



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ
ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ
ΒΙΟΙΑΤΡΙΚΗ»**

**EFFICIENT SLICE AND TILE BASED PARALLELIZATION
OF VIDEO ENCODING IN HEVC**

Papadopoulos Panagiotis

MASTER THESIS

Supervisor

**Loukopoulos Athanasios, Lecturer at the Department of
Computer Science and Biomedical Informatics,
University of Thessaly**

Lamia, 3/2017



**ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΔΙΑΤΜΗΜΑΤΙΚΟ ΜΕΤΑΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΠΛΗΡΟΦΟΡΙΚΗ ΚΑΙ ΥΠΟΛΟΓΙΣΤΙΚΗ ΒΙΟΙΑΤΡΙΚΗ
ΚΑΤΕΥΘΥΝΣΗ**

**«ΠΛΗΡΟΦΟΡΙΚΗ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΑΣΦΑΛΕΙΑ,
ΔΙΑΧΕΙΡΙΣΗ ΜΕΓΑΛΟΥ ΟΓΚΟΥ ΔΕΔΟΜΕΝΩΝ ΚΑΙ
ΠΡΟΣΟΜΟΙΩΣΗ»**

**EFFICIENT SLICE AND TILE BASED PARALLELIZATION
OF VIDEO ENCODING IN HEVC**

Papadopoulos Panagiotis

MASTER THESIS

Supervisor

**Loukopoulos Athanasios, Lecturer at the Department of Computer Science
and Biomedical Informatics, University of Thessaly**

Lamia, 3/2017

«Υπεύθυνη Δήλωση μη λογοκλοπής και ανάληψης προσωπικής ευθύνης»

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, και γνωρίζοντας τις συνέπειες της λογοκλοπής, δηλώνω υπεύθυνα και ενυπογράφως ότι η παρούσα εργασία με τίτλο «Efficient slice and tile based parallelization of video encoding in HEVC» αποτελεί προϊόν αυστηρά προσωπικής εργασίας και όλες οι πηγές από τις οποίες χρησιμοποίησα δεδομένα, ιδέες, φράσεις, προτάσεις ή λέξεις, είτε επακριβώς (όπως υπάρχουν στο πρωτότυπο ή μεταφρασμένες) είτε με παράφραση, έχουν δηλωθεί κατάλληλα και ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή. Αναλαμβάνω πλήρως, ατομικά και προσωπικά, όλες τις νομικές και διοικητικές συνέπειες που δύναται να προκύψουν στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής.

Ο/Η ΔΗΛΩΝ/ΟΥΣΑ

Ημερομηνία: ../3/2017

Υπογραφή

EFFICIENT SLICE AND TILE BASED PARALLELIZATION OF VIDEO ENCODING IN HEVC

Papadopoulos Panagiotis

Three-member Committee:

Loukopoulos Athanasios, Lecturer at the Department of Computer Science and Biomedical Informatics, University of Thessaly (supervisor)

Stamoulis George, Professor at the Department of Electrical and Computer Engineering, University of Thessaly

Anagnostopoulos Ioannis, Assistant Professor at the Department of Computer Science and Biomedical Informatics, University of Thessaly

Scientific Advisor:

Koziri Maria, Adjunct Professor at the Department of Computer Science, University of Thessaly

Contents

CHAPTER 1: INTRODUCTION	1
CHAPTER 2: RELATED WORK.....	4
CHAPTER 3: SLICE-BASED PARALLELIZATION	7
Static even assignment (Static)	7
Weight based algorithm (Weight).....	7
Time based slice load balancing using average CTU times (TSLB -Avg).....	7
Time based slice load balancing using actual CTU times (TSLB-C).....	9
Time based slice load balancing for Low Delay (TSLB*)	9
CHAPTER 4: TILE-BASED PARALLELIZATION	12
CHAPTER 5: EXPERIMENTS.....	14
Slice Partitioning.....	14
Tile Partitioning	20
CHAPTER 6: CONCLUSIONS AND FUTURE WORK.....	27
REFERENCES	28
APPENDIX.....	30
LIST OF FIGURES	30
LIST OF TABLES.....	30

CHAPTER 1: INTRODUCTION

Parts of the master thesis have been published in [37] and in [38]. 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 master thesis 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.

Also nowadays, large multimedia content providers and social media networks struggle at keeping pace with the popularity explosion of smart devices [28] and the resource demands it entails in order to perform filtering [25], processing [36], storage and delivery [35]. As an example, Cisco reported in [27] 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) [30] 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 [19] to HEVC [17], 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 [4]. 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) [5]. 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 thesis 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 thesis is organized as follows: Chapter II provides a brief overview of the related work. Chapter III illustrates the slice-based parallelization algorithms which are experimentally evaluated in Chapter V. Chapter IV illustrates the tile-based parallelization algorithm *TTLB* which is experimentally evaluated in Chapter V. Finally, Chapter VI summarizes the thesis and gives the conclusions.

CHAPTER 2: 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 thesis 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 thesis. 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 we also compare the performance of our algorithms against the one in the aforementioned paper.

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 [12] 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 [29] and [32], has also attracted SIMD parallelization efforts. 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 of motion estimation with CUDA and equivalent implementations using MPI and OpenMP. The authors concluded that GPUs offer significant advantages. In [21] a framework to analyze the dependencies of neighboring CTUs is introduced. CTUs form a DAG which is then scheduled for parallel computation. Finally, in [22] 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., [7] and [24] to name a few, and in HEVC, e.g., [1], [31], [15]. In [24] adaptive Macroblock assignment to slices based on weighted past average (WPA) of Macroblock coding times is considered. A similar approach was evaluated in [31] for HEVC. 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. Concerning HEVC, the authors in [15] 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 thesis, some of the ideas discussed there are applicable in the case of tiles as well. Specifically, our proposed algorithm TTLB is inspired by [24] in order to use the coding times of CTUs to estimate tile load. Furthermore, the idea of [7] 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: [26], [11], [33] and [34]. In [33] 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 [11]. There too, only fixed size tiles were considered.

