## 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, tested on multicore platforms but also well suited to distributed memory 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 of coding efficiency degradation. The results show that near ideal efficiencies are obtained using up to $10$ cores. The parallel algorithms developed support all standard modes proposed by the reference software.

**Keywords:** Parallel algorithms, video coding, HEVC, multicore, performance, GOP-based algorithms.

## 1 Introduction

Recently, the new High Efficiency Video Coding (HEVC) [1] standard has been 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 [2] 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 too long to become a reality. Evenmore, the new standard supports high quality color depth at 8 and 10 bits. The new HEVC standard achieves the same video quality than the H.264/AVC high profile at approximately half the bit-rate.

Regarding complexity, HEVC decoder has a similar behavior to the H.264/AVC one [3]. 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.

Several works about complexity analysis and parallelization strategies for the emerging HEVC standard can be found in the literature [4] [5] [6]. Most of the 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.

At the moment, there are only a few works focused on the HEVC encoder. In [7] 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 [8] 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 with a negligible loss in coding performance.

In this paper we will analyze the available parallel strategies in the HEVC standard and their viability over the reference software, called the HEVC test model (HM). We will focus on applying parallel processing techniques to HEVC encoder in order to significantly reduce the computational power requirements without disturbing the coding efficiency. Our proposals use OpenMP programming paradigm working at a coarse grain parallelization level which we call GOP-based level. GOP-based approaches encode simultaneously several Group Of Pictures (GOP). Depending on how these GOPs are conformed and distributed it is critical to obtain good parallel performance, taking also into account the level of coding efficiency degradation.

After implementing the algorithms in HEVC reference software for all supported modes (All Intra (AI), Random Access (RA), Low-Delay B (LB), and Low-Delay P (LP)), some experiments were performed showing interesting results. In general, all proposed versions attain high parallel efficiency results, showing that GOP-based parallelization approaches should be taken into account to reduce the HEVC video encoding complexity and achieving near ideal efficiencies with speed-ups up to $10x$ when using up to $10$ cores, proving the high scalabitity of our parallelization proposals.

The remainder of this paper is organized as follows: In Section 2 an overview of the available profiles in HEVC and common condition test 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 proposed, 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 [9] 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.

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 [10]. 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. 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. The GOP (Group Of Pictures) size used is 8. Reference frames can be located earlier or later than the frame we are currently coding. So, in this mode, coding (and decoding) order is not the same as rendering order. So as 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. We have a value of 16 for 20 fps, 24 for 24 fps, 32 for 30 fps, 48 for 50 fps, and 64 for 60 fps. 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 sequence, being the GOP size equal to $4$. All the reference pictures are located earlier than the current frame. These two modes achieve better compression performance than AI mode and do not suffer from the delay that RA mode introduces. 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. This classification should carefully be applied taking into account the available parallel hardware resources in order to perform the most adequate and efficient implementation. So, we define from coarser to finer grain parallelism levels: GOP, tile, slice, and wavefront. When designing a HEVC parallel version we first 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 can obtain good parallel efficiency on both shared memory and distributed memory platforms. 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 narrower than a full picture. Tiles also permit 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 for transmission in a separate network adaptation layer unit 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 more concerned with error resilience or maximum transmission unit size matching than a parallel coding technique, although it has undoubtedly been exploited for this purpose in the past.

Wavefronts split 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 [11] 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.

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 GOP-based parallel algorithms

The HEVC reference software proposes four working modes, based on I-frames, B-frames and P-frames, on the other hand these standard modes are also based on the coding structure, i.e. the intra period, the GOP size and the reference pictures structure. In our work the version $10.0$ of the HEVC reference software has been used. The experiments reported were performed using the Main profile (i.e. bit depth of 8 bits).

The parallel algorithms designed are based on the GOP structure. We have developed several strategies, described below, but in all of them one GOP is assigned to one process, the synchronization processes are located after the GOP encoding, and the reference pictures structure is not shared. The main goals of these mild restrictions are, on the one hand, to fill the bitstream as the information is available, and on the other hand, to be able to extend the work to distributed memory platforms without drastically increasing the amount of information that should be transmitted.

Note that, the AI mode differs from the rest of modes on both the GOP size (equal to $1$ for the AI mode and greater than $1$ for the rest of modes) and in that all frames are computed as I-frames, this means that no reference frames are used in the AI mode. Therefore the AI parallel algorithm does not disturb the behavior of the sequential algorithm respect to quality and bit-rate.

We have developed five parallel approaches, all of them support the LB, LP and RA modes, except Option IV that only supports the AI mode. As we have said in the AI mode no reference frames are used at all, therefore the reference pictures structure is not used. Thus Option I and Option IV are similar but the last one has been tuned to fit the AI mode. The first four approaches developed are:

- Option I: (LB, LP and RA) all processes encode the first I-frame and include it in the reference picture list, after that each GOP is processed by one process, so processes will encode isolated GOPs that link in the reference picture list.

- Option II: (LB, LP and RA) the video sequence is divided in as many parts as the number of parallel processes, before processing the block of adjacents GOPs each process encodes the first GOP, that is the first I-frame.

- Option III: (LB, LP and RA) in this approach the video sequence is also divided as many parts as the number of processes, similar to Option II, but in this case each process starts encoding one I-frame at the beginning of its block of adjacents GOPs.

- Option IV: (AI) as Option I each GOP is sequentially assigned to one process, but all GOPs are composed of a single I-frame.

