



Adaptive quadtree splitting parallelization (AQSP) algorithm for the VVC standard

Alberto González-Ruiz¹ · Antonio Jesús Díaz-Honrubia¹ ·
Santiago Tapia-Fernández¹ · David García-Lucas² ·
Gabriel Cebrián-Márquez³ · Luis Mengual-Galán¹

Accepted: 23 February 2024
© The Author(s) 2024

Abstract

The Versatile Video Coding (VVC) standard, also known as H.266, was released in 2020 as the natural successor to the High Efficiency Video Coding (HEVC) standard. Among its innovative coding tools, VVC extended the concept of quadtree (QT) splitting to the multi-type tree (MTT) structure, introducing binary and ternary partitions to enhance HEVC's coding efficiency. While this brought significant compression improvements, it also resulted in a substantial increase in encoding time, primarily due to VVC's larger Coding Tree Unit (CTU) size of 128×128 pixels. To mitigate this, this work introduces a flexible parallel approach for the QT traversal and splitting scheme of the VVC encoder, called adaptive quadtree splitting parallelization (AQSP) algorithm. This approach is based on the distribution of coding units (CUs) among different threads using the current depth level of the QT as a basis to minimize the number of broken dependencies. In this way, the algorithm achieves a good trade-off between time savings and coding efficiency. Experimental results show that, when compared with the original VVC encoder, AQSP achieves an acceleration factor of 2.04× with 4 threads at the expense of a low impact in terms of BD rate. These outcomes position AQSP competitively in comparison with other state-of-the-art approaches.

Keywords Parallelization · VVC · Quadtree

1 Introduction

The significant growth of video traffic over the Internet, which currently accounts for more than 75% of the total traffic [1], and the increasing adoption of video formats like 4K/8K, 360-degree, high dynamic range (HDR) and wide color gamut (WCG), have motivated the development of a new video coding standard. The Versatile Video Coding (VVC) standard, conceived by the Joint Video

Extended author information available on the last page of the article

Exploration Team (JVET) and officially ratified in July 2020, represents the latest flagship in video compression technology [2]. In comparison with its precursor, the High Efficiency Video Coding (HEVC) standard [3], VVC delivers a notable reduction in bit rate, albeit at the expense of a noticeable increase in computational complexity.

The coding efficiency of VVC is attributed to the novel coding tools it introduces. In particular, the new coding structure, called quadtree (QT) plus multi-type tree (MTT), allows for greater flexibility in handling diverse video content by accommodating coding units (CUs) in a frame depending on the complexity of the scene. This structure ensures that the encoding process can adapt efficiently, allocating more bits to complex regions while conserving resources in simpler areas. This, combined with advanced tools for intra-frame and inter-frame prediction, enhanced motion compensation and improved entropy coding, makes VVC a cutting-edge standard capable of delivering high-quality video while reducing bit rates, meeting the demands of the modern digital landscape.

All these novel tools, while reducing bit rates significantly, increase the computational demands of both the encoder and the decoder. In particular, the encoder needs to perform the so-called rate distortion optimization (RDO) to find a balance between minimizing bit rates and preserving video quality. To make informed decisions, the encoder must consider various quantization levels, motion vectors and other encoding parameters, which necessitates extensive computational resources and time-consuming processes. The QT structure described above alone results in 341 leaf nodes that need to be evaluated to find a sub-optimal coding efficiency.

To tackle this complexity, fast encoding algorithms can significantly accelerate the encoding process, making it more suitable for real-time applications. However, these speed-focused techniques often come at the cost of large coding efficiency penalties. Parallel architectures, in contrast, represent a promising way to mitigate the inherent complexity of VVC by distributing smaller, more manageable encoding tasks across multiple cores or processors. This approach not only accelerates the encoding process but also enhances the scalability of VVC on modern multi-core CPUs and GPUs.

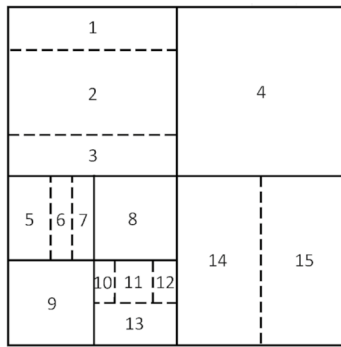
In this regard, this paper proposes the adaptive quadtree splitting parallelization (AQSP) algorithm. AQSP distributes multiple processing threads across the CUs of the QT coding structure to parallelize the entire encoding process. The main contribution of this proposal lies in its intelligent distribution of processing threads, aimed at maximizing CPU utilization while minimizing the number of broken dependencies between neighbor CUs. As a result, the experimental evaluation of AQSP shows an acceleration factor of 2.04× with 4 threads at the expense of only 5.11% BD rate (BDR) [4]. These outcomes position AQSP competitively with respect to similar works in the state of the art.

