journal manuscript No.
(will be inserted by the editor)

Performance Analysis of Frame Partitioning in Parallel HEVC Encoders

H. Migallón · P. Piñol · O. López-Granado · V. Galiano · M.P. Malumbres

Received: date / Accepted: date

Abstract The new video coding standard HEVC includes two concepts that allow to partition a frame into regions that can be independently encoded and decoded. These two concepts are named "Tiles" and "Slices". In this paper we present and analyze updated and optimized parallel versions of the HEVC encoder based on tiles and slices. We have evaluated the benefits and drawbacks of both approaches in terms of computational times, rate distortion performance and video application. The results show that both approaches obtain good speed-ups, being the parallel version based on tiles the one that obtains the best trade-off between speed-up achieved (up to 9.3x) and rate distortion performance loss (1.6% BD-rate for AI mode and 2.2% for LB mode on average).

Keywords Tiles, HEVC, Video coding, Parallel algorithms, Multicore, Performance

This research was supported by the Spanish Ministry of Economy and Competitiveness under Grant TIN2015-66972-C5-4-R, co-financed by FEDER funds.(MINECO/FEDER/UE)

H. Migallón
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.
Tel.: +34-966658390
Fax: +34-966658814
E-mail: hmigallon@umh.es

P. Piñol
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

O. López-Granado
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

V. Galiano
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

M.P. Malumbres
Physics and Computer Architecture Dept. Miguel Hernández University, 03202 Elche, Spain.

1 Introduction

The high improvement in the coding performance achieved by the current video coding standard (HEVC - High Efficiency Video Coding) [1] over its predecessor (H.264/AVC - Advanced Video Coding) [2] comes at a cost: an incresase in its computational complexity. The continuous rise in the power of hardware devices is not enough to offset this drawback. This is the reason why the acceleration of HEVC video encoding/decoding has become a research topic with growing interest. Many of the proposals to accelerate HEVC processing are based on the use of parallel computing. HEVC itself includes some new features which allow high-level parallelization (at picture or subpicture level), like Wavefront Parallel Processing (WPP), slices and tiles, and some new features which allow low-level parallelization (inside the encoding process), like Local Parallel Method [3] which allows parallel motion estimation. In [4] the authors use the combination of two types of hardware architectures (GPUs and CPUs). GPUs are used to parallelize the Motion Estimation algorithm used for inter-picture prediction, while CPUs add a higher level of parallelism (using WPP or a GOP-based algorithms) by means of multicore processing. Fast decoding of pre-encoded multimedia content, like digital cinema and video on demand has motivated several works like [5] and [6], focused on parallelizing the decoding side of HEVC. However, the highest computational complexity is found at the encoder side. Parallelizing the video encoding part can be useful for applications like video recording and live event streaming. There are works that examine and propose low-level parallel techniques for encoding video sequences with HEVC, like the parallelization of the motion estimation module [7] and the parallelization of the intra prediction module [8]. Our work is based in high-level parallel techniques, by using tiles and slices to take advantage of shared memory architectures. In [9], authors compare slices and tiles encoding performance in HEVC. They show their results in terms of percentage of bit rate increase/decrease, using square tile partitioning and slices with nearly the same number of CTUs than the ones inside a tile. In our study we also evaluate tiles and slices performance but we will deal with both complexity reduction (related with the encoding process) and R/D performance. In a previous work [10], authors evaluate the impact in the encoding performance of the different tile partitioning schemes. In this work we have extended the previous study to a wider set of video resolutions and we perform a comparison between tiles and slices, to see the benefits of using tiles or slices depending on the video content resolution and to analyze the parallel scalability of both approaches as the number of procceses increase.

The rest of the paper is organized as follows. In Section 2 we will present the main aspects of frame partitioning in HEVC. Section 3 will describe the parallel algorithms based on tiles and slices partitioning. In Section 4 the results of our tests will be presented and analyzed. At last, several conclusions will be drawn in Section 5.

