Chi, C. C., Alvarez-Mesa, M., Lucas, J., Juurlink, B., & Schierl, T.

Parallel HEVC Decoding on Multi- and Many-core Architectures

A Power and Performance Analysis

This is a post-peer-review, pre-copyedit version of an article published in Journal of Signal Processing Systems. The final authenticated version is available online at: http://dx.doi.org/10.1007/s11265-012-0714-2

Chi, C. C., Alvarez-Mesa, M., Lucas, J., Juurlink, B., & Schierl, T. (2012). Parallel HEVC Decoding on Multi- and Many-core Architectures. Journal of Signal Processing Systems, 71(3), 247–260. https://doi.org/10.1007/s11265-012-0714-2

Noname manuscript No.
(will be inserted by the editor)

Parallel HEVC Decoding on Multi- and Many-core Architectures

A Power and Performance Analysis

Chi Ching Chi · Mauricio Alvarez-Mesa · Jan Lucas · Ben Juurlink · Thomas Schierl

Received: date / Accepted: date

Abstract The Joint Collaborative Team on Video Decoding is developing a new standard named High Efficiency Video Coding (HEVC) that aims at reducing the bitrate of H.264/AVC by another 50%. In order to fulfill the computational demands of the new standard, in particular for high resolutions and at low power budgets, exploiting parallelism is no longer an option but a requirement. Therefore, HEVC includes several coding tools that allows to divide each picture into several partitions that can be processed in parallel, without degrading the quality nor the bitrate. In this paper we adapt one of these approaches, the Wavefront Parallel Processing (WPP) coding, and show how it can be implemented on multi- and many-core processors. Our approach, named Overlapped Wavefront (OWF), processes several partitions as well as several pictures in parallel. This has the advantage that the amount of (thread-level) parallelism stays constant during execution. In addition, performance and power results are provided for three platforms: a server Intel CPU with 8 cores, a laptop Intel CPU with 4 cores, and a TILEGx36 with 36 cores from Tilera. The results show that our parallel HEVC decoder is capable of achieving an average frame rate of 116 fps for 4k resolution on a standard multicore CPU. The results also demonstrate that exploiting more parallelism by increasing the number of cores can improve the energy efficiency measured in terms of Joules per frame substantially.

Keywords HEVC · Video coding · Parallel processing · Power analysis · Real-time 4k · UHD

Chi Ching Chi, Mauricio Alvarez-Mesa, Jan Lucas and Ben Juurlink
Technische Universität Berlin, Sekretariat EN 12, Einsteinufer 17, 10587 Berlin, Germany
Tel.: +49.30.314-73130
Fax: +49.30.314-22943
E-mail: {chi.c.chi,mauricio.alvarezmesa,j.lucas,b.juurlink}@tu-berlin.de

Mauricio Alvarez-Mesa and Thomas Schierl
Fraunhofer-Institute for Telecommunications, Heinrich-Hertz-Institut, Einsteinufer 37, 10587 Berlin, Germany
Tel.: +49 30 31002-227
Fax: +49 30 31002-190
E-mail: thomas.schierl@hhi.fraunhofer.de

1 Introduction

Recent increasing demands to support higher resolutions such as 4k or UHD in consumer video devices have driven the video codec development towards higher compression rates. To meet these demands the Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T and ISO/IEC MPEG has started a project to develop a new video coding standard aiming to reduce the bitrate of the H.264/AVC High Profile [13] by another 50%. The target application is, besides 4k resolution, to also the support native HD on mobile devices. Future extensions of the standard also aims to support high quality color depth of up to 14 bit, and higher chrominance fidelity with 4:2:2 and 4:4:4 chroma subsampling. Some of the application use cases, which have been selected for the first test model evaluation, are random access, such as used in Video-on-Demand or Broadcast applications and low delay for conversational applications. The HEVC project started in 2010, it has been published in July 2012 as a Draft International Standard and is scheduled for finalization in early 2013 [22]. The HEVC project uses the HEVC test Model (HM), which is the reference software, to integrate and evaluate new coding tools for standardization.

To support 4k resolution in real-time at frame rates of 50 and higher, HEVC includes several so-called cod-

ing tools that partition each picture into several partitions that can be processed in parallel, without degrading the quality nor the bitrate. For lower resolutions the provided parallelism can be exploited to improve power efficiency of computer systems, which we will show in this paper. Improvements to power efficiency is of key importance in the increasingly mobile market, because power is not scaling down at the same rate as feature size, the so-called power wall.

To investigate if contemporary multi-/many-cores are able to decode 4k HEVC video sequences in real-time with limited power budgets, we perform a performance and power analysis of (parallel) HEVC decoding. In particular, our contributions can be summarized as follows:

- We improve the single-threaded performance compared to the HEVC test Model (HM) 8.0 by an average of 4.1× using both architectural independent and more architectural specific optimizations.
- We show that by using the novel overlapped wavefront approach (OWF) on top of the optimized single-threads baseline, high speedups can be obtained resulting in much higher than real-time performance (up to 186 fps) for 4k sequences.
- Performance, power, and energy efficiency results are provided for three platforms: a server Intel CPU with 8 cores, a laptop Intel CPU with 4 cores, and a TILE-Gx36 with 36 cores from Tilera.

This paper is organized as follows: first, in Section 2, we present a brief overview of the HEVC standard. Then, in Section 3 we describe the tools for parallel processing that have been included in HEVC. In Section 4 we present the details of the implementation of an optimized parallel HEVC decoder. Section 5 describes the experimental setup, followed by experimental result in Section 6. Finally we summarize and conclude the paper in Section 7.

2 Overview of the HEVC Codec

HEVC is based on the same structure as prior hybrid block-based video codecs such as H.264/AVC, but with enhancements and generalizations in each coding stage. Figure 1 depicts a general diagram of the HEVC decoder and its coding stages [23].

In HEVC the motion compensation uses the same quarter pixel motion resolution, but the derivation of interpolated pixels is generalized using a larger 8-tap interpolation filter for luma and 4-tap interpolation filter for chroma. Intra prediction is generalized as well by parametrizing the prediction angle, allowing 33 different angles. The transform is still an integer transform but allows more block sizes, ranging from 4×4 to 32×32, and has higher internal processing precision. CABAC is the only entropy coding algorithm available in HEVC with improvements to coefficient scan patterns and context grouping to improve implementation efficiency. As in H.264/AVC, an in-loop deblocking filter is applied to reduce blocking artifacts. The HEVC deblocking filter is only applied to edges on a 8×8 grid creating opportunities to filter edges in parallel. In HEVC also an additional in-loop filter is included: the sample adaptive offset (SAO) filter [11]. The SAO filter can be activated on a CTB basis by transmitting offset values or using the offset values of the top or left neighboring CTB. These offsets can either correspond to the intensity bands of pixel values (band offset mode) or the difference compared to neighboring pixels (edge offset mode).

HEVC also defines a more efficient block structure, called Coding Tree Blocks (CTBs). The sequence is coded using a CTB size is of 16×16, 32×32, or 64×64 pixels. Each CTB can be recursively subdivided using a quad-tree segmentation in coding units (CUs), which can in turn be further subdivided in prediction units (PUs) and transform units (TUs) [14]. Each CTB can be split structure. Coding units can be subdivided down to a minimum CU size of 8×8. The minimum prediction units size is 4×8 and 8×4, and minimum TU size is 4×4 pixels.

