Low Power HEVC Software Decoder for Mobile Devices

Erwan Raffin, Erwan Nogues, Wassim Hamidouche, Seppo Tomperi, Maxime Pelcat, Daniel Menard

Low Power HEVC Software Decoder for Mobile Devices

Erwan Raffin, Erwan Nogues, Wassim Hamidouche,
Maxime Pelcat, Daniel Menard
INSA Rennes, IETR, UMR CNRS 6164, UEB
20 Av. Buttes de Coesmes, 35708 Rennes, France
firstname.lastname@insa-rennes.fr

Seppo Tomperi
VTT Technical Research Centre
of Finland, Oulu, Finland
seppo.tomperi@vtt.fi

[1] ***Abstract*—In the context of mobile handheld devices, energy consumption is a primary concern and the process of video decoding is often among the most resource-intensive applications. Recent embedded processors are equipped with advanced features such as Dynamic Voltage Frequency Scaling (DVFS) in order to reduce their power consumption. These features can be used to perform low power video decoding when no hardware decoding support is available for a given standard.**
High Efficiency Video Coding (HEVC) is a recent video standard offering state-of-the-art compression rates and advanced parallel processing solutions. This paper presents strategies for the power optimization of a real-time software HEVC decoder on Neon architecture. These strategies include the exploitation of data and task-level parallelism, as well as the use of a new frequency control system to optimize the processor DVFS, based on an estimation of the decoding complexity. Extensive power measurement results, based on a multi-core ARM big.LITTLE processor, are provided and compared to state-of-the-art. These results show that the proposed open-source implementation can reach an energy consumption below 21 nJ/px for HD decoding at 2.2 Mbits/s.

I. Introduction

The first version of the HEVC standard [5, 21] was finalized in January 2013 by ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG) under a partnerships known as the joint collaborative team on video coding (JCT-VC). The HEVC standard enables a gain of up to 50% in terms of subjective video quality [20] with respect to the H.264/AVC high profile [24]. Moreover, the second version of the HEVC standard enables new coding functionalities including an efficient coding of video at high bitdepth (up to 14 bits) and with high color format fidelity: 4:2:2 and 4:4:4. The HEVC rate-distortion gain together with these new functionalities enable the proliferation of new video services such as video transmission at ultra high quality over broadband and wireless networks for both fixed and mobile devices. On the other hand, the HEVC standard was designed with a particular attention to meet with complexity requirements. Therefore, early after the standardization of the first HEVC version, a number of software implementations have emerged to offer real time decoding solutions for HEVC coded video contents [14, 7, 23, 8, 10, 2]. These software implementations offer flexibility, fast time-to-market and are well suited for quick adaptation to standard evolution. Moreover, software decoder can be easily implemented on embedded platforms not dedicated to video processing. Authors in [7] studied the complexity assessment of the HEVC decoder, which remains quite similar to the complexity of the H.264/AVC decoder. The proposed decoder has been optimized for both x86 and ARM architectures and compared with the HEVC reference software model (HM) decoder. In [8], authors investigate HEVC decoder optimizations for real time decoding on an Intel processor. The presented decoder is capable of processing several pictures in parallel based on the frame-based parallelism. Low level optimizations for x86 processors together with frame-based parallelism enable the decoding of 4K ultra high definition HEVC video on an Intel i7 processor running at 2.5 Ghz. The power consumption performance of this parallel HEVC decoder have been assessed on different multi-core Intel architectures in [10].

In this paper, we propose a low power HEVC software decoder for mobile devices. Three strategies are investigated for lowering the power consumption of this software HEVC decoder:

- The first strategy consists in exploiting data-level parallelism in order to reduce the decoding time for a fixed power consumption and thus reduce energy consumption. The strategy is generic for all processors supporting Single Instruction Multiple Data (SIMD) and the presented implementation is specific to ARM-based processors with a NEON architecture extension.
- The second strategy, based on task-level parallelism, is obtained by exploiting frame-based parallelism which decodes a set of HEVC frames in parallel. This requires a Symmetric Multi-Processor (SMP) operating system managing pthreads such as Linux.
- The third strategy is based on a new technique to exploit efficiently the processor Dynamic Voltage Frequency Scaling (DVFS) capabilities in the context of video decoding. For real-time decoding, processing must be done at a given pace. The over-abundance of processing capabilities makes room for reducing the power consump-

[1]This document is a preprint version. The final publication is available at link.springer.com: http://link.springer.com/article/10.1007%2Fs11554-015-0512-8

tion of the decoder by adapting dynamically its voltage and frequency. Classical DVFS policies fail at adapting efficiently the processor clock frequency due to the huge variation of the decoding time from one frame to another. We propose a new strategy that uses information from the decoding application to enable better DVFS adaptations.

These three solutions have been considered since many recent mobile handheld devices offer SIMD, SMP operating system and DVFS. Moreover, the frame-based parallelism can be used for all HEVC bitstream whatever the coding configuration. The presented HEVC decoder is based on the open source software project *OpenHEVC* [2]. The performance of the proposed decoder in terms of energy consumption is assessed after applying the three energy optimization strategies.
The rest of this paper is organized as follows. Section II presents a short overview of the HEVC standard and the existing real time HEVC decoder implementations. Section III describes the proposed decoding system including the *OpenHEVC* decoder, and the 3 energy optimization strategies. The power consumption performance of the proposed decoder is assessed on ARM platform in Section V. Finally, Section VI concludes this paper.

II. Related work

A. HEVC standard

The HEVC standard can reach the same subjective video quality than its predecessor H.264/AVC at about half bitrate [20]. This gain is obtained thanks to new tools adopted in the HEVC standard, such as quadtree-based block partitioning, large transform and prediction blocks, accurate intra/inter predictions, the in-loop sample adaptive offset (SAO) filter and highly adaptive entropy coder named Context Adaptive Binary Arithmetic Coder (CABAC) [21]. The HEVC encoded frame is partitioned into Coding Tree Units (CTUs), each containing one luma Coding Tree Block (CTB) and two chroma CTBs. Recursive subdivision of a CTU results in Coding Unit (CU) leaves with the corresponding Coding Blocks (CBs). The CU can be split into Prediction Units (PUs), a basic entity for intra and inter predictions, and recursively split into Transform Units (TUs), a basic entity for residual coding. The HEVC standard was designed with a particular attention to complexity, where several solutions were defined to leverage multi-core architecture and enhance both encoding/decoding frame rate and frame latency. These high level parallel processing solutions, including wavefront, independent slices and tiles, are described in the next section.