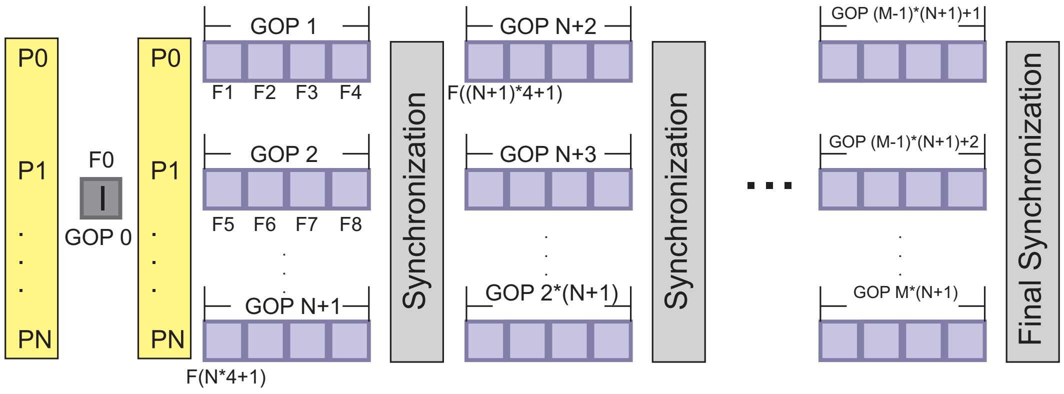


Figure 1: Option I: Parallel distribution.

Figure 1 shows the parallel distribution performed when Option I is used. As we have said, a synchronization process is located after each GOP encoding. All processes compute the first GOP, i.e. one I-frame, but obviously only the root process (P0) writes data on the bit stream. After that, each process encodes a GOP of 4 frames (or 8 for RA mode). The GOP assigned to each process depends on the rank of the parallel process, because the GOPs are assigned sequentially to each process.

As each process will have its own working buffers in order to store the reference picture list, the real pattern of the reference pictures used changes from parallel to sequential algorithm and also it changes depending on the number of processes used in the parallel algorithm. Note that each process builds its own buffer with the reference picture list numbering the frames according to the order they are stored in the local buffer. For instance, in sequential processing, the second frame of a GOP uses frames *-1 -2 -6 -10* as reference pictures (*-1* means the previous frame, and so on). Considering a GOP size equal to 4 (as in LB and LP modes), 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 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, and we will propose several parallel strategies in order to minimize this issue.

The parallel distribution performed in Option II is represented in Figure 2. In this proposal we still perform a synchronization process after each GOP. All processes compute the the first frame as an I-frame, that is the first GOP, but obviously again only the root process (P0) writes data on the bit stream. This first frame is included in the reference picture list of all processes, but in this case the reference pictures are not

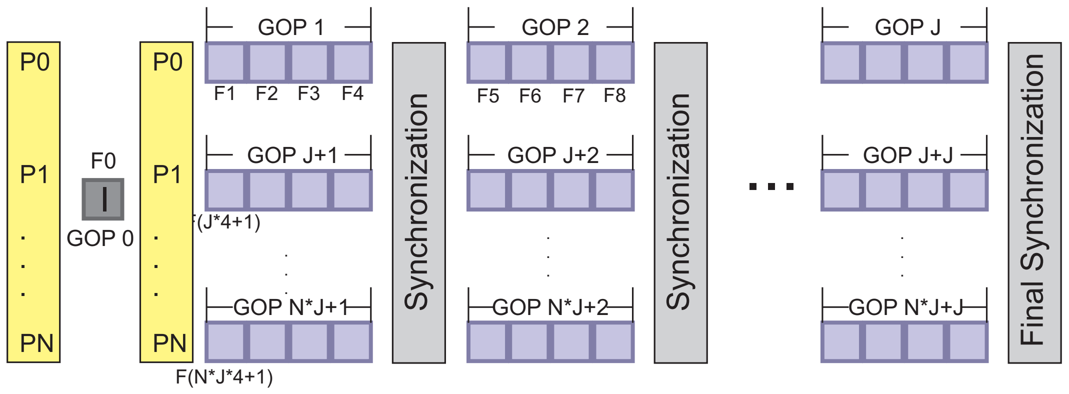


Figure 2: Option II: Parallel distribution.

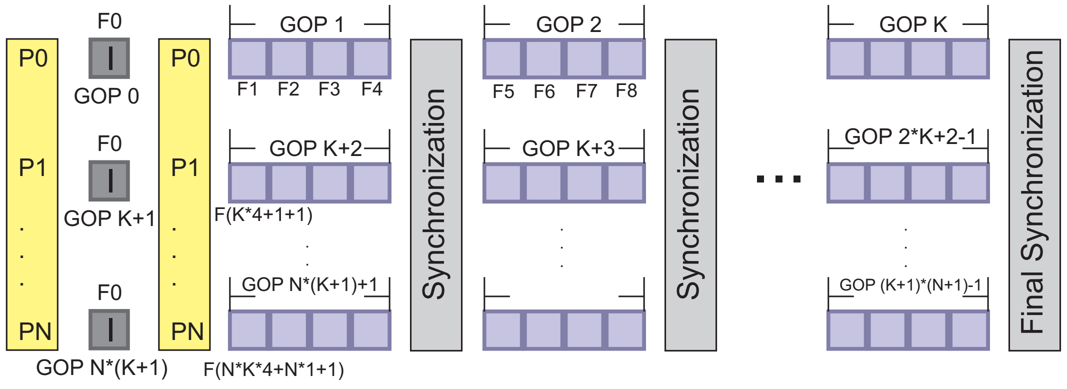


Figure 3: Option III: Parallel distribution.

significantly disturbed, because each process works with a group of adjacent GOPs. In the previous example, the pattern is only altered for the first three GOPs. From this point onward all reference pictures needed are available in the private working buffer of each process.

The parallel distribution of Option III, showed in Figure 3, presents a parallel structure similar to Option II. We still assign a block of adjacent GOPS to each process, but we change the structure of frames indicated by the LB, LP and RA modes, since in this case each process does not encode the first frame of the video sequence as an I-frame, instead of that, each process encodes the first frame of its block of adjacent GOPs as an I-frame.