3 Parallel Video Decoding with HEVC

Previous video codecs, in particular H.264/AVC, have been parallelized using mainly slice-level or macroblock-level parallelism [17,21]. In H.264/AVC, as well as in HEVC, a picture can be partitioned in multiple arbitrarily sized slices for independent processing. Having multiple slices in a picture, however, degrades objective and subjective quality due to additional slice header overhead and slice boundary discontinuities [18]. In H.264/AVC independent macroblocks inside a frame can be reconstructed in parallel using a wavefront approach [24]. Furthermore, macroblocks from different frames can be processed in parallel provided the dependencies due to motion compensation are handled correctly [18]. Entropy decoding, however, can only be parallelized at the frame (slice) level and therefore it has to be decoupled from macroblock reconstruction. Although this approach can scale to a many-core architecture it increases the memory usage [9].

In order to solve the above mentioned problems in HEVC two tools aiming at facilitating high level parallel

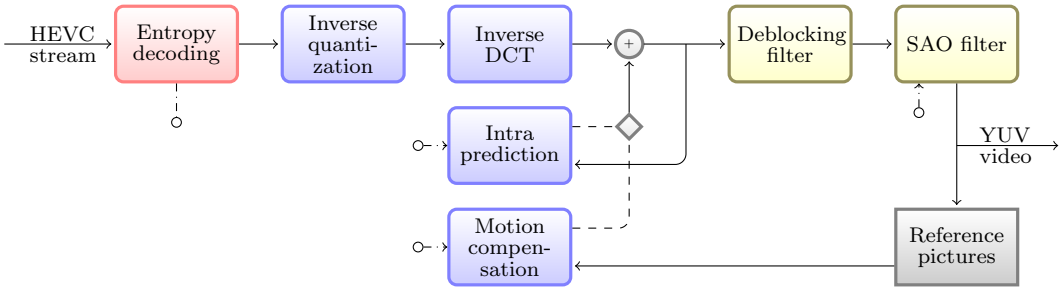


Fig. 1 Block diagram of the HEVC decoder

processing have been included (draft) standard: Wavefront Parallel Processing (WPP) [15] and Tiles [12]. These tools allow to subdivide each picture into multiple partitions that can be processed in parallel. Tiles allow to divide the picture in rectangular groups of CTBs separated by vertical and horizontal boundaries. Tiles boundaries, similarly to slice boundaries, break all the dependencies and because of that have high coding losses and can generate boundary artifacts.

WPP defines one picture partition per CTB row, but does not require special handling of line borders preserving the entropy, prediction or filtering dependencies. The header overhead is small as it only requires the partition entry point offsets to be signaled additionally. As a result the rate-distortion loss of a WPP bitstream is small compared to a non-parallel bitstream, while enabling a decent amount of parallelism that increases with the picture resolution.

Before WPP was completely defined another tool called entropy slices was considered in HEVC [19]. Entropy slices break entropy dependencies but maintain the prediction dependencies. When using one entropy slice per CTB row it is possible to exploit wavefront parallelism in a similar way to WPP. An implementation of a parallel HEVC decoder using wavefront processing with entropy slices on a multicore system with 12 cores showed a speedup of 7.3 for 4K resolution [2].

The scalability of wavefront processing is limited by the reduced number of independent blocks (CTBs or macroblocks) at the beginning and at the end of each frame. To solve this limitation, and increase the parallel scalability of WPP, a technique called Overlapped Wavefront (OWF) has been proposed [8]. With OWF multiple pictures can be decoded simultaneously resulting in a more constant parallelism during execution. An implementation of OWF on a multicore system consisting of 12 cores has shown average speedups of 10X for 4K resolution. An in-depth analysis of the parallelization tools included in HEVC has shown that when WPP is combined with the OWF algorithm it has a better scalability than Tiles [7].

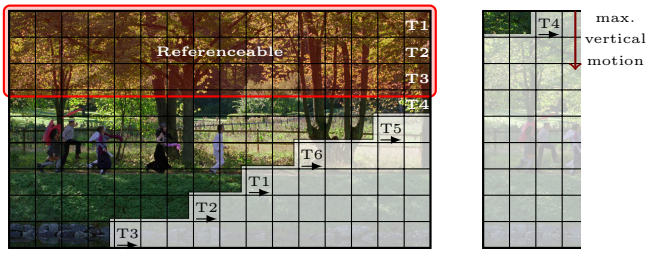


Fig. 2 Frames can be overlapped with a restricted motion vector size, because the reference area is fully decoded.

4 Optimized Parallel HEVC Decoder Implementation

To be able to provide representative power and performance results an optimized parallel HEVC decoder is developed. The developed decoder is compatible with the coding tools described in the HEVC 8.0 draft standard [5]. We first discuss the employed parallelization strategy followed by a concise overview of the steps involved in decoding the Coding Tree Blocks (CTBs) in our implementation. Thereafter, we present the improvements in the single-threaded performance over the HEVC test Model (HM) reference code and briefly discuss where the main improvements originate from.

4.1 Overlapped Wavefront

As mentioned in Section 3, by using the WPP coding tool in HEVC one thread for each CTB row can be used to decode each picture in parallel. When the WPP coding tool is used, the bitstream contains an entry point offset for each CTB row. These offsets allows up to a number threads equal to the number of CTB rows to start decoding in parallel, with a small coding efficiency cost of around 1 percent [15]. In previous work it has been found that a high parallelization efficiency can be achieved when WPP is combined with the overlapped execution of consecutive frames [8]. Figure 2 illustrates the overlapped wavefront (OWF) approach.

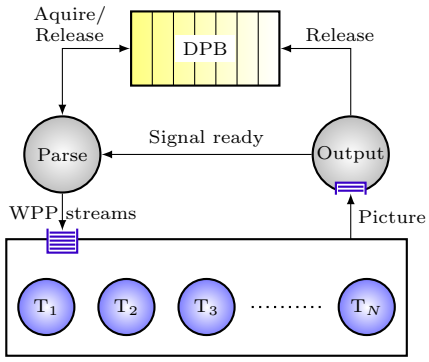


Fig. 3 Decoder organization supporting multi-threaded overlapping wavefront execution.

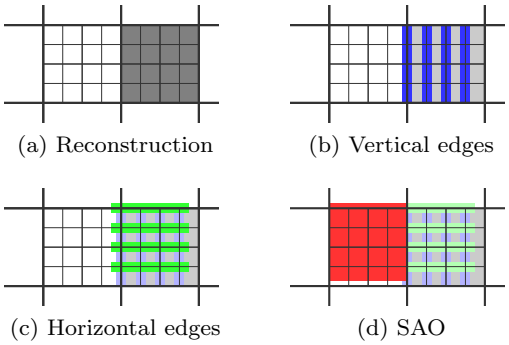


Fig. 4 Order and translation of filtering steps to allow CTB based execution.

