Shared Memory Tile-based vs Hybrid Memory GOP-based Parallel Algorithms for HEVC Encoder

Héctor Migallón, Otoniel López-Granado, Vicente Galiano, Pablo Piñol, and Manuel P. Malumbres

Miguel Hernández University, Department of Physics and Computer Architecture, Avda. de la Universidad s/n, 03202, Elche, Alicante, Spain

Abstract. After the emergence of the new High Efficiency Video Coding standard, several strategies have been followed in order to take advantage of the parallel features available in it. Many of the parallelization approaches in the literature have been performed in the decoder side, aiming at a real-time decoder. However, the most complex part of the HEVC codec is the encoder. In this paper, we perform an analysis of two parallelization proposals. One of them is based on tiles, employing shared memory architectures and the other one is based on Groups Of Pictures, employing distributed shared memory architectures. The results show that good speed-ups are obtained for the tile-based proposal, especially for high resolution video sequences, but the scalability decreases for low resolution video sequences. The GOP-based proposal outperforms the tile-based proposal when the number of processes increases. This benefit grows up when low resolution video sequences are compressed.

Keywords: HEVC, video coding, parallel encoding, shared memory, distributed shared memory

1 Introduction

The emergence of the new High Efficiency Video Coding (HEVC) standard [1], developed by the Joint Collaborative Team on Video Coding (JCT-VC), makes possible to deal with nowadays and future multimedia market trends, like 4K- and 8K-definition video content. The HEVC standard improves coding efficiency in comparison with the H.264/AVC [2] High profile, yielding the same video quality at half the bit rate [3]. However, this increase in the compression efficiency is bound to an increase in the computational complexity. Several works about complexity analysis and parallelization strategies for the HEVC standard can be found in the literature [4] [5]. Many of the parallelization research efforts have been conducted on the HEVC decoding side, like the developments in [6] and [7]. Nevertheless, the HEVC encoder's complexity is several orders of magnitude greater than the HEVC decoder's complexity. A number of works can also be found in the literature about parallelization on the HEVC encoder side. In [8]

the authors propose a fine-grain parallel optimization in the motion estimation module of the HEVC encoder. The work presented in [9] focuses in the intra prediction module, removing data dependencies between sub-blocks. Some recent works focus on changes in the scanning order. For example, in [10], the authors propose a frame scanning order based on a diamond search, obtaining a good scheme for massive parallel processing. In [11], the authors propose to change the HEVC deblocking filter processing order. In [12], the authors present a coarse grain parallelization of the HEVC encoder based on Groups of Pictures (GOPs). In [13], the authors compare slices and tiles encoding performance in HEVC. In [14], a parallelization of the HEVC encoder at slice level is evaluated. In [15], two parallel versions of the HEVC encoder using slices and tiles are analyzed.

In this work, we present a comparison of two parallelization proposals of the HEVC encoder. The first one is performed at a tile level, especially suitable for shared memory architectures, whereas the other one is performed at a GOP level, over hybrid memory architectures. The aim of this paper is to determine which parallelization approach obtains a better performance and scalability as the number of processes increases.

2 Shared Memory Tile-based Parallel Algorithm

Tiles are rectangular divisions of a video frame which can be independently encoded and decoded. This is a new feature included in the HEVC standard which is not present in previous standards. The shape of tiles and the absence of a data header for each tile, make them more efficient than slice partitioning, regarding compression performance [13]. In our tile-based parallel algorithm, each frame is split in as many tiles as the number of parallel processes to use, and each process handles one different tile. When all the processes have finished their work, synchronization is carried out, in order to properly write the encoded bit stream and to proceed with the next frame. A tile consists of an integer number of Coding Tree Units (CTUs), and, for a specific frame resolution, we can obtain multiple and heterogeneous tile partition layouts. For example, the partition of a frame into 8 tiles can be done by dividing the frame into 1 column by 8 rows (1x8), 2 columns by 4 rows (2x4), 4 columns by 2 rows (4x2), and 8 columns by 1 row (8x1). In addition, the width of each tile column and the height of each tile row can be set up independently, so we can have a plethora of symmetric and asymmetric layouts. In our work, for each layout, we have used the column widths and the row heights which produce the most homogeneous tile shapes. In the example shown in Fig. 1, a Full HD (1920x1080 pixels) frame is divided into 10 tiles using a layout of 5 columns (with a width of 6 CTUs each), and 2 rows (with a height of 8 and 9 CTUs, respectively). As every tile has an integer number of CTUs, in many layouts a tile partition where every node processes the same number of CTUs is not possible. Moreover, a perfect load balanced layout (in which each process encodes the same number of CTUs) does not always guarantee an optimal processing work balance because the computing resources needed to encode every single CTU are not exactly the same. Different layouts

