Adaptive Tile Parallelization for Fast Video Encoding in HEVC

Maria Koziri[1], Panos K. Papadopoulos[2], Nikos Tziritas[3*], Antonios N. Dadaliaris[1], Thanasis Loukopoulos[2], Samee U. Khan[4], Cheng-Zhong Xu[3,5]

[1]Computer Science, University of Thessaly, Lamia, Greece, {mkoziri, dadaliaris}@uth.gr
[2]Computer Science and Biomedical Informatics, University of Thessaly, Lamia, Greece, {ppapadopoulos, 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]Electrical and Computer Engineering, Wayne State University, Detroit, USA, czxu@wayne.edu

Abstract— **As large multimedia providers rely more and more on Cloud resources to perform video coding and transcoding, designing fast and efficient coders that take advantage of parallelization, particularly for the new video standard HEVC, is of paramount importance. Tiles were introduced in HEVC to provide an alternative parallelization granularity compared to the previous standard H.264/AVC. The premise was that with the rectangular tile shapes, spatial correlation between samples could be better exploited compared to slices, leading to increased coding efficiency. While a significant amount of research was done in parallelizing video coding, few works exist on tile parallelization in HEVC. In this paper we tackle the problem of balancing the CPU core load by dynamically adapting tile sizes. It is shown that the proposed adaptive method leads to significant reduction of load imbalances and consequently, achieves a better speedup compared to static, uniform CTU-tile assignment. Furthermore, these gains come at no cost quality wise.**

Keywords— video coding; tile parallelization; adaptive tile sizing; HEVC; load balancing

I. Introduction

Nowadays, large multimedia content providers and social media networks struggle at keeping pace with the popularity explosion of smart devices [9] and the resource demands it entails in order to perform filtering [2], processing [25], storage and delivery [24]. As an example, Cisco reported in [7] that the mobile network traffic increased by 75% in 2015, the majority of which (54%) was video. Even if the uploaded user videos are already compressed in some format, there is an ardent need for transcoding or scalable video coding (SVC) [14] the original sequence to different resolutions and bitrates in order to support streaming at devices of different characteristics residing in networks of various loss rates. Furthermore, transcoding might also involve changing the compression standard e.g., from H.264/AVC [29] to HEVC [23], the new video coding standard, and in its basic form it entails decoding the original sequence and re-encoding it again.

Due to the massive number of videos streamed every day, media providers rely more and more to Cloud resources for video coding purposes. But video coding is a computationally expensive task on its own, particularly as resolution becomes higher. Consider for instance that when an encoder nominally achieves real time performance in some configuration, it means that in order to encode a movie in this configuration the amount of time might equal (but not exceed) movie length. Thus, it is apparent that the computational burden placed in related Cloud services is tremendous and speeding up the encoding process is of utmost importance for both scalability and sustainability reasons.

Such speedup can only be achieved through efficient parallelization [5]. The new video standard, HEVC, offers three coarse-grained parallelization opportunities suitable for parallelization on a CPU core level namely: slice-level, tile-level and wavefront parallelization (WPP) [6]. While slices existed in H.264/AVC, the other two methods are new in HEVC and their potential is not fully explored yet.

In this paper we focus on tile-level parallelization at the encoding part of HEVC. Specifically, we investigate the potential of using CTU encoding time (Coding Tree Unit, i.e., the block of pixels where a frame is split into in HEVC; equivalent but not identical to Macroblocks in H.264/AVC) in order to adapt tile size so that CPU cores are load balanced and consequently increased speedup is achieved. Our contributions include the following:

- An algorithm (Time-based Tile Load Balancing *TTLB*) is proposed that adaptively defines tile partitioning based on the coding times of CTUs.
- Evaluation against the *Static* approach that evenly partitions a frame into tiles and keeps the partition fixed, shows significant speedup improvement. Moreover, this improvement comes at no extra cost compared to Static partitioning. These results highlight the merits of our approach.

