Taylor & Francis
Taylor & Francis Group

Automatika

Journal for Control, Measurement, Electronics, Computing and Communications

ISSN: 0005-1144 (Print) 1848-3380 (Online) Journal homepage: https://www.tandfonline.com/loi/taut20

Dynamic load balancing algorithm based on HEVC tiles for just-in-time video encoding for heterogeneous architectures

Leon Dragić, Igor Piljić & Mario Kovač

To cite this article: Leon Dragić, Igor Piljić & Mario Kovač (2019) Dynamic load balancing algorithm based on HEVC tiles for just-in-time video encoding for heterogeneous architectures, Automatika, 60:2, 239-244, DOI: 10.1080/00051144.2019.1618527

To link to this article: https://doi.org/10.1080/00051144.2019.1618527

Published online: 20 May 2019.

AUTOMATIKA
2019, VOL. 60, NO. 2, 239–244
https://doi.org/10.1080/00051144.2019.1618527

REGULAR PAPER

OPEN ACCESS Check for updates

Dynamic load balancing algorithm based on HEVC tiles for just-in-time video encoding for heterogeneous architectures

Leon Dragić, Igor Piljić and Mario Kovač

Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia

ABSTRACT

This paper proposes a novel algorithm for dynamic tile partitioning to achieve the optimal workload balance for parallel processing architectures in just-in-time HEVC encoding. Tile boundaries are dynamically shifted depending on the tile cost, a value that denotes predicted computational complexity of a single tile in a frame. The overall cost of a tile is determined as a combination of costs of three computationally most expensive and resource-hungry operations in HEVC encoding: prediction, transformation, and entropy coding. The algorithm aims at exploiting different types of processing architectures, from homogeneous multicore CPU architectures to heterogeneous architectures in the actual conditions in which streaming servers operate. The experimental results show that the proposed algorithm outperforms uniform tiling, by up to 5.5% in processing time, while maintaining the same video quality and bitrate. Compared to the state-of-the-art algorithms, the proposed algorithm achieves up to 8.85% speedup depending on the number of videos that are being encoded concurrently on a video streaming server.

ARTICLE HISTORY
Received 21 February 2019
Accepted 4 April 2019

KEYWORDS
HEVC; video coding; parallel processing; tile; workload balance; heterogeneous architectures

Introduction

IP video traffic will be 82% of all consumer Internet traffic by 2022, which is an increase from 75% compared to 2017 [1]. High Efficiency Video Coding (HEVC/H.265) is the latest standard that achieves approximately 50% bitrate reduction compared with its predecessor AVC [2], but at the same time increases the computational complexity and resource requirements by up to 10 times [3]. To deal with the computational complexity, HEVC introduces two new parallelization concepts: Tiles and Wavefront parallel processing (WPP). Tiles, performance-wise, outperform other parallelization concepts used in HEVC [4]. The computational complexity and parallelization concepts become even more essential when video processing is constricted with the timing requirement. One of such examples is just-in-time video encoding, a process in which a video source is encoded in real-time. To maximize performance, it is necessary to fully exploit parallel processing architectures and to do that workload must be optimally balanced. In this paper, a novel workload balancing algorithm for Just-In-Time video encoding based on HEVC standard is presented. The algorithm can target different architectures, from homogeneous multicore CPU to heterogeneous accelerator-based architectures.

The algorithm extracts the information from the previously encoded frames to predict workload cost for each tile. Tile cost is based on the number of operations needed to encode the tile in the previous frame. After the cost for each tile is calculated, tile boundaries are adjusted to balance the workload between processing cores.

The rest of the paper is organized as follows: In section two, previous work and related state-of-the-art algorithms are described; In section three, a novel algorithm for workload balancing using tiles in HEVC is presented; In section four, experimental results are shown.

Related work

The parallelization optimization and workload balancing in video encoding have been subject of previous research, but only a few of them focused on tiles and even less considered their implementation for heterogeneous platforms with just-in-time requirements.

