The Journal of Supercomputing manuscript No.
(will be inserted by the editor)

Parallel strategies analysis over the HEVC encoder

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

Received: date / Accepted: date

Abstract Recently, a new video coding standard called HEVC has been developed to deal with the nowadays media market challenges, being able to reduce to the half, on average, the bit stream size produced by the former video coding standard H.264/AVC at the same video quality. However, the computing requirements to encode video improving compression efficiency have significantly been increased. In this paper, we focus on applying parallel processing techniques to HEVC encoder in order to significantly reduce the computational power requirements without disturbing the coding efficiency. So, we propose several parallelization approaches to the HEVC encoder which are well suited to multicore architectures. Our proposals use OpenMP programming paradigm working at a coarse grain level parallelization which we call GOP-based level. GOP-based approaches encode simultaneously several groups of consecutive frames. Depending on how these GOPs are conformed and distributed it is critical to obtain good parallel performance, taking also into account the level

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 grant TIN2011-26254 and Generalitat Valenciana under grant ACOMP/2013/003.

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

H. Migallón
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.

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

of coding efficiency degradation. The results show that near ideal efficiencies are obtained using up to 12 cores.

Keywords Parallel algorithms, video coding, HEVC, multicore, performance

1 Introduction

The new High Efficiency Video Coding (HEVC) standard has been recently developed by the Joint Collaborative Team on Video Coding (JCT-VC) which was established by the ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG). This new standard will replace the current H.264/AVC [1] standard in order to deal with nowadays and future multimedia market trends. 4K definition video content is a nowadays fact and 8K definition video will not take long to become a reality too. Furthermore, the new standard supports high quality color depth at 8 and 10 bits. The HEVC standard delivers the same video quality than the H.264/AVC high profile at half the bit rate.

Regarding complexity, HEVC decoder has a similar behavior to the H.264-/AVC one [2]. However, HEVC encoder is several times more complex than H.264/AVC encoder and it will be a hot research topic in years to come. At the time of developing this work, the current version of the reference software, called HEVC test model (HM), is HM 10.0 which corresponds to the HEVC text specification draft 10 [3]. A good overview of HEVC standard can be found in [4].

We can find in the literature several works about complexity analysis and parallelization strategies for the emerging HEVC standard as in [5] [6] [7]. Most of the available HEVC parallelization proposals are focused in the decoding side, looking for the most appropriate parallel optimizations at the decoder that provide real-time decoding of High-Definition (HD) and Ultra-High-Definition (UHD) video contents. The most recent and efficient decoding algorithm is presented in [8] where authors present and approach of Wave Parallel Processing (WPP) called Overlapped Wavefront (OWF). In that approach each Coding Tree Block (CTB) row is decoded by a thread (WPP) and also the decoding execution of consecutive frames is overlapped using a restricted motion vector size. Results show that an speed-up of 7.6x is achieved when using 8 cores with a negligible loss in rate/distortion (R/D).

Currently, there are few works focused on the HEVC encoder. In [9] authors propose a fine-grain parallel optimization in the motion estimation module of the HEVC encoder allowing to perform the motion vector prediction in all Prediction Units (PUs) available at the Coding Unit (CU) at the same time. In [10] authors propose a parallelization inside the intra prediction module that consists on removing data dependencies among subblocks of a CU, obtaining interesting speed-up results. In [11] authors propose a parallel algorithm at CTU (Coding Tree Unit) level for the intra coding mode. The algorithm, which has been implemented on x265 [12], launches the intra prediction of each CTU on each thread achieving a speed-up up to 5x when using 7 cores.

In this paper we will analyze the available parallel strategies in the HEVC standard and their viability over the HM reference software. Furthermore, we present a parallelization alternative for the HEVC encoder which is specially suited for low delay encoding profiles. Our proposal works at Group Of Pictures (GOP) processing level, following different parallel GOP-based strategies and analyzing the overall behavior in terms of complexity reduction and coding efficiency.

The remainder of this paper is organized as follows: in Section 2 an overview of the available profiles in HEVC and common test conditions are presented. Section 3 provides an overview of the high-level parallelism strategies proposed in the HEVC standard. Section 4 presents the GOP-based parallel alternatives we propose for the low delay application profile, while in Section 5 a comparison between the proposed parallel alternatives is presented. Finally, in Section 6 some conclusions and future work are discussed.