Instead of waiting for the entire picture to finish threads can already start decoding the next frame to mitigate the parallelism ramping inefficiencies of regular wavefront execution. As the figure illustrates, to overlap consecutive pictures, a restriction on the size of the vertical component of the motion vectors is required to ensure that the reference area is available. The maximum number of parallel CTB rows using OWF, can be derived using,

$$PAR_{OWF} = \lfloor (H_{Pic} - MMV - 8)/H_{CTB} \rfloor \quad (1)$$

where H_{Pic} is the picture height in pixels, H_{CTB} is the CTB height in pixels, MMV denotes the maximum size of the vertical motion vector component. Eight pixels are additionally subtracted to take the delay of filters (deblocking filter and SAO) and additional pixel rows required by the interpolation filter into account, which will be clarified in the next section. Because currently the HEVC draft does not define the MMV, we instead assume the same the MMV as H.264 of 512 pixels for 1080p and doubled this to 1024 for 2160p. This restriction allows up to 8 threads to be used for 1080p and up to 17 threads for 2160p resolution sequences.

Figure 3 depicts the decoder organization used for the implementation of OWF. The decoder consists of two "control" threads (parse and output) and N "worker" threads. The parse thread acquires a free picture buffer from the display picture buffer (DPB) for every new picture and pushes a task to the shared worker queue for each WPP partition it encounters. The worker threads pop tasks from the queue in order and the wavefront dependencies are maintained among themselves. The worker thread that decodes the last CTB of a picture notifies the output thread of completion by pushing the completed picture. The output thread reorders the decoded pictures in presentation order and releases the pictures after they are displayed/outputted.

The parse thread is also is responsible of releasing no longer used reference pictures. In HEVC the reference pictures that need to be kept in the DPB are signaled for every slice, which is a departure from H.264 where the reference picture that need to be released after decoding the slice are signaled instead. For overlapped execution this is problematic, as in case the current picture uses a reference picture that is not used in the next picture, a reference picture can be released to early.

A solution is, to instead of releasing reference pictures directly when they are not present in the reference picture set (RPS), to release the reference picture when it is not present in the RPS of two consecutive pictures. This delays the release of the reference pictures by one picture. In addition for this scheme to work, it must be ensured that at any time a maximum of two pictures are in-flight. This is implemented by having the parse thread wait until the output threads notifies the completion of a picture if already two pictures are in-flight.

4.2 Coding Tree Block Decoding

A requirement for OWF execution is that all the decoding steps for one CTB are performed before continuing with the next CTB to ensure that the required reference area is available for the threads processing the consecutive picture. In addition, performing all the decoding steps on a CTB basis also improves overall implementation efficiency compared to performing the decoding steps on a slice or picture basis due to increased data locality. The two in-loop filters, deblocking and SAO filter, use and could alter the pixels from surrounding CTBs, which are not all available at the time of decoding the CTB. To process these filters on a CTB basis, therefore, requires delaying them as illustrated in Figure 4.

Figure 4 shows the sequence of filters that are applied after parsing and reconstruction (prediction and transform) of a CTB. In this example the CTB split depth is 2 for each leaf CU and no further prediction and transform subdivision is assumed. First the vertical

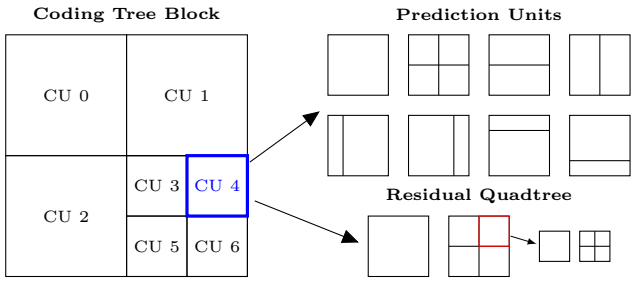


Fig. 5 Subdivision of a CTB in coding units, prediction units, and transform units.

edges of the CTB are deblocked followed by the horizontal edges. The deblocking of the horizontal edges must be delayed horizontally 4 pixels, because HEVC specifies that the horizontal edges must be deblocked using the vertically deblocked pixels as input. These 4 pixels have not been vertically deblocked yet as the last edge belongs to the next CTB. In turn the SAO filter is also delayed because it uses the horizontally filtered pixels as input, and would require a minimum translation of 4 pixels upwards and 1 pixel to the left. Delaying the filter 1 pixel to the left, however, would introduce that the SAO application window would cross 4 CTBs, which all might have different SAO filter types. We decided to delay the SAO filter one entire CTB horizontally to reduce this control overhead.

The parsing and reconstruction follows the quadtree CTB structure illustrated in Figure 5. Each CTB can be split into four CUs which can be further split in smaller CUs given a minimum CU size of 8×8 pixels. Each leaf CU can be intra or inter predicted and has one of the prediction unit (PU) shapes. In case of intra CUs only the first two PU shapes are available, while for inter CUs all the PU shapes are possible. For each PU a different intra-prediction mode or motion vector and reference index pair can be derived. Inter PUs can directly be motion compensated after deriving the motion vectors and reference indices. Intra prediction, however, has to be performed for each transform unit (TU) following the residual quadtree (RQT) block structure.

For each CU a RQT containing TUs can be transmitted in the bitstream. Like the CTB quadtree the RQT can also be split further, but instead has a minimum size of 4×4 pixels. In our implementation the coefficient parsing and inverse transform follow each other directly for optimal locality. Also adding the residual to the prediction and clipping is merged with the inverse transform.

4.3 Single-threaded Performance Improvements

In addition to the parallelization, also single-threaded performance has been significantly improved compared to the reference HM code. The performance improvements do not result from a few concentrated changes, but instead originate from many small improvements over the entire codec which include both architecture independent and architecture specific changes. Some of the more prominent architecture independent changes are a much simplified neighbor context derivation (for parsing, prediction, and filtering), fusing many kernel loops (transform-add-clip, interpolation-weighting, inverse quantization and coefficient parsing), skipping zero block transform, implementing branchless CABAC, removing redundantly stored syntax elements, use of a scratchpad for better TLB locality, switching from CTB to CU based reconstruction, CTB-based filtering, using reference pictures with 8-bit pixel depth when possible, internal bit depth of 8-bit, improved Annex B parser and emulation prevention, etc.

For the architecture specific improvements the performance improvements originate mostly from SIMD optimizations, which are applied to accelerate several time consuming kernels such as the 8-tap interpolation filter, inverse transform with block sizes up to 32×32, and the SAO filter. Also attention was paid to prefetching reference blocks in the interpolation filter and write-combine store operation when writing back the final reconstructed picture to the memory. For the Tilera architecture also the scratchpad memory allocated for each thread is locally homed, which improves the cache utilization by having no redundant cache line copies present as long as the decoding thread remain pinned to the same core.

It should be noted that additional improvements can be achieved by applying SIMD optimizations to the deblocking filter and intra-prediction. For the deblocking filter, the performance gains with SIMD will be smaller compared to inverse transform or interpolation filters mainly because of the branches introduced by the filter adaptation. Although intra-prediction can benefit from SIMD optmization it consumes a small fraction of the total execution time.

5 Experimental Setup

5.1 Platforms

Our experimental setup consists of three different platforms with different number of cores, microarchitectures and performance levels. Table1 presents a summary of the main properties of the three platforms.

