Proceedings of the 15th International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2015
3–7July, 2015.

MPI-based parallel strategies for HEVC encoder

Héctor Migallón[1], Vicente Galiano[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`, `vgaliano@umh.es`, `pablop@umh.es`, `otoniel@umh.es`, `mels@umh.es`

Abstract

The HEVC video coding standard launched on 2013, is able to reduce to the half, on average, the bit stream size produced by H.264/AVC encoder at the same video quality, but it requires nearly 70% more time than H.264/AVC to encode a video sequence. In this paper we propose several parallelization approaches to the HEVC encoder. Our proposals use MPI programming paradigm working at a coarse grain level parallelization, we call GOP-based level. This approach encode simultaneously several groups of consecutive frames. To obtain good parallel performance, a right GOP conformation and distribution should be applied.

Key words: Parallel algorithms, video coding, HEVC, multicore, performance

1 Introduction

The High Efficiency Video Coding (HEVC) standard [1] has been launched on January 2013 by the Joint Collaborative Team on Video Coding (JCT-VC). This new standard replaces the current H.264/AVC [2] standard in order to deal with nowadays and future multimedia market trends like 4K and 8K definition video content and high quality color depth at 10 bit. HEVC greatly improved the coding efficiency over its predecessor (H.264/AVC) by a factor of almost twice while maintaining an equivalent visual quality [3].

Concerning complexity, in [4], Bossen et al. studied the complexity aspects of HEVC encoding and decoding software. This study concludes that the encoding process is much more challenging than the decoding process, e.g., encoding one second of a 1080p60 HD (High Definition) video with the reference software encoder can take longer than one hour running in an off-the-shelf desktop computer.

We can find in the literature several works about complexity analysis and parallelization strategies for the emerging HEVC standard as in [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. In [7] and [8] the authors present a technique called Overlapped Wavefront (OWF) for the HEVC decoder which is a variant of Wavefront Parallel Processing (WPP) in which the executions over consecutive pictures are overlapped. In a multi-threaded approach of the HEVC decoder, a picture is decoded by several threads at the same time, and each thread decodes different Coding Tree Block (CTB) rows. In these works, authors claim that a single thread may continue processing the next picture when it finishes the current one, without waiting for the other threads. These variations allow a better parallel processing efficiency, reducing the overall decoding time. Recently, in [9] the authors combine Tiles, WPP with SIMD (Single-Instruction Multiple-Data instruction set extension to the x86 architecture) instructions to develop a real-time HEVC decoder.

At the present time, there are few works focused at the HEVC encoder. In [10] 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 [11] authors propose a parallelization inside the Intra prediction module that consist on removing data dependencies among subblocks of a CU, obtaining interesting speed-up results. Other proposals are focused on changes in the scanning order. For example, in [12] the authors propose a CU scanning order based on a diamond search obtaining a good scheme for massive parallel processing. Also in [13] the authors propose to change the HEVC deblocking filter processing order obtaining time savings of 37.93% over many-core architectures. In [14] authors combine a GPU-based motion estimation algorithm with two different parallelization techniques: WPP and group of pictures (GOP). This approach allows a multicore system to process multiple coding tree units (CTUs) by splitting the frame in rows or the sequence in GOPs, respectively. In either case, the motion estimation of these regions is issued to the GPU device obtaining speed-ups of up to 3.93x for 4 processes.

In this paper, we will focus on applying parallel processing techniques to the HEVC encoder in order to significantly reduce the computational power requirements without disturbing the coding efficiency. Our proposals use MPI (Message-Passing Interface) programming paradigm working at a coarse grain parallelization level called GOP-based level. GOP-based approaches encode simultaneously several GOPs and depending on how these GOPs are conformed and distributed it is critical to obtain good parallel performance.

The remainder of this paper is organized as follows, in Section 2 an overview of the available profiles and parallel strategies in HEVC are presented. Section 3 present the parallel strategies proposed for distributed memory architectures, while in Section 4 an evaluation of the proposed parallel algorithms is presented. Finally, in Section 5 some conclusions are drawn.

