tiles	--tiles-width-split=u(x tiles) --tiles-height-split=u(y tiles)
x265	-I 1 --no-scenecut --ipratio 1 -p (preset) -f (frames) -q (qp)
	--input-res (res) --fps (fps) --no-asm --tune psnr --psnr
	--log-level debug --no-progress --threads (threads) --hash 3
1 thread	--frame-threads 1

A. *Analysis setup*

The RD comparisons between Kvazaar, HM, x265, and f265 are based on the sequence-specific bitrate 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 [19]. The selected test sequences, listed in Table III, are from HEVC common test conditions (classes *A-F*) [20]. All these sequences are in 4:2:0 color format, for which PSNR$_{AVG}$ values per picture are computed as

$$\mathrm{PSNR_{AVG}} = (6 \times \mathrm{PSNR_Y} + \mathrm{PSNR_U} + \mathrm{PSNR_V})/8, \qquad (1)$$

where PSNR$_Y$, PSNR$_U$, and PSNR$_V$ components are obtained by averaging their picture-specific values.

In our evaluations, the bit rates were derived from the file sizes of the encoded videos whereas PSNR$_{AVG}$ values were computed from the associated *Mean Square Error* (*MSE*) values. MSE values were obtained by comparing the encoder output bit stream to the original YUV file with a PSNR filter of FFmpeg 2.5 [21]. The output bit stream was also decoded with HM 16.0 to check its validity.

The sequence-specific bitrate differences have been compared in terms of the *Bjøntegaard delta bitrate* (*BD-rate*) [22]. The RD curves for the BD-rate computations have been interpolated with the piecewise cubic interpolation [23] through experimentally specified RD points that represent the QPs of 22, 27, 32, and 37.

The speedups of Kvazaar, x265, and f265 were computed over HM by measuring the wall clock times of each encoder in the same four RD points as in BD-rate computation and then averaging the speedups. For a reliable comparison, only one encoder instance was run at a time and dynamic overclocking was disabled on the processor. Our profiling environment is detailed in Table IV.

B. *Single-threaded implementations*

Table V reports the RDC characteristics of a single-threaded Kvazaar, x265, and f265 over HM. Kvazaar$_{rd2}$ attains an average speedup of 1.9× (1.5-3.7×) over HM with BD-rate loss of 1.5% (0.7-2.5%). The BD-rate gain is very similar to x265$_{placebo}$, but Kvazaar$_{rd2}$ is 1.05× faster than x265$_{placebo}$. Compared with f265$_{q50}$, Kvazaar$_{rd2}$ is 4.0× faster with a substantially lower BD-rate increase.

The average BD-rate loss of Kvazaar$_{rd1}$ is 8.5% (6.0-10.9%) over HM, but the speedup is 9.0×. Kvazaar$_{rd1}$ outperforms x265$_{ultrafast}$ without compromises. In this case, Kvazaar$_{rd1}$ reduces the BD-rate gain from 13.2% to 8.5% while almost doubling the speedup. While f265$_{q10}$ is faster than Kvazaar$_{rd1,}$ it has a significant BD-rate penalty.

C. Multi-threaded implementations

Fig. 1 and Fig. 2 depict speedups and BD-rate overheads of Kvazaar, x265, and f265 over HM as a function of threads. A single-threaded HM is used as an anchor for our evaluations. In each graph, parallelization of Kvazaar is illustrated with two curves, one for tiles and the other for WPP. Fig. 1(b) and Fig. 2(b) also include an additional curve for HM to show the penalty of tiles on its BD-rate. The profiling platform has an 8 core processor with 16 logical cores, so tests were performed with up to 16 threads to see the effects of Hyper-threading.

D. Parallel scalability analysis with tiles

Parallel scalability with tiles is only analyzed between Kvazaar and f265 because x265 does not support tiles. The selected tile configurations are 2×2 for 4 threads, 2×4 for 8 threads, 3×4 for 12 threads, and 4×4 for 16 threads. Bitstreams of the smaller sequences do not conform to HEVC Main profile when encoded with many tiles due to the minimum tile size being restricted to 256×64 luma samples [1]. Therefore, we chose to increase the number of tile rows before incrementing the number of tile columns. Having 16 tiles in such small sequences is not a practical use case for tiles, but was done in order to test the threading capability of f265, which does not support picture-level parallel processing.