System	Intel X86-LP	Intel X86-HP	Tilera TILE-Gx
Processor	Core i7-2760qm	Xeon E5 2687W	TILE-Gx8036
μarchitecture	Sandy Bridge	Sandy Bridge	TILE-Gx
Num. cores	4	8	36
SMT	2-way	2-way	no
Frequency [GHz]	0.8-2.4	1.2-3.1	1.0
Voltage [V]	0.76-1.06	0.84-1.20	0.96
LLC Cache [MB]	6	20	9
Memory	2-ch. DDR3 1600 MHz	4-ch. DDR3 1600 MHz	2-ch. DDR3 1333 MHz
Process [nm]	32	32	40
Operating system	Kubuntu 12.04	Kubuntu 12.04	Tilera MDE-4.0.3.145127
Linux kernel	3.2.0-25	3.2.0-29	2.6.38.8
Compiler	GCC 4.6.3	GCC 4.6.3	GCC 4.6.3
Compiler flags	-O3, AVX enabled	-O3, AVX enabled	-O2

Table 1 Properties of the three different systems used in the experiments.

To test a high performance multicore platform we selected a server with a Xeon E5 2687W processor that consists of 8 Intel X86_64 cores running at 3.1 GHz. We will refer to this system as X86-HP. As a power-optimized multicore platform we used a laptop with a Core i7-2760qm processor that has 4 Intel X86_64 cores running at up to 2.4 GHz. We will refer to it as X86-LP.

To evaluate a many core processor we used a TILE-Gx8036 processor on a TILEncore-Gx36 card which is connected via PCIe to a host system. The TILE-Gx8036 has 36 cores running at 1.0 GHz, where each core is a 64-bit VLIW processor. All the cores are connected with a mesh on-chip interconnect network [3]. The chip includes other peripherals such as the cryptographic unit (MICA) and 4 network interfaces (mPIPE) wich are not used in the experiments reported in this paper, except for one of the Ethernet interfaces and the power sensors. In the rest of the paper we will refer to this system as TILE-Gx.

5.2 Power measurement

To measure the power on the Intel platforms we used the Running Average Power Limit (RAPL) feature introduced with the Sandy Bridge microarchitecture [10]. RAPL uses a architectural power predictor, that is also exposed to software, to implement power capping of the chip and implement more consistent turbo clocking behavior. The architectural power predictor updates the model-specific register (MSR) once every millisecond, and provides high accuracy and correlation with actual power consumption [20]. RAPL exposes the energy consumed by the complete package (complete CPU die), and only the cores and their caches. Additionally, depending on the model RAPL exposes the power of the integrated graphics processing unit or the DRAM controllers. To verify the accuracy we have compared the power reported by RAPL for the two Intel platforms with the power measured for the entire system at different voltage and frequency points, and we observed very good correlation at all operating points. For our power measurements we access the RAPL MSR for the complete package power via PAPI [6].

We measure the power of the TILE-Gx8036 CPU core using the INA219 power monitor chip. This chip measures power by measuring voltage and current by the voltage drop over a shunt resistor. The INA219 contains the required signal condition circuits, a 12-Bit ADC and an I2C bus interface. The measurement error of the INA219 is lower than 0.5%, while the used shunt resistors provide better than 1% accuracy. Overall the error of the measurements should thus be within $\pm 1.5\%$.

The Tilera TILEncore-Gx36 PCIe card contains multiple power monitors, measuring the different voltage rails on the board. The rails are measured behind the power conversion and therefore do not include the losses of the power conversion circuits. For comparability with the Intel RAPL counters we only record the power consumed by the core voltage rail.

The power monitors can be queried using the Tilera provided board test kit (BTK). We used this capability to sample the core power at a approximately 10 Hz rate and save power and timestamps to a data file. When we run applications on the board we record start and end time stamps and average all power samples collected during this time interval to calculate average power for the application. Energy was then calculated by multiplying runtime by average power.

5.3 Test Sequences and HEVC Encoding

Because parallelism is mainly required at HD resolutions, we selected videos for 1080p (1920×1080) and

Options	Value
Max. CU size	64×64
Max. partition depth	4
Transform size: Min.-Max.	4-32
Period of I-frames	256
Number of B-frames (GOP Size)	8
Number of reference frames	4
Motion estimation algorithm	EPZS [25]
Search range	48
Asymmetrical Motion Partition	Enabled
Internal bit depth	8
Sample Adaptive Offset (SAO)	Enabled
Wavefront Parallel Processing (WPP)	Enabled
Quantization Parameter (QP)	22, 26, 30, 34

Table 2 Coding options.

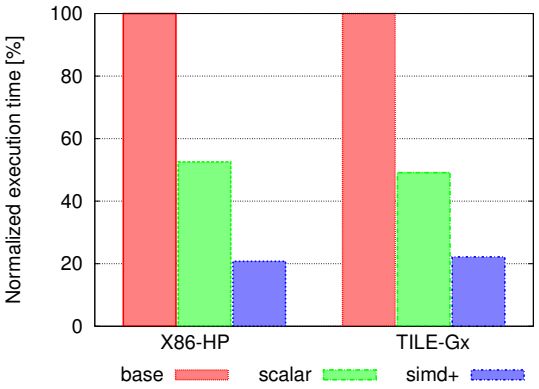


Fig. 6 Normalized average execution that shows the effect of architecture independent (scalar) and architecture dependent optimizations (simd+) with respect to reference code (scalar version).

2160p (4096×2160) resolutions. 1080p is representative for current high definition systems, while 2160p is representative for the next generation of high quality video. For 1080p we used the 5 test sequences described in the HEVC "common conditions" [4]. For 2160p resolution we use four videos from the EBU (European Broadcasting Union) 4K testset [16]. 1080p sequences have 8 bit per sample and 2160p sequences have 10-bit per sample. 1080p sequences are in YUV 4:2:0 format, and 2160p sequences were originally in 4:2:2 format but were converted to 4:2:0 format (because currently the HEVC reference encoder can not handle formats different than 4:2:0)

All the test sequences have been encoded with the HEVC HM reference encoder version 8.0 (svn revision r2738) [5]. Encoding options are based on main HEVC main profile using the random access configuration [4]. Table 2 shows the main configuration parameters of the encoder. In order to enable parallel processing WPP has been enabled. In addition, for supporting OWF, the maximum length of the vertical motion vectors has been constrained to 512 pixels for 1080p and 1024 for 2160p. As a result a maximum of 8 and 17 threads can be used for 1080p and 2160p respectively. Table 3 shows the resulting bitrate and weighted PNSR (0.75×U + 0.125×U + 0.125×V) for all the videos under consideration.

6 Experimental Results

For the experiments that do not use frequency and voltage scaling the x86 platforms are configured at their highest rated frequency and DVFS and Turbo Boost are disabled in the OS and BIOS, respectively. For improved reproducibility and reduced effect of the OS thread scheduling policies, threads are pinned to cores. In the X86 platforms the decodings include runnings with and without simultaneous multithreading (SMT) enabled.

6.1 Single-threaded Optimizations

In Section 4.3 we described the single-threaded optimizations applied to the HEVC decoder. Figure 6 shows the normalized execution of the architecture independent (scalar) and architecture dependent (simd+) optimizations compared to the reference decoder (compiled with autovectorization disabled). For the X86-HP platform the scalar optimizations give a reduction of 48% in execution time compared to the baseline, and the simd++ optimizations give an additional 32% reduction. For the TILE-Gx architecture the results are similar: a 51% reduction in execution time due to scalar optimizations and an additional 27% reduction due to simd+ optimizations. The optimized decoder that includes all the optimizations will be used as baseline for the parallel executions that will be presented in the next sections.