also produce different bit streams with different R/D performance. Note that, in the last steps of a tile encoding, every process needs information from the rest of the frame. Because of the amount of data to be shared between processes, this proposal is especially suitable for shared memory architectures and prevents distributed memory platforms from obtaining good parallel efficiencies.

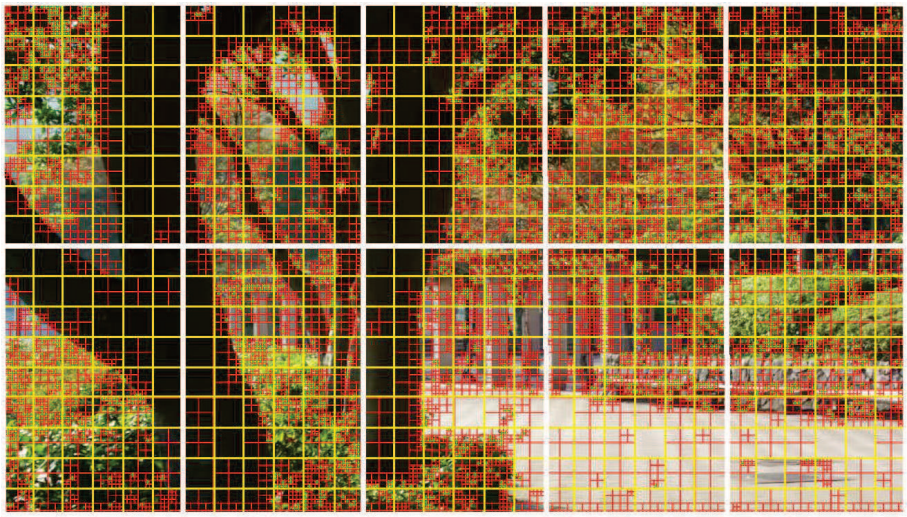


Fig. 1. Layout of a Full HD frame (1920x1080) from ParkScene sequence with 10 tiles (5 columns with a width of 6 CTUs; 2 rows with a height of 8 and 9 CTUs each)

3 Hybrid Memory GOP-based Parallel Algorithm

In this section, we depict a parallel algorithm for the HEVC encoder at a GOP level. This algorithm, named DMG-AI (Distributed Memory GOP - All Intra), is fully described in [12], and it has been implemented on a hybrid memory framework, managed by the Message Passing Interface (MPI) [16] platform. In the algorithm, there is a coordinator process which performs the data load balancing by distributing GOPs among the coding processes. It delivers the raw video data to the rest of processes, and also collects both statistical data and encoded data, in order to write the final bit stream output. In the beginning of the coding procedure, the coordinator process receives an MPI message from the rest of processes (coding processes), requesting a new GOP to encode. Note that, in All Intra mode, a GOP consists of one single video frame. After receiving the requests, the coordinator process sends one different GOP of the video sequence to each one of the requesting processes. When one of the coding processes finishes its work, it sends the encoded data to the coordinator process. Then the

coordinator process sends a new GOP to the coding process. This procedure is repeated until all the GOPs of the video sequence have been encoded. In this algorithm, when one coding process becomes idle, it is immediately assigned a new GOP to encode, so there is no need to wait until the rest of processes have finished their work. This fact yields good speed-up values. The coordinator process is mapped onto a processor which also runs one coding process, because its computational load is negligible. The final bit stream is exactly the same as the one produced by the sequential algorithm, so there is no R/D degradation.