The motivation for the tile partitioning algorithm in [34] 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 [4], [5] 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 [26] 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 [26] as orthogonal to ours.

CHAPTER 3: SLICE-BASED PARALLELIZATION

In this chapter we describe the algorithms that participate in the experimental evaluation of Chapter V. We start with the algorithms that don't consider hierarchical coding and proceed with TLSB* (published in [37]).

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.

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.

Time based slice load balancing using average CTU times (TSLB -Avg)

TSLB-Avg works on a slice level. Let S_i denote the i^{th} slice ($0 \leq i \leq S-1$) 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).

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

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:

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

(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.

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

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

Starting from the first slice (S_0) and continuing until S_{S-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-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).

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.

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 we used the default configuration for hierarchical P frames in the reference software HM 16.7 which is also depicted in Fig. 2.

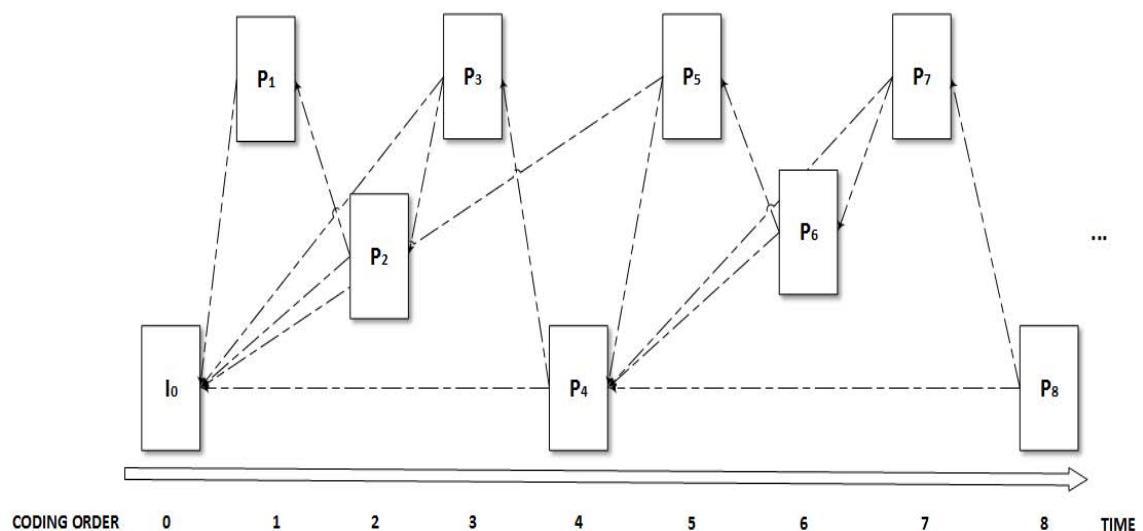


Figure 2. GOP structure.

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 thesis 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:

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

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

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

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

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-2} in an iterative manner. The last unassigned CTUs are allocated to S_{S-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

CHAPTER 4: TILE-BASED PARALLELIZATION

The Time-based Tile Load Balancing algorithm (TTLB) (published in [38]) 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 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 ($1 \leq k \leq N$), and TR_l be the height in CTUs of the l^{th} tile row, ($1 \leq l \leq M$). The algorithm assigns tile column widths using the following:

$$W = \sum_{j=1}^Y C_j \quad (9)$$

$$TC_k = \sum_{j=s}^e C_j \leq \left\lfloor \frac{W}{N} \right\rfloor < \sum_{j=s}^{e+1} C_j : s = 1 + \sum_{u=1}^{k-1} TC_u, (1 \leq k \leq N-1) \quad (10)$$

$$TC_N = Y - \sum_{k=1}^{N-1} TC_k \quad (11)$$

Namely, it calculates the total time of all CTUs in (9), 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 (10) and (11). 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:

$$TR_l = \sum_{i=s}^e R_i \leq \left\lfloor \frac{W}{M} \right\rfloor < \sum_{i=s}^{e+1} R_i : s = 1 + \sum_{u=1}^{l-1} TR_u, (1 \leq l \leq M-1) \quad (12)$$

$$TR_M = X - \sum_{l=1}^{M-1} TR_l \quad (13)$$

Figs. 3,4 presents screenshots from the Bosphorus sequence [11] with a partitioning in 12 tiles using TTLB. Notice that compared to the initial cut at frame 0 (Fig. 3), TTLB

in frame 5 (Fig. 4) has reduced the size of the tile enclosing the boat where most of the motion takes place.

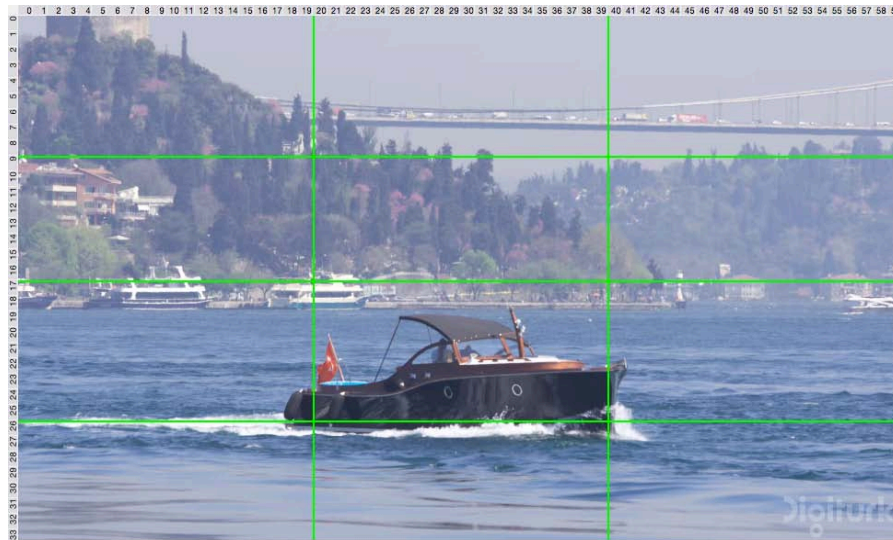


Figure 3. Screenshot from Bosphorus (frame 0).