The remainder of this paper is organized as follows. Section 2 summarizes the main features and coding tools of the VVC standard, in particular its coding

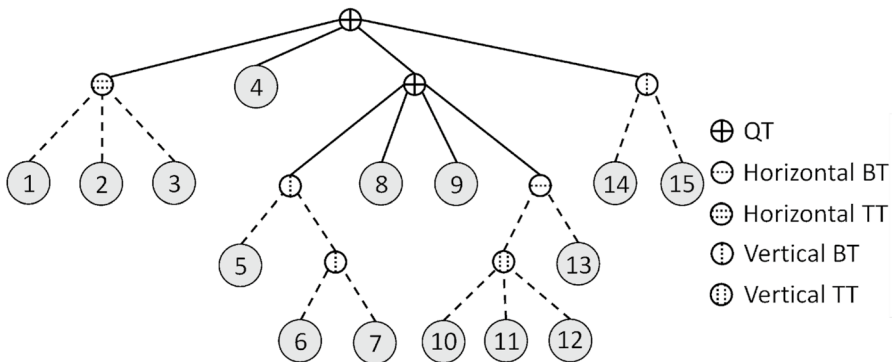
structure. Section 3 analyzes the existing related works regarding parallel implementations of the VVC encoder. The description of the AQSP proposal can be found in Sect. 4 and its evaluation in Sect. 5. Finally, the main conclusions are drawn in Sect. 7.

2 Technical background

As mentioned above, the partitioning scheme of VVC represents a significant evolution over its predecessors. The HEVC standard relies on Coding Tree Units (CTUs) with sizes of up to 64×64 pixels and recursive QT partitioning into four CUs as small as 8×8 pixels. In contrast, VVC introduces a new CTU size of 128×128 pixels and employs the innovative MTT structure for partitioning [5]. This MTT framework brings enhanced flexibility and adaptability in image partitioning, which is executed in two stages: First, the QT structure allows recursive division of CUs into four equal-sized square blocks; and second, the binary tree (BT) and ternary tree (TT) and



(a) Example of CTU partitioning.



(b) Final QT plus MTT for the given partitioning.

Fig. 1 Example of CTU partitioning using the QT plus MTT scheme

(TT) structures enable horizontal and vertical splitting of the QT leaf nodes, with the exception of the exclusive use of BT for 128×128 pixel QT leaf nodes. Figure 1 illustrates an example of CTU splitting (Fig. 1a) using the new QT plus MTT partitioning (Fig. 1b).

VVC also introduces significant advancements in the inter-prediction module [6]. It incorporates the VVC affine motion compensated prediction, employing two distinct motion compensation models. Moreover, the implementation of the sub-block-based temporal motion vector prediction (SbTMVP) tool improves the temporal motion vector prediction (TMVP) technique of HEVC. This enhancement contributes to the adaptability and accuracy of motion vector encoding, allowing for more refined choices between four-luma-sample, integer-luma-sample or quarter-luma-sample motion vector (MV) units.

Regarding the intra-prediction module, VVC distinguishes itself from HEVC by introducing a substantial expansion in prediction modes [7]. While HEVC offers 33 modes, VVC extends this to a total of 65. This expansion enables VVC to make more effective predictions according to the characteristics of the image characteristics. Regarding the Planar and DC modes, they remain the same as in HEVC.

In other modules, the main innovations include the introduction of the multiple transform selection (MTS) technique [8], which encompasses a set of different transform functions, and the employment of a high-precision MV storage, allowing for up to 1/16 fraction accuracy [9].

3 Previous works in the topic of VVC encoding parallelization

In the context of parallel algorithms for VVC coding, the existing literature is relatively limited due to the recency of the standard. By examining the available research, this section presents a comprehensive review of the current state of the art in parallel algorithms for VVC coding.

In [10], a frame partitioning algorithm for parallel processing is presented using the tile grid partitioning of VVC. The proposed algorithm introduces a Tile and Rectangular Slice (TRS) partitioning solution that takes into account both the spatial texture of the content and the encoding times of previously encoded frames. The goal is to find the best partitioning configuration that minimizes the trade-off between multi-thread encoding time and encoding quality loss. The TRS partitioning allows for processing rectangular regions of a frame with independent threads, thereby increasing the partitioning flexibility. Experimental results conducted on the VVC test model (VTM) 6.2 [11] in a 4-thread configuration on FHD content achieve an encoding speedup of 3.1×, but with a 1.57% penalty in terms of BDR.

An intra-fast coding proposal including different parallel configurations is described in [12]. First, some scalar operations are replaced by single-instruction–multiple-data (SIMD) instructions in several VTM 10.0 encoder modules:

discrete transform, inverse transform, quantization, inverse quantization, intra-prediction and deblocking filter. For parallel processing, wave front parallel processing (WPP) is employed, and horizontal/vertical deblocking filters are separately processed by different threads. Additionally, the intra-mode decision has been modified by a coarse-to-fine search in angular direction granularity, while block partitioning remains the same. The achieved results demonstrate a significant acceleration in the encoding process, reaching a speedup of 710x. However, the accompanying compression penalty is substantial, to the extent that it nullifies the overall compression advantage offered by VVC over HEVC, with a BD rate of 49%. It is also worth noting that the authors do not use sequences in class D of the common test conditions [13] for VVC, whose results are expected to worsen the overall performance of the proposed algorithm.

