Efficient Parallel Framework for HEVC Motion Estimation on Many-Core Processors

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

Abstract—**High Efficiency Video Coding (HEVC) provides superior coding efficiency than previous video coding standards at the cost of increasing encoding complexity. The complexity increase of motion estimation (ME) procedure is rather significant, especially when considering the complicated partitioning structure of HEVC. To fully exploit the coding efficiency brought by HEVC requires a huge amount of computations. In this paper, we analyze the ME structure in HEVC and propose a parallel framework to decouple ME for different partitions on many-core processors. Based on local parallel method (LPM), we first use the directed acyclic graph (DAG)-based order to parallelize coding tree units (CTUs) and adopt improved LPM (ILPM) within each CTU (DAGILPM), which exploits the CTU-level and prediction unit (PU)-level parallelism. Then, we find that there exist completely independent PUs (CIPUs) and partially independent PUs (PIPUs). When the degree of parallelism (DP) is smaller than the maximum DP of DAGILPM, we process the CIPUs and PIPUs, which further increases the DP. The data dependencies and coding efficiency stay the same as LPM. Experiments show that on a 64-core system, compared with serial execution, our proposed scheme achieves more than 30 and 40 times speedup for 1920 × 1080 and 2560 × 1600 video sequences, respectively.**

Index Terms—**Coding efficiency, degree of parallelism (DP), efficient parallel framework, High Efficiency Video Coding (HEVC), many-core processors, motion estimation (ME).**

I. Introduction

HIGH Efficiency Video Coding (HEVC) is the state-of-the-art video coding standard [2]–[5]. Compared with H.264/Advanced Video Coding (AVC), HEVC doubles the coding efficiency [6], which is largely benefited from more sophisticated motion estimation (ME) design [7], [8]. The price to be paid for higher coding efficiency is higher computational complexity. The HEVC encoders are expected to be several times more complex than H.264/AVC encoders [9], [10]. The HEVC ME is the most computationally expensive operation in the HEVC encoder [11]. Video coding has been restricted in many fields because of its high complexity [12]–[19]. As a result, it is important to accelerate HEVC, especially ME.

We are witnessing a paradigm shift in computer architecture toward many-core processors [20]–[23], which are good candidates for speeding up HEVC ME. A central question is whether HEVC ME can scale to such a large number of cores. If HEVC ME is not extensively parallelizable, cores will be left unused and its performance might suffer. Efficient parallelization of ME on many-core processors is challenging, because ME has complicated data dependencies which provides insufficient degree of parallelism (DP) for so many cores [24], [25]. In addition, parallelization may have significant coding efficiency loss [26]–[28].

In general, there are two ways to parallelize ME on many-core processors: 1) global parallel method (GPM) [26]–[28] and 2) local parallel method (LPM) [24], [25]. The GPM provides a high DP but has nonignorable coding efficiency loss; LPM has ignorable coding efficiency loss but the DP of LPM is not adequate for many-core processors. The GPM is widely adopted for H.264/AVC ME [26]–[28]. First, ME is carried out in parallel for all the 4×4 submacroblocks (MBs) within the same frame. Then, ME of other sub-MB partitions can be obtained in parallel by the ME of 4×4 sub-MB. The GPM eliminates the data dependencies among blocks within the same frame and provides a high DP for H.264/AVC ME. However, GPM takes no account of the data dependencies among the block partitions, which has nonignorable coding efficiency loss. The HEVC ME is highly sequential and has a highly flexible hierarchy of unit representation [29]. If we apply GPM directly to HEVC ME, it will lead to significant coding efficiency loss. The HEVC includes three block concepts [29]: 1) coding tree unit (CTU); 2) coding unit (CU); and 3) prediction unit (PU). The PUs are the basic units used for ME. The LPM introduces the concept of ME region (MER) [24], [25] and divides each CTU into a number of nonoverlapped parallel MERs. From MER to MER, ME is carried out sequentially. Within each MER, ME is carried out in parallel for all the PUs. The LPM has been adopted into the HEVC standard. The LPM eliminates the data dependencies among PUs within the same MER, which has little coding efficiency loss. However, the MERs and CTUs have to be processed sequentially. The maximum DP of LPM is still very insufficient to keep the coding efficiency from losing too much.

