Multi-Level Parallelization Scheme for Distributed HEVC Encoding on Multi-Computer Systems

Sami Ahovainio, Alexandre Mercat, Marko Viitanen, Jarno Vanne
Computing Sciences, Tampere University, Finland
{sami.ahovainio, alexandre.mercat, marko.viitanen, jarno.vanne}@tuni.fi

***Abstract*— High Efficiency Video Coding (HEVC) creates the conditions for cost-effective video transmission and storage but its inherent computational complexity calls for efficient parallelization techniques. This paper provides HEVC encoders with a holistic parallelization scheme that exploits parallelism at data, thread, and process levels at the same time. The proposed scheme is implemented in the practical Kvazaar open-source HEVC encoder. It makes Kvazaar exploit parallelism at three levels: 1) Single Instruction Multiple Data (SIMD) optimized coding tools at the data level; 2) Wavefront Parallel Processing (WPP) and Overlapped Wavefront (OWF) parallelization strategies at the thread level; and 3) distributed slice encoding on multi-computer systems at the process level. Our results show that the proposed process-level parallelization scheme increases the coding speed of Kvazaar by 1.86× on two computers and up to 3.92× on five computers with +0.19% and +0.81% coding losses, respectively. Exploiting all these three parallelism levels on a five-computer setup gives almost a 25× speedup over a non-parallelized single-core implementation.**

Keywords—High Efficiency Video Coding (HEVC), multi-level parallelization, HEVC parallelization strategies, distributed HEVC encoding, multi-computer systems

I. INTRODUCTION

Digital video has become ubiquitous in our everyday life thanks to a myriad of multimedia devices and services that support emerging new video formats such as 4K *Ultra High Definition (UHD)* or 360-degree video. Cisco [1] reports that 75% of total IP traffic is dedicated to video in 2017 and estimates it to grow to 82% by 2022. Over the past decades, MPEG and ITU-T have addressed this explosive growth by announcing a series of video coding standards out of which *High Efficiency Video Coding* (*HEVC/H.265*) [2] represents the state-of-the-art. Currently, MPEG and ITU-T are also seeking coding gain beyond HEVC by developing a new standard called *Versatile Video Coding* (*VVC/H.266*) [3] but the standardization work is still in progress.

This work deals with the widespread HEVC standard, which is published as twin text by ITU, ISO, and IEC as ITU-T H.265 | ISO/IEC 23008-2. HEVC Main profile improves coding efficiency by around 23% and 40% [4] over the preceding *Advanced Video Coding* (*AVC/H.264*) standard [5] in *All-Intra* (*AI*) and *Random Access* (*RA*) configurations, respectively. However, the coding gain comes at a cost of over 3.2× and 1.2× complexity in these configurations [4], which tends to act as a barrier for developing practical applications. Tackling this complexity calls for efficient techniques for HEVC codec parallelization.

Fig. 1 summarizes the parallelization strategies [6] specified for HEVC. As shown in Fig. 1(a), an input video to an HEVC encoder is split into *Groups of Pictures* (*GOPs*) where the first frame acts as a *Random Access Point* (*RAP*), i.e., a frame that an HEVC decoder can decode without referring to any other frames. These GOPs can also be encoded in parallel as no dependencies exist between them.

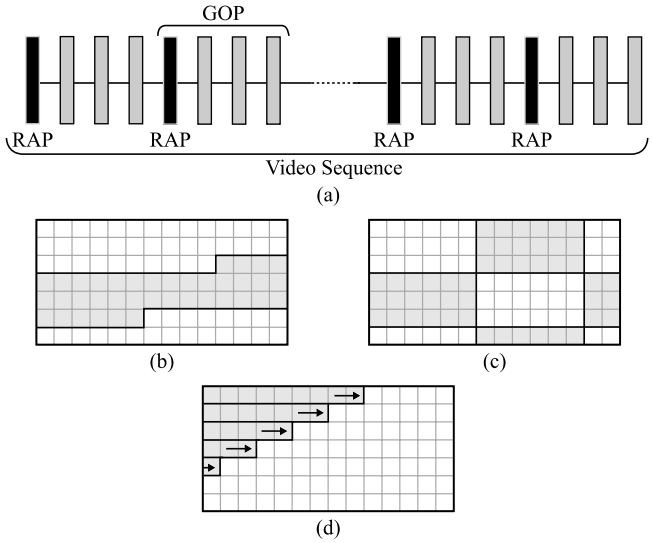

Fig. 1. Overview of the HEVC parallelization strategies. (a) Video sequence partitioning into GOPs. (b) Frame partitioning into slices. (c) Frame partitioning into tiles. (d) WPP processing along CTU rows.

