Chi Ching Chi, Álvarez-Mesa, M., Juurlink, B., Clare, G., Henry, F., Pateux, S., & Schierl, T.

Parallel Scalability and Efficiency of HEVC Parallelization Approaches

Chi Ching Chi, Alvarez-Mesa, M., Juurlink, B., Clare, G., Henry, F., Pateux, S., & Schierl, T. (2012). Parallel Scalability and Efficiency of HEVC Parallelization Approaches. IEEE Transactions on Circuits and Systems for Video Technology, 22(12), 1827–1838. https://doi.org/10.1109/tcsvt.2012.2223056

Parallel Scalability and Efficiency of HEVC Parallelization Approaches

Chi Ching Chi, Mauricio Alvarez-Mesa, *Member, IEEE,* Ben Juurlink, *Senior Member, IEEE,* Gordon Clare, Félix Henry, Stéphane Pateux, and Thomas Schierl, *Member, IEEE*

***Abstract*—Unlike H.264/advanced video coding, where parallelism was an afterthought, High Efficiency Video Coding currently contains several proposals aimed at making it more parallel-friendly. A performance comparison of the different proposals, however, has not yet been performed. In this paper, we will fill this gap by presenting efficient implementations of the most promising parallelization proposals, namely *tiles* and *wavefront parallel processing* (WPP). In addition, we present a novel approach called *overlapped wavefront* (OWF), which achieves higher performance and efficiency than tiles and WPP. Experiments conducted on a 12-core system running at 3.33 GHz show that our implementations achieve average speedups, for 4k sequences, of 8.7, 9.3, and 10.7 for WPP, tiles, and OWF, respectively.**

***Index Terms*—High Efficiency Video Coding (HEVC), parallel programming, video coding.**

I. Introduction

RECENT DEMANDS on video coding support for even higher resolutions such as 4k or UHD in consumer devices are driving the video coding development further. To meet these demands, the Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T and ISO/IEC Moving Pictures Experts Group has started a new project to develop a new video coding standard, called *High Efficiency Video Coding* (HEVC) [1]. The HEVC project aims at reducing the bitrate compared to H.264/AVC [2] by another 50%.

The price to be paid for higher coding efficiency is higher computational complexity. While state-of-the-art single-core processors are capable of decoding a 1080p H.264/AVC video sequence in real time, it is very unlikely that single-core processor performance will increase to a point where they can decode a 2160p50 HEVC video in real time. While International Technology Roadmap for Semiconductors [3] predicts the transistor speed will continue to improve by 25% per technology node, Borkar and Chien [4] predict only 15% improvements due to energy constraints, forcing cores to operate at low frequency and near threshold voltage. Furthermore, in low power devices state-of-the-art high-performance cores cannot be employed, and simpler cores clocked at a lower frequency must be used. Therefore, to obtain real-time HEVC decoding performance, parallelism is no longer an option but a necessity.

Unlike H.264/AVC, where parallelism was an afterthought, the current HEVC draft contains several proposals aiming at making the codec better parallelizable. H.264/AVC supports *slices*, which were introduced mainly to prevent loss of quality in the case of transmission errors, but they can also be used to parallelize the decoder. Employing slices for parallelism, however, has several problems [5]. First and foremost, using many slices to increase parallelism incurs significant coding losses. Second, the number of slices is determined by the encoder and if the decoder relies on slices to obtain real-time performance, it may not achieve this if it receives a video sequence with only one or a few slices per frame. The main parallelization approaches currently included in the HEVC draft [Tiles and wavefront parallel processing (WPP)], on the other hand, allow creation of picture partitions that can be processed in parallel without incurring high coding losses. A detailed performance comparison of different proposals has not yet been performed, however, and this paper fills the gap. Although these tools can be used for both parallel encoding and decoding, we mainly focus here on the parallel decoding capabilities, but the scalability results presented in this paper are also indicative for parallel encoding.

The main contributions of this paper can be summarized as follows.

1) We present a novel approach called overlapped wavefront (OWF) that achieves higher performance than the approaches currently included in the HEVC draft.
2) We present efficient parallel implementations of the tiles, WPP, and OWF methods that strike a good balance between design effort, amount of parallelism, data locality, and synchronization overhead.
3) We present experimental results on a parallel system with 12 cores running at 3.33 GHz for 1080p, 1600p, and 2160p video sequences. We compare the parallelization approaches in terms of speedup, scalability, and parallelization efficiency.

Manuscript received April 15, 2012; revised July 24, 2012; accepted August 22, 2012. Date of publication October 9, 2012; date of current version January 8, 2013. This paper was recommended by Associate Editor B. Pesquet-Popescu.

C. C. Chi, M. Alvarez-Mesa, and B. Juurlink are with the Embedded Systems Architecture Group, Technische Universität Berlin, Berlin 10587, Germany (e-mail: chi.c.chi@tu-berlin.de; mauricio.alvarezmesa@tu-berlin.de; b.juurlink@tu-berlin.de).

M. Alvarez-Mesa and T. Schierl are with the Multimedia Communication Group, Fraunhofer Heinrich Hertz Institute, Berlin 10587, Germany (e-mail: mauricio.alvarez.mesa@hhi.fraunhofer.de; thomas.schierl@hhi.fraunhofer.de).

G. Clare, F. Henry, and S. Pateux are with Orange Labs Research and Development, Rennes Atalante Beaulieu, Cesson Sevigné 35512, France (e-mail: felix.henry@orange.com; gordon.clare@orange.com; stephane.pateux@orange.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

This paper is organized as follows. Section II describes the parallelization approaches currently included in the HEVC draft. Section III presents an analysis of the coding efficiency of the different parallelization methods. In Section IV, the parallel implementations are described. Experimental results and their analysis are presented in Section V. Finally, conclusions are drawn in Section VI.

II. Video Codec Parallelization Approaches

A. Previous Parallelization Strategies

Previous video codecs, in particular H.264/AVC, have been parallelized using either frame-level, slice-level, or macroblock-level parallelism. Each of these approaches, however, has some limitations such as limited scalability, significant coding losses, or high memory requirements.

1) *Frame-Level Parallelism:* Frame-level parallelism consists of processing multiple frames at the same time provided that the motion compensation dependences are satisfied [6]. Frame-level parallelism is sufficient for multicore systems with just a few cores. Because it is relatively simple to implement and does not incur coding losses, it has been employed in popular H.264/AVC encoders and decoders [7], [8]. However, frame-level parallelism has a number of limitations. First, the parallel scalability is determined by the lengths of the motion vectors. If, due to fast motion, motion vectors are long, there is little parallelism. Second, the workload of each core may be imbalanced because the frame decoding time can vary significantly. Finally, frame-level parallelism increases the frame rate but does not improve the frame latency.

2) *Slice-level Parallelism:* In H.264/AVC, as in most current hybrid video coding standards, each frame can be partitioned into one or more slices in order to add robustness to the bitstream. Slices in a frame are completely independent from each other [9] and therefore they can also be used for parallel processing. Slice-level parallelism, however, also has a number disadvantages. The first one is that the number of slices is determined by the encoder and, in most cases, encoders use only one slice per frame, resulting in bitstreams with no slice-level parallelism at all. Second, although slices are completely independent from each other, the H.264/AVC deblocking filter can be applied across slice boundaries. Finally, slices reduce the coding efficiency significantly. Due to the these disadvantages, exploiting slice-level parallelism is only advisable when there are a few slices per frame [10].

3) *Macroblock-level Parallelism:* Independent macroblocks inside a frame can be reconstructed in parallel using a wavefront approach [11]. Furthermore, macroblocks from different frames can be processed in parallel provided the dependences due to motion compensation are handled correctly [6]. Entropy decoding, however, can only be parallelized at the frame (slice) level and therefore it has to be decoupled from macroblock reconstruction [12]. Although this approach can scale to a many-core architecture [5], it has some limitations too. First, the decoupling of entropy decoding (ED) and reconstruction increases the memory usage. Furthermore, this strategy only reduces the frame latency for the reconstruction stage but not for the ED stage.

0	1	2	3	12	13	14	21	22	23	24
4	5	6	7	15	16	17	25	26	27	28
8	9	10	11	18	19	20	29	30	31	32
33	34	35	36	41	42	43	47	48	49	50
37	38	39	40	44	45	46	51	52	53	54
55	56	57	58	63	64	65	69	70	71	72
59	60	61	62	66	67	68	73	74	75	76

Fig. 1. Picture divided into nine tiles, showing the scanning order of the CTBs.