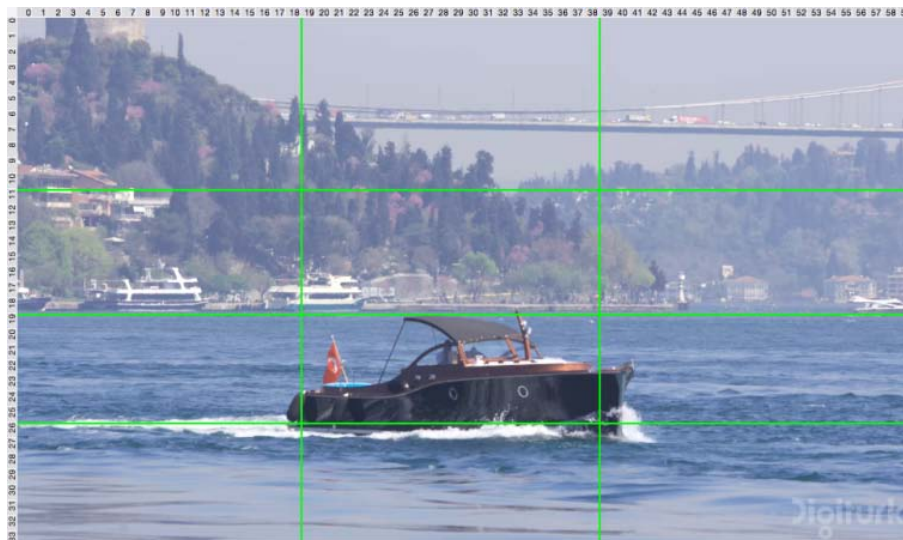


Figure 4. Screenshot from Bosphorus (frame 5).

CHAPTER 5: EXPERIMENTS

Slice Partitioning

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:

$$100(\text{MAX_Slice_Time} - \text{MIN_Slice_Time})/\text{MIN_Slice_Time}$$

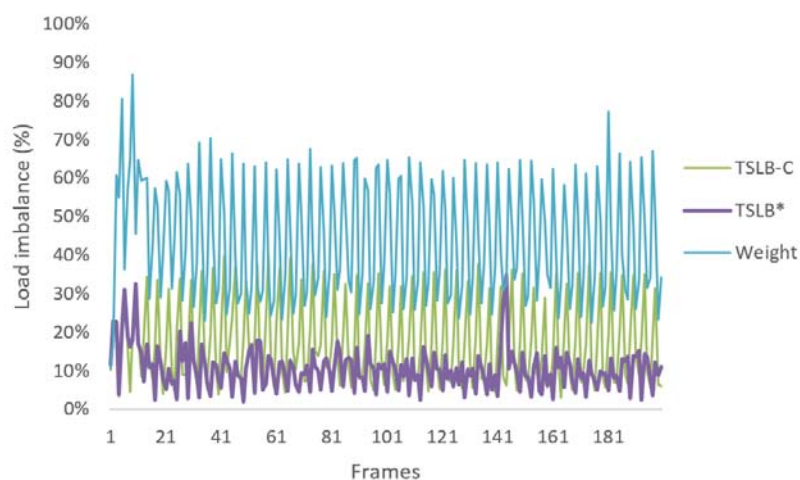


Figure 5. Bosphorus, 4 slices.

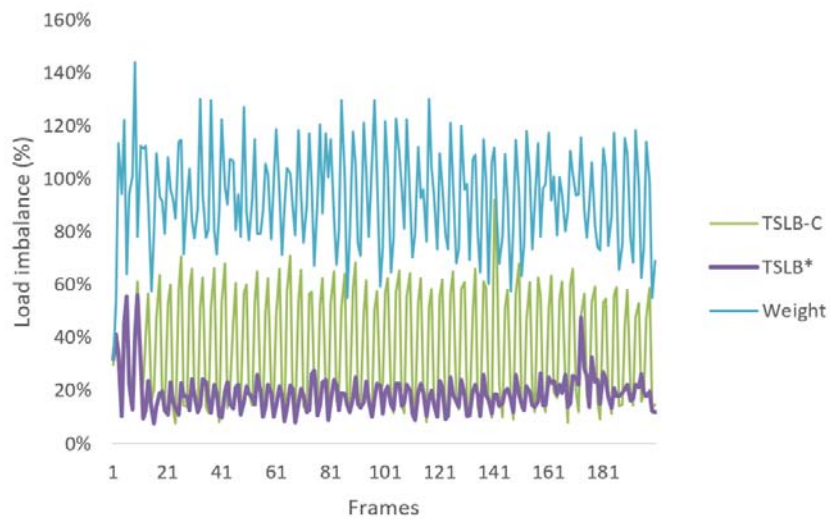


Figure 6. Bosphorus, 12 slices.

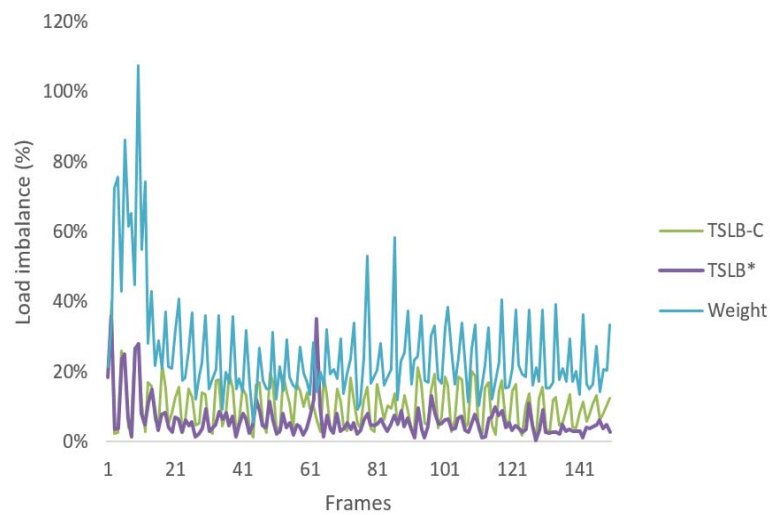


Figure 7. Traffic, 4 slices.