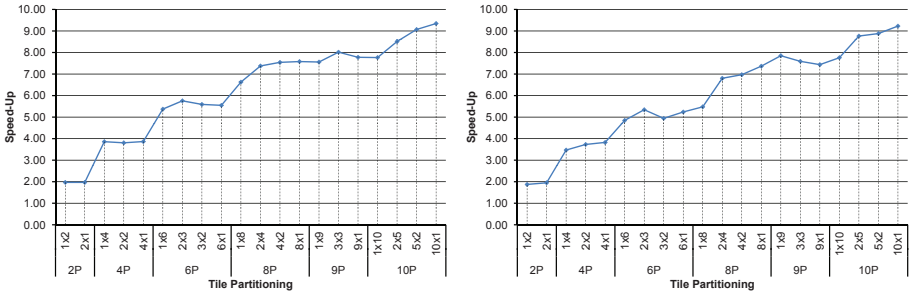
4 Numerical Experiments

In this section, we present the evaluation of the parallel algorithms introduced in sections 2 and 3, regarding parallel performance, PSNR and bit rate. For the tests, the HEVC reference software HM v16.3 [17] has been modified to implement both algorithms. For the shared memory tile-based parallel algorithm, the OpenMP API v3.1 [18] has been used, and for the GOP-based parallel algorithm, MPI v2.2 has been used, regardless of whether a strictly distributed memory, a strictly shared memory, or a hybrid memory layout is used to run the GOP-based tests. The parallel platform used is a HP Proliant SL390 G7 (a distributed memory multiprocessor with 24 nodes). Each node is equipped with two Intel Xeon X5660. Each X5660 includes six processing cores at 2.8 GHz. QDR Infiniband has been used as the communications network. The video sequences used in the experimental tests are the following:

- *Traffic* (TRAFFI), *PeopleOnStreet* (PEOPON): 2560×1600 pixels
- *ParkScene* (PARKSC), *Tennis* (TENNIS): 1920×1080 pixels
- *FourPeople* (FOURPE), *Kristen&Sara* (KRI&SA): 1280×720 pixels
- *PartyScene* (PARTSC), *BasketballDrill* (BASKDR): 832×480 pixels

Tile-based Parallel Algorithm

First of all, we analyze the performance of the tile-based parallel algorithm in order to see how the tile partitioning affects both the speed-up and the coding efficiency. We have evaluated this proposal for 2, 4, 6, 8, 9, and 10 processes. In Fig. 2, we present the encoding speed-up evolution for *Traffic* and *People* sequences in AI mode. A speed-up of up to 9.3x is obtained when 10 processes are used. Note that, for a certain number of parallel processes, different speed-ups are obtained depending on the tile partitioning layouts. This is mainly due to the fact that some tile partition layouts produce an unbalanced processing load. For example, for a video resolution of 2560x1600 (and a CTU size of 64x64), a frame consists of 40x25 CTUs. If we divide the frame using the 10x1 layout, then each one of the 10 processes will have to encode the same number of CTUs (4x25=100 CTUs). This represents a perfectly balanced load. But if we divide the frame using the 1x10 layout, then 5 processes will have to encode 40x2=80 CTUs. The other 5 processes will have to encode 40x3=120 CTUs, so they will



(a) Traffic AI mode.

(b) PeopleOnStreet AI mode.

Fig. 2. Speed-Up evolution for 4K video sequences with different number of processes and tile partitioning for QP=37

have to deal with 50% more CTUs. In general, tile partitioning layouts based on columns of CTUs or on square tiles obtain better parallel performance.

In Fig. 3 we show the average speed-up value for each video resolution with the best tile partitioning layout. As can be seen, for 4K, Full HD, and HD Ready video resolutions we obtain a good speed-up scalability up to 10 cores. However, for lower resolutions the speed-up scalability drops from 8 cores and beyond. Besides, in general, the average speed-up for a given number of processes decreases as the video resolution does.