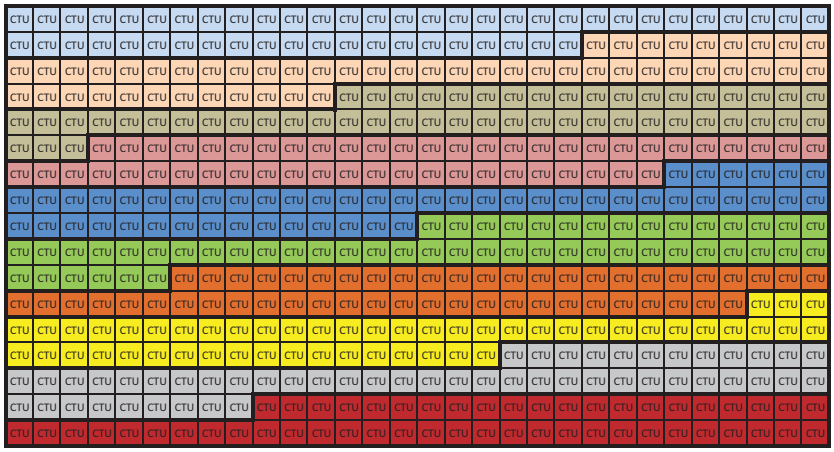


Fig. 1 Division of a full-HD frame (1920x1080 pixels) into 10 slices (of 51 CTUs each)

2 HEVC frame partitioning schemes

Tiles and slices are two elements present in the HEVC standard which have something in common: they are regions of a same frame which can be decoded (and also encoded) in an independent way. This characteristic enables them as valid parallelization approaches for video encoding and decoding processes. The independence of tiles has a drawback: the existing redundancy between nearby pieces of data which belong to different tiles cannot be exploited. This aftermath is also applicable to slices and makes coding efficiency (regarding R/D performance) decrease. Slices, moreover, are composed of a header and data. This structure may turn out to be useful to provide an encoded video sequence with error resilience features (because the loss of a single slice does not prevent the rest of the slices in the same frame from being properly decoded), but the inclusion of a header in every slice also causes a R/D performance decrease. The overhead introduced by slices' headers depends on the video resolution and on the number of slices per frame. Slices are formed by a consecutive (in raster scan order) number of Coding Tree Units (CTUs), as can be seen in Fig. 1. Tiles, however, are rectangular regions resulting from the division of a frame into one or several rows and columns (see Fig. 2). Tile division is very flexible because each one of the rows can have a different height (with an integer number of CTUs), and each one of the columns can have a different width (with an integer number of CTUs). This fact can lead to a wide variety of layouts for the division of a frame into tiles. In the example shown in Fig. 1, a full-HD (1920x1080) frame is divided into 10 slices of 51 consecutive CTUs each. In the example shown in Fig. 2, a full-HD (1920x1080) frame is divided into 10 tiles using a partitioning scheme of 5 columns with a width of 6 CTUs and 2 rows with a height of 8 and 9 CTUs, respectively.

In this work we have implemented both a tile based and a slice based parallelization of the HEVC video encoder for shared memory platforms. The

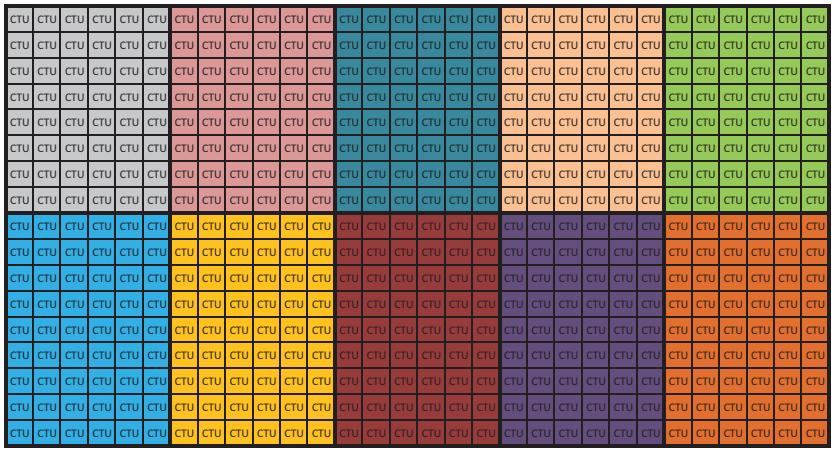


Fig. 2 Division of a full-HD frame (1920x1080 pixels) into 10 tiles (5 columns with a width of 6 CTUs; 2 rows with a height of 8 and 9 CTUs each)

