Slice-Based Parallelization in HEVC Encoding: Realizing the Potential through Efficient Load Balancing

Maria Koziri[1], Panos Papadopoulos[2], Nikos Tziritas[3], Antonios N. Dadaliaris[1], Thanasis Loukopoulos[2], Samee U. Khan[4]
[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@siat.ac.cn
[4]Electrical and Computer Engineering, North Dakota State University, Fargo, USA, samee.khan@ndsu.edu

***Abstract*—The new video coding standard HEVC (High Efficiency Video Coding) offers the desired compression performance in the era of HDTV and UHDTV, as it achieves nearly 50% bit rate saving compared to H.264/AVC. To leverage the involved computational overhead, HEVC offers three parallelization potentials namely: wavefront parallelization, tile-based and slice-based. In this paper we study slice-based parallelization of HEVC using OpenMP on the encoding part. In particular we delve on the problem of proper slice sizing to reduce load imbalances among threads. Capitalizing on existing ideas for H.264/AVC we develop a fast dynamic approach to decide on load distribution and compare it against an alternative in the HEVC literature. Through experiments with commonly used video sequences, we highlight the merits and drawbacks of the tested heuristics. We then improve upon them for the case of Low-Delay by exploiting GOP structure. The resulting algorithm is shown to clearly outperform its counterparts achieving less than 10% load imbalance in many cases.**

***Keywords*— slice parallelization; load balancing; HEVC; encoder; load distribution; video coding; OpenMP**

I. Introduction

The ever increasing demands for high definition video, has driven the development of a new video coding standard HEVC [17] capable of providing increased compress ratios without sacrificing video quality. As HEVC is gradually replacing its predecessor H.264/AVC [19], optimization of encoding and decoding time becomes of paramount importance. Recognizing the benefits from parallelization, HEVC offers three main options: tile, slice and wavefront parallelism. In this paper we turn our attention on slice level parallelism in the encoder side, using the reference software HM 16.7 [8] and OpenMP [14] for thread programming.

Our contributions include the following:

- We further confirm earlier findings that using static, fixed size slices leads to load imbalances among threads (see for instance [1]).
- We develop a heuristic called TSLB (time-based slice load balancer) which assigns load based on the time complexity of the previous frame. Two variations were tested. The first used the average CTU time per slice (TSLB-Avg) as an estimator while the second (TSLB-C) the actual time of each CTU. It should be noted that TSLB-C borrows ideas from existing work in H.264/AVC [24] without though being identical. Through experimental evaluation TSLB heuristics were shown to outperform static slice assignment as well as the algorithm presented in [1].
- Results for TSLB establish the actual time complexity of frames as a fast and efficient estimator. We further improve on initial results by exploiting GOP structure in the case of Low-Delay (LD), which is similar to but not identical with hierarchical P coding [9]. The resulting load balancer termed TSLB* is shown to be a clear winner among its counterparts, with thread imbalances rarely exceeding 20%.

To the best of our knowledge, this is the first work providing empirical evidence on the performance of five (including Static) slice balancing schemes for HEVC. Furthermore, the concept of factoring hierarchical P coding in slice balancing decisions is novel. The performance of TSLB* as shown in the experiments illustrates 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 the algorithms which are experimentally evaluated in Section IV. Finally, Section V summarizes the paper.

II. Related Work

Parallel techniques have been broadly applied in video coding since the emergence of MPEG-2 back in the 90s, see for instance [2]. In [12] parallelization of an AVS encoder with SIMD instructions was presented. In [4] a performance analysis is conducted both for the encoding and the decoding side of HEVC, illustrating the need for efficient parallel implementations. In [5] the three different parallelization opportunities in HEVC namely wavefront, tiles and slices are discussed with a particular interest on the first one, while [6] focuses on wavefront parallelization, on the decoding side.

Parallelizing the motion estimation process received much attention. In [22] different parallelization degrees are discussed varying from single CU to groups of CUs. In [18] a combined GPU – multi core CPU approach for parallel motion estimation is presented, while in [13] a comparative evaluation is provided between GPU implementation with CUDA and equivalent implementations using MPI and OpenMP for parallel motion estimation. In [21] a framework to analyze the dependencies of neighboring CTUs is introduced. CTUs form a DAG which is then scheduled for parallel computation. A similar approach is also followed in [23] but for intra encoding using the open source x265 encoder [20].