2 HEVC profiles

In [13] the JCT-VC defines the common test conditions and software reference configurations to be used for HEVC experiments. In that paper it can be found a series of settings in order to evaluate HEVC video codec and to compare the different contributions made to it.

A total of 24 video sequences are specified, arranged in 6 classes. Also the Quantization Parameter (QP) and the set of configuration files for the encoding process are detailed. Using these common conditions, makes easier to perform comparisons between innovative proposals. JCT-VC also provides a spreadsheet to calculate Rate-Distortion (RD) curves and the percentage of gain in bit rate, by using Bjontegaard-Delta (BD) measurements [14].

Classes from A to E include natural video sequences at diverse frame sizes. Class F comprises sequences than contain synthetic video in part of them or in its whole. Two of the sequences in class A have a bit depth of 10 bits and the rest of the sequences have a bit depth of 8 bits. The frame rate of the sequences ranges from 20 to 60 fps.

Configuration files are provided within reference software package [15]. There are 8 different test conditions which are a combination of 2 bit depths: Main (8 bits) and Main10 (10 bits) with 4 coding modes: All Intra (AI), Random Access (RA), Low-Delay B (LB), and Low-Delay P (LP).

In All Intra mode every frame is coded as an I-frame i.e. it is coded without any motion estimation/compensation. So each frame is independent from the other frames in the sequence. This mode gets lower compression rates (compared to the other 3 modes) because P-frames and B-frames can usually obtain better compression rates than I-frames at the same quality level. On the other hand, the coding process for All Intra mode is faster than for the other 3 modes because no time is wasted in motion estimation. Every frame is coded in rendering order. Applications that require a fast encoding process and are

not concerned about limited bandwidth or storage capacity, fit perfectly in this coding mode.

Random Access mode combines I-frames and B-frames in GOPs of 8 frames. A B-frame is a frame that uses motion estimation/compensation in order to achieve good compression rates. Each block of a B-frame can use up to 2 reference frames, so in the coding process 2 lists of reference pictures are maintained. Reference frames can be located earlier or later than the frame we are currently coding. In this mode, coding (and decoding) order is not the same as rendering order. To allow navigating along the coded sequence (pointing to a certain moment) or to allow functions like fast forward, an I-frame is inserted periodically. Depending on the frame rate of each sequence the intra refresh period varies. The intra period is a multiple of 8 (the size of the GOP) which inserts an I-frame approximately every second. Applications that do not have time constraints (when coding a video sequence) and need features like the aforementioned fast forward, are the target applications of this coding mode.

Low-Delay modes (LP and LB) code each frame in rendering order. First an I-frame is inserted in the coded bit stream and then only P-frames (or B-frames) are used for the rest of the complete video sequence. Here 'P' stands for Predictive and 'B' stands for Bipredictive. This means that a P-frame uses only one previously encoded and decoded frame as reference for motion estimation and compensation and a B-frame uses a pair of frames to perform these tasks. In LB mode this pair of frames is always selected from previous frames (in rendering order). On the contrary, RA mode uses past and future frames as reference for B-frames and this introduces a delay because you have to wait for future pictures to be available in order to encode/decode the present frame.In LP and LB modes GOP size is 4. This two modes achieve better compression performance than AI mode and do not suffer from the delay introduced by RA mode. Applications like video-conference which have bandwidth and time constraints can benefit from low delay modes.

3 HEVC high-level parallelism strategies

High-level parallel strategies may be classified in a hierarchical scheme depending on the desired parallel grain size. So, we define from coarser to finer grain parallelism levels: GOP, tile, slice, and wavefront. When designing an HEVC parallel version we first need to analyze the available hardware where the parallel encoder will run, in order to determine which parallelism levels are the most appropriate.

The coarsest parallelization level, GOP-based, is based on breaking the whole video sequence in GOPs in such a way that the processing of each GOP is completely independent from the other GOPs. In general, this approach will provide the best performance. However, depending on the way we define the GOPs structure and remove the inter-GOP dependencies, the coding performance may be affected.