encoding of each tile/slice is assigned to a different core. We have evaluated the performance of the parallel versions of the video encoder for 2, 4, 6, 8, 9, and 10 processes and compared the obtained results with those provided by the sequential version, both regarding R/D performance and computing performance. As stated before, we map the tiles/slices per frame onto the same number of processes. For a certain number of tiles per frame, we can find different frame partitions. For example, if we want to divide the frame into 9 tiles we can choose three main different tile distributions: 9x1 (9 columns by 1 row), 1x9 (1 column by 9 rows), and 3x3 (3 columns by 3 rows). In order to get the maximum parallel computing efficiency we will use the layouts that provide the most balanced load distribution, i.e., producing tiles with equal or similar number of CTUs. A balanced load distribution does not always guarantee a balanced work distribution because the resources needed to encode a single CTU may vary, but a completely unbalanced load distribution will likely bring a low parallel computing efficiency. As a measure of the load distribution balance we have calculated the maximum theoretical parallel efficiency, considering the same computational complexity for each CTU. A value of 100% denotes that all the processes will encode the same number of CTUs (and this number is exactly the average value). In Table 1, we have enumerated the different tile partitions (layouts) that we will use in our tests for different number of processes (NP) and different video resolutions. In Table 2, the data corresponding to the slice partitions is presented. In these two tables the average number of CTUs per tile/slice is shown (AvgCTU), and the number of CTUs in the biggest tile/slice of the frame partition (MaxCTU) is presented. The percentage of load balance (Bal %) is an indicator of how near every layout is from the theoretical optimal load balance.

If we divide, for example, a frame with 2560x1600 pixels into 10 tiles, and choose the 10x1 layout, then we will have 10 tiles of 100 CTUs each, which

Table 1 Tile based layouts and load balance percentage for different video resolutions

(a) 2560x1600 (40x25 CTUs)

NP	Lay.	AvgCTU	MaxCTU	Bal %
1P	1x1	1000	1000	100%
2P	1x2	500	520	96%
	2x1	500	500	100%
4P	1x4	250	280	89%
	2x2	250	260	96%
	4x1	250	250	100%
6P	1x6	166.7	200	83%
	2x3	166.7	180	93%
	3x2	166.7	182	92%
	6x1	166.7	175	95%
8P	1x8	125	160	78%
	2x4	125	140	89%
	4x2	125	130	96%
	8x1	125	125	100%
9P	1x9	111.1	120	93%
	3x3	111.1	126	88%
	9x1	111.1	125	89%
10P	1x10	100	120	83%
	2x5	100	100	100%
	5x2	100	104	96%
	10x1	100	100	100%

(b) 1920x1080 (30x17 CTUs)

NP	Lay.	AvgCTU	MaxCTU	Bal %
1P	1x1	510	510	100%
2P	1x2	255	270	94%
	2x1	255	255	100%
4P	1x4	127.5	150	85%
	2x2	127.5	135	94%
	4x1	127.5	136	94%
6P	1x6	85	90	94%
	2x3	85	90	94%
	3x2	85	90	94%
	6x1	85	85	100%
8P	1x8	63.8	90	71%
	2x4	63.8	75	85%
	4x2	63.8	72	89%
	8x1	63.8	68	94%
9P	1x9	56.7	60	94%
	3x3	56.7	60	94%
	9x1	56.7	68	83%
10P	1x10	51	60	85%
	2x5	51	60	85%
	5x2	51	54	94%
	10x1	51	51	100%

(c) 1280x720 (20x12 CTUs)

NP	Lay.	AvgCTU	MaxCTU	Bal %
1P	1x1	240	240	100%
2P	1x2	120	120	100%
	2x1	120	120	100%
4P	1x4	60	60	100%
	2x2	60	60	100%
	4x1	60	60	100%
6P	1x6	40	40	100%
	2x3	40	40	100%
	3x2	40	42	95%
	6x1	40	48	83%
8P	1x8	30	40	75%
	2x4	30	30	100%
	4x2	30	30	100%
	8x1	30	36	83%
9P	1x9	26.7	40	67%
	3x3	26.7	28	95%
	9x1	26.7	36	74%
10P	1x10	24	40	60%
	2x5	24	30	80%
	5x2	24	24	100%
	10x1	24	24	100%

(d) 832x480 (13x8 CTUs)