In [5], authors adapt tile boundaries to maximize coding efficiency, but only on Intra frames. They use the frame variance map to determine highly correlated regions and group them in tiles. However, all-intra configuration requires more time for frame processing which is not suitable for just-in-time video encoding.

Power efficient workload balancing using tiles is introduced in [6]. The algorithm uses a number of CTUs (Coding Tree Units) as a main parameter for tile distribution, ignoring video content. Such an approach causes less precise balance between tiles since more CTUs do not necessarily imply an increase

CONTACT Leon Dragić leon.dragic@fer.hr Faculty of Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia

in computational requirements. Video sequences used were limited to 832 × 480 pixels or less, while no focus was set on high-resolution sequences.

An algorithm, called Time-based Tile Load Balancing algorithm or TTLB, for workload balancing using tiles, is presented in [7]. It calculates the processing time of each tile and uses it to dynamically adapt tile boundaries. The similar algorithm, that uses processing time as the main parameter, is analyzed in [8]. Authors measure processing time of each CTU and then determine the best possible distribution, first vertically, then horizontally. There are several known problems that affect the predictability of execution time for individual tile data computations and thus compromise strict processing time boundaries defined in Just-in-Time codec applications. Distribution of threads to processing cores by the OS as well as the movement of data to be processed is widely known to be sources of uncertainty regarding the kernel execution start and end time. In addition to that, in heterogeneous, accelerator-based architectures, the time to process tile data is also related to the type of the heterogeneous processing core. All the above can lead to the suboptimal distribution of the tile workload between processing cores. Use of the historical tile processing time as the only parameter for future workload distribution can thus lead to the suboptimal solution.

Tile adaptation in the decoding process is investigated in [9]. The proposed algorithm uses CTU complexity matrix that is composed of a number of bits in each CTU. This matrix is then used in combination with the decoding time of the tile to distribute workloads by adjusting the tile boundaries. This algorithm is designed mostly for the decoding process, which is much less computationally demanding than encoding.

Algorithm

Uniform tile distribution sets tile boundaries uniformly across the frame so that every tile contains approximately the same number of CTUs. However, a uniform distribution can lead to inefficient workload balance that results in performance degradation. Ideally, the complexity would be measured by the number of operations necessary to encode the individual tile, but the overhead of exact calculation would suppress all performance benefits gained by workload balancing.

The algorithm proposed in this paper approximates the processing complexity of a tile based on the number of operations needed to encode the tile in the previous frame. This approximation will be referred to as the tile workload cost.

The initial step was to profile and determine kernels that have the most impact on the frame processing time. Three key kernels were identified: prediction, transformation, and entropy coding. Other parts of the algorithm are either negligible in terms of processing time or related to one of the three mentioned kernels. Total workload cost for selected tile was therefore defined as follows (1).

$$TW_C = \psi P_C + \tau T_C + \varepsilon E_C \tag{1}$$

where P_C, T_C and E_C are prediction cost, transformation cost, and entropy coding cost respectively. Coefficients ψ, τ and ε represent the weight of each individual cost. Next step was to determine and define parameters used to approximate each of the defined costs. Prediction process for each CTU containing CUs (Coding Units) [10] consists of estimating motion to determine a motion vector and forming a residual block in the first phase of the encoding process and of motion compensation and block reconstruction in the second phase [11]. Profiling and analysis showed that cost of these operations mainly depends and can be approximated by two factors: the number of block comparisons in the motion estimation process (i.e. SAD) and size of CU block. Processing complexity of a single comparison is linearly scaled depending on a CU block size. Thereby block comparison cost (BC_C) can be described as

$$BC_C = \frac{BlockSize * BlockSize}{MinBlockSize * MinBlockSize} \tag{2}$$

where *BlockSize* represents the size of CU block and *MinBlockSize* represents minimal CU block size of 4 × 4.

