A Highly Parallel Framework for HEVC Coding Unit Partitioning Tree Decision on Many-core Processors

Chenggang Yan, Yongdong Zhang, *Senior Member, IEEE*, Jizheng Xu, *Senior Member, IEEE*, Feng Dai, Liang Li, Qionghai Dai, *Senior Member, IEEE*, and Feng Wu, *Fellow, IEEE*

***Abstract*—High Efficiency Video Coding (HEVC) uses a very flexible tree structure to organize coding units, which leads to a superior coding efficiency compared with previous video coding standards. However, such a flexible coding unit tree structure also places a great challenge for encoders. In order to fully exploit the coding efficiency brought by this structure, huge amount of computational complexity is needed for an encoder to decide the optimal coding unit tree for each image block. One way to achieve this is to use parallel computing enabled by many-core processors. In this paper, we analyze the challenge to use many-core processors to make coding unit tree decision. Through in-depth understanding of the dependency among different coding units, we propose a parallel framework to decide coding unit trees. Experimental results show that, on the Tile64 platform, our proposed method achieves averagely more than 11 and 16 times speedup for 1920x1080 and 2560x1600 video sequences, respectively, without any coding efficiency degradation.**

***Index Terms*—CU partitioning tree decision, HEVC, many-core processors, parallel framework.**

I. Introduction

HIGH EFFICIENCY VIDEO CODING (HEVC) is the state-of-the-art video coding standard [1]–[4]. Compared with H.264/AVC, HEVC provides a similar reconstructed quality with about half of bitrate [5], which largely benefits from a highly flexible hierarchy of HEVC coding unit (CU) partitioning [6]. In HEVC, each frame is divided into non-overlapping coding tree units (CTUs), which can be recursively split into smaller coding units (CUs) by using a generic quad-tree partitioning structure. A CU is the basic unit for video coding, processing and splitting. Thus, for a CTU, the CU partitioning tree (CUPT) controls how a CTU is coded with CUs with variable block sizes and coding modes, which significantly influence the coding efficiency. The price to be paid for higher coding efficiency is higher computational complexity. To decide the optimal CUPT greatly increases the search domain and the computational complexity of rate-distortion optimization (RDO) at the encoder [7]–[9].

To speed up the decision process of CUPT, many researchers have tried to reduce the search space by avoiding searching the full branches of the quad-tree [10]. In order to guarantee the coding efficiency, many branches of the quad-tree can't be skipped and the speedup is no more than two times. Meanwhile, many researchers only consider the RD-based intra mode selection. However, inter mode selection is much more time-consuming, which cannot be ignored.

Many-core processors are good candidates for speeding up compression algorithms, but only in the case that compression algorithms can be highly parallelized [11]–[16]. Efficient parallelization of CUPT decision (CUPTD) on many-core processors is challenging, because CUPTD has complicated data dependencies which provides insufficient degree of parallelism for so many cores. If CUPTD isn't extensively parallelizable, cores will be left unused and performance might suffer.

We propose a highly parallel framework for CUPTD. Firstly we analyze the dependencies among neighboring CTUs within the same frame and use the directed acyclic graph (DAG)-based order to parallelize CTUs as described in our previous work [14]. Then we analyze the dependencies in CU-level within the same frame:

- There exist completely independent CUs (CICUs), which have no data dependencies on other CUs within the same CTU.
- There exist partially independent CUs (PICUs), which have no data dependencies on other CUs when related CUs have been processed within the same CTU.

In order to further increase the degree of parallelism overhead, we process the CICUs at the beginning of processing each CTU and the PICUs when their related CUs have been processed, which exploit the implicit CU-level parallelism. To the best of our knowledge, it is the first time to have a parallel solution to CUPTD.

Our testing platform is Tile64, which is a member of TILERA many-core processor family [17]. Experiments demonstrate that

Manuscript received December 23, 2013; accepted March 04, 2014. Date of publication March 11, 2014; date of current version March 18, 2014. This work was supported by National Key Technology Research and Development Program of China under Grant 2012BAH06B01, and by the National Nature Science Foundation of China under Grants 61272323, 61102101, and 61379084 . The work of C. Yan was performed during his internship at Microsoft Research Asia. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Oscar C. Au.

C. Yan is with the Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS, Beijing 100190, China, the Department of Automation, Tsinghua University, Beijing 100084, China, and also with the University of Chinese Academy of Sciences, Beijing, 100049, China.

Y. Zhang, F. Dai, and L. Li are with the Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS, Beijing 100190, China (email: zhyd@ict.ac.cn).

