Proceedings of the 14th International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2014
3–7July, 2014.

Subpicture Parallel Approaches of HEVC Video Encoder

Hector Migallón[1], Pablo Piñol[1], Otoniel López-Granado[1] and Manuel P. Malumbres[1]

[1] *Department of Physics and Computer Architecture, Miguel Hernández University*

emails: hmigallon@umh.es, pablop@umh.es, otoniel@umh.es, mels@umh.es

Abstract

The new video coding standard HEVC has nearly doubled the compression efficiency of previous standards, but this increase in efficiency comes at a certain cost: a huge increment in computing complexity. In order to reduce the time needed to encode a video sequence with HEVC, we have used two coarse-grain parallel schemes based on slices and tiles, and have measured the benefits and drawbacks of these approaches. In our tests we obtain speed ups of up to 9.2x using 10 processes, with a maximum R/D loss of 0.07dB.

Key words: slices, tiles, HEVC, video coding, parallelism

1 Introduction

Ultra high video resolution formats defined for both Digital Cinema (DC) and Ultra High Definition Television (UHDTV), like 4K and 8K, and the use of High Frame Rate (HFR) formats, like 48 fps for digital cinema and 60 fps for digital television and home recordings (120 fps in a near future), entail an exponential increase in the size of raw video sequences. In order to deal with these huge amounts of data to efficiently compress them, the Joint Collaborative Team on Video Coding (JCT-VC) has developed a new video coding standard, named High Efficiency Video Coding (HEVC) [1]. JCT-VC is formed by members of the ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG). The new standard achieves nearly a 50% of bit rate saving, if compared with the previous video coding standard H.264/AVC (Advanced Video Coding) [2]. The increase in the efficiency of this new video coding standard is bound to an increase in computational complexity. To address the increase in complexity we make use of parallelization techniques,

 ISBN: 978-84-616-9216-3

and thus take advantage of the available parallel computing architectures. In this work we use a shared-memory multi-core architecture to evaluate two parallel strategies in HEVC.

HEVC includes some new features which allow high-level parallel computing (at a picture or subpicture level), like Wavefront Parallel Processing (WPP) and tiles, and some new features which allow low-level parallel computing (inside the encoding process), like Local Parallel Method which allows parallel motion estimation. In this work we compare two high-level parallel computing approaches and, more specifically, we study how tiles behave in comparison with slices. Slices were introduced in H.264/AVC and have been maintained in HEVC. They can be used for parallelization purposes and to add error resilience to the bitstream (for resynchronization after data losses). They can also be used to do a proper packetization of data to adjust each packet to the network MTU (Maximum Transfer Unit).

Some works like [3] are focused on parallelizing the decoding side of HEVC. The research in how to accelerate video decoding is based on the need of quick decoding of pre-encoded multimedia content, like digital cinema and video on demand. In our work we concentrate in the parallelization of video encoding, which can be useful for applications like video recording and live event streaming. An application like video conference will need both encoding and decoding parallel approximations.

There are works that examine and propose low-level parallel techniques for encoding video sequences with HEVC. These techniques include examples like the parallelization of the motion estimation calculations [4] and the parallelization of the intra prediction module [5]. Our work is based in high-level parallel techniques, by using tiles and slices to take advantage of shared-memory multi-core architectures.

In [6] authors compare slices and tiles encoding performance in HEVC. They show their results in terms of percentage of bit rate increase/decrease. In our study we also evaluate slices and tiles performance but we will focus on complexity reduction related with the encoding process.

The rest of the paper is organized as follows. In Section 2 we will present the main aspects of slices and tiles in HEVC. In Section 3 we will outline the video configurations used for our experiments. In Section 4 the results of our tests will be presented and analyzed. At last, several conclusions will be drawn in Section 5.