Tiles are used to split a picture horizontally and vertically into multiple sub pictures. By using tiles, prediction dependencies are broken just at tile boundaries. Consecutive tiles are represented in raster scan order. The scan order of Coding Tree Blocks (CTBs) remains a raster scan. When splitting a picture horizontally, tiles may be used to reduce line buffer sizes in an encoder, as it operates on regions which are narrower than a full picture. Tiles also allow the composition of a picture from multiple rectangular sources that are encoded independently.

Slices follow the same concept as in H.264/AVC allowing a picture to be partitioned into groups of consecutive Coding Tree Units (CTUs) in raster scan order. Each slice can be transmitted in a different Network Abstraction Layer Unit (NALU) that may be parsed and decoded independently, except for optional inter slice filtering. There is a break in prediction dependences at slices boundaries, which causes a loss in coding efficiency. The use of slices is concerned with error resilience or with maximum transmission unit size matching but it has undoubtedly been exploited for parallel purposes in the past.

Wavefront Parallel Processing (WPP) technique splits a picture into CTU rows, where each CTU row may be processed by a different thread. Dependences between rows are maintained except for the CABAC [16] context state, which is reinitialized at the beginning of each CTU row. To improve the compression efficiency, rather than performing a normal CABAC reinitialization, the context state is inherited from the second CTU of the previous row.

All high-level parallelization techniques become more useful with image sizes growing beyond HD for both encoder and decoder. At small frame sizes where real-time decoding in a single-threaded manner is possible, the overhead associated with parallelization removes any meaningful benefit. For large frame sizes it might be useful to enforce a minimum number of picture partitions to guarantee a minimum level of parallelism for the decoder.

Current HM reference software does not directly support most of the high-level parallelism approaches mainly due to its implementation design. In the next section we will present several GOP-based parallelization approaches that may be implemented in cluster-based or multicore-based hardware architectures.

4 Parallel algorithms

We have parallelized the HEVC reference software for LB and AI modes. This two modes are useful for applications that have time constraints, so we think they can benefit from parallelization strategies. Obviously, this work can be easily extended to use LP mode, but this is not true for RA mode due to the way it uses reference frames. In particular, RA mode uses both past and future frames as reference pictures so dependencies between frames are tighter than in the two evaluated modes.

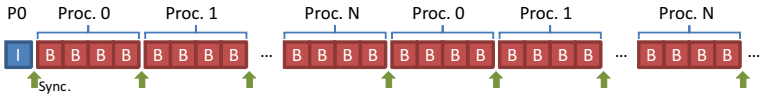


Fig. 1 Option I parallel distribution.

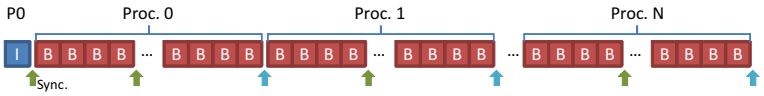


Fig. 2 Option II parallel distribution.

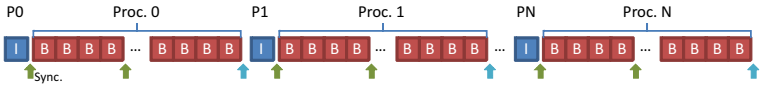


Fig. 3 Option III parallel distribution.

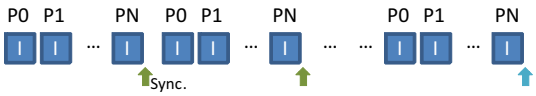


Fig. 4 Option IV parallel distribution.

The GOP-based parallel algorithms developed are suitable to run on shared memory platforms, distributed memory platforms and hybrid platforms. First of all, note that in AI mode the GOP size is 1, and, moreover, each frame is computed with no reference frames, therefore the GOP-based algorithm is inherently parallel, with the exception of the constraint of synchronization processes in order to generate the correct bit stream. In LB mode the GOP size used is 4, however this value could be changed. Note that the GOP size is the minimum number of adjacent frames allocated to each process. As we have said, we have developed synchronous algorithms where the synchronization processes are performed after the GOP computations. At these synchronization points each process writes data in the bit stream, obviously in an orderly way. In this work we describe and analyze the four parallel approaches that we have developed. The first three options are applied to LB mode, and the last one is applied to AI mode, namely:

- Option I: (LB) in this option we sequentially assign each GOP to one process in the parallel execution, so processes will encode isolated GOPs.
- Option II: (LB) in this approach we divide the sequence in as many parts as the number of parallel processes, so that each process will encode a block of adjacent GOPs.
- Option III: (LB) similar to Option II, except that each process begins the coding by inserting an I-frame.
- Option IV: (AI) similar to Option I where each GOP is sequentially assigned to one process, but here a GOP consists of only one I-frame.