The aforementioned works differ from this paper in the parallelization scope they consider. More closely related are the works done for slice level parallelism in H.264/AVC, e.g., [7], [10], [16] and [24] whereby slice level parallelism is discussed. In [24] adaptive Macroblock assignment to slices is considered. The technique is based on weighted past average (WPA) calculation with a factor of 0.5 in order to estimate Macroblock cost for the next frame. Macroblocks are then distributed in slices so as to minimize differences in aggregated cost. The TSLB-C algorithm borrows the idea of using the actual Macroblock (CTU in HEVC) coding time as an estimator without though using WPA.

In [7] 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. Dynamically defining slice number exceeds the scope of the paper. In [10] an algorithm that adapts slice size to improve load balance is proposed. The scheme uses a fast motion estimation preprocessing step and then applies weights to Macroblocks depending on the results. As a consequence it is not directly applicable to HEVC. In [16] hierarchical parallelization is considered in two levels. In a first level different GOPs are distributed to computing nodes. Each frame in a GOP is encoded using slice-level parallelism. Adaptive slice resizing though is not considered.

Concerning HEVC, the authors in [15] evaluated slice-based parallelism under different encoding scenarios. However, load balancing slices was not taken into account. Perhaps, the closest to our work is [1] whereby SIMD based parallelization is discussed as well as slice-level parallelization with adaptive CTU-slice assignment. In the experiments of this paper we also compare the performance of our algorithms against the one in the aforementioned paper.

III. Load Balancing Algorithms

In this section we describe the algorithms that participate in the experimental evaluation of Sec. IV. We start with the algorithms that don't consider hierarchical coding and proceed with TLSB*.

A. *Static even assignment (Static)*

Under this scheme CTUs are evenly distributed to slices and this allocation remains fixed for all frames. This method is used as a performance yardstick.

B. *Weight based algorithm (Weight)*

The algorithm proposed in [1] is based on assigning a weight cost on every CU depending on whether the collocated CU in the previous frame was encoded as Skip, Inter or Intra and its corresponding depth in the quadtree. Table I reproduces the weight matrix for convenience.

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

The algorithm calculates each CTU weight as the summation of the corresponding CU weights and slice weights as the summation of the related CTU weights. It then assigns the CTUs at each slice so that slices become balanced in weight terms.

C. *Time based slice load balancing using the average CTU times in slices (TSLB-Avg)*

TSLB-Avg works on a slice level. Let S_i denote the i^{th} slice ($0 \le i \le S-1$) where S is the total number of slices. Let T_{ij} be the actual running time to compress S_i at the j^{th} frame and C_{ij} be the total number of CTUs in S_i. TSLB-Avg will assign CTUs to slices proportionally to the actual slice compression times (of the corresponding slice in the previous frame) as follows. First for each slice the difference between its time and the average slice time is calculated as per (1).

$$D_{ij} = T_{ij} - \left(\sum_{x=0}^{S-1} T_{xj}/S\right) \tag{1}$$

If the difference is positive, the slice should leave CTUs in order to close down to the average time, otherwise it should get more. The number of CTUs to be left or acquired is given by:

$$A_{ij} = \begin{cases} D_{ij}C_{ij}/T_{ij} & , \; D_{ij} > 0 \\ D_{ij}C_{(i+1)j}/T_{(i+1)j} & , \; D_{ij} < 0 \end{cases} \tag{2}$$

(2) states that if S_i should leave some of its CTUs then the average CTU time in S_i (T_{ij}/C_{ij}) should be used to calculate how many CTUs must be left in order for S_i to have computational time equaling the average of all slices. Otherwise, if it should get CTUs, these CTUs will come from the subsequent slice, thus, the average CTU time at slice S_{i+1} is used. The number of CTUs to leave or acquire is set to $\lfloor |A_{ij}| \rfloor$.