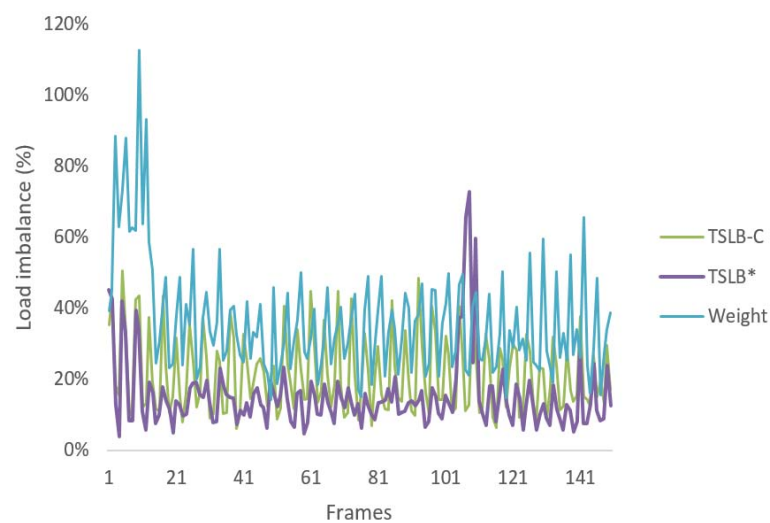


Figure 8. Traffic, 12 slices.

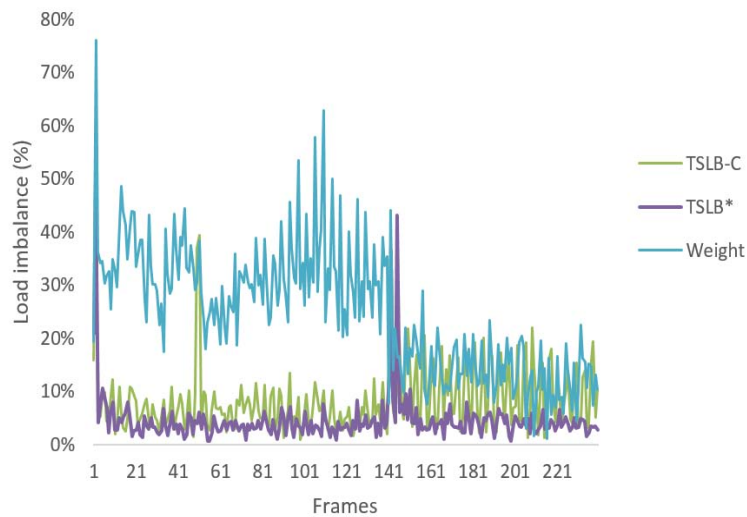


Figure 9. Kimono, 4 slices.

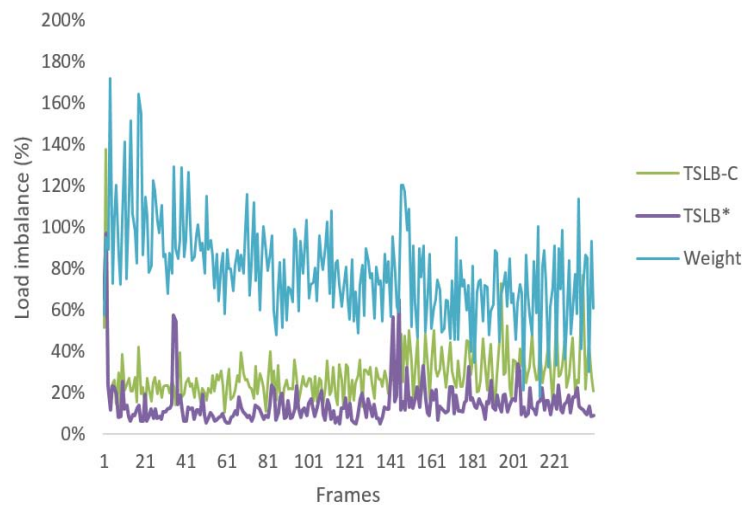


Figure 10. Kimono, 12 slices.

Figs. 5-10 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. 8 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 presented algorithms, when using the hyper threading capabilities of some processors.

To better illustrate the performance difference of algorithms in Figs. 11-13 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: $(\text{Static_time} - \text{Alg_time}) / \text{Static_time}$. 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.

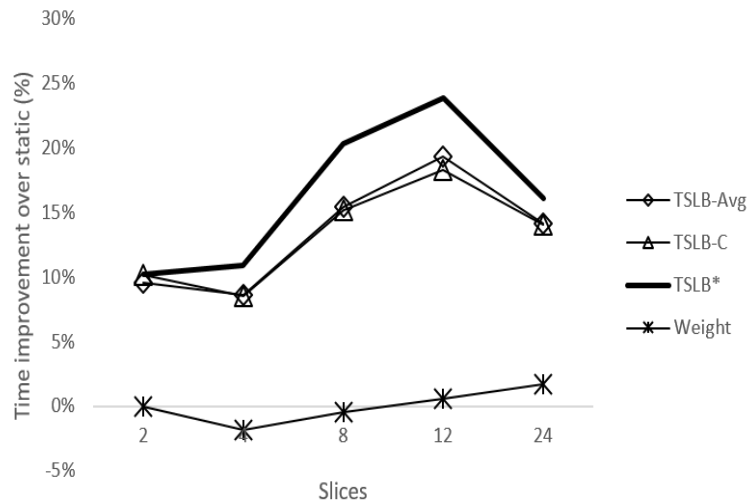


Figure 11. Bosphorus.