Figure 1 shows the parallel distribution performed when Option I is used. As we have said the synchronization processes are located after GOP compu-

tations. The root process (Proc. 0 or P0) encodes the first frame as an I-frame and then sends it to the rest of threads so they can use it as a reference picture. This threads are idle until the root process has computed this first frame.

Note that the I-frame encoding is very fast, therefore the parallel algorithm performance is not significantly affected. After that, all processes encode a GOP of 4 B-frames. All processes, except the root process, encode their first B-frame without a nearby reference picture, therefore the number of bits needed to encode this first B-frame in those processes is greater than the number of bits needed to encode the first B-frame of P0.

The proposed parallel algorithms, are developed providing each process with its own working buffer, considering, on the one hand, to increase the performance of the developed algorithms, and on the other hand, to design suitable algorithms for shared memory platforms, distributed memory platforms and hybrid platforms. This fact changes the real pattern of the reference pictures used. For instance, in sequential processing the second B-frame of a GOP uses frames *-1 -2 -6 -10* as reference pictures (*-1* means the previous frame, and so on). As the GOP size is 4, frame *-2* points to the last frame of the previous GOP (the frame two positions before the current frame in the original video sequence). In parallel processing, as we assign isolated GOPs to each process, the previous GOP is not the previous adjacent GOP in the original video sequence and therefore frame *-2* will not point to the frame two positions before the current frame. If, for instance, the number of processes is 6, then the previous GOP for this process will be located in the video sequence 6 GOPs away from the current GOP. So for the second B-frame of a GOP, the reference picture *-2* will point to frame *-22 (-2-(6-1)x4=-22)* in the original video sequence. We can conclude that both parallel and sequential algorithms will produce different bit streams. We will analyze, in Section 5, the impact of this fact in terms of PSNR and bit rate.

In Figure 2 we can see a representation of the Option II parallel distribution. As in Option I, all processes, except the root process, encode their first B-frame without a nearby I-frame reference picture. The root process encodes the first I-frame. The improvement respect to the Option I algorithm is that the reference pictures are not significantly disturbed, because each process works with a large number of adjacent GOPs. In the previous example, for the second frame of the GOP, the pattern is only altered for the first three GOPs of a thread. From this point onward all reference pictures needed are available in the private working buffer of each process. In order to minimize the distortion of the reference pictures we assign the maximum number of adjacent GOPs to each process.

Figure 3 shows the parallel distribution of Option III, where the parallel structure is similar to Option II. In Option II we assign one group of adjacent GOPs per process. In Option III, moreover, we consider each group as a video sequence, and start the encoding procedure of each process computing the first frame as an I-frame. In this case, the parallel Option III and sequential executions can be exactly the same if in the sequential execution we do a slight change in the standard configuration.

Number of Processes	240 frames	480 frames
2	120	240
4	60	120
6	40	80
8	*30*	60
10	24	48
12	20	40

Table 1 *IntraPeriod* parameter to match sequential and parallel execution in Option III.

Acronym	File name	Width	Height	Frame rate	Total frames
TF	Traffic.yuv	2560	1600	30	150
BQ	BQTerrace.yuv	1920	1080	60	600
FP	FourPeople.yuv	1280	720	60	600
RH	RaceHorses.yuv	832	480	30	300
BP	BasketballPass.yuv	416	240	50	500

Table 2 Test video sequences.

In order to get the same bit stream with both the parallel and sequential algorithms we must change the *IntraPeriod* parameter according to the number of processes of the parallel execution. Table 1 shows the value of the *IntraPeriod* parameter when we compute 240 and 480 frames. Moreover the I-frame included must be an IDR (Instantaneous Decoding Refresh), so we set the *DecodingRefreshType* parameter equal to 2.