NP	Lay.	AvgCTU	MaxCTU	Bal %
1P	1x1	104	104	100%
2P	1x2	52	52	100%
	2x1	52	56	93%
4P	1x4	26	26	100%
	2x2	26	28	93%
	4x1	26	32	81%
6P	1x6	17.3	26	67%
	2x3	17.3	21	83%
	3x2	17.3	20	87%
	6x1	17.3	24	72%
8P	1x8	13	13	100%
	2x4	13	14	93%
	4x2	13	16	81%
	8x1	13	16	81%
9P	1x9	N/A	N/A	N/A
	3x3	11.6	15	77%
	9x1	11.6	16	72%
10P	1x10	N/A	N/A	N/A
	2x5	10.4	14	74%
	5x2	10.4	12	87%
	10x1	10.4	16	65%

means 100% of load balance (all tiles have the same number of CTUs). If we, instead, select the 1x10 layout, then we will have 5 tiles with 80 CTUs each, and 5 tiles with 120 CTUs each. The most probable scenario is that the 5 processes managing the "small" tiles remain idle waiting for the processes in charge of the "big" tiles. In this case, a maximum load balance index of 83% would be achieved. So, for a specific number of processes (NP), the selected layout may affect the parallel efficiency. Also note that a single layout can provide different load balance percentages depending on the resolution of the video sequence. For example, the 4x1 layout obtains 100% and 94% of load balance for 2560x1600 and 1920x1080 video resolutions, respectively.

3 Tile-based and slice-based parallel algorithms

Tile-based and slice-based algorithms are based on a parallel structure developed over the HEVC reference software HM-16.3 [11]. Note that, the algo-

Table 2 Slice based layouts and load balance percentage for different video resolutions

(a) 2560x1600 (40x25 CTUs)

NP	AvgCTU	MaxCTU	Bal %
1P	1000	1000	100%
2P	500	500	100%
4P	250	250	100%
6P	166.7	167	99%
8P	125	125	100%
9P	111.1	112	99%
10P	100	100	100%

(b) 1920x1080 (30x17 CTUs)

NP	AvgCTU	MaxCTU	Bal %
1P	510	510	100%
2P	255	255	100%
4P	127.5	128	99%
6P	85	85	100%
8P	63.8	64	99%
9P	56.7	57	99%
10P	51	51	100%

(c) 1280x720 (20x12 CTUs)

NP	AvgCTU	MaxCTU	Bal %
1P	240	240	100%
2P	120	120	100%
4P	60	60	100%
6P	40	40	100%
8P	30	30	100%
9P	26.7	27	99%
10P	24	24	100%

(d) 832x480 (13x8 CTUs)

NP	AvgCTU	MaxCTU	Bal %
1P	104	104	100%
2P	52	52	100%
4P	26	26	100%
6P	17.3	18	96%
8P	13	13	100%
9P	11.6	12	96%
10P	10.4	11	95%

rithms presented in [12] and [13] have been developed using an older version of the HEVC reference software (HM-10.0). In this work, we have improved them on several issues such as a) disk access time reduction, b) significant memory savings to share the original frame, the reconstructed frame, and the reference picture lists. All of these changes do not affect the R/D behavior of the parallel algorithms with respect to the sequential algorithm. Furthermore, we have included a process that automatically determines the number of CTUs assigned to each slice and the required settings in the encoding configuration parameters.

Both algorithms share a single parallel structure, but the parallel procedures are developed on different levels of the reference software. The tile-based parallel algorithm is parallelized in a lower level than the slice-based parallel algorithm. Considering both the complexity of the HEVC standard and its implementation (i.e the HM-16.3 reference software) the parallel structure is conformed on a high level of the reference software. The reference software is developed in C++, based on a lot of complex objects, but we will only enumerate the main objects in order to describe our parallel algorithms.

As we have said, Algorithm 1 describes the parallel skeleton for both parallel algorithms. Experimentally, we have confirmed that the best parallel performance is obtained when a) the original frame is read only once and shared by all threads, and b) when the bitstream is conformed only by the sequential thread. The objects referenced in Algorithm 1 are: *TEncTop* object which includes all the information about the encoding process; *outputAccessUnits* object which stores portions of the final bitstream; *TComPicYuv* object which points to original and reconstructed frames; and *ListPicYuvRec* object which points to the reference picture list(s).