The literature on parallel techniques for the VVC standard is still in its early stages and is expected to incorporate more techniques in the coming years. The study and improvement of parallel tools have been a major focus within the scientific community, especially in comparison with prior standards like HEVC and AVC. As a result, there are numerous proposals concerning multi-frame, multi-slice and tile partitioning approaches, as referenced in works such as [14–19]. However, many of these proposals, particularly those centered on multi-slice approaches, primarily aim to achieve a perfect workload distribution among slices by creating and modifying the sizes of new slices to maximize acceleration. As a result, since these proposals prioritize parallelism over video coding efficiency, they often compromise the overall compression performance. On the other hand, proposals like [20] focus primarily on coding efficiency. In this work, authors allow for the existence of references between slices, which can potentially have a negative impact on slice utilization if dependencies are not properly handled. However, authors in [20] demonstrate that a three-stage approach (pre-slice encoding, slice encoding and post-slice encoding) can achieve an acceleration close to the theoretical maximum. Therefore, it is crucial to design algorithms that balance both aspects: fast encoding and coding efficiency.

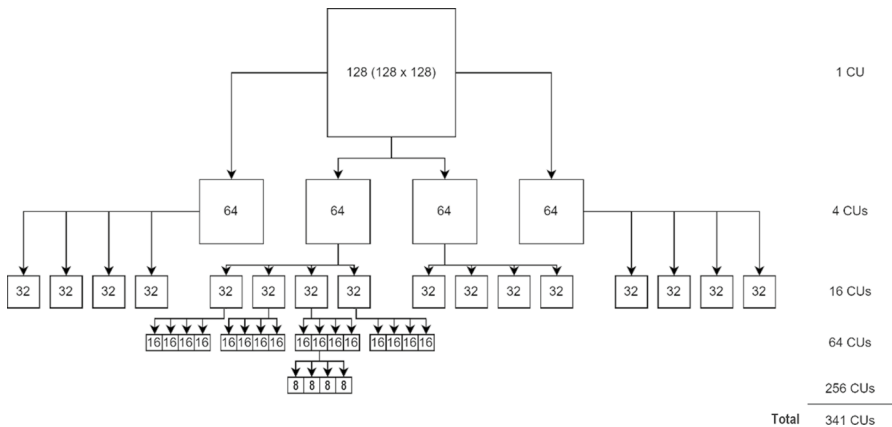


Fig. 2 Representation of the QT structure

4 Proposed adaptive quadtree splitting parallelization (AQSP) algorithm description

This section aims to describe the proposed adaptive quadtree splitting parallelization (AQSP) algorithm. As explained in Sect. 2, and depicted in Fig. 2, the QT structure in the VVC standard can have up to 5 depth levels. Specifically, depth level 0 (CU size of 128×128 pixels) has only 1 node, depth level 1 (CU size of 64×64 pixels) has 4 nodes, depth level 2 (CU size of 32×32 pixels) has 16 nodes, depth level 3 (CU size of 16×16 pixels) has 64 nodes, and depth level 5 (CU size of 8×8 pixels) has 256 nodes. In total, there are 341 nodes whose evaluation is, in principle, independent of each other, making them ideal for parallelization.

However, ME algorithms, for example, do not perform a complete search to avoid unaffordable execution times. Instead, they employ heuristic algorithms that use the best MV candidate obtained from the antecedent CUs as a starting point to perform ME. This approach results in a sequential dependency in the tree traversal that affects multiple tools in VVC. For instance, intra-prediction relies on the most probable mode (MPM) list that is constructed based on the information of the left and above neighboring blocks. Similarly, inter-prediction uses techniques such as matrix-weighted intra-prediction (MIP) and the extended merge prediction method that use coding information coming from neighboring blocks [21]. In this regard, it is particularly important to highlight the fast encoding algorithm proposed in [22] that limits the tree depth based on the depth of neighboring CUs. Therefore, the proposed AQSP algorithm needs to consider these dependencies to minimize disruptions when distributing work among different threads. Specifically, the upper and left boundary sub-blocks of the CU processed by a thread may not fully benefit from the aforementioned techniques.

4.1 Node distribution algorithm

The most straightforward work distribution technique among all the available threads would be to create a pool with all the CUs (or QT nodes) in the order they should be processed sequentially. Then, when a thread finishes processing a node, it would proceed to the next one in the pool that has not been previously processed or is currently being processed by another thread. However, this approach would result in breaking some dependencies, leading to significant compression penalties.