1) Parallelism in HEVC: Three high level parallel processing solutions [11], including idependent slice, tile and wavefront, are defined in the HEVC standard to simultaneously process multiple regions of a single picture. The frame can be partitioned into one or many independent slices, mainly to increase the bitstream robustness. The tile concept splits the picture into rectangular groups of CTUs, called tiles. Independent slices and tiles reset the probabilities of the CABAC and remove intra prediction dependencies and thus can be used for parallel encoding and decoding. However, limiting intra prediction and resetting CABAC probabilities decrease the coding performance in terms of rate distortion, especially for large numbers of tiles/slices per frame. Moreover, in-loop filters cannot be performed in parallel at the tile/slice edges without additional control mechanisms. The Wavefront Parallel Processing (WPP) solution complements slices and tiles by splitting the frame into CTU rows [12]. In WPP mode, the CABAC context is initialized at the beginning of each CTU row. The overhead caused by this initialization is limited since the CABAC context at each CTU row is initialized by the CABAC context state at the second CTU of the previous CTU row. The decoding of each CTU row can be carried out on a separate thread with a minimum delay of two CTUs between adjacent CTU rows. These three high-level parallel processing solutions depend on the bitstream, and can be used only when these tools are enabled by the encoder. In this paper we consider the frame-level parallelism that enables to simultaneously process multiple frames regardless of the encoding configuration [19].

B. Low power HEVC decoders

Hardware implementations of HEVC decoder have been recently proposed. In [16], an HEVC decoder supporting 4K resolution at 30 fps is proposed. Implementation results are given for 40nm CMOS technology. The clock frequency is set to 200 MHz. An energy consumption for the core around 0.3 nJ/px is reported. Performances are obtained thanks to a efficient pipeline scheme, a specific cache for the motion compensation filter and a unified prediction engine supporting hierarchical coding structure and many prediction and transform block sizes. In [18], an HEVC decoder supporting 4K resolution at 60fps is proposed and results are given for a 28nm CMOS technology. The clock frequency is set to 350 MHz. An energy consumption for the core around 0.2 nJ/px is reported. An adaptive coding unit balance architecture is used to optimize the pipeline buffer. Wave-front parallel processing is exploited by using a dual-core architecture. A weighted memory management unit is used to optimize the access to external DRAM. By specializing the circuit to application requirements, hardware implementation enables obtaining high performance with low power consumption. This specialization is at the expense of low flexibility and long development times. Recent multi-core and many-core architectures have become an interesting solution to implement an HEVC decoder. Recent works [10] have demonstrated the ability to obtain a real-time software decoder on high resolutions such as 4K. This software implementation targets a laptop Intel CPU with 4 cores, and a TILE-Gx36 with 36 cores from Tilera. These targets have moderate power consumptions and are dedicated to high-performance computing. For these multi-core and many-core processors, task level parallelism is used to exploit WPP coding. Moreover, performance has been significantly improved compared to the reference HM code by integrating architecture independent and architecture dependent optimizations. Results are reported for Full HD and Ultra HD resolutions. For the

4-core Intel CPU, an energy consumption of 74 nJ/px and 86 nJ/px are obtained respectively for the Full HD and Ultra HD resolutions. For the 36-core Tilera CPU, an energy consumption of 161 nJ/px and 84 nJ/px are obtained respectively for the Full HD and Ultra HD resolutions.

In this paper, we target mobile handheld devices based on ARM processor. Data-level parallelism and task level parallelism are exploited to reduce the decoding time. A DVFS strategy adapted to video decoding has been used to improve the energy efficiency compared to existing DVFS policies.

III. Decoding system description

The proposed energy efficient HEVC decoder is based on the open source *OpenHEVC* project [2] ported on a low-power embedded platform based on the heterogeneous computing architecture ARM big.LITTLE.

A. Decoder architecture

The *OpenHEVC* decoder is developped in C programming language on the top of the *FFmpeg* library [1]. The *OpenHEVC* decoder implements a conforming HEVC decoder and supports the three main profiles defined in the HEVC standard, namely Main, Main 10 and Main Still Picture profiles [21]. Figure 1 shows the block diagram of the *OpenHEVC* decoder. All decoding steps are performed at the level of a CTU. The decoder browses in raster scan the CTUs within the slice and calls the recursive function *hls_coding_tree* to decode each CTU. This function scans the CUs within a CTU in z-scan and calls for each CU the *hls_coding_unit* function. This calls specific functions to perform inverse prediction and inverse transform operations. Once all CUs within the CTU are decoded, the decoder performs deblocking and SAO filters on the decoded CTU. The *OpenHEVC* decoder also supports the high level parallel processing solutions defined in the HEVC standard including wavefront, slice and tile parallelism solutions. In addition, the *OpenHEVC* decoder has been recently extended to support the range extensions profiles including Monochrome, 4:2:2 and 4:4:4 enhanced chroma sampling structures as well as bit depths in 10 and 12 bits. The decoder successfully passed the decoding of all range extensions conformance bitstreams of bitdepth lower than 14 bits.

B. Embedded Platform

The targeted embedded platform is an octa-core Exynos 5410 SoC based on the big.LITTLE configuration with two clusters: one is composed of four ARM Cortex-A15 cores and the other of four ARM Cortex-A7 cores, both supporting DVFS per core. Only one cluster can be used at a time. This SoC is widely used in recent smart phones and tablets [4] for its low-power capabilities. Linux operating system is used on this platform to provide pthread capabilities for the parallel implementation of the decoder on a multi-core processor.

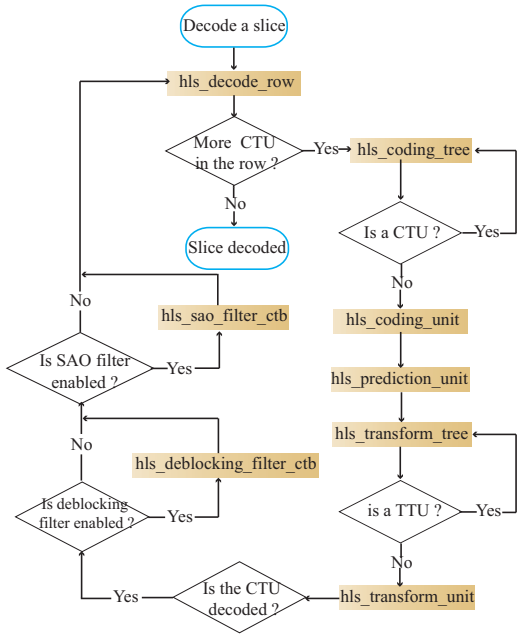

Fig. 1. Block diagram of the architecture of the OpenHEVC decoder (TTU: Transform Tree Unit)

IV. Energy efficient HEVC decoder

The global power consumption in current CMOS integrated chips can be broken down into the sum of a dynamic power and a static power. To reduce the influence of static power, Dynamic Power Management (DPM) is used to turn a core into a lower power state when this core is not used. A popular execution strategy called *race-to-idle* was implemented to execute a task as fast as possible, after which the processor enters in a sleep state (if no other task is available). The Linux implementation of this mechanism is called the *Performance* governor. To reduce the influence of dynamic power, DVFS is used to reduce both the clock frequency f and the voltage V until the real-time constraint is fulfilled. The *ondemand* frequency governor in Linux supports this strategy by increasing the clock frequency of the CPU as long as the workload exceeds an `upthreshold` limit. To improve performance, various techniques have been studied in the past at both application level [17] and architecture level [15]. The main goal of these studies is to spread the workload [22] in order to lower the clock frequency [15], thus lowering the dynamic power dissipation while keeping the required performance.