Inside a GOP, a single frame can be further decomposed into one or multiple *slices*, as exemplified in Fig 1(b), where a frame is divided into three slices. A slice, in turn, is composed of consecutive raster-scan ordered *Coding Tree Units* (*CTUs*) that can be decoded independently of the other slices. As shown in Fig. 1(c), a frame can also be divided into *tiles*, which divide a frame into rectangular groups of CTUs. As for slices, tile boundaries break coding dependencies so multiple tiles can be encoded in parallel.

For thread-level parallelization, HEVC introduces *Wavefront Parallel Processing* (*WPP*) [7] illustrated in Fig. 1(d). WPP allows processing multiple CTU rows in parallel without breaking coding dependencies between them. Thus, it introduces less coding loss than slices and tiles. The non-normative *Overlapped Wavefront* (*OWF*) [8] technique extends the execution of WPP across consecutive frames.

As for now, several parallelization approaches have been proposed for HEVC encoders in the literature. Some of them [9], [10] stayed at the process level and distributed the GOP encoding among multiple CPU cores. The existing thread-level approaches [11], [12] used an adaptive workload balancing to allocate tiles between CPU cores. Authors in [13], [14] proposed parallelized the most time-consuming coding tools at the data level.

In this work, we address HEVC encoder parallelization at all practical levels and propose a multi-level parallelization scheme to make the most of distributed HEVC encoding on multiple computers. The proposed scheme is designed for the open-source Kvazaar HEVC encoder [15] due to its capacity

This work was supported in part by the European Celtic-Plus project VIRTUOSE and the Academy of Finland (decision no. 301820). The authors would also like to thank all contributors of the Kvazaar open-source project.

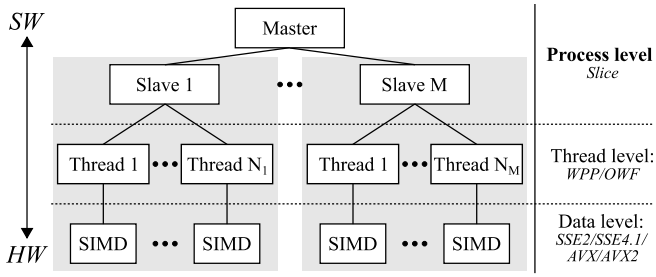

Fig. 2. Overview of the parallelization levels in the proposed solution.

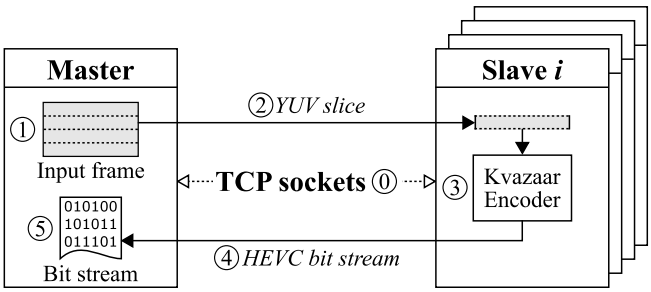

Fig. 3. Overview of the distributed encoding on multi-computer systems.

of practical encoding. This work extends Kvazaar parallelization to multi-computer systems and makes Kvazaar exploit parallel encoding at the following three levels:

1) **Data level**: the most time-consuming coding tools are optimized for *Single Instruction Multiple Data* (*SIMD*) processor extensions.
2) **Thread level**: the encoding process of CTU rows is parallelized on multiple CPU cores with WPP and OWF parallelization techniques.
3) **Process level**: the encoding process of HEVC slices is parallized on multiple computers by using *Transmission Control Protocol* (*TCP*) over *10 Gigabit Ethernet* (*10GbE*) for communication.