J. Xu and F. Wu are with Microsoft Research Asia, Beijing 100190, China.

Q. Dai is with the Department of Automation, Tsinghua University, Beijing 100084, China.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LSP.2014.2310494

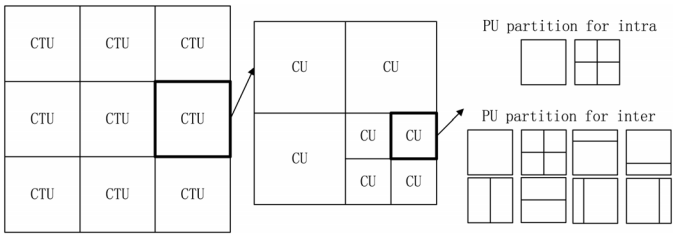

Fig. 1. Flexible hierarchy of unit representation.

our proposed method significantly saves more time than the default encoding scheme in HM 7.0.

The remainder of this letter is organized as follows. Section II gives a review of the CUPTD. Section III presents the proposed parallel framework. The experimental results are elaborated in Section IV. Finally, Section V concludes this letter.

II. HEVC CUPTD

Hierarchy of HEVC CUPTD has been proved effective [6]. As shown in Fig. 1, each frame in HEVC is divided into CTUs, which can be recursively split into a quad-tree of smaller CUs. Regions of different sizes can be better coded by using variable sizes of CUs. However, Hierarchy of HEVC CUPTD greatly increases the search domain and the computational complexity of RDO at the encoder. For example, consider an image of 2560×1600 pixels. In H.264/AVC, the size of macroblock is fixed and there is only one way to split the frame. But in HEVC, if the CTU size is fixed as 64×64 and the maximum quad-tree depth is 4, we have to search $40 \times 25 \times (1+4^1+4^2+4^3) = 85000$ CU branches in the frame. Video coding has been restricted in many fields because of its high complexity [11]–[13]. As a result, it's important to accelerate HEVC, especially CUPTD.

Efficient parallelization of HEVC CUPTD on many-core processors is challenging, because CUPTD has complicated data dependencies. For RD-based intra prediction coding tools, the coding structure follows the overall architecture of the codec. Images are split into CU, prediction unit (PU) and transform units (TU). CU is used to separate the intra and inter coded blocks. As shown in Fig. 1, CUs can be further split into two PU partition modes for intra prediction. CUs can also be split into TUs by using a generic quad-tree segmentation structure. Instead of applying the intra coding at PU level, HEVC conducts intra prediction in TU level sequentially, which always utilize the nearest neighboring reference samples from the already reconstructed TUs. There are only nine intra modes available for 4×4 luma blocks in H.264/AVC [18]. To enhance the coding efficiency of HEVC, HEVC provides as many as 35 prediction modes [1]. All the prediction modes utilize the same basic set of reference samples from above and to the left of the image block. Just like H.264/AVC, left, above, and above-right neighboring reconstructed sample will be used for intra prediction. What's more, below-left neighboring reconstructed samples are rarely available in the traditional macroblock based H.264/AVC coding structure, but they are also used for HEVC intra prediction because hierarchical coding architecture makes them available more frequently. For RD-based inter prediction coding tools, HEVC adopts motion vector competition mechanism, that the best motion vector predictor is selected from a given advanced motion vector prediction candidate list (AMVPCL). As shown in Fig. 1, CUs can be further split into eight PU partition modes for inter prediction. Each PU has AMVPCLs for every available reference frame. The AMVPCL is composed of both spatial candidates and temporal candidates. Spatial candidates are classified into top and left categories, which need the motion information of neighboring left, left-down, upper, upper-left and upper-right PUs. AMVPCL derivation process has to be done sequentially on both the encoder and the decoder sides. According to RD-based intra/inter prediction, we find that the search of the current CU branch may have data dependencies on its neighboring left, left-down, upper, upper-left and upper-right CU branches.

III. Highly Parallel Framework for HEVC CUPTD

In this section, we will present the proposed parallel framework. Firstly, we will formulate the problem of HEVC CUPTD. Then we will analyze the dependencies in CTU-level and use the directed acyclic graph (DAG)-based order to parallelize CTUs as described in our previous work [14]. After that, we exploit the implicit CU-level parallelism.