2 HEVC setups and parallel approaches

In HEVC, the encoding procedure is performed by dividing each video frame into small squares and processing them in raster scan order. These squares are called Coding Units (CU). CUs can be encoded by using spatial redundancy (*intra* CUs) or by using temporal redundancy (*inter* CUs). For our evaluations, we have chosen two different HEVC setups: All Intra (AI) setup and Low-Delay B (LB) setup. AI setup encodes every frame of a sequence as an I-frame. I-frames only use *intra* coding. This means that every CU is encoded exploiting spatial redundancy. Previously encoded CUs within the same frame are used to make a prediction of the current CU. This prediction is substracted from the current CU and then the residuum is encoded. In LB setup the coding structure of the sequence contains an I-frame followed by B-frames. Low-Delay means that reference frames are always selected from previous frames (in rendering order). B-frames can contain both *inter* CUs and also *intra* CUs. In B (bidirectional prediction) frames, *inter* CUs are predicted by an interpolation of two similar CUs chosen from two different reference frames. This interpolation is substracted from the current CU and then the residuum is encoded. On the one hand, B-frames are more efficient than I-frames (regarding compression). This means that, at the same level of quality, bit streams generated by using LB setup will be much smaller than bit streams generated by using AI setup (at a same level of quality). On the other hand, processing a B-frame is a heavier task than processing an I-frame. So, processing the whole video sequence with LB setup will take much longer than processing the video with AI setup. These two results are the consequence of the same process: motion estimation. Motion estimation is a hard task but gets very good efficiencies in coding terms.

In order to accelerate the encoding process of HEVC by using a parallelization scheme we can use different strategies. The coarsest parallelization level approach takes GOPs and assigns them to different processors. Other approaches introduce the parallelization scheme into sub-picture divisions and assign each division to a different processor. Three of these approaches are tile-based parallelization, slice-based parallelization and WPP. The approach selected in this work is the GOP-based parallelization scheme, which is well suited for distributed-memory architectures. The other parallelization levels mentioned (tiles, slices and WPP) are not suitable for distributed memory platforms, because they work with fragments of a frame. In the following section we will outline the parallel algorithms proposed, based on the GOP-based approach, which will be implemented and evaluated using a distributed memory architecture.

3 Distributed memory parallel strategies based on GOP structure

In this section, we propose algorithms designed to be executed on distributed memory platforms, whose parallelization is based on the GOP structure. In these strategies at least one GOP is assigned to each processor, where one GOP consists of four frames in the LB mode and one frame in the AI mode. We propose two different strategies in this work. In one strategy, the work assigned to each processor depends on both the size of the video sequence to be encoded and the number of processors to be used. In the other strategy, the coordinator process is responsible for assigning new GOP computation to each encoding process. We have developed four algorithms, named Distributed Memory GOP (DMG) algorithms:

- DMG-1G (LB mode): all processes encode the first I-frame and include it in their reference picture list. When a process has completed its work, it asks the coordinator process for the next GOP to be encoded.
- DMG-1B (LB mode): the video sequence is divided in as many parts as the number of available processors. Each processor computes its partial video sequence as an independent video sequence.
- DMG-GB (LB mode): first, all processes encode the first I-frame (as DMG-1G does) and then, when a process has completed its work, the coordinator process assigns a fixed number of contiguous GOPs (GOP_BLOCK) to it. The new GOP_BLOCK allocation depends on which process requests workload first.
- DMG-AI (AI mode): following the DMG-1G scheme, the coordinator process assigns one frame to each process that becomes idle (every frame of the sequence is encoded as an I-frame).

Figure 1 shows the parallel distribution performed in the DMG-1G algorithm. After the encoding process of the first GOP (the first I-frame) all processors request another GOP. When one processor finishes its job and becomes idle, it requests a new GOP. The coordinator process manages the GOPs sent to each process. It listens to the requests from the encoding processes and sends each new GOP to be encoded. Note that, both the coordinator process and the first encoding process are mapped on the same processor. We have observed that the computational load of the coordinator process is negligible. The parallel distribution of DMG-AI algorithm, shown in Figure 2, is similar to the parallel distribution of DMG-1G algorithm, taking into consideration that, in DMG-AI algorithm, each GOP is composed by one I-frame. One GOP computed by one process is denoted by $G_{(a\,b)}$ and $I_{G_{(a\,b)}}$ in Figures 1 and 2 respectively, being a the rank of the parallel process and

