Heuristics for Tile Parallelism in HEVC

Maria Koziri[1], Panos K. Papadopoulos[2], Nikos Tziritas[3], Nikos Giachoudis[2], Thanasis Loukopoulos[2], Samee U. Khan[4,5], Georgios I. Stamoulis[6]

[1]Computer Science, University of Thessaly, Lamia, Greece, mkoziri@uth.gr
[2]Computer Science and Biomedical Informatics, University of Thessaly, Lamia, Greece, {ppapadopoulos, ngiachou, luke}@dib.uth.gr
[3]Research Center for Cloud Computing, Chinese Academy of Sciences, Shenzhen Institutes of Advanced Technology, Shenzhen, China, {nikolaos, cz.xu}@siat.ac.cn
[4]Electrical and Computer Engineering, North Dakota State University, Fargo, USA, samee.khan@ndsu.edu
[5]National Science Foundation Arlington, USA, skhan@nsf.gov
[6]Electrical and Computer Engineering, University of Thessaly, Volos, Greece, georges@uth.gr

***Abstract*— HEVC has emerged as the new video coding standard promising increased compression ratios compared to its predecessors. This performance improvement comes at a high computational cost. For this reason, HEVC offers three coarse grained parallelization potentials namely, wave front, slices and tiles. In this paper we focus on tile parallelism which is a relatively new concept with its effects not yet fully explored. Particularly, we investigate the problem of partitioning a frame into tiles so that in a resulting one on one tile-CPU core assignment the cores are load balanced, thus, maximum speedup can be achieved. We propose various heuristics for the problem with a focus on low delay coding and evaluate them against state of the art approaches. Results demonstrate that particular heuristic combinations clearly outperform their counterparts in the literature.**

Keywords— video coding; tiles; parallelism; HEVC; partitioning

I. INTRODUCTION

Tiles were introduced in HEVC [1] as an alternative way to achieve coarse grained parallelism that didn't previously exist in H.264/AVC [2]. Under tile partitioning a frame is split into M horizontal zones, each comprised of consecutive full rows of CTUs (Coding Tree Units) and N vertical zones each comprised of a number of consecutive full CTU columns. The intersections of the M horizontal zones with the N vertical ones form the $M \times N$ tile partitioning of the frame. An example is given in Fig. 1.

Dependencies are broken on tile boundaries, allowing each tile to be encoded separately from the rest and creating the potential for parallelization. Since all tiles of a frame must be encoded before proceeding with the following frame, it is straightforward that the frame encoding time will be dictated by the slowest tile encoding task. Of particular interest is the case where enough CPU cores exist to perform a one on one tile-core assignment. In this case balancing core load is equivalent to balancing the size (in computational complexity terms) of the tiles. This process can be split into two parts. The first one is to estimate the CTU computational cost based on the information available, while the second one is to define the required $M \times N$ tile partitioning so that the maximum tile cost, defined by aggregating the costs of the corresponding CTUs is minimized.

In this paper we propose heuristic partitioning scheme and evaluate its combinations with various CTU cost estimators as well as other approaches for tile partitioning, existing in the literature. Our contributions include:

- we propose a heuristic (Iterative Optimal 1D Partitioning, IOP for short), which invokes a 1D optimal algorithm initially proposed for the array partitioning problem [3] in the theory field and adapt it so as to tackle the related 2D problem in a practical fast manner for HEVC encoding;
- we evaluate the most promising heuristic combinations for a number of commonly used test sequences and compare them against the state of the art solutions.

Results indicate that in Low Delay coding, the IOP partitioning heuristic coupled with the estimator that uses CTU time together with GOP structure, outperforms its counterparts in the related literature, with time improvement approaching on average +0.9 speedup with 12 cores. The rest of the paper is organized as follows. Section II discusses the related work. Section III illustrates the partitioning algorithm

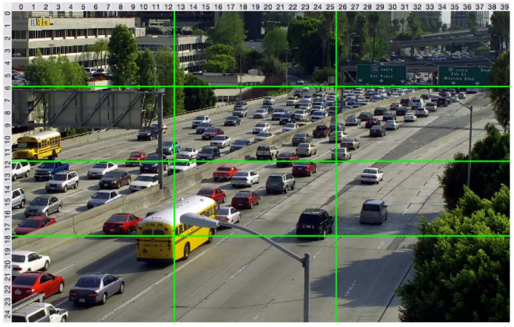


Fig. 1. Example of a 4×3 tile partitioning in the Traffic sequence.