Manuscript received November 4, 2013; revised February 24, 2014, May 21, 2014, and June 20, 2014; accepted June 30, 2014. Date of publication July 8, 2014; date of current version December 3, 2014. This work was supported in part by the National Key Technology Research and Development Program of China under Grant 2012BAH06B01 and in part by the National Nature Science Foundation of China under Grants 61472203, 61272323, 61102101, and 61379084. This paper was recommended by Associate Editor W. Zeng.

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

Y. Zhang, F. Dai, and J. Zhang are with the Key Laboratory of Intelligent Information Processing, Chinese Academy of Sciences, Institute of Computing Technology, Beijing 100190, China (e-mail: 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/TCSVT.2014.2335852

1051-8215 © 2014 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

For example, in the HEVC setting, the maximum DP is 8 which is not adequate for many-core processors.

On the premise of keeping data dependencies the same as LPM, we further analyze the dependencies in different levels of data granularity within the same frame. Based on LPM, we propose an efficient parallel framework for HEVC ME, which largely increases the DP than LPM. Meanwhile, the coding efficiency of our method stays the same as LPM, which is much better than GPM.

1) We use the directed acyclic graph (DAG)-based order to parallelize CTUs and adopt improved LPM (ILPM) within each CTU (DAGILPM). The data dependencies among neighboring CTUs are caused by the PUs. The current CTU has data dependencies on its neighboring left, upper, upper-left, and upper-right CTUs. We generate a DAG [30], [31] to capture the dependency relationships among neighboring CTUs. We use the DAG-based order to parallelize CTUs, which exploits the implicit CTU-level parallelism. Meanwhile, we adopt ILPM within each CTU, which increases the PU-level parallelism than LPM. The maximum DP of DAGILPM reaches 495 and 660 for 1920 × 1080 and 2560 × 1600 video sequences, respectively.
2) After DAGILPM, there is an insufficient DP at the beginning and at the end of processing a frame. After carefully reviewing all the PUs, we find that there exist completely independent PUs (CIPUs) and partially independent PUs (PIPUs). The CIPUs have no data dependencies on other PUs within the same frame. The PIPUs have no data dependencies on other PUs within the same CTU. When the DP is smaller than the maximum DP of DAGILPM, we process the CIPUs and PIPUs, which further increases the DP. The data dependencies and coding efficiency stay the same as LPM.

A part of this paper has been presented in [1]. Based on our previous work [1], we further use ILPM and find the PIPUs. The DAG-based order was also used for deblocking filter [14], intra-mode decision [31], [32], and CU partitioning tree decision [33]. This paper uses the DAG-based order to parallelize ME. Our proposed parallel framework is suitable to many-core processors. Our testing processor is a Tile64 [32]. Experiments demonstrate that our proposed method significantly improves the performance compared with GPM and LPM.

The rest of this paper is organized as follows. Section II gives a review of HEVC ME and related work. Section III presents the proposed highly parallel framework for HEVC ME. The experimental results are elaborated in Section IV. Finally, Section V concludes this paper.

II. HEVC ME and Related Work

A. HEVC Motion Estimation

The HEVC provides a highly flexible hierarchy of unit representation for ME, which includes three block concepts [29]: CTU, CU, and PU (Fig. 1). Hierarchy of unit representation has been proved effective [34]–[36]. Each frame is divided into CTUs, which can be recursively split into smaller CUs by using a generic quadtree segmentation structure. The CU can be further split into PUs, which have eight partition modes used for ME. The PUs are the basic units used for carrying the motion data related to ME. If a neighboring PU is coded, it will be available for the current PU. The current PU may have data dependencies on its neighboring left, left-down, upper, upper-left, and upper-right PUs, whose motion data may be available for the current PU [2].

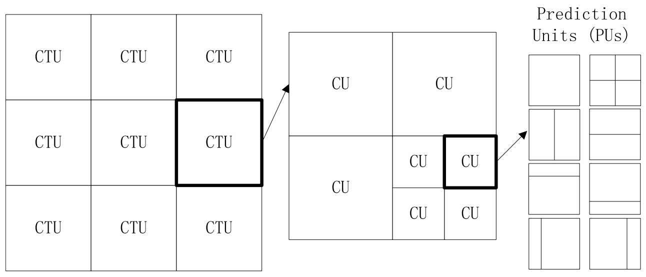


Fig. 1. Flexible hierarchy of unit representation for ME.