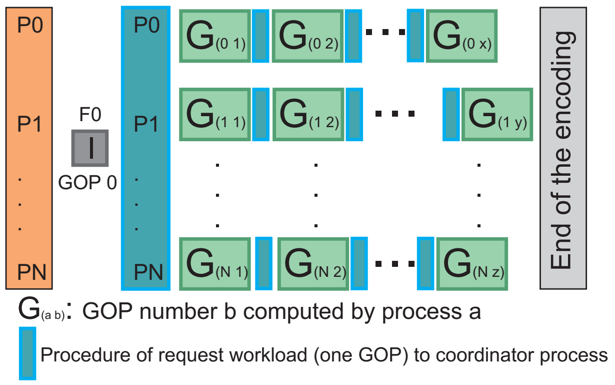

P0
P1
PN
F0
I
GOP 0
G(0 1)
G(0 2)
G(0 x)
G(1 1)
G(1 2)
G(1 y)
G(N 1)
G(N 2)
G(N z)
End of the encoding
G(a b): GOP number b computed by process a
Procedure of request workload (one GOP) to coordinator process

Figure 1: DMG-1G: Parallel distribution.

b the number of GOP computed by the process a. Also, on both figures, x, y and z denote the number of GOPs computed by processes *0*, *1* and N respectively (apart from the first I-frame). The total number of GOPs computed by each process may vary depending on the computational load assigned to each process. In contrast with DMG-1G and DMG-AI algorithms, in DMG-1B algorithm the coordinator process is not necessary (see Figure 3). In this algorithm, each processor encodes a section of the video stream of the same length. Each processor encodes its section of the video sequence as an independent video sequence, so the first frame of each section is encoded as an I-frame and the rest of the frames are encoded in GOPs of 4 frames, following the LB mode. DMG-GB parallel distribution is shown in Figure 4. When an encoding process requests new workload, the coordinator process sends a group of contiguous GOPs to it. We denote this group of GOPs as GOP_BLOCK. The main goals of DMG-GB algorithm are: a) the number of communications with the coordinator process is diminished, and b) the Peak Signal-to-Noise Ratio (PSNR) and bit rate differences with respect to the bit stream produced by the sequential algorithm are reduced. In Figure 4, one GOP_BLOCK computed by one process is denoted by $G_{(a\,GBd)}$, where a denotes the rank of the parallel process and GBd denotes the GOP_BLOCK number d computed by the process a.

4 Numerical experiments

In this section we analyze the parallel algorithms described in Section 3, in terms of parallel performance, PSNR and bit rate. The developed parallel strategies are designed to run on distributed memory platforms. We have used MPI [15] in order to manage the parallel system. The parallel platform used is a distributed memory multiprocessor with 24 nodes HP Proliant SL390 G7. Each node is equipped with two Intel Xeon X5660. Each X5660

ISBN: 978-84-617-2230-3

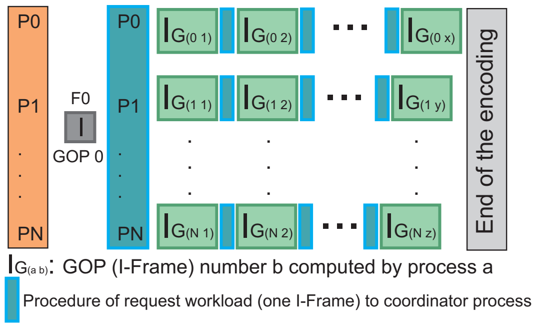

P0
P1
PN
F0
I
GOP 0
P0
P1
PN
$I_{G(0\ 1)}$ $I_{G(0\ 2)}$ … $I_{G(0\ x)}$
$I_{G(1\ 1)}$ $I_{G(1\ 2)}$ … $I_{G(1\ y)}$
$I_{G(N\ 1)}$ $I_{G(N\ 2)}$ … $I_{G(N\ z)}$
End of the encoding
$I_{G(a\ b)}$: GOP (I-Frame) number b computed by process a
Procedure of request workload (one I-Frame) to coordinator process

Figure 2: DMG-AI: Parallel distribution.