6.2 Performance

We executed the optimized parallel decoder on the three platforms under study for all the videos at different QP values and measured the execution time. Based on it we computed the performance, expressed in frames per second. Tables 4 and 5 show the performance for 1080p and 2160p resolutions respectively. They include results for experiments using one thread and the maximum thread count.

The X86-HP platform achieves the highest performance, with up to 414 fps for 1080p and up to 185 fps for 2160p. When using 8 threads it is possible to decode

video	resolution	frames	QP22		QP26		QP30		QP34	
			bitrate [Kb/s]	YUV-PSNR	bitrate [Kb/s]	YUV-PSNR	bitrate [Kb/s]	YUV-PSNR	bitrate [Kb/s]	YUV-PSNR
BasketballDrive	1080p50	500	16595	40.44	6889	39.09	3651	37.74	2091	36.25
BQTerrace	1080p60	600	38394	38.83	8866	37.21	2823	36.22	1236	34.92
Cactus	1080p50	500	16588	39.30	6030	38.07	3148	36.74	1776	35.21
Kimono	1080p24	241	4422	42.25	2326	40.79	1302	39.19	732	37.48
ParkScene	1080p24	240	6401	40.71	3093	38.70	1600	36.81	829	34.92
DancerPillar	2160p50	500	21071	41.60	3042	41.05	1573	40.36	928	39.46
DancerWater	2160p50	500	35657	43.22	18966	41.96	10104	40.54	5302	39.00
FountainPan	2160p50	500	105154	41.02	52612	39.20	26369	37.46	12791	35.81
LupoPuppet	2160p50	500	56928	40.80	21889	39.94	11554	39.00	6236	37.95

Table 3 Bitrate (in Kb/s) and weighted PSNR (in dB) for all the encoded video sequences.

all the 2160p sequences with more than 50 fps, even the most difficult ones. The X86-LP platform achieves between 76 and 230 fps when using 4 cores for the 1080p sequences and between 23 to 86 fps for the 2160p sequences.

On the TILE-Gx platform, real-time is achieved for most 1080p sequences, except those that require 60 fps at low QPs (smaller than 26). For the 2160p sequences the performance, at the maximum core count, is between 16 and 51 fps. For most of the sequences it is not possible to reach the real-time performance. The main limitation is the the single threaded performance, which is significantly lower compared to the other architectures because of the frequency and microarchitectural disadvantages. Although there are more cores available, we can not use more threads because of the maximum limit of the OWF algorithm has been reached.

The results show that the performance depends heavily on the video content and the bitrate. On the one hand, for sequences with complex or fast motion, such as *LupoPuppet*, there are less skip blocks and more motion compensation operations need to be applied per frame. On the other hand, when the bitrate increases (and the QP decreases) the number of coefficients that needs to be parsed increases as well, resulting in more CABAC operations. Due to its sequential behavior CABAC has a low IPC and cannot be optimized with SIMD instructions.

6.3 Speedup

Figure 7 shows the average speedup achieved using multiple cores compared to the optimized code as baseline for each of the three test platforms. The figure shows that the scaling for the X86 platforms is high, and because of the higher parallelism 2160p scales better for higher core counts than 1080p. Also SMT shows up to 25% performance improvement at low core count and around 12% performance improvement at high core count.

For the TILE-Gx platform similar speedup results are observed up to 8-cores, with a speedup of 6.8× and 7.6× for 1080p and 2160p respectively. At 17 cores, however, a moderate speedup of 14× is achieved which is partly caused by the thread stalls resulting from maintaining the wavefront dependencies. As will be shown in the next section the contention on the TileGx36 memory subsystem reduces scalability at high core counts as well.

6.4 Power and Energy

Figure 8 shows the power in Watts (W) for each platform using different number of cores. The power numbers indicate that a high amount of power is associated with the high performance of the X86-HP platform, with over 100 W of power at the highest core count. The X86-LP and TILE-Gx platforms fare much better in this aspect with a maximum of 31.6 W when using 4 cores with SMT and 20 W at 17 cores, respectively. Also it can be observed that the TILE-Gx platform has a relatively high idle to load power ratio. This can be explained due to the larger amount of power management options available on the X86 platforms. On the TILE-Gx platform the OS does not implement DVFS and no clock or power gating is available/performed. In contrast the X86 platforms implements DVFS (although disabled for this experiment), and many power states for different parts of the chip.

Despite the usage of fine-grained power gating on the X86 platforms, the power consumption for 1 core is higher than the maximum power consumption divided by the number of cores on the chip. This is caused by the parts of the chip that are always on, such as the PCI-e controller and QPI interfaces, and because parts of the chip cannot be power gated when at least one thread

Video	QP	X86-LP		X86-HP		Tilera	
		1 thread	4 threads	1 thread	8 threads	1 thread	8 threads
BasketballDrive	22	28.8	107.0	35.6	219.7	5.9	36.3
	26	39.8	148.3	48.6	302.3	7.9	49.2
	30	48.0	179.2	57.9	368.2	9.3	60.0
	34	54.9	204.8	65.9	424.4	10.4	67.6
BQTerrace	22	20.0	76.3	25.1	172.6	4.2	28.5
	26	41.3	151.1	50.4	318.8	8.2	50.8
	30	58.4	211.3	70.2	431.0	11.0	69.2
	34	68.0	246.7	81.0	501.7	12.4	78.2
Cactus	22	31.9	116.9	39.4	247.3	6.6	40.6
	26	51.3	186.4	62.2	363.9	10.2	59.3
	30	63.9	230.6	76.4	449.6	12.2	72.1
	34	74.6	267.7	88.6	529.7	13.8	83.8
Kimono1	22	34.8	131.1	42.7	283.0	7.0	45.5
	26	43.0	161.1	52.2	346.8	8.4	56.0
	30	50.4	187.5	60.9	398.7	9.6	63.5
	34	57.5	210.9	68.8	433.2	10.7	70.1
ParkScene	22	29.4	110.6	36.2	235.8	6.1	38.3
	26	39.1	146.3	47.8	305.3	7.9	49.6
	30	48.0	178.8	58.0	369.2	9.3	60.1
	34	56.8	208.7	68.3	425.5	10.7	69.0
Average		47.0	173.1	56.8	356.3	9.1	57.4

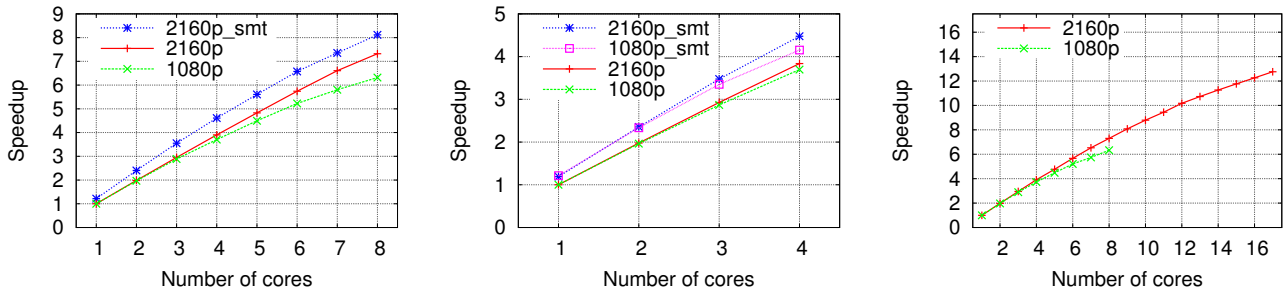
Table 4 Performance in frames per second for 1080p videos at different bitrates for three different platforms.