When S_i leaves $\lfloor |A_{ij}| \rfloor$ CTUs ($D_{ij}>0$ in (2)), these CTUs will be assigned on the subsequent slice S_{i+1}. This should be factored in the calculation of (1) for S_{i+1} by adding the overhead incurred by the $\lfloor |A_{ij}| \rfloor$CTUs inherited from S_i. A similar observation holds when S_i must acquire CTUs belonging to S_{i+1}. (3) and (4) incorporate the above remarks.

$$D'_{ij} = \begin{cases} D_{ij} & , i = 0 \\ D_{ij} + \frac{\lfloor |A'_{(i-1)j}| \rfloor T_{(i-1)j}}{C_{(i-1)j}} & , i > 0 \land A_{(i-1)j} > 0 \\ D_{ij} - \frac{\lfloor |A'_{(i-1)j}| \rfloor T_{ij}}{C_{ij}} & , i > 0 \land A_{(i-1)j} < 0 \end{cases} \quad (3)$$

$$A'_{ij} = \begin{cases} D'_{ij} C_{ij} / T_{ij} & , D'_{ij} > 0 \\ D'_{ij} C_{(i+1)j} / T_{(i+1)j} & , D'_{ij} < 0 \end{cases} \quad (4)$$

Starting from the first slice (S_0) and continuing until $S_{S\text{-}2}$ in an iterative manner, the algorithm uses (1), (3) and (4) to calculate how many CTUs a slice must get or leave. The last slice $S_{S\text{-}1}$ gets the remaining unassigned CTUs. To have a visual representation of how TSLB-Avg performs, Fig. 1 shows the size assignment of 4 slices in the 5th frame of the Bosphorus sequence [11]. Notice, that the third slice which includes most of the boat movement is smaller compared to the rest.

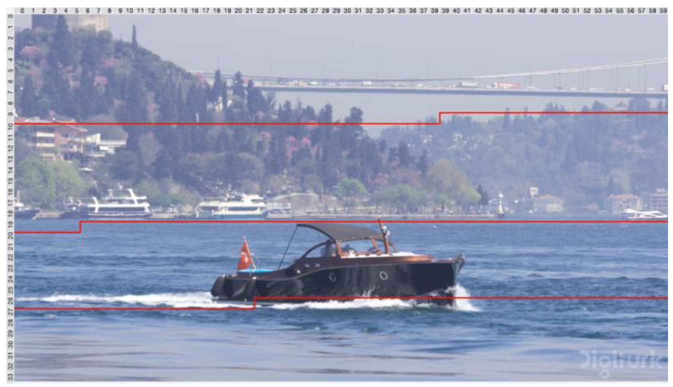

Figure 1. Screenshot from Bosphorus (frame 5).

D. *Time based slice load balancing using actual CTU times (TSLB-C)*

TSLB-C works in a similar manner to TSLB-Avg. The difference is that instead of using average CTU times in (3) and (4) it uses the actual CTU coding times.

E. *Time based slice load balancing for Low Delay (TSLB*)*

One of the common test conditions defined in JCT-VC [3] is LD (Low Delay) which uses a hierarchical GOP structure. In all the experiments of the paper we used the default configuration for hierarchical P frames in the reference software HM 16.7 which is also depicted in Fig. 2.

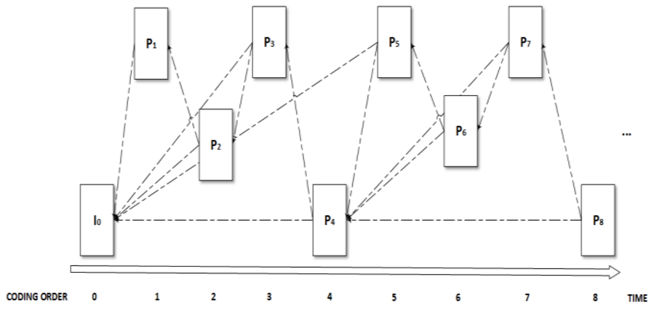

Figure 2. GOP structure.

Hierarchical P frames prediction structure is based on the decomposition into layers. Within each layer frames share the same parameters (e.g QP offsets, QP factors, temporal id etc.) and the same pattern in the group of reference pictures. In the case of temporal scalability, those layers are known as temporal layers and the prediction can only occur from a picture in the same or lower layer [9]. This restriction is not present in the structure introduced in LD configuration of HEVC, as each frame may always reference the previous one, regardless of the layer it belongs to. However, as the scope of this paper does not cover scalability this has no impact.