The rest of the paper is organized as follows: Section II provides a brief overview of the related work. Section III illustrates TTLB which is experimentally evaluated in Section IV. Finally, Section V summarizes the paper and gives the conclusions.

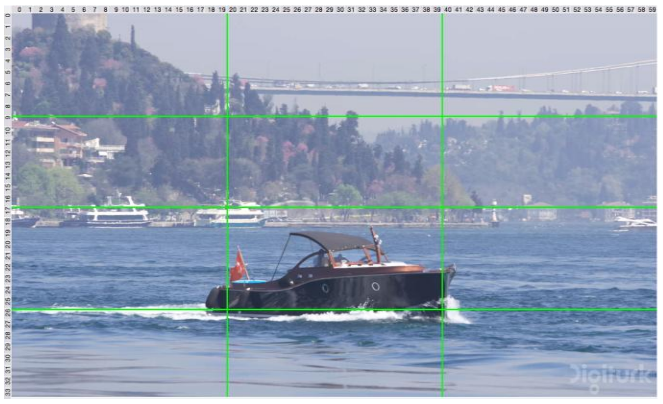
Figure 1(a). Screenshot from Bosphorus (frame 0).

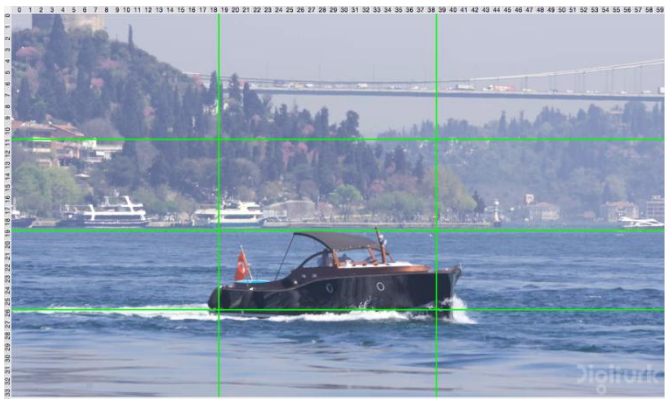
Figure 1(b). Screenshot from Bosphorus (frame 5).

II. Related Work

Research in the area of video coding parallelization can be broadly categorized depending on whether it considers fine or coarse-grained parallelization.

Fine-grained approaches usually comprise of works applying SIMD parallelism. In [17] SIMD operations at the CPU-core level were applied to efficiently implement an AVS decoder. DCT and cost function parallelization for HEVC is discussed (among others) in [1]. Motion estimation, either with the Sum of Absolute Differences (SAD) metric or with other heuristic approaches, e.g., the ones in [13] and [16], has also attracted SIMD parallelization efforts. In [28] a combined GPU – multi core CPU approach for parallel motion estimation is presented, while in [19] a comparative evaluation is provided between GPU implementation of motion estimation with CUDA and equivalent implementations using MPI and OpenMP. The authors concluded that GPUs offer significant advantages. In [26] a framework to analyze the dependencies of neighboring CTUs is introduced. CTUs form a DAG which is then scheduled for parallel computation. Finally, in [27] different parallelization degrees for motion estimation are discussed varying from single CU to groups of CUs.

The aforementioned works are orthogonal to ours since in principle tile parallelization can be combined with SIMD approaches using GPUs.

In the coarse-grained category a significant amount of past work concerned slice parallelization both in H.264/AVC, e.g., [8] and [30] to name a few, and in HEVC, e.g., [1], [15], [21]. In [30] adaptive Macroblock assignment to slices based on weighted past average (WPA) of Macroblock coding times is considered. A similar approach was evaluated in [15] for HEVC. In [8] the problem of balancing slices was tackled by assigning more slices than the existing cores in an effort to reduce parallelization granularity, thus, achieving better balance. Concerning HEVC, the authors in [21] evaluated slice-based parallelism under different encoding scenarios considering fixed slice sizes. Contrary, in [1] slice-level parallelization with adaptive CTU-slice assignment is discussed. The proposed algorithm is based on assigning weights to CTUs depending on the mode and depth of CTU encoding and assigning CTUs to slices so that aggregate weights are balanced.