As discussed above, the utilization of information reuse techniques between consecutive CUs presents challenges in certain scenarios. For instance, the absence of an initial reference point for seeking an optimal MV in inter-prediction can result in a propensity to converge toward local optima. A more significant concern arises from the inherent limitations imposed on the partitioning of a CU based on the

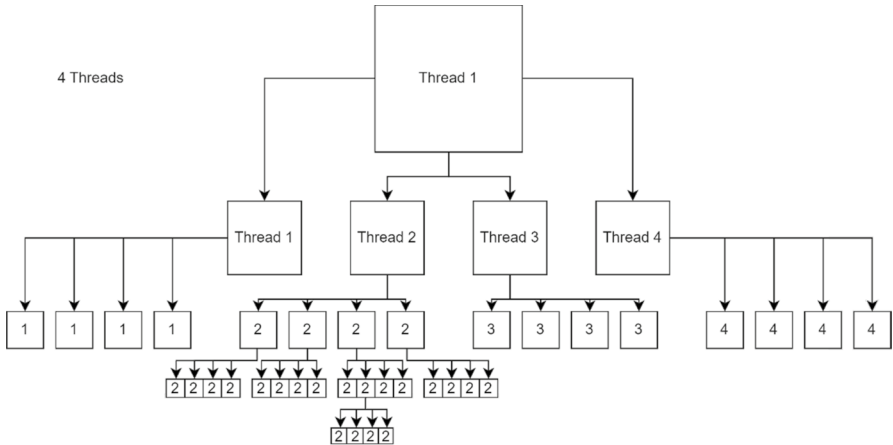
dimensions of its neighboring units. When the adjacent CUs have not yet been encoded completely, they are treated as if they have the maximum size possible. Consequently, the current CU may automatically skip splitting modes, impacting the overall coding process. To address this issue, a preliminary scheduling process is required to determine which node should be processed by each thread. This scheduling should take into account the expected workload for each node and thread, as well as the dependencies between a node and other nodes, including its parent.

Taking the previous aspects into consideration, the proposed AQSP algorithm makes use of the QT depth levels to distribute work among different threads. First of all, the algorithm determines the QT level at which the distribution will occur, denoted as $level_{dist}$, considering the number of threads that are going to be used, n_{th} , which is an input parameter for the algorithm. The level used for this purpose is the first level with, at least, as many nodes as threads are going to be used. Such a number can be obtained as follows:

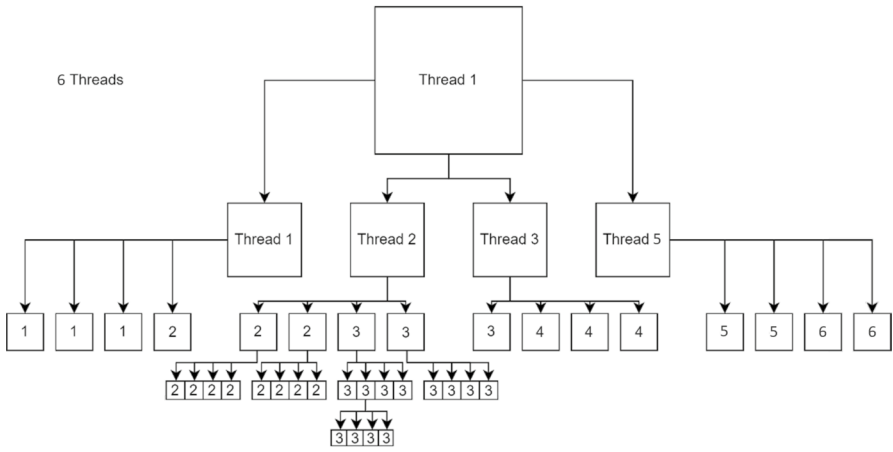
$$level_{dist} = \lceil \log_4 (n_{th}) \rceil$$

After determining the distribution level, the nodes at that level are distributed to each thread. To achieve this, the number of nodes per thread is calculated by dividing the number of nodes in the level by the number of threads. If the remainder of this division is not 0, then as many threads as indicated by this remainder are assigned one extra CU.

After each node at $level_{dist}$ has been assigned to a thread, all the successor nodes of a node at that level are assigned to that same thread using a top–bottom approach. Given that an ancestor node may have children assigned to different threads, every node on a given level is assigned to the same thread as its leftmost child using a bottom–top approach. Algorithm 1 describes the proposed scheduling process, which takes the number of threads to be used, n_{th} , as parameter, and returns $nodes$, which is an array of integers. Each position in this array represents the number of the thread to which the node at depth level i and position j at that depth level has been assigned. Figure 3a and b illustrates the results of this algorithm for 4 and 6 threads, respectively.



(a) Example using 4 threads.



(b) Example using 6 threads.