The intuition behind TSLB* 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. Notice that TSLB-Avg, TSLB-C and Weight will use P7 instead. Another change TSLB* introduces, 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 from P7 instead of P8. The assignment process of TSLB* is summarized and generalized for arbitrary GOP sizes (let G) in the following equations:

$$E_{ij} = \begin{cases} T_{ij} & , 1 + jmodG \in [2, G-1] \\ C_{ij} T_{i(j-G+1)} / C_{i(j-G+1)} & , 1 + jmodG = G \\ C_{ij} T_{i(j-1)} / C_{i(j-1)} & , 1 + jmodG = 1 \end{cases} \quad (5)$$

$$D_{ij} = E_{ij} - \left(\sum_{x=0}^{S-1} E_{xj} / S\right) \quad (6)$$

$$D'_{ij} = \begin{cases} D_{ij} & , i = 0 \\ D_{ij} + \frac{\lfloor |A'_{(i-1)j}| \rfloor E_{(i-1)j}}{C_{(i-1)j}} & , i > 0 \land A_{(i-1)j} > 0 \\ D_{ij} - \frac{\lfloor |A'_{(i-1)j}| \rfloor E_{ij}}{C_{ij}} & , i > 0 \land A_{(i-1)j} < 0 \end{cases} \quad (7)$$

$$A'_{ij} = \begin{cases} D'_{ij} C_{ij} / E_{ij} & , D'_{ij} > 0 \\ D'_{ij} C_{(i+1)j} / E_{(i+1)j} & , D'_{ij} < 0 \end{cases} \quad (8)$$

The rest of the algorithm is similar to TSLB-Avg, namely at each frame j TSLB* starts calculating the assignment from S_0 using (5)-(8) adding or subtracting $\lfloor |A'_{ij}| \rfloor$ CTUs to the current assignment and proceeds up to $S_{S\text{-}2}$ in an iterative manner. The last unassigned CTUs are allocated to $S_{S\text{-}1}$. When implementing the algorithm we chose to use TSLB-Avg for the first GOP and the estimations of TSLB* from the second GOP onwards.

TABLE II. 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

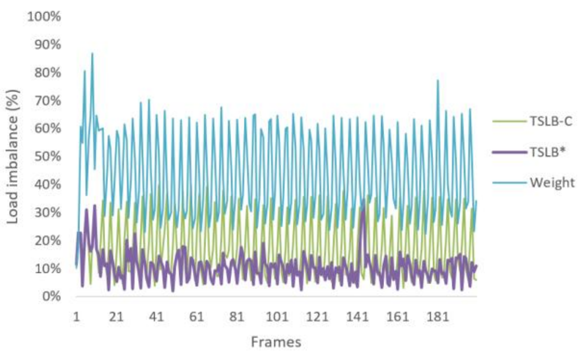

Figure 3(a). Bosphorus, 4 slices.

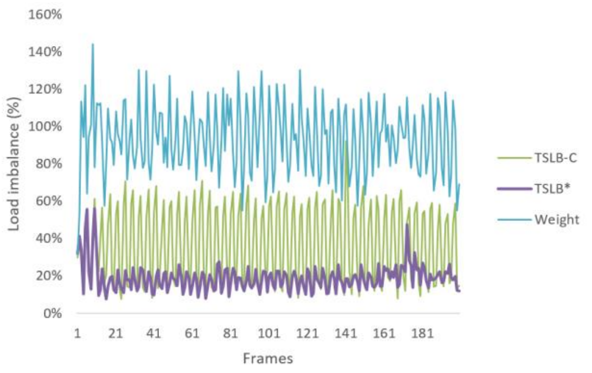

Figure 3(b). Bosphorus, 12 slices.