2 Slices and tiles

As stated before, slices were introduced in the previous video coding standard, H.264/AVC, and have been maintained in HEVC. In HEVC, a frame is divided into Coding Tree Units (CTUs) in order to be encoded. CTUs can have a size of 64x64, 32x32 or 16x16 pixels. Slices are encoded regions of a frame which can be independently decoded/encoded and are composed by a certain number of consecutive CTUs in raster order (from top-left to bottom-right). Figure 1 *(left)* shows the partition of a frame into slices. The independence of

 ISBN: 978-84-616-9216-3

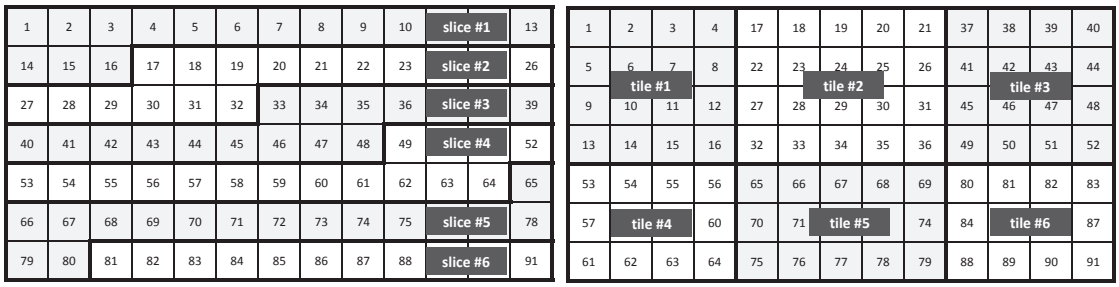


Figure 1: A frame divided into slices *(left)* and into tiles *(right)*. CTUs are numbered in encoding order.

slices makes them candidates for parallelization schemes. But the benefit of parallelization comes at a certain cost. Dividing a frame into slices will yield a larger bitstream than encoding that frame as a whole. This is due to the following reasons, (1) as one slice can be independently decoded, data from other slices cannot be used for prediction purposes (intra prediction, motion vector prediction, arithmetic encoder contexts, ...) decreasing compression efficiency, (2) every slice has a slice header and when we divide a frame into many slices, or when the slice payload is small (in relation to the header), a considerable amount of overhead is introduced, (3) the way in which CTUs are picked to form a slice (raster order) penalizes compression efficiency.

In order to provide HEVC with parallel capabilities without losing compression efficiency, JCT-VC has introduced a new element which was not present in previous standards: tiles [7][8]. Tiles are encoded regions of a frame which are independently decodable/encodable and are formed by a certain number of CTUs arranged in a rectangular way. Figure 1 *(right)* shows the partition of a frame into some tiles. Prediction with elements from outside the tile is forbidden (like in the slice case) so some compression efficiency is lost, but the overhead due to headers is directly eliminated because tiles do not have any header. CTUs inside a tile are encoded following raster order (within that tile). Tiles are numbered following raster order (see Figure 1 *(right)*).

In our tests, we have assigned the processing of each slice (tile) to a different core. We have done our tests for 1, 2, 4, 6, 8 and 10 cores, and divided each frame into the corresponding number of slices (tiles).

3 Common conditions

In [9] the author established some common conditions and software configurations in order to evaluate the suitability of new candidate designs to be incorporated to HEVC. By using these common conditions, different proposals can be compared with equity. In our experiments

 ISBN: 978-84-616-9216-3

we have used a subset of these common conditions which will be detailed in this section.

We have selected the sequence named *FourPeople* which has a resolution of 1280x720 pixels and a frame rate of 60 frames per second. We have obtained measurements for both All Intra (AI) and Lowdelay B (LB) encoding modes. In AI mode every frame is encoded as an I frame, where all its CTUs are encoded without motion estimation/compensation. Only spatial (*intra*) prediction is used and CTUs (and its divisions) can only be predicted from other CTUs inside the same slice (or tile). In LB mode the CTUs of a frame can be encoded by using spatial (*intra*) prediction or motion (*inter*) estimation/compensation. To calculate the motion estimation and compensation of a CTU, a frame needs one or more reference pictures. They are frames which have been previously encoded, and then decoded, and serve as a reference for other frames. These reference pictures are located in the Decoded Picture Buffer. In LB mode, "B" stands for Biprediction, this means that the motion estimation and compensation operations can use up to two reference frames at the same time, by interpolating their reconstructed pixels. In HEVC, frames can use a reference picture that will be displayed later than present frame. In this case, at the decoder side, the reference picture is decoded first to be available for motion compensation, but it cannot be displayed until all previous pictures (in rendering order) have been decoded. This introduces some delay in the rendering of the frames. But in LB mode, "Lowdelay" means that all reference pictures will be chosen from previous frames (in rendering order) so when an encoded frame is received and decoded, it can be immediately displayed without any delay. Encoding with AI mode is faster than with LB mode, but gets lower compression ratios. Each encoding mode can be suitable for different types of applications depending whether low bit rates are mandatory or a very quick response time is demanded without bit rate limitations. In our experiments four values for the Quantization Parameter (QP) have been selected, namely 22, 27, 32 and 37. QP is used to adjust the encoded bitstream to the level of compression desired and, therefore, the level of quality of the reconstructed video sequence. A lower value of QP produces greater bitstreams (lower compression ratio) and correspondingly higher quality reconstructed sequences.

