Parallel Palette Mode Decoding for HEVC SCC

Shurui Ye
University of Science and Technology of China
yeshurui@mail.ustc.edu.cn

Zhibo Chen
University of Science and Technology of China
chenzhibo@ustc.edu.cn

Wenhao Zhang
Intel Corporation
wenhao.zhang@intel.com

Lidong Xu
Intel Corporation
lidong.xu@intel.com

***Abstract*—Palette mode is one of the major coding tools for the Screen Content Coding (SCC) extension to HEVC standard. On the other hand, as the parallel computing capability explosively grows in recent computing industry, people are exploring the possibility of parallelism as much as possible, to achieve video codec products with higher speed and larger throughput. However, the newly introduced palette technology poses great challenges to the implementations of SCC decoder. Under the current palette framework, the pixel reconstructing process has to wait until all the palette indices are parsed, and cannot be operated in parallel due to the syntax dependencies. This paper intends to investigate an efficient parallel decoding architecture for the emerging palette mode to tackle these conundrums. Each largest 32×32 coding unit (CU) using palette mode is split into multiple sub-CUs and coded independently. We did integrated investigation to illustrate the trade-off between parallelism and coding efficiency, as an important guidance for SCC codec parallel implementation. We tested the scenario to split the CU into 8 sub-CUs, which can achieve 320% speed acceleration with negligible loss in RD performance.**

***Keywords*—HEVC SCC; palette mode; parallel; rate-distortion**

I. Introduction

In January 2013, the first version of High Efficiency Video Coding (HEVC) [1] was finalized by the Joint Collaborative Team on Video Coding (JCT-VC). Although HEVC provides an approximate 50% bit-rate reduction at the same subjective quality level over Advanced Video Coding (AVC/H.264) [2], both of them focus more on camera-captured video. With rapid advancements made in semiconductors, networking communications and computers, a proliferation of applications use video devices to display screen content (SC), which include wireless display, screen/desktop sharing and cloud computing, etc [3]. Other than camera-captured video, SC video has some unique features. For example, there are many repeating patterns among current picture and only a few numbers of colors in a limited region. Therefore, there is an inexorable trend to improve the coding efficiency for SC video. A Call for Proposals (CfP) [4] on the extensions of the HEVC for Screen Content Coding (SCC) was issued in January 2014. Palette mode [5] is one of these new tools used for SCC.

As we know, one coding unit (CU) has a limited number of colors in SC video. In order to fully use this feature, palette mode introduces a lookup table, i.e. palette table. The entries of palette table are generated from color clustering which are regarded as the representative color values of current CU. After that, palette indices are generated for each pixel in the current CU to map to the entries of the palette table. Both of palette table and palette index map should be transmitted to the decoder, which could result in non-negligible signaling overhead. So there are many coding methods like palette table prediction and run-based index map coding to improve the coding performance. However, the parallelism of palette mode hasn't been considered much. [6] is the only one related work to the best of our knowledge, in which the author proposed to split all of the CUs into two sub-CUs to realize the parallelism of the palette mode. Obviously, this method is suboptimal since different size of CUs have the different demand for parallelism. Secondly, this proposal was designed under the outdated SCM3.0 [7] framework. The syntax architecture of palette mode has changed a lot in the new SCM5.0 [8] framework. One of the consequent challenges is that the decoder can't start reconstructing the pixels until the parsing process finishes, which isn't covered by [6].

In this paper, we propose a new palette mode decoding architecture to improve the parallelism of implementation of SCC. Our main contributions are three-fold. Firstly, we propose only to split the largest size of CU using palette mode, since smaller CUs don't pose any challenge to implementation. Secondly, our method can reduce the waiting time for reconstructing and in the meantime achieve a remarkable parallelism of palette mode. In addition, we conclude that the parallelism won't constantly increase with incremental number of split sub-CUs. Combining with the rate-distortion (RD) performance, we have also discussed about the optimal number of sub-CUs under different scenarios. Experiment shows that the proposed method can achieve an average 320% speed acceleration at the decoder with average less than 1.4% increase of BD-rate (Bjontegaard-Delta-rate) [9], where we choose to split the largest 32×32 CU into 8 sub-CUs.

The remainder of this paper is organized as follows. Section II introduces the current palette mode in HEVC SCC. The details of parallel palette mode decoding are described in section III. The parallelism, RD performance and complexity of proposed parallel architecture are analyzed in section IV. Experiment results are presented in section V. Finally, we make a conclusion of this paper.

II. Palette Mode In The Current HEVC SCC

Palette mode is a lookup table method. It composes of two major operations: deriving palette table and coding CU using the palette table.