As illustrated in Fig. 1(a), the relative speed difference between the high-quality presets of Kvazaar and f265 increases with more threads, as moving from a single thread to 16 threads increases the speedup of Kvazaar$_{rd2}$ from 4.0× to 5.6× over f265$_{q50}$. Both encoders scale almost linearly up to 8 threads, but f265$_{q50}$ gains only an additional 8% speedup going from 8 to 16 threads, whereas Kvazaar$_{rd2}$ speeds up by 17%. The respective BD-rates increase with the number of tiles, as is shown in Fig. 1(b) and Fig. 2(b), but the growth rates remain quite stable between HM, Kvazaar$_{rd2}$, and f265$_{q50}$. The average BD-rate between Kvazaar$_{rd2}$ and f265$_{q50}$ with 16 tiles is -4.5%.

As is shown in Fig. 2(a), f265$_{q10}$ is faster than Kvazaar$_{rd1}$ with 1-8 threads but has trouble scaling beyond that, allowing Kvazaar$_{rd1}$ to catch up. The tile configuration of 3×4 seems to be particularly inefficient for f265$_{q10}$ causing performance drop with 12 threads. The average BD-rate between Kvazaar$_{rd1}$ and f265$_{q10}$ with 16 tiles is -14.2%.

E. Parallel scalability analysis with WPP

Parallel scalability with WPP is analyzed between Kvazaar and x265, as f265 does not support WPP. WPP has a diminutive effect on BD-rate in all evaluated encoders. Therefore, Fig. 1(b) and Fig. 2(b) do not contain a separate curve for HM with WPP.

As is shown in Fig. 1(a) and 1(b), the speed and BD-rate figures of Kvazaar$_{rd2}$ are very similar to x265$_{placebo}$ with 1-4 threads, but Kvazaar$_{rd2}$ starts to get ahead at 8 threads, being 1.3× faster. Using 16 threads, the gap is widened to 1.4×.

TABLE III. TEST SEQUENCES

Class	Format	Sequence	Frames	Frame rate
A	2560×1600 (1600p)	Traffic	150	30 fps
		PeopleOnStreet	150	30 fps
B	1920×1080 (1080p)	Kimono	240	24 fps
		ParkScene	240	24 fps
		Cactus	500	50 fps
		BQTerrace	600	60 fps
		BasketballDrive	500	50 fps
C	832×480 (WVGA)	RaceHorses	300	30 fps
		BQMall	600	60 fps
		PartyScene	500	50 fps
		BasketballDrill	500	50 fps
D	416×240 (WQVGA)	RaceHorses	300	30 fps
		BQSquare	600	60 fps
		BlowingBubbles	500	50 fps
		BasketballPass	500	50 fps
E	1280×720 (720p)	FourPeople	600	60 fps
		Johnny	600	60 fps
		KristenAndSara	600	60 fps
F	WVGA	BasketballDrillText	500	50 fps
	1024×768	ChinaSpeed	500	30 fps
	720p	SlideEditing	300	30 fps
		SlideShow	500	20 fps

TABLE IV. PROFILING PLATFORM

Processor	Intel Core i7-5960X Extreme (8 × 3.0 GHz)
Memory	32 GB
L1 cache	8 × 32 KB (instruction) + 8 × 32 KB (data)
L2 cache	8 × 256 KB
L3 cache	20 MB
Compiler	MS Visual Studio 12.0.30723.00 Update 3
Operating system	64-bit MS Windows 8.1

Kvazaar$_{rd2}$ scales almost linearly to the number of physical cores in the processor, whereas the performance of x265$_{placebo}$ begins to dip at 8 threads. This dip in performance per thread is highest for small sequences. For class D, it becomes noticeable already at 4 threads. Kvazaar avoids similar problems by taking resolution into account. The average BD-rate between Kvazaar$_{rd2}$ and x265$_{placebo}$ using WPP is -0.1%.

According to our high-speed preset comparison, Fig. 2(a) and Fig. 2(b), Kvazaar$_{rd1}$ is twice as fast as x265$_{ultrafast}$, with a better BD-rate. Both encoders scale nearly linearly until 8 threads, with Kvazaar$_{rd1}$ gaining an additional 26% speedup going from 8 to 16 threads whereas x265 gained 7%. The average BD-rate between Kvazaar$_{rd1}$ and x265$_{ultrafast}$ using WPP is -4.2%.

Compared with x265$_{placebo}$, x265$_{ultrafast}$ had no scaling problems with small sequences. This difference is likely due to x265$_{ultrafast}$ using a CTB size of 32×32 instead of to 64×64, which enables fitting more WPP threads into each picture. Both presets of Kvazaar use 64×64 CTB size.