As previously mentioned, the Option IV parallel strategy is the same as the Option I, where the first one is applied to LB, LP and RA modes, and the last one is applied to AI mode. Figure 4 shows the parallel distribution for Option IV. Note that a GOP always consists of one I-frame, and moreover these I-frames are IDRs, therefore there are no differences between the parallel and the sequential execution, because this mode does not use the reference picture list. Note that in this case, as in Option I and Option II, in the synchronization processes after each GOP the bit stream can be updated with data provided by all processes, while in Option III the bit stream can only

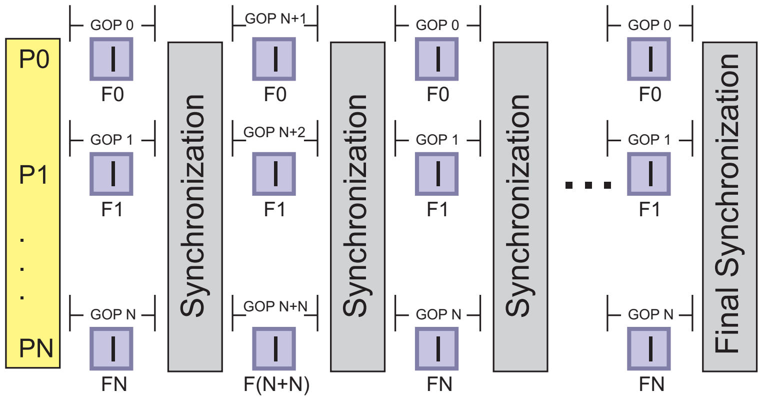


Figure 4: Option IV: Parallel distribution.

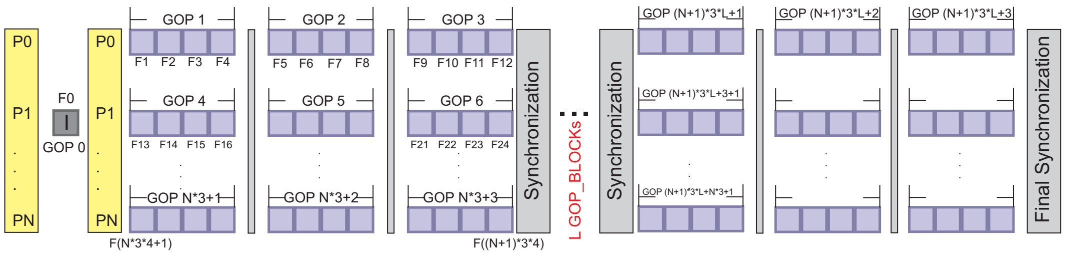
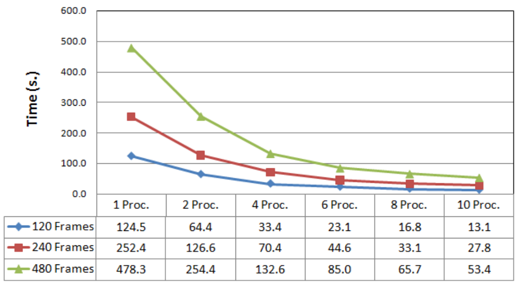


Figure 5: Option V: Parallel distribution.

be updated with data provided by the root process, and only at the end of the video encoding we can update the bit stream with the data provided by the rest of processes.

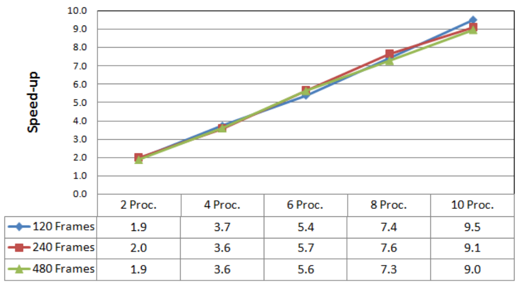
Finally we have developed another strategy, named Option V, with both main goals, a) all processes are able to write data in the bit stream more frequently than Option III, and b) to produce slight changes in the frame pattern of the standard modes LB, LP and RA, and therefore attempt to reduce PSNR and bit-rate effects introduced by the parallel algorithm. The proposed Option V strategy combines features of the other presented proposals. Firstly, all processes encode the first GOP, composed by a single I-frame, introducing it in each local buffer which stores the reference picture list, and in the same way as in Option I and Option II, only the root process updates the bit stream. After that, a fixed number of adjacent GOPs (named GOP_BLOCK) are assigned to each process depending on their parallel rank. Obviously the size of the GOP_BLOCK must be the same for all processes. Figure 3 shows the Option V parallel structure when the size of the GOP_BLOCK is equal to $3$. Note that the root process can update the bit stream after each GOP computation, while the rest of processes can update the bit stream after the GOP_BLOCK computation. We want to remark that when we increase the GOP_BLOCK size the disturbance of the reference picture list decreases.

Remark that figures 1, 2, 3 and 5 show parallel distributions considering the GOP size equal to $4$, such as in LB and LP modes, but not in RA mode in which the GOP size is equal to $8$. Regarding the Option V algorithm the GOP_BLOCK size sets



| | 1 Proc. | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|---|
| 120 Frames | 124.5 | 64.4 | 33.4 | 23.1 | 16.8 | 13.1 |
| 240 Frames | 252.4 | 126.6 | 70.4 | 44.6 | 33.1 | 27.8 |
| 480 Frames | 478.3 | 254.4 | 132.6 | 85.0 | 65.7 | 53.4 |

(a) Time (s.)



| | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|
| 120 Frames | 1.9 | 3.7 | 5.4 | 7.4 | 9.5 |
| 240 Frames | 2.0 | 3.6 | 5.7 | 7.6 | 9.1 |
| 480 Frames | 1.9 | 3.6 | 5.6 | 7.3 | 9.0 |

(b) Speed-up

Figure 6: Option IV parallel algorithms when computing $120$, $240$ and $480$ frames.

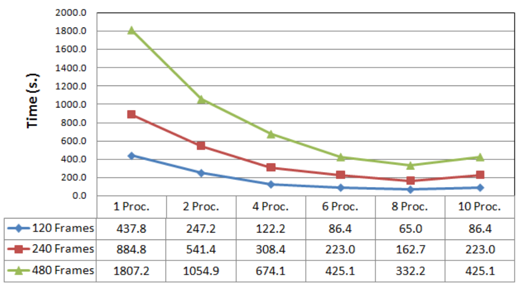
the number of GOPs, thus in the example shown in Figure 3 the number of frames included in one GOP_BLOCK is equal to $12$ (three GOPs of four frames) for LB and LP modes; while for RA mode it is equal to $24$ (three GOPs of eight frames).