Fig. 3 QT node distribution example for different threads

Algorithm 1 nodes_distribution(n_th)

```

1:  $level\_dest \leftarrow \text{ceil}(\log_4(n\_th))$ 
2:  $nodes\_level\_dest \leftarrow 4^{level\_dest}$ 
3:  $nodes\_per\_thread \leftarrow nodes\_level\_dest / n\_th$ 
4:  $threads\_extra\_node \leftarrow nodes\_level\_dest \% n\_th$ 
5:
6: // Set threads of distribution level (threads with an extra node)
7: for  $i = 0, \dots, threads\_extra\_node - 1$  do
8:   for  $j = 0, \dots, nodes\_per\_thread$  do
9:      $nodes_{level\_dest, i*(nodes\_per\_thread+1)+j} \leftarrow i$ 
10:   end for
11: end for
12:  $assigned\_nodes \leftarrow threads\_extra\_node * (nodes\_per\_thread + 1)$ 
13:
14: // Set threads of distribution level (remaining threads)
15: for  $i = 0, \dots, nodes\_level\_dest - threads\_extra\_node - 1$  do
16:   for  $j = 0, \dots, nodes\_per\_thread - 1$  do
17:      $nodes_{level\_dest, assigned\_nodes+i*nodes\_per\_thread+j} \leftarrow i$ 
18:   end for
19: end for
20:
21: // Set threads of successors
22: for  $i = level\_dest + 1, \dots, 4$  do
23:   for  $j = 0, \dots, 4^i$  do
24:      $nodes_{i,j} \leftarrow nodes_{i-1, \text{floor}(j/4)}$ 
25:   end for
26: end for
27:
28: // Set threads of ancestors
29: for  $i = 0, \dots, level\_dest + 1$  do
30:   for  $j = 0, \dots, 4^i$  do
31:      $nodes_{i,j} \leftarrow nodes_{i+1, 4*j}$ 
32:   end for
33: end for
34: return  $nodes$ 

```

As shown in Fig. 3b, the algorithm can handle number of threads that are not multiple of 4. To achieve optimal node distribution (where no extra nodes are assigned to any thread), the number of threads should be a divisor of the total number of nodes at the distribution level. Given that the algorithm adapts the level at

which the distribution is done, the number of threads to be used so that the distribution is optimal must be a power of 2.

4.2 Optimal mode decision

It must be considered that, after all the nodes have been processed, the optimal splitting needs to be selected. This is achieved by comparing the Lagrangian cost of the current node with the sum of the Lagrangian costs of its four children. If the cost of the node itself is lower than the sum of its children, then the QT will not be split for that node and the encoding mode decision for the node will be the one selected as optimal during node processing. However, if the cost of the node exceeds the sum of its children's costs, the decision for that node is overwritten as "split using QT."

In the sequential algorithm, this can be accomplished with the backtracking process that is inherent to any tree traversal algorithm. However, in the proposed AQSP algorithm, this process needs to be done after all the threads have completed their work. For this reason, the cost and the mode for each node must be stored during processing. Otherwise, all the modes (except QT splitting) would be needed to be re-evaluated at the leaves of the QT.

After the parallel stage of the algorithm, the update of the costs and encoding modes is done using a bottom-up approach as shown in Algorithm 2. In this algorithm, $mode_{i,j}$ represents an array of integers, where each value represents the index of the encoding mode to be used for the node at depth level i and position j , and $cost_{i,j}$ is an array of floats, indicating the cost of using the aforementioned encoding mode in that same node. These values must be initialized during node processing.

Algorithm 2 tree_backtracking()

```

1: for  $i = 3, \dots, 0$  do
2:   for  $j = 0, \dots, 4^i$  do
3:     if  $cost_{i,j} \geq cost_{i+1,4*j} + cost_{i+1,4*j+1} + cost_{i+1,4*j+2} + cost_{i+1,4*j+3}$  then
4:        $cost_{i,j} \leftarrow cost_{i+1,4*j} + cost_{i+1,4*j+1} + cost_{i+1,4*j+2} + cost_{i+1,4*j+3}$ 
5:        $mode_{i,j} \leftarrow \text{QUADTREE\_SPLIT}$ 
6:     end if
7:   end for
8: end for
9: return  $modes$ 

```

Once the optimal modes are known, the proposed AQSP algorithm splits the current CU into four sub-CUs when the node is not a leaf of the final QT. Then, it skips the evaluation of all modes in the leaves except for the optimal, avoiding the repetition of any evaluation.

5 AQSP algorithm evaluation

Regarding the experimental setup, the hardware platform used in the tests was composed of an Intel Xeon Silver 4314 CPU with 16 physical cores and simultaneous multi-threading technology (up to 32 logical cores), each of which running at 2.40 GHz and 16 GB of main memory. The encoders were compiled with GCC 11.3.0 and executed on Ubuntu 22.04.2 LTS. Sequences were encoded according to the CTC document issued by the JVET to compare proposals in the field of VVC [13]. These CTC ensure standardized comparisons across experiments. Specifically, the random access (RA) configuration has been used, with QP values of {22, 27, 32, 37}. From the video sequences suggested for this configuration, classes A1, B, C and D have been selected for the experiments. The names and resolutions of the sequences are as follows:

- Class A1 (3840 × 2160 pixels): Tango2 and Campfire.
- Class B (1920 × 1080 pixels): Kimono, ParkScene, Cactus, BasketballDrive and BQTerrace.
- Class C (832 × 480 pixels): RaceHorsesC, BQMall, PartyScene and BasketballDrill.
- Class D (416 × 240 pixels): RaceHorses, BQSquare, BlowingBubbles and BasketballPass.