All strategies exploiting DVFS consist in *executing as slow as possible while not missing a deadline*. It is particularly well suited for online video decoding where a a steady playback is required. Hence the decoder can adapt its decoding speed as long as the deadline is met. Figure 2 illustrates an example of two different execution strategies, one running at f_1 and another at f_2 where $f_1 = 2.f_2$. Assuming that the deadline matches the decoding of f_2, any decoding using a higher working frequency would finish before and would create

some slack time (i.e. time of inactivity). This slack time can also be used to stretch the decoding process and meet closely the deadline when the decoding is processed at f_2.

In CMOS circuits, the dynamic power consumption is given by the following equation:

$$P_{dyn} = C_{eff}.V^2.f \quad (1)$$

where C_{eff} stands for the circuit effective capacitance and V is the supply voltage associated to the working frequency f [9].

In the example of Figure 2, the input data can be processed at

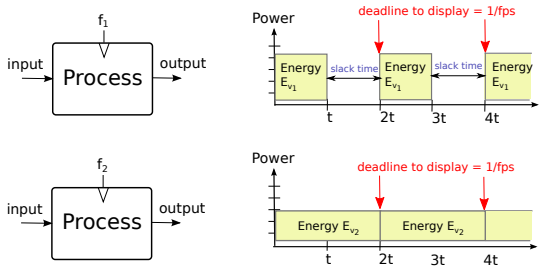

Fig. 2. Two execution strategies: race-to-idle (top) and as slow as possible (bottom)

either f_1 or f_2 corresponding to supply voltage $V_1 = 1.36V$ and $V_2 = 1.06V$ respectively [1]. Assuming that the processing time is doubled when the process runs at f_2, then the energy saving is computed as follows:

$$\frac{E_{V_2}}{E_{V_1}} = \frac{C_{eff}.V_2^2.\frac{f}{2}.2.t}{C_{eff}.V_1^2.f.t} = (\frac{V_2}{V_1})^2 \sim \frac{78}{100} \quad (2)$$

In order to obtain a reliable system, the video decoding performance is usually much higher than the one required to keep the pace of the display rate. Hence, the system can afford dynamic frequency downscaling while offering the same quality of service. As a result, in the presented example, DVFS makes the power consumption decrease to $P_{V_2} = C_{eff}.(V_2)^2 \sim \frac{39}{100}.P_{V_1}$ and results in an energy saving of 22%.

To obtain an energy efficient decoder, optimizations can be applied at different level as depicted in Figure 3. The main challenge of these optimizations is to maximize the slack-time in order to reduce the supply voltage.

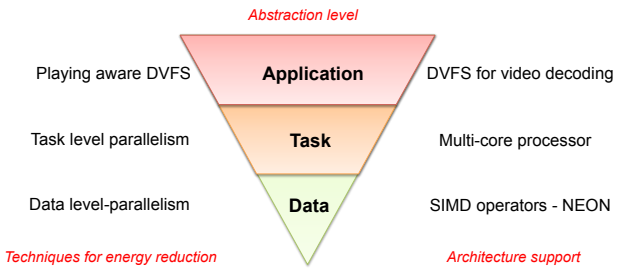

Fig. 3. Different level of optimization for energy reduction

Data level parallelism is used to process several data in parallel leading to a reduction of the execution time of these processing.

Task level parallelism is used to process several task in parallel on different cores. By dispatching the different tasks on the available cores, the load of each core is reduced and thus the clock frequency can be diminished.

In this paper, we introduce an application-aware DVFS strategy that is used to exploit efficiently the slack-time despite its variations over time. Modern video compression schemes use variable bit-rate coding leading to highly variable processing loads. Moreover, the decoding time of a frame depend of its type (I, P, B) and its content. Thus, the decoding time can vary on a large degree (more than ten-fold) from one frame to another. Given that the decoding time varies and can not be predicted easily from the decoding of the previous frame, classical DVFS policies, like *ondemand* frequency governor in Linux, tend to overestimate the clock frequency, and thus waste energy. The rest of this section presents the three developped strategies for lowering the energy of the HEVC decoder.

A. Data level parallelism

In this section, parallelism is exploited at the low level of the application. The source code of the *OpenHEVC* decoder is heavily optimized with SIMD instructions to exploit data-level parallelism available in different blocks of the application. For ARM processor, NEON vector processing instruction set is well suited for performance optimizations. The most time-consuming blocks including motion compensation, inverse transform, deblocking filter and Sample Adaptive Offset (SAO) filter have been optimized with NEON instructions. For concision reason, only the optimization of the SAO is detailed in this paper. This block is not too complex to be described and it perfectly illustrates the NEON optimization capability for this type of processing largely used in video processing.

SAO filter is used to reduce sample distortion by classifying reconstructed sample into category, obtaining offset for category, and then adding that offset to sample value [13]. Each offset is calculated by encoder and explicitly signaled in bitstream. SAO consists of two separate modes selectable by encoder in Coding Tree Unit (CTU) basis: Band Offset (BO) and Edge Offset (EO). For BO the classification of samples is done based on the sample values. For EO classification is done based on neighboring samples. Adding the offset is similar in both modes.

For BO classification, the sample value range is divided into 32 bands. Offsets for four consecutive bands and starting band position are delivered in bitstream to decoder. Classification of sample is done using 5 most significant bits of reconstructed sample value. For 8-bit values the classification is obtained by right shifting each sample value by 3 bits. NEON instruction Vector Shift Right (`VSHR`) can be used for vectorized shift.

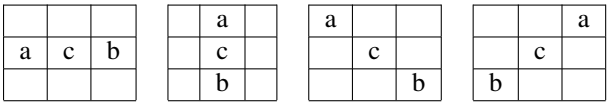

Fig. 4. EO classification modes: Horizontal (EO 0), Vertical (EO 1), 135° diagonal (EO 2) and 45° diagonal (EO 3)

Then offset values can be obtained by using NEON's table look-up instruction `VTBL`. It takes three parameters: destination 64-bit d-register, list of d-registers containing look-up table and d-register with indices. Maximum size for NEON look-up table using four 64-bit d-register is 32. This matches exactly the required length for BO look-up table.

When offset is obtained, it then must be added to sample value. For 8-bit samples the sample value is unsigned value within range from 0 to 255. Offset value is signed and in range from -7 to +7. Saturating addition must be performed to keep the value into range from 0 to 255. Given that ARM NEON instruction set does not have saturating addition of mixed signed and unsigned values, a pre-processing is required before using the saturating addition `VQADD`.