Although works on slice-level parallelization differ in scope from the paper, some of the ideas discussed there are applicable in the case of tiles as well. Specifically, our proposed algorithm TTLB is inspired by [30] in order to use the coding times of CTUs to estimate tile load. Furthermore, the idea of [8] is also applicable for tiles but only in the cases where video quality is not too important.

More closely related are the works done for tile level parallelism in HEVC such as: [3], [12], [18] and [22]. In [18] the potentials introduced to video coding with the advent of tiles in HEVC are examined. Performance issues using fixed size tiles are discussed. Another work presenting results from tile based parallelization but for the case of intra encoding is [12]. There too, only fixed size tiles were considered.

The motivation for the tile partitioning algorithm in [22] 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 [5], [6] that increasing the number of tiles has a negative quality effect on compression, we followed an alternative path whereby there was one on one correspondence between tiles and CPU cores. As shown in the experiments, the lack of load balancing potential by using fairly large instead of small tiles, is more than compensated through the adaptive tile resizing mechanism that clearly outperforms a comparable Static approach.

Finally, in [3] 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. As such, we view the work in [3] as orthogonal to ours.

III. Load Balancing Algorithm

The Time-based Tile Load Balancing algorithm (TTLB) defines tile sizes using the CTU encoding times of the previous frame. Assume that a frame consists of $X\times Y$ CTUs arranged in X CTU rows and Y CTU columns. Furthermore, let the tile partitioning be into $M\times N$ tiles, with M being the tile rows and N being the tile columns. TTLB first calculates the total time of each CTU row (let R_i) and each CTU column (let C_j), by aggregating the encoding times of CTUs belonging to the

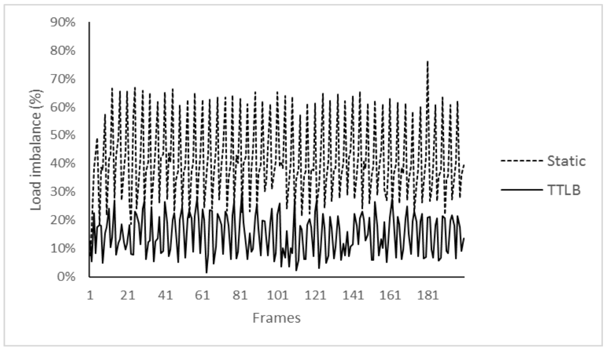

Figure 2(a). Bosphorus, 4 tiles.

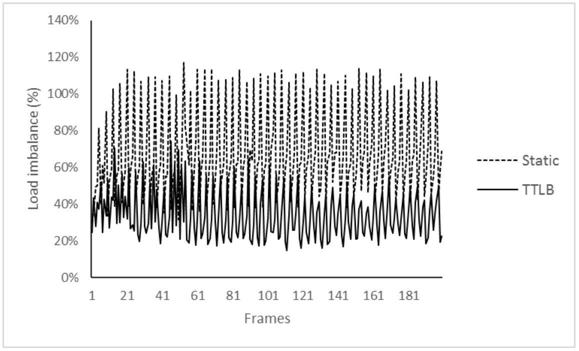

Figure 2(b). Bosphorus, 12 tiles.

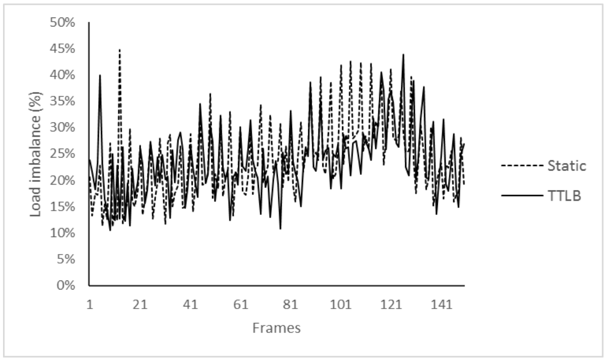

Figure 3(a). Traffic, 4 tiles.