To the best of our knowledge, our proposal is the first open-source solution, which enables HEVC encoder to exploit all these three levels of parallelism simultaneously. It is available online at *https://github.com/ultravideo/kvaShare/* and presented through a live demonstration [16].

The remainder of the paper is outlined as follows. Section 2 gives an overview of the related work. The data- and thread-level parallelization techniques of Kvazaar are described in Section 3. Section 4 introduces our process-level solution for distributing encoding process on a multi-computer system. Section 5 describes our experimental setup and results. Finally, Section 6 concludes the paper.

II. Related Work

Since the HEVC standard was finalized, many parallelization approaches have been published in the literature to speed up HEVC encoding. They can be classified into data-, thread-, and process-level approaches.

At the data level, Yan et al. proposed a parallel framework for HEVC *motion estimation* (*ME*) on a many-core platform [13]. Later, they applied *Direct Acyclic Graph* (*DAG*) and extended their framework to support CTU-level parallelization [17]. Radicke et al. [18] accelerated ME on a *Graphics Processing Units* (*GPUs*). Heng et al. [14] combined parallel GPU implementation of ME-, slice-, and GOP-level parallelization at the CPU level. GPU motion estimation was done in a separate computer which also handled input reading, preprocessing filter and transmitting the input frames to the encoders.

At the thread level, Khan et al. [11] presented a parallel architecture to process tiles on multiple CPU cores. They used online workload allocator to dynamically adapt the tile splitting in order to obtain better load balancing between cores. Similarly, Storch et al. [12] used the workload of previous frames to dynamically define the vertical and horizontal boundaries of tiles.

At the process level, Migallon et al. [9] made use of GOP independency and presented a parallel GOP-based strategy for distributed memory platforms. They used *Message Passing Interface* (*MPI*) programming paradigm to encode several GOPs simultaneously. Later, they extended the work with thread-level parallelism in HEVC intra coding [10].

III. Data- and Thread-Level Parallelization

Fig. 2 summarizes the parallelization levels of the proposed scheme. They are all implemented in the open-source Kvazaar HEVC encoder [15].

A. Data-Level Parallelization

At the data level, the most time-consuming coding tools are optimized with Intel Intrinsics or x86-64 assembly instructions. Kvazaar includes a dynamic dispatch mechanism for SIMD-optimized functions, i.e., it automatically selects the best SIMD optimizations according to the instruction sets supported by the CPU. Optimizations for SSE2, SSE4.1, AVX, and AVX2 are supported, but the majority of them are for AVX2 [19].

B. Thread-Level Parallelization

At the thread level, Kvazaar implements the WPP and OWF parallelization strategies [20] through multithreading. When WPP is enabled, multiple CTU rows of the frame can be encoded in parallel. Processing of a CTU row can be started once the first two CTUs of the previous row have been encoded. WPP complies with the regular raster scan order and does not break the encoding dependencies.

To improve the ramping inefficiencies of WPP [6], the non-normative OWF technique overlaps the execution of consecutive frames. When a CTU row is encoded and no more rows are available in the current frame, a new CTU row can be taken from the next frame instead of waiting for completion of the current frame. Kvazaar automatically sets the threads and OWF values for maximum CPU utilization.

IV. Parallelization on Multi-Computer Systems

This work extends Kvazaar parallelization to multi-computer systems at the process level. HEVC slice partitioning is applied to distribute the encoding process among the computers. The number of slices per frame is made equal to that of the computers and the co-located slices in the frames are always assigned to the same computer. Each slice is encoded independently for maximum parallelization. The proposed technique is built on top of the data and thread-level parallelization, as described in Fig. 2.

As shown in Fig. 3, the proposed setup is composed of a single master and multiple slave processes. The master takes

an input video in the YUV format, assigns encoding tasks to the slaves, and finally outputs the HEVC bitstream. Communication between the master and slaves is handled through a TCP/IP socket connection over 10 GbE links. The encoding process includes the following six steps (S0-S5) out of which the steps S1-S5 are iterated on each video frame:

S0) **Initialization**. The socket connections are created between the master and slaves after which the master sends the coding parameters, such as the input frame size and the associated slice offset, to the slaves.