Category	Condition
1	$c < a \wedge c < b$
2	$(c < a \wedge c = b) \vee (c = a \wedge c < b)$
3	$(c > a \wedge c = b) \vee (c = a \wedge c > b)$
4	$c > a \wedge c > b$
0	None of the above

TABLE I
SAMPLE CLASSIFICATION RULES FOR EDGE OFFSET

Edge Offset mode is more complex. Sample classification is done based on comparison between current and neighboring samples. There are four different EO categorization modes. Encoder will select one mode for each Coding Tree Block (CTB) that enables EO. EO modes are shown in Figure 4. Current sample is marked with letter c. Neighboring samples a and b are used for classification. For given mode, each sample is classified into one of the five categories listed in Table I. Category 0 means that offset is not applied. For other categories the offset is explicitly signaled in bitstream by encoder. Faster way to calculate edge classification is with equation:

$$Category = T_{EIC}[2 + \mathrm{sign}_3(c-a) + \mathrm{sign}_3(c-b)] \quad (3)$$

where the table T_{EIC} is made-up of the following element $\{1, 2, 0, 3, 4\}$ and the sign_3 function is defined as follow

$$\mathrm{sign}_3(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x = 0 \\ -1 & \text{if} \quad x < 0 \end{cases} \quad (4)$$

For NEON the simplest case is EO class 1. With 32 sample wide CTB block it is possible to do whole processing in registers. First a-row of 32 samples is loaded to two consecutive 128-bit NEON registers. Then similarly c-row and b-row are loaded into registers. NEON Vector Compare Greater Than (`VCGT`) instruction can be used to compare sample rows. The behaviour of `VCGT` instruction is shown in Equation (6). Equation (5) shows how this property can be used to calculate sign_3 with NEON using only three instructions, two `VCGT` and one subtraction. Using `VCGT` is better way than calculating sign_3 with subtraction and clipping to {-1,1} range. This would also require three instructions, but also needs two additional NEON registers for limits -1 and 1.

$$\mathrm{sign}_3(c-a) = \texttt{VCGT}(a, c) - \texttt{VCGT}(c, a) \quad (5)$$

with,

$$\texttt{VCGT}(x, y) = \begin{cases} -1 & \text{if } x > y \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

Results of some of these calculation can be reused. Instead of calculating $\mathrm{sign}_3(c-a)$ for each element in next iteration, $-\mathrm{sign}_3(c-b)$ from previous iteration can be used. With NEON, full rows are processed at once in vector registers. Whole b-row can be used as c-row in next iteration. For EO 0 calculated $\mathrm{sign}_3(c-a)$ values may be reused by calculating one extra $\mathrm{sign}_3(c-a)$ value per row and then using `VEXT` command to extract all first elements to new vector. The content of the new vector is $-\mathrm{sign}_3(c-b)$. Diagonal EO class 2 and 3 are more complex to optimize for NEON. Content of c-row won't be the same in next iteration as was content of b-row. Instead, one additional sample is required to the beginning (EO 2) or to end of the row (EO 3). This extra sample must be loaded to additional vector register and taken into use with Vector Extract command `VEXT` when required. This, in addition to large CTB block sizes, may require more NEON registers than are available at one time.

General NEON optimization methods have been used for SAO optimization. Interleaving instructions `VTBL` with 32 sample look-up table requires 3 CPU cycles. Due to NEON dual pipeline this leaves room for other instructions. `VADD` and `VQADD` instructions can be interleaved with `VTBL` instructions to fill these bubble cycles. Loaded data and calculated results are reused when possible. For load and store operations, data must be aligned for cache line size.

B. Task level parallelism

The frame-based parallelism allows to simultaneously process multiples frames, regardless of the coding configuration, under the restriction that the motion compensation dependencies are satisfied [19]. Figure 5 illustrates the concept of the frame-based parallelism in the HEVC standard where all frames within a group of pictures (GOP) are decoded in parallel. As shown in Figure 5, frame-based parallelism requires communication between threads decoding different frames in order to ensure that the prediction unit (PU) used as reference for inter prediction is already available (decoded) in the reference frame. Therefore, the encoding/decoding of n frames can be carried out in parallel with a delay in the Inter coding configuration. The performance of the frame-based parallelism strongly depends on the coding structure and the ranges of the motion vectors. Moreover, the frame-based parallelism improves the decoding frame rate but not

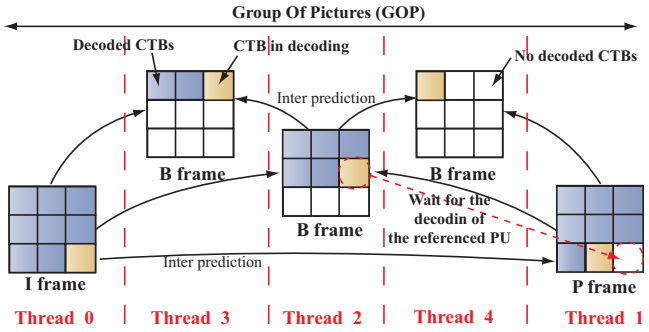

Fig. 5. Principle of the frame-based parallelism in HEVC

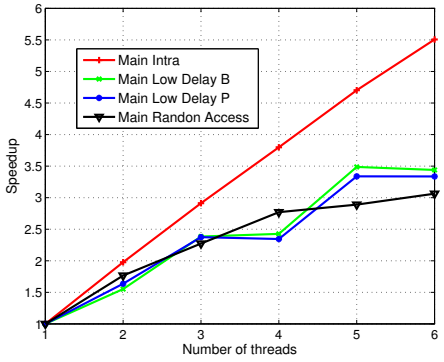

Fig. 6. Speedup performance of the frame-based parallelism implementation in the $OpenHEVC$ decoder run on a general purpose processor

the decoding frame latency.

When the frame-based parallelism is enabled in *OpenHEVC*, n instances of the decoder are created, with n the number of decoding threads. Each decoder has its own local and global structures, while some information in the global structure are shared between the n decoders, such as the Decoded Pictures Buffer (DPB) and the lists related to the video headers (VPS, SPS and PPS lists). An inter thread control solution is implemented to ensure that the PU required for motion compensation is decoded at the reference frame. Otherwise, the decoding thread waits until the PU is decoded in the reference frame. Once the required PU is decoded, all waiting threads are unblocked to pursue the decoding process.

The implementation of the frame-based parallelism in *OpenHEVC* requires more memory allocation than the wavefront and tile implementations. This is because n copies of both local structure and global structure, except the shared structures including the DPB, are allocated for the n parallel decoders. Moreover, the memory size of the global structure without the DPB represents 30% of the decoder memory including a DPB with four pictures. However, this implementation manages only one DPB, which represents 60% of the decoder memory in single thread configuration. The speedup performance of the frame-based parallelism in the *OpenHEVC* decoder on an Intel 6 cores processor (1 thread per core) is shown in Figure 6 for four coding configurations: Main Intra, Main Low Delay B, Main Low Delay P and Main Random Access. In the Intra Main configuration, all frames are Intra coded without motion compensation prediction and then can be decoded in parallel without any delay. Thus, in such a configuration, the speedup is near optimal (ie. close to the number of threads selected for the decoding). However, for the Inter coding configurations, the speedup is decreased by the delay required to complete the decoding of the PUs in the reference frame. Inter-thread communication may also affect the performance of the frame-based solution in Inter coding configurations.