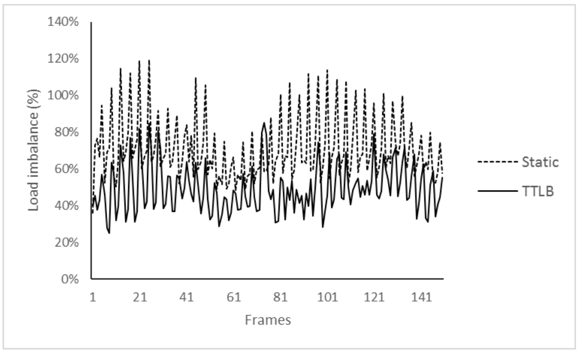

Figure 3(b). Traffic, 12 tiles.

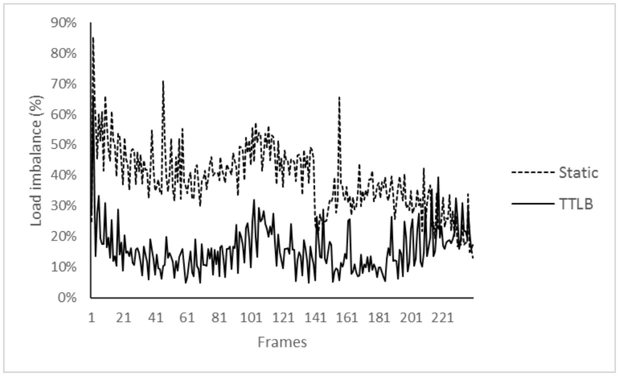

Figure 4(a). Kimono, 4 tiles.

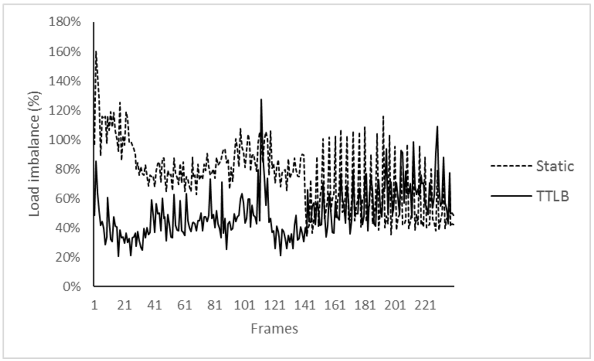

Figure 4(b). Kimono, 12 tiles.

respective row or column (*i*th and *j*th respectively). It then defines the vertical split into *N* tile columns and then the horizontal split into *M* tile rows. Let TC_k be the width in CTUs of the *k*th tile column ($1 \le k \le N$), and TR_l be the height in CTUs of the *l*th tile row, ($1 \le l \le M$). The algorithm assigns tile column widths using the following:

$$W = \sum_{j=1}^{Y} C_j \tag{1}$$

$$TC_k = \sum_{j=s}^{e} C_j \le \left\lfloor \frac{W}{N} \right\rfloor < \sum_{j=s}^{e+1} C_j : \; s = 1 + \sum_{u=1}^{k-1} TC_u \quad , (1 \le k \le N\text{-}1) \tag{2}$$

$$TC_N = Y - \sum_{k=1}^{N-1} TC_k \tag{3}$$

Namely, it calculates the total time of all CTUs in (1), and then attempts to assign at each tile column a width that will lead to equal time cost assignment (if possible) at each tile column as per (2) and (3). Specifically, it starts by defining the width of the first tile column. To do so it adds CTU columns starting from the first one until the total time of CTUs in the assigned columns is the maximum possible that doesn't exceed the required time cost assignment. The algorithm then proceeds by assigning CTU columns to the second tile column starting with the CTU column that follows the last assigned CTU column. The last tile column gets the CTU columns that remain from the previous assignments.