ISBN 978-0-9928626-7-1 © EURASIP 2017

and the cost estimation heuristics. Performance evaluation is done in Section IV. Finally, Section V concludes the paper.

II. Related Work

Video coding parallelization has attracted much interest in the past. Works in the area include both fine and coarse grained approaches.

Fine grained methods usually rely on applying SIMD (Single Instruction Multiple Data) optimizations at one or more stages of the codec. Parallelizing DCT with SIMD instructions was one of the targets of [4]. In [5] a combined CPU-GPUs approach for parallel motion estimation is presented, while [6] includes a comparative study between motion estimation parallelism using CUDA cores, MPI and OpenMP. Such works are rather orthogonal to ours, since they tackle parallelism at a level lower than tiles, thus, can be (in principle) incorporated into tile parallelism approaches.

The coarse-grained category includes parallelism at the wavefront level, at slice and tile levels. In wavefront [7] each thread is assigned a CTU row and can commence the encoding of the corresponding CTUs once the first two CTUs of the previous CTU row are encoded. Slice parallelism also existed in H.264/AVC and has attracted significant interest. In [8] Macroblock assignment to slices was done based on the weighted past average of Macroblock coding times. In [9] the problem of balancing slices was tackled by introducing more slices than the number of available cores. In HEVC, the authors in [10] evaluated slice parallelism using fixed slices under various encoding scenarios, while in [4] the CTU cost estimation used for adapting slice size, was based on weighting upon the depth of each CU comprising the CTU. Last, in [11] the GOP structure for LD setting was used to estimate CTU cost. Although these works are not directly applicable for tile partitioning, we evaluate the IOP heuristic proposed in this paper using as CTU cost estimators the ones proposed in [4], [8] and [11].

More closely related are the works on tile parallelism in HEVC. In [12] the merits of tiles in HEVC are discussed focusing on evaluation using uniform static tile partitions. Another work presenting results for tile parallelization but for the case of intra encoding is [13]. There too, only fixed uniformly sized tiles are considered. The motivation for the tile partitioning algorithm in [14] is to use more tiles compared to the available cores in order to facilitate load balancing. The method is based on deriving a static tile partition based (among others) on pixel variance and the required throughput. Tiles are then assigned to cores using a bin packing technique. Since it is well documented, e.g., [15], that increasing the number of tiles has a negative quality effect on compression (albeit smaller than slices), we followed an alternative path whereby there was one on one correspondence between tiles and CPU cores. In [15] an adaptive content tile partitioning algorithm is proposed. The size of tiles is decided so as to reduce the losses in coding efficiency generated by the use of tiles. Instead, we focus on improving the encoding time by reducing tile load imbalances, thus, this work is orthogonal to ours.

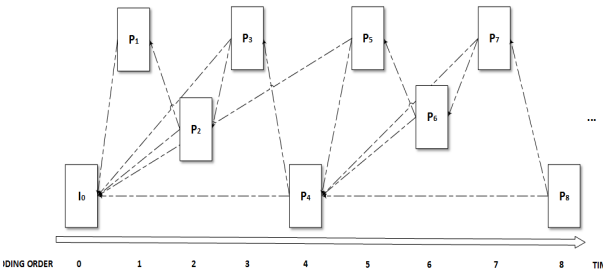


Fig. 2. GOP structure.

TABLE I. Weight Matrix

CU Size	Skip	Inter	Intra
64×64	109	760	52
32×32	42	280	16
16×16	9	71	3
8×8	2	19	1

Perhaps, the closest work is [16] where the authors advocate the use of a similar to [4] CTU weighting scheme for tile partitioning. In the paper we compare the performance of this estimator, both with the partitioning algorithm used in [4] and with the newly proposed one here. Results demonstrate that the estimator we propose that is based on CTU coding time and GOP structure is superior.

III. Heuristics

In this paper we consider the problem of defining an $M \times N$ tile partitioning such that the maximum tile cost is minimized. Both M and N are assumed to be known and remain unchanged throughout the encoding process. As already motivated the problem consists of two sub-problems. The first one is to estimate the CTU costs while the second is to define the partitioning given the estimated costs. In the sequel we present the heuristics evaluated in the paper.