4 Numerical experiments

A shared memory platform has been used in order to evaluate the parallel performance of our parallel proposals, moreover the impact of the parallel algorithms is discussed in terms of PSNR and bit rate. The platform used is a multicore with two Intel XEON X5660 hexacores at up to 2.8 GHz and 12MB 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 [10]. The compiler used was *g++* compiler *v*.4.1.2.

Both parallel approaches, the one based on slices and the one based on tiles have been tested using the cited video sequence *FourPeople*. In the experiments reported we have used

 ISBN: 978-84-616-9216-3

both LB and AI modes, encoding 30 and 120 frames with the four different values of QP.

Firstly we analyze the parallel slice-based proposal, in which the number of slices depends on the number of the parallel processes. As we have said, when we use slices the scan order of CTUs is raster scan order. To completely avoid all dependencies between slices, we set the parameter *LFCrossSliceBoundaryFlag* to a value of 0 in the configuration file. When the number of parallel processes increases, the number of slices increases, and therefore the slice size (number of CTUs per slice) decreases. As the size of the test video sequence is $1280x720$ pixels, and considering a CTU size of $64x64$ pixels, each frame is composed by $20x12$ CTUs. Figure 2 shows the slice structure when 4 and 8 parallel processes are used. Note that, in order to improve the parallel behavior, all slices should have an equal (or similar) number of CTUs.

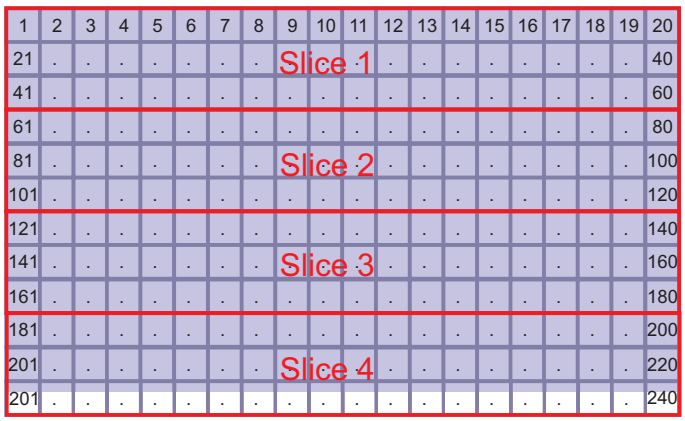
Figure 3 presents the computational times obtained by the slice-based parallel algorithm using AI and LB modes, processing 120 frames, and the corresponding speed-ups. Note that the parallel algorithm obtains speed-ups up to 8.6x for 10 processes in AI mode, and efficiency is always above 0.8. The parallel behavior of AI mode is better than LB mode, due to the motion estimation/compensation process carried out in LB mode, which produces different residual data on each slice, and therefore the time required by the entropy encoder to process residual data in each slice diverges from one slice to another. On average, the efficiency for LB mode is 0.78.

As the number of slices per frame increases, the bit rate also increases. Figure 4 shows the percentage of bit rate increase produced by the slice-based parallel algorithm. Note that, in any case, the bit rate overhead is less than 8.8%.