Tile row heights are defined in a similar manner to tile columns using the following:

$$TR_l = \sum_{i=s}^{e} R_i \le \left\lfloor \frac{W}{M} \right\rfloor < \sum_{i=s}^{e+1} R_i : \; s = 1 + \sum_{u=1}^{k-1} TR_u \quad , (1 \le l \le M\text{-}1) \tag{4}$$

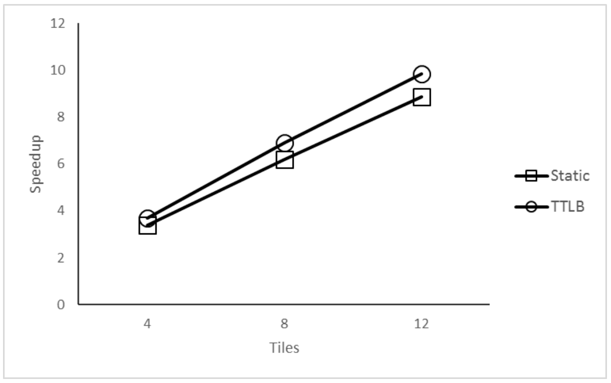

Figure 5(a). Bosphorus, QP=32.

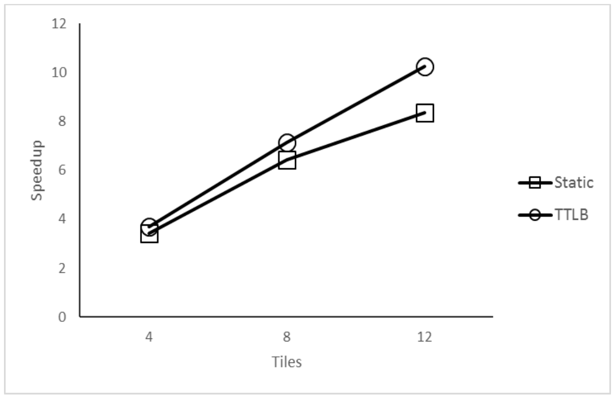

Figure 5(b). Bosphorus, QP=22.

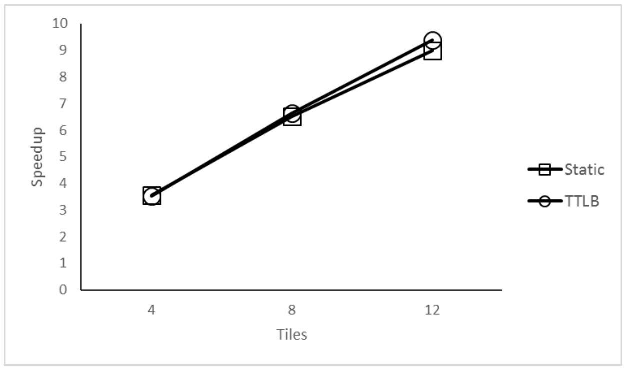

Figure 6(a). Traffic, QP=32.

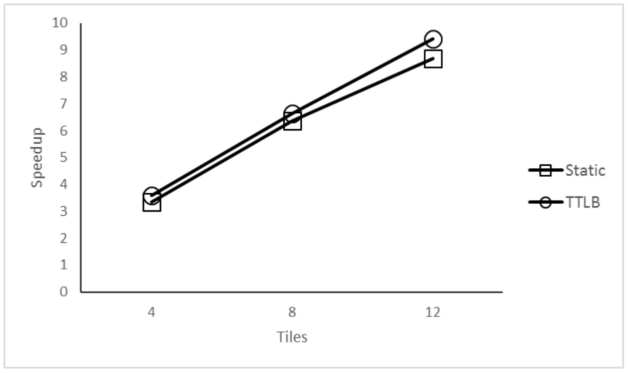

Figure 6(b). Traffic, QP=22.