S1) **Frame splitting**. The master splits the input YUV video into M rectangular slices frame-by-frame, M being the number of available slaves. Rectangular slices are adopted since the slaves can process them like regular input sequences. To maximize the load balancing, the slice height h_i for the slave i is defined as

$$h_i = \begin{cases} \left\lfloor \frac{H}{\eta \times M} \right\rfloor \times \eta & \text{if } i \text{ is odd and } i < M, \\ \left\lceil \frac{H}{\eta \times M} \right\rceil \times \eta & \text{if } i \text{ is even and } i < M, \\ H - \sum_{i=0}^{i<M-1} h_i & \text{otherwise,} \end{cases} \quad (1)$$

where $i \in \{0, M-1\}$, η is the height of the CTU (in this work, $\eta = 64$), and H is the height of the input YUV video. The width of all slices equals that of the input.

S2) **Raw data transmission**. The master sends the slices to each slave frame-by-frame.

S3) **Slice encoding.** Each slave executes an independent instance of Kvazaar, which is slightly modified to produce custom headers. A slave starts the encoding process as soon as it receives a slice from the master. The proposed solution allows encoding each slice independently without any synchronization between the slaves. The implemented data- and thread-level parallelization techniques ensure maximum CPU utilization in each computer.

S4) **Bitstream transmission.** After a slave is ready, it sends the HEVC bitstream to the master. Each slave outputs a full-sized frame containing only valid slice data and the rest of the frame is left empty.

S5) **Bitstream parsing.** The master buffers the individual HEVC bitstreams until it has received all slices of a frame. Finally, the master removes the duplicate headers from the bitstream, such as the PPS and SPS included in the first frame, arranges them in the correct order, and outputs a complete bitstream conforming to HEVC.

V. Experimental Results

Kvazaar is currently the front-runner among existing open-source HEVC intra encoders [21] so AI coding configuration of Kvazaar was chosen for our experiments. Kvazaar is able to attain real-time intra coding speed up to 4K resolution on a single 22-core processor with a speed-optimized *ultrafast* preset [22]. In that case, the communication overhead of the process-level parallelization would become more dominant with respect to the overall encoding time, which reduces the motivation for using multi-computer approach. Instead, the benefits of the process-level parallelization become apparent in more computation-intensive use cases of Kvazaar such as in high-quality encoding. Therefore, Kvazaar *veryslow* preset [15] was used in our experiments.

TABLE I. Test Sequences

Sequence name	Number of frame	Frame rate	Resolution
BasketballDrive	500	50fps	1920×1080 (1080p)
BQTerrace	600	60fps	1920×1080 (1080p)
Cactus	500	50fps	1920×1080 (1080p)
Kimono	240	24fps	1920×1080 (1080p)
ParkScene	240	24fps	1920×1080 (1080p)
Traffic	150	30fps	2560×1600 (1600p)
PeopleOnStreet	150	30fps	2560×1600 (1600p)
Jockey	600	120fps	3840×2160 (2160p)
Shake_N_Dry	300	120fps	3840×2160 (2160p)

TABLE II. Profiling Platform

Processor	Intel Core i7 (4 × 3.4GHz)
Core	Hyper-threading
Memory	16/32 GB
Instruction set	AVX, AVX2
Compiler	MS Visual studio 2017 v15.4.2
Operating system	Windows 7/10

A. Experimental Setup

The experiments were conducted on nine test video sequences listed in Table I. They differ in terms of number of frames, frame rate, and spatial resolution.

Our multi-computer test setup was composed of five similar computers connected with 10GbE links. The essential features of these computers are detailed in Table II. During the experiments, the number of computers was varied from one to five. One of the computers was running the master process. It also acted as a slave, since the master process is not computationally heavy.

The coding efficiency was compared in terms of *Bjøntegaard Delta Bit Rate* (*BD-BR*) [23] difference in percent for the same *Peak Signal-to-Noise Ratio* (*PSNR*) using four *Quantization Parameter* (*QP*) values: 22, 27, 32, and 37. The coding speeds of the benchmarked encoder configurations were measured separately for each four QP and the results were averaged. For a reliable comparison, only one encoder configuration was run at a time. Each test run was also repeated four times and the coding speeds were averaged for improved accuracy.