C. Efficient DVFS for video decoding application

As explained in the previous section, data level parallelism and task level parallelism reduce the decoding time. This execution time reduction is translated into an energy consumption decrease thanks to DVFS which adapts the clock frequency and the voltage supply to the processing load. However, predicting the complexity of the forthcoming frame to decode can be a real challenge. In Figure 7, the number of cycles needed to decode a bitstream is plotted frame by frame.

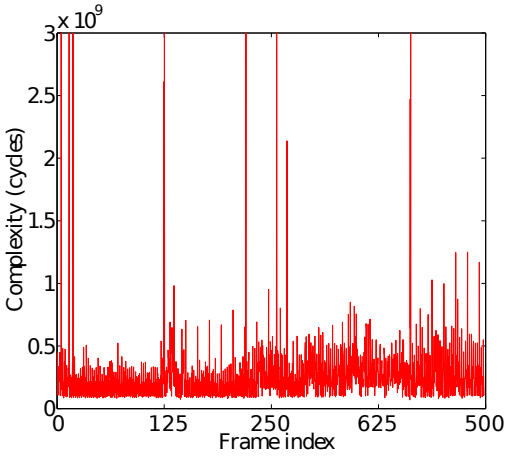

Fig. 7. Decoding complexity for the 480p QP27 sequence called BasketballDrill

Since the *ondemand* governor makes no assumption on the application nature, it is proposed to utilize the video decoding information to do an optimized DVFS video decoder. If the CPU frequency is changed arbitrarily, the decoding of a complex picture may not be completed within the display rate constraint. Hence it is proposed to add modules to predict the complexity of the on-going decoding sequence and handle very high complex decoding with dedicated frame buffering. This Playing Aware DFVS (PAD) is able to adapt to the processing load variation.

Design an energy efficient video decoder is only possible if actual coding procedures are taken into account. As per complexity, video coding tools are sorted out into two categories: reference frame coding and residual coding. Where reference frame coding generates highly complex frame to decode, the residual information can take much less operation. Figure 8 depicts the PAD decoder. Added to the HEVC decoder, new

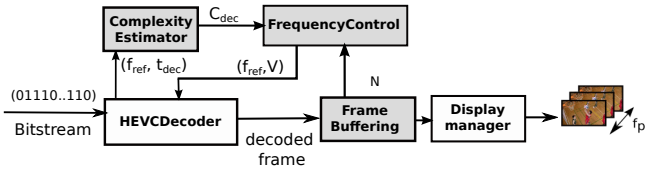

Fig. 8. HEVC decoder using our proposal - Playing aware decoder. New blocks are in grey

blocks are in charge of adapting the video decoding to the actual display rate.

- *ComplexityEstimator* block is in charge of estimating the average complexity of the scene being decoded. It can compute the complexity in cycles from the working frequency of the CPU and the time needed to decode the on-going sequence. Every decoded frame inputs a low pass filter to remove the highly complex frame to decode. As such, it tracks dynamically the residual coding elements. The function is described in Algorithm 1.

Input: Current frequency f_{ref}; Decoding time in seconds T_{dec}
Param: Sliding window length L (in frames)
Output: Frame Complexity in cycles C_{dec}

for $i \leftarrow (L-1)$ **to** L **do**
  $C(i) \leftarrow C(i+1)$
end
$C(L) \leftarrow f_{ref}.T_{dec}$ // Update the sliding window ;
$C_{dec} \leftarrow (\sum_{i=1}^{L}(C(i)))/L$;

Algorithm 1: Estimation of a frame decoding complexity based on the average decoding time

- *FrequencyControl* block adapts dynamically the working frequency to align with the decoding scene. As input parameters, it can handle all the possible working frequencies of the under-lying platform. The primary purpose of this block is to minimize the slack time between the decoding function and the display manager. This block computes the adapted reference frequency f_{ref} and associated voltage V. Algorithm 2 describes this process.
- *FrameBuffering* block stores several frames to cope with sudden complex frames to decode. Hence it ensures the real time behavior of the system ensuring the steady video play back. Evaluating the gauge of the playback buffer, it can also send an information to the *FrequencyControl* block when the level of available frames to display becomes critically low. It asks for a change a working frequency to speed up processing accordingly. It happens especially when the Intra coded frames are being decoded. As two Intra coded frames can barely be consecutive, the real time can be guaranteed. This processing is described in Algorithm 3.

Input: Remaining number of frames N; Frame Complexity in cycles C_{dec};
Param: F_{Disp} frame rate playing demand in fps; Available frequencies in the SoC in Hz $[f_{min}, f_1, ..., f_{max}]$; Frame buffer high threshold T_h; Frame buffer low threshold T_l
Output: Frequency the next period f_{ref}

$f_{opt} \leftarrow C.F_{Disp}$ // eq. (1);
$f_{ref} \leftarrow min[f_{min}, f_1, ..., f_{max}]$ s.t $f_{ref} \geq f_{opt}$ // eq. (2);
if $N > T_h$ **then**
  $f_{ref} \leftarrow f_{ref} - 1$ // the buffer is nearly full;
end
if $N < T_l$ **then**
  $f_{ref} \leftarrow f_{ref} + 1$ // the buffer is nearly empty;
end

Algorithm 2: Frequency control process

Input: Last decoded frame $DecodedFrame$;
Param: Size of the frame buffer $BufferSize$
Output: Pointer of the next frame to display $DisplayPt$; Remaining number of frames in the buffer N;

$DecodedFramePt \leftarrow StoreFrame(DecodedFrame)$;
$DisplayPt \leftarrow UpdateDisplayPt(DisplayPt)$;
$N \leftarrow DecodedFramePt - DisplayPt$;
if $N < 0$ **then**
  $N \leftarrow 0$ // the buffer is empty, a deadline is missed;
end

Algorithm 3: Output frame buffer management

V. Results and analysis

A. Experimental configuration

In our experiments, an octa-core Exynos 5410 SoC is used. This SoC based on the big.LITTLE configuration with four ARM Cortex-A15 cores and four ARM Cortex-A7 cores providing a set of 17 DVFS configurations from 250 MHz to 1.6 GHz. Only four cores can run at a time. From 250 MHz to 600 MHz, the SoC uses the Cortex-A7 cluster. From 800 MHz to 1.6 GHz, the Cortex-A15 cluster is used.

The main characteristics of the platform, in terms of hardware and software, are given in Table II.