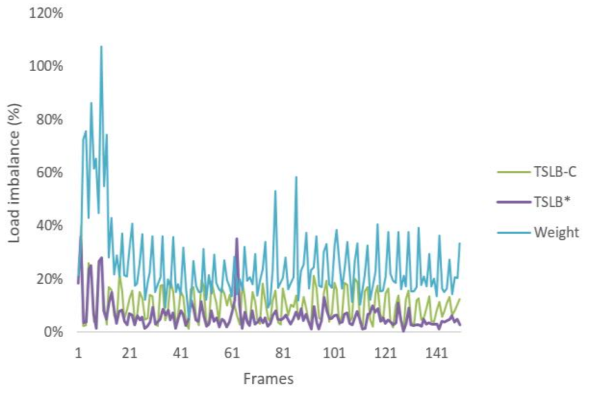

Figure 4(a). Traffic, 4slices.

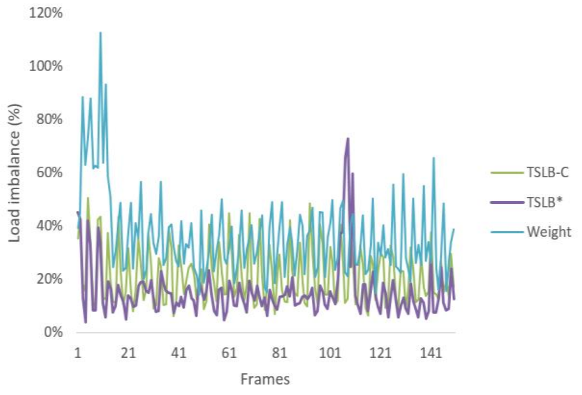

Figure 4(b). Traffic, 12 slices.

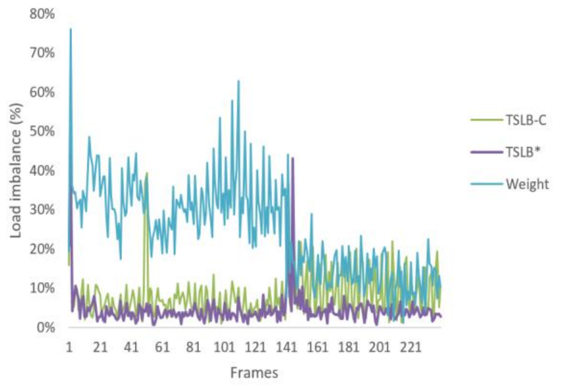

Figure 5(a). Kimono, 4 slices.

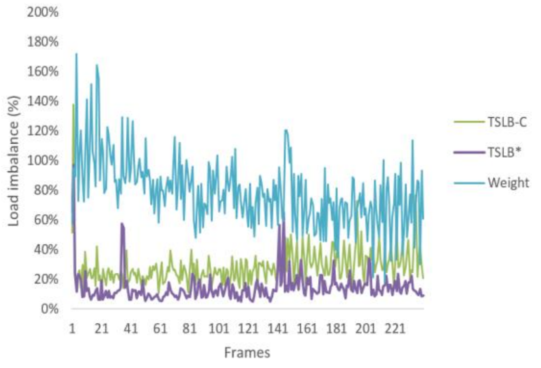

Figure 5(b). Kimono, 12 slices.

IV. EXPERIMENTS

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 three sequences (summarized in Table II) one each for FullHD, 2K and 4K. 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 with the structure shown in Fig. 2. 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 measured the performance of the algorithms from two aspects. The first is the time required to process a frame, while the second is the load imbalance incurred among the execution time of slices measured as the following percentage:
100(MAX_Slice_Time – MIN_Slice_Time)/MIN_Slice_Time

Figs. 3-5 plot the imbalance experienced in the three sequences for two different number of slices: 4 and 12. To avoid cluttering, the performance of Static and TSLB-Avg are omitted. The first gave performance worse than the Weight algorithm, while the second one comparable to TSLB-C. The figures show that there exist periodic peaks which correspond to GOP changes. It is evident from the plots that TSLB* (the intended line) clearly outperforms other alternatives especially in the 4K sequence.

We would like to note that the peak incurred by TSLB* in the Kimono sequence around frame 141 is due to scene change. As part of our future work we plan on incorporating scene detection in TSLB*. Contrary to the above the peak incurred in Fig. 4(b) around frame 110 is not due to scene change. Nevertheless, this doesn't diminish the overall performance of TSLB*.