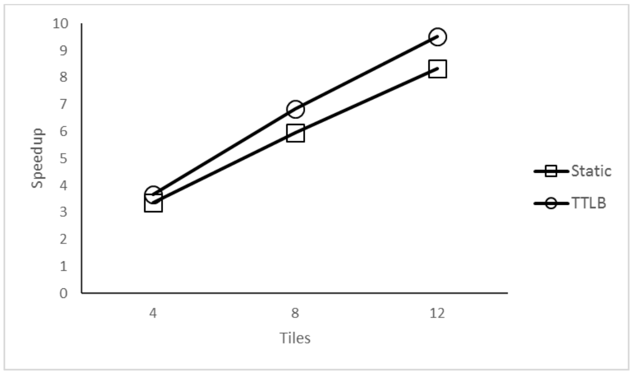

Figure 7(a). Kimono, QP=32.

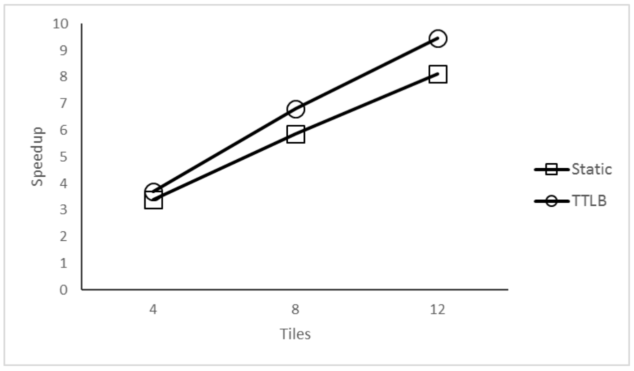

Figure 7(b). Kimono, QP=22.

$$TR_M = X - \sum_{l=1}^{M-1} TR_l \qquad (5)$$

Fig. 1 presents screenshots from the Bosphorus sequence [12] with a partitioning in 12 tiles using TTLB. Notice that compared to the initial cut at frame 0 (Fig.1 (a)), TTLB in frame 5 (Fig. 1(b)) has reduced the size of the tile enclosing the boat where most of the motion takes place.

IV. Experiments

We implemented TTLB using the reference software HM 16.7 [10] and OpenMP [20] for threading. We conducted experiments on a Linux server with two 6-core Intel Xeon E5-2630 CPUs running at 2.3GHz. We used three sequences of different resolution, summarized in Table I.

TABLE I. Video Sequences

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

In order to save time in the experiments we used the first 200 frames of the Bosphorus sequence instead of the complete one. All results were obtained assuming the LD scenario with an initial I frame followed by P frames and a GOP size of 4 [4] which is similar but not identical to hierarchical P coding [11]. Unless otherwise stated, QP 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. In the experiments we compared the performance of TTLB against the static, uniform assignment obtained by using the relevant option in the reference software. We measured the achievable speedup, PSNR and bitrate differences as well as the load imbalance incurred among the execution time of tiles measured as the following percentage:

100(MAX_Tile_Time – MIN_Tile_Time)/MIN_Tile_Time

Figs. 2, 3 and 4 show the load imbalance experienced by both Static and TTLB for two different tile numbers 4 and 12. It can be observed that in all sequences but for Traffic with 4 tiles (Fig. 3(a)), TTLB is able to reduce significantly the load imbalances that occur by Static. This improvement is more evident for 12 tiles, which is expected since more tiles lead to more potential in exploiting spatial locality of video motion. In the Traffic sequence and for 4 tiles the gains over Static are rather limited. This is due to the fact that in this sequence there is motion almost everywhere in the frame. Thus, compared to the other two sequences there exists less potential for improvement. As a further indication for the above, notice that Static in Fig. 3(a) exhibits an imbalance of less than 30% for the biggest part, leaving little room for improvement. Judging from the figures as a whole, we can say that using TTLB drops load imbalance to less than 20% for 4 tiles while it also drastically improves load balance in the case of 12 tiles. Performance for 8 tiles (not shown) was found to fall in the middle between the performance with 4 and 12 tiles.