B. Parallelization Strategies in HEVC

In order to overcome the limitations of the parallelization strategies employed in H.264/AVC, two tools aiming at facilitating high-level parallel processing have been included in the HEVC draft: WPP and tiles. Both of these tools allow subdivision of each picture into multiple partitions that can be processed in parallel. Each partition contains an integer number of coding tree blocks (CTBs) that may or may not have dependences on CTBs of other partitions. When WPP or tiles are enabled, the bitstream contains entry point offsets (in the slice header) that indicate the start position of each picture partition. This is necessary for each core to immediately access the partition it is assigned to decode.

1) *Tiles:* When tiles are enabled the picture is divided in rectangular groups of CTBs separated by vertical and horizontal boundaries [13]. The number of tiles and the location of their boundaries can be defined for the entire sequence or changed from picture to picture. Tile boundaries, similarly to slice boundaries, break parse and prediction dependences so that a tile can be processed independently, but the in-loop filters (deblocking and SAO) can still cross tile boundaries.

Tiles change the regular CTB scan order to a tiles scan order, of which an example is given in Fig. 1. Constraints are set on the relationship between slices and Tiles. At least one of the following conditions should be true for each slice and tile in a picture: all CTBs in a slice belong to the same tile, or all CTBs in a tile belong to the same slice.

Tiles do not require communication between processors for CTB ED and reconstruction, but communication is needed if the filtering stages operate in the crossing mode. Although a noncrossing mode is specified in which data exchange is not required between processors, it can result in visual artifacts. Additionally, the tiles scan order complicates the raster scan processing typically performed on single-core implementation.

Compared to slices, tiles have a better coding efficiency because tiles allow picture partition shapes that contain samples with a potential higher correlation than slices, and because tiles reduce slice header overhead. But, similar to slices, the rate-distortion loss increases with the number of tiles, due to the breaking of dependences along partition boundaries and the resetting of CABAC probabilities at the beginning of each partition.

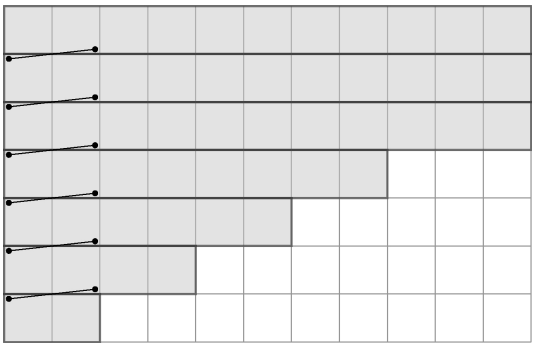


Fig. 2. WPP processes rows of CTBs in parallel, each row starting with the CABAC probabilities available after processing the second CTB of the row above.

2) *Wavefront Parallel Processing (WPP):* When WPP is enabled, each CTB row of a picture is a separated partition [14]. Compared to slices and tiles, however, no coding dependences are broken at row boundaries. Additionally, CABAC probabilities are propagated from the second CTB of the previous row, to further reduce the coding losses (Fig. 2). Also, WPP does not change the regular raster scan order. Because dependences are not broken the rate-distortion loss of a WPP bitstream is small compared to a nonparallel bitstream. Furthermore, a WPP bitstream can be losslessly transcoded to/from a nonparallel bitstream with only an entropy-level conversion [15].

When WPP is enabled, a number of processors up to the number of CTB rows can work in parallel to process the lines. The wavefront dependences, however, do not allow all the CTB rows to start decoding at the beginning of the picture. Consequently, the CTB rows also cannot finish decoding at the same time at the end of the picture. This introduces parallelization inefficiencies (we will refer to them as ramping inefficiencies) that become more evident when a high number of processors is used.

3) *Overlapped Wavefront (OWF):* The ramping inefficiencies of WPP can be mitigated by overlapping the execution of consecutive pictures. When a thread has finished a CTB row in the current picture and no more rows are available, it can start processing the next picture instead of waiting for the current picture to finish. We call this nonnormative technique OWF and it can be used for enhancing the implementation efficiency of WPP [16].

To support overlapped wavefront execution, the motion vectors must be constrained to ensure that when a coding unit (CU) is decoded, all its reference area is available, without requiring that the full reference picture is available. This can be guaranteed by limiting only the maximum downward length of the vertical component of the motion vector. This restriction ensures that the reference area has been decoded, provided the number of CTB row decoding threads is below a certain limit [17]. Vertical motion vector restriction is usually part of the profile and level definition of a video standard. At the time of writing, however, these limits have not yet been defined for the HEVC main profile [18].

Fig. 3 illustrates overlapped frame decoding as well as the relation between the maximum motion vector length and

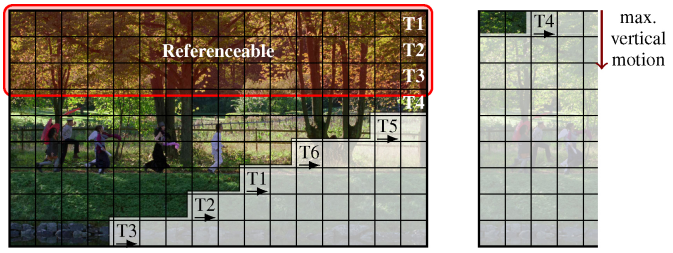


Fig. 3. Frames can be overlapped with a restricted motion vector size, because the reference area is fully decoded.

TABLE I
COMPARISON OF PARALLELIZATION APPROACHES

Properties	Slices	Tiles	WPP/OWF
Coding losses	Very high	High	Low
Boundary artifacts	Yes	Yes	No
Single-core issues	No	Yes	No
Parallel scalability	Medium	Medium	Medium/high
Region of interest	No	Yes	No

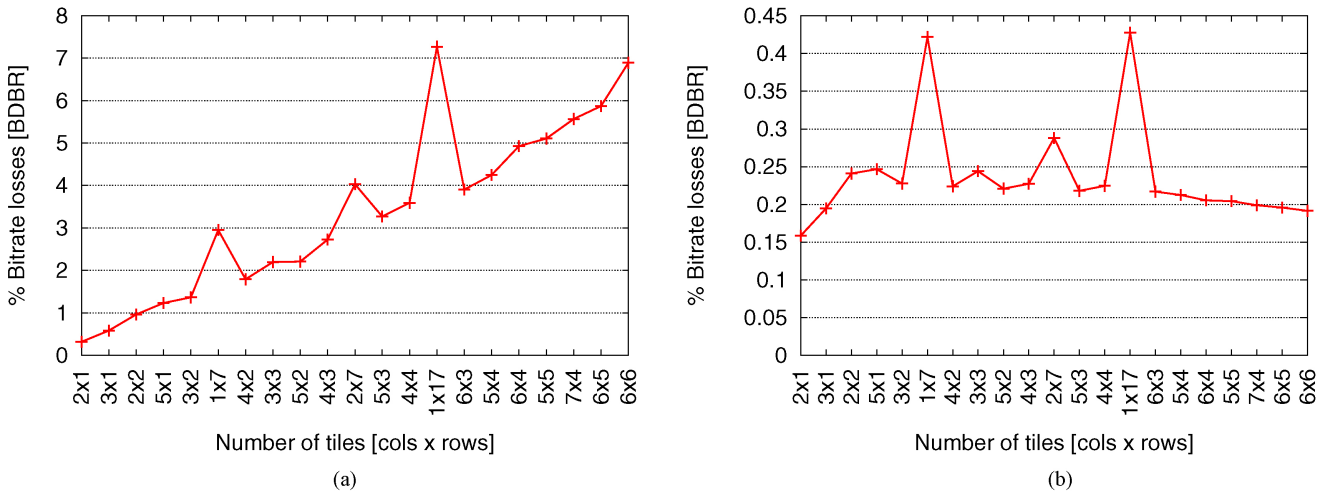
the admissible number of CTB row decoding threads. In this example there are nine CTB rows in a frame and the vertical component of the motion vector is constrained to 1/4th of the picture height. Then, provided there are fewer than six decoding threads, it is ensured that when an CTB is decoded, its reference area is available, before the entire reference picture fully decoded.

4) *Application Use Cases:* It is clear from the previous sections that tiles and WPP have different merits and disadvantages. Table I presents a summary of the main properties of the parallelization approaches.

WPP is generally well suited for the parallelization of the encoder and decoder because it allows a high number of picture partitions with low compression losses. Additionally, it does not introduce artifacts at partition boundaries as is the case for slices and tiles [19]. WPP can also be used for low-delay applications, especially those requiring subpicture delay (also called ultralow delay). In such scenarios, it is needed for the encoder to transmit a picture partition as soon as it has been encoded. This can be achieved by combining WPP with multiple slices or dependent slices [20]. Dependent slices are similar to slices except that no dependences are broken along slice boundaries and the header is much smaller (it inherits most of its characteristics from the parent slice).