SoC	Samsung Exynos5 Octa 5410 ARM Cortex-A15 Quad 1.6 GHz max. ARM Cortex-A7 Quad 1.2 GHz max.
Memory	2GB LPDDR3 @ 800MHz
OS	Ubuntu 14.04 with GCC 4.8.2
HEVC decoder	OpenHEVC branch shm6.1, version fc26f36...

TABLE II
Experimental configuration

1) Power measurement: Power measurement have been performed on the platform using external power sensors. There

are four individual sensors measuring the power consumption of the Cortex-A15, the Cortex-A7, the GPU, and the memory (on-package DRAM). GPU power consumption has not been taken into account as it is not used at all, thus the presented power figures correspond to the sum of the three others sensors. Power measurements are sampled with a period of 100 ms. Two different techniques have been implemented, one based on a shell script, the other based on a dedicated thread integrated in a wrapper around the decoder. The latter technique has been used for the power measurement presented in Section V-D to reduce impact on system performance.

2) HEVC bitstreams: In order to conduct our experiments in a well-defined environment, the bitstreams from the common test conditions defined in [6] are used. These bitstreams are available on the BBC website [3]. Experiments are conducted on 8 bit (low complexity) and random access bitstreams where I and B frame types are present, i.e. both intra and inter prediction have been used for the encoding. In Table III, the selected bitstreams used for the experiments are detailed, All of these bitstreams are 10 seconds long, and when not specified have a QP of 27.

Class	Sequences	Resolution	Frame rate (Hz)
B	*Kimono (K)*	1920x1080	24
	Cactus (C)		50
C	*BasketballDrill (BD)*	832x480	50
	PartyScene (PS)		50
	BQMall (BQM)		60
	RaceHorses (RH)		30
E	*KristenAndSara (KAS)*	1280x720	60

TABLE III
VIDEO SEQUENCES CONSIDERED IN THE EXPERIMENTS.

B. Energy reduction with data level parallelism

In this part, the energy consumption saving and performance obtained by data level parallelization is analyzed. In order to measure it, experiments have been conducted using one of the four cores of the big.LITTLE processor at maximum frequency (i.e. using the ARM Cortex-A15 core running at 1.6 GHz) on Class C bitstreams. This choice has been done because this configuration illustrates the need of data level optimizations to reach real-time video decoding on a mono-core architecture. It can be noticed here that process binding to one core has been done explicitly to ensure stable results. Indeed, unstability can be typically caused by the operating system which can move some threads during execution.

Results for all data-level optimizations are presented in Table IV. These figures represent the energy consumption and time saving (i.e. speedup) in percentage relatively to non-optimized version. Table V details the same figures per functional block where SAO stands for Sample Adaptive Offset filter, DF for Deblocking Filter, IDCT for Inverse Discrete Cosine Transform, MC cpel and lpel for Motion Compensation on chroma and luma, respectively.

Bitstream	Energy Saving	Speed Up
BasketballDrill	40.40%	36.79%
PartyScene	43.66%	39.96%
BQMall	45.96%	41.87%
RaceHorses	45.17%	41.16%

TABLE IV
TOTAL ENERGY SAVING AND SPEED UP OVER NON-OPTIMIZED DECODER THANKS TO DATA-LEVEL PARALLELISATION

.

As depicted in Table IV the total energy consumption saving obtained is between 40.4% and 46%. Table V pinpoints that the most energy saving is obtained thanks to motion compensation optimization on luma with up to 33.49%. As mentioned in [7], this functional block is known to be one of the most time consuming function block. This function block corresponds to a 8-tap motion compensation filter function.

The same tendency as for the energy saving is obtained for the associated speedup. When all the optimizations are used, the energy saving is greater than time saving. This is due to the contribution of motion compensation optimization which results in a greater energy saving than time saving in opposition to all others functional blocks.

C. Energy reduction with task level parallelism

In this part, task level parallelism effect on performance and energy consumption is analyzed. Task level parallelism is used in addition of data level parallelism. At task level, the frame-based parallelism allows to simultaneously process multiple frames in parallel which allows to take advantage of the multi-core architecture of recent processors. In OpenHEVC decoder, this feature has been implemented by reusing FFmpeg runtime which offers a generic way to support frame-based multi-thread execution. As in previous experiments, an explicit process binding on processor cores has been done to ensure more deterministic behavior. For multi-thread execution, the following binding rules have been defined. If the number of threads is less or equal than the number of cores then the application is bound on the number of threads (starting from 0). As an example, an execution using two threads will be bound on cores ID 0 to 1. If the number of threads is greater than the number of cores then all cores are used.

The speedup according to the number of threads using the performance Linux governor with no frame rate constraint (i.e. real time decoding scenario) is presented on Figure 9. These results show two interesting points. First, it shows that the OpenHEVC decoder is able to take advantage of up to 64 threads which is more than the 4 available cores (using more threads cause memory overflow). This is achieved thanks to the high level of parallelism offered by the HEVC frame-based parallelism combined with its implementation through the FFmpeg runtime. Second, as the speedup grows up to 64 threads, the slack-time for a whole sequence is also increased,

Bitstream	Optimized Functional Block									
	SAO		DF		IDCT		MC cpel		MC lpel	
	ES	SU	ES	SU	ES	SU	ES	SU	ES	SU
BasketballDrill	7.37	7.50	4.27	4.26	4.05	4.52	2.85	3.72	23.89	22.06
PartyScene	3.30	4.09	2.01	2.19	2.78	3.26	3.66	4.50	31.88	29.53
BQMall	4.73	5.22	2.58	3.02	2.30	2.80	3.76	4.62	33.49	30.46
RaceHorses	7.87	9.19	3.52	3.82	3.54	3.97	1.90	3.43	27.74	25.26

TABLE V
ENERGY CONSUMPTION SAVING (ES) AND SPEED UP (SU) IN % OVER NON-OPTIMIZED VERSION THANKS TO DATA-LEVEL OPTIMIZATION PER FUNCTIONAL BLOCK

.

so this could be an interesting way to save energy in a real time decoding scenario.