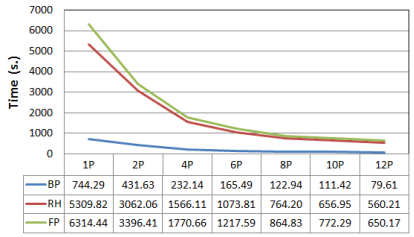
Finally, in Figure 4 the parallel distribution for Option IV is shown. Note that the parallel structure is similar to the parallel structure of Option I, but the GOP consists of one I-frame, therefore there are no differences between the parallel and the sequential execution.

5 Numerical experiments

The proposed algorithms have been tested on a shared memory platform evaluating parallel performance, PSNR and bit rate. The multicore platform used consists of 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 [17]. The compiler used was $g++$ compiler v.4.1.2.

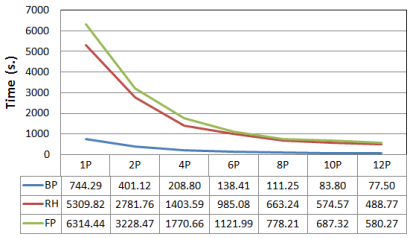
The testing video sequences used in our experiments are listed in Table 2 and we present results for them using LB mode and AI mode, encoding 120, 240 and 480 frames.

In Figure 5 we present the computation times for Option I, Option II, Option III, and Option IV parallel algorithms over FP, RH and BP test video sequences encoding 240 frames. The results show a good parallel behavior in all cases. On the other hand, all parallel algorithms using just 1 process are identical to the sequential version, therefore the sequential reference algorithm for Options I, II and III is the same sequential algorithm (the LB sequential



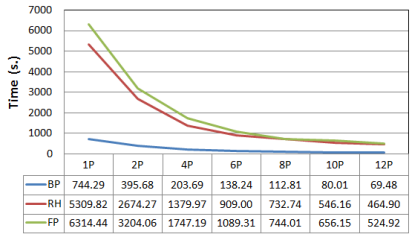
	1P	2P	4P	6P	8P	10P	12P
BP	744.29	431.63	232.14	165.49	122.94	111.42	79.61
RH	5309.82	3062.06	1566.11	1073.81	764.20	656.95	560.21
FP	6314.44	3396.41	1770.66	1217.59	864.83	772.29	650.17

(a) Option I.



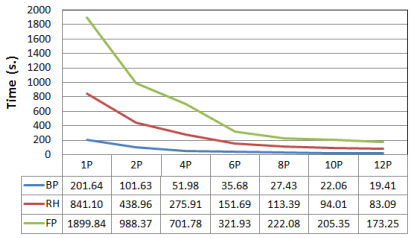
	1P	2P	4P	6P	8P	10P	12P
BP	744.29	401.12	208.80	138.41	111.25	83.80	77.50
RH	5309.82	2781.76	1403.59	985.08	663.24	574.57	488.77
FP	6314.44	3228.47	1770.66	1121.99	778.21	687.32	580.27

(b) Option II.



	1P	2P	4P	6P	8P	10P	12P
BP	744.29	395.68	203.69	138.24	112.81	80.01	69.48
RH	5309.82	2674.27	1379.97	909.00	732.74	546.16	464.90
FP	6314.44	3204.06	1747.19	1089.31	744.01	656.15	524.92

(c) Option III.



	1P	2P	4P	6P	8P	10P	12P
BP	201.64	101.63	51.98	35.68	27.43	22.06	19.41
RH	841.10	438.96	275.91	151.69	113.39	94.01	83.09
FP	1899.84	988.37	701.78	321.93	222.08	205.35	173.25

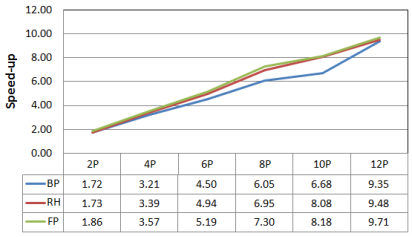
(d) Option IV.

Fig. 5 Computational times for the parallel algorithms (FP, RH and BP). 240 frames.