A. Deriving Palette Table

Firstly, a modified K-means method is used to cluster the colors of current CU into K sets, where K is the palette table size. Secondly, the major colors (as the palette table entries)

978-1-4799-5341-7/16/$31.00 ©2016 IEEE

are generated by averaging each color cluster. Then pruning is performed to ensure there are no duplicated values in one palette table. In order to reduce the overhead of signaling, palette table entries from previous CUs are used to predict the major colors in the current palette table.

B. Coding CU Using the Palette Table

Palette index map is generated by classifying each pixel of current CU using palette table. In order to converting a two-dimensional index map into an one-dimensional vector, horizontal or vertical traverse scan is applied according to the brought RD cost. Then three modes are used to encode the index map. A flag named run type will be signaled to the decoder to indicate which mode is used for current pixel. Those three modes are specified as follows:

1) Copy-left mode: Two syntax elements (palette indices and run lengths) are signaled to indicate the following run length pixels have the same index as the current pixel (noted as palette index). For example in Fig. 1, the pixels in region 1 are coded by copy-left mode with palette index equaling 0 and run length equaling 3.

2) Copy-above mode: The index of the current pixel can copy from the above directly. Only the run length are required to indicate how many indices are also copied from the corresponding above indices directly. In Fig. 1, the run length for region 2 is 2.

3) Escape mode: Because palette table only contains limited major colors of the current CU, escape mode is used to code low frequency pixels which are not included into palette table like the pixel in region 3 (showed in Fig. 1). These pixels are directly coded into bitstream with lossless or lossy.

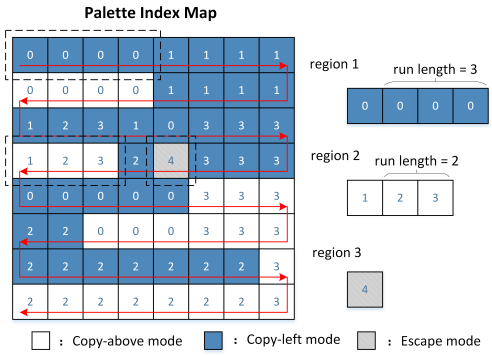


Fig. 1. Encode the index map using the three modes by horizontal traverse scan.

For signaling order, one non-negative value named *NumIds* is signaled first to indicated how many palette indices are coded in copy-left mode in the current CU. Then a list containing all palette indices coded in copy-left mode is signaled. Finally, the syntax elements of run types and run lengths are signaled interleavely for each copy mode.

III. Parallel Palette Mode Decoding

In SCM5.0, the syntax elements of palette indices (coded in bypass mode) for a CU are bonded together to be transmitted before the other syntax elements. It improves the throughput of CABAC (Context Adaptive Binary Arithmetic Coding) by grouping together the bypass coded bins. However, this operation also brings a bottleneck (as showed in Fig. 3) that reconstruction must be held during parsing the palette indices, and cannot be started until the decoder starts parsing of the first run syntax. Beyond that, copy-above mode makes the adjacent lines have strong dependency, which causes the decoder unable to reconstruct pixels in parallel. These defects will be more serious with respect to the large size of CU. So we propose to split the large CU into multiple sub-CUs to code respectively, thereby reducing the waiting time for parsing and realizing an efficient parallel decoding at the decoder.

Palette mode is used for the 8×8, 16×16 and 32×32 sizes of CU. Since the splitting operation will inevitably bring an overhead and the large CUs pose greater challenge to implementation, we only apply splitting to the largest 32×32 CUs. The splitting lines are parallel with the scanning direction instead of intersected with each other. This is because the intersected splitting is equivalent to regarding a 32×32 CU as four 16×16 CUs, which has already been checked to be suboptimal in the lower CU quad-tree splitting stage. We define N_{scu} to denote the number of the split sub-CUs. The candidate N_{scu} could be 2, 4, 8, 16 or 32, and the splitting operation is showed in Fig. 2. These sub-CUs are coded independently but share a common palette table. In order to eliminate the dependency between the adjacent sub-CUs, copy-above mode should be disabled for the first line of each sub-CU.

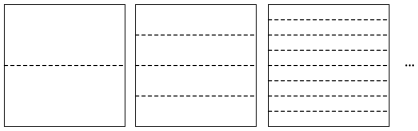


Fig. 2. Examples of splitting CU into 2, 4, 8 sub-CUs respectively.