Following the CTC, the results are presented in terms of BDR, which measures the increment in bit rate for a fixed objective quality, and the acceleration

Table 1 Results of the proposed AQSP algorithm for the QT search

Class	Sequence	BDR (%)	Global acceleration	Local acceleration
Class A1	Tango2	2.27	1.71×	2.34×
	Campfire	3.18	1.83×	2.76×
Class B	Kimono	2.19	1.97×	3.04×
	ParkScene	4.89	2.03×	2.90×
	Cactus	5.34	1.56×	1.93×
	BasketballDrive	4.10	1.85×	2.65×
	BQTerrace	6.28	2.01×	3.09×
Class C	BasketballDrill	8.67	1.63×	2.13×
	BQMall	4.53	2.23×	3.29×
	PartyScene	3.20	2.21×	3.01×
	RaceHorsesC	9.49	2.77×	3.65×
Class D	BasketballPass	10.31	2.37×	3.35×
	BQSquare	0.76	2.51×	3.21×
	BlowingBubbles	2.14	1.96×	2.77×
	RaceHorses	9.25	1.99×	2.72×
Average		5.11	2.04×	2.86×

Table 2 Impact of the number of threads used in the proposed AQSP algorithm

Class	Sequence	4 threads		8 threads	
		BDR (%)	Acceleration	BDR (%)	Acceleration
Class B	Kimono	2.19	1.97×	4.35	2.13×
Class C	BQMall	4.53	1.63×	6.52	2.36×
Class D	BlowingBubbles	2.14	1.96×	3.49	2.12×
Average		2.95	1.85×	4.79	2.20×

Table 3 Results of the proposed algorithm compared to state-of-the-art methods

	BDR (%)	Acceleration
Proposed AQSP algorithm 4 threads	Class A1: 2.73 Class B: 4.56 Class C: 6.47 Class D: 5.62	Class A1: 1.77× Class B: 1.88× Class C: 2.21× Class D: 2.21×
Amestoy et al. [10] 4 threads	FHD: 1.57 UHD: 1.27	FHD: 3.10× UHD: 3.27×
Amestoy et al. [10] 8 threads	FHD: 2.80 UHD: 2.33	FHD: 5.07× UHD: 5.34×
Amestoy et al. [10] 12 threads	FHD: 3.90 UHD: 3.20	FHD: 6.44× UHD: 7.09×

achieved by the VVC encoder that includes the proposed algorithm with respect to the original VVC encoder. The peak signal-to-noise ratio (PSNR) used to calculate the BDR corresponds to the luminance component. All these metrics have been extracted using version 20.0 of the reference software, i.e., VTM 20.0 [11].

Table 1 shows the BDR, the global acceleration achieved by the parallel algorithm presented in the previous section and the local acceleration of the part of the encoder that is being parallelized. It can be seen that the algorithm presents a low BDR of 5.11%, while achieving a global acceleration factor of 2.04× with respect to the original version of the VTM encoder and a local parallelization acceleration of 2.86× (being 4× the theoretical maximum local acceleration that can be achieved with 4 threads). Additionally, the results remain consistent across all sequences, regardless of their class, indicating that the algorithm scales effectively to various video resolutions (since it focuses on acceleration at QT level rather than the frame level).

Furthermore, Table 2 shows the impact of increasing the number of threads in the results obtained by the proposed AQSP algorithm for a representative subset of sequences. When using 8 threads instead of 4, the global acceleration reaches 1.85×, with a BD rate of 4.79%. It is worth noting that the BD rate increases by 63%, while the acceleration increment of 19%, suggesting that increasing the number of threads may not always yield significant benefits.

6 Comparison with state-of-the-art proposals

In this section, a comparative analysis of results with other parallel coding techniques published for VVC is presented. The primary focus of the evaluation is the achieved acceleration compared to the compression penalty introduced when compared to sequential coding as shown in Table 3.

In comparison with the proposal presented in [10], it is noteworthy that the proposed AQSP algorithm, while exhibiting a slightly higher BD rate and a marginally lower acceleration, introduces a crucial consideration. The proposal in [10] relies on the use of tiles, a technique demonstrated to result in significant subjective quality degradation attributable to the perceptible boundaries between them, as substantiated in [23] and [24]. This visual impairment goes unaccounted for in metrics like the BDR, which exclusively leverages objective quality assessments. The visual quality decline induced by the presence of tiles might potentially be mitigated through the application of filters. However, that such filters are acknowledged for their computational intensity, thereby reducing the overall acceleration achieved by the that approach.