Tiles can also be used for a general parallelization of encoder and decoder. The amount of parallelism is not fixed, as in WPP, allowing the encoder to adjust the number of tiles according to its computing resources. Because tiles can be used to divide the picture into multiple rectangles spanning the picture horizontally and vertically, they are better suited for region of interest (ROI) coding. In conversational applications, for example, tiles in combination with a tracking algorithm can be used to dynamically adjust the size and error protection of the ROIs.

In order to simplify the implementation, the HEVC standard does not allow use of tiles and WPP simultaneously in the same compressed video sequence. It may be interesting, how-



(a) (b)

Fig. 4. Coding losses (using weighted YUV-BDBR) of different tile configurations for 1080 videos. (a) Total coding losses. (b) Coding losses per partition.

ever, to allow some combination of these tools in the future. For instance, it could be necessary to divide an ultrahigh-resolution video into subpictures using tiles with WPP inside each subpicture, to enable real-time encoding/decoding.

III. Coding Efficiency Analysis

The parallelization approaches considered in HEVC rely on creating picture partitions. Coding losses may appear due to breaking dependences for prediction, CABAC context modeling, and/or slice header overhead. In general, having more partitions leads to higher compression losses. In this section we present a quantitative analysis of the coding efficiency of the different parallelization tools.

A. Choosing the Number of Picture Partitions

For slices and tiles, the number of partitions can be freely chosen by the encoder. For WPP they are fixed to one partition per CTB row. In order to have a common baseline for slices and WPP, we selected a configuration with one picture partition per CTB row. This approach results in 17, 25, and 34 picture partitions for 1080p, 1600p, and 2160p resolutions, respectively.

For OWF the same picture partitioning configuration as WPP is used. Additionally, a restriction in the maximum length of the vertical motion vector is applied. This limit is defined as 1/4th of the picture height.

In the case of tiles, there is more freedom since tiles can have row and column partitions. We evaluated tiles configurations with different number of rows and columns (N columns $\times$ M rows) in order to find the best partitions in terms of amount of parallelism and coding losses.

For tractability, we performed the tiles analysis only for 1080p resolution. Fig. 4 shows the coding losses for the different tiles configurations. (The details of the input videos are presented in Section III-B.) The coding losses increase with the total number of tiles, but there are configurations that have more coding losses than others, especially those with only row partitions (1×7 and 1×17). To better appreciate the coding losses of different tiles shapes, in Fig. 4 we show the coding losses per partition. According to these results, tiles shapes that are as square as possible have less coding losses. Square tiles have the lowest perimeter to area ratio and, therefore, break less dependences than nonsquare Tiles.

For the performance evaluation we selected the tiles configurations with the closest matching degree of parallelism compared to slices and WPP. The selected tiles configurations are 6×3, 6×4 and 7×5 for 1080p, 1600p, and 2160p resolutions, respectively.

B. Coding Efficiency

Because support for parallelism will be useful mainly for HD resolutions, we selected videos for 1080p (1920×1080), 1600p (2500×1600p), and 2160p (3840×2160p) resolutions. For 1080p and 1600p, we used the test sequences described in the HEVC common conditions [21]. For 2160p resolution, we use two videos from the SVT High Definition Multi Format Test Set [22].

From the HEVC, common conditions we selected the random access high-efficiency (RA-HE) settings, which achieves the highest compression efficiency and includes, among others, CABAC entropy coding, SAO filter, and ALF [21]. For the encoder we used HM-4.1, which is the same version of HM used later for the experiments described in Section V.

For comparing coding losses we computed the Bjϕntegaard Delta Bit Rate (BDBR) using a weighted average for luma and chroma components ($0.75Y + 0.125U + 0.125V$) for all the videos [23]. As a baseline we used a configuration without picture partitions (i.e., one slice and one tile per picture, no WPP). All the videos were encoded for four quantization parameter (QP) values (22, 27, 32, and 37). Table II shows the main properties of the test sequences, including resolution, frame rate, number of frames and the resulting bitrate (in kb/s) and the combined YUV-PSNR (in dB).

Table III shows the coding losses for each one of the three picture partition strategies. One slice per CTB row (1 slice/row) exhibits the largest coding penalty, 7.99% on average for the three resolutions, as slices break ED and

TABLE II

BITRATE (IN KB/S) AND WEIGHTED PSNR (IN DB) FOR ALL THE ENCODED VIDEO SEQUENCES

Video	Resolution	Frames	QP22		QP27		QP32		QP37	
			Bitrate	YUV-PSNR	Bitrate	YUV-PSNR	Bitrate	YUV-PSNR	Bitrate	YUV-PSNR
BasketballDrive	1080p50	500	17 846	40.69	6244	39.03	2933	37.28	1539	35.50
BQTerrace	1080p60	500	40 238	39.15	8263	37.22	2823	35.88	1295	34.36
Cactus	1080p50	500	18 965	39.47	6181	37.94	2934	36.23	1514	34.37
Kimono	1080p24	241	4741	42.57	2184	40.73	1068	38.65	537	36.60
ParkScene	1080p24	240	7398	40.99	3199	38.56	1463	36.18	678	34.03
NebutaFestival	1600p60	300	220 079	39.52	95 871	34.74	29 830	31.48	8133	30.06
PeopleOnStreet	1600p30	150	32 978	41.62	15 839	38.96	8326	36.37	4666	34.01
SteamLocomotive	1600p60	302	24 477	43.04	6747	41.78	2980	40.64	1475	39.25
Traffic	1600p30	150	13 005	42.19	5291	39.84	2546	37.55	1322	35.29
CrowdRun	2160p50	500	152 516	38.23	48 316	36.33	21 986	34.63	11 340	32.84
ParkJoy	2160p50	500	193 158	38.14	69 015	35.73	29 866	33.60	13 931	31.64

TABLE III

COMPARISON OF PICTURE PARTITION APPROACHES

	VIDEO	1 SLICE/ROW		WPP/OWF		(COLS×ROWS) TILES	
		Partitions	BDBR	Partitions	BDBR	Partitions	BDBR
1080p	*BasketballDrive*	17	11.483	17	1.732	6×3	4.62
	BQTerrace	17	6.878	17	0.942	6×3	3.544
	Cactus	17	8.135	17	1.663	6×3	3.295
	Kimono	17	10.756	17	1.763	6×3	5.094
	ParkScene	17	6.936	17	1.006	6×3	2.968
1600p	*Nebuta*	25	5.279	25	0.99	6×4	4.157
	PeopleOnStreet	25	4.308	25	1.049	6×4	2.004
	SteamLocomotive	25	22.229	25	1.827	6×4	11.299
	Traffic	25	7.011	25	1.023	6×4	3.056
2160p	*CrowdRun*	34	6.043	34	0.538	7×5	2.176
	ParkJoy	34	4.83	34	0.621	7×5	2.155

Numbers on the table represent coding losses compared to having one slice per picture (1 tile, and no WPP) using average YUV-BDBR.

prediction dependences, and have a complete slice header per partition. Tiles, with an average of 3.73% coding losses, are more efficient than slices. WPP, with an average of 1.07%, has the lowest coding losses of all the partition strategies. Because WPP allows crossing partitions for both ED and prediction, the small remaining losses are due to the entry points and the reduced CABAC contexts training. In the case of OWF no additional coding losses compared to WPP were observed for any of the tested sequences. Some coding losses may appear in videos that exhibit fast vertical motion due to the restriction of the vertical motion vectors size.

IV. IMPLEMENTATION

For tiles, WPP, and OWF, a pipelined decoder organization is used as illustrated in Fig. 5. It consists of three pipeline stages: *parse*, *issue*, and *output*. Each of these stages is performed by a different thread. The parse thread performs emulation prevention, high-level syntax parsing, and allocates an entry in the decoded picture buffer. When it detects a slice network abstraction layer (NAL) unit [9], the NAL unit payload is sent to the input queue of the issue thread. The issue thread creates for each picture partition in the payload a work unit and sends this to the shared input queue of the decoding threads, indicated by D_i in the figure. The output thread reorders the pictures and manages the picture buffer. With such an organization of the parallel decoder, the parsing and issuing of the next picture can be performed in parallel with the processing of the current picture.