Algorithm 1 Parallel skeleton of both algorithms

```
1: Sequential thread (ST) reads configuration file
2: for i = 0 to (Number of processes - 1) do
3:     ST clones TEncTop object
4:     ST clones outputAccessUnits object
5: end for
6: All threads creates TComPicYuv object
7: ST allocates original and reconstructed current frame
8: All threads creates ListPicYuvRec object
9: if Id_thread > 0 then
10:    Update TComPicYuv object to point to the original and reconstructed frames of the
       ST
11:    Update ListPicYuvRec object to point to the list of reconstructed frames of the ST
12: end if
13: while frames_encoded ≤ frames_to_be_encoded do
14:    ST reads frame
15:    OpenMP barrier
16:    Encode_frame (TEncTop objects)
17:    if GOP is finished then
18:        OpenMP barrier
19:        ST thread prepares, sorts and writes the bitstream (outputAccessUnits)
20:    end if
21: end while
```

The first 12 lines of Algorithm 1 represent its initialization stage, where one thread (ST) reads de configuration parameters and clones several objets for the available working threads. Then, every thread loads a copy of the original and reconstructed frame in their corresponding objects and prepares to start the coding process updating some objects and creating additional ones (as required by the HEVC reference software). In line 13, a loop is defined to encode every frame of the input video. In each iteration, the ST reads a new frame and then a barrier is established to synchronize all working threads just before encoding. In line 16, the *Encode_frame* function will perform the encoding using a tile-based (defined in algorithm 3) or slice-based (defined in algorithm 2) approach. Once the frame is encoded, the algorithm checks if it is the last frame of the actual Group of Pictures (GOP). If so, it waits in a barrier to synchronize all threads (assuring all threads finish its encoding task) and proceed to write the bitstream of current GOP. This operation is performed by the ST thread. If the current encoded frame is not the last frame of a GOP, the loop iterates toi encode the next frame of the video sequence.

In Algorithm 2 the maximum number of CTUs assigned to each slice is calculated taking into account that the number of slices is the same as the number of processes and also, that the computational load is balanced. Note that, in the sequential algorithm, CABAC (Context-adaptive binary arithmetic coding) uses information (contexts) from previous slices of the current frame. However, in the slice-based parallel version, all slices are encoded concurrently, so we can not use the statistical data obtained from the other slices. After initialization, every thread invokes *CompressSlice* function to encode the assigned slice following the steps found at HEVC reference software. When one

Algorithm 2 Slice-based Encode_frame function

```
1: Sequential thread (ST) computes the maximum number of CTUs for each slice
2: All threads initialize CABAC contexts
3: CompressSlice (thread_id)
4: {
5: Obtain FirstCTU and LastCTU of the slice thread_id
6: for i = FirstCTU to i ≤ LastCTU do
7:     Encode CTU[i]
8: end for
9: Reconstruct slice
10: Filter Process
11: }
12: OpenMP barrier
13: Finish the slice encoding (ST thread)
```

thread finish the encoding process, it waits for the other threads in the barrier. When all threads finish their slice coding process, then the ST thread may complete the last encoding steps.

Regarding the tile-based parallel algorithm, Algorithm 3 includes the most important steps performed inside the *CompressTile* procedure (line 3 of the Algorithm 2) following a very similar procedure than the one used in the slice-based algorithm. The main differences between them are (a) the localization and the encoding order of the CTUs belonging to one slice/tile, and (b) the way the output bistream is conformed.

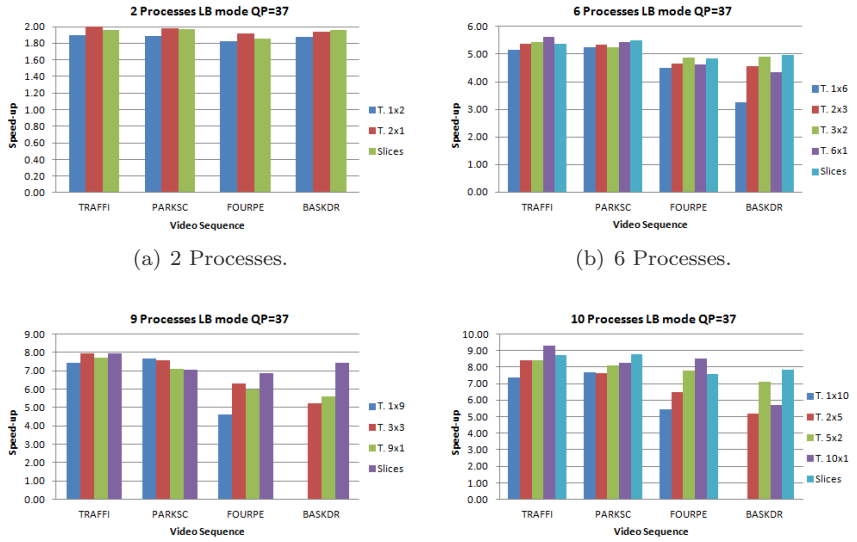
Remark that the slice-based and tile-based parallel algorithms are mutually exclusive, therefore when the tile-based algorithm is used automatically no slices are used and viceversa.