Table VI summarizes the sequence-specific RDC characteristics of 16-threaded Kvazaar with WPP. On average, Kvazaar$_{rd2}$ and Kvazaar$_{rd1}$ are 18.6 and 85.9 times faster than single threaded HM, at a cost of 1.6% and 8.6% increase in BD-rate. The per sequence speedups are roughly in line with what could be expected based on the performance of the single

TABLE V. RDC RESULTS OF SINGLE-THREADED KVAZAAR, X265, AND F265 ENCODERS OVER HM 16.0 IN AI CODING.

Sequence	High-quality presets						High-speed presets					
	Kvazaar$_{rd2}$		x265$_{placebo}$		f265$_{q50}$		Kvazaar$_{rd1}$		x265$_{ultrafast}$		f265$_{q10}$	
	BD-rate	Speedup	BD-rate	Speedup	BD-rate	Speedup	BD-rate	Speedup	BD-rate	Speedup	BD-rate	Speedup
Traffic	1.52%	1.8×	1.23%	1.8×	4.45%	0.4×	7.14%	8.2×	12.48%	4.4×	22.72%	14.6×
PeopleOnStreet	1.31%	1.8×	1.50%	1.9×	4.27%	0.5×	8.23%	8.6×	12.56%	4.6×	23.21%	15.6×
Kimono	0.67%	2.0×	1.24%	1.9×	5.75%	0.4×	5.92%	8.5×	15.94%	4.2×	40.49%	14.9×
ParkScene	0.69%	1.7×	1.03%	1.7×	4.38%	0.5×	6.04%	8.1×	11.19%	4.4×	18.15%	14.6×
Cactus	1.39%	1.8×	1.38%	1.8×	5.33%	0.5×	8.03%	8.5×	12.91%	4.5×	23.70%	15.4×
BQTerrace	1.39%	1.8×	0.28%	1.8×	6.22%	0.5×	7.29%	8.8×	11.18%	4.7×	23.16%	16.6×
BasketballDrive	1.23%	2.0×	1.88%	2.0×	5.64%	0.5×	8.67%	8.9×	14.04%	4.5×	30.01%	16.0×
RaceHorses	0.81%	1.7×	1.09%	1.7×	4.47%	0.5×	6.61%	8.1×	10.27%	4.5×	19.45%	14.6×
BQMall	1.35%	1.8×	1.72%	1.9×	3.32%	0.5×	9.24%	8.6×	10.28%	4.6×	19.88%	15.5×
PartyScene	1.25%	1.5×	1.18%	1.6×	3.79%	0.5×	7.78%	8.0×	7.96%	4.9×	13.54%	15.1×
BasketballDrill	2.16%	1.8×	1.41%	1.8×	5.03%	0.5×	9.76%	8.4×	12.89%	4.6×	30.09%	15.1×
RaceHorses	0.92%	1.5×	1.49%	1.7×	3.84%	0.5×	8.20%	7.8×	9.93%	4.5×	18.06%	13.9×
BQSquare	1.38%	1.5×	1.16%	1.7×	4.23%	0.5×	8.00%	8.3×	8.23%	4.9×	16.75%	16.2×
BlowingBubbles	1.39%	1.5×	1.38%	1.6×	3.24%	0.5×	8.45%	7.9×	8.11%	4.9×	13.93%	14.4×
BasketballPass	1.02%	1.6×	1.67%	1.8×	3.28%	0.5×	9.06%	7.9×	10.02%	4.4×	21.34%	14.1×
FourPeople	1.45%	2.1×	2.17%	2.0×	3.04%	0.5×	8.99%	9.4×	12.07%	4.4×	24.64%	16.4×
Johnny	1.83%	2.7×	2.18%	2.1×	4.04%	0.5×	10.78%	10.9×	14.77%	4.4×	39.83%	18.3×
KristenAndSara	1.70%	2.5×	1.88%	2.0×	3.83%	0.5×	10.85%	10.4×	13.34%	4.4×	33.24%	17.7×
BasketballDrillText	2.33%	1.7×	1.34%	1.7×	6.45%	0.5×	9.87%	8.3×	12.63%	4.6×	27.39%	15.1×
ChinaSpeed	2.52%	1.9×	2.27%	1.8×	18.99%	0.5×	9.97%	8.9×	24.23%	4.6×	38.58%	16.3×
SlideEditing	1.56%	1.9×	2.07%	1.8×	25.63%	0.5×	8.42%	9.7×	28.82%	4.9×	41.71%	17.3×
SlideShow	2.43%	3.7×	1.94%	2.3×	9.49%	0.5×	9.13%	14.5×	15.41%	4.5×	36.09%	21.2×
Average	1.47%	1.9×	1.52%	1.8×	6.30%	0.5×	8.47%	8.9×	13.15%	4.6×	26.18%	15.9×