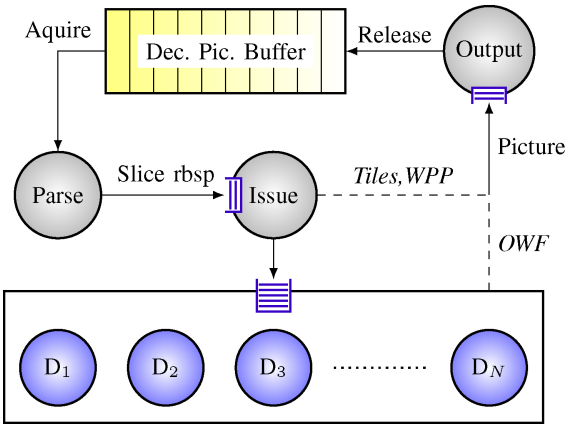


Fig. 5. General decoder architecture.

When a decoding thread D_i is free, it tries to acquire a work unit from the shared queue and starts processing. In the case of tiles and WPP, the issue thread waits until all work issued to the queue has been processed and the decoding threads have signaled completion. Thereafter, the issue thread informs the output thread of completion. In the OWF approach, on the other hand, the decoding thread that finishes the last CTB row of a picture signals the output thread. This allows the issue thread to fill the decoder queue with partitions of the

next picture to allow overlapping the execution of consecutive frames.

A. Tiles Decoding

Each tile can be entropy decoded and reconstructed in parallel. The filtering stages, however, can cross tile boundaries and cannot be performed straightforwardly in the tile entropy decode and reconstruction loop. Performing the filtering in the tile decoding loop would require saving and restoring border data, and complex control logic for detecting and handling tile boundaries conditions when multiple filters are used.

An alternative is to perform the filters in separated passes. In HEVC, the filtering stages themselves are parallel, meaning that they can be applied in parallel for each CTB, but one after the other, meaning that they have to be performed in separate passes. The parallel tiles decoder is then implemented in five parallel phases: 1) entropy decode and reconstruction; 2) vertical edge filtering; 3) horizontal edge filtering; 4) SAO; and 5) ALF. These phases are separated with barrier synchronizations.

Barriers are implemented by letting the issue thread wait for the work units issued to the shared input queue of the decoder threads to complete. For the parallel filtering phases, instead of issuing a work unit for each tile, only one work unit is submitted to the shared queue for each decoding thread. The decoding threads employ an atomic counter to distribute the CTBs that need to be filtered. Using this scheme the filter task sizes are independent of the tiles configuration and a smaller task size can be chosen without increasing synchronization overhead. In our implementation the atomic counter is incremented by eight and the decoding threads process eight consecutive CTBs at a time, starting from the previous CTB count of the atomic counter. An exploration showed that processing eight CTBs at a time strikes a good balance between parallelism, locality, and synchronization overhead. When the atomic counter exceeds the number of CTBs in a picture, the decoding threads signal their completion.

B. Wavefront Decoding

In the wavefront decoders (WPP and OWF), each decoding thread processes a CTB row of the picture. The wavefront dependences are maintained between the decoding threads by using a lock protected counter for each CTB row. This counter indicates the number of CTBs that have been processed in the associated CTB row. Each decoding thread checks the progress of the thread processing the previous CTB row before decoding the next CTB in its row. To maintain the wavefront dependences, the thread processing the previous row must have processed the CTB top-right of the current CTB. In other words, at any time the thread processing the previous row must have processed two CTBs more than the thread processing the current row.

For both WPP and OWF, additional memory optimizations can be applied. First, each decoding step can be performed in a single CTB decoding pass for improving data locality. Moreover, for OWF this is required in order to make the reference area available directly after decoding a CTB. In HEVC this means that, in addition to the entropy decode and the reconstruction steps, all the filters (deblocking filter, the SAO filter and the ALF) must also be performed in the CTB decoding loop. Due to pixel dependences, however, the filtering steps cannot be straightforwardly performed on the current CTB, but must be delayed and performed in the order depicted in Fig. 6.

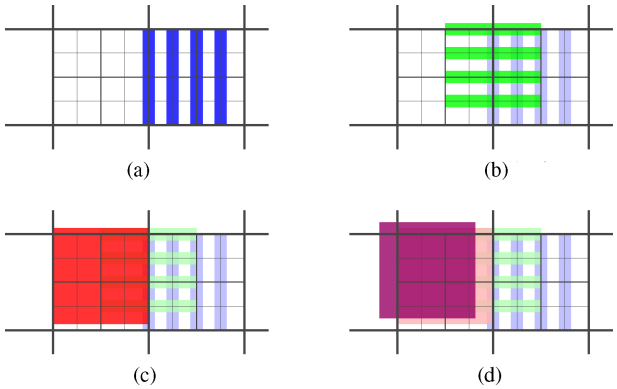


Fig. 6. Order and delay of filtering steps necessary due to dependences between pixels. (a) Vertical edges. (b) Horizontal edges. (c) SAO. (d) ALF.

First the vertical edges of the current CTB need to be filtered. In contrast to H.264/AVC, the deblocking of the horizontal edges cannot be performed on all edges of the current CTB, because it requires filtered vertical edges as input. To satisfy this dependency, horizontal edge filtering is performed as the second step and, furthermore, delayed by half a CTB, as illustrated in Fig. 6(b). In this figure the fact that the vertical edges have been filtered is indicated by using a lighter color than the color used in Fig. 6(a). In the next step the SAO filter is performed on the deblocked pixels, and is delayed by one full CTB horizontally and four pixels vertically to fulfill the dependences. Technically, both the deblocking filter and SAO filter can be postponed less, but this would introduce additional overhead (more `if`-statements in the code). In the final step the ALF is applied on the SAO filtered pixels. The ALF employs two filter shapes for each pixel, an 11×5 cross and a 9×5 snowflake filter shape. Because ALF uses the same filter shapes for the chroma components and uses the same filter coefficients, it has to be delayed by 10 pixels horizontally and 4 pixels vertically, in the case of YCbCr 4:2:0. We choose to delay the filter 12 pixels for implementation efficiency, because (in HM 4.1) ALF uses the same coefficients for a 4×4 pixel block.

The second optimization consists in the removal of intermediate picture buffers for filtering without introducing complex border exchange. In HEVC, the SAO filter and the ALF use the output of the previous filter, and for each pixel adjacent unfiltered pixels are used to derive the filtered pixel. Therefore, the filtered pixel cannot overwrite the unfiltered pixel as it is still required for filtering adjacent pixels. Typically this is addressed by storing the filtered pixels in a different picture buffer, illustrated in Fig. 7(a), at the cost of reduced cache locality because the working set becomes at least twice as large (Tiles reuse the deblock output picture for the ALF output picture).

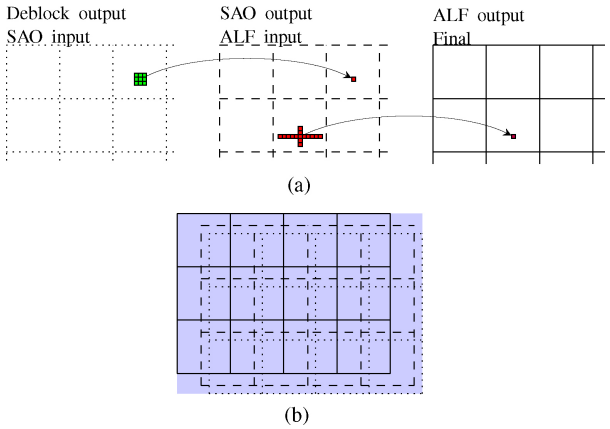


Fig. 7. Intermediate picture buffers. (a) Separated for tiles. (b) Combined for WPP and OWF.

TABLE IV
EXPERIMENTAL SETUP

System		Software	
Processor	Intel X5680	HEVC encoder	HM-4.1-r1527
μarchitecture	Westmere	Boost C++	1.46.1
Sockets	2	Compiler	gcc 4.6.1
Cores/socket	6	Opt. level	-O3
Frequency	3.33 GHz	Kernel	3.0.0-16
L3-cache	12 MB/socket	OS	Kubuntu 11.10
SMT	Disabled	Measurement	perf, PAPI
TurboBoost	Disabled	Tools	taskset, time

In OWF and WPP, because of the wavefront order of processing, intermediate picture buffers that are required for storing the intermediate results of the filters can be combined with the final picture. This combination of picture buffers is illustrated in Fig. 7(b). Instead of using a separate picture buffer, the intermediate picture buffers can point to the same memory space but with an offset determined by the pixel filter sizes. The filters can employ these *virtual* picture buffers as if there were separate, using a larger stride. The additional memory requirements are small, with only three extra pixel lines in height and 6 pixels columns in width. To keep the CTBs aligned to 16 bytes, however, a horizontal offset of 8 pixels is used per picture buffer.