Prediction cost can be now defined as a sum of all block comparison costs (*nBC*) in a tile (3). The number of block comparison costs (*nBC*) is the number of block comparisons performed for the specific tile region in the previous frame.

$$P_C = \sum_{1}^{nBC} BC_C \tag{3}$$

The transformation [12] of residual after the prediction, along with inverse transformation is another complex kernel in HEVC algorithm. Similar as for prediction cost, transform cost depends on the block size and number of transform operations. Standard HEVC DCT transformation is implemented using matrix multiplication so each element of block matrix that has *BlockSize*BlockSize* elements has exactly *BlockSize* multiplications and *(BlockSize-1)* additions. Therefore, if we scale factors so that the minimum block of 4 × 4 has transform block cost of 1, transform block cost can be calculated as in (4). Overall transform cost is the sum of all transform blocks (*nTB*) and is shown in (5).

$$TB_C = \frac{BlockSize * BlockSize}{MinBlockSize * MinBlockSize} * \frac{BlockSize}{MinBlockSize} \tag{4}$$

$$T_C = \sum_{1}^{nTB} TB_C \tag{5}$$

Notice that both prediction and transformation cost depend on the search algorithm and RDO mode decision. The number of block comparisons (*nBC*) for prediction cost and the number of transform block operations (*nTB*) for transform cost will increase with the complexity of RDO.

HEVC uses CABAC (Context-Based Adaptive Binary Arithmetic Coding) [13] to encode syntax values and output final bitstream. The higher the number of bits needed to encode the tile, the more computationally expensive this phase of the encoding is. The cost of entropy coding can thus be approximated with a number of bits used to encode one tile (6).

$$E_C = nBits \tag{6}$$

Coefficients ψ, τ and ε in (1) are one of the key elements of our approach and are calculated a priori, depending on the environment on which video encoder will be deployed. They determine which of the three costs is given more weight in the determination of overall tile workload cost.

Calculating tile workload cost for all tiles is done to determine which tile is more computationally expensive so that tile boundaries can be adjusted to distribute the workload more efficiently. Adjustment of tiles is done as follows:

(1) Tiles in the first frame are divided uniformly
(2) The workload cost is calculated for each tile in the previous frame
(3) The workload cost of each tile row is calculated as the sum of all tiles in a row
(4) Workload cost of each CTU row is calculated as an average from tile row workload cost. Consequently, all CTU rows in same tile row have identical workload cost

$$TW_{C(tileRow)} = \sum_{i=0}^{numOfTilesInRow} TW_{C(tileRow,i)} \tag{7}$$

(5) Tile boundary is moved vertically by one CTU row
(6) Tile row costs are recalculated by adding/removing CTU row cost from tile rows
(7) Steps (4) and (5) are repeated until the difference between workload cost of upper tile row and bottom tile row is minimal
(8) The process is done for each horizontal tile boundary by repeating steps (4)-(6)
(9) After adjustment of all horizontal boundaries, the analog process is done for vertical boundaries
(10) Interval of tile boundaries adjustment can be defined in the configuration

Experimental results

In this work, just-in-time HEVC encoder developed as a part of previous work and research was used. Ten videos were used in the experiment, with different spatial resolutions (shown in Table 1). Higher resolution videos were used in the experiment because exploiting parallel architectures and efficient workload balancing has a higher impact on the performance and efficiency of the encoder.

All tests were conducted on a system with two Intel Xeon E5-2630 v3 processors, each with 8 cores and 16 threads. In-depth bitstream analysis and verification of compliance to the standard was done using Elecard StreamEye tool [14]. Two experiments were made. The goal of the first experiment was to validate the efficiency of the algorithm presented in the section above by calculating the workload costs of tiles and distributing the workload evenly between processing cores. The second experiment was designed to validate and measure the performance of the algorithm in actual real-life conditions, where a large number of videos are being encoded concurrently on the same system. Video sequences were divided into 4 tiles, 2 horizontal and 2 vertical (2 × 2), and each of the tiles was processed in parallel on a separate thread. The configuration of the encoder was set for fast encoding, aiming for the Just-In-Time encoding, and is shown in Table 2. Different quantization parameters (22, 27, 32, and 37) were used, as defined in Common Test Conditions [15] to test the performance of the algorithm.

Determining workload cost requires coefficients ψ, τandεthat depend on processing core types on which specific kernel is executed. In this scenario, CPU cores of the same type were used but were differentiated based on the inclusion of AVX extensions. Both sets of cores

Table 1. Test video sequences.