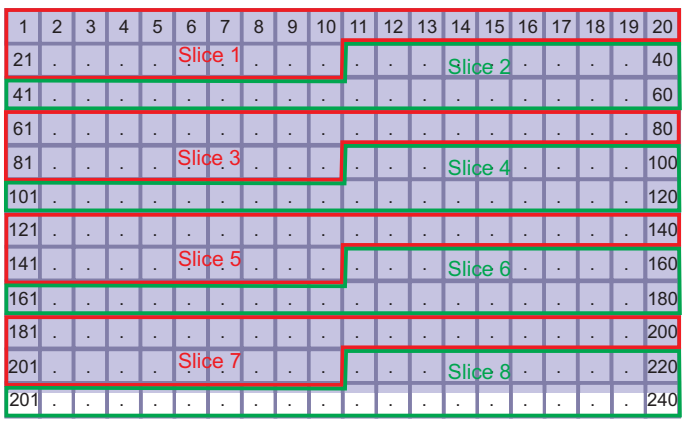
Regarding quality, in Figure 5 we present R/D information for both AI and LB modes. These figures represent the bit rate (vertical bars) and PSNR (horizontal lines) produced by the parallel algorithm using up to 10 processes and compressed for different QPs. As it can be seen, in both cases there are slight differences both in PSNR and bit rate between the sequential algorithm (with one slice per frame) and the parallel version (with two or more slices per frame), being the maximum PSNR difference less than 0.07 dB, and the bit rate increment lower than 15.2%, at high compression ratios (QP=37), for 10 processes, and encoding 120 frames. As previously said, as the number of slices increases, a greater number of headers are included in the final bitstream and a greater impact at high compression ratios is obtained.

The second parallel proposal presented is based on tiles. As we did with slices, we set the parameter *LFCrossTileBoundaryFlag* to 0 in the configuration file. The tile structure is more versatile than the slice structure, moreover, the tile structure modifies the CTU scan order. Our approach uses vertical tiles. Figure 6 shows the tile structure when 2 and 6 parallel processes are used. An horizontal size of "10 — 10" and "4 — 4 — 3 — 3 — 3 — 3 " is chosen for 2 and 6 processes, respectively. For 4, 6 and 10 processes we have chosen an horizontal size of "5 — 5 — 5 — 5 ", "3 — 3 — 3 — 3 — 2 — 2 — 2 — 2 " and " 2 —

 ISBN: 978-84-616-9216-3



(a) 4 slices per frame.



(b) 8 slices per frame.

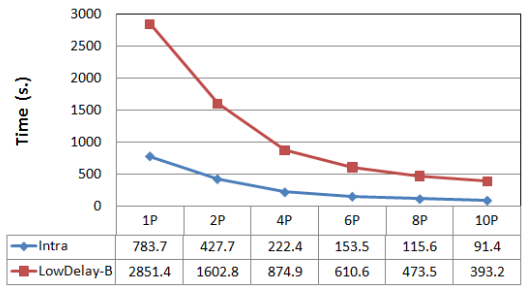
Figure 2: Slice structure and CTU scan order for the *FourPeople* video sequence.

2 — 2 — 2 — 2 — 2 — 2 — 2 — 2 — 2 ”, respectively.

The computational times and speed-ups obtained by the tile-based algorithm are presented in Figure 7, for the processing of 120 frames with AI and LB modes. Note that the tile-based parallel algorithm obtains speed-ups up to 9.2x for 10 processes in AI mode, and up to 7.8x in LB mode. As for the slice-based algorithm, the parallel behavior is slightly better when using AI mode than when using LB mode.

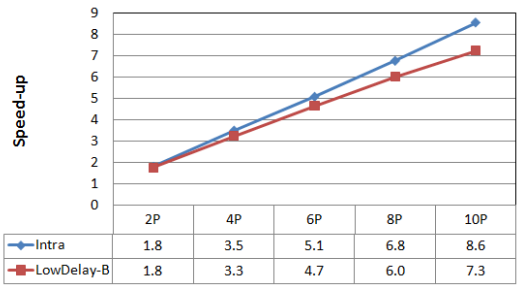
Figure 8 shows good parallel efficiency obtained by the tile-based algorithm, but in contrast to the expected behavior, the efficiency for AI mode using 10 processes is better than the efficiency of using 6 and 8 processes. By using only vertical tiles, the computational load is not balanced in all cases. In particular, when using 8 tiles, there are processes which

 ISBN: 978-84-616-9216-3



	1P	2P	4P	6P	8P	10P
Intra	783.7	427.7	222.4	153.5	115.6	91.4
LowDelay-B	2851.4	1602.8	874.9	610.6	473.5	393.2

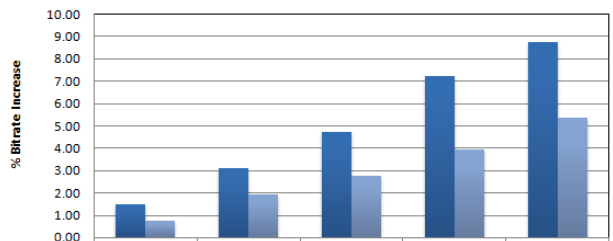
(a) Computational times.