V. EXPERIMENTAL EVALUATION

The parallel HEVC decoders have been implemented using the HM 4.1 codebase as the starting point. Multithreading has been performed using the C++ Boost libraries, which offer a convenient C++ wrapper around platform-dependent threading libraries such as Pthreads. For the experiments, the sequences encoded with 1 slice per frame are used as the baseline against which speedup will be measured.

The system employed to measure performance is a cache-coherent shared memory system with two Intel Xeon X5680 processors, each with six cores. More details of the hardware/software environment are listed in Table IV.

A. Improved Baseline Implementation

To give a more realistic view of the scalability and memory bandwidth requirements of HEVC for both the tiles and WPP approaches, two general optimizations have been applied. First, the deblocking filter is made compliant to the one in HM 6.0. In HM 6.0, the horizontal edge filter uses the pixels produced by the vertical edge filter for the filter strength decisions, instead of the unfiltered picture. This eliminates one intermediate picture copy step. The second optimization has been applied to the CTB data structure. For each CTB in a picture and each picture in the reference picture buffer, the transform coefficients (24 kb per CTB) and IPCM related data structures (12 kB per CTB) are stored. The lifetime of these data structures is only one CTB, and can be reused for each CTB. Therefore, only one of each of these data structures is allocated per decoding thread instead. This not only decreases the memory requirements significantly, but also the memory bandwidth requirements due to improved cache locality.

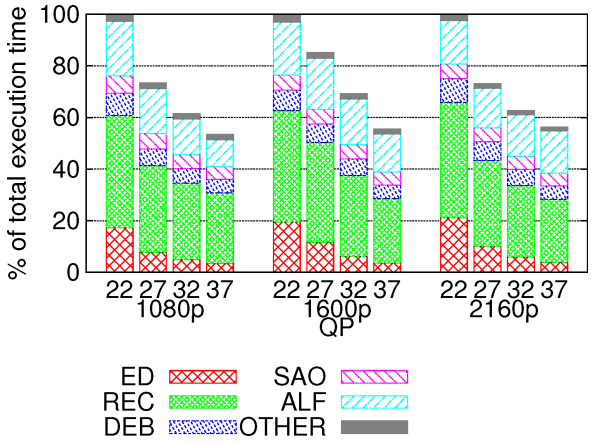


Fig. 8. Profiling of the sequential decoder using one slice per frame. Average over all videos in each resolution. Baseline is QP 22 for each resolution.

These optimizations improve the performance of the decoder by 11% on average for the three considered resolutions compared to the HM 4.1 decoder. The performance of the parallel decoders will be compared to the performance of this improved sequential decoder.

For moving the ALF in the CTB decoding loop, described in Section IV-B, the CTB-based ALF syntax [24], which is accepted in HM 6.0 is required. We have implemented a similar syntax on top of the HM 4.1 code base, in which we limited the ALF on/off signaling to the CTB level.

B. Profiling Analysis

Fig. 8 breaks down the execution time of the sequential HEVC decoder in time spent on ED, reconstruction (REC), deblocking filter (DEB), the SAO filter, and the ALF. OTHER consists mainly of initializing the CTB data structure, bitstream emulation prevention, and high-level syntax parsing. An execution profile is provided for each combination of QP and resolution, and have been normalized to that of QP 22 (so QP 22 is 100%). The profiles show that most of the time, 44%, is spent in the reconstruction stage, which includes coefficient transform, intra prediction, and motion compensation. For all resolutions, increasing the QP reduces the total execution time, to around 55% for QP 37. Increasing the QP reduces the execution time mainly for ED stage, but also for the

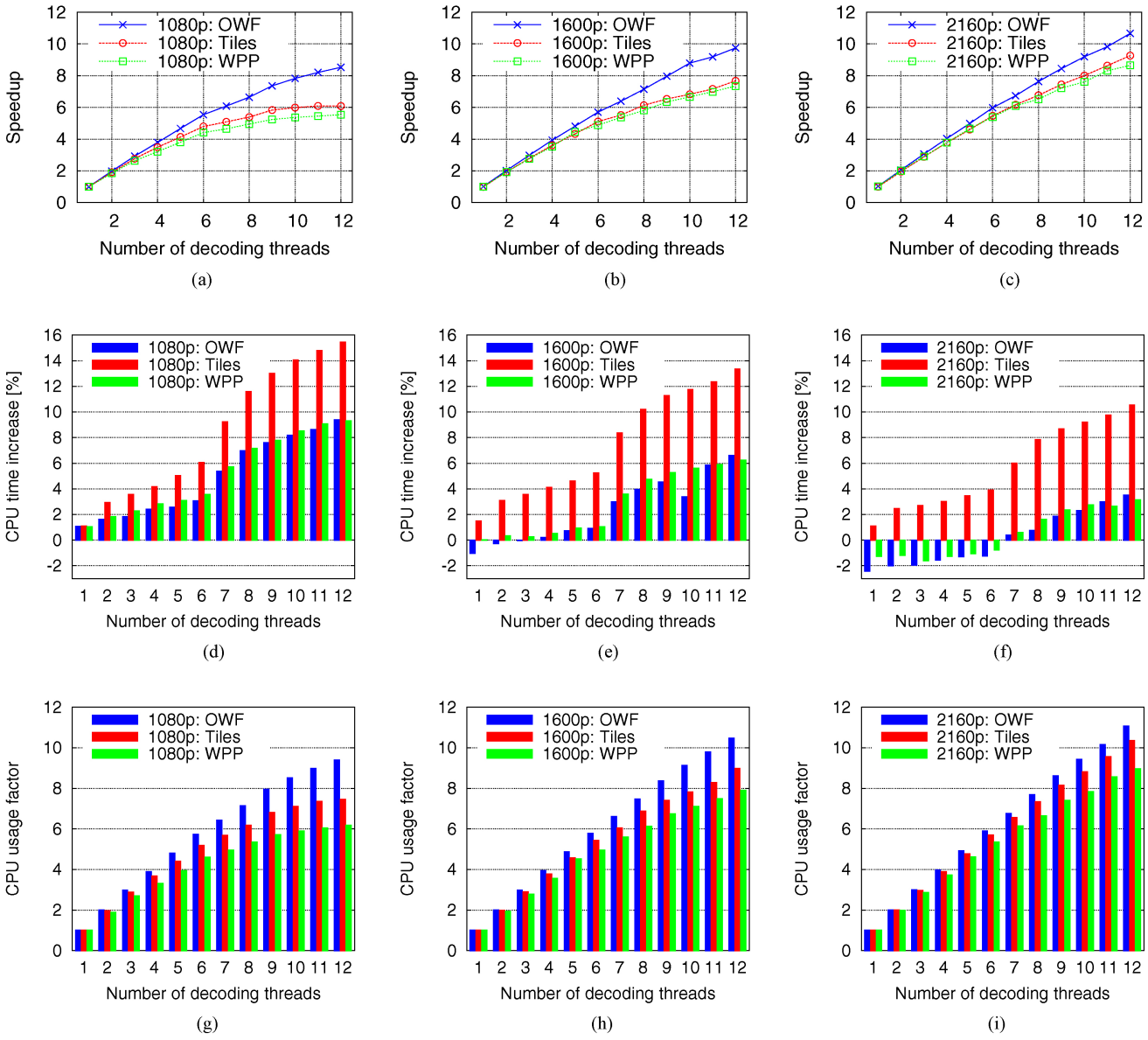


Fig. 9. Speedup, CPU time increase, and CPU usage factor, for 1080p, 1600p, and 2160p. (a) Speedup for 1080p. (b) Speedup for 1600p. (c) Speedup for 2160p. (d) CPU time increase 1080p. (e) CPU time increase 1600p. (f) CPU time increase 2160p. (g) CPU usage factor 1080p. (h) CPU usage factor 1600p. (i) CPU usage factor 2160p.

reconstruction stage. Of the three in-loop filters, the ALF is the most time-consuming one. The SAO and the deblocking filter have similar complexity at high QP, while the deblocking filter is slightly more complex at low QP.

C. Parallel Scalability Analysis

In the experiments, we pin decoding threads to cores to reduce the influence of the OS scheduling policy and improve the reproducibility of the results. The experiments are performed for all sequences that have been listed in Table II and using 1 to 12 decoding threads. The speedup as a function of the number of decoding threads is depicted in Fig. 9(a)–(c).