B. Analysis of Data- and Thread-Level Scalability

The first experiments evaluated the efficiency of the data and thread-level parallelization techniques on one computer. The results show that the SIMD optimizations speed up Kvazaar by 1.64× on average compared with its non-parallelized single-core implementation. However, the speed-ups are not consistent across the entire QP range but they variate from 1.37× at QP = 22 to 1.91× at QP = 37. The discrepancy is mainly caused by sequential entropy coding tools whose execution time is dependent on the QP value.

The thread-level parallelization on the four CPU cores achieves an average speed-up of 3.90× regardless of the QP value. Furthermore, the combined speed-up of these two parallelization levels is up to 6.35× on average.

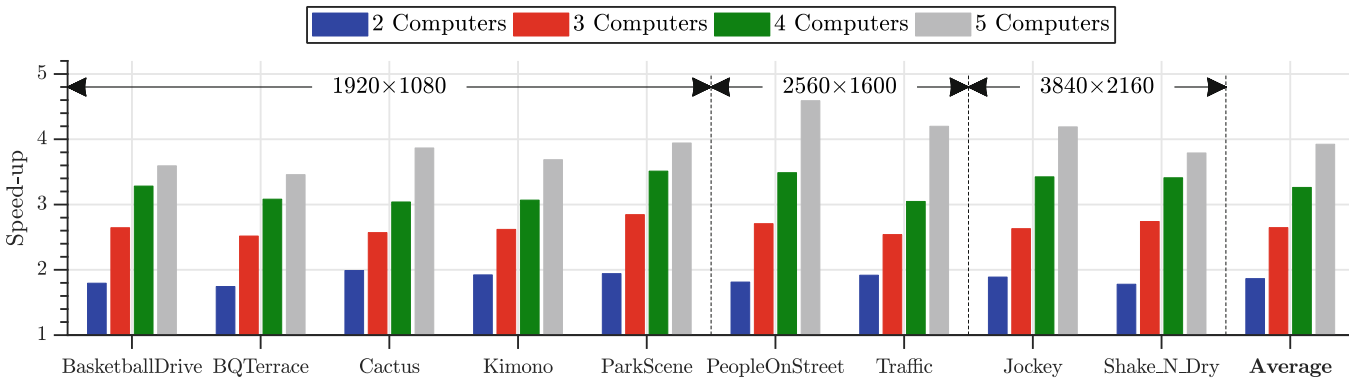

Fig. 4. Speed-up of the proposed solution with differnet number of computers (averaged over the QP values of 22, 27, 32, and 37).

TABLE III. BD-BR OVERHEAD FOR DIFFERENT COMPUTER COUNTS

Sequence	BD-BR			
	Number of computer			
	2	3	4	5
BasketballDrive	+0.35%	+0.72%	+1.01%	+1.46%
BQTerrace	+0.10%	+0.23%	+0.34%	+0.44%
Cactus	+0.20%	+0.51%	+0.73%	+1.14%
Kimono	+0.29%	+0.61%	+0.93%	+1.15%
ParkScene	+0.14%	+0.30%	+0.42%	+0.56%
PeopleOnStreet	+0.16%	+0.31%	+0.47%	+0.62%
Traffic	+0.10%	+0.25%	+0.33%	+0.44%
Jockey	+0.25%	+0.43%	+0.66%	+0.93%
Shake_N_Dry	+0.15%	+0.25%	+0.43%	+0.56%
Average	**+0.19%**	**+0.40%**	**+0.59%**	**+0.81%**

C. *Analysis of Process-Level Scalability*

Fig. 4 presents the speed-up results of the process-level parallelization on two, three, four, and five computers. The anchor was a SIMD-optimized and multithreaded instance of Kvazaar that was executed on a single computer.

The results show that the proposed framework achieved average speed-ups from 1.86× on two computers to 3.92× on five computers. The results are consistent across the QP values with a relative standard deviation of less than 3% on average. The complexity overhead of the process-level parallelization, including frame splitting and data transmission between the computers, is less than 0.1% of the entire encoding time. Hence, it can be considered negligible.