A. Problem Formulation

Each frame is divided into non-overlapping CTUs, where V_{io} denotes i_o-th CTU within the frame. Let M denotes the maximum depth of the CTU. Let m_o denotes a parameter for deciding the minimum CU size. The minimum CU size is $2^{m0+1} \times 2^{m0+1}$ and the maximum CU size is $2^{m0+M} \times 2^{m0+M}$. The CU at depth m, where $0 <= m < M$, is of size $2^{m0+(M-m)} \times 2^{m0+(M-m)}$. Fig. 2 shows an example of the formulation for HEVC CUPTD. In Fig. 2, $M = 4$ and $m_o = 2$, the minimum CU size is 8×8 and the maximum CU size is 64×64. Each CU at depth m is denoted by $V_{io,i1,\ldots im}$. i_o indexes the location of the root CTU within the frame, $0 <= i_1, \ldots, i_m <= 3$. We use V_{im}^- to denote $V_{io,i1,\ldots im}$. We denote $G(V_{im}^-)$ to be the best RD cost computed for the CU, V_{im}^-, assuming that V_{im}^- is not split into sub-CUs. We also denote $H(V_{im}^-)$ to be the best RD cost computed for the CU, V_{im}^-, without any restriction on whether it is split or not. HM-7.0 encoder optimizes the RD cost for V_{im}^- by using the following recursive relationship (see (1), shown at the bottom of the page), where H_0 and H_1 represents the overhead of not splitting the CU and splitting the CU respectively. HM-7.0 encoder tries to compute the best RD cost starting from $H(V_{i0})$.

$$H(V_{im}^-) = \begin{cases} \min\{H0 + G(V_{im}^-), H1 + \sum_{im+1=0}^{3} H(V_{im}^-), im+1)\} & if \quad m < M-1 \\ H0 + G(V_{im}^-) & otherwise \end{cases} \tag{1}$$

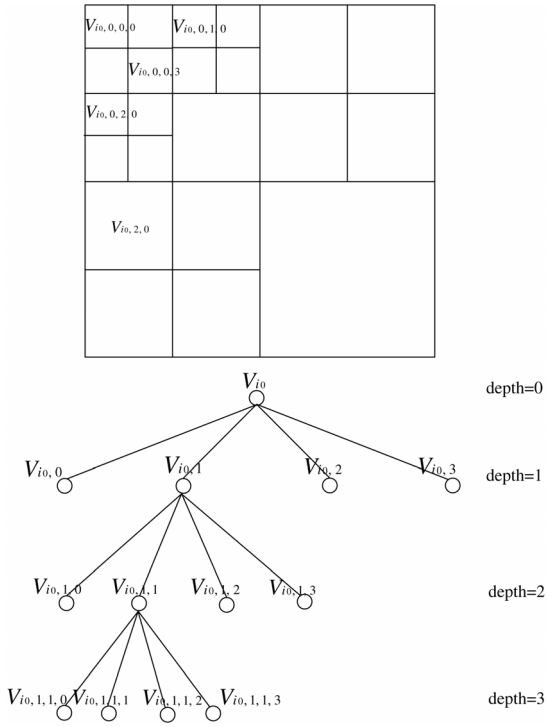

Fig. 2. Example of the formulation for HEVC CUPTD, where each CTU V_{io} is recursively split into sub-CUs $V_{io,i1,...im}$.

B. CTU-Level Parallelism

In this section, we will analyze the data dependencies among neighboring CTUs. When computing the $H(V_{i0})$ of the current CTU V_{io}, the $H(V_{i1})$ of the current CTU's neighboring left-down CTU V_{i1} isn't computed yet. So the current CTU has no data dependency on its adjacent left-down CTU. Meanwhile, the best RD costs in the current CTU's neighboring left, upper, upper-left, and upper-right CTUs are computed. The current CTU has data dependencies on its neighboring left, upper, upper-left, and upper-right CTUs. When processing the current CTU, the left, upper, upper-left and upper-right CTUs should have been completed processed. After analyzing the data dependencies among neighboring CTUs, we use the same DAG-based order as described in our previous work [14] to parallelize CTUs, which exploits the CTU-level parallelism.

C. CU-Level Parallelism

After using the DAG-based order to parallelize the CTUs, we exploit the implicit CU-level parallelism within each CTU. On the basis of analyzing the Data dependencies among neighboring CTUs, we define CICU and PICU, which satisfy certain conditions and thus can provide more parallel flexibility. When computing the $H(V_{i0})$ of the current CTU V_{io}, the left, upper, upper-left and upper-right CTUs should have been completely decided RD-based inter/intra modes. CICUs meet the following conditions:

- The CICU's left boundary and CTU's left boundary overlap.
- The CICU's upper boundary and CTU's upper boundary overlap.