The speedup results show that for each resolution, OWF achieves the highest speedup followed by tiles and WPP. The results also show that, in general, the speedup increases with the resolution, albeit slightly more for tiles and WPP than OWF. The speedup of OWF scales almost perfectly (linearly) up to 12 cores for 2160p, and up to 6 cores for 1080p. WPP scales worse because of the parallelism ramping inefficiencies.

The speedup of tiles, even though it has no dependences in each parallel phase, is lower than the speedup achieved by OWF at almost every core count. Furthermore, the difference increases with the number of decoding threads. Fig. 10 shows the speedup for the tiles decoder separated for each parallel phase. The deblocking filter exhibits the lowest scalability specially at 1080p and 1600p, mainly because of the small amount of work in each filtering pass. The speedup of SAO is limited for core counts bigger than 4 also because of its fine granularity. Entropy decoding and reconstruction, and ALF, exhibit the highest speedups, although with big differences between resolutions caused mainly by load unbalance issues. At

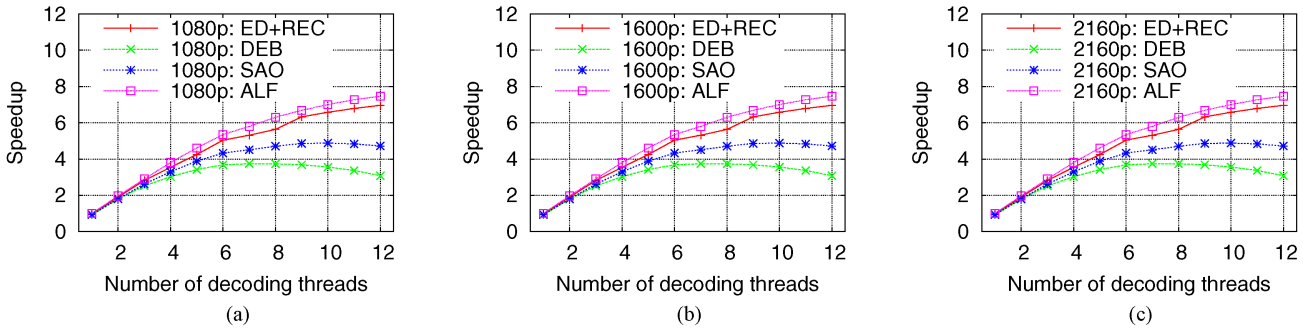


Fig. 10. Speedup for each frame pass in tiles decoder. (a) 1080p. (b) 1600p. (c) 2160p.

2160p resolution there are 2.92 tiles per processor, compared to 1.5 for 1080p.

The speedup results can be explained in more detail by examining the CPU time increase and CPU usage factor results shown in Fig. 9(d)–(i). The relation between the speedup, CPU time increase, and CPU usage factor is given by

$$Speedup = \frac{S_T}{P_T} = \frac{S_T}{\frac{S_T \times CPU_{IF}}{CPU_{UF}}} = \frac{CPU_{UF}}{CPU_{IF}} \quad (1)$$

where S_T is the sequential execution time, P_T is the parallel execution time, CPU_{UF} is the CPU usage factor, and CPU_{IF} is the CPU inefficiency factor. The CPU time increase is the percentile form of the CPU inefficiency factor. We introduced the CPU_{UF} and CPU_{IF} metrics, because they represent the *scalability* and *(in)efficiency* of the parallel solution, and can be easily derived from the CPU time (user + system time) measured by the `time` command.

The CPU time increase corresponds to the increase in CPU time compared to the baseline, using one decoding thread for the sequences encoded with 1 slice per frame. The CPU time is the aggregated active thread time spent. A thread is active only when a thread is scheduled by the operating system. Ideally this metric should stay constant when increasing the number of threads as increases in CPU time represents inefficiencies in the parallel hardware, algorithm, and/or implementation. A higher CPU time indicates either increased number of executed instructions or fewer instructions per cycle (IPC). Extra instructions can be required, for example, due to polling synchronization or excessive blocking and waking up of threads. We have measured no significant increase in the number of executed instructions, and therefore the increase in CPU time is mainly caused by a decreasing IPC. With higher number of threads, contention on shared resources, such as shared caches and memory controllers, and cache coherence misses result in IPC decreases. A more in-depth analysis of IPC will be performed in Section V-E.

The CPU time results show that tiles have a much larger CPU time increase compared to WPP and OWF. This is caused by implementing the in-loop filters in separate picture passes for Tiles. Having additional passes results in more requests to the shared L3 cache and the off-chip memory controller as the picture does not fit in the private L2 caches. Due to contention on these resources when increasing the number of threads, the individual requests take more time. For WPP and OWF all the kernels are performed in one pass. Most of the pixels touched in each kernel are almost immediately reused in the next stage, and are highly likely to remain in the private caches. Although it is possible to implement in-loop filters on a CTB basis for tiles to reduce the memory bandwidth requirements, such an implementation has high implementation cost and, especially, complexity as mentioned previously in Section IV-A.

The CPU usage factor shows the average aggregated CPUs used during execution. This factor is derived by dividing the CPU time by the parallel execution time. Ideally this is equal to the number of used threads and shows the scalability of the parallel algorithm. The figures show that OWF in all cases has the highest CPU usage factor, followed by tiles, and finally WPP. With higher resolutions the scalability is higher in general as there is more parallelism available.

The CPU usage factor is, however, not ideal for any of the strategies. For OWF this is caused by dependency stalls. Dependency stalls originate from the variability in the CTB execution times. Wavefront execution must respect the dependences to the top right CTB, which results in dependency stalls when the difference in CTB execution times between rows become too large. With higher resolutions there is more tolerance to these execution time differences and, therefore, have better scalability. WPP has in addition to the dependency stalls also the ramping stalls, which occur at the beginning and end of decoding a picture when not all decoding threads can be active. Tiles, while having completely independent parallel phases, also does not scale ideally. This originates from a load balancing problem, in which not every decoding thread gets the same amount of work to process. For instance in 1080p there are 18 tiles for the entropy and reconstruction phase. This does not divide evenly when there are, e.g., four decoding threads. Furthermore, similar to CTBs, tiles vary in execution time which exacerbates the problem.

D. Throughput

The throughput in frames per second (f/s) for each resolution, QP, and parallelization approach is listed in Table V. The performance is shown for 1, 6, and 12 decoding threads. The table shows that 6 decoding threads are sufficient for every parallelization approach to achieve real-time 1080p50 on average, while OWF is also very close to achieving this for QP 22. The average performance for 2160p resolution sequences is still significantly below the real-time requirements, even with

TABLE V

PERFORMANCE IN FRAMES PER SECOND FOR DIFFERENT QP VALUES

Res.	Thr.	QP22			QP27			QP32			QP37			AVG		
		WPP	Tiles	OWF	WPP	Tiles	OWF	WPP	Tiles	OWF	WPP	Tiles	OWF	WPP	Tiles	OWF
1080p	1	8.8	8.7	8.8	11.5	11.4	11.5	13.7	13.8	13.7	15.9	16.0	15.9	12.5	12.5	12.5
1080p	6	40.2	42.9	49.3	50.8	55.3	64.4	60.3	65.7	75.6	68.2	75.1	87.0	54.9	59.7	69.1
1080p	12	50.9	57.6	78.8	64.2	70.7	99.3	75.0	81.5	115.0	84.9	90.6	129.7	68.7	75.1	105.7
1600p	1	4.0	4.0	4.1	4.9	4.8	4.9	5.8	5.7	5.8	7.0	6.9	7.0	5.4	5.3	5.5
1600p	6	19.8	20.6	23.0	23.8	24.6	27.7	27.6	29.1	32.6	33.0	34.9	38.7	26.0	27.3	30.5
1600p	12	29.9	31.6	39.1	35.6	37.2	47.4	41.7	43.1	56.2	48.7	50.4	65.1	39.0	40.6	51.9
2160p	1	1.5	1.5	1.6	2.1	2.1	2.2	2.5	2.4	2.5	2.8	2.7	2.8	2.2	2.2	2.3
2160p	6	8.1	8.5	9.1	11.2	11.3	12.5	13.1	13.1	14.4	14.6	14.6	16.0	11.8	11.9	13.0
2160p	12	13.2	14.8	16.0	17.9	19.2	22.2	21.1	22.0	26.1	23.3	24.2	28.7	18.8	20.1	23.3