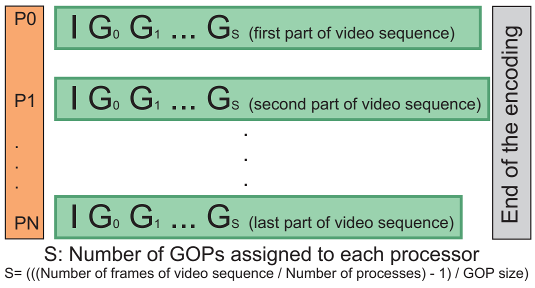

P0
P1
PN
I G_0 G_1 … G_S (first part of video sequence)
I G_0 G_1 … G_S (second part of video sequence)
I G_0 G_1 … G_S (last part of video sequence)
End of the encoding
S: Number of GOPs assigned to each processor
S= (((Number of frames of video sequence / Number of processes) - 1) / GOP size)

Figure 3: DMG-1B: Parallel distribution.

includes six cores at 2.8 GHz. In the experiments reported, we have mapped one MPI process for each node, except when using a coordinator process, where two MPI processes are mapped to one node (being one of them the coordinator process).

The video sequences used are *BQTerrace (BQ)* and *FourPeople (FP)*, both containing 600 frames at $60Hz$ with a frame size equal to 1920×1080 and 1280×720 pixels, respectively. We have run the parallel algorithms encoding 600 frames with AI and LB modes.

In Figure 5, we present the computational results for DMG-AI parallel algorithm. Figures 5(a) and 5(c) show the computational times when encoding 600 frames for BQ and FP sequences respectively, and figures 5(b) and 5(d) show the corresponding speed-up. Note that DMG-AI algorithm is the only strategy developed for AI mode, in which there are no dependencies between frames. Therefore, both the bit stream and PSNR do not differ from those produced by the sequential algorithm. As we have said, in AI mode there are no dependencies with future or past frames, so the order in which the frames are encoded does not change the results, neither the bit stream nor, obviously, the PSNR value.

Regarding Figure 2, the DMG-AI algorithm includes a coordinator process. The coordinator process and an encoding process are mapped in the same node but in different MPI processes. However, this fact does not reduce the parallel performance, as it can be

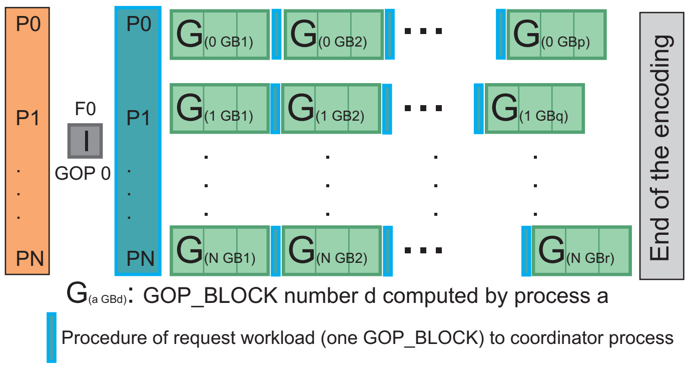
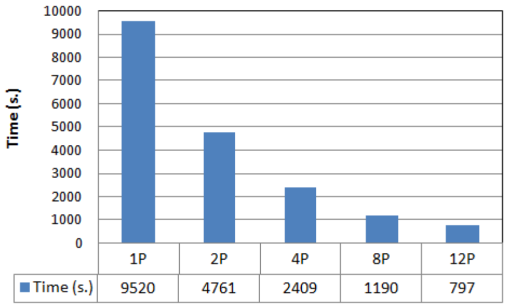

P0
P1
PN
F0
I
GOP 0
G(0 GB1)
G(0 GB2)
G(0 GBp)
G(1 GB1)
G(1 GB2)
G(1 GBq)
G(N GB1)
G(N GB2)
G(N GBr)
End of the encoding
G(a GBd): GOP_BLOCK number d computed by process a
Procedure of request workload (one GOP_BLOCK) to coordinator process

Figure 4: DMG-GB: Parallel distribution.

seen in figures 5(b) and 5(d), in which ideal efficiencies are obtained. This ideal behavior is expected for this algorithm, since there are no dependencies between different I-frames, and each one is encoded independently of the other frames. In LB mode, the motion estimation algorithm performed over B-frames, reduces considerably the output bit rate but significantly increases the time needed to encode the sequence, mainly due to the CU searching and partitioning algorithm. Looking at figures 5(a) and 6(a), for BQ sequence, we can see that for one processor (sequential algorithm), DMG-1G and DMG-1B algorithms for LB mode require 3.5$\times$ the time required to encode the sequence in AI mode.