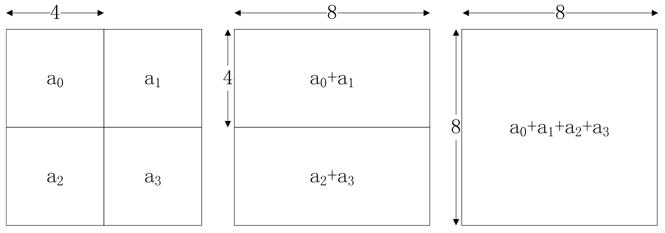


Fig. 2. GPM for H.264/AVC ME. First, SAD of all 4×4 sub-MB partitions are calculated. SAD of $8 \times 4, 8 \times 8, \ldots$ sub-MB partitions are obtained by the summation of different combination of SAD of 4×4 sub-MB partitions.

B. Related Work

The GPM eliminates the data dependencies among all the blocks within the frame, which provides a high DP and has nonignorable coding efficiency loss. The LPM eliminates the data dependencies among all the blocks within the same MER, which has ignorable coding efficiency loss. But, the DP of LPM is not adequate for many-core processors.

1) GPM for H.264/AVC ME: Parallelization of ME is an afterthought in H.264/AVC, where GPM is widely adopted [26]–[28]. H.264/AVC supports MB partitioning with variable block sizes. The luma component of each MB could be divided into four partition modes: 16 × 16, 8 × 16, 16 × 8, and 8 × 8 pixels. As shown in Fig. 2, if an 8 × 8 partition is selected, each 8 × 8 block can be further divided into sub-MB partitions: 8 × 8, 4 × 8, 8 × 4, and 4 × 4 pixels. The GPM first calculates the sum of absolute differences (SAD) values of all 4 × 4 sub-MB partitions within a frame. Then, the SAD values of $8 \times 4, 8 \times 8, \ldots$ sub-MB partitions are obtained by the summation of different combination of SAD of 4 × 4 sub-MB partitions. Then, GPM chooses the best ME mode among all the candidates. All the 4 × 4 SAD calculation can be processed in parallel and GPM provides a high DP for ME. However, GPM takes no account of the data dependencies among the block partitions, which has nonignorable coding efficiency loss.

2) LPM for HEVC ME: If we apply GPM directly to HEVC ME, all the PUs are unavailable for each other within the frame. The GPM leads to significant coding efficiency loss

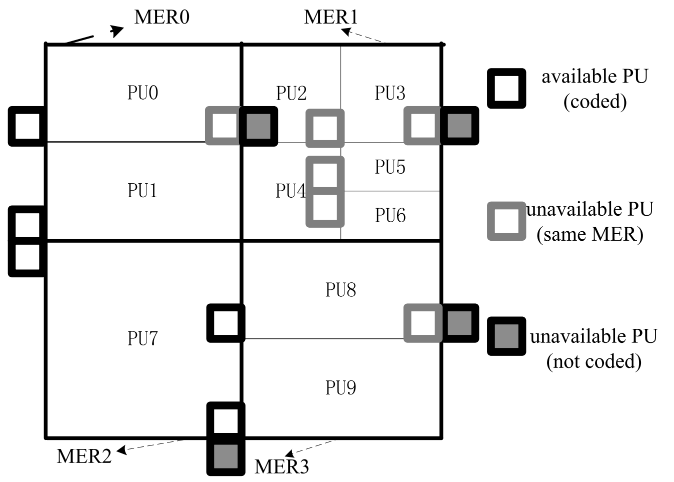


Fig. 3. Example of MER and available PUs for PU1, PU5, and PU9.

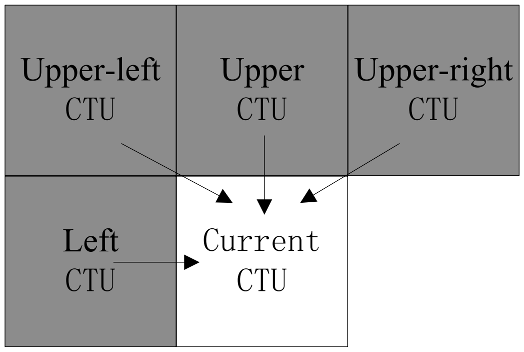


Fig. 4. Data dependencies among neighboring CTUs. The arrows indicate dependencies.