A. CTU Cost Minimization

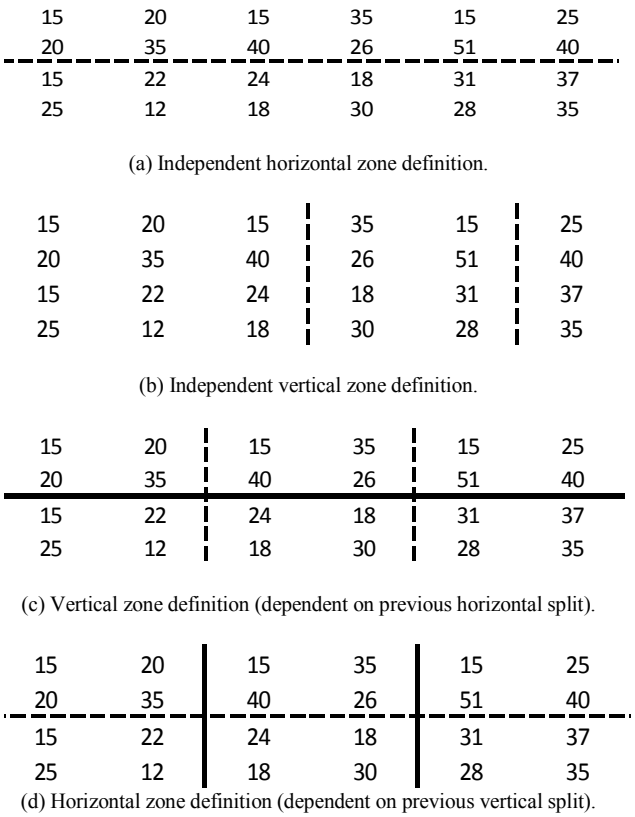
Before proceeding with the presentation of the heuristics, we first introduce some notation. We denote by t_{ij} the encoding time of the j^{th} CTU in the i^{th} frame and w_{ij} the cost that an estimator predicted before the encoding of the i^{th} frame.

Previous frame heuristic (PF). PF estimates w_{ij} as being equal to $t_{(i-1)j}$, i.e., it uses the coding time the CTU exhibited in the last coded frame to estimate its cost in the following frame. The idea is similar to the one presented for slice parallelism in [8].

Weighted CU depth (Weight). The algorithm proposed in [4] and [16] is based on assigning a weight cost on every CU depending on whether in the previous frame it was encoded as Skip, Inter or Intra and its corresponding depth in the quadtree. Table I reproduces the weight matrix for convenience. The heuristic then calculates CTU weight as the summation of the CU weights it consists of.

Low Delay estimator (LDE). One of the common test conditions is LD (Low Delay) which uses a hierarchical GOP structure. In all the experiments of the paper we used the

ISBN 978-0-9928626-7-1 © EURASIP 2017



(a) Independent horizontal zone definition.

(b) Independent vertical zone definition.

(c) Vertical zone definition (dependent on previous horizontal split).

(d) Horizontal zone definition (dependent on previous vertical split).

Fig. 3. An example run of the IOP algorithm.

default configuration for hierarchical P frames in the reference software HM 16.7 which is also depicted in Fig. 2. The intuition behind LDE is that the time complexity of frames belonging to the base layer such as P4 and P8 in Fig. 2 will be better predicted by the preceding frame of the base layer rather than the previous frame number wise. In the example, this means that P8 will be estimated using P4 rather than P7. Another change of LDE versus PF, concerns the estimation of the frame that immediately follows a base layer frame. Instead of using the base layer frame, it uses the frame immediately preceding it. For instance the estimation of P9 (not shown in Fig. 2) will be done using the CTU times exhibited in P7 instead of P8 as PF would have done.

B. Tile Partitioning

Having defined the estimated costs of the CTUs in the frame that is to be encoded, the tile partitioning algorithm attempts to define an optimal $M \times N$ tile partitioning such that the maximum tile cost is minimized. It is worth noting that the problem has been tackled in the theory area, under the context of array partitioning [3], which was shown to be NP-hard. Nevertheless, the adoption in video coding of heuristics from the theory area was not performed yet (to the best of our knowledge). This is presumably due to the fact that the approximation schemes proposed are rather complex in nature, thus, might compromise running time. Here, we follow an alternative approach whereby we adopt a fast 1D optimal scheme to run for the 2D tile case.

The 1D optimal algorithm. Assume we have an array of size S which must be split into M bins so that the maximum cost of a bin, defined as the aggregate cost of its cells, is minimized. This problem can be solved to optimality by the algorithm that follows.