The average parallel efficiency, i.e., the ratio between speed-up and the number of available computers, decreases as a function of the computer count. It reaches 93.3% for two computers and reduces to 88.2%, 81.54%, and 78.48% for three, four, and, five computers, respectively. Moreover, the parallel efficiency is the smallest with the lowest resolution when the number of computers is greater than three. These losses come from the imbalance between slice heights, h_i in (1). In addition, the slices differ in content, which also affects the encoding complexity.

Table III reports the corresponding BD-BR results over a single-computer implementation. The process-level parallelization increases BD-BR linearly from +0.19% on two computers to +0.81% on five computers on average. In AI coding configuration, this overhead comes mainly from signaling the slice header in the bitstream. Hence, it is inherent to HEVC standard and cannot be optimized out.

D. *Comparison with Prior-Art*

A fair comparison with the related parallelization approaches is out of reach in terms of coding speed, BD-BR, and parallel efficiency. The existing solutions implement or combine different levels of parallelization. Moreover, their results are directly linked to the experimental setups, adopted encoders, and hardware platforms, which strongly differ from one solution to another. Closest related works [9] [10] apply MPI to distribute HEVC encoding process on multiple computers. MPI takes care of the data dependencies and communications between processes.

To the best of our knowledge, the proposed framework is the first one that instantiates encoders with separate environments, internal variables, and data. The slave instances are able to perform encoding without any synchronization or data transfers between other instances, regardless of the coding configuration.

Our future work will improve the load balancing between slaves, which currently acts as the main performance bottleneck. Several existing solutions [24], [25] have shown that more sophisticated load balancing techniques can improve performance by up to 10%. The proposed scheme will also be implemented on an asymmetric multicomputer setup that is composed of CPUs with drastically different computational capabilities. These approaches will be experimented with new coding configurations, such as with HEVC inter coding.

VI. CONCLUSION

This paper presented a multi-level parallelization scheme for HEVC encoders by addressing their parallelism at data, thread, and process levels simultaneously. The proposed scheme was implemented in the open-source Kvazaar HEVC encoder. A special attention was paid to the proposed process-level parallelization technique that applies HEVC slice partitioning and distributes the SIMD-optimized and multithreaded Kvazaar among multiple computers.

Our results illustrate that the process-level parallelization accelerates Kvazaar high-quality coding by 1.86× on two computers and by 3.92× on five computers with the respective coding costs of +0.19%, and +0.81%. Moreover, combining all these three parallelism levels together on a five-computer setup speeds up Kvazaar by almost 25× over a non-parallelized single-core implementation of Kvazaar. To the best of our knowledge, the proposed scheme makes Kvazaar the first open-source HEVC encoder that is able to exploit all these three levels of parallelism simultaneously.

REFERENCES

[1] Cisco, Cisco Visual Networking Index: Forecast and Trends, 2017.

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on circuits and systems for video technology,* vol. 22, pp. 1649-1668, 2012.

[3] J.-R. Ohm and G. J. Sullivan, "Versatile Video Coding--towards the next generation of video compression," in *Picture Coding Symposium 2018*, 2018.

[4] J. Vanne, M. Viitanen, T. D. Hamalainen, and A. Hallapuro, "Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs," *IEEE Transactions on Circuits and Systems for Video Technology,* vol. 22, pp. 1885-1898, 2012.

[5] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H. 264/AVC video coding standard," *IEEE Transactions on circuits and systems for video technology,* vol. 13, pp. 560-576, 2003.

[6] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel scalability and efficiency of HEVC parallelization approaches," *IEEE Transactions on circuits and systems for video technology,* vol. 22, pp. 1827-1838, 2012.

[7] G. Clare, F. Henry and S. Pateux, "Wavefront parallel processing for HEVC encoding, and decoding," *document JCTVC-F274. Torino, Italy, Jully,* 2011.

[8] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, V. George and T. Schierl, "Improving the parallelization efficiency of HEVC decoding," in *2012 19th IEEE International Conference on Image Processing*, 2012.