The current CU at depth m ($V_{io,i1,...im}$) is CICU if the following condition satisfies:

$$\sum_{k=1}^{m} ik = 0 \tag{2}$$

The $G(V_{im}^{-})$ of CICU V_{im}^{-} has no dependency on other CUs within the same CTU. For example, as shown in Fig. 2, CU$V_{i0,0}$ and $V_{i0,0,0}$ meet requirements of CICU. Their neighboring related CTUs have been processed. The $G(V_{i0,0})$ and $G(V_{i0,0,0})$ can be computed at first in parallel.

We further define PICUs, which meet the following conditions:

- PICUs don't meet requirements of CICUs.
- The PICU's left boundary and CTU's left boundary overlap or the $H(V_{im}^{-})$ of neighboring left largest size CU V_{im}^{-} has been computed.
- The PICU's upper boundary and CTU's upper boundary overlap or the $H(V_{im}^{-})$ of neighboring upper and upper-right largest size CUs V_{im}^{-} have been computed.

Within the same CTU, when the CICU have been processed in parallel at first, PICU will appear and can be processed in parallel. For example, as shown in Fig. 2, when the $H(V_{i0,0})$ of the CU$V_{i0,0}$ has been computed, CU $V_{i0,1,0}$ and $V_{i0,2,0}$ meet the requirements of PICU. The $G(V_{i0,1,0})$ and $G(V_{i0,2,0})$ can be computed in parallel. So the $G(V_{im}^{-})$ of PICU V_{im}^{-} can be computed when their related CU have decided how to split.

IV. Experimental Results

A. Input Stream and Environment Conditions

To compare our proposed method with serial execution, we adopt an encoder migrated from HEVC reference software HM7.0 [19] without any optimization. The input videos in our experiments contain a list of standard test sequences with 64 frames. We select the profile 'randomaccess_main'. The default encoding test conditions are specified in [19]. The experiment platform of this letter is based on Tile64, which is a member of TILERA many-core platform and contains 64 processing cores [17]. In order to avoid the impact of special platform, we do not use any Tile64 platform-dependent optimizations.

B. Speedup Analysis

Fig. 3 shows the speedup of CTU-level parallelism and our proposed method compared to serial execution using 64 cores. The speedup of our proposed method and CTU-level parallelism can be calculated as follows:

$$S_{\mathrm{Pr}\,oposed} = \frac{T_{serial}}{T_{\mathrm{Pr}\,oposed}} \tag{3}$$

$$S_{CTU} = \frac{T_{serial}}{T_{CTU}} \tag{4}$$

where $\mathrm{T}_{proposed}$, T_{CTU} and T_{serial} are respectively the CUPTD time of serial execution, CTU-level parallelism and proposed method. Table I shows the speedup of our proposed method compared to serial execution using 64 cores. From Fig. 3 and Table I, We get two major observations:

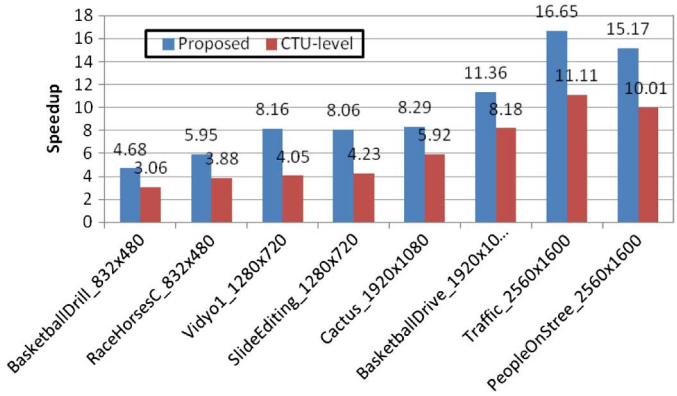

Fig. 3. The speedup of CTU-level parallelism and our proposed method compared to serial execution using 64 cores, $QP = 32$.