Video	QP	X86-LP		X86-HP		Tilera	
		1 thread	4 threads	1 thread	8 threads	1 thread	17 threads
DancerPillar	22	11.7	44.8	14.0	101.3	2.2	27.3
	26	18.0	68.4	20.6	151.3	3.2	41.8
	30	21.0	79.2	23.6	171.6	3.7	46.9
	34	23.1	86.8	25.8	185.6	4.0	51.0
DancerWater	22	9.9	38.3	11.8	84.4	1.9	21.6
	26	12.3	47.0	14.4	103.4	2.3	26.1
	30	14.7	55.9	17.1	121.5	2.7	31.5
	34	16.8	64.0	19.3	137.9	3.0	37.0
FountainPan	22	6.1	23.8	7.5	56.2	1.3	16.4
	26	8.2	31.9	10.0	74.3	1.6	21.1
	30	10.4	40.1	12.6	90.5	2.0	26.1
	34	12.6	48.3	14.9	109.9	2.3	30.6
LupoPuppet	22	8.7	33.8	10.7	79.7	1.7	22.0
	26	12.6	48.9	15.0	112.4	2.3	31.3
	30	14.6	56.6	17.1	128.6	2.6	36.5
	34	16.4	62.9	18.9	142.2	2.9	40.5
Average		13.6	51.9	15.8	115.7	2.5	31.7

Table 5 Performance in frames per second for 2160p videos at different bitrates for three different platforms.

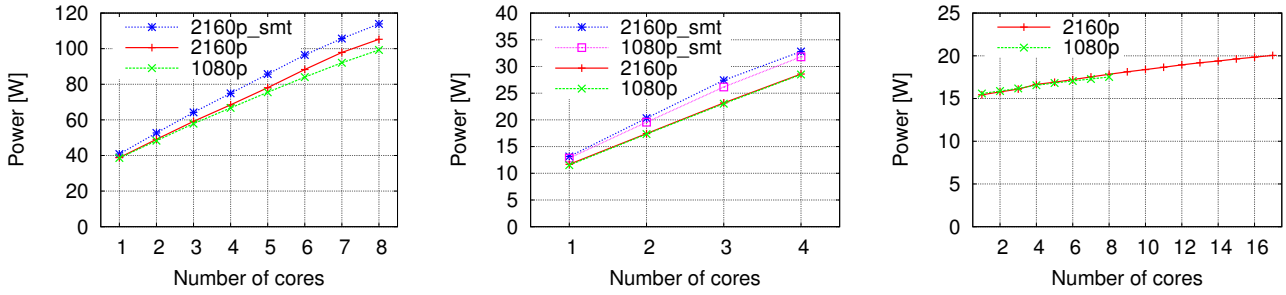
is actively using it, such as the memory controllers and L3 cache partitions.

The power efficiency of the chip depends both on the performance and the power [1]. Figure 9 shows the power efficiency expressed in Joules per frame for the different platforms at different core counts. The common trend for all platforms is that using more cores improves power efficiency. While the power increases with the core count, the performance increases to a greater extent. This especially holds for the TILE-Gx platform due to the relatively high idle power. On the X86 platforms SMT improves power efficiency at low counts, but loses its effectiveness at higher core counts. At their most efficient points the X86-LP platform achieves the lowest energy per frame (179 mJ/F and 614 mJ/F for 1080p and 2160p), followed by the TILE-Gx (334 mJ/F and 696 mJ/F for 1080p and 2160p), and finally the X86-HP (298 mJ/F and 995 mJ/F for 1080p and 2160p).



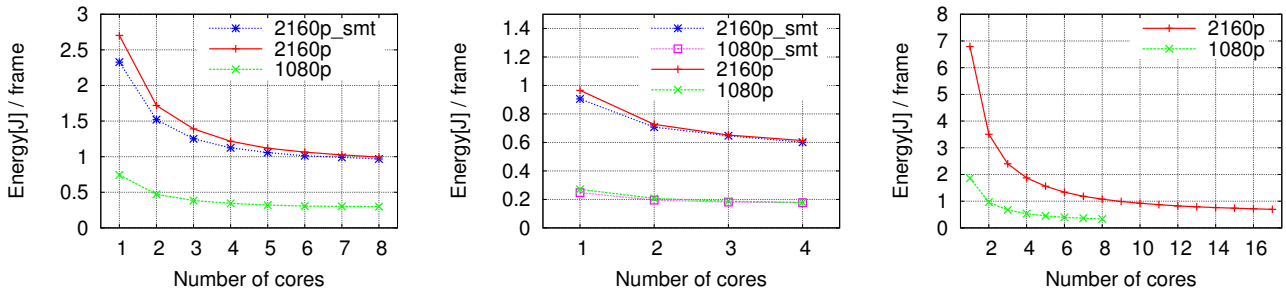
(a) X86-HP (b) X86-LP (c) Tilera

Fig. 7 Speedup for X86-HP, X86-LP and Tilera



(a) X86-HP (b) X86-LP (c) Tilera

Fig. 8 Power for X86-HP, X86-LP and Tilera



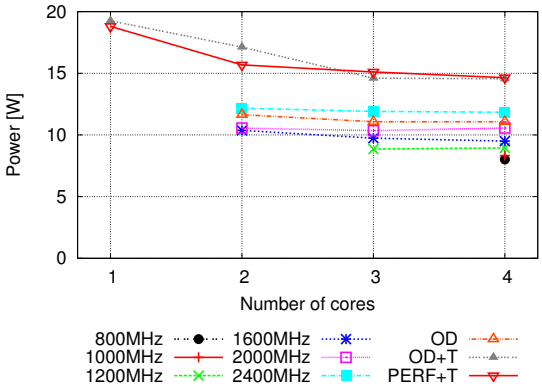
(a) X86-HP (b) X86-LP (c) Tilera

Fig. 9 Energy per frame for X86-HP, X86-LP and Tilera

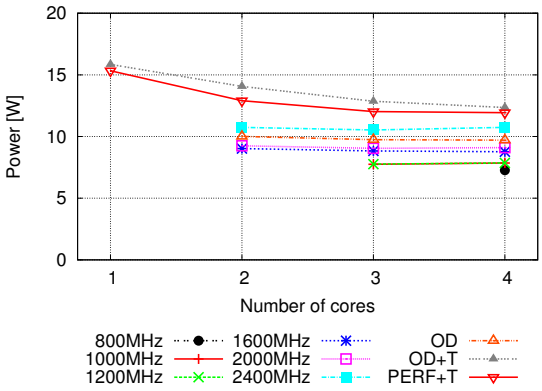
6.5 Frequency and Voltage Scaling on Intel SandyBridge

In many practical applications of a HEVC decoder, decoding at highest possible speed is not desired. Instead the decoder needs to meet a certain frame rate for real-time performance. To measure the power efficiency for these use cases, we have conducted additional experiments in the X86-LP platform in which we limited the decoding speed to 50 fps at different voltage/frequency operating points. These include six static configuration points with frequencies ranging from 800 MHz to 2.4 GHz, and three dynamic configurations: "On Demand" (OD), "On Demand with Turbo" (OD+T) and "Perf+T" (Performance with Turbo"). With OD the processor runs at the lowest possible frequency and increases to maximum when CPU usage reaches 100%. OD+T adds the Turbo Boost feature that allows the processor to dynamically increase the speed above its nominal operating frequency [20]. Finally, in Perf+T the processor is set to its maximum frequency with Turbo Boost enabled.