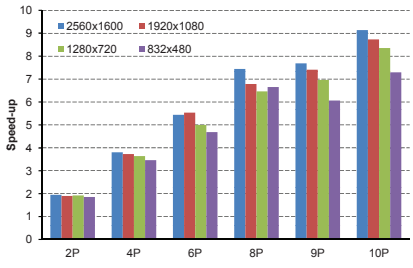


Fig. 3. Average speed-up evolution using the best tile partitioning layout.

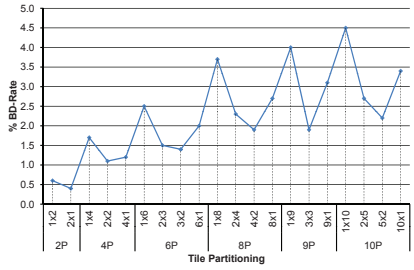


Fig. 4. % BD-Rate evolution for Tennis (1920x1080) video sequence.

Regarding R/D performance, in Fig. 4 we show the % of BD-rate increase for each tile partitioning layout, using Bjontegaard method [19]. This value measures the bit rate overhead compared with the sequential version using one tile per frame. As can be seen, the overhead increases as the number of processes does. This is an expected behavior because tiles are known to reduce compression efficiency. Square-like tile partitioning layouts perform better than the oth-

ers, providing lower overhead values. In these tiles, more redundancies between neighboring CTUs of the same tile exist, which can be exploited.

GOP-based Parallel Algorithm

To evaluate the DMG-AI algorithm on hybrid memory platforms, we have run our experiments for 10, 16, 20, and 24 coding processes. The hardware platform described before is a Distributed Shared Memory (DSM) system. It consists of 24 nodes (Distributed Memory (DM) architecture), with 12 cores (Shared Memory (SM) platforms). Table 1 shows the different combinations tested, where N denotes the number of active nodes and R is the number of cores used in each node. When N is 1, we have a pure SM platform, and when R is 1, we have a pure DM system. In this framework, the different setups tested are transparent to the algorithm because all the processing units are considered MPI processes, regardless of the memory arrangement. Figure 5 shows the algorithm framework, where P denotes the coding processes and I denotes the frames of the sequence.

Table 1. DSM parallel structures.

Number of MPI Processes	Number of nodes x Number of cores (N,R)
10	(10, 1), (5, 2), (2, 5), (1, 10)
16	(16, 1), (8, 2), (2, 8), (1, 16)
20	(20, 1), (5, 4), (4, 5), (2, 10)
24	(8, 3), (6, 4), (4, 6), (3, 8)

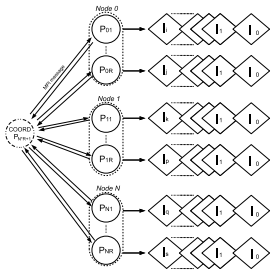


Fig. 5. DMG-AI: Parallel distribution.

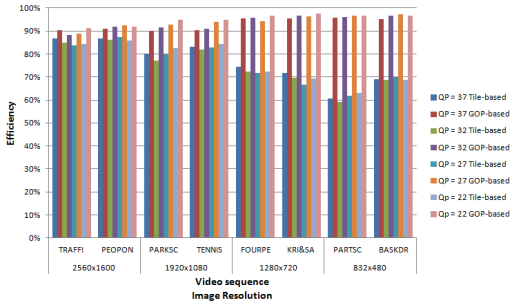


Fig. 6. Tile based vs. GOP based parallel efficiencies.

Tile-based vs. GOP-based Parallel Algorithms

Figure 6 presents the parallel efficiency for 10 processes and for both algorithms. For the tile-based proposal, the efficiency values for the different tile layouts

are averaged, and for the GOP-based approach, the results for the different framework setups are averaged. The GOP-based efficiency values are always better than the ones obtained by the tile-based proposal. This benefit is only slight for high resolution sequences, but it becomes significant as the resolution decreases. Most times, the best efficiency values are obtained when a low value for the Quantization Parameter (QP) is used. A low QP value is used when we want to improve the reconstructed video quality (and a larger bit stream is generated). In this situation, the workload increases and the parallel efficiency improves. In the hybrid memory platform, disk reading is a bottleneck. We have observed that, in this approach, when the relationship between computing load and disk reading time decreases, the parallel efficiency also decreases. Regarding R/D performance, the GOP-based approach always outperforms the tile-based proposal because, as stated before, it produces exactly the same bit stream than the sequential version and, therefore, there is no R/D degradation.