In Fig. 3, the syntax structures of SCM5.0 and our proposal are showed, as well as their corresponding parallel implementation solutions. Due to the syntax design of SCM5.0, two threads can be utilized to perform fast decoding of a palette CU: one for parsing syntax from bitstream and the other one for reconstructing pixels. These two threads cannot be fully parallelized as mentioned previously. In contrast, the CU can be split into independent sub-CUs to be decoded when our proposal is applied. Each sub-CU has its own syntax elements, such as number of indices, index map, run types and run lengths. It indicates that the decoder can start pixel reconstructing for a sub-CU immediately after it receives all the information of this sub-CU. Take the case with 2 sub-CU splitting as an example, three thread can be launched: one for parsing (since CABAC decoding cannot be parallelized) and the other two for reconstruction. It improves parallelism effectively compared with SCM5.0. Fig. 3 also provides the diagram with 4 sub-CUs splitting. In general, more sub-CUs being used, more potential parallelism speedup could be obtained.

IV. Parallelism, RD Performance and Complexity Anaylysis

This section provides the analysis about parallelism, RD performance and complexity of our proposed parallel palette mode decoding (PPMD) architecture.

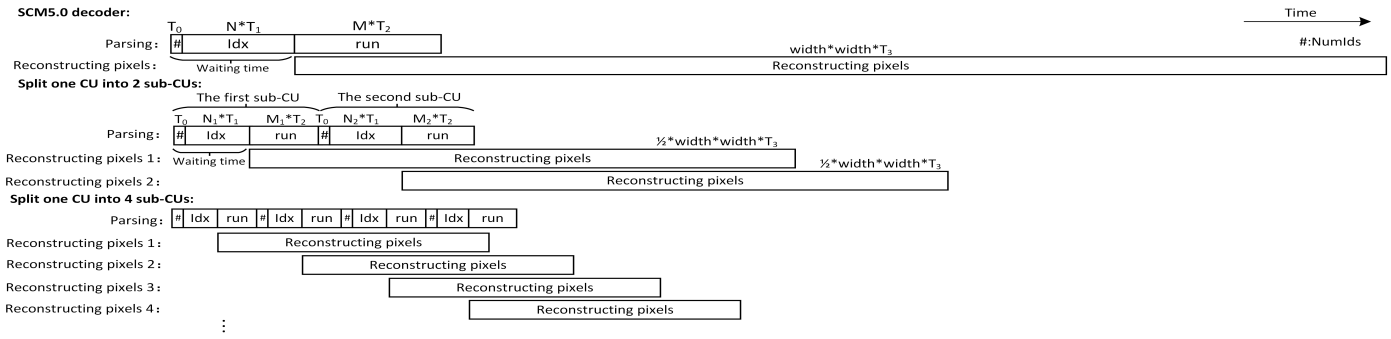


Fig. 3. SCM5.0 decoder and PPMD with Different N_{scu} (All the variables are defined in section IV-A).

A. Parallelism analysis

For easy explanation, we suppose the time consumptions of parsing one number and one index are T_0 and T_1 respectively. Then the time for parsing run type and run length is T_2, reconstructing one pixel needs time T_3. We set SCM5.0 as the anchor, in which the decoding process begins with parsing the number of indices (denoted by N). In the following, N indices are parsed before parsing the run. The number of the run length values is denoted by M. Once decoder start parsing the first run syntax, pixel reconstructing begins. Since the reconstruction process needs the information of syntax elements, its ending must be posterior to that of the parsing process as showed in Fig. 3. The number of pixels in one line is denoted by $width$. So we can express the time used for decoding palette index map for the anchor as :

$$T_{anchor} = T_0 + N \cdot T_1 + width^2 \cdot T_3 \tag{1}$$

In our method, when decoder starts to parsing the first run for the last sub-CU, the last pixel reconstructing thread will be open. Before that, the syntax elements that needs to be parsed includes: numbers of indices and the indices for n sub-CUs, and run types and run lengths for the front *(n-1)* sub-CUs (see Fig. 3). So the whole time consumption for our method is :

$$T_{parallel} = n \cdot T_0 + \sum_{i=1}^{n} N_i \cdot T_1 + \sum_{i=1}^{n-1} M_i \cdot T_2 + \frac{width^2 \cdot T_3}{n} \tag{2}$$

Where N_i and M_i denote the number of indices and the cycles of run types and run lengths for each sub-CU, respectively. Since the essence of the operators are parsing a value or reconstruction a pixel, T_0, T_1, T_2, and T_3 can be seen as approximately equal. We can estimate the percentage of $SpeedUp$ as:

$$\begin{aligned} SpeedUp &= (T_{anchor}/T_{parallel}) \times 100\% \\ &= \frac{1 + N + width^2}{n + \sum_{i=1}^{n} N_i + \sum_{i=1}^{n-1} M_i + \frac{width^2}{n}} \times 100\% \end{aligned} \tag{3}$$