For these experiments the Cactus 1080p50 sequence encoded with QP 26 and 30 is used for which decoding speed is close to the average. Figure 10 shows the power



(a) QP26



(b) QP30

Fig. 10 Power at different Frequency/Voltage configurations for Cactus 1080p50 sequence at two different QP encodings with real-time decoding for the Intel X86-LP system.

consumption for the different core count/frequency points that achieve 50 fps. The voltage and frequency scale linearly with respect to each other following the range reported in Table 1.

The results show that it is not possible to achieve real-time decoding with only one core, even at the maximum nominal frequency. This is only possible when Turbo Boost is enabled, but with Turbo Boost the decoder uses 2.4 times more power compared to the most efficient setting. With two cores it is possible to decode in real-time at 1.6 GHz with around 50% of the power used with one core. Using four cores at 800 MHz is the most efficient setting for this experiment using just 8.0 and 7.3 W (or 159 mJ/frame and 145 mJ/frame) for QP 26 and QP 30 respectively.

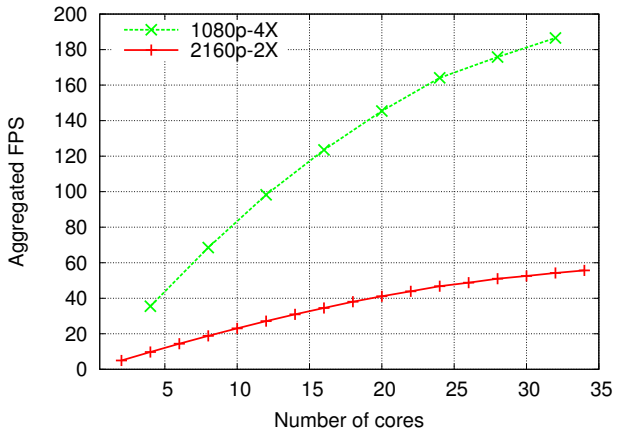
Our results also show that the standard DVFS strategy (OD) is producing suboptimal results. DVFS is only slightly more efficient than always running at stock 2.4 GHz clock speed when Turbo Boost is disabled. When decoding using four cores DVFS is using between 33% and 46% extra power compared to running at a fixed 800 MHz clock speed. Also enabling Turbo Boost results in poor power efficiency due to operating at a higher voltage and frequency (3.2-3.4 GHz). These results show that with playback type of workloads the default configurations on many systems, in which both Turbo Boost and DVFS are enabled, produces a much lower power efficiency than the system is able to.

6.6 Increasing the workload on TILE-Gx

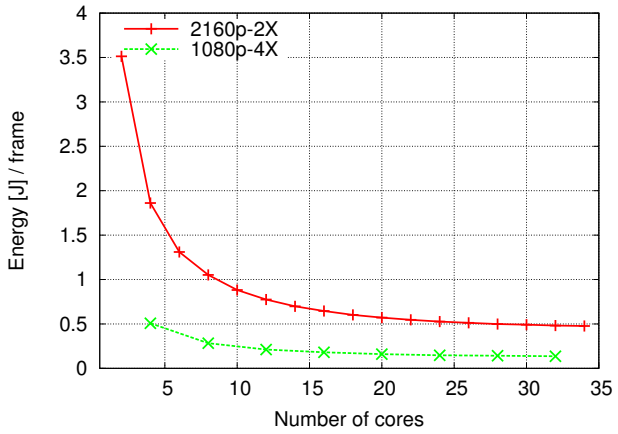
On the TILE-Gx system we have more cores available than we are able to use with one bitstream. Combined with the high idle power this impacts the power efficiency negatively. For that reason we measured the performance and energy per frame while decoding two 2160p bitstreams or four 1080p bitstreams at the same time. This way we are able to use most of the cores. The results of these experiments can be seen in Figure 11. The figure shows that the energy used per frame decreased by using more cores.

Compared to using only 8 cores for one 1080p bitstream, using four 1080p bitstreams at a time decreases the energy per frame from 0.333 J/F to 0.136 J/F. Similarly for 2160p the energy per frame decreases from 696 mJ/F to 477 mJ/F. The TILE-Gx processor is able to achieve better energy efficiency, when most cores are used, compared to both Intel platforms, despite its process technology disadvantage (40nm vs. 32nm). The lowest energy per frame achieved by X86-HP platform is 299 mJ/F for 1080p and 968 mJ/F for 2160p, and for X86-LP platform this is 177 mJ/F for 1080p and 601 mJ/F for 2160p. The power efficiency results are even slightly pessimistic as part of the idle power is consumed the on-chip accelerators for high-speed networking (mPIPE) and cryptography (MiCa). In our test setup network is required in order to start executions remotely from the host machine, but is not strictly required for the decoding process. Disabling these accelerators lowered the power by 2.0 Watts both in idle and full load, leading to approximately 9% lower mJ/F.

While the aggregated performance using multiple streams is much higher than a single one, the speedup, however, is not linear and especially nearing the end of the curve starts to saturate. When scaling to the number of cores that the TILE-Gx offers, the effects of contention on shared resources become more visible. More optimizations targeting the memory hierarchy, such as improved data prefetching, could improve the results to a greater extend than the Intel platforms which have less cores and relatively more cache memory.



(a) FPS



(b) Energy per frame

Fig. 11 Aggregated frames per second and energy per frame for Tilera when increasing the total load: 4 times for 1080p (1080p-4X) and 2 times for 2160p (2160p-2X)

7 Conclusions

In this paper we have presented a power and performance analysis of an optimized parallel HEVC decoder. The parallelization strategy, called Overlapped Wavefront (WPP), which is an extension of the Wavefront Paralllel Processing (WPP) tool, allows to process multiple picture partitions as well as multiple pictures in parallel with minimal compression losses.

The parallel decoder has been evaluated on three different architectures: a high performance 8-core Intel server processor, a 4-core Intel mobile processor and a 36 core low power Tilera processor. In addition to performance results we have measured power and computed energy efficiency in term of Joules per frame.