	2P	4P	6P	8P	10P
Intra	1.8	3.5	5.1	6.8	8.6
LowDelay-B	1.8	3.3	4.7	6.0	7.3

(b) Speed-up.

Figure 3: Parallel slice-based algorithm for AI and LB modes, processing 120 frames at QP=37.



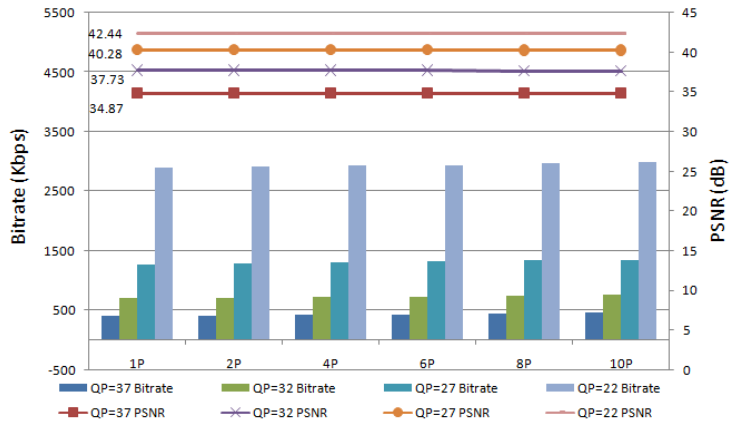
	2P	4P	6P	8P	10P
LowDelay-B	1.49	3.09	4.73	7.25	8.76
Intra	0.77	1.93	2.75	3.93	5.37

Figure 4: Bit rate increment (%) for the slice-based parallel algorithm, encoding 30 frames at QP=32.

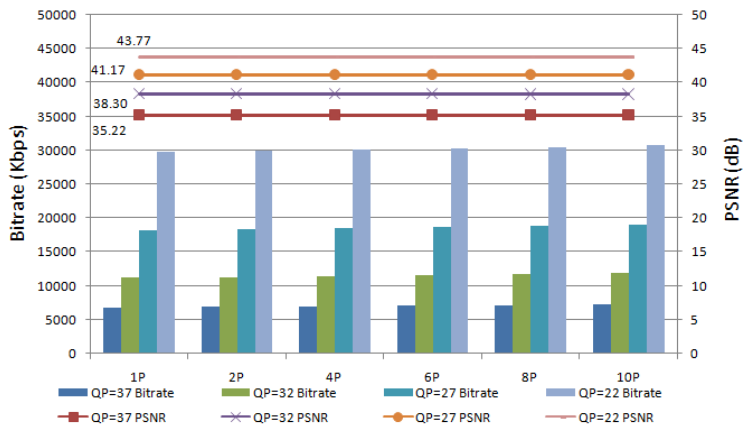
have to encode 36 CTUs (tiles 1, 2, 3 and 4) and other processes which have to encode 24 CTUs (tiles 5, 6 ,7 and 8). On the contrary, when using 10 tiles per frame, every process has to encode the same number of CTUs (24).

Figure 9 shows R/D data for the tile-based algorithm. On the one hand, we want to remark that in both figures there are slight differences both in PSNR and bit rate between the sequential and the parallel version (i.e. including vertical tiles), being the maximum PSNR difference less than 0.03 dB, and the percentage of bit rate increase lower than 6.5%, at high compression ratios (QP=37). On the other hand, as expected, the bit rate increase introduced by the tile-based algorithm is lower than the one introduced by the slice-based algorithm.

 ISBN: 978-84-616-9216-3



(a) LB mode.