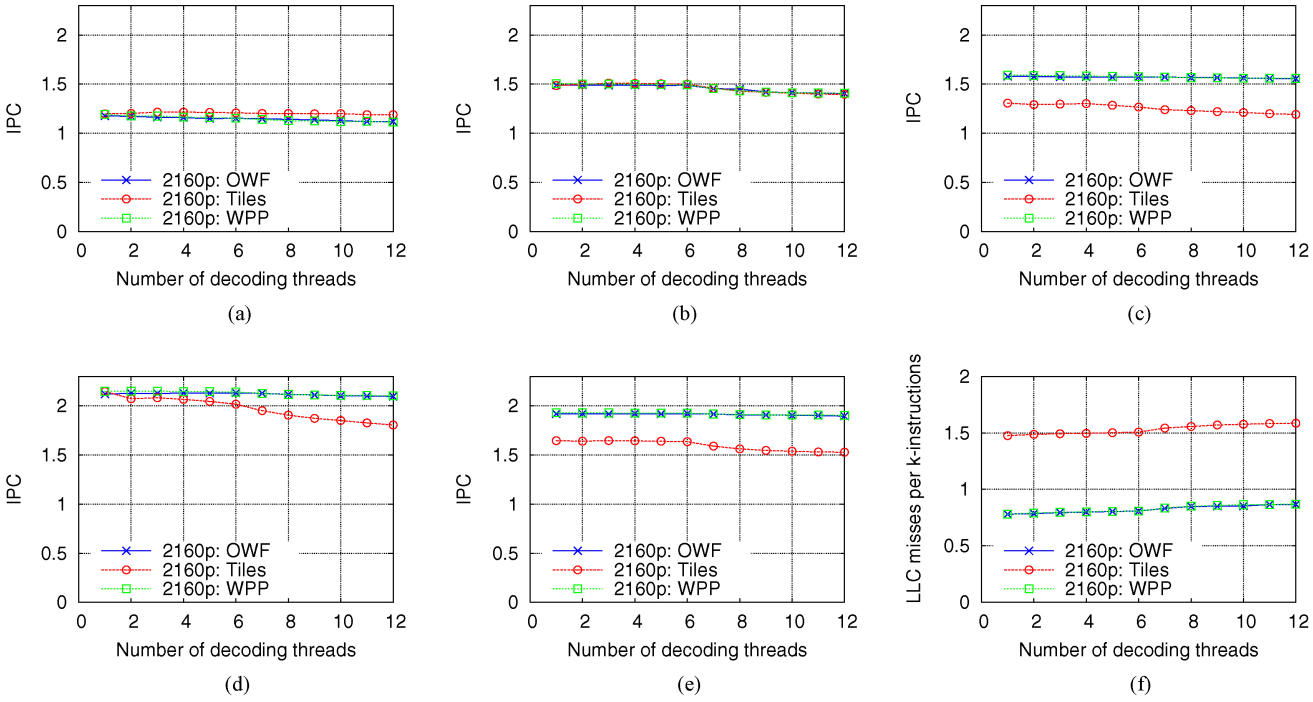


Fig. 11. Performance analysis for 2160p resolution: IPC for different kernels and LLC misses per thousand instructions. (a) ED. (b) REC. (c) DEB. (d) SAO. (e) ALF. (f) LLC misses.

12 decoding threads for OWF. Single-threaded performance, however, can be improved by performing platform independent optimizations as well as using the SIMD extensions included in almost all modern processors.

E. Performance Analysis

Finally, we have analyzed the effect of the parallelization strategies on each individual kernel and the off-chip bandwidth requirements. In Fig. 11(a)–(e) the IPC of each kernel is plotted against the number of decoding threads. Fig. 11(f) shows the last-level-cache (LLC) misses per thousand instructions. For up to six cores the number of LLC misses directly translate in to memory bandwidth. For higher than six cores this also includes the intersocket coherence traffic. For brevity, the figures only shows the results for 2160p as the other resolutions show similar results.

For ED the overall IPC is low due to the nature of the CABAC algorithm. When increasing the number of decoding threads the IPC for tiles stay constant while for OWF and WPP it decreases slightly. This is because for tiles, ED is completely independent, while for WPP and OWF contexts are selected based on the entropy decoded syntax elements of the top neighboring row. The extra latency of fetching this data reduces the IPC, because this cannot be hidden with other instructions due to the low amount of instruction-level parallelism.

For reconstruction the three approaches are performing similar. With higher number of decoding threads the IPC decreases because of contention to the shared cache and memory controller for reading the reference data. While the tiles approach has better motion compensation bandwidth characteristics, this will only be apparent in multicore architectures without a shared cache and private cache to cache transfers.

The IPC of the three in-loop filters and the LLC misses results show clearly the advantage of performing the filters in the CTB decode loop. Because each filter kernel reuses the

pixel data produced of the previous kernel, this data remains in the private caches for OWF and WPP. For tiles, however, each filter is implemented using a separate pass. This approach has low cache locality as the entire picture must be processed completely before the pixels can be reused. Because the data cannot remain in the private caches more shared cache and off-chip requests are required, leading to overall reduced performance. By performing the tiles in-loop filtering on the CTB decoding loop the LLC misses can be reduced up to 45%.

VI. Conclusion

The upcoming video coding standard HEVC is targeting not only even higher compression rates than current standards, but also being designed with high-level parallel processing capabilities. Different parallelization strategies were proposed and are included in the current draft. The main proposals are slices, WPP, and Tiles. All these techniques share in common the idea of creating picture partitions that can be processed in parallel by multiple threads/cores in a parallel system. They differ, however, in terms of data dependences, rate-distortion performance, and parallel scalability. Until now they were compared only in terms of coding efficiency, but a detailed performance comparison of all of them had not yet been performed.

In this paper, we filled this void by presenting a detailed performance comparison of the main approaches, namely WPP and Tiles. We also presented a novel approach called OWF, that can be implemented on top of WPP, and achieves higher performance and scalability than both WPP and Tiles. For implementing OWF it is only needed to restrict the downward motion vector size, which would probably be present in the specification of HEVC profiles and levels as it had happen with H.264/AVC. OWF attains higher performance than WPP because consecutive pictures can be decoded simultaneously, which in turn implies that there is a constant amount of parallelism.

We implemented a general parallel HEVC decoder that supports multiple parallelization strategies and evaluated it on a parallel platform with 12 cores (2 sockets with 6 cores each) running at 3.33 GHz. Comparing WPP to tiles, our experiments show that the tiles approach achieves slightly higher performance than WPP (7% higher on average over all resolutions at 12 cores). In the presented implementation, however, WPP has higher memory bandwidth efficiency than Tiles. Implementing a memory bandwidth optimized tiles decoder can reduce the memory bandwidth requirements significantly, but is paired with high implementation complexity.

The proposed OWF decoder attains the highest performance (28% higher on average than Tiles) and is able to achieve real-time performance for 1080p50 videos, but only 25.4 f/s for 2160p. It needs to be mentioned, however, that the single threaded performance can be increased by using SIMD instructions. This will allow achieving 4K real-time decoding at 50 or 60 f/s, or, alternatively, using fewer cores to obtain real-time performance for lower resolutions.

References

[1] G. J. Sullivan and J.-R. Ohm, "Recent developments in standardization of high efficiency video coding (HEVC)," in *Proc. SPIE*, Aug. 2010, p. 77980V.

[2] *Advanced Video Coding for Generic Audiovisual Services*, ITU-T Rec. H.264 and ISO/IEC 14496-10 (MPEG-4 AVC), 2003.

[3] ITRS, "International Technology Roadmap for Semiconductors 2011 update system drivers," 2011.

[4] S. Borkar and A. A. Chien, "The future of microprocessors," *Commun. ACM*, vol. 54, pp. 67–77, May 2011.

[5] B. Juurlink, M. Alvarez-Mesa, C. C. Chi, A. Azevedo, C. Meenderinck, and A. Ramirez, *Scalable Parallel Programming Applied to H.264/AVC Decoding*. Berlin, Germany: Springer, 2012.

[6] C. Meenderinck, A. Azevedo, M. Alvarez, B. Juurlink, and A. Ramirez, "Parallel scalability of video decoders," *J. Signal Process. Syst.*, vol. 57, pp. 173–194, Nov. 2009.

[7] X264 Developers. (2011). *x264. A Free H.264/AVC Encoder* [Online]. Available: http://www.videolan.org/developers/x264.html

[8] FFmpeg Developers. (2011). *Libavcodec H.264 Decoder* [Online]. Available: http://ffmpeg.org

[9] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.

[10] M. Roitzsch, "Slice-balancing H.264 video encoding for improved scalability of multicore decoding," in *Proc. 7th ACM IEEE Int. Conf. Embedded Softw.*, Oct. 2007, pp. 269–278.