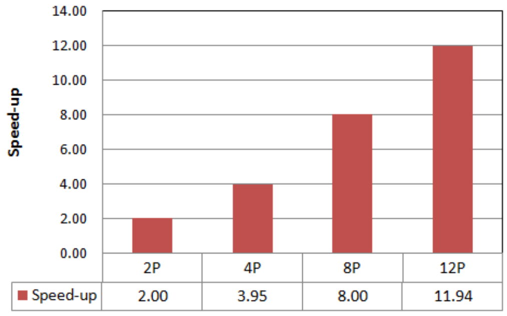
In Figure 6 we present the computational times for DMG-1G and DMG-1B parallel algorithms using LB encoding mode, while in Figure 7 we show the corresponding speed-ups. As it can be seen, execution times differ in the parallel executions. For both BQ and FP sequences, DMG-1G algorithm is substantially slower than DMG-1B algorithm. One of the reasons for this behavior is that all frames assigned to each process in DMG-1B are contiguous, so the motion estimation algorithm performed in B-frames may converge faster than in a randomized sequence, as used in DMG-1G.

As mentioned in Section 3, the parallel algorithm does not provide the same results than the ones produced by the sequential algorithm, as shown in Figure 8. The bit rate increases as the number of processes does in DMG-1G and DMG-1B parallel algorithms. If we compare the output bit stream size between both parallel algorithms, the bit rate in DMG-1G is considerably bigger than in DMG-1B. This is due to the reallocation process of B-frames in DMG-1B, performed by the coordinator process. As there are no adjacent frames, the residuum obtained in the motion estimation process is higher and so the bit rate increases. On the other hand, in DMG-1G algorithm, all the frames assigned to each process are contiguous, so adjacent frames may be similar and the residuum will be lower. As previously mentioned, in DMG-AI algorithm the generated bit stream is independent of the number of processes used to encode the sequence, because no motion estimation is



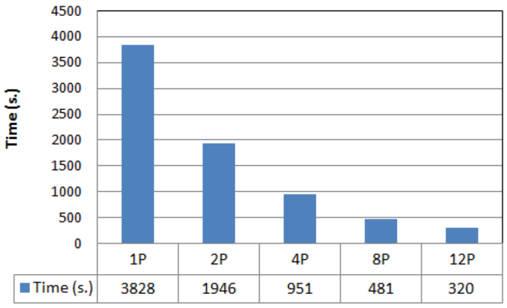
	1P	2P	4P	8P	12P
🟦 Time (s.)	9520	4761	2409	1190	797

(a) Time (s.), BQ sequence.



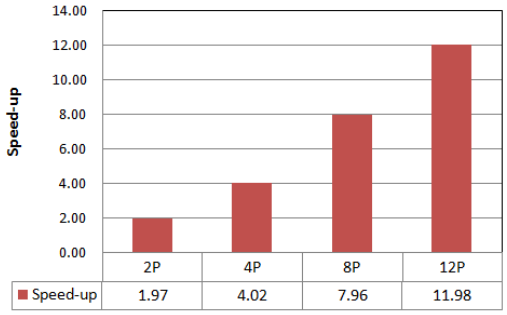
	2P	4P	8P	12P
🟥 Speed-up	2.00	3.95	8.00	11.94

(b) Speed-up, BQ sequence.



	1P	2P	4P	8P	12P
🟦 Time (s.)	3828	1946	951	481	320

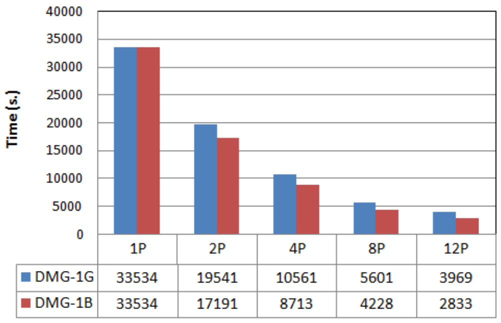
(c) Time (s.), FP sequence.