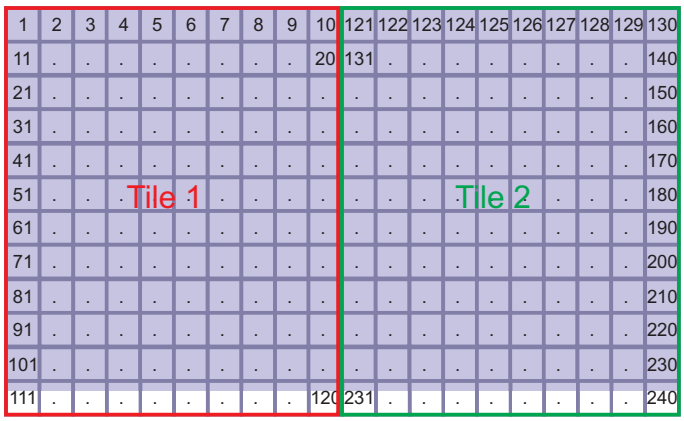
(b) AI mode.

Figure 5: R/D of both AI and LB modes for the slice-based parallel algorithm, encoding 120 frames at QP=32.

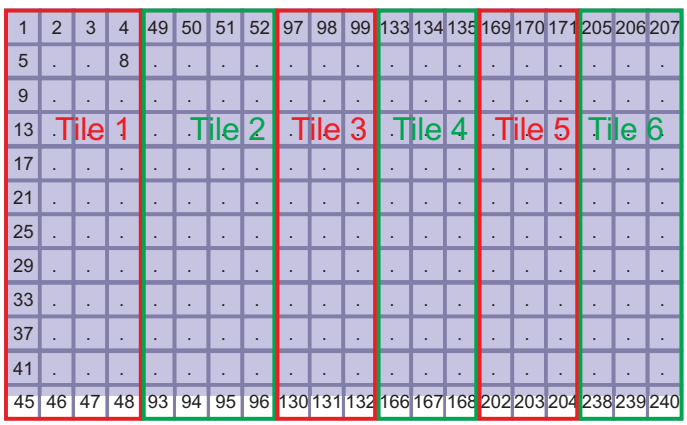
5 Conclusions

In this paper we have proposed two algorithms for HEVC video encoder parallelization. These algorithms are slice-based and tile-based approaches, which divide a frame into different number of slices or vertical tiles, depending on the number of processes used. The algorithms proposed have been analyzed for both All Intra and Lowdelay B modes. The results show that speed-ups up to 8.6x and 7.3x can be obtained for the All Intra mode and Lowdelay B modes, respectively, when the slice-based algorithm is used, and speed-ups up

 ISBN: 978-84-616-9216-3



(a) 2 vertical tiles.

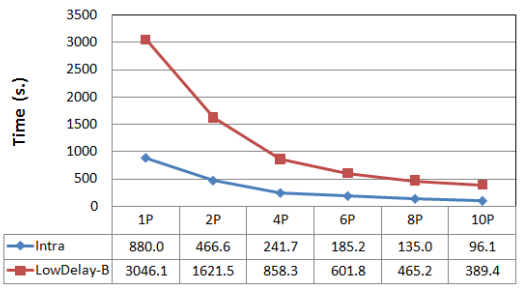


(b) 6 vertical tiles.

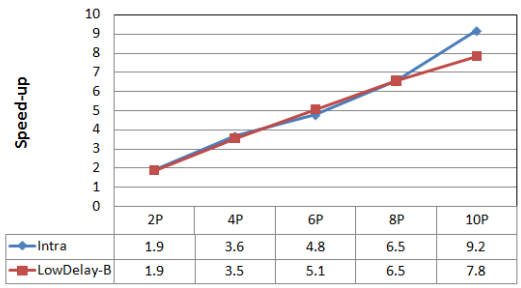
Figure 6: Tile structure and CTU scan order for the *FourPeople* video sequence.

to 9.2x and 7.8x when the tile-based algorithm is used, using up to 10 processes. Regarding encoding performance, there is an increase in the bit rate in both coding modes, being on average lower than 6% for the Lowdelay B mode, and lower on average than 3% for the All Intra mode when the slice-based algorithm is used, and being even lower when the tile-based algorithm is used. Also, there is a negligible loss in quality in both cases. As future work, we will explore several tile structures in order to improve the parallel performance, R/D cost and bit rate increase.

 ISBN: 978-84-616-9216-3



(a) Computational times.



(b) Speed-up.

Figure 7: Parallel tile-based algorithm for AI and LB modes, processing 120 frames at QP=32.