Given that [10] represents the sole state-of-the-art proposal for VVC providing results on parallelization, it becomes pertinent to consider HEVC parallelization proposals for comparative analysis. In this sense, most of these studies focus on the acceleration achieved, often neglecting to address the associated coding penalty. For instance, the authors in [25] report a noteworthy global acceleration of 10.32 \times while using 64 threads to accelerate the HEVC encoder. This equates to a thread productivity metric (calculated as the achieved acceleration divided by the number of threads, meaning that the closer to 1.0, the better) of 0.16. In contrast, the proposed AQSP algorithm exhibits a thread productivity of 0.51 for 4 threads and 0.28 for 8 threads, demonstrating its competitive efficiency in parallel processing scenarios.

7 Conclusions

This work proposes the AQSP algorithm, a parallel algorithm for fast traversal and encoding of the QT in the new VVC video coding standard. The algorithm distributes different nodes to be processed across multiple threads taking into consideration existing dependencies between nodes to minimize encoding efficiency penalties and broken dependencies due to parallelism. The results show that the proposed algorithm accelerates the original VTM 20.0 encoder by 2.04 \times , resulting in a 51.0% reduction in encoding time, with a negligible BDR penalty of 5.11%.

Author contributions AGR implemented the algorithm in the VTM source code and prepared the figures included in the manuscript. AJDH participated in the design of the proposed algorithm, prepared the experimental environment, performed part of the test and wrote part of the main manuscript. STF participated in the design of the proposed algorithm, solved implementation aspects related to parallelism and wrote part of the main manuscript. DGL collaborated with the aspects of the implementation related to the VVC encoder, performed part of the experiments, prepared the comparison with other proposals and wrote that part in the manuscript. GCM participated in the design of the proposed algorithm, solved

implementation aspects related to the VVC encoder, performed part of the experiments and wrote part of the main manuscript. LMG participated in the design of the proposed algorithm, solved implementation aspects related to parallelism and wrote part of the main manuscript. All authors reviewed the manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work has been supported by the Spanish Ministry of Science and Innovation, the Agencia Estatal de Investigación (10.13039/501100011033) and the European Commission (FEDER: A way to make Europe) under projects PID2021-128167OA-I00, PID2021-123627OB-C52 and PID2022-142332OA-I00, and by the Regional Government of Castilla-La Mancha under project SBPLY/21/180501/000195.

Availability of data and materials Not applicable.

Declarations

Conflict of interest The authors declare no Conflict of interest.

Ethical approval Not applicable.






Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. CISCO: Cisco visual networking index: forecast and trends (2018 to 2023). Technical report
2. ISO/IEC, ITU-T (2020) Versatile Video Coding (VVC). ITU-T Recommendation H.266 and ISO/IEC 23090-3. Video standard
3. ISO/IEC, ITU-T (2013) High Efficiency Video Coding (HEVC). ITU-T Recommendation H.265 and ISO/IEC 23008-2. Video standard
4. Bjøntegaard G (2008) Improvements of the BD-PSNR Model. Technical Report VCEG-A111, ITU-T SG16 Q6
5. Huang Y-W, An J, Huang H, Li X, Hsiang S-T, Zhang K, Gao H, Ma J, Chubach O (2021) Block partitioning structure in the VVC standard. *IEEE Trans Circuits Syst Video Technol* 31(10):3818–3833. <https://doi.org/10.1109/TCSVT.2021.3088134>
6. Yang H, Chen H, Chen J, Esenlik S, Sethuraman S, Xiu X, Alshina E, Luo J (2021) Subblock-based motion derivation and inter prediction refinement in the versatile video coding standard. *IEEE Trans Circuits Syst Video Technol* 31(10):3862–3877. <https://doi.org/10.1109/TCSVT.2021.3100744>
7. Pfaff J, Filippov A, Liu S, Zhao X, Chen J, De-Luxán-Hernández S, Wiegand T, Ruffitskiy V, Ramasubramonian AK, Auwera G (2021) Intra prediction and mode coding in VVC. *IEEE Trans Circuits Syst Video Technol* 31(10):3834–3847. <https://doi.org/10.1109/TCSVT.2021.3072430>
8. Zhao X, Kim S-H, Zhao Y, Egilmez HE, Koo M, Liu S, Lainema J, Karczewicz M (2021) Transform coding in the VVC standard. *IEEE Trans Circuits Syst Video Technol* 31(10):3878–3890. <https://doi.org/10.1109/TCSVT.2021.3087706>
9. Chien W-J, Zhang L, Winken M, Li X, Liao R-L, Gao H, Hsu C-W, Liu H, Chen C-C (2021) Motion vector coding and block merging in the versatile video coding standard. *IEEE Trans Circuits Syst Video Technol* 31(10):3848–3861. <https://doi.org/10.1109/TCSVT.2021.3101212>