Our parallel HEVC decoder is the first to achieve a frame rate of more than 100 fps at 4k resolution using a standard multi-core CPU. With the 8 core Intel processor we achieved a speedup of 6.3 for 1080p and 7.3 for 2160p. In the Tilera processor a maximum speedup of 12.8 with 17 cores is achieved but result only in an average of 31.7 fps for 2160p. Our parallelization approach enables up to 17 threads for 4k and up to 8 for 2k which is not sufficient to fully utilize the Tilera many-core. Therefore, we have also conducted experiment with decoding four 1080p sequences in parallel and two 4k sequences. In these cases the aggregate performance is 186 fps and 55.6 fps, respectively. For these configurations the Tilera processor obtains a better energy efficiency compared to the server and laptop Intel CPUs.

The results show that in general using more of the available processors improves the energy efficiency in terms of energy per frame, in particular for a small number of cores. For example, for 4k resolution, on the server CPU going from 1 to 2 cores improves the energy per frame from 743 mJ/frame to 471 mJ/frame, and going to 3 cores improves it further to 384 mJ/frame. Going beyond 4 cores still improves the energy efficiency, but to a lesser extend.

Because the obtained performance, in some cases, is beyond the requirements of real-time video decoding, the additional parallelism in the application can be used to improve power efficiency. For example, on the Intel mobile CPU, we found that 1080p real-time decoding at 50 fps requires 1 core at maximum frequency and Turbo Boost that consume 19.2 W. Alternatively, the same performance can be achieved with 4 cores running at 800 MHz consuming only 8 W. It has been observed, however, that current dynamic voltage and frequency scaling approaches (DVFS) are not able to reach the optimal power point.

References

1. Akenine-Möller, T., Johnsson, B.: Performance per What? Journal of Computer Graphics Techniques (JCGT) **1**(1), 37–41 (2012)
2. Alvarez-Mesa, M., Chi, C.C., Juurlink, B., George, V., Schierl, T.: Parallel Video Decoding in the Emerging HEVC Standard. In: Proceedings of the 37th International Conference on Acoustics, Speech, and Signal Processing (ICASSP) (2012)
3. Bell, S., Edwards, B., Amann, J., Conlin, R., Joyce, K., Leung, V., MacKay, J., Reif, M., Bao, L., Brown, J., Mattina, M., Miao, C.C., Ramey, C., Wentzlaff, D., Anderson, W., Berger, E., Fairbanks, N., Khan, D., Montenegro, F., Stickney, J., Zook, J.: TILE64 - Processor: A 64-Core SoC with Mesh Interconnect. In: Digest of Technical Papers of the IEEE International Solid-State Circuits Conference (ISSCC), pp. 88–598 (2008)
4. Bossen, F.: Common Test Conditions and Software Reference Configurations. Tech. Rep. JCTVC-H1100 (2012)
5. Bross, B., Han, W.J., Sullivan, G.J., Ohm, J.R., Wiegand, T.: High Efficiency Video Coding (HEVC) Text Specification Draft 8. Tech. Rep. JCTVC-J1003 (2012)

6. Browne, S., Dongarra, J., Garner, N., Ho, G., , Mucci, P.: A Portable Programming Interface for Performance Evaluation on Modern Processors. International Journal of High Performance Computing Applications **14**(3), 189–204 (2000)
7. Chi, C.C., Alvarez-Mesa, M., Juurlink, B., Clare, G., Henry, F., Pateux, S., , Schierl, T.: Parallel Scalability and Efficiency of HEVC Parallelization Approaches. IEEE Transactions of Circuits and Systems for Video Technology (2012)
8. Chi, C.C., Alvarez-Mesa, M., Juurlink, B., George, V., Schierl, T.: Improving the Parallelization Efficiency of HEVC Decoding. In: Proceedings of IEEE International Conference on Image Processing (ICIP) (2012)
9. Chi, C.C., Juurlink, B.: A QHD-capable Parallel H.264 Decoder. In: Proceedings of the International Conference on Supercomputing (ICS), pp. 317–326 (2011)
10. David, H., Gorbatov, E., Hanebutte, U.R., Khanna, R., Le, C.: RAPL: Memory power estimation and capping. In: Proceedings of the ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED), pp. 189–194 (2010)
11. Fu, C.M., Chen, C.Y., Tsai, C.Y., Huang, Y.W., Lei, S.: CE13: Sample Adaptive Offset with LCU-Independent Decoding. Tech. Rep. JCTVC-E409 (2011)
12. Fuldseth, A., Horowitz, M., Xu, S., Zhou, M.: Tiles. Tech. Rep. JCTVC-E408 (2011)
13. Advanced Video Coding for Generic Audiovisual Services. ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 AVC) (2003)
14. Han, W.J., Min, J., Kim, I.K., Alshina, E., Alshin, A., Lee, T., Chen, J., Seregin, V., Lee, S., Hong, Y.M., Cheon, M.S., Shlyakhov, N., McCann, K., Davies, T., Park, J.H.: Improved Video Compression Efficiency Through Flexible Unit Representation and Corresponding Extension of Coding Tools. IEEE Transactions on Circuits and Systems for Video Technology **20**(12), 1709–1720 (2010)
15. Henry, F., Pateux, S.: Wavefront Parallel Processing. Tech. Rep. JCTVC-E196 (2011)
16. Hoffman, H., Kouadio, A., Thomas, Y., Visca, M.: The Turin Shoots. In: EBU Tech-i, 13, pp. 8–9. European Broadcasting Union (EBU) (2012). URL http://tech.ebu.ch/docs/tech-i/ebu_tech-i_013.pdf
17. Juurlink, B., Alvarez-Mesa, M., Chi, C.C., Azevedo, A., Meenderinck, C., Ramirez, A.: Scalable Parallel Programming Applied to H.264/AVC Decoding. Springer (2012)
18. Meenderinck, C., Azevedo, A., Alvarez, M., Juurlink, B., Ramírez, A.: Parallel Scalability of Video Decoders. Journal of Signal Processing Systems **57**, 173–194 (2009)
19. Misra, K., Zhao, J., Segall, A.: Entropy Slices for Parallel Entropy Coding. Tech. Rep. JCTVC-B111 (2010)
20. Rotem, E., Naveh, A., Rajwan, D., Ananthakrishnan, A., Weissmann, E.: Power-Management Architecture of the Intel Microarchitecture Code-Named Sandy Bridge. IEEE Micro **32**(2), 20–27 (2012)
21. Seitner, F.H., Schreier, R.M., Bleyer, M., Gelautz, M.: Evaluation of Data-parallel Splitting Approaches for H.264 Decoding. In: Proceedings of the International Conference on Advances in Mobile Computing and Multimedia, pp. 40–49 (2008)
22. Sullivan, G.J., Ohm, J.R.: Recent Developments in Standardization of High Efficiency Video Coding (HEVC). In: Proceedings of SPIE, Applications of Digital Image Processing XXXIII, pp. 77,980V–77,980V–7 (2010)
23. Sullivan, G.J., Ohm, J.R., Han, W.J., Wiegand, T.: Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Transactions on Circuits and Systems for Video Technology (2012)
24. der Tol, E.B.V., Jaspers, E.G.T., Gelderblom, R.H.: Mapping of H.264 Decoding on a Multiprocessor Architecture. In: Proceedings of SPIE, 5022, Image and Video Communications and Processing, pp. 707–718 (2003)
25. Tourapis, A.M.: Enhanced Predictive Zonal Search for Single and Multiple Frame Motion Estimation. In: Proceedings of SPIE Visual Communications and Image Processing 2002, pp. 1069–1079 (2002)