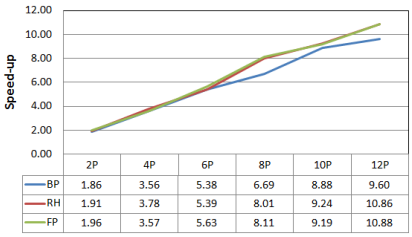
algorithm), while the reference algorithm for Option IV is the sequential AI execution.

Figure 6 shows the speed-up associated to the results shown in Figure 5. This figure confirms the good behavior of all proposed parallel algorithms, obtaining nearly ideal efficiencies in some cases. Note that the speed-up increases as the number of processes increases up to the maximum number of available cores, and the ratio of increase remains constant. In Figure 7 we compare the speed-up obtained by all proposed parallel algorithms when encoding 240 frames of the BQ test video sequence. As it can be seen, Option III obtains the best results for the LB mode being the behavior similar in the rest of tested video sequences. Also, Option IV (AI mode) obtains similar speed-ups to Option III (LB mode).

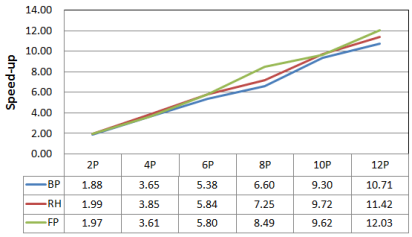
Figure 8 shows both PSNR and bitrate evolution as a function of the number of cores used for RH video sequence. As it can be seen, Option I and Option II approaches produce a bitrate increase as the number of processes increases. Option I algorithm drastically changes the structure of the reference pictures, and as a consequence it causes the large bit rate increase. Note that the first frames of each process are encoded without nearby reference frames, increasing the bit rate as the number of processes increases. Finally, the bit rate increase is greater in Option III because the initial frame is an I-frame whereas the initial frame in Option II algorithm is a B-frame. In GOP based parallel algorithms, this fact cannot be avoided without altering the quality criterion. However, in Option III approach, the increased bitrate is lower when



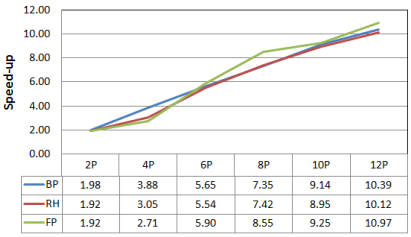
(a) Option I.



(b) Option II.



(c) Option III.



(d) Option IV.

Fig. 6 Speed-up for the parallel algorithms (FP, RH and BP). 240 frames.

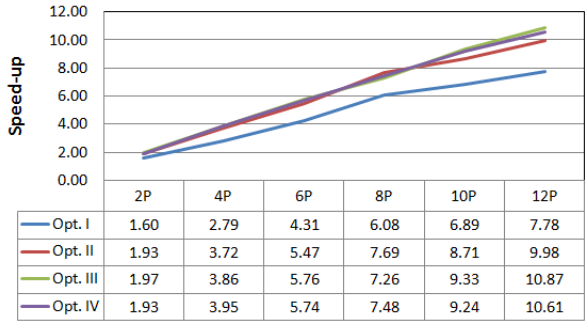


Fig. 7 Speed-up for the parallel algorithm BQ. 240 frames.

the number of total encoded frames is increased and also, the quality improves as the bitrate increases.

Finally, we have modified the low delay profile configuration in order to obtain the same PSNR and bit rate results using both parallel and sequential versions of Option III algorithm and have encoded 480 frames of BP video sequence. In Figure 9 *NON_EQUIV* curve represents the speed-up achieved when parallel and sequential algorithms obtain slightly different results, and *EQUIV* curve represents the speed-up achieved when they provide equivalent executions. We can conclude that the proposed Option III algorithm obtains a good efficiency with slightly divergences.