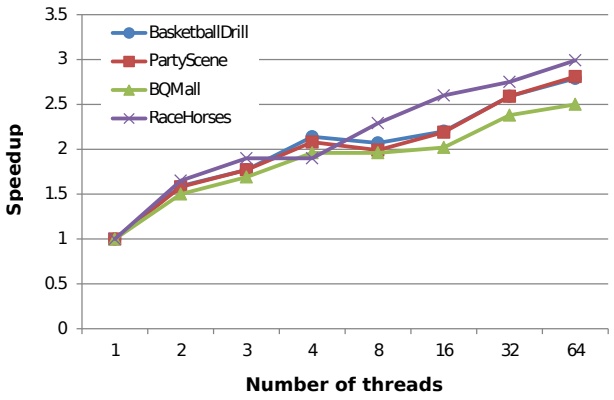

Fig. 9. SpeedUp over sequential execution according to the number of threads using performance Linux governor with no frame rate constraint. The decoding architecture is an ARM big.LITTLE embedded processor

The energy consumption obtained according to the number of threads using *ondemand* Linux governor under the source frame rate constraint is presented in Figures 10. When using 1 thread (i.e. one core), no DVFS can be used as the highest frequency is needed to be able to perform real time decoding but, DPM is applied as the 3 other cores are unused. When using from 2 to 3 threads the system makes use of both DVFS and DPM, 34 % of energy consumption saving is obtained with 2 threads on PartyScene. With 4 and more threads a saving of up to 46 % is obtained on the same sequence. In these case all cores operate, thus only DVFS can be used. But, in practice, in these last cases, the frequency stay most of the time at the highest frequency. This behavior can be explained by thread management which costs in terms of computation and context switching leading to a higher CPU load. So, unlike the fact that more slack time is available with more than 4 threads, it clearly appears that using more than 4 threads does not save energy with the *ondemand* governor. The next section presents an alternative technique based on application aware DVFS mechanism.

D. Energy reduction with efficient DVFS

To exploit efficiently the slack-time provided by data-level and task-level parallelism, a Playing Aware DVFS (PAD) is

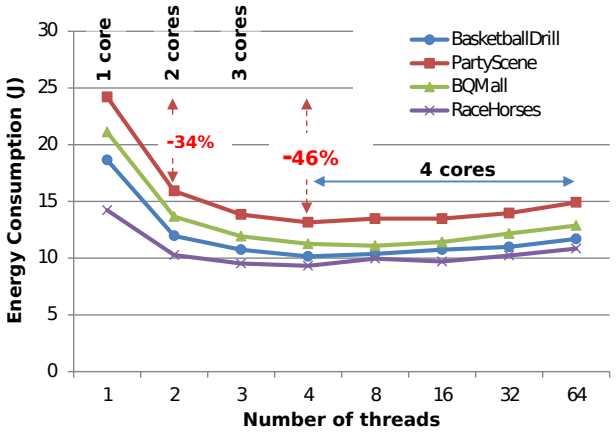

Fig. 10. Energy Consumption (in J) according to the number of threads using *ondemand* Linux governor in real time decoding scenario

used. By taking into account the application itself further energy reduction can be obtained.

In this section, the performance of the PAD decoder is analyzed and compare to the legacy *ondemand* governor. For the different class C bitstreams, the energy consumption is reported in Joules in Table VI for the legacy *ondemand* (OD) governor and for the PAD technique using the best multi-thread configuration for the latter. The energy saving using the PAD is reported relatively to the *ondemand* governor.

Table VI shows that the *ondemand* governor optimize the clock frequency and the supply voltage for the targeted platform. Several parameters improve the efficiency of the system. At first, changing the QP to a less complex video offers energy reduction with significant gain. For example, when the PartyScene sequence is decoded from QP22 to QP37, the energy drops from 23.2 J to 8.6 J with *ondemand* governor. In a similar manner, the proposed PAD decoder offer similar gains with a energy consumption from 17.7 J to 4.1 J. Secondly, reducing the display rate is also beneficial to reduce the energy consumption. For example, the RaceHorses bitstream is sampled at 30 fps where the BQMall is sampled at 60 fps. Hence the RaceHorses bitstream uses 4.9 J at QP37 where the BQMall uses 9.4 J for the same QP.

Furthermore, the PAD decoder outperforms the *ondemand* decoder as per energy efficiency. The energy savings can reach

Bitstream	QP	OD (J)	PAD (J)	Energy Saving (%)
BasketballDrill	37	8.5	4	52.94
	32	9.7	4.4	54.64
	27	11.5	5.2	54.78
	22	15	8.9	40.67
PartyScene	37	8.6	4.1	52.33
	32	11.2	4.8	57.14
	27	14.7	8.8	40.14
	22	23.2	17.7	23.71
BQMall	37	9.4	4.5	52.13
	32	10.1	5	50.50
	27	12	6.9	42.50
	22	15.4	12.7	17.53
RaceHorses	37	4.9	3.2	34.69
	32	5.7	3.9	31.58
	27	9.8	4.2	57.14
	22	13.4	5.3	60.45

TABLE VI
ENERGY CONSUMPTION (IN J) WITH ONDEMAND (OD) GOVERNOR AND PAD MECHANISM

up to 60% in the reported test cases. The PAD decoder makes the most of the a-priori knowledge of the playing requirements of the sequence. As such, the working frequency is close to the most appropriate frequency to decode the sequence acknowledging the display rate. The results in Table VI show the savings are of the same order whatever the content is or whatever the system characteristics (display rate).

E. Results Summary

The gain of the first two optimizations, i.e. at data-level and task-level, can be considered in two ways. Firstly, it saves energy by reducing the decoding time when decoding as fast as possible by making good use of the platform SIMD and SMP features. Secondly, it offers slack-time in real-time scenario that is used by efficient DVFS, i.e. the PAD, by taking advantage of the DVFS and the big.LITTLE architecture of the platform.

It results, for data-level parallelism optimization, a total energy consumption saving (offline decoding) on one core compared to non-optimized code between 40% and 46%. Task-level parallelism gives, compared to a mono-core execution (real-time decoding), from 34% to 46% of energy saving when using native DVFS technique. The proposed enhanced DVFS allows up to 60% of energy saving compared to native one. By integrating all the optimizations, energy consumption is divided by a factor of between 4.3 and 5.9.

F. Comparison with state of the art decoders

OpenHEVC is able, on our platform, to decode a 1080p HEVC bitstream at 50 fps with a QP equal to 27 (Cactus bitstream) in real time which represents the highest performance that can be reached at the time of writing. In this case, the 4 Cortex-A15 are working at full speed, the measured energy consumption is closed to 70 nJ/px. When using the 4 Cortex-A7 at full speed, a 1080p sequence at 24 fps with a QP equal to 37 (Kimono bitstream) can be decoded in real-time with a consumption of 17 nJ/px. It should be noted that there is no energy saving opportunity as all cores are working and no slack-time is available, thus DVFS and DPM techniques are useless.

The energy efficiency of the proposed HEVC decoder associated with PAD is compared to state of the art decoders, hardware and software, in Figure 11. It is obvious that ASIC decoders surpasses all software ones. On the software side, it shows that our decoder has a better energy efficiency compared to other software implementation [10] at 1080p even though no PAD can be used. For lower resolution the energy consumption is under 40 nJ/px except for RaceHorses which is known to be a complex bitstream to decode as it requires higher bitrate than others at same resolution.