Video	Resolution	Number of frames (frame rate)
BasketballDrive	1920 × 1080	500 (50 fps)
BQTerrace	1920 × 1080	600 (60 fps)
BlueSky	1920 × 1080	217 (25 fps)
PedestrianArea	1920 × 1080	375 (25 fps)
RiverBed	1920 × 1080	250 (25 fps)
RushHour	1920 × 1080	500 (25 fps)
Traffic	2560 × 1600	150 (30 fps)
DuckTakeOff	3840 × 2160	500 (50 fps)
Beauty	3840 × 2160	600 (120 fps)
Bosphorus	3840 × 2160	600 (120 fps)

Table 2. HEVC encoder configuration.

Coding option	Parameter
QP	22, 27, 32, 37
Search algorithm	Three step search
Search range	12
GOP	IPPPPPPPPPPP ...
SAO and deblocking filter	Disabled
Max CU	32 × 32

Table 3. Benchmarked performance of kernels per core.

Core	P_c/t	T_c/t	E_c/t
CPU	769.965	5.582	10.597
CPU + AVX	854.568	39.316	10.597

Table 4. Kernel coefficients per core.

Core	ψ	τ	ε
CPU	0.001299	0.179147	0.094366
CPU + AVX	0.00117	0.025435	0.094366

were benchmarked, and the results are shown in the following table:

Table 3 shows that this CPU core can process 769.965 prediction costs, 5.582 transformation costs or 10.597 entropy costs in a unit of time. As can be seen from the table, AVX extension has a big impact on the performance of the transformation kernel while it has no effect on entropy kernel due to its sequential nature. Based on acquired data, we can now extract the required coefficients as the ratio between kernel performance:

$$\psi = \frac{1}{P_C/t} \tag{8}$$

$$\tau = \frac{1}{T_C/t} \tag{9}$$

$$\varepsilon = \frac{1}{E_C/t} \tag{10}$$

Coefficients for each core are shown in the following table:

After determining coefficients for the algorithm, defined a set of video sequences were encoded on CPU + AVX extension, with all the combinations of QP for two cases: using uniform tiles and using a proposed algorithm with associated coefficients (Table 4).

$$TW_C = 0.00117 * P_C + 0.025435 * T_C + 0.094366 * E_C \tag{11}$$

Processing times of encoding each video sequence were compared and results are shown in Table 5.

Results show that the algorithm outperforms uniform tiles on average by 2.5–5.5%, depending on QP. The impact on the quality of the video and bitrate was also measured. Deviation of PSNR is less than 0.01% on each video sequence, while bitrate loss is approximately 0.04% on average.

As can be seen in Table 5, gains of using the algorithm vary significantly, depending on the content of the video sequence. In some video sequences, performance gains were up to 12.05% while in the worst case, the algorithm was slower by 1.25%. Some videos have evenly distributed motion throughout the frame, so uniform tiles already have a balanced workload. Adjusting tile boundaries in these cases will not significantly improve or reduce performance (e.g. *Riverbed*).

Table 5. Comparison of using our algorithm vs uniform tiles – numbers represent the speedup of the proposed algorithm compared to uniform tiles.

	Quantization parameter			
Video	22	27	32	37
BasketballDrive	12.05%	11.64%	9.25%	7.97%
BQTerrace	5.68%	9.45%	8.56%	4.18%
Traffic	−0.35%	1.8%	−0.94%	−1.25%
BlueSky	9.35%	9.55%	8.29%	3.53%
PedestrianArea	4.17%	1.57%	1.29%	0.59%
Riverbed	−2.39%	−0.64%	2.10%	0.80%
RushHour	8.47%	1.75%	3.49	3.65%
DuckTakeOff	4.67%	7.15%	7.77	5.67%
Beauty	3.64%	10.91%	0.37	0.00%
Bosphorus	5.24	1.58%	−0.09%	0.31%
Average	5.05%	5.48%	4.01%	2.55%

Table 6. Comparison of using our algorithm vs performance time algorithm – numbers represent the average improvements on encoding times in comparison with uniform tiles.