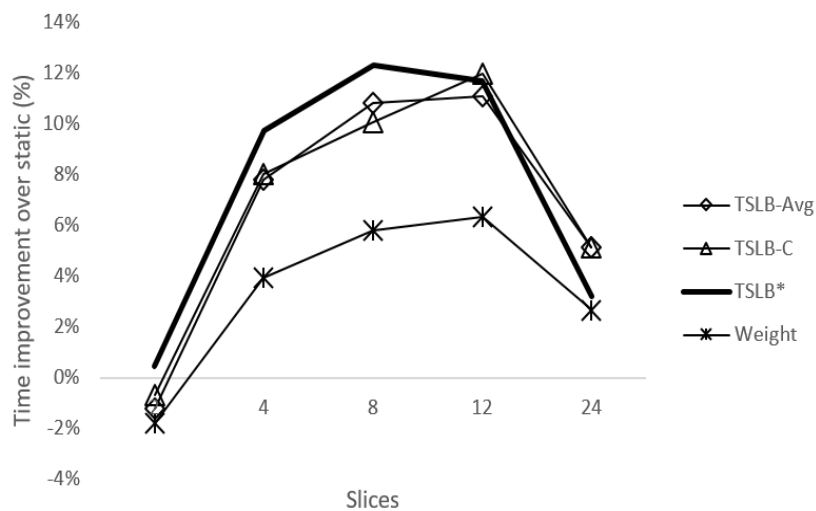


Figure 12. Traffic.

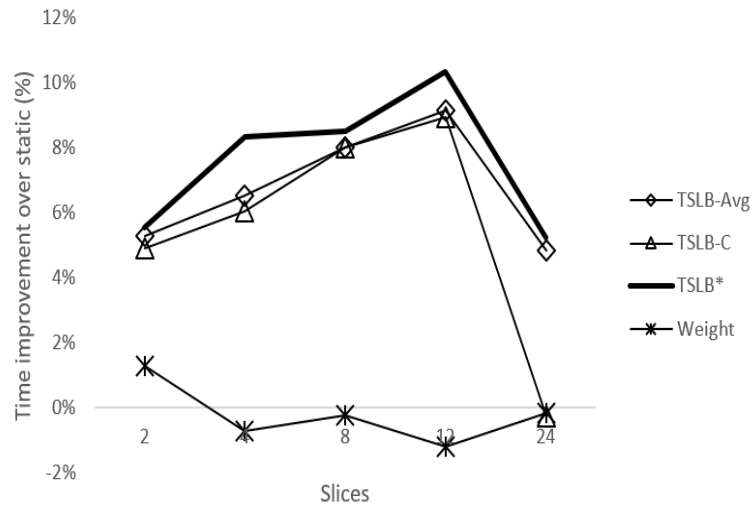


Figure 13. Kimono.

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.

Tile Partitioning

We implemented TTLB using the reference software HM 16.7 [8] and OpenMP [14] 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 II.

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 [3] which is similar but not identical to hierarchical P coding [9]. 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:

$$100(\text{MAX_Tile_Time} - \text{MIN_Tile_Time})/\text{MIN_Tile_Time}$$

Figs. 14-19 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. 16), 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. 16 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.

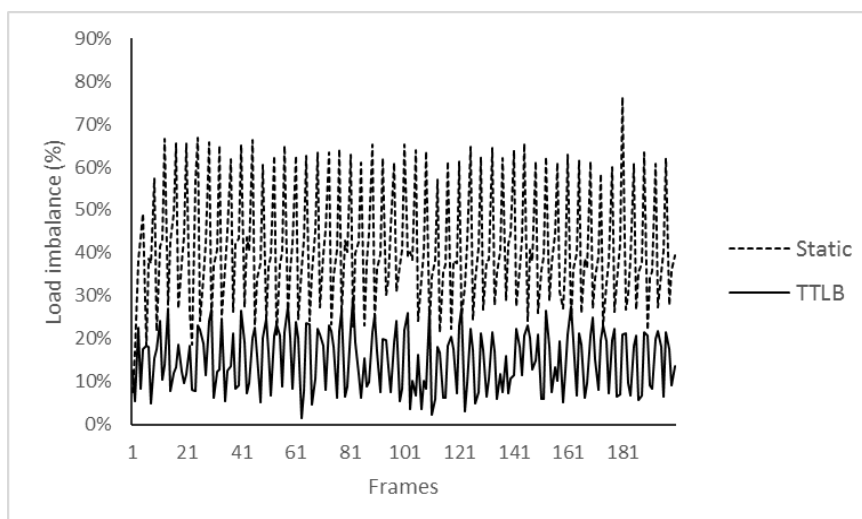


Figure 14. Bosphorus, 4 tiles.

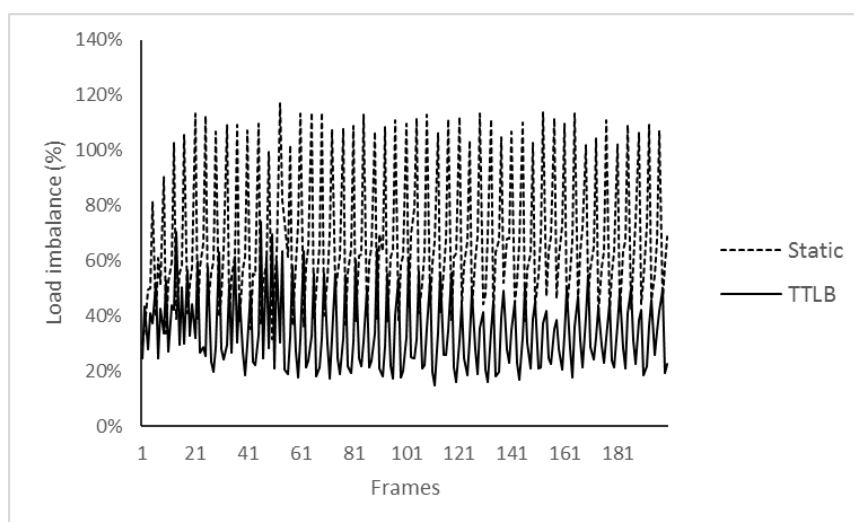


Figure 15. Bosphorus, 12 tiles.