Next we conducted experiments with the following slice numbers: 2, 4, 8, 12 and 24. Recall from the experimental setup that there are 12 cores available in the server running the experiments. Nevertheless, we wanted to test how the algorithms will fair when less cores than slices are available. Table III summarizes the relevant speedups achieved by each algorithm. Bolded entries indicate the winner in every run.

TABLE III. SPEEDUPS

		Slice Number				
		2	**4**	**8**	**12**	**24**
Bosphorus	Static	1.74	3.35	5.83	8.09	10.44
	TSLB-Avg	1.92	3.67	6.90	10.03	12.16
	TSLB-C	1.93	3.66	6.88	9.90	12.15
	TSLB*	**1.94**	**3.76**	**7.32**	**10.63**	**12.45**
	Weight	1.74	3.29	5.80	8.14	10.63
Traffic	Static	1.94	3.43	6.45	9.26	11.33
	TSLB-Avg	1.92	3.71	7.24	10.41	**11.95**
	TSLB-C	1.93	3.72	7.18	**10.52**	11.94
	TSLB*	**1.95**	**3.79**	**7.36**	10.48	11.71
	Weight	1.91	3.57	6.85	9.89	11.64
Kimono	Static	1.85	3.56	6.76	9.69	11.46
	TSLB-Avg	**1.96**	3.81	7.35	10.67	12.05
	TSLB-C	1.95	3.79	7.35	10.64	11.43
	TSLB*	**1.96**	**3.88**	**7.39**	**10.81**	**12.10**
	Weight	1.88	3.53	6.74	9.57	11.44

TSLB* is a clear winner in the Bosphorus and Kimono sequences, while for a larger slice number in the Traffic sequence it is defeated by TSLB variants. Another observation that can be made is that the performance difference versus the Static algorithm tends to increase to the number of slices. We should also note that the performance of TSLB* is particularly high in the 4K sequence, giving a +2.52 speedup factor versus Static and +0.6 versus the second alternative when slices equaled 12. In contrast, the Weight algorithm achieves only marginally better performance compared to Static. Finally, the run with 24 slices over 12 cores provides a margin for improvement for all algorithms, while not changing the relevant performance order in most cases. This result is particularly important indicating that further improvement can be expected for the algorithms presented in the paper, when using the hyper threading capabilities of some processors.

To better illustrate the performance difference of algorithms in Figs. 6-8 we plot the percentage of improvement in execution time terms of each algorithm as compared to the Static. Specifically, we measure the improvement as follows: (Static_time-Alg_time)/Static_time. TSLB* (bold unmarked line) is shown to reduce the execution time of Static by more than 20% in the Bosphorus, more than 10% for Traffic and more than 8% for the Kimono sequence.

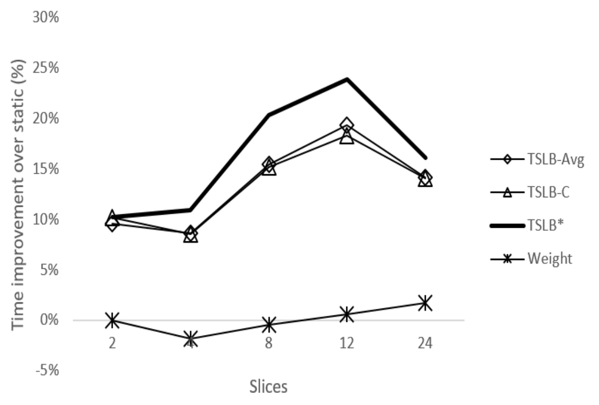

Figure 6. Bosphorus.

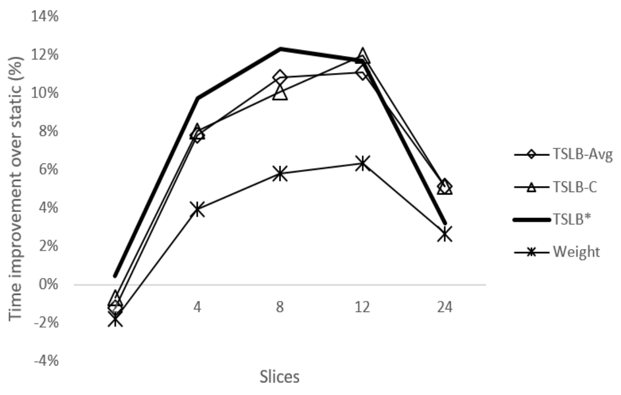

Figure 7. Traffic.