Given a specific bin size B, it is identified whether all array elements can fit into M bins in a consecutive manner, or not. For instance, assuming the following array A={10,12,15,5,8}, M=3 and B=20, it is clear that bin1={10}, bin2={12}, bin3={15,5}, leaving the last array element unassigned (assigning it to bin1 and bin2 is illegal, while in bin3 capacity violation occurs). We will refer to such bin assignments that leave some array elements unassigned as infeasible.

Let C be the total cost of all elements in the array. Clearly, the optimal maximum bin cost can't be lower than: $B_{min} = \lfloor C/M \rfloor$. It also can't be larger than C. It is straightforward to observe that given the range $[B_{\min}, C]$ the optimal bin size is the first integer in this range that produces a feasible bin assignment. More accurately: $B_{min} \leq B_{opt} \leq C$, $\forall B < B_{opt}$ assignments are infeasible and $\forall B \geq B_{opt}$ assignments are feasible. In order to identify B_{opt} binary search can be used. To reduce the number of iterations, instead of searching in the range $[B_{\min}, C]$, the algorithm first checks for feasibility $B=B_{\min}$ (in case C/M isn't an integer the assignment can be identified as infeasible without checking bin assignments). If the assignment is feasible, $B_{\text{opt}}=B_{\min}$ and the algorithm terminates. Otherwise, it checks for the feasibility of $B=2B_{\min}$. If $2B_{\min}$ also results in infeasible placement it checks for $2(2B_{\min})$ and so on so for until a feasible assignment is identified for $B=2^kB_{\min}$. Then a binary search approach is used to identify B_{opt} in the range of $(B_{\min}, 2^kB_{\min}]$. We should notice that in almost all cases at the experiments k=1.

2D application. Assume $M \times N$ is the required tile split. The role of the 2D array elements is taken by the CTU cost estimations. The algorithm proposed in this paper (*Iterative Optimal 1D Partitioning – IOP*) performs the following steps. First, it defines the horizontal split into M zones by running the previously described 1D algorithm, over a 1D array of size equaling frame height in CTUs and with the cost of each element derived by summing up the costs of the CTUs in the corresponding CTU row of the frame. Next, it defines the vertical split into N zones using the 1D algorithm, over an array of size equaling frame width in CTUs and with the cost of each element derived by summing up the costs of the CTUs in the corresponding CTU column of the frame.

The intersection of the M and N zones, form the initial tile partitioning. The process is illustrated in Fig. 3 (a) and (b). For instance, defining the vertical split into N=3 zones, is equivalent to solving the 1D problem using 3 bins over the array A={75, 89, 97, 109, 125, 137}, which gives B_{opt} =261 and the corresponding vertical cut depicted in Fig. 3 (b).

The derived partitioning is further improved in an iterative manner as follows. First, the vertical split is optimized using the horizontal split previously derived (Fig. 3 (c)). To do so, a variation of the 1D approach is used, whereby the participating bins equal the number of tiles (6 in the example).

ISBN 978-0-9928626-7-1 © EURASIP 2017

In calculating feasibility for a particular bin size B, columns are added one by one, up to the point where adding a column would violate the capacity of one of the M bins as defined by the horizontal partition (M=2 in the example). The rest of the algorithm remains the same, i.e., it searches for the first feasible assignment using binary search in the range $[B_{\min}, 2^k B_{\min}]$ with $B_{min} = \lfloor C/MN \rfloor$. So for instance, having defined the horizontal partitioning of the solid line in Fig. 3 (c), the vertical repartitioning is done as follows (assuming T_{00} is the top leftmost tile and W_{00} the total cost of its elements). The total cost of the elements is C=632, $B_{\min}$=105 and since C/MN isn't an integer it leads to an infeasible assignment. Then the algorithm checks for B=210. The first column is added giving W_{00}=35, W_{10}=40 (the cost of other tiles is 0). Since both are less than 210, the second column is checked. Adding it, results to: W_{00}=90 and W_{10}=74. Then the third column gives: W_{00}=145, W_{10}=116 and the fourth column results to: W_{00}=206, W_{10}=164. Adding the fifth column would result in capacity violation, therefore the first two tiles are defined as having a width of 4columns. The last two columns will be assigned to T_{01} and T_{11} and all the elements of the array will be assigned using the 6 available bins/tiles (in fact just 4). Since the assignment is feasible, in a binary search manner B=152 will be checked next and so on so for, until B_{opt}=131 is defined, leading to the partition shown with dashed lines in Fig. 3 (c).