	Quantization parameter			
Algorithm	22	27	32	37
Proposed algorithm	5.05%	5.48%	4.01%	2.55%
TTLB	5.94%	5.89%	4.55%	4.14%
Difference	0.89%	0.41%	0.54%	1.59%

In fast motion video sequences where motion is localized to a specific region in the frame (e.g. *BasketballDrive*), which is usually the most performance-demanding coding scenario, the algorithm can increase performance by up to 12%.

The results were also compared with Time-based Tile Load Balancing Algorithm (TTLB) [7] and are shown in Table 6.

In the described environment, with same sets of video sequences, and same configuration, TTLB achieves up to 1.59% better performance on average than proposed algorithm that uses tile workload cost, depending on QP. TTLB algorithm relies entirely on performance time measurement for workload balance, so an idle system with enough processing cores of the same type presents an ideal environment for deploying this algorithm. However, in real-life scenarios in which video content service provider deals with a vast number of parallel jobs, it is almost impossible to expect an idle system. So, rather than running the encoding process on an idle system, we tried to simulate the environment where multiple videos are encoded at the same time. To achieve this, we ran multiple tasks at the same time, with each task encoding all 10 test video sequences, with 4 different QP's, measuring the time needed to finish all jobs, with TTLB and with the proposed algorithm. As can be seen in Figure 1, the proposed algorithm achieves better performance as the number of jobs running concurrently on a system increases. In the heavily-utilized system, the processing time of the tile does not accurately represent tile complexity, considering that a lot of time is spent on other

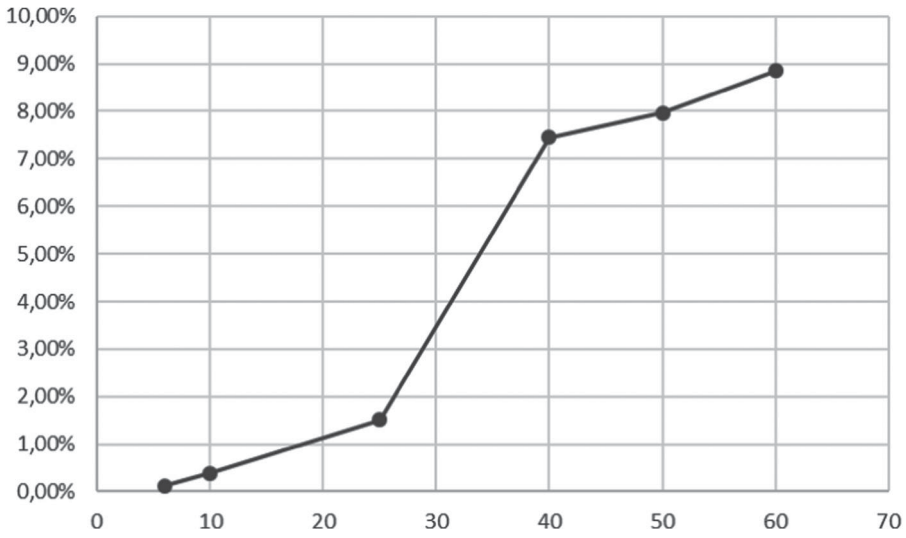

Figure 1. Improvement in performance (%) when using proposed algorithm vs TTLB, depending on the number of jobs running concurrently.

processes (e.g. resource contention, memory accesses, etc.), so tiles rearrangement in case of TTLB algorithm results in unbalanced workloads. Contrarily, the proposed algorithm avoids this problem, and calculates tile complexity without relying on the utilization of the system, achieving up to 8.85% better results on the heavily-loaded system.

Conclusion

In this paper, we presented a novel algorithm that approximates tile's computational complexity to achieve better workload balance by dynamically adapting tile boundaries. Special focus was set on its usability on all platforms, from homogeneous to heterogeneous accelerator-based systems. Proposed algorithm achieves 2.5–5.5% speedup on average, depending on QP when compared to uniform tiles, while keeping the same quality and bitrate. Proposed algorithm provides similar performance as other state-of-the-art algorithms for tile balancing in an ideal environment consisting of idle and homogeneous systems. However, in a real-life scenario where video processing systems are heavily-utilized by a vast number of concurrent jobs, the proposed algorithm achieves up to 8.85% improvement in processing time.