5 Conclusions

In this paper we have compared two parallelization proposals for the HEVC encoder using the All Intra coding mode. The first one is based on tiles and it is especially suitable for shared memory platforms. It obtains good speed-up values, although for low resolution sequences, the parallel scalability decreases. Moreover, the R/D performance decreases as the number of tiles increases. The other approach, which is based on GOPs, is suitable for both shared and distributed memory architectures. It yields good parallel performance, obtaining efficiency values of up to 97%. Besides, it outperforms the tile based proposal, especially when low resolution video sequences are encoded and for a number of processes higher than 10. The GOP-based approach has been tested using up to 24 processes, showing good scalability, without varying R/D performance. We have observed that the disk access is a bottleneck and, in future work, this can be avoided with the use of a parallel disk access system.

Acknowledgments

This research was supported by the Spanish Ministry of Economy and Competitiveness (MINECO) and the European commission (FEDER funds) under Grant TIN2015-66972-C5-4-R.

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. 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. 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," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1827–1838, Dec 2012.
6. C. Chi, M. Alvarez-Mesa, J. Lucas, B. Juurlink, and T. Schierl, "Parallel HEVC Decoding on Multi- and Many-core Architectures," *Journal of Signal Processing Systems*, vol. 71, no. 3, pp. 247–260, 2013.
7. B. Bross, M. Alvarez-Mesa, V. George, C. C. Chi, T. Mayer, B. Juurlink, and T. Schierl, "HEVC Real-time Decoding," *Proc. SPIE*, vol. 8856, pp. 88 561R–88 561R–11, 2013.
8. Q. Yu, L. Zhao, and S. Ma, "Parallel AMVP candidate list construction for HEVC," in *VCIP'12*, 2012, pp. 1–6.
9. 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.
10. A. Luczak, D. Karwowski, S. Mackowiak, and T. Grajek, "Diamond Scanning Order of Image Blocks for Massively Parallel HEVC Compression," in *Computer Vision and Graphics*, vol. 7594. Springer Berlin Heidelberg, 2012, pp. 172–179.
11. C. Yan, Y. Zhang, F. Dai, and L. Li, "Efficient Parallel Framework for HEVC Deblocking Filter on Many-Core Platform," in *Data Compression Conference (DCC), 2013*, March 2013, pp. 530–530.
12. H. Migallón, V. Galiano, P. Piñol, O. López-Granado, and M. P. Malumbres, "Distributed Memory Parallel Approaches for HEVC Encoder," *The Journal of Supercomputing*, pp. 1–12, 2016.
13. K. Misra, A. Segall, M. Horowitz, S. Xu, A. Fuldseth, and M. Zhou, "An Overview of Tiles in HEVC," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 7, no. 6, pp. 969–977, Dec 2013.
14. P. Piñol, H. Migallón, O. López-Granado, and M. P. Malumbres, "Slice-based Parallel Approach for HEVC Encoder," *The Journal of Supercomputing*, vol. 71, no. 5, pp. 1882–1892, 2015.
15. H. Migallón, P. Piñol, O. López-Granado, and M. P. Malumbres, "Subpicture Parallel Approaches of HEVC Video Encoder," in *2014 International Conference on Computational and Mathematical Methods in Science and Engineering*, vol. 1, 2014, pp. 927–938.
16. MPI Forum, "MPI: A Message-Passing Interface Standard. Version 2.2," September 4th 2009, available at: http://www.mpi-forum.org (Dec. 2009).
17. HEVC Reference Software, "https://hevc.hhi.fraunhofer.de/svn/svn_HEVC-Software/ tags/HM-16.3/."
18. "OpenMP Application Program Interface, version 3.1," *OpenMP Architecture Review Board. http://www.openmp.org*, 2011.
19. G. Bjontegaard, "Improvements of the BD-PSNR Model," Video Coding Experts Group (VCEG), Berlin (Germany), Tech. Rep. VCEG-M33, July 2008.