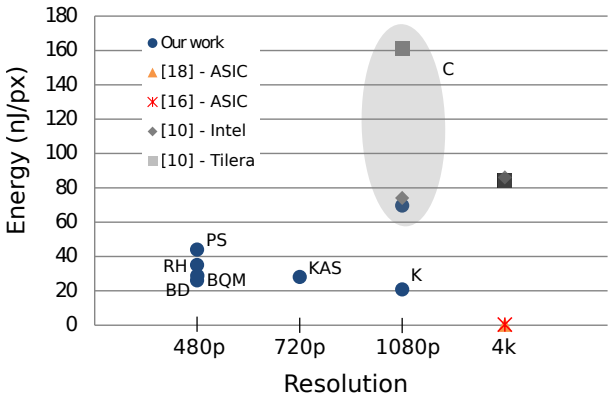

Fig. 11. Energy efficiency (in nJ/px) of the proposed solution compared to state of the art decoders

VI. CONCLUSION

In this paper, three optimization strategies are presented to reduce the energy consumption of a software HEVC decoder. These strategies, respectively based on data parallelism, task parallelism and DVFS, are applied to the open source OpenHEVC decoder and lead to an energy consumption below 21 nJ/px for HD decoding at 2.2 Mbits/s on a multi-core ARM big.LITTLE processor.

Future work include testing the optimized decoder over different hardware architectures and investigating other parallel processing solutions from the HEVC standard to study their impact on energy consumption.

ACKNOWLEDGMENT

This work is partially supported by BPI France, Region Ile-de-France, Region Bretagne and Rennes Metropole through the GreenVideo Project and by European Celtic-Plus through the H2B2VS Project.

The authors would like to thank all the contributors of the OpenHEVC open source project leaded by Mickaël Raulet.

REFERENCES

[1] FFmpeg: Open source and cross-platform multimedia library. In *http://www.ffmpeg.org*.

[2] Open source HEVC decoder (OpenHEVC). In *https:://github.com/OpenHEVC*.

[3] BBC HEVC bistreams: . In *ftp://ftp.kw.bbc.co.uk/hevc/hm-15.0-anchors/*.

[4] GALAXY S4 Teardown: . In *http://www.techinsights.com/inside-samsung-galaxy-s4/, 2013*.

[5] High Efficiency Video Coding. In *Rec. ITU-T H.265 and ISO/IEC 23008-2*. Sapporo, JP, January 2013.

[6] F. Bossen. *Common Conditions and Software Reference Configurations*. Document JCTVC-H1100, Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, San Jose, CA, Feb. 2012.

[7] J. F. Bossen, B. Bross, K. Suhring, and D. Flynn. HEVC complexity and implementation analysis. *IEEE Transactions on Circuits and Systems for Video Technology*, 22: 1685–1696, December 2012.

[8] B. Bross, M. Alvarez-Mesa, V. George, C.Chi-Ching, T. Mayer, B. Juurlink, and T. Schierl. HEVC real-time decoding. In *SPIE Conference on Applications of Digital Image Processing*. San Diego, California, August 2012.

[9] T. D. Burd and R. W. Brodersen. Energy efficient cmos microprocessor design. In *System Sciences, 1995. Proceedings of the Twenty-Eighth Hawaii International Conference on*, volume 1, pages 288–297. IEEE, 1995.

[10] C. Chi, M. Alvarez-Mesa, J. Lucas, B. Juurlink, and T. Schierl. Parallel hevc decoding on multi- and many-core architectures. *Journal of Signal Processing Systems*, 71(3):247–260, 2013. ISSN 1939-8018. doi: 10.1007/s11265-012-0714-2. URL http://dx.doi.org/10.1007/s11265-012-0714-2.

[11] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schier. Parallel Scalability and Efficiency of HEVC Parallelization Approaches. *IEEE Transactions on Circuits and Systems for Video Technology*, 22:1827–1838, December 2012.

[12] G. Clare, F. Henry, and S. Pateux. Wavefront Parellel Processing for HEVC Encoding and Decoding. In *document JCTVC-F274*. Torino, Italy, Jully 2011.

[13] C.-M. Fu, E. Alshina, A. Alshin, Y.-W. Huang, C.-Y. Chen, C.-Y. Tsai, C.-W. Hsu, S.-M. Lei, J.-H. Park, and W.-J. Han. Sample adaptive offset in the hevc standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 22(12):1755–1764, Dec 2012. ISSN 1051-8215.

[14] W. Hamidouche, M. Raulet, and O. Deforges. Parallel shvc decoder: Implementation and analysis. In *Multimedia and Expo (ICME), 2014 IEEE International Conference on*, pages 1–6, July 2014. doi: 10.1109/ICME.2014.6890300.

[15] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava. Power optimization of variable voltage core-based systems. In *Design Automation Conference, 1998. Proceedings*, pages 176–181, 1998.

[16] C.-T. Huang, M. Tikekar, C. Juvekar, V. Sze, and A. Chandrakasan. A 249mpixel/s hevc video-decoder chip for quad full hd applications. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2013 IEEE International*, pages 162–163, Feb 2013. doi: 10.1109/ISSCC.2013.6487682.

[17] S. Jafri, M. Tajammul, A. Hemani, K. Paul, J. Plosila, and H. Tenhunen. Energy-aware-task-parallelism for efficient dynamic voltage, and frequency scaling, in cgras. In *SAMOS XIII*, pages 104–112, 2013.

[18] C.-C. Ju, T.-M. Liu, Y.-C. Chang, C.-M. Wang, H.-M. Lin, C.-Y. Cheng, C.-C. Chen, M.-H. Chiu, S.-J. Wang, P. Chao, M.-J. Hu, F.-C. Yeh, S.-H. Chuang, H.-Y. Lin, M.-L. Wu, C.-H. Chen, and C.-H. Tsai. A 0.2nj/pixel 4k 60fps main-10 hevc decoder with multi-format capabilities for uhd-tv applications. In *European Solid State Circuits Conference (ESSCIRC)- 40th*, pages 195–198, Sept 2014. doi: 10.1109/ESSCIRC.2014.6942055.

[19] C. Meenderinck, A. Azevedo, M. Alvarez, B. Juurlink, and A. Ramirez. Parallel scalability of video decoders. *Journal of Signal Processing Systems*, 57:173–194, November 2009.

[20] J. R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand. Comparaison of the Coding Efficiency of Video Coding standards including High Efficiency Video coding (HEVC). *IEEE Transactions on Circuits and Systems for Video Technology*, 22:1858–1870, December 2012.

[21] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand. Overview of the high efficiency video coding standard. *IEEE Transactions on Circuits and Systems for Video Technology*, 22:1648–1667, December 2012.

[22] I. Takouna, W. Dawoud, and C. Meinel. Accurate mutlicore processor power models for power-aware resource management. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 419–426, 2011. doi: 10.1109/DASC.2011.85.

[23] M. Tikekar, C. T. Huang, C. Juvekar, V. Sze, and A. P. Chandrakasan. HEVC complexity and implementation analysis. *IEEE Journal of Solid-State Circuits*, 49:1685–1696, Junuary 2014.

[24] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions on Circuit and Systems for Video Technology*, 13(7):560–576, July 2003.