	2P	4P	8P	12P
🟥 Speed-up	1.97	4.02	7.96	11.98

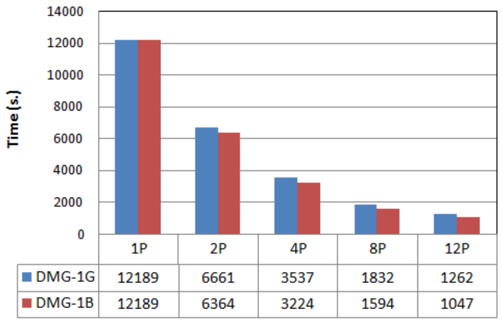
(d) Speed-up, FP sequence.

Figure 5: DMG-AI parallel algorithm when computing 600 frames.



	1P	2P	4P	8P	12P
🟦 DMG-1G	33534	19541	10561	5601	3969
🟥 DMG-1B	33534	17191	8713	4228	2833

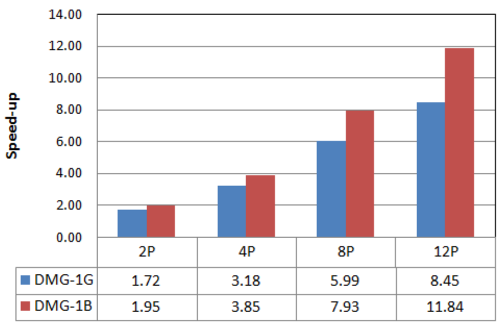
(a) BQ sequence.



	1P	2P	4P	8P	12P
🟦 DMG-1G	12189	6661	3537	1832	1262
🟥 DMG-1B	12189	6364	3224	1594	1047

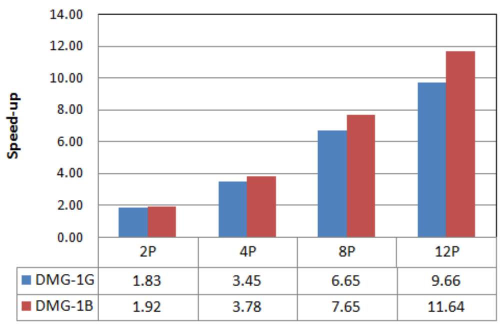
(b) FP sequence.

Figure 6: Computational times for DMG-1G and DMG-1B parallel algorithms.



	2P	4P	8P	12P
🟦 DMG-1G	1.72	3.18	5.99	8.45
🟥 DMG-1B	1.95	3.85	7.93	11.84

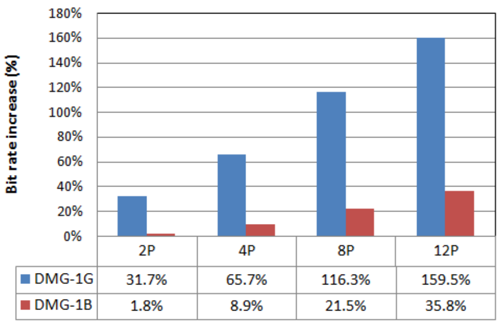
(a) BQ sequence.



	2P	4P	8P	12P
🟦 DMG-1G	1.83	3.45	6.65	9.66
🟥 DMG-1B	1.92	3.78	7.65	11.64

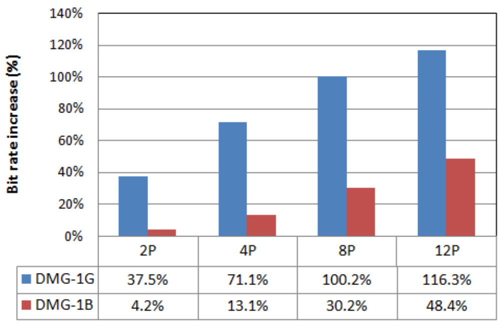
(b) FP sequence.

Figure 7: Speed-up for DMG-1G and DMG-1B parallel algorithms.



	2P	4P	8P	12P
🟦 DMG-1G	31.7%	65.7%	116.3%	159.5%
🟥 DMG-1B	1.8%	8.9%	21.5%	35.8%

(a) BQ sequence.