Algorithm 3 Tile-based Encode_frame function

```
1: Sequential thread (ST) creates Substreams (one for each tile)
2: CompressTile (thread_id)
3: {
4: Obtain the list of CTUs of assigned tile to substream thread_id
5: for i = 0 to i < Number_of_CTUs_in_tile do
6:     Obtain the next CTU to be encoded
7:     Encode CTU
8: end for
9: Reconstruct tile
10: Filter Process
11: }
12: OpenMP barrier
13: Finish the tile encoding (ST thread)
```

4 Numerical experiments

The proposed parallel algorithms have been tested on a shared memory platform consisting of two Intel XEON X5660 hexacores at 2.8 GHz and 12MB



(a) 2 Processes.

(b) 6 Processes.

(c) 9 Processes.

(d) 10 Processes.

Fig. 3 Speed-up evolution for Traffic, Park Scene, Four People, and Basketball Drill video sequences with different number of processes and tile and slice partitioning for QP=37 and LB mode.

cache per processor, and 48 GB of RAM. The operating system used is CentOS Linux 5.6 for x86 64 bit. The parallel environment has been managed using OpenMP [14]. The compiler used is $g++$ compiler v.4.1.2. The reference encoder software used is HM 16.3 [11].

The eight video sequences used in our experiments are *Traffic* and *People On Street* (2560x1600, 150 frames at $30MHz$), *Tennis* and *Park Scene* (1920x1080, 240 frames at $24MHz$), *Four People* and *Kristen & Sara* (1280x720, 500 frames at $50MHz$), and *Party Scene* and *Basketball Drill* (832x480, 600 frames at $60MHz$)), and we obtained results using Low-delay B (LB) and All Intra (AI) coding modes at different Quantization Parameters (QPs) (22, 27, 32, 37).

As can be seen in Fig. 3, the tile-level parallelization algorithm obtains a good parallel performance and also nice scalability results. Looking at Fig. 3, for 10 processes, there are differences in the parallel performance when we use different tile partitioning layouts. In general, tile partitioning layouts based on columns of CTUs or square tiles obtain better parallel performance. As it would be expected, this effect will depend on the video resolution. For a video resolution of 2560x1600, and a CTU size of 64x64, the number of CTUs in a frame is 40x25. If we divide the frame with the 1x10 layout we have 5 processes with 40x2 CTUs and 5 processes with 40x3 CTUs. On the other side, if we divide the frame with the 10x1 layout we have 10 processes with 4x25 CTUs. In the first case (1x10), 5 processes have to perform a 50% more work

than the other 5 proccesses. Usually the more balanced the computational load is, the better parallel performance is achieved, except for some sequences where this is not accomplished. In those exceptions, even if each process has the same number of CTUs, the computational complexity inherent to each CTU differs, producing that some processes finish before the others. As can be obserbed in Fig. 3, the slice partitioning layouts obtain similar results than the best results obtained by the tile partitioning scheme. In general, when using the tile partitioning schemes, the obtained efficiencies are slighty better when encoding high resolution video sequences with a low number of processes.