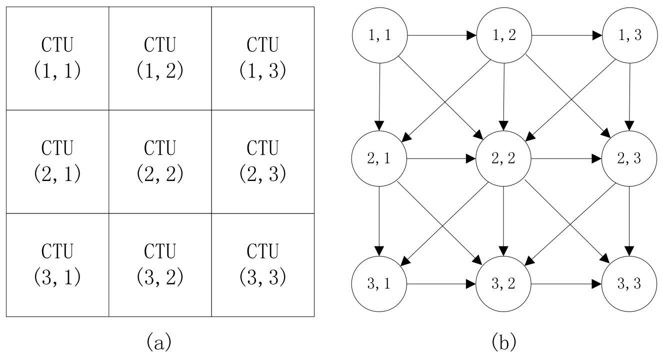


Fig. 5. (a) Each CTU in the frame is mapped into a point in a 2-D coordinate plane. (b) DAG for representing the dependency relationships of CTUs.

although it provides a high DP. The LPM [24], [25] introduces the concept of MER and divides each CTU into a number of nonoverlapped MERs. All the MERs are exact square shapes with the same size. The LPM introduces a new availability rule for PUs.

Fig. 3 shows an example of MER and available PUs for PU1, PU5, and PU9. There are four MERs within the CTU. From MER0 to MER3, ME is carried out sequentially. All the PUs residing in the same MER are unavailable for each other. Within the same MER, ME is carried out in parallel for all the PUs. For example, PU8 and PU9 are within MER3 and unavailable for each other. When processing MER3, PU8 and PU9 have no data dependencies on each other. So, they can be processed in parallel. The maximum DP of LMP ($\mathrm{MP_{LPM}}$) can be expressed as

$$\mathrm{MP_{LPM}} = \frac{S(\mathrm{MER})}{\min(S(\mathrm{PU}))} = \frac{M \times M}{32} \tag{1}$$

where M is the length of MER, $S($MER) indicates the pixel number of MER, and $S($PU) indicates the pixel number of PU. When the size of PU is 8×4 or 4×8, $S($PU) will reach the minimum. In order to guarantee the coding efficiency, M is commonly equal to 16 or 8 today [24], [25]. So, the $\mathrm{MP_{LPM}}$ is equal or less than 8. Thus, LPM cannot provide a sufficient DP for many-core processors.

III. Highly Parallel Framework for HEVC ME

In this section, on the premise of keeping data dependencies and coding efficiency the same as LPM, we will first generate a DAG [30] to capture the dependency relationships among neighboring CTUs. We will use the DAG-based order to parallelize CTUs and adopt ILPM within each CTU (DAGILPM). Then, we will find the CIPUs and PIPUs. When the DP is smaller than the maximum DP of DAGILPM, we will process the CIPUs and PIPUs.

A. DAG Improved LPM

1) Data Dependencies Among Neighboring CTUs: In this section, we will analyze the data dependencies among neighboring CTUs. The CTUs are processed in row scanning order. The data dependencies among neighboring CTUs are caused by the PUs. When processing the current CTU, the PUs within the current CTUs neighboring left-down CTU are not coded yet, which will be unavailable for the PUs in the current CTU. So, the current CTU has no data dependency on its adjacent left-down CTU. Meanwhile, the PUs in the current CTUs neighboring left, upper, upper-left, and upper-right CTUs are coded. The current CTU has data dependencies on its neighboring left, upper, upper-left, and upper-right CTUs (Fig. 4). The arrows indicate dependencies. When processing the current CTU, the left, upper, upper-left, and upper-right CTUs should have been completely processed if they exist.

2) DAG for CTUs: After analyzing the data dependencies among neighboring CTUs, we generate a DAG to capture the dependency relationships of CTUs. As shown in Fig. 5(a), we first map each CTU in the frame into a point in a 2-D coordinate plane as

$$i = \mathrm{ceil}\left(\frac{k}{W}\right) \tag{2}$$

$$j = k \bmod W \tag{3}$$

where i is coordinate value of the horizontal axis, j is coordinate value of the vertical axis, k is the time stamp of the CTU, W is the horizontal CTU number of the frame, and the ceil function returns the value of a number rounded upward to the nearest integer.