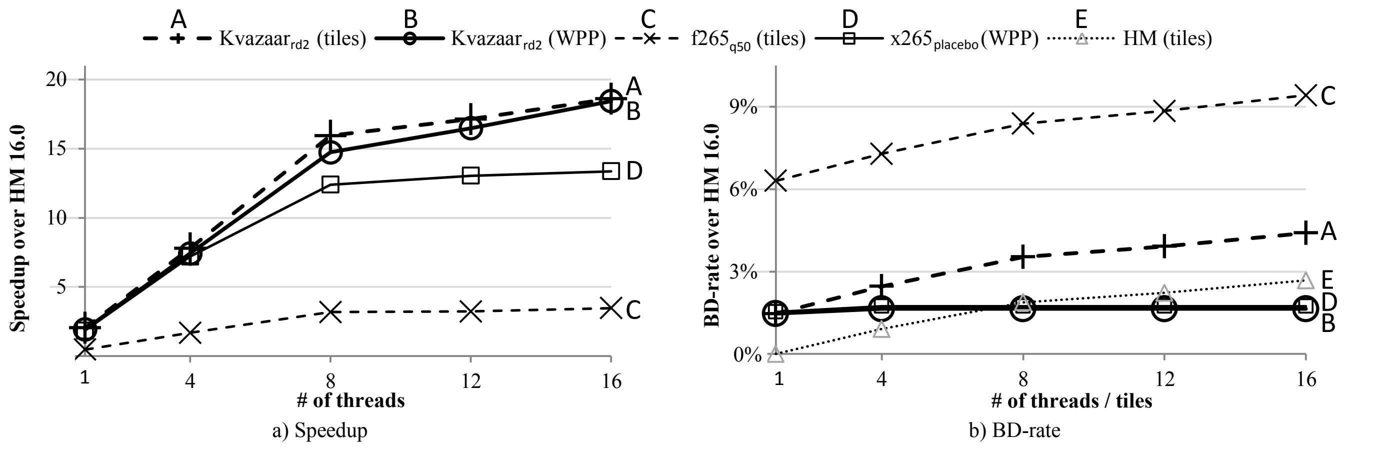

Fig. 1. Comparison of Kvazaar, x265, and f265 high-quality presets.

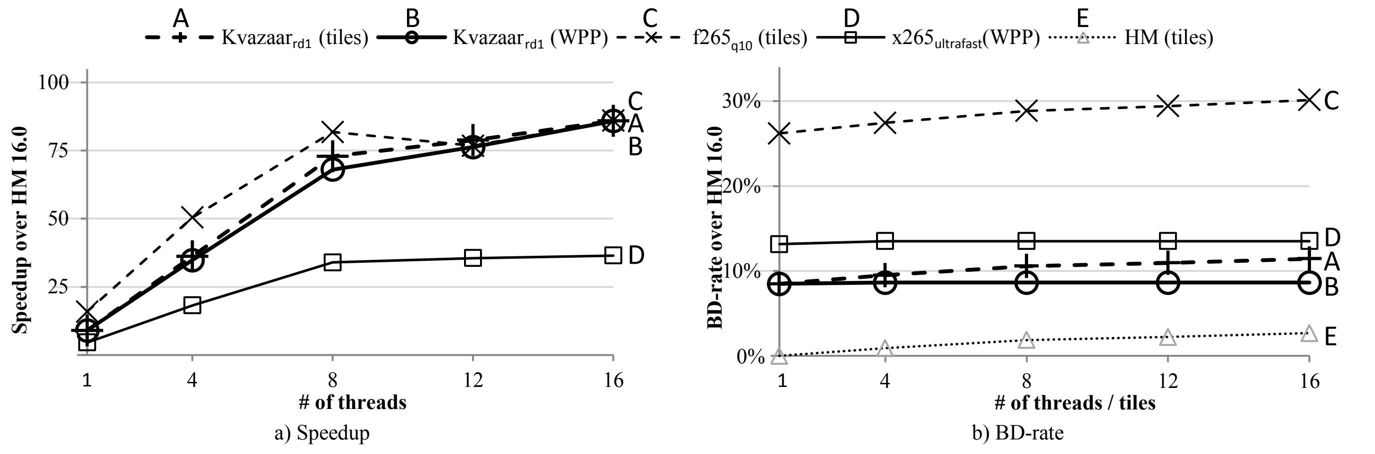

Fig. 2. Comparison of Kvazaar, x265, and f265 high-speed presets.

threaded version of Kvazaar shown in Table V, while BD-rate is slightly higher due to the use of WPP.