[9] H. Migallón, V. Galiano, P. Piñol, O. López-Granado, and M. P. Malumbres, "Distributed memory parallel approaches for HEVC encoder," *The Journal of Supercomputing,* vol. 73, pp. 164-175, 2017.

[10] H. Migallón, P. Piñol, O. López-Granado, V. Galiano, and M. Malumbres, "Frame-Based and Subpicture-Based Parallelization Approaches of the HEVC Video Encoder," *Applied Sciences,* vol. 8, p. 854, 2018.

[11] M. U. K. Khan, M. Shafique, and J. Henkel, "Software architecture of high efficiency video coding for many-core systems with power-efficient workload balancing," in *Proceedings of the conference on Design, Automation & Test in Europe*, 2014.

[12] I. Storch, D. Palomino, B. Zatt, and L. Agostini, "Speedup-Oriented History-Based Tiling Algorithm for the HEVC Standard Targeting an Efficient Parallelism Exploration," *Journal of Integrated Circuits and Systems,* vol. 13, pp. 1-8, 2018.

[13] C. Yan, Y. Zhang, F. Dai, and L. Li, "Highly parallel framework for HEVC motion estimation on many-core platform," in *2013 Data Compression Conference*, 2013.

[14] T. K. Heng, W. Asano, T. Itoh, A. Tanizawa, J. Yamaguchi, T. Matsuo, and T. Kodama, "A highly parallelized H. 265/HEVC real-time UHD software encoder," in *2014 IEEE International Conference on Image Processing (ICIP)*, 2014.

[15] UltraVideoGroup, "Kvazaar," [Online]. Available: https://github.com/ultravideo/kvazaar.

[16] S. Ahovainio, A. Mercat, and J. Vanne, "Live Demonstration: Multi-Laptop HEVC Encoding," in *IEEE Int. Symp. Circuits Syst.*, Seville, 2020.

[17] C. Yan, Y. Zhang, J. Xu, F. Dai, L. Li, Q. Dai, and F. Wu, "A highly parallel framework for HEVC coding unit partitioning tree decision on many-core processors," *IEEE Signal Processing Letters,* vol. 21, pp. 573-576, 2014.

[18] S. Radicke, J.-U. Hahn, Q. Wang, and C. Grecos, "Many-core hevc encoding based on wavefront parallel processing and gpu-accelerated motion estimation," in *International Conference on E-Business and Telecommunications*, 2014.

[19] A. Lemmetti, A. Koivula, M. Viitanen, J. Vanne, and T. D. Hämäläinen, "AVX2-optimized Kvazaar HEVC intra encoder," in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016.

[20] A. Koivula, M. Viitanen, J. Vanne, T. D. Hämäläinen, and L. Fasnacht, "Parallelization of Kvazaar HEVC intra encoder for multi-core processors," in *2015 IEEE workshop on signal processing systems (SiPS)*, 2015.

[21] A. Mercat, A. Lemmetti, M. Viitanen, and J. Vanne, "Acceleration of Kvazaar HEVC Intra Encoder With Machine Learning," in *2019 IEEE International Conference on Image Processing (ICIP)*, 2019.

[22] A. Ylä-Outinen, A. Lemmetti, M. Viitanen, J. Vanne, and T. D. Hämäläinen, "Kvazaar: HEVC/H. 265 4K30p intra encoder," in *2017 IEEE International Symposium on Multimedia (ISM)*, 2017.

[23] G. Bjontegaard, "Calculation of average PSNR differences between RD-Curves," Austin, 2001.

[24] M. Koziri, P. Papadopoulos, N. Tziritas, A. N. Dadaliaris, T. Loukopoulos, and S. U. Khan, "Slice-based parallelization in HEVC encoding: realizing the potential through efficient load balancing," in *2016 IEEE 18th International Workshop on Multimedia Signal Processing (MMSP)*, 2016.

[25] Y.-J. Ahn, T.-J. Hwang, D.-G. Sim, and W.-J. Han, "Complexity model based load-balancing algorithm for parallel tools of HEVC," in *2013 Visual Communications and Image Processing (VCIP)*, 2013.