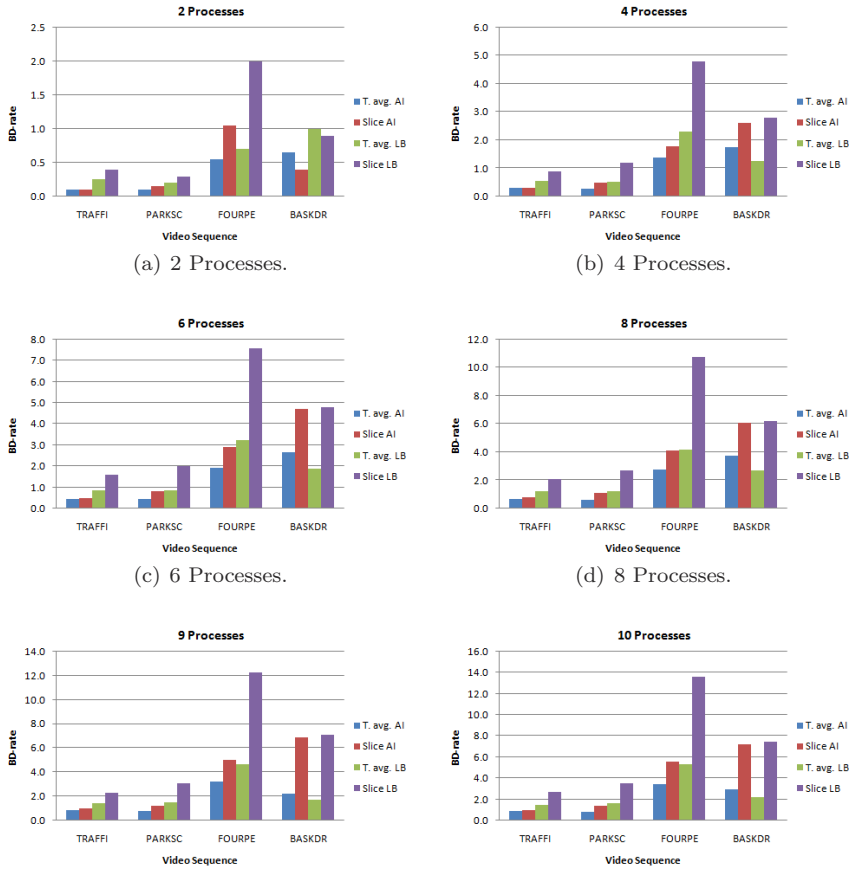
In all the experiments performed, good efficiencies are obtained for both LB and AI encoding modes. In general, the best results obtained by both partitioning schemes are quite similar, except for some particular cases. For example, for Basketball Drill video sequence, when using 9 processes, the efficiency is 69% (3x3) and 67% (9x1) using tile partitioning, and 79% when using slice partitioning. However, the maximum ideal efficiencies presented in tables 1 and 2 are 77% (3x3) and 72% (9x1) for tile partitioning and 96% for slice partitioning. Therefore, we can affirm that the tile partitioning schemes obtain results more close to the ideal efficiencies than the slice partitioning layouts.

Regarding R/D behavior, in Fig. 4 we present the BD-rate evolution for four video sequences (Traffic, Park Scene, Four People and Basketball Drill) as a function of the number of processes, considering both partitioning schemes. BD-Rate is a metric that shows the R/D performance comparison between two video encoders. It mesaures the average bitrate overhead produced by one encoder with respect to the other (reference encoder) at different compression levels. As can be seen, the BD-rate increases as the number of processes does. This is an expected behavior because both tiles and slices are independent structures and therefore, the arithmetic encoder works in an independent way on each tile/slice, and no information of previously encoded tiles/slices is available. Although not shown, we verified that square tile partitioning performs better for both LB and AI coding modes, because more information of neighbouring CTUs is available for inter and intra prediction. As stated before, every slice contains a slice-header, and these headers also contribute to obtain worse R/D performance.

Finally, after analyzing all the experimental results we obtained, we propose the use of the tile partitioning scheme using square-like partitionung layouts to develope HEVC encoder parallel versions, since the computationall efficiency is close to the idel one and the penalization in R/D performance of parallel version with respect the sequential version is reasonably low.

5 Conclusions

In this paper we present a full experimental study of both tile and slice parallelization approaches of the HEVC encoder. After the analysis of the tiles partitioning schemes, we can assess that both square tile and column tile par-



(a) 2 Processes. (b) 4 Processes.

(c) 6 Processes. (d) 8 Processes.

(e) 9 Processes. (f) 10 Processes.