## 5 Numerical experiments

In this section we analyze the parallel algorithms described in Section 4, in terms of parallel performance, PSNR and bit-rate. As the experiments reported were obtained on a shared memory platform, we have used the OpenMP [12] programming paradigm. In particular the multicore platform used is a HP Proliant SL390 G7 with two Intel Xeon X5660, each CPU with six cores at 2.8 GHz, therefore the experiments reported use up to $10$ processes. The testing video sequence used is *BQSquare_416x240_60.yuv*, and disposes of $600$ frames at $60Hz$ with a frame size equal to $416x240$ pixels. We have run the parallel algorithms encoding $120$, $240$ and $480$ frames for AI, LB and LP modes, and encoding $256$ and $512$ frames for RA mode. The GOP size for LB and LP modes is equal to $4$ while for RA mode is equal to $8$. Also, LB and LP modes compute only the first frame as I-frame, while RA mode inserts one I-frame (CDR type) every $32$ frames in our experiments. In the AI mode all frames are I-frames and one GOP consists on one I-frame. In all cases the value of quantization parameter (QP) is equal to $32$.

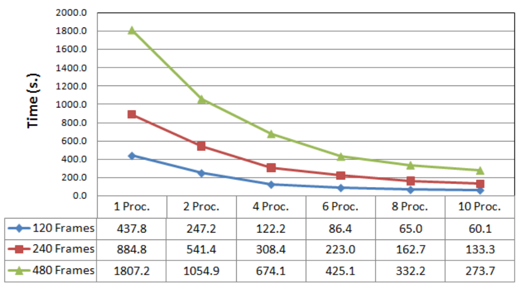
In Figure 6, we present computational results for the Option IV parallel algorithm. Figure 6(a) shows the computational times when encoding $120$, $240$ and $480$ frames, and Figure 6(b) shows the speed-up corresponding to Figure 6(a). Note that Option IV is the only algorithm related to the AI mode due to the simplicity of this mode, in which there are no dependencies between frames. The parallel algorithm developed offers good time reductions when increasing the number of processes, achieving speed-ups close to the ideal ones. Remark that in Option IV the reference sequential execution (i.e using $1$ process) obtains the same results (bit rate and PSNR) than the parallel executions, since we are using AI mode configuration.

In Figure 7 we present the computational times for Option I, Option II and Option III parallel algorithms, for the LB mode. The results show a good parallel behavior



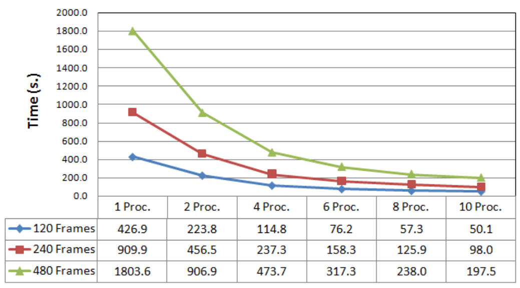
| | 1 Proc. | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|---|
| ◆ 120 Frames | 437.8 | 247.2 | 122.2 | 86.4 | 65.0 | 86.4 |
| ■ 240 Frames | 884.8 | 541.4 | 308.4 | 223.0 | 162.7 | 223.0 |
| ▲ 480 Frames | 1807.2 | 1054.9 | 674.1 | 425.1 | 332.2 | 425.1 |

(a) Option I.



| | 1 Proc. | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|---|
| ◆ 120 Frames | 437.8 | 247.2 | 122.2 | 86.4 | 65.0 | 60.1 |
| ■ 240 Frames | 884.8 | 541.4 | 308.4 | 223.0 | 162.7 | 133.3 |
| ▲ 480 Frames | 1807.2 | 1054.9 | 674.1 | 425.1 | 332.2 | 273.7 |

(b) Option II.



| | 1 Proc. | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|---|
| ◆ 120 Frames | 426.9 | 223.8 | 114.8 | 76.2 | 57.3 | 50.1 |
| ■ 240 Frames | 909.9 | 456.5 | 237.3 | 158.3 | 125.9 | 98.0 |
| ▲ 480 Frames | 1803.6 | 906.9 | 473.7 | 317.3 | 238.0 | 197.5 |

(c) Option III.

Figure 7: Computational times for the parallel algorithms for the LB mode when computing $120$, $240$ and $480$ frames.

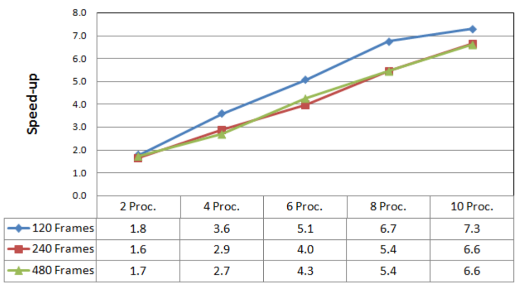
in all attempts. Note that when using just $1$ process, all proposed algorithms show similar timings than the ones obtained by the sequential version.

Figure 8 shows the speed-ups associated to the results shown in Figure 7. This figure confirms the good behavior of the proposed parallel algorithms, obtaining good speed-ups in all cases. However, the results obtained using both Option II and Option III are significantly better than those obtained by Option I. Note that the reference picture list performed in each process when using Option I does not include adjacents GOPs.

Figures 9 and 10 show the computational times and the speed-ups, respectively, when parallel algorithms encode the video sequence following the LP mode. It should be noted that the results shown are better than those obtained when the LB is used. In particular, when using the Option III parallel algorithm the speed-up results are as good as those obtained using the Option IV parallel algorithm.

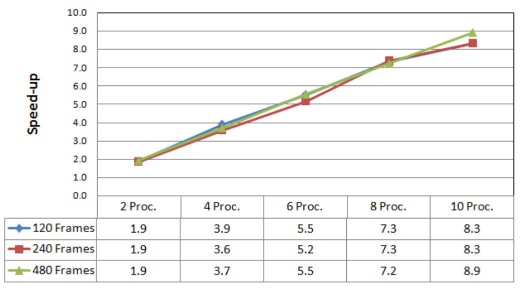
Taking into account all previous computational results, we can conclude that Option II and Option III obtain better performance than Option I, when LB or LP modes are used. In order to analyze the computational behavior when the RA mode is used we test our parallel proposals encoding $256$ and $512$ frames. Note that the RA mode works with a GOP size equal to $8$ and inserts and I-frame every $32$ frames. Similar conclusions are obtained for RA mode, looking at computational times and speed-ups shown in Figure 11.