TABLE I
THE SPEEDUP OF OUR PROPOSED METHOD COMPARED TO SERIAL EXECUTION USING 64 CORES

Resolution of Sequences	Sequences	Speedup QP=22	Speedup QP=32	Average Speedup
832x480	BasketballDrill	5.44	4.68	**5.32**
	RaceHorsesC	6.22	5.95	
	BQMall	4.98	4.41	
	BasketballDrillText	5.68	5.21	
1280x720	Vidyo1	9.55	8.16	**8.31**
	SlideEditing	8.92	8.06	
	SlideShow	8.32	7.26	
	Vidyo3	8.63	7.52	
1920x1080	Cactus	9.95	8.29	**11.20**
	BasketballDrive	14.24	11.36	
	Kimono	11.28	9.12	
	ParkScene	13.82	11.51	
2560x1600	Traffic	19.66	16.65	**16.45**
	PeopleOnStreet	17.13	15.17	
	Nebuta	15.86	12.72	
	SteamLocomotive	18.55	15.81	

- As the resolution of frame increases, the speedup of CTU-level parallelism and our proposed method increases because the degree of parallelism increases [14].
- Our proposed method accelerates a lot more than serial execution and CTU-level parallelism. Compared with serial execution, our proposed method achieves averagely more than 11 and 16 times speedup for 1920x1080 and 2560x1600 video sequences, respectively.

V. CONCLUSIONS

HEVC CUPTD greatly increases the computational complexity at the HM-7.0 encoder. We propose an efficient parallel framework for HEVC CUPTD on many-core processors. After analyzing the dependencies among neighboring CTUs within the same frame, we firstly use the DAG-based order to parallelize CTUs as described in our previous work [14]. In order to further increase the degree of parallelism overhead, we process the CICUs at the beginning of processing each CTU and the PICUs when their related CUs have been processed, which exploit the implicit CU-level parallelism. Experiments conducted on Tile64 platform demonstrate that our method saves more time than the default encoding scheme in HM 7.0.

REFERENCES

[1] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.
[2] R. Sjöberg, Y. Chen, A. Fujibayashi, M. M. Hannuksela, J. Samuelsson, T. K. Tan, Y.-K. Wang, and S. Wenger, "Overview of hevc high-level syntax and reference picture management," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1858–1870, Dec. 2012.
[3] M. Zhou, W. Gao, M. Jiang, and H. Yu, "HEVC lossless coding and improvements," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1839–1843, Dec. 2012.
[4] C. Yeo, Y. Tan, and Z. Li, "Dynamic range analysis in HEVC residual coding and reconstruction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 7, pp. 1131–1136, Jul. 2013.
[5] J. Ohm, G. J. Sullivan, and H. Schwarz *et al.*, "Comparison of the coding efficiency of video coding standards—including high efficiency video coding (HEVC)," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1669–1684, Dec. 2012.
[6] Y. Yuan, I.-K. Kim, X. Zheng, L. Liu, and X. Cao *et al.*, "Quadtree based non-square block structure for inter frame coding in HEVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1707–1719, Dec. 2012.
[7] J. Vanne, M. Viitanen, T. D. Hamalainen, and A. Hallapuro, "Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1885–1898, Dec. 2012.
[8] F. Bossen, B. Bross, K. Sühring, and D. Flynn, "HEVC complexity and implementation analysis," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1685–1696, Dec. 2012.
[9] G. Correa, P. Assuncao, L. Agostini, and L. A. da Silva Cruz, "Performance and computational complexity assessment of high-efficiency video encoders," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1899–1909, Dec. 2012.
[10] L. Shen, Z. Liu, and X. Zhang *et al.*, "An effective CU size decision method for HEVC encoders," *IEEE Trans. Multimedia*, vol. 15, pp. 465–470, Jan. 2013.
[11] 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 Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1827–1838, Dec. 2012.
[12] Z. Xiao and B. M. Baas, "A 1080p H.264/AVC baseline residual encoder for a fine-grained many-core system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 7, pp. 890–902, Jul. 2011.
[13] Y. Zhang *et al.*, "Efficient parallel framework for H.264/AVC deblocking filter on many-core platform," *IEEE Trans. Multimedia*, vol. 14, no. 3, pp. 510–524, 2012.
[14] C. Yan *et al.*, "Highly parallel framework for HEVC motion estimation on many-core platform," in *Data Compression Conf.*, Snowbird, UT, 2013, pp. 63–72.
[15] C. Yan *et al.*, "Parallel deblocking filter for HEVC on many-core processor," *Electron. Lett.*, accepted for publication.
[16] C. Yan *et al.*, "Parallel deblocking filter for H.264/AVC implemented on Tile64 platform," in *Int. Conf. Multimedia and Expo (ICME)*, Barcelona, Spain, 2011, pp. 1–6.
[17] S. Bell *et al.*, "TILE64-Processor: A 64-core SoC with mesh," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2008, pp. 88–598.
[18] T. Wiegand, G. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.
[19] F. Bossen, "Common test conditions and software reference configurations," in *JCTVC-I1100*, Apr. 2012.