Having defined a new vertical split, the algorithm does the same with the horizontal partitioning as shown in Fig. 3(d). The algorithm then proceeds in an iterative manner whereby at each iteration, both vertical and horizontal split improvements are considered. It terminates at the first iteration that couldn't improve the partitioning.

IV. Experiments

We implemented the tile parallelism heuristics using the HM 16.7 reference software and OpenMP. We conducted experiments on a Linux server with two 6-core Intel Xeon E5-2630 CPUs running at 2.3GHz using hyper threading. We used Class A and B test sequences together with Bosphorus which is 4K. The characteristics of the sequences are summarized in Table II. All results were obtained assuming the LD scenario with an initial I frame followed by P frames and a GOP size of 4 with the structure shown in Fig. 2. QP

TABLE II. Video Sequences

Name	Resolution	Frames per second (fps)	Total frames	CTUs per frame
Bosphorus	3840×2160	120	200/600	2040
PeopleOnStreet	2560×1600	30	150	1000
Traffic	2560×1600	30	150	1000
BasketballDrive	1920×1080	50	500	510
BQTerrace	1920×1080	60	600	510
Cactus	1920×1080	50	500	510
Kimono	1920×1080	24	240	510
ParkScene	1920×1080	24	240	510

TABLE III. Speedups per sequence.

		4	8	12
Bosphorus	Static	3,34	6,11	8,80
	IOP-PF	**3,69**	6,80	9,96
	IOP-LDE	3,68	**6,94**	**10,14**
	IOP-Weight	3,39	6,31	8,94
	[16]	3,30	6,22	9,09
People On Street	Static	3,48	**6,82**	8,64
	IOP-PF	3,64	6,64	**9,65**
	IOP-LDE	**3,67**	6,63	**9,65**
	IOP-Weight	3,57	6,51	9,17
	[16]	3,55	6,53	8,92
Traffic	Static	3,53	6,44	8,83
	IOP-PF	3,41	6,51	9,05
	IOP-LDE	**3,63**	6,65	**9,31**
	IOP-Weight	3,56	**6,66**	9,21
	[16]	3,41	6,54	9,16
Basketball Drive	Static	3,30	5,99	7,93
	IOP-PF	3,44	6,18	8,53
	IOP-LDE	**3,56**	**6,25**	**8,58**
	IOP-Weight	3,19	5,64	7,53
	[16]	3,03	5,08	6,73
BQTerrace	Static	3,48	5,95	8,27
	IOP-PF	3,28	**6,31**	8,54
	IOP-LDE	3,55	6,29	**8,56**
	IOP-Weight	3,53	6,05	8,35
	[16]	**3,58**	6,21	8,51
Cactus	Static	3,06	5,71	7,70
	IOP-PF	3,33	6,39	**8,66**
	IOP-LDE	**3,44**	**6,41**	8,64
	IOP-Weight	3,15	5,70	7,68
	[16]	3,34	5,62	7,42
Kimono	Static	3,34	5,88	8,19
	IOP-PF	3,60	**6,86**	9,44
	IOP-LDE	**3,71**	6,84	**9,48**
	IOP-Weight	3,37	5,89	8,13
	[16]	3,48	6,31	8,46
ParkScene	Static	3,41	5,76	7,98
	IOP-PF	3,41	6,25	8,57
	IOP-LDE	**3,57**	**6,38**	**8,80**
	IOP-Weight	3,45	5,87	7,99
	[16]	3,42	6,10	8,41

ISBN 978-0-9928626-7-1 © EURASIP 2017

TABLE IV. AVERAGE SPEEDUP IN ALL SEQUENCES

Heuristic	4	8	12
Static	3,37	6,08	8,29
IOP-PF	3,47	6,49	9,05
IOP-LDE	**3,60**	**6,55**	**9,15**
IOP-Weight	3,40	6,08	8,38
[16]	3,39	6,08	8,34

was set to 32, bit depth was 8, CTU size 64×64, max depth for partitioning was set to 4 and search mode to TZ.