Next we plot the speedups over a base scenario with no tile parallelization. Results are shown in Figs. 5, 6 and 7 for two different QPs 32 and 22. We can observe that the performance gains in Bosphorus and Kimono are substantial. In all the figures the performance gap over Static increases to the number of tiles, leading in certain cases to a difference in speedup of roughly 2 (Fig. 5(b)). In the Traffic sequence the gains are less impressive and are considerable only for QP=22.

Finally, we measured the impact on quality TTLB has. Table II summarizes performance. Specifically it records: (i) the difference in Y-PSNR between TTLB and Static, and (ii) the difference in bitrate between TTLB and Static measured as the following percentage:

100(bitrate(TTLB) –bitrate(Static))/bitrate(Static)

As a consequence of the above, positive values on Y-PSNR and negative values for bitrate percentage indicate TTLB is better than Static.

Observe that the differences in Y-PSNR are rather negligible (in the order of the third digit). A similar observation holds true for the bitrate which increases by at most 0.48% while there exist cases where it decreases (maximum value of 0.61%). These results are very encouraging towards TTLB indicating that the increased performance over Static comes at virtually no cost quality wise.

Summarizing the results from the experiments we can state that TTLB is able to improve encoding time compared to a parallel encoder implementation that uses Static tiles. The gains are particularly substantial for sequences exhibiting motion at specific frame parts, and less so for sequences exhibiting motion throughout the whole frame (or little motion overall). However, even in such cases some marginal gains can be expected. Furthermore, the performance improvement of TTLB comes at no quality loss compared to Static. Finally, TTLB is rather simple to implement once tile parallelization is implemented, making it a definite candidate for adoption in related HEVC encoders.

TABLE II. QUALITY METRICS

		Tile Number					
		Y-PSNR			**bitrate %**		
	QP	**4**	**8**	**12**	**4**	**8**	**12**
Bosphorus	22	-0.006	0.001	-0.000	0.48%	-0.09%	0.09%
	27	0.002	-0.007	-0.002	-0.24%	-0.09%	0.12%
	32	0.001	0.006	-0.003	-0.61%	-0.43%	-0.24%
	37	-0.010	0.001	-0.009	0.07%	-0.27%	0.02%
Traffic	22	-0.001	0.000	0.001	-0.21%	-0.02%	0.05%
	27	0.006	0.002	-0.002	-0.15%	-0.12%	-0.01%
	32	-0.001	0.010	0.007	0.05%	-0.15%	-0.10%
	37	-0.009	0.013	-0.001	-0.12%	-0.22%	0.10%
Kimono	22	0.002	-0.001	-0.000	-0.08%	0.04%	0.02%
	27	-0.001	-0.003	-0.000	0.08%	0.09%	0.02%
	32	0.002	-0.001	-0.001	0.20%	0.12%	0.12%
	37	0.001	-0.003	0.005	0.17%	0.20%	0.52%

V. CONCLUSIONS

Designing fast video encoders that capitalize on the HEVC parallelization potentials is crucial in order to minimize Cloud resource consumption by large multimedia providers. In this paper we tackled the problem of adaptive tile parallelization in HEVC. We proposed an algorithm, named TTLB that dynamically adjusts tile sizes using CTU encoding time, with the aim of balancing CPU core load. Experiments demonstrate that TTLB achieves substantially better speedup compared to the static, uniform partitioning, without sacrificing quality.

REFERENCES

[1] 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.

[2] N. Assimakis, M. Adam, M. Koziri, S. Voliotis, and K. Asimakis, "Optimal Decentralized Kalman Filter and Lainiotis Filter," Digital Signal Processing, vol. 23(1), pp. 442-452, 2013.