Our future work will include using this algorithm on HEVC transcoder, where statistics from previously decoded frames can be used to balance the workload of the encoder. The algorithm will also be deployed on heterogeneous accelerator-based architectures to investigate possible speedup and workload balancing between different types of processing cores. Use of intelligent resource manager and scheduler in heterogeneous systems would also probably boost the benefits of using the proposed algorithm and will also be the topic of our future research in this domain.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This project has received funding from the European Union's H2020 Future and Emerging Technologies under [grant agreement No 671668].

ORCID

Leon Dragić http://orcid.org/0000-0002-4558-7269
Igor Piljić http://orcid.org/0000-0003-2345-0322
Mario Kovač http://orcid.org/0000-0002-8365-7002

References

[1] Cisco. Cisco visual networking index: forecast and methodology, 2017–2022. 2018 Nov 26. [Online]. Available from: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html
[2] Sullivan GJ, Ohm JR, Han WJ, et al. Overview of the high efficiency video coding (HEVC) standard. IEEE Trans Circuits Syst Video Technol. 2012;22(12):1649–1668.
[3] Bossen F, Bross B, Sühring K, et al. HEVC complexity and implementation analysis. IEEE Trans Circuits Syst Video Techn. 2012;22(12):1685–1696.
[4] Misra K, Segall A, Horowitz M, et al. An overview of tiles in HEVC. IEEE J Sel Top Signal Process. 2013;7(6):969–977.
[5] Blumenberg C, Palomino D, Bampi S, et al. Adaptive content-based tile partitioning algorithm for the HEVC standard. 2013 Picture Coding Symposium, PCS 2013 – Proceedings; 2013, p. 185–188.
[6] Shafique M, Khan MUK, Henkel J. Power efficient and workload balanced tiling for parallelized high efficiency video coding. 2014 IEEE International Conference on Image Processing (ICIP); 2014 Oct 27–30; Paris, France. p. 1253–1257.
[7] Koziri MG, Papadopoulos P, Tziritas N, et al. Adaptive tile parallelization for fast video encoding in HEVC. Proceeding 12th IEEE International Conference on

Green Computing and Communications (GreenCom 2016), IEEE; Chengdu (China); 2016 Dec.
[8] Storch I, Palomino D, Zatt B, et al. Speedup-aware history-based tiling algorithm for the HEVC standard. 2016 IEEE International Conference on Image Processing (ICIP); 2016 Sep 25–28; Phoenix, AZ, USA. p. 824–828.
[9] Baik H, Song H. A complexity-based adaptive tile partitioning algorithm for HEVC decoder parallelization. 2015 IEEE International Conference on Image Processing (ICIP); Sept 2015. p. 4298–4302.
[10] Kim I, Min J, Lee T, et al. Block partitioning structure in the HEVC standard. IEEE Trans Circuits Syst Video Technol. 2012;22(12):1697–1706.
[11] Ugur K, Alshin A, Alshina E, et al. Motion compensated prediction and interpolation filter design in H.265/HEVC. IEEE J Sel Top Signal Process. 2013;7(6):946–956.
[12] Budagavi M, Fuldseth A, Bjøntegaard G, et al. Core transform design in the high efficiency video coding (HEVC) standard. IEEE Trans Circuits Syst Video Technol. 2013;7(6):1029–1041.
[13] Auyeung C, Xu J, Korodi G, et al. A combined proposal from JCTVC-G366, JCTVC-G657, and JCTVC-G768 on context reduction of significance map coding with CABAC, Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G1015, Geneva, Nov 2011.
[14] Elecard: StreamEye software. Available from: https://www.elecard.com/products/video-analysis/streameye
[15] Bossen F. Common test conditions and software reference configurations. Tech. Rep. JCTVC-H1100, 2012.