We experimented with three different tile numbers (in one slice): 4 (2×2), 8 (4×2) and 12 (4×3). Each tile was assigned a separate CPU core on a one on one basis. We compared the combination of IOP together with the PF, Weight and LDE estimators versus a static approach that assigns tile sizes in a uniform manner and doesn't change it (Static) and the algorithm of [16]. In all cases we measured the achievable speedup versus a sequential execution. Table III shows the achievable speedups for each sequence. Bold values represent the algorithmic winner in this particular setting. Notice, that IOP-LDE and IOP-PF are winners over the rest, with the first combination achieving slightly better overall performance compared to the second one, as depicted more clearly in Table IV that illustrates the average performance in speedup terms over all test sequences.

Summarizing the results, the achievable speedup of IOP, make it a clear winner against Static and the method presented in [16], with speedup differences of roughly 0.5 in the 8 core case and 0.9 in the 12 core scenario. Among its variants the combination with LDE achieves the best performance with the PF following closely. Before ending the section we would like to mention that IOP itself contributed a negligible overhead, hence, the improved speedup achieved in the coding process. This was due to the fact that after the initial partitioning, the number of iterations performed by IOP was never more than four (in 36.24% of the cases it was one and in 60.84% two).

V. CONCLUSIONS

In this paper we considered the problem of defining tile partitioning such that the threads tasked with the encoding of each tile are load balanced. We proposed a partitioning heuristic called IOP and evaluated it with different CTU cost estimators. IOP-LDE and IOP-PF were found to outperform both the static uniform approach and another alternative from the literature.

ACKNOWLEDGMENT

Samee U. Khan's work was supported by (while serving at) the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Nikos Tziritas' work was supported by NSFC and PIFI International Scholarship under the grants 61550110250 and 2017VCT0001, respectively.

REFERENCES

[1] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) standard," IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1649-1668, Dec. 2012.

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

[3] A. Mingozzi, S. Ricciardelli, M. Spadoni, "Partitioning a matrix to minimize the maximum cost," Discrete Appl. Math., vol. 62, no. 1-3, pp. 221–248, 1995.

[4] Y.-J. Ahn, T.-J. Hwang, D.-G. Sim, and W.-J. Han, "Implementation of fast HEVC encoder based on SIMD and data-level parallelism," EURASIP J. Image and Video Processing, vol. 16, 2014.

[5] X. Wang, L. Song, M. Chen, and J-J. Yang, "Paralleling variable block size motion estimation of HEVC on multi-core CPU plus GPU platform," ICIP 2013, pp. 1836-1839.

[6] E. Monteiro, B. B. Vizzotto, C. M. Diniz, M. Maule, B. Zatt, and S. Bampi, "Parallelization of full search motion estimation algorithm for parallel and distributed platforms," International Journal of Parallel Programming, vol. 42(2), pp. 239-264, 2014.

[7] C.C. Chi, M.A. Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel scalability and efficiency of HEVC parallelization approaches," in IEEE Trans. Circuits Syst. Video Technol., vol. 22, no. 12, pp. 1827-1838, Dec. 2012.

[8] L. Zhao, J. Xu, Y. Zhou, and M. Ai, "A dynamic slice control scheme for slice-parallel video encoding," ICIP 2012, pp. 713-716.

[9] J.-F. Franche, and S. Coulombe, "A multi-frame and multi-slice H.264 parallel video encoding approach with simultaneous encoding of prediction frames," CECNet 2012, pp. 3034-3038.

[10] P. Piñol, H. M. Gomis, O. M. L. Granado, and M. P. Malumbres, "Slice-based parallel approach for HEVC encoder," Journal of Supercomputing, vol. 71(5), pp. 1882-1892, 2015.

[11] M.G. 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," MMSP 2016, pp. 1-6.

[12] K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, "An overview of tiles in HEVC," IEEE Journal of Selected Topics in Signal Processing, vol. 7, no. 6, pp. 969-977, Dec. 2013.

[13] A. Koivula, M. Viitanen, J. Vanne, T. D. Hämäläinen, and L. Fasnacht, "Parallelization of Kvazaar HEVC intra encoder for multi-core processors," SiPS 2015, pp. 1-6.

[14] M. Shafique, M. U.K. Khan, and J. Henkel, "Power efficient and workload balanced tiling for parallelized high efficiency video coding," ICIP 2014, pp. 1253-1257.

[15] C. Blumenberg, D. Palomino, S. Bampi, and B. Zatt, "Adaptive content-based tile partitioning algorithm for the HEVC standard," PCS 2013, pp. 185-188.

[16] Y.-J. Ahn, T.-J. Hwang, D.-G. Sim, and W.-J. Han, "Complexity model based load-balancing algorithm for parallel tools of HEVC," VCIP 2013.