We extracted the 1st frames from four standard test sequences that often use palette mode for our analysis, respectively. All the QPs are set to be 32. we tested the parallelism under different N_{scu} settings, i.e. 2, 4, 8, 16, 32 respectively. Fig. 4 shows the corresponding result. As we can see, the parallelism can't keep a continuous increase with the incremental N_{scu}. When N_{scu} is below 8, the parallelism keeps growing. But this tendency will become flat as the number between 8 and 16, until gradually dropping with the number surpassing 16. The curves in Fig. 4 can be illustrated as follows. The copy-above mode is disabled for the first line of each sub-CU, we can't use the above line to save the bits. As a result, the total time of parsing will be increased. When N_{scu} is too large, the saved time benefited from the parallel reconstruction will be engulfed by the incremental parsing time.

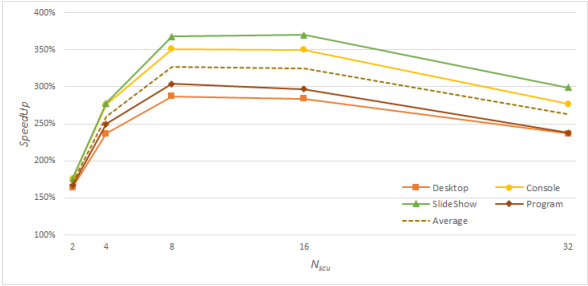


Fig. 4. Speeding up of PPMD with different N_{scu} for different test sequences.

B. RD performance analysis

We tested the first 50 frames of two sequence classes most often using palette mode under All Intra (AI) configuration, respectively. The test shows using palette mode for SC video will bring about 50% gain compared with non-palette mode in SCM5.0. BD-rate is used as the metric to evaluate coding efficiency, and the increase of BD-rate is equivalent to the increase of bit-rate under the same PSNR (Peak Signal to Noise Ratio), i.e. the loss of coding efficiency. The summary of BD-rate with different N_{scu} is showed in Fig. 5. With N_{scu} increaing, BD-rate increases gradually. So if we want to realize the parallelism with minimum loss of coding efficiency, splitting 32×32 CU into 2 sub-CUs is the best choice, which can achieve 170% parallelism with 0.55% BD-rate increase. On the contrary, splitting CU into 8 sub-CUs will realize the best 320% speed accelerate attached with 1.44% BD-rate increase.

C. Complexity analysis

From section III, It can be seen that the proposed PPMD architecture only needs to disable the first line of each sub-CUs. Compared with the advancement of parallelism, those extra operations can be ignored.

TABLE I
RD PERFORMANCE OF ALL THE TEST SEQUENCES USING PPMD WITH N_{scu} = 8 V.S. SCM5.0

	All Intra			Random Access			Low Delay B		
	G/Y	B/U	R/V	G/Y	B/U	R/V	G/Y	B/U	R/V
RGB, text & graphics with motion, 1080p	1.5%	1.6%	1.7%	1.2%	1.4%	1.3%	1.1%	1.3%	1.2%
RGB, mixed content, 1440p & 1080p	0.7%	0.8%	0.8%	0.5%	0.8%	0.6%	0.9%	1.0%	1.1%
RGB, Animation, 720p	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	-0.1%	0.0%	0.0%
RGB, camera captured, 1080p	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
YUV, text & graphics with motion, 1080p	1.9%	2.1%	2.1%	1.2%	1.4%	1.8%	1.5%	1.9%	2.4%
YUV, mixed content, 1440p & 1080p	0.8%	1.6%	1.5%	0.8%	2.0%	1.7%	0.9%	1.7%	1.6%
YUV, Animation, 720p	0.0%	0.1%	0.1%	-0.1%	0.0%	0.1%	-0.1%	-0.3%	-0.1%
YUV, camera captured, 1080p	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Overall Average	1.2%	1.4%	1.4%	0.8%	1.1%	1.2%	0.9%	1.2%	1.4%

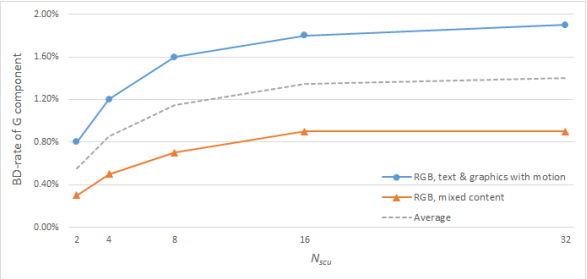
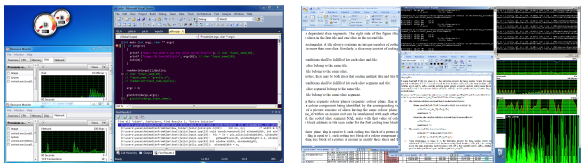


Fig. 5. BD-rate of parallel palette mode decoding with different N_{scu} for different test sequence classes.