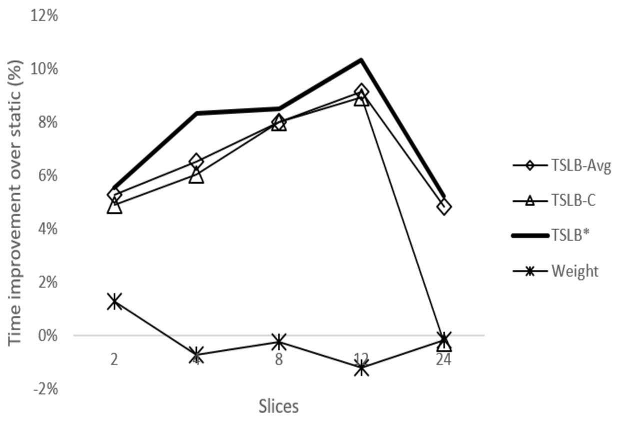

Figure 8. Kimono.

A last note concerns video quality. It was observed in our experiments that as slice number increased, quality dropped. This trend is known from H.264/AVC. Nevertheless, for a fixed slice number both PSNR and bit rate experienced only tiny differences among the algorithms. This is especially encouraging for TSLB* since it indicates that its performance gains, especially against the Static, come at no cost quality wise. We should also like to add that from our experience, once slice parallelization is implemented, developing any of the algorithms described (TSLB* as well) demands little programming effort. Hence, TSLB* poses as the most viable solution (currently) to the problem of slice balancing in particular when Low Delay hierarchical P frames are considered.

Summarizing our findings we can state the following:

- There exists a performance margin to gain versus the Static approach. This margin depends on the sequence as well as the slices used.
- Actual coding time of slices is a superior criterion compared to the preprocessed weight costs in [1].
- By incorporating GOP structure in the decision mechanism a very efficient load balancer can be designed.

V. Conclusions and Future Work

In this paper we tackled the problem of load balancing slices in HEVC. We proposed a simple and fast algorithm named TSLB that comes in two versions. In the first one slice balancing decisions are taken using the recorded slice time while in the second CTU times. The initial design is extended for hierarchical GOP structures, resulting in TSLB*. TSLB* was shown to outperform both the Static option and another alternative from the relevant literature. Reductions in the execution time of Static slice-parallelization were between 8% and 25% for the majority of test cases.

Acknowledgment

This work was partially supported by travel grant of the postgraduate program: “Informatics and Computational Biomedicine”, School of Science, Univ. of Thessaly.

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] S.M. Akramullah, I. Ahmad, and M.L. Liou, “A Data-Parallel Approach for Real-Time MPEG-2 Video Encoding,” Journal of Parallel and Distributed Computing, vol. 30, pp. 129-146, 1995.

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

[4] 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.

[5] 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.

[6] C. C. Chi, M. A. Mesa, J. Lucas, B. H. H. Juurlink, and T. Schierl, “Parallel HEVC Decoding on Multi- and Many-core Architectures - A Power and Performance Analysis,” Signal Processing Systems vol. 71(3), pp. 247-260, 2013.

[7] 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.

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

[9] 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.

[10] B. Jung, and B. Jeon, “Adaptive slice-level parallelism for H.264/AVC encoding using pre macroblock mode selection,” J. Visual Communication and Image Representation, vol. 19(8), pp. 558-572, 2008.

[11] 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.

[12] 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(2), pp. 673-681, 2011.

[13] 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.

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

[15] 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.

[16] A. Rodríguez, A. González, and M. P. Malumbres, “Hierarchical Parallelization of an H.264/AVC Video Encoder,” in Proc. PARELEC 2006, pp. 363-368.

[17] 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.

[18] 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.

[19] 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.

[20] x265 HEVC encoder. http://x265.org.

[21] 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.

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

[23] Y. Zhao, L. Song, X. Wang, M. Chen, and J. Wang, “Efficient realization of parallel HEVC intra encoding,” In Proc. ICME Workshops pp. 1-6, 2013.

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