10. Amestoy T, Hamidouche W, Bergeron C, Menard D (2020) Quality-driven dynamic VVC frame partitioning for efficient parallel processing. In: 2020 IEEE International Conference on Image Processing (ICIP), pp 3129–3133 . <https://doi.org/10.1109/ICIP40778.2020.9190928>
11. JVET: VVC Test Model - Version 20.0. https://vcgit.hhi.fraunhofer.de/jvet/VVCSoftware_VTM/-/tree/VTM-20.0
12. Kawamura K, Unno K, Kidani Y (2021) Fast still picture coding for VVC. In: 2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), pp 70–73 (2021)
13. Li, X., Suehring K (2021) Common test conditions and software reference configurations. In: Technical Report JVET-H1010, Joint Video Experts Team (JVET)
14. Ba K, Jin X, Goto S (2010) A dynamic slice-resize algorithm for fast H.264/AVC parallel encoder. In: 2010 International Symposium on Intelligent Signal Processing and Communication Systems, pp. 1–4. <https://doi.org/10.1109/ISPACS.2010.5704638>
15. Zhao L, Xu J, Zhou Y, Ai M (2012) A dynamic slice control scheme for slice-parallel video encoding. In: 2012 19th IEEE International Conference on Image Processing, pp 713–716. <https://doi.org/10.1109/ICIP.2012.6466959>
16. Rodriguez A, Gonzalez A, Malumbres MP (2006) Hierarchical parallelization of an H.264/AVC video encoder. In: International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06), pp 363–368. <https://doi.org/10.1109/PARELEC.2006.42>
17. Storch I, Palomino D, Zatt B, Agostini L (2016) Speedup-aware history-based tiling algorithm for the HEVC standard. In: 2016 IEEE International Conference on Image Processing (ICIP), pp 824–828. <https://doi.org/10.1109/ICIP.2016.7532472>
18. Ahn Y-J, Hwang T-J, Sim D-G, Han W-J (2013) Complexity model based load-balancing algorithm for parallel tools of HEVC. In: 2013 Visual Communications and Image Processing (VCIP), pp 1–5. <https://doi.org/10.1109/VCIP.2013.6706451>
19. Blumenberg C, Palomino D, Bampi S, Zatt B (2013) Adaptive content-based Tile partitioning algorithm for the HEVC standard. In: 2013 Picture Coding Symposium (PCS), pp 185–188. <https://doi.org/10.1109/PCS.2013.6737714>
20. Franche J-F, Coulombe S (2012) A multi-frame and multi-slice H.264 parallel video encoding approach with simultaneous encoding of prediction frames. In: 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet), pp 3034–3038. <https://doi.org/10.1109/CECNet.2012.6202018>
21. Browne A, Ye Y, Hwan Kim S (Jun 2023) Algorithm description for Versatile Video Coding and Test Model 20 (VTM 20). Technical Report JVET-AD2002, Joint Video Experts Team (JVET)
22. Yamamoto Y, Ikai T, Yasugi Y (Jan 2017) Improved fast encoding setting. Technical Report JVET-E0023, Joint Video Experts Team (JVET)
23. Ahrar AM, Roodaki H (2021) A new tile boundary artifact removal method for tile-based viewport-adaptive streaming in 360° videos. *Multimed Tools Appl* 80:29785–29803
24. Gankhuyag G, Jeong J, Kim Y-H (2019) Advanced motion-constrained AV1 encoder for 8K 360 VR tiled streaming. In: 2019 International Conference on Information and Communication Technology Convergence (ICTC), pp 682–684. <https://doi.org/10.1109/ICTC46691.2019.8939730>
25. Yan C, Zhang Y, Xu J, Dai F, Li L, Dai Q, Wu F (2014) A highly parallel framework for HEVC coding unit partitioning tree decision on many-core processors. *IEEE Signal Process Lett* 21(5):573–576. <https://doi.org/10.1109/LSP.2014.2310494>

Authors and Affiliations

Alberto González-Ruiz¹ · **Antonio Jesús Díaz-Honrubia¹**  ·
Santiago Tapia-Fernández¹  · **David García-Lucas²**  ·
Gabriel Cebrián-Márquez³  · **Luis Mengual-Galán¹** 

✉ Antonio Jesús Díaz-Honrubia
antoniojesus.diaz@upm.es

Alberto González-Ruiz
alberto.gonzalez.ruiz@alumnos.upm.es

Santiago Tapia-Fernández
santiago.tapia@upm.es

David García-Lucas
garcialucdavid@uniovi.es

Gabriel Cebrián-Márquez
gabriel.cebrian@uclm.es

Luis Mengual-Galán
lmengual@fi.upm.es

¹ Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, Campus de Montegancedo, 28660 Madrid, Spain

² Department of Computer Science, University of Oviedo, Calle Leopoldo Calvo Sotelo, 18, 33007 Oviedo, Spain

³ High Performance Networks and Architectures group, Universidad de Castilla-La Mancha, Calle Investigación, 2, 02071 Albacete, Spain