As we have said, the algorithm proposed in [11], achieves a speed-up of up to 5x when using 7 cores. Both the cited algorithm and our Option IV

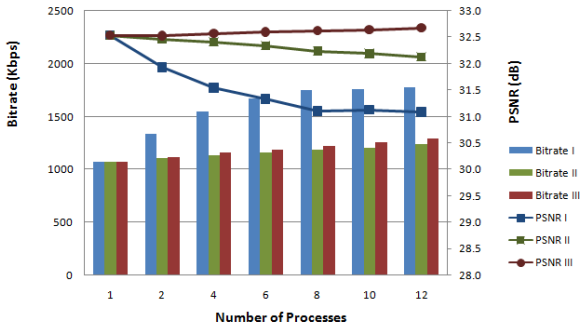
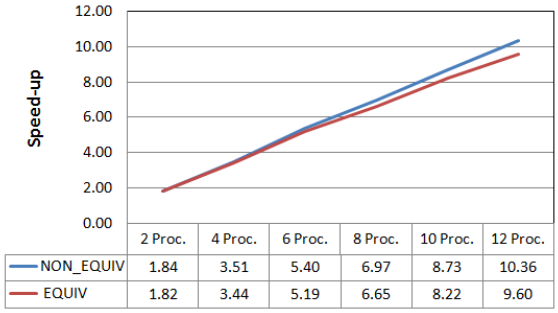


Fig. 8 Rate-Distortion for RH sequence, encoding 240 frames with different number of processes.



	2 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.	12 Proc.
NON_EQUIV	1.84	3.51	5.40	6.97	8.73	10.36
EQUIV	1.82	3.44	5.19	6.65	8.22	9.60

Fig. 9 Speed-up for Option III parallel algorithms. BP video sequence encoding 480 frames.

approaches, encode a video sequence using intra mode. Note that figures 7 and 6 show results of a speed-up above 5.5x using 6 processes (or cores). As shown, our parallel algorithm obtains better speed-ups using less processes.

6 Conclusions

In this paper we have proposed several parallel algorithms for HEVC video encoder. These algorithms are based on a coarser grain parallelization approach with the organization of video frames in GOPs and different GOP process allocation schemes. They are specially suited for multicore architectures. After implementing the algorithms in HEVC software, some experiments were performed showing interesting results as follows (a) GOP organization determines the final performance in terms of speed-up/efficiency, being the best approach Option IV (AI mode) when comparing both sequential and parallel versions, and (b) both Option II and Option III algorithms introduce a bit rate overhead as the number of processes increases being greater in Option III, and in Option III algorithm the PSNR slightly increases. In general, all proposed versions attain high parallel efficiency results, showing that GOP-based paral-

lelization approaches should be taken into account to reduce the HEVC video encoding complexity. As future work, we will explore hierarchical parallelization approaches combining GOP-based approaches with slice and wavefront parallelization levels.

References

1. 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*.
2. J. Ohm, G. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards - including high efficiency video coding (hevc)," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1669–1684, 2012.
3. B. Bross, W. Han, J. Ohm, G. Sullivan, Y.-K. Wang, and T. Wiegand, "High efficiency video coding (HEVC) text specification draft 10," *Document JCTVC-L1003 of JCT-VC, Geneva*, January 2013.
4. G. Sullivan, J. Ohm, W. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *Circuits and systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1648 –1667, December 2012.
5. F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC complexity and implementation analysis," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1685–1696, 2012.
6. 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.
7. E. Ayele and S.B.Dhok, "Review of proposed high efficiency video coding (HEVC) standard," *International Journal of Computer Applications*, vol. 59, no. 15, pp. 1–9, 2012.
8. 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. [Online]. Available: http://dx.doi.org/10.1007/s11265-012-0714-2
9. Q. Yu, L. Zhao, and S. Ma, "Parallel AMVP candidate list construction for HEVC," in *VCIP'12*, 2012, pp. 1–6.
10. 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.
11. Y. Zhao, L. Song, X. Wang, M. Chen, and J. Wang, "Efficient realization of parallel HEVC intra encoding," in *IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, July 2013, pp. 1–6.
12. x265 project, http://code.google.com/p/x265.
13. F. Bossen, "Common test conditions and software reference configurations," Joint Collaborative Team on Video Coding, Geneva, Tech. Rep. JCTVC-L1100, January 2013.
14. G. Bjontegaard, "Improvements of the BD-PSNR model," Video Coding Experts Group (VCEG), Berlin (Germany), Tech. Rep. VCEG-M33, July 2008.
15. HEVC Reference Software, https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-10.0/.
16. D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 620–636, 2003.
17. "Openmp application program interface, version 3.1," *OpenMP Architecture Review Board. http://www.openmp.org*, 2011.