Then, we use a DAG to represent the execution flow of the CTUs and the precedence constraints among the CTUs [30]. As shown in Fig. 5(b), the DAG is marked as $G = (V, E)$, which consists of a set of vertices V and edges E. Vertices

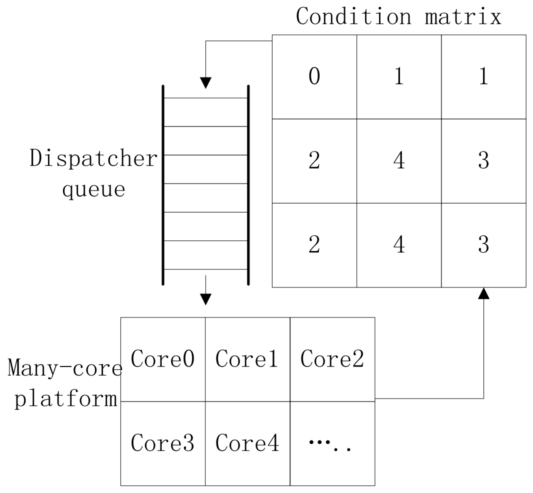


Fig. 6. DAG-based order to parallelize CTUs.

are numbered according to the coordinate values of CTUs in the 2-D coordinate plane. For example, vertex $v_{i,j}$ in Fig. 5(b) represents the CTU with coordinate values (i, j) in Fig. 5(a). If vertex $v_{m,n}$ has data dependency on $v_{i,j}$, vertex $v_{i,j}$ is a parent of vertex $v_{m,n}$ and there will exist an edge $(v_{i,j}, v_{m,n}) \in E$. When vertex $v_{i,j}$ is processed, vertex $v_{i,j}$ and edge $(v_{i,j}, v_{m,n})$ will be removed from the DAG. The precedence constraint means that when the in-degrees of some vertices are zero, these vertices can be processed in parallel. In order to parallelize the vertices, it is important to record and update the in-degrees of all the vertices. We get the initial values of the in-degrees by the adjacency matrix. We generate the adjacency matrix A of the DAG as

$$A_{(i,j),(m,n)} = \begin{cases} 1, & (v_{i,j}, v_{m,n}) \in E \\ 0, & \text{otherwise} \end{cases}$$
$$\text{s.t.} \quad 1 \le i, m \le H \quad 1 \le j, n \le W \tag{4}$$

where H is the vertical CTU number of each frame, and A is a 2-D matrix.

The initial in-degree $D_{m,n}$ of vertex $v_{m,n}$ in the DAG can be summarized as

$$D_{m,n} = \sum_{i=1}^{H} \sum_{j=1}^{W} A_{(i,j),(m,n)}$$
$$\text{s.t.} \quad 1 \le m \le H \quad 1 \le n \le W \tag{5}$$

where D is a 2-D matrix, which represents the initial state of the in-degrees of the DAG.

3) Parallelizing CTUs Using DAG-Based Order: After getting the initial values of the in-degrees, we use a DAG-based order to parallelize the CTUs as shown in Fig. 6. The condition matrix (CM) is a 2-D matrix, which is designed to record the number of related CTUs for each CTU. The initial value of CM is set equal to D. When some entries in CM are zero, the corresponding CTUs can be processed in parallel. When a CTU with coordinate (i, j) in CM is processed, the entries of coordinates $(i + 1, j)$, $(i+1, j-1)$, $(i, j+1)$, and $(i+1, j+1)$ in CM will minus one. Furthermore, the dispatcher queue (DQ) is a waiting queue, whose elements are the coordinates of available CTUs. The pseudocode of the DAG-based order is expressed as follows.

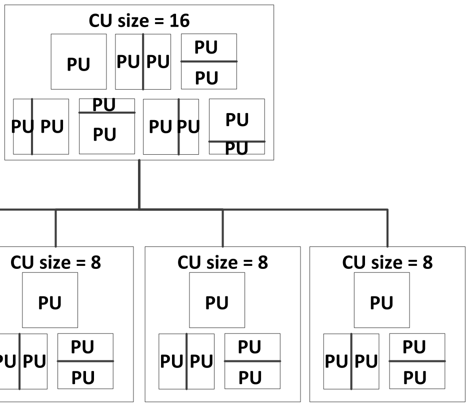


Fig. 7. Example of ILPM. The length of MER is 16. Within the same MER, all the PUs with different partition modes in different CU depths can be processed in parallel.