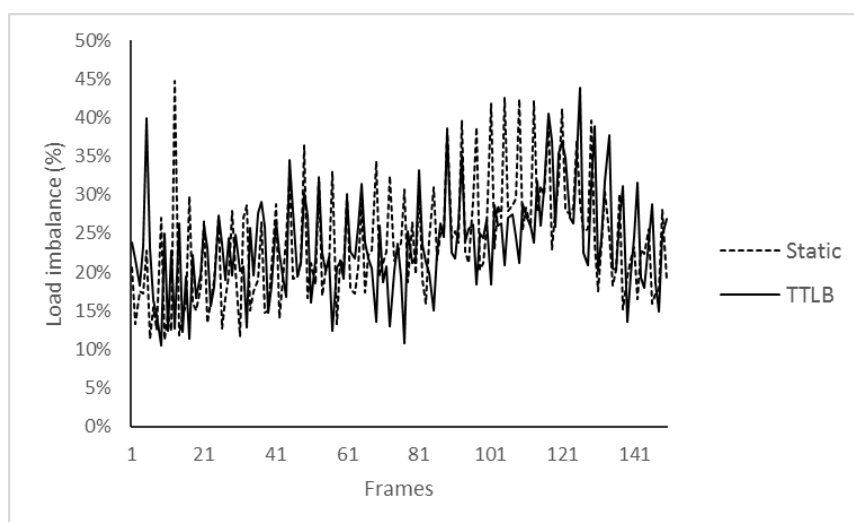


Figure 16. Traffic, 4 tiles.

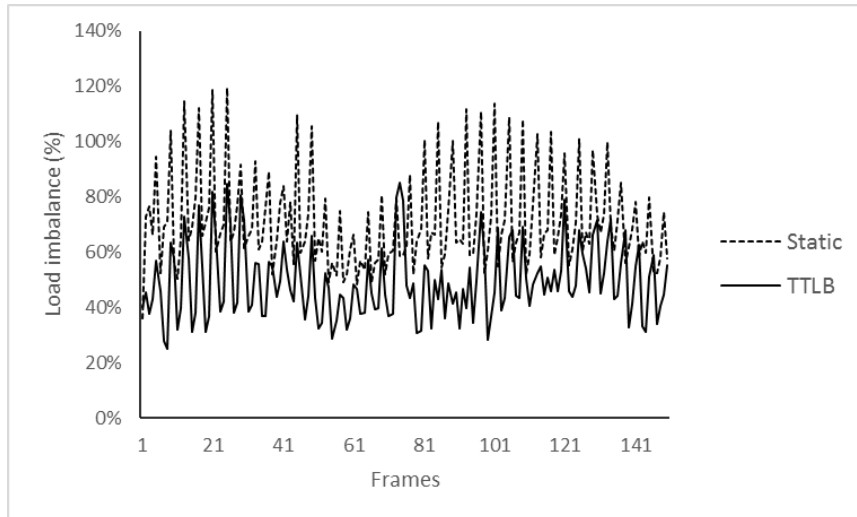


Figure 17. Traffic, 12 tiles.

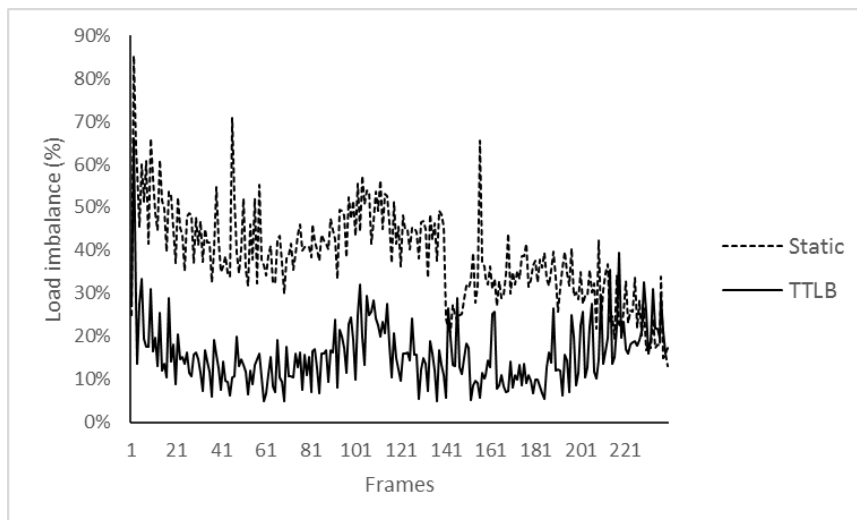


Figure 18. Kimono, 4 tiles.

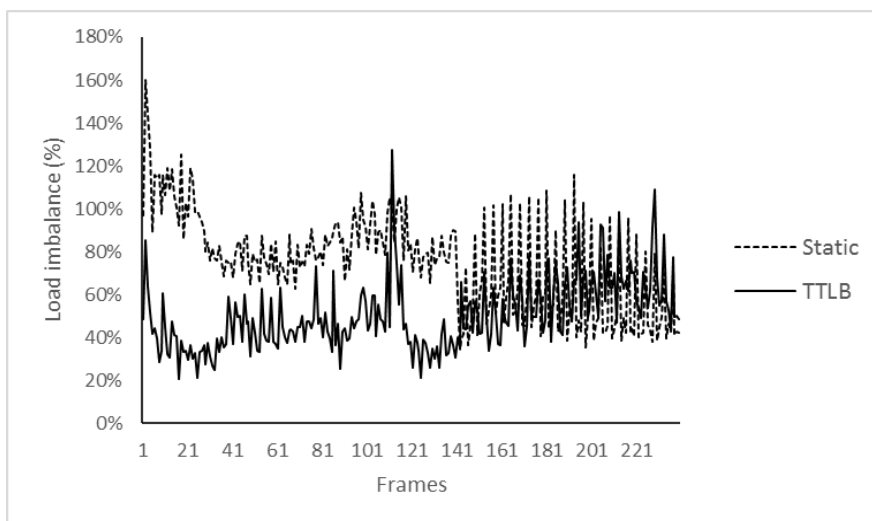


Figure 19. Kimono, 12 tiles.