As said in Section 4, the parallel versions do not provide the same results than the



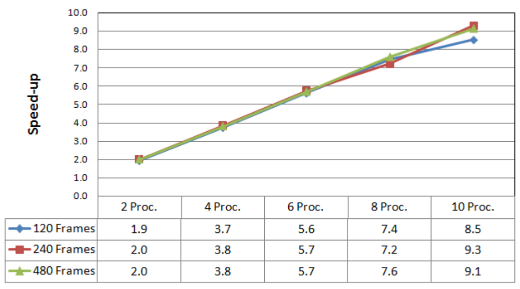
| | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|
| ◆ 120 Frames | 1.8 | 3.6 | 5.1 | 6.7 | 7.3 |
| ■ 240 Frames | 1.6 | 2.9 | 4.0 | 5.4 | 6.6 |
| ▲ 480 Frames | 1.7 | 2.7 | 4.3 | 5.4 | 6.6 |

(a) Option I.



| | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|
| ◆ 120 Frames | 1.9 | 3.9 | 5.5 | 7.3 | 8.3 |
| ■ 240 Frames | 1.9 | 3.6 | 5.2 | 7.3 | 8.3 |
| ▲ 480 Frames | 1.9 | 3.7 | 5.5 | 7.2 | 8.9 |

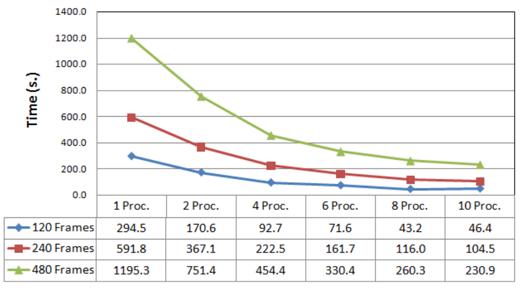
(b) Option II.



| | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|
| ◆ 120 Frames | 1.9 | 3.7 | 5.6 | 7.4 | 8.5 |
| ■ 240 Frames | 2.0 | 3.8 | 5.7 | 7.2 | 9.3 |
| ▲ 480 Frames | 2.0 | 3.8 | 5.7 | 7.6 | 9.1 |

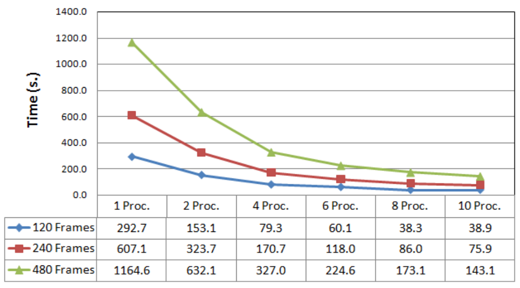
(c) Option III.

Figure 8: Speed-up for the parallel algorithms for the LB mode when computing 120, 240 and 480 frames.



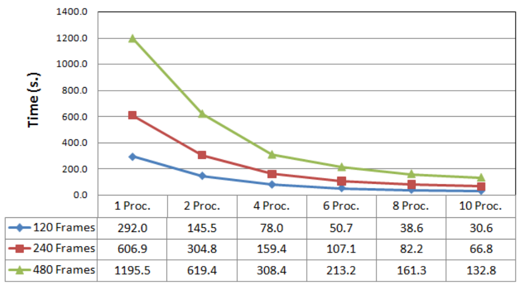
| | 1 Proc. | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|---|
| ◆ 120 Frames | 294.5 | 170.6 | 92.7 | 71.6 | 43.2 | 46.4 |
| ■ 240 Frames | 591.8 | 367.1 | 222.5 | 161.7 | 116.0 | 104.5 |
| ▲ 480 Frames | 1195.3 | 751.4 | 454.4 | 330.4 | 260.3 | 230.9 |

(a) Option I.



| | 1 Proc. | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|---|
| ◆ 120 Frames | 292.7 | 153.1 | 79.3 | 60.1 | 38.3 | 38.9 |
| ■ 240 Frames | 607.1 | 323.7 | 170.7 | 118.0 | 86.0 | 75.9 |
| ▲ 480 Frames | 1164.6 | 632.1 | 327.0 | 224.6 | 173.1 | 143.1 |

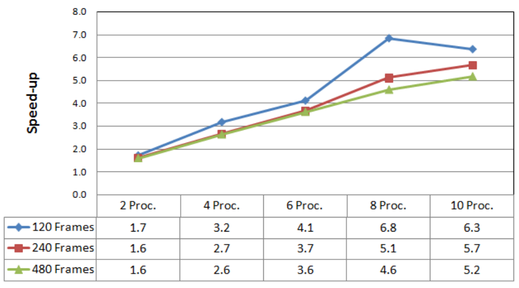
(b) Option II.



| | 1 Proc. | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|---|
| ◆ 120 Frames | 292.0 | 145.5 | 78.0 | 50.7 | 38.6 | 30.6 |
| ■ 240 Frames | 606.9 | 304.8 | 159.4 | 107.1 | 82.2 | 66.8 |
| ▲ 480 Frames | 1195.5 | 619.4 | 308.4 | 213.2 | 161.3 | 132.8 |

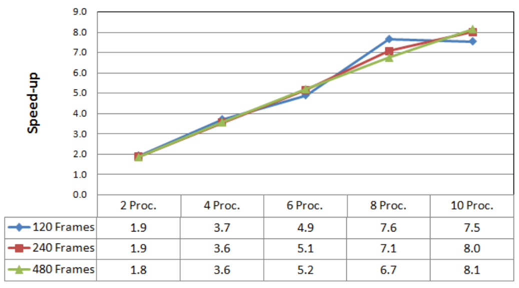
(c) Option III.

Figure 9: Computational times for the parallel algorithms for the LP mode when computing 120, 240 and 480 frames.