1) *Step 1:* Initialize DQ and CM. DQ is a waiting queue. CM is designed to record the number of related CTUs for each CTU.
2) *Step 2:* When some entries in CM become zero, get the corresponding coordinates and push them into DQ.
3) *Step 3:* Get coordinates from DQ and process corresponding CTUs in parallel on many-core processor.
4) *Step 4:* Update CM. When a CTU with coordinate (i, j) in CM is processed, the entries of coordinates $(i+1, j)$, $(i+1, j\text{-}1)$, $(i, j+1)$, and $(i+1, j+1)$ in CM will minus one.
5) *Step 5:* Repeat above steps 2–4 until each frame is over.

B. Improved LPM

After using the DAG-based order to parallelize CTUs, we adopt ILPM within each CTU, which exploits the implicit PU-level parallelism. The ILPM is based on LPM [24], [25]. The LPM introduces the concept of MER and divides each CTU into a number of nonoverlapped MERs. All the MERs are exact square shapes with the same size. When a CTU is being processed, the MERs within the CTU are being processed sequentially. The CTUs can be recursively split into smaller CUs by using a generic quadtree segmentation structure. The CU has at most eight partition modes for PUs. Within the same MER, the PUs with the particular partition mode can be processed in parallel. We go further than LPM. We mark the MER as the root node of a quadtree. Then, we generate the depth of the quadtree (N) as

$$N = \log_2 M - 3 \tag{6}$$

where M is the length of MER.

Within the same MER, CUs can be recursively split into smaller CUs using a generic quadtree segmentation structure. All the PUs with different partition modes in different CU depth can be processed in parallel. Fig. 7 shows an example of ILPM, where the length of MER is 16. There are three and seven partition modes used in 8×8 and 16×16 CU, respectively. Because all the partition modes belong to the same MER, they are unavailable for each other and have no data dependencies. The ME is carried out in parallel for all the partition modes.

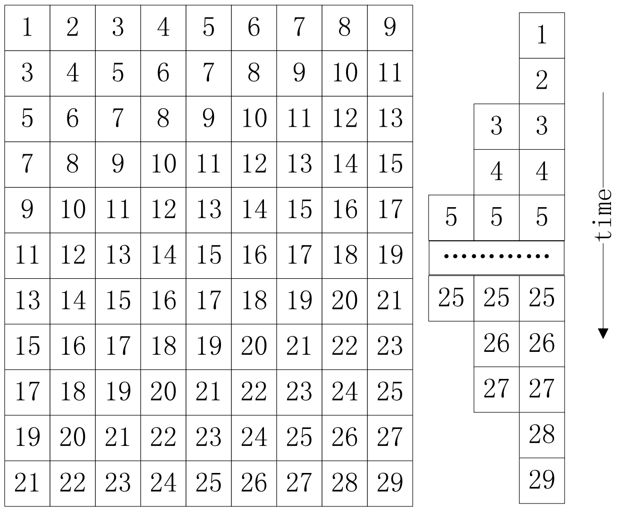


Fig. 8. Example of DAG-based order to parallelize the CTUs for a frame.

The DP of ILPM can be described as

$$f(n) = 4 * f(n+1) + P_n \tag{7}$$

$$P_n = \begin{cases} 5 & n = 3 \\ 13 & n < 3 \end{cases} \tag{8}$$

$$f(N) = P_N \tag{9}$$

where $f(n)$ is the total DP when the depth of the coding tree is set to n, and P_n is the DP of the given CU size at the nth level. For example, as shown in Fig. 7, N and $f(N)$ are 1 and 5, respectively. The total DP of the quadtree is 33, which is much larger than that of LPM.

C. Parallelism Analysis of DAGILPM

The processing time of CTUs is different from each other. In order to analyze the maximum DP conveniently, we assume all the processing time of CTUs is the same. An example of DAG-based order to parallelize the CTUs for a frame is illustrated in Fig. 8. Rectangles represent CTUs, which are processed according to their numbers. The CTUs with the same numbers are processed concurrently. The maximum DP of CTU ($\mathrm{MP_{CTU}}$) can be calculated as

$$\mathrm{MP_{CTU}} = \min\left(\mathrm{ceil}\left(\frac{W}{2}\right), H\right). \tag{10}$$

After adopting ILPM within each MER, the maximum DP of DAGILPM ($\mathrm{MP_{DAGILPM}}$) can be summarized as

$$\mathrm{MP_{DAGILPM}} = \mathrm{MP_{CTU}} \times \mathrm{MP_{ILPM}}. \tag{11}$$

Table I compares the maximum DP of LPM with that of DAGILPM. The size of CTU is usually set as 64 × 64 [2]. $\mathrm{MP_{DAGILPM}}$ is much larger than $\mathrm{MP_{LPM}}$. When the size of MER is 16 × 16, $\mathrm{MP_{DAGILPM}}$ reaches 495 and 660 for 1920 × 1080 and 2560 × 1600 video sequences, respectively.