Next we plot the speedups over a base scenario with no tile parallelization. Results are shown in Figs. 20-25 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. 21). In the Traffic sequence the gains are less impressive and are considerable only for QP=22.

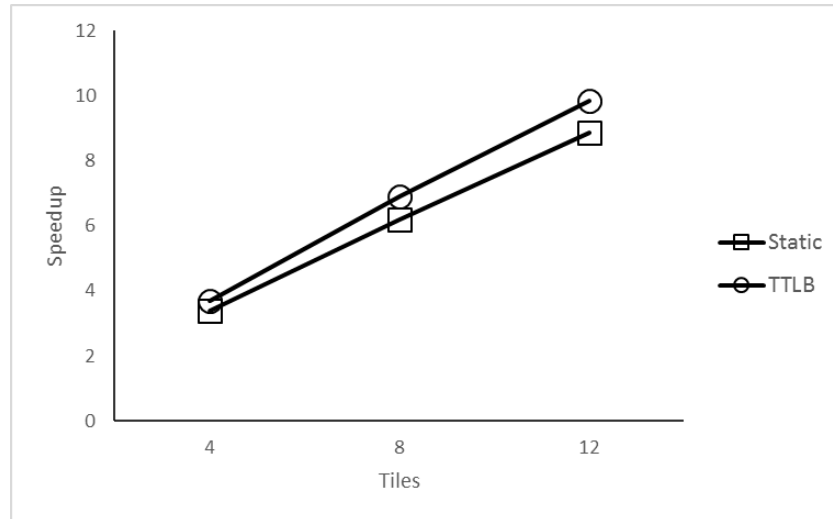


Figure 20. Bosphorus, QP=32.

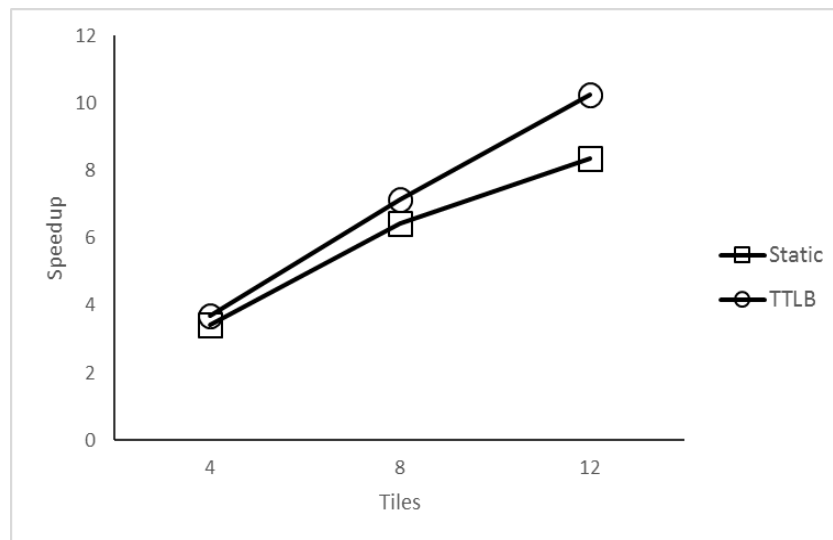


Figure 21. Bosphorus, QP=22.

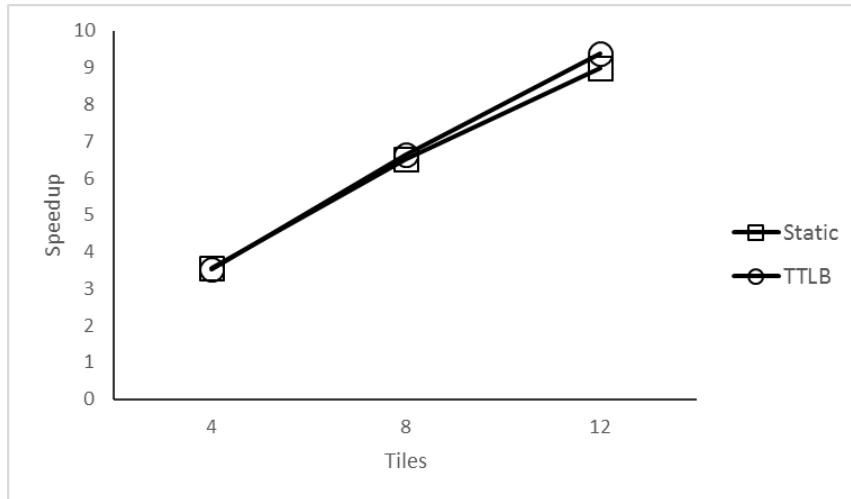


Figure 22. Traffic, QP=32.

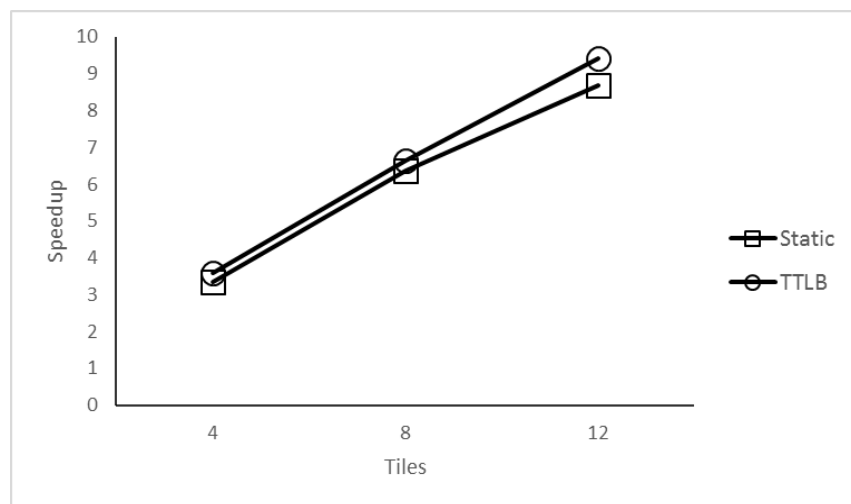


Figure 23. Traffic, QP=22.