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

[4] F. Bossen, Common Test Conditions and Software Reference Configurations, document JCTVC-H1100, JCT-VC, San Jose, CA, Feb. 2012.

[5] F. Bossen, B. Bross, K. Sühring, and D. Flynn, "HEVC Complexity and Implementation Analysis," IEEE Trans. Circuits Syst. Video Techn. vol. 22(12), pp. 1685-1696, 2012.

[6] 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 Transactions on Circuits and Systems for Video Technology, vol. 22, no. 12, pp. 1827-1838, Dec. 2012.

[7] Cisco Systems Inc. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2015–2020 (White Paper). Available at: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[8] J.-F. Franche, and S. Coulombe, "A multi-frame and multi-slice H.264 parallel video encoding approach with simultaneous encoding of prediction frames," in Proc. of the 2012 Int. Conf. on Consumer Electronics, Communications and Networks (CECNet), pp. 3034-3038, Apr. 2012.

[9] G. Haralabopoulos, I. Anagnostopoulos, and S. Zeadally, "Lifespan and propagation of information in On-line Social Networks: A case study based on Reddit," J. Network and Computer Applications, vol. 56, pp. 88-100, Oct. 2015.

[10] HM 16.7 reference software. http://hevc.hhi.fraunhofer.de

[11] D. Hong, M. Horowitz, A. Eleftheriadis, and T. Wiegand, "H.264 hierarchical P coding in the context of ultra-low delay, low complexity applications," PCS 2010, pp. 146-149.

[12] A. Koivula, M. Viitanen, J. Vanne, T. D. Hämäläinen, and L. Fasnacht, "Parallelization of Kvazaar HEVC intra encoder for multi-core processors," in Proc. IEEE Workshop Signal Process. Syst., Hangzhou, China, Oct. 2015, pp. 1-6.

[13] M.G. Koziri, A.N. Dadaliaris, G.I. Stamoulis, and I. Katsavounidis, "A Novel Low-Power Motion Estimation Design for H.264," ASAP 2007, pp. 247-252.

[14] M.G. Koziri and A. Eleftheriadis, "Joint Quantizer Optimization for Scalable Coding," ICIP 2010, pp. 1281-1284.

[15] 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, in press.

[16] M.G. Koziri, G.I. Stamoulis, and I. Katsavounidis, "Power Reduction in an H.264 Encoder Through Algorithmic and Logic Transformations," ISLPED 2006, pp. 107-112.

[17] M.G. Koziri, D. Zacharis, I. Katsavounidis, and N. Bellas, "Implementation of the AVS Video Decoder on a Heterogeneous Dual-Core SIMD Processor," IEEE Trans. Consumer Electronics, vol. 57, no. 2, pp. 673-681, May 2011.

[18] 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.

[19] E. Monteiro, B. B. Vizzotto, C. M. Diniz, M. Maule, B. Zatt, 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.

[20] OpenMP API. http://openmp.org

[21] 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.

[22] 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.

[23] 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.

[24] S. Traverso, K. Huguenin, I. Trestian, V. Erramilli, N. Laoutaris, and K. Papagiannaki, "Social-Aware Replication in Geo-Diverse Online Systems," IEEE Trans. Parallel Distrib. Syst., vol. 26(2), pp. 584-593, 2015.

[25] N. Tziritas, T. Loukopoulos, S.U. Khan, and C.-Z. Xu, "Distributed Algorithms for the Operator Placement Problem," IEEE Trans. on Computational Social Systems, vol.2(4), pp. 182-196, 2015.

[26] C. Yan, Y. Zhang, F. Dai, and L. Li, "Highly Parallel Framework for HEVC Motion Estimation on Many-Core Platform," DCC 2013, pp. 63-72.

[27] Q. Yu, L. Zhao, and S. Ma, "Parallel AMVP candidate list construction for HEVC," VCIP 2012, pp. 1-6.

[28] 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.

[29] 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.

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