[11] E. B. V. der Tol, E. G. T. Jaspers, and R. H. Gelderblom, "Mapping of H.264 decoding on a multiprocessor architecture," in *Proc. SPIE*, 2003, pp. 707–718.

[12] C. C. Chi and B. Juurlink, "A QHD-capable parallel H.264 decoder," in *Proc. Int. Conf. Supercomput.*, 2011, pp. 317–326.

[13] A. Fuldseth, M. Horowitz, S. Xu, and M. Zhou, "Tiles," Tech. Rep. JCTVC-E408, Mar. 2011.

[14] F. Henry and S. Pateux, "Wavefront parallel processing," Tech. Rep. JCTVC-E196, Mar. 2011.

[15] G. Clare and F. Henry, "An HEVC transcoder converting non-parallel bitstreams to/from WPP," Tech. Rep. JCTVC-J0032, May 2012.

[16] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, V. George, and T. Schierl, "Improving the parallelization efficiency of HEVC decoding," in *Proc. IEEE ICIP*, to be published.

[17] M. Alvarez-Mesa, V. George, T. Schierl, and B. Juurlink, "Improving parallelization efficiency of WPP using overlapped wavefront," Tech. Rep. JCTVC-J425, Jul. 2012.

[18] B. Bross, W.-J. Han, G. J. Sullivan, J.-R. Ohm, and T. Wiegand, "High efficiency video coding (HEVC) text specification draft 8," Tech. Rep. JCTVC-J1003, Jul. 2012.

[19] J. Viéron and J.-M. Thiesse, "On tiles and wavefront tools for parallelism," Tech. Rep. JCTVC-I0198, May 2012.

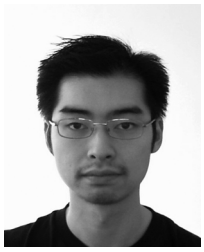
[20] T. Schierl, V. George, A. Henkel, and D. Marpe, "Dependent slices," Tech. Rep. JCTVC-I0229, Apr. 2012.

[21] F. Bossen, "Common test conditions and software reference configurations," Tech. Rep. JCTVC-F900, Dec. 2011.

[22] L. Haglund, "The SVT high definition multi format test set," Tech. Rep., Sveriges Television, Feb. 2006.

[23] G. Bjøntegaard, "Calculation of average PSNR differences between RD-curves," Tech. Rep. VCEG-M33, ITU-T Video Coding Experts Group (VCEG), 2001.

[24] C.-Y. Chen, C.-Y. Tsai, C.-M. Fu, Y.-W. Huang, S. Lei, I. S. Chong, M. Karczewicz, T. Yamakage, T. Itoh, T. Watanabe, and T. Chujoh, "One-stage/two-stage SAO and ALF with LCU-based syntax," Tech. Rep. JCTVC-H0274, Feb. 2012.



Chi Ching Chi received the B.Sc. degree in electrical engineering in 2008 and the M.Sc. degree in computer engineering in 2010, both from the Delft University of Technology, Delft, The Netherlands. He is currently pursuing the Ph.D. degree with the Embedded Systems Architecture Group, Technical University of Berlin, Berlin, Germany.

His current research interests include multi- and many-core architectures, operating systems, parallel programming models and languages, and video compression applications.

Mauricio Alvarez-Mesa received the M.Sc. degree in electronic engineering from the University of Antioquia, Medellin, Colombia, in 2000, and the Ph.D. degree in computer science from the Universitat Politècnica de Catalunya (UPC), Barcelona, Spain, in 2011.

From 2006 to 2011, he was an Adjunct Lecturer with UPC. He was an Intern with IBM Haifa Research Labs, Haifa, Israel, in 2007, and a Visiting Researcher with Technische Universität Berlin (TU Berlin), Berlin, Germany, in 2011. In January 2012, he joined the Multimedia Communications Group, Fraunhofer Heinrich Herz Institut, Berlin, and the Embedded Systems Architecture Group, TU Berlin. His current research interests include parallel processing for multimedia applications, multi- and many-core architectures, and video coding.

Ben Juurlink (SM'XX) received the M.Sc. degree from Utrecht University, Utrecht, The Netherlands, and the Ph.D. degree from Leiden University, Leiden, The Netherlands.

He is currently a Professor of embedded systems architectures with the Electrical Engineering and Computer Science Faculty, Technische Universität Berlin, Berlin, Germany. From 1997 to 1998, he was a Post-Doctoral Research Fellow with the Heinz Nixdorf Institute, Paderborn, Germany. From 1998 to 2009, he was a Faculty Member with the Computer Engineering Laboratory, Delft University of Technology, Delft, The Netherlands. He has authored or co-authored more than 100 papers in international conferences and journals. His current research interests include multi- and manycore processors, instruction-level parallel and media processors, low-power techniques, and hierarchical memory systems.

Dr. Juurlink has been the Leader of several national projects, a Work Package Leader in several European projects, and is currently a Coordinator of the EU FP7 project low-power GPU (lpgpu.org). He is a member of ACM and High Performance and Embedded Architecture and Compilation Network of Excellence. He has served on many program committees, is an Area Editor of *Microprocessors and Microsystems: Embedded Hardware Design* (MICPRO), and is the General Co-Chair of the High-Performance and Embedded Architectures and Compilers 2013 Conference. He received the Best Paper Award at the IASTED International Conference on Parallel and Distributed Computing and Systems in 2002.

Gordon Clare received the M.Sc. degree from the University of Auckland, Auckland, New Zealand, in 1984.

From 1985 to 1989, he was a Development Engineer with Rakon Computers, Sydney, Australia. From 1989 to 1991, he was with Software Development International, heading a team of developers creating a network management system for fault tolerant environments. From 1991 to 1997, he was with CISRA, a Canon Research Center, Sydney, focusing on real-time hardware and software image processing solutions. After moving to France in 1997, he was with several international companies, developing document management systems and a search engine. As a Consultant from 2005 to 2010, he was developing H.264 and scalable video coding (SVC) solutions at Canon and France Telecom-Orange, Rennes, France. Since 2010, he has been with France Telecom-Orange, continuing his research related to HEVC standardization and development.

Félix Henry received the Dipl.-Ing. (M.Sc.) degree in telecommunications from Telecom SudParis, Évry, France, in 1993, and the Ph.D. degree in wavelet image coding from Telecom ParisTech, Paris, France, in 1998.

He started his career in 1995 with Canon Research Center, France, where he focused on still image compression and video coding. He has participated actively in JPEG2000 standardization and holds more than 100 patents, as a co-inventor, in the domain of image and video processing. He joined France Telecom-Orange in 2010 as a Video Coding Project Leader, and since then he has been actively involved in the development of HEVC, where his areas of interest include transform coefficient coding and high-level parallel processing.

Stéphane Pateux received the Dipl.-Ing. (M.Sc.) degree from École Polytechnique, Palaiseau, France, and the Ph.D. degree from the University of Rennes, Rennes, France, in 1998.

He started his career with the National Institute for Research in Computer Science and Control (INRIA) in 1995. He worked on image and video compression, content analysis, and watermarking. In 2004, he joined France Telecom-Orange. He participated in the development of AVC and SVC standards. In 2010, he actively participated in the first development of the new upcoming HEVC video standard. His research led to several international publications (conference, journals, books, and patents) and to Ph.D. studies. Since September 2010, he has been the Head of Research Object Voice and Video Coding at Orange Labs, Paris, France. This research program merges all research activities in audio, voice and video coding, as well as the development of associated quality metrics.

Thomas Schierl received the Diplom-Ingenieur degree in computer engineering and the Doktor der Ingenieurwissenschaften (Dr.-Ing.) degree in electrical engineering and computer science from the Berlin University of Technology, Berlin, Germany, in 2003 and 2010, respectively.

He has been with the Fraunhofer Institute for Telecommunications HHI, Berlin, since 2004. Since 2010, he has been the Head of the Multimedia Communications Group, Image Processing Department, Fraunhofer HHI. He has co-authored various Internet Engineering Task Force (IETF) RFCs, and authored the IETF real-time transport protocol payload format for H.264 SVC, as well as for HEVC. In 2007, he visited the Image, Video, and Multimedia Systems Group of Prof. B. Girod at Stanford University, Stanford, CA, for different research activities. His current research interests include mobile media streaming and content delivery.

Dr. Schierl is a co-editor of the MPEG Standard on Transport of H.264 SVC, H.264 MVC, and HEVC over MPEG-2 Transport Stream in the ISO/IEC MPEG group. He is also a co-editor of the AVC File Format.