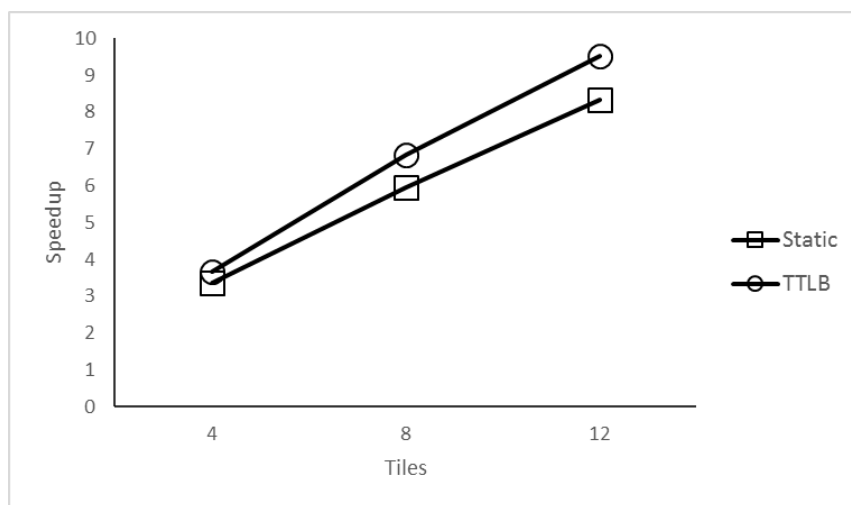


Figure 24. Kimono, QP=32.

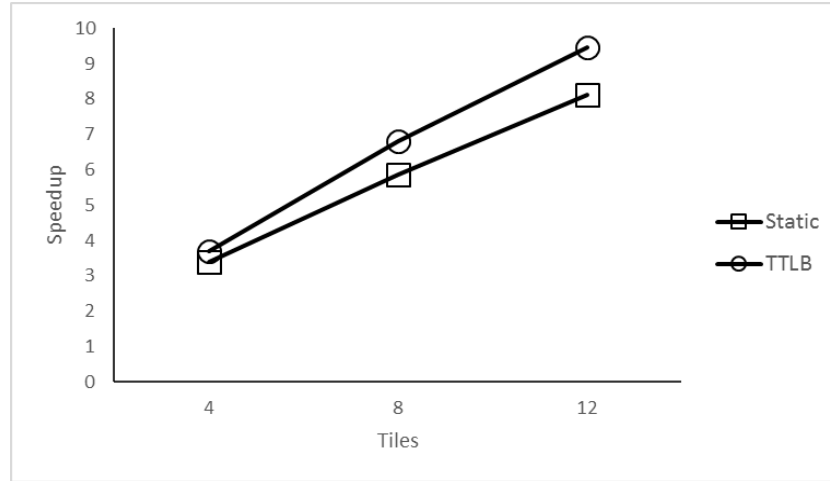


Figure 25. Kimono, QP=22.

Finally, we measured the impact on quality TTLB has. Table IV 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:

$$100(\text{bitrate}(\text{TTLB}) - \text{bitrate}(\text{Static})) / \text{bitrate}(\text{Static})$$

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 IV. QUALITY METRICS

		Tile Number					
		Y-PSNR			bitrate %		
	<i>QP</i>	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%

CHAPTER 6: CONCLUSIONS AND FUTURE WORK

Firstly, in this master thesis 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.

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 master thesis we also 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] 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 Technol.* 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, "Parallelizing 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.
- [25] 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.
- [26] C. Blumenberg, D. Palomino, S. Bampi, and B. Zatt, "Adaptive content-based Tile partitioning algorithm for the HEVC standard," PCS 2013, pp. 185-188.
- [27] 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>
- [28] 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.
- [29] 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.
- [30] M.G. Koziri and A. Eleftheriadis, "Joint Quantizer Optimization for Scalable Coding," ICIP 2010, pp. 1281-1284.
- [31] 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.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] 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.
- [36] 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.
- [37] 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," Proc. 18th Int. Workshop on Multimedia Signal Processing (MMSP 2016), IEEE, Montreal, Canada, Sept. 2016.
- [38] M.G. Koziri, P. Papadopoulos, N. Tziritas, A.N. Dadaliaris, T. Loukopoulos, S.U. Khan, and C.-Z. Xu, "Adaptive Tile Parallelization for Fast Video Encoding in HEVC", Proc. 12th IEEE Int. Conf. on Green Computing and Communications (GreenCom 2016), IEEE, Chengdu, China, Dec. 2016.

APPENDIX

LIST OF FIGURES

Figure 1. Screenshot from Bosphorus (frame 5).....	9
Figure 2. GOP structure.	9
Figure 3. Screenshot from Bosphorus (frame 0).....	13
Figure 4. Screenshot from Bosphorus (frame 5).....	13
Figure 5. Bosphorus, 4 slices.	14
Figure 6. Bosphorus, 12 slices.	15
Figure 7. Traffic, 4 slices.	15
Figure 8. Traffic, 12 slices.	15
Figure 9. Kimono, 4 slices.	16
Figure 10. Kimono, 12 slices.	16
Figure 11. Bosphorus.	18
Figure 12. Traffic.	18
Figure 13. Kimono.	19
Figure 14. Bosphorus, 4 tiles.	21
Figure 15. Bosphorus, 12 tiles.	21
Figure 16. Traffic, 4 tiles.	21
Figure 17. Traffic, 12 tiles.	22
Figure 18. Kimono, 4 tiles.	22
Figure 19. Kimono, 12 tiles.	22
Figure 20. Bosphorus, QP=32.....	23
Figure 21. Bosphorus, QP=22.....	23
Figure 22. Traffic, QP=32.....	24
Figure 23. Traffic, QP=22.....	24
Figure 24. Kimono, QP=32.....	24
Figure 25. Kimono, QP=22.....	25

LIST OF TABLES

TABLE I. WEIGHT MATRIX.....	7
TABLE II. VIDEO SEQUENCES	11
TABLE III. SPEEDUPS	17
TABLE IV. QUALITY METRICS	26