The reported RDC characteristics and almost linear parallel scalability of Kvazaar have been obtained by combining three design approaches: 1) individual algorithm optimizations such as reduction of angular intra modes in RMD, retaining of transform skip candidates through fast mode search, and lightweight CABAC context tracking; 2) an efficient threading model built on properly sized work units; and 3) an efficient C implementation that minimizes heap memory allocation.

V. Conclusions

This paper presented parallel Kvazaar HEVC intra encoder for multicore processors. Currently, our Kvazaar is the only practical open-source HEVC encoder that implements threading for both tiles and WPP. In addition, its intra coding characteristics were shown to be superior by benchmarking its *C* implementation on an 8-core processor against recent versions of x265 and f265 using the default configuration of HM 16.0 as an anchor. The benchmarking results illustrate that the high-quality preset of 16-threaded Kvazaar achieves speedups of 18.6× over single threaded HM, 1.4× over x265, and 5.4× over f265, with the respective BD-rate differences of +1.6%, -0.1%, and -4.5%. Correspondingly, the speedups of Kvazaar high-speed preset rise up to 85.9× over single threaded HM, and 2.4× over x265. In this case, the BD-rate of Kvazaar is 8.6% over HM, but 4.2% under x265. The high-speed preset of Kvazaar and f265 are equally fast, but Kvazaar has 14.2% lower BD-rate.

Acknowledgment

This work was supported in part by the European Celtic-Plus Project H2B2VS. The authors would particularly like to thank all contributors of our Kvazaar open-source project [11].

TABLE VI. RDC Results of 16-Threaded Kvazaar over HM 16.0

Sequence	Kvazaar$_{rd2}$ (WPP)		Kvazaar$_{rd1}$ (WPP)	
	BD-rate	**Speedup**	**BD-rate**	**Speedup**
Traffic	1.55%	17.9×	7.16%	83.2×
PeopleOnStreet	1.41%	17.3×	8.31%	79.1×
Kimono	0.90%	19.3×	6.13%	85.4×
ParkScene	0.73%	17.6×	6.10%	85.1×
Cactus	1.47%	17.6×	8.11%	81.6×
BQTerrace	1.40%	19.2×	7.29%	81.3×
BasketballDrive	1.50%	17.0×	8.91%	78.5×
RaceHorses	0.98%	17.1×	6.80%	81.2×
BQMall	1.53%	17.5×	9.38%	83.2×
PartyScene	1.27%	14.7×	7.78%	77.7×
BasketballDrill	2.16%	16.3×	9.80%	77.7×
RaceHorses	1.13%	16.0×	8.39%	75.1×
BQSquare	1.40%	14.2×	8.02%	75.7×
BlowingBubbles	1.46%	16.1×	8.52%	85.3×
BasketballPass	1.21%	14.8×	9.27%	73.4×
FourPeople	1.70%	20.6×	9.17%	90.5×
Johnny	2.59%	25.8×	11.45%	104.4×
KristenAndSara	2.19%	24.3×	11.26%	99.9×
BasketballDrillText	2.37%	16.6×	9.92%	80.0×
ChinaSpeed	2.62%	18.2×	10.06%	85.7×
SlideEditing	1.70%	18.5×	8.60%	92.9×
SlideShow	2.87%	32.5×	9.62%	132.4×
Average	1.64%	18.6×	8.64%	85.9×

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, Dec. 2012, pp. 1792-1801.

[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] K. Misra, A. Segall, M. Horowitz, X. Shilin, A. Fuldseth, and Z. Minhua, "An Overview of Tiles in HEVC," *IEEE J. Select. Topics in Signal Process.*, vol. 7, no. 6, Dec. 2013, pp. 969-977.

[7] F. Henry and S. Pateux, "Wavefront parallel processing," *Document JCTVC-E196*, Geneva, Switzerland, Mar. 2011.

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

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

[10] f265 [Online]. Available: http://f265.org/

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

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

[13] Y. Zhao, L. Song, X. Wang, M. Chen, and J. Wang, "Efficient realization of parallel HEVC intra encoding." *in Proc. IEEE Int. Conf. Multimedia and Expo*, San Jose, CA, USA, Jul. 2013.

[14] 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.

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

[16] 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.

[17] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel scalability and efficiency of HEVC parallelization approaches," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, Dec. 2012, pp. 1827-1838.

[18] 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.

[19] 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.

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

[21] *FFmpeg* [Online]. Available: http://ffmpeg.org

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

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