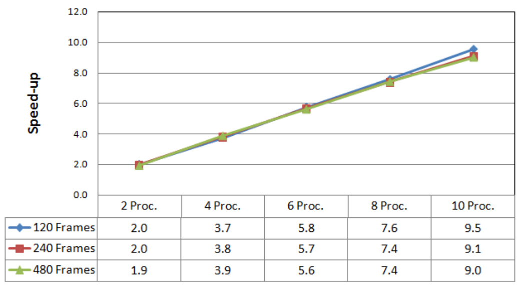
| | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|
| 120 Frames | 1.7 | 3.2 | 4.1 | 6.8 | 6.3 |
| 240 Frames | 1.6 | 2.7 | 3.7 | 5.1 | 5.7 |
| 480 Frames | 1.6 | 2.6 | 3.6 | 4.6 | 5.2 |

(a) Option I.



| | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|
| 120 Frames | 1.9 | 3.7 | 4.9 | 7.6 | 7.5 |
| 240 Frames | 1.9 | 3.6 | 5.1 | 7.1 | 8.0 |
| 480 Frames | 1.8 | 3.6 | 5.2 | 6.7 | 8.1 |

(b) Option II.



| | 2 Proc. | 4 Proc. | 6 Proc. | 8 Proc. | 10 Proc. |
|---|---|---|---|---|---|
| 120 Frames | 2.0 | 3.7 | 5.8 | 7.6 | 9.5 |
| 240 Frames | 2.0 | 3.8 | 5.7 | 7.4 | 9.1 |
| 480 Frames | 1.9 | 3.9 | 5.6 | 7.4 | 9.0 |

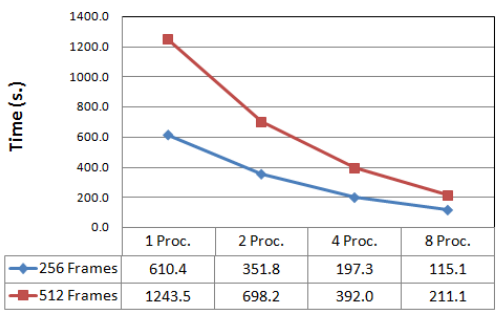
(c) Option III.

Figure 10: Speed-up for the parallel algorithms for the LP mode when computing $120$, $240$ and $480$ frames.

ones produced by the sequential algorithm. So, in Figure 12 we show how parallel versions modify the sequential version bit-rate. It is important to remark that Figure 12 shows results for options II and III, but not for Option IV, because in this case the parallel and the sequential versions exhibit the same bit-rate. Furthermore, we can observe that the bit-rate increase introduced by Option I algorithm is not acceptable. This algorithm drastically changes the structure of the reference pictures and as a consequence it causes the large bit-rate increase. In all cases the bit-rate increase becomes larger, in percentage, as the number of processes does and becomes shorter, in percentage, as the number of frames encoded does.

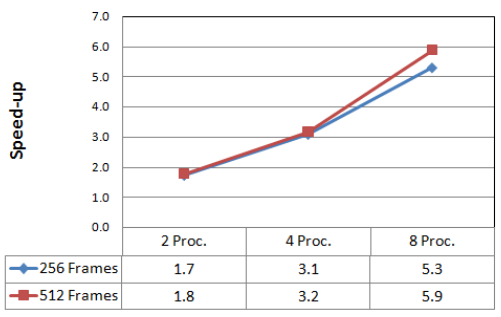
Table 1 shows the PSNR data, i.e. a quality measurement, for the parallel algorithms II and III. We can observe that the quality of the encoded video decreases when using Option II algorithm, although, in Figure 12, we have shown that the bit-rate increases. In contrast, the bit-rate increase for Option III algorithm shown in Figure 12 is compensated by a quality increase as it can be seen in Table 1.

Finally we will analyze the Option V parallel algorithm taking into account a new parameter which modifies the parallel algorithm. As we can see in Figure 5, in the Option V parallel algorithm not only one GOP or one block of GOPS is assigned to each process, but several blocks (no adjacents) of GOPs (adjacents) are assigned to each process. The number of adjacent GOPs that perform one GOP_BLOCK is the new parameter. Note that if the size of the GOP_BLOCK is equal to 1 both Option V and Option I parallel algorithms are identical, on the other hand if the GOP_BLOCK



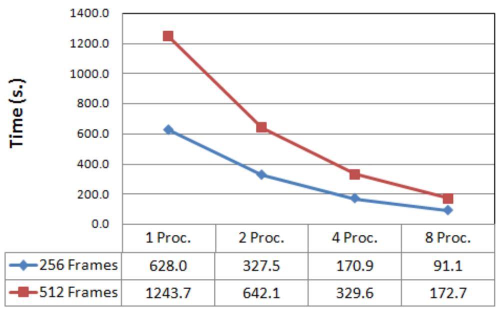
| | 1 Proc. | 2 Proc. | 4 Proc. | 8 Proc. |
|---|---|---|---|---|
| 256 Frames | 610.4 | 351.8 | 197.3 | 115.1 |
| 512 Frames | 1243.5 | 698.2 | 392.0 | 211.1 |

(a) Option I.



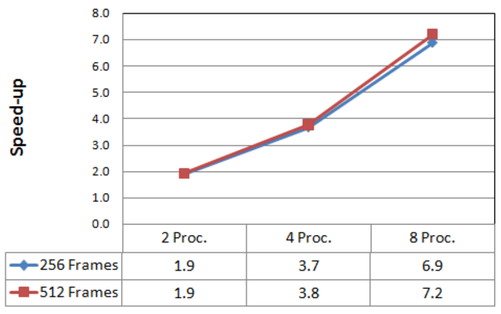
| | 2 Proc. | 4 Proc. | 8 Proc. |
|---|---|---|---|
| 256 Frames | 1.7 | 3.1 | 5.3 |
| 512 Frames | 1.8 | 3.2 | 5.9 |

(b) Option I.