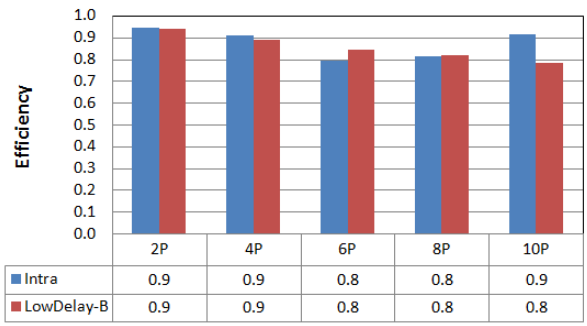


Figure 8: Efficiency for the tile-based parallel algorithm, encoding 120 frames at QP=32.

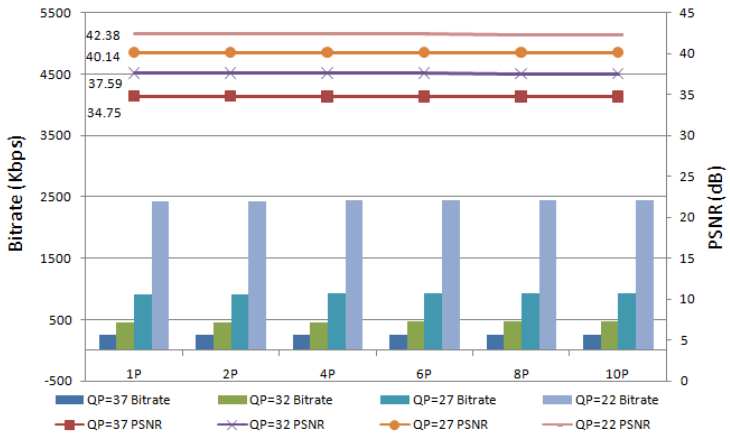
Acknowledgements

This research was supported by the Spanish Ministry of Education and Science under grant TIN2011-27543-C03-03, the Spanish Ministry of Science and Innovation under grants TIN2011-26254 and TIN2011-15734-E.

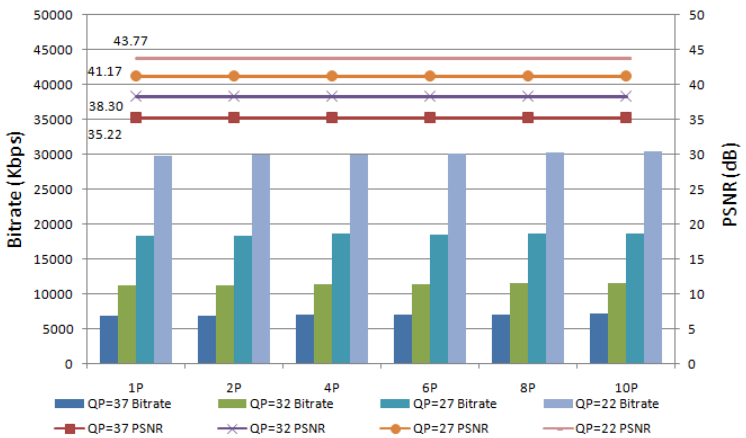
References

[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.

 ISBN: 978-84-616-9216-3



(a) LB mode.



(b) AI mode.

Figure 9: R/D of both AI and LB modes for the tile-based parallel algorithm, encoding 120 frames at QP=32.

[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.

[3] 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.

[4] Q. Yu, L. Zhao, and S. Ma, "Parallel AMVP candidate list construction for HEVC,"

 ISBN: 978-84-616-9216-3

in *VCIP'12*, 2012, pp. 1–6.

[5] 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.

[6] 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.

[7] A. Fuldseth, "Replacing slices with tiles for high level parallelism," in *Proc. JCTVC-D227, 4th Meeting of Joint Collaborative Team on Video Coding (JCT-VC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11*, 2011.

[8] A. Fuldseth, M. Horowitz, S. Xu, K. Misra, A. Segall, and M. Zhou, "Tiles for managing computational complexity of video encoding and decoding," in *Picture Coding Symposium (PCS), 2012*, May 2012, pp. 389–392.

[9] F. Bossen, "Common test conditions and software reference configurations," Joint Collaborative Team on Video Coding (JCT-VC), Geneva (Switzerland), Tech. Rep. JCTVC-L1100, January 2013.

[10] "Openmp application program interface, version 3.1," *OpenMP Architecture Review Board. http://www.openmp.org*, 2011.

 ISBN: 978-84-616-9216-3