From Fig. 8, we find that the DP of DAGILPM is low at the beginning and at the end of processing a frame, which is not believed to be adequate for many-core processors. There are many idle cores at the beginning and at the end of processing a frame, which influences the performance. The average DP (ADP) can be calculated as

$$\mathrm{ADP} = \left(\frac{1 + \mathrm{MP_{CTU}}}{2}\right) \times \mathrm{MP_{LPM}}. \tag{12}$$

TABLE I
MAXIMUM DP OF LPM AND DAGILPM

Resolution of MER	Resolution of CTU	MP_{LPM}	$MP_{DAGILPM}$
8x8	832x480	2	35
8x8	1280x720	2	50
8x8	1920x1080	2	75
8x8	2560x1600	2	100
16x16	832x480	8	231
16x16	1280x720	8	330
16x16	1920x1080	8	495
16x16	2560x1600	8	660

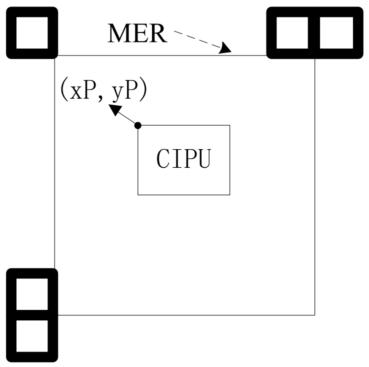


Fig. 9. MERs neighboring left, left-down, upper, upper-left, and upper-right PUs are available for PUs within the MER.

D. CIPUs and PIPUs

In order to further increase the ADP, we process the CIPUs and PIPUs when the DP is smaller than $\mathrm{MP_{DAGILPM}}$. The CIPUs have no data dependencies on other PUs within the same frame. The PIPUs have no data dependencies on other PUs within the same CTU.

1) Completely Independent PUs: On the basis of LPM, we find out CIPUs. The CTUs and MERs in a frame are processed sequentially in scan order. So only the MERs neighboring left, left-down, upper, upper-left, and upper-right PUs are available for PUs within the MER (Fig. 9). For example, PU9 has data dependencies on MER3's neighboring left and upper-left PUs (Fig. 3). If PUs within the MER have no data dependencies on the MERs neighboring left, left-down, upper, upper-left, and upper-right PUs, they will have no data dependencies on all the PUs within the frame. The CIPUs meet the following conditions.

1) A CIPUs left boundary and the corresponding MERs left boundary do not overlap.
2) A CIPUs upper boundary and the corresponding MERs upper boundary do not overlap.

The neighboring PUs of the CIPU belong to the same MER or are not coded, which are unavailable for the CIPU. For example, as shown in Fig. 3, PU5 meets requirements of CIPU. PU5 and its neighboring left, left-down, upper,

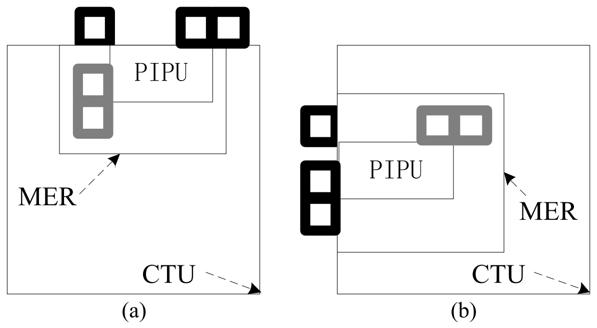


Fig. 10. (a) PIPUs left boundary and the corresponding MERs left boundary do not overlap. Meanwhile, the PIPUs upper boundary and the CTUs upper boundary overlap. (b) PIPUs upper boundary and the corresponding MERs upper boundary do not overlap. Meanwhile, the PIPUs left boundary and the CTUs left boundary overlap.

and upper-left PUs all belong to the MER1. Meanwhile, its neighboring upper-right PU is not coded yet. So, PU5 has no data dependencies on its neighboring PUs. PU5 can be processed at anytime within the frame. Let (xP, yP) be the coordinates of the top-left corner pixel of the current PU. The current PU is CIPU if the following condition satisfies:

$$xP \bmod M \neq 0 \quad \text{and} \quad yP \bmod M \neq 0 \tag{13}$$

where mod is modulus operation.