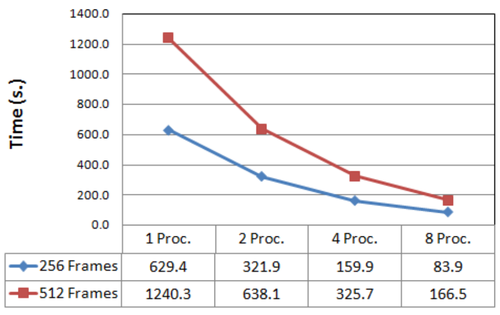
| | 1 Proc. | 2 Proc. | 4 Proc. | 8 Proc. |
|---|---|---|---|---|
| 256 Frames | 628.0 | 327.5 | 170.9 | 91.1 |
| 512 Frames | 1243.7 | 642.1 | 329.6 | 172.7 |

(c) Option II.



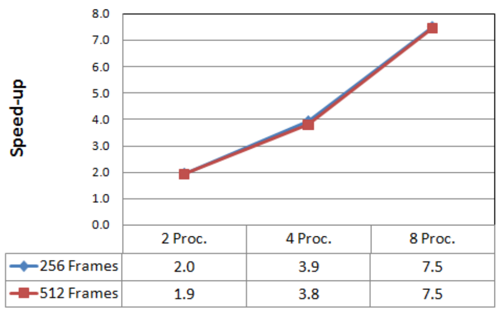
| | 2 Proc. | 4 Proc. | 8 Proc. |
|---|---|---|---|
| 256 Frames | 1.9 | 3.7 | 6.9 |
| 512 Frames | 1.9 | 3.8 | 7.2 |

(d) Option II.



| | 1 Proc. | 2 Proc. | 4 Proc. | 8 Proc. |
|---|---|---|---|---|
| 256 Frames | 629.4 | 321.9 | 159.9 | 83.9 |
| 512 Frames | 1240.3 | 638.1 | 325.7 | 166.5 |

(e) Option III.

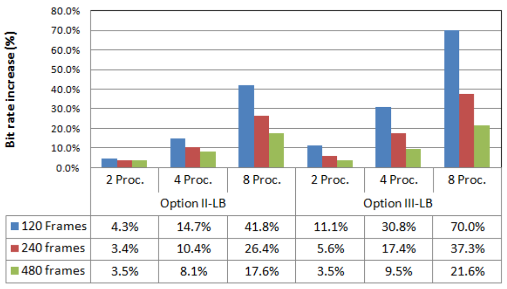


| | 2 Proc. | 4 Proc. | 8 Proc. |
|---|---|---|---|
| 256 Frames | 2.0 | 3.9 | 7.5 |
| 512 Frames | 1.9 | 3.8 | 7.5 |

(f) Option III.

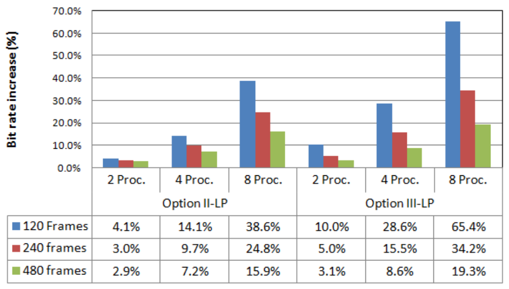
Figure 11: Computational times for the parallel algorithms for the RA mode when computing $256$ and $512$ frames.

size is equal to *Number of Frames to Encode / Number of Processes* Option V and Option II parallel algorithms are the same. Now will analyze the Option V algorithm attending to bit rate and PSNR data. The results reported were obtained encoding $256$ and $512$ frames. Note that when we encode $512$ using $8$ processors each process computes $64$ frames, i.e. $16$ GOPs when LB or LP mode is used and $8$ GOPs when RA mode is used. As expected, as we increase the GOP_BLOCK size the bit rate decreases, see Figure 13(a) for LP mode and Figure 14(a) for RA mode, and also the PSNR improves, see figures 13(b) and 14(b) for LP and RA modes respectively.



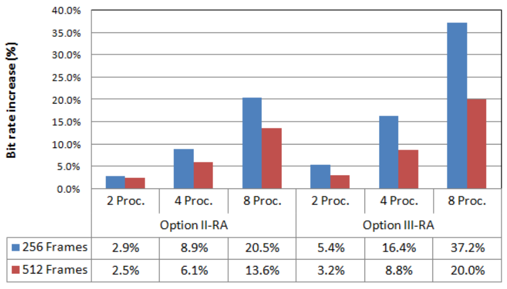
| | 2 Proc. | 4 Proc. | 8 Proc. | 2 Proc. | 4 Proc. | 8 Proc. |
|---|---|---|---|---|---|---|
| | Option II-LB | | | Option III-LB | | |
| 🟦 120 Frames | 4.3% | 14.7% | 41.8% | 11.1% | 30.8% | 70.0% |
| 🟥 240 frames | 3.4% | 10.4% | 26.4% | 5.6% | 17.4% | 37.3% |
| 🟩 480 frames | 3.5% | 8.1% | 17.6% | 3.5% | 9.5% | 21.6% |

(a) LB mode.



| | 2 Proc. | 4 Proc. | 8 Proc. | 2 Proc. | 4 Proc. | 8 Proc. |
|---|---|---|---|---|---|---|
| | Option II-LP | | | Option III-LP | | |
| 🟦 120 Frames | 4.1% | 14.1% | 38.6% | 10.0% | 28.6% | 65.4% |
| 🟥 240 frames | 3.0% | 9.7% | 24.8% | 5.0% | 15.5% | 34.2% |
| 🟩 480 frames | 2.9% | 7.2% | 15.9% | 3.1% | 8.6% | 19.3% |

(b) LP mode.



| | 2 Proc. | 4 Proc. | 8 Proc. | 2 Proc. | 4 Proc. | 8 Proc. |
|---|---|---|---|---|---|---|
| | Option II-RA | | | Option III-RA | | |
| 🟦 256 Frames | 2.9% | 8.9% | 20.5% | 5.4% | 16.4% | 37.2% |
| 🟥 512 Frames | 2.5% | 6.1% | 13.6% | 3.2% | 8.8% | 20.0% |

(c) RA mode.