Fig. 4 Average BD-Rate evolution for Traffic, Park Scene, Four People and Basketball Drill video sequences with different number of processes and tile/slice partitioning for all QPs and AI and LB mode.

titioning layouts obtain the best speed-ups (up to 9.3x for 10 processes) in the tested video sequences. Although, in some experiments, column-based tile partitioning obtain better parallel efficiency, on average, square tile partitioning layouts present a better performance in both speed-up and R/D. Besides, the increment in BD-rate is low in all cases, specially when square tile partitioning is applied, because more information of neighboring CTUs is available for the inter and intra prediction processes. The maximum BD-rate increment using square tile partitioning layouts is 5.5% for Four People video sequence in LB mode, but on average the BD-rate increment is 1.6% for AI mode and 2.2% for LB mode.

Regarding the slice-based parallel approach, similar speed-ups are obtained to the ones achieved by the tile based algorithm. However, the extra headers

required by the slice based approach have an important impact in the R/D behavior. The maximum BD-rate increment is 19.0% in LB mode using 10 processes and, on average, the BD-rate increment is 2.5% for AI mode and 4.9% for LB mode.

Therefore, after the comparison between the tile-based and the slice-based parallelizations of HEVC, we can assess that the tile-based approach is the one that best behavior obtains because it provides as good computational performance as the one obtained with slice-based schemes, but with a significant improve in R/D performance. However, we should take into account the video sequence resolution in order to perform an adequate tile partitioning layout, in such a way that the number of CTUs in each tile will be nearly the same to keep the computational load balanced.

References

1. B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan, Y.-K. Wang, and T. Wiegand, "High Efficiency Video Coding (HEVC) text specification draft 10," Joint Collaborative Team on Video Coding (JCT-VC), Geneva (Switzerland), Tech. Rep. JCTVC-L1003, January 2013.
2. ITU-T and ISO/IEC JTC 1, "Advanced video coding for generic audiovisual services," *ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC) version 16, 2012*, 2012.
3. M. Zhou, "AHG10: Configurable and CU-group level parallel merge/skip," Joint Collaborative Team on Video Coding-H0082, Tech. Rep., 2012.
4. G. Cebrián-Márquez, J. L. Hernández-Losada, J. L. Martínez, P. Cuenca, M. Tang, and J. Wen, "Accelerating HEVC using heterogeneous platforms," *The Journal of Supercomputing*, vol. 71, no. 2, pp. 613–628, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11227-014-1313-8
5. M. Alvarez-Mesa, C. Chi, B. Juurlink, V. George, and T. Schierl, "Parallel video decoding in the emerging HEVC standard," in *International Conference on Acoustics, Speech, and Signal Processing, Kyoto*, March 2012, pp. 1–17.
6. C. C. Chi, M. Alvarez-Mesa, J. Lucas, B. Juurlink, and T. Schierl, "Parallel HEVC decoding on multi- and many-core architectures," *Journal of Signal Processing Systems*, vol. 71, no. 3, pp. 247 –260, 2013.
7. Q. Yu, L. Zhao, and S. Ma, "Parallel AMVP candidate list construction for HEVC," in *VCIP'12*, 2012, pp. 1–6.
8. J. Jiang, B. Guo, W. Mo, and K. Fan, "Block-based parallel intra prediction scheme for HEVC," *Journal of Multimedia*, vol. 7, no. 4, pp. 289 –294, August 2012.
9. K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, "An overview of tiles in HEVC," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 7, no. 6, pp. 969–977, Dec 2013.
10. P. P. nol, O. López-Granado, H. Migallón, V. Galiano, and M. Malumbres, "Tile partition analysis for a parallel HEVC encoder," in *Proceedings of the 16th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE)*, 2016, pp. 989–998.
11. HEVC Reference Software, https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.3/.
12. H. Migallón, P. Piñol, O. López-Granado, and M. P. Malumbres, "Subpicture parallel approaches of HEVC video encoder," in *2014 International Conference on Computational and Mathematical Methods in Science and Engineering*, vol. 1, 2014, pp. 927–938.
13. P. Piñol, H. Migallón, O. López-Granado, and M. P. Malumbres, "Slice-based parallel approach for HEVC encoder," *The Journal of Supercomputing*, vol. 71, no. 5, pp. 1882–1892, 2015.
14. "Openmp application program interface, version 3.1," *OpenMP Architecture Review Board. http://www.openmp.org*, 2011.