(a) *Program* frame #0 (b) *Desktop* frame #0

Fig. 6. Examples of test sequences for SCC

V. EXPERIMENT RESULT

Our proposed method is implemented in HEVC SCC Test Model (SCM5.0) to verify the effectiveness. The experiments are performed under the All Intra (AI), Random Access (RA) and Low Delay (LD) configurations based on the common test conditions for SCC specified in [10]. From Fig. 6 we can see that SC video is very different from camera captured video. Specifically, SC video mainly contains some windows of text/program edit and waveforms. These content have a few kinds of different color values, which are very suitable for using palette mode. From the RD performance analysis in section IV-B, the high parallelism is always along with sacrifice of compression efficiency. Basically, we hope to realize a higher parallelism as possible as we can, and in the meantime keep less than 2% BD-rate increase. So the best choice is 8-splitting. The SCM5.0 is denoted as anchor. From table I it can be seen that splitting 32×32 CU into 8 sub-CUs brings average 1.2%/0.8%/0.9% BD-rate increase on component G/Y for AI/RA/LDB testing cases, respectively. Compared with the 320% speed acceleration in Section IV, this can be neglected in real world implementations and applications.

VI. CONCLUSION

In the current palette mode of HEVC SCC, pixel reconstruction must wait a long time for parsing palette index and can't be operated in parallel either. In this paper, we propose a solution to solve those problems. We only split the largest 32×32 CU into 8 sub-CUs.. The proposed PPMD architecture can shorten the waiting time for reconstruction and achieve 320% speed accelerate, making the palette mode of HEVC SCC design more friendly to high-throughput implementation, at the cost of negligible loss in RD performance. In addition, We reveal the trade-off between parallelism and coding performance, which will further guide the the parallel implementation for SCC codec.

VII. ACKNOWLEDGEMENT

This work was supported by the NSFC (Natural Science Foundation of China) under Grant 61571413, and National 973 Program of China under Grant 2015CB351803, and in part by NSFC Grant 61390514 and Intel ICRI MNC.

REFERENCES

[1] Sullivan G J, Ohm J, Han W J, et al. Overview of the High Efficiency Video Coding (HEVC) Standard[J]. Circuits & Systems for Video Technology IEEE Transactions on, 2012, 22(12):1649-1668.

[2] Joint Video Team of ITU-T VCEG and ISO/IEC MPEG, ITU-T Rec.H.264 | ISO/IEC 14496-10 MPEG-4 Part 10 (AVC), 2003.

[3] H. Yu, K. McCann, R. Cohen, and P. Amon, Requirements for an extension of HEVC for coding of screen content, ISO/IEC JTC 1/SC 29/WG 11 Requirements subgroup, San Jose, California, USA, document MPEG2014/N14174, Jan. 2014.

[4] ISO/IEC JTC1/SC29/WG11 and ITU-T Q6/SG16, MPEG2014/N14175/VCEGAW90, Joint Call for Proposals for Coding of Screen Content, San Jose, USA, Jan. 2014.

[5] Xiu X, He Y, Joshi R, et al. Palette-based Coding in the Screen Content Coding Extension of the HEVC Standard[J].

[6] Y.-J. Chang, C.-L. Lin, C.-C. Lin, C.-H. Hung, J.-S. Tu, Non-CE1: Parallel processing methods for index map coding, JCT-VC 20th Meeting, JCTVC-T0086, 2015.

[7] R. Joshi, J. Xu, R. Cohen, S. Liu, Z. Ma, Y. Ye, Screen Content Coding Test Model 3 Encoder Description (SCM 3), JCT-VC 19th Meeting, JCTVC-S1014, 2014.

[8] R. Joshi, J. Xu, R. Cohen, S. Liu, Y. Ye, Screen Content Coding Test Model 5 Encoder Description (SCM 5), JCT-VC 21th Meeting, JCTVC-U1014, 2015.

[9] Bjontegaard G. Calcuation of average PSNR differences between RD-curves[J]. Doc. VCEG-M33 ITU-T Q6/16, Austin, TX, USA, 2-4 April 2001, 2001.

[10] H. Yu, R. Cohen, K, Rapaka, J. Xu, Common Test Conditions for Screen Content Coding, JCT-VC 21th Meeting, JCTVC-U1015, 2015.