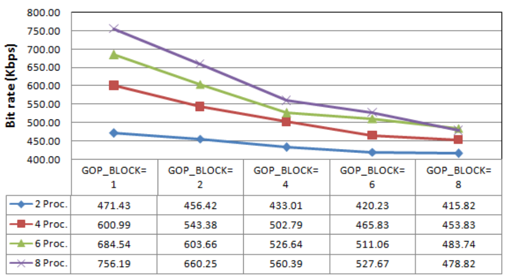
Figure 12: Percentage of bit-rate increase for the parallel algorithms.

## 6 Conclusions

In this paper we have proposed several parallel algorithms of the 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. A good parallel behavior has been shown in the experiments reported, which were obtained using a multicore platform. However the developed algorithms are able to run on distributed memory architectures since a coarser grain parallelization has been used. We have presented results using the different modes proposed by the reference software, analyzing its performance. After implementing the algorithms in the HEVC software some experiments were performed showing interesting results as (a) GOP

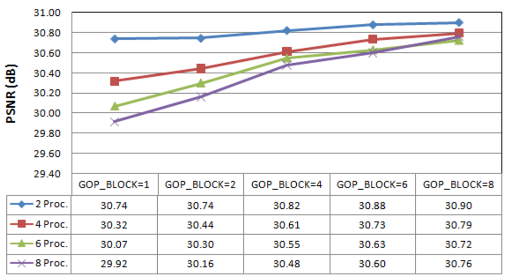
| **Algorithms** | **1 Proc** | **2 Proc** | **4 Proc** | **8 Proc** |
|---|---|---|---|---|
| **Option II-LB-480 frames** | 31.28 | 31.24 | 31.19 | 31.04 |
| **Option III-LB-480 frames** | 31.28 | 31.29 | 31.32 | 31.37 |
| **Option II-LP-480 frames** | 31.15 | 31.11 | 31.05 | 30.94 |
| **Option III-LP-480 frames** | 31.15 | 31.16 | 31.18 | 31.21 |
| **Option II-RA-512 frames** | 31.92 | 31.90 | 31.87 | 31.80 |
| **Option III-RA-512 frames** | 31.92 | 31.93 | 31.94 | 31.95 |

Table 1: Luminance PSNRs (dB) for parallel algorithms.



| | GOP_BLOCK=1 | GOP_BLOCK=2 | GOP_BLOCK=4 | GOP_BLOCK=6 | GOP_BLOCK=8 |
|---|---|---|---|---|---|
| 2 Proc. | 471.43 | 456.42 | 433.01 | 420.23 | 415.82 |
| 4 Proc. | 600.99 | 543.38 | 502.79 | 465.83 | 453.83 |
| 6 Proc. | 684.54 | 603.66 | 526.64 | 511.06 | 483.74 |
| 8 Proc. | 756.19 | 660.25 | 560.39 | 527.67 | 478.82 |

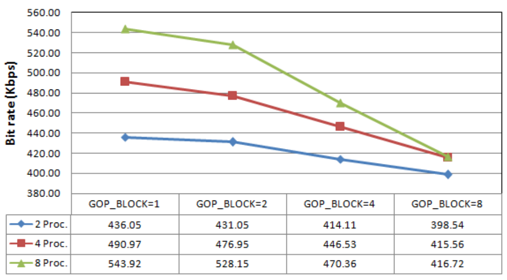
(a) Bit rate.



| | GOP_BLOCK=1 | GOP_BLOCK=2 | GOP_BLOCK=4 | GOP_BLOCK=6 | GOP_BLOCK=8 |
|---|---|---|---|---|---|
| 2 Proc. | 30.74 | 30.74 | 30.82 | 30.88 | 30.90 |
| 4 Proc. | 30.32 | 30.44 | 30.61 | 30.73 | 30.79 |
| 6 Proc. | 30.07 | 30.30 | 30.55 | 30.63 | 30.72 |
| 8 Proc. | 29.92 | 30.16 | 30.48 | 30.60 | 30.76 |

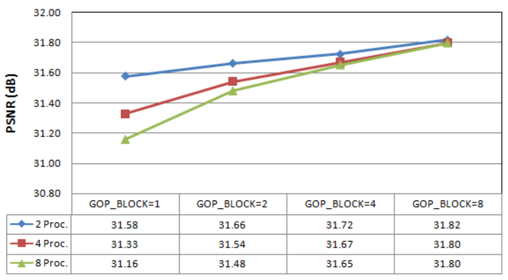
(b) PSNR.

Figure 13: Bit rate and PSNR for Option V algorithm varying the GOP_BLOCK size for LP mode.



| | GOP_BLOCK=1 | GOP_BLOCK=2 | GOP_BLOCK=4 | GOP_BLOCK=8 |
|---|---|---|---|---|
| 2 Proc. | 436.05 | 431.05 | 414.11 | 398.54 |
| 4 Proc. | 490.97 | 476.95 | 446.53 | 415.56 |
| 8 Proc. | 543.92 | 528.15 | 470.36 | 416.72 |

(a) Bit rate.



| | GOP_BLOCK=1 | GOP_BLOCK=2 | GOP_BLOCK=4 | GOP_BLOCK=8 |
|---|---|---|---|---|
| 2 Proc. | 31.58 | 31.66 | 31.72 | 31.82 |
| 4 Proc. | 31.33 | 31.54 | 31.67 | 31.80 |
| 8 Proc. | 31.16 | 31.48 | 31.65 | 31.80 |

(b) PSNR.

Figure 14: Bit rate and PSNR for Option V algorithm varying the GOP_BLOCK size for RA mode.

organization determines the final coding performance, being the best approach the Option IV (AI mode) algorithm when comparing both sequential and parallel versions in terms of speed-up/efficiency; (b) although the Option III algorithm introduces a bit-rate overhead as the number of processes increases, the overall parallel performance and the improvements in PSNR make it a good approach when LB, LP or RA coding modes are demanded; and (c) the Option V algorithm offers similar features than Option III with the improvement to be able to update the bitstream during encoding process with data obtained from all processes, not just from the root process.In general, all proposed versions attain high parallel efficiency results, showing that GOP-based parallelization 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 tile parallelization levels, which are aimed to exploit the shared memory parallelism rather than the distributed memory parallelism.

## 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. 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.

[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] 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.

[4] 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.

[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] 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.

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

[10] HEVC Reference Software, https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-10.0/.

[11] 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.

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