	2P	4P	8P	12P
🟦 DMG-1G	37.5%	71.1%	100.2%	116.3%
🟥 DMG-1B	4.2%	13.1%	30.2%	48.4%

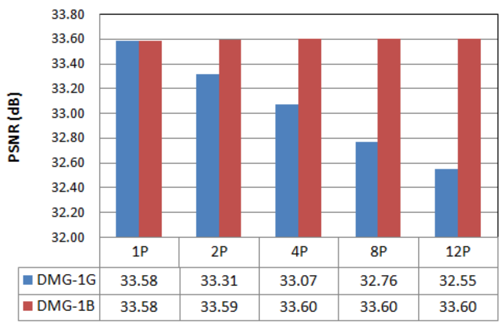
(b) FP sequence.

Figure 8: Percentage of bit rate increase for DMG-1G and DMG-1B parallel algorithms.

performed.

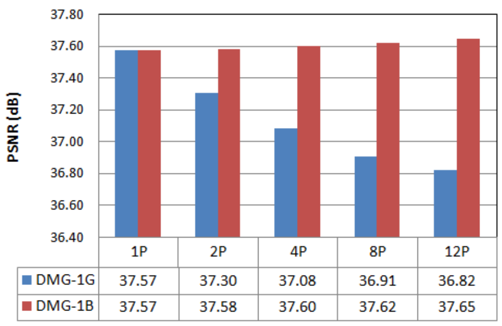
Another important value that we must consider is PSNR. That is, the generated bit stream is different for each algorithm and PSNR indicates us the distortion introduced by the algorithm in relation to the number of processes used in a parallel execution. Figure 9 shows the PSNR value as the video quality measurement for DMG-1G and DMG-1B parallel algorithms. We can observe that the video quality decreases when using DMG-1G algorithm, even if the bit rate increases, as shown in Figure 8. However, when using DMG-1B parallel algorithm, the PSNR value remains unchanged for BQ video sequence and, in some cases, the quality slightly improves for FP video sequence, with a lower increment in the bit rate than the one introduced by the DMG-1G parallel algorithm.

In order to analyze the DMG-GB parallel algorithm, we will take into account the GB (GOP_BLOCK) size. In the DMG-GB parallel algorithm, one block of consecutive GOPs



	1P	2P	4P	8P	12P
🟦 DMG-1G	33.58	33.31	33.07	32.76	32.55
🟥 DMG-1B	33.58	33.59	33.60	33.60	33.60

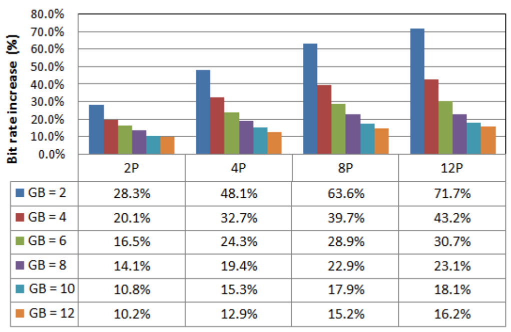
(a) BQ sequence.



	1P	2P	4P	8P	12P
🟦 DMG-1G	37.57	37.30	37.08	36.91	36.82
🟥 DMG-1B	37.57	37.58	37.60	37.62	37.65

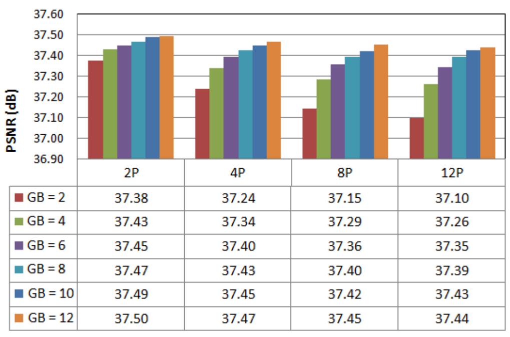
(b) FP sequence.

Figure 9: PSNR for DMG-1G and DMG-1B parallel algorithms.



	2P	4P	8P	12P
🟦 GB = 2	28.3%	48.1%	63.6%	71.7%
🟥 GB = 4	20.1%	32.7%	39.7%	43.2%
🟩 GB = 6	16.5%	24.3%	28.9%	30.7%
🟪 GB = 8	14.1%	19.4%	22.9%	23.1%
🟦 GB = 10	10.8%	15.3%	17.9%	18.1%
🟧 GB = 12	10.2%	12.9%	15.2%	16.2%

(a) Bit rate.