2) Partially Independent PUs: When processing the current CTU, the left, upper, upper-left, and upper-right CTUs should have been completely processed. The PUs within the neighboring left, upper, upper-left, and upper-right CTUs are coded. If the available PUs of the current PU all belong to the neighboring left, upper, upper-left, and upper-right CTUs, the current PU is a PIPU. As shown in Fig. 10, we further define PIPUs, which meet the following conditions.

1) A PIPUs left boundary and the corresponding MERs left boundary do not overlap. Meanwhile, the PIPUs upper boundary and the CTUs upper boundary overlap.
2) A PIPUs upper boundary and the corresponding MERs upper boundary do not overlap. Meanwhile, the PIPUs left boundary and the CTUs left boundary overlap.

For example, as shown in Fig. 3, the available PUs of PU1 all belong to neighboring left CTU. PU1 can be processed at anytime within the current CTU. Let (xP, yP) be the coordinates of the top-left corner pixel of the current PU. The current PU is a PIPU if one of the following conditions satisfies:

$$xP \bmod M \neq 0 \quad \text{and} \quad yP \bmod C = 0 \tag{14}$$

$$xP \bmod C = 0 \quad \text{and} \quad yP \bmod M \neq 0 \tag{15}$$

where C is the length of CTU.

E. Summarization of Proposed Method

On the premise of keeping data dependencies and coding efficiency the same as LPM, we propose a highly parallel framework for HEVC ME.

1) First, we generate a DAG to capture the dependency relationships among neighboring CTUs. We use the DAG-based order to parallelize CTUs and adopt ILPM within each CTU (DAGILPM), which exploits the implicit DP.
2) Then, we find the CIPUs and PIPUs. When the DP is smaller than the maximum DP of DAGILPM or the number of processing cores, we process the CIPUs and PIPUs, which further increases the DP.

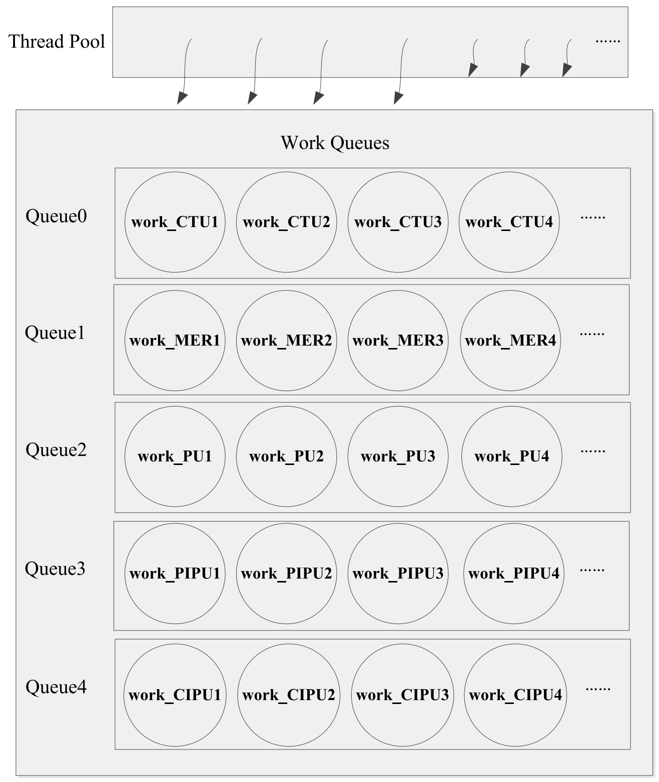


Fig. 11. Thread pool and work queues in our implementation.

IV. Experimental Results

A. Input Stream and Environment Conditions

To compare our proposed method with LPM [24] and GPM, we adopt an encoder migrated from HEVC reference software HM12.0 [37] without any optimization