	2P	4P	8P	12P
🟥 GB = 2	37.38	37.24	37.15	37.10
🟩 GB = 4	37.43	37.34	37.29	37.26
🟪 GB = 6	37.45	37.40	37.36	37.35
🟦 GB = 8	37.47	37.43	37.40	37.39
🟦 GB = 10	37.49	37.45	37.42	37.43
🟧 GB = 12	37.50	37.47	37.45	37.44

(b) PSNR.

Figure 10: Bit rate and PSNR for DMG-GB algorithm varying the GB size, for FP video sequence.

is assigned to each process, so the number of GOPs of each block (GB) is a new parameter. Note that if the GB size is equal to 1, both DMG-1G and DMG-GB parallel algorithms are identical. Figure 10 shows bit rate and PSNR values for DMG-GB algorithm varying the GB size parameter. As we can see, bit rate decreases and video quality improves as the GB size increases.

5 Conclusions

In this paper we have proposed several parallel algorithms of the HEVC video encoder, specially suited for distributed memory platforms. The algorithms proposed are based on a coarse grain parallelization approach with the organization of video frames in GOPs and

ISBN: 978-84-617-2230-3

the proposal of different GOP allocation schemes. We have presented results using both AI and LB encoding modes, defined in the HEVC reference software, and we have analyzed their performance. Ideal parallel behavior has been shown in the experiments reported for DMG-AI parallel algorithm. Moreover, both DMG-1G and DMG-GB algorithms achieve good parallel performance. In DMG-GB algorithm we can tune the GB parameter to adjust the PSNR and bit rate behavior. After implementing the algorithms in the HEVC software, some experiments were performed showing interesting results as (a) the best approach is the DMG-AI (AI mode) algorithm when comparing both sequential and parallel versions in terms of speed-up, obtaining the sam bit stream than for the sequential algorithm; (b) for the rest of the algorithms, the GOP organization determines the final coding performance, causing a bit rate overhead as the number of processors increases but the overall parallel performance (except for DMG-1G proposal) makes them a good approach; and (c) all strategies which include a coordinator process, have the ability of load-balancing the input workload among the available processors. As future work, we will combine 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 is partly supported by the European Union FEDER (CAPAP-H5 network) under grant TIN2014-53522-REDT, by the Spanish Ministry of Education and Science under grant TIN2011-27543-C03-03, and the Spanish Ministry of Science and Innovation under grant TIN2011-26254.

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*, 2012.

[3] 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.

[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] C. C. Chi, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, , and T. Schierl, “Parallel scalability and efficiency of HEVC parallelization approaches,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1827 –1838, 2012.

[8] C. C. Chi, M. Alvarez-Mesa, J. Lucas, B. Juurlink, and T. Schierl, “Parallel HEVC decoding on multi- and many-core architectures,” *Journal of Signal Processing Systems*, vol. 71, no. 3, pp. 247 –260, 2013.

[9] 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.

[10] Q. Yu, L. Zhao, and S. Ma, “Parallel AMVP candidate list construction for HEVC,” in *VCIP’12*, 2012, pp. 1–6.

[11] 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.

[12] L. Bolc, R. Tadeusiewicz, L. Chmielewski, and K. Wojciechowski, “Diamond scanning order of image blocks for massively parallel HEVC compression,” *Lecture Notes in Computer Science*, vol. 7594, pp. 172 –179, 2012.

[13] C. Yan, Y. Zhang, F. Dai, and L. Liang, “Efficient parallel framework for HEVC deblocking filter on many-core platform,” in *Data Compression Conference (DCC)*, 2013.

[14] G. Cebrián-Márquez, J. L. Hernández-Losada, J. L. Martínez, P. Cuenca, M. Tang, and J. Wen, “Accelerating HEVC using heterogeneous platforms,” *Journal of Supercomputing*, vol. 71, no. 2, pp. 613 –628, February 2015.

[15] MPI Forum, “MPI: A Message-Passing Interface Standard. Version 2.2,” September 4th 2009, available at: http://www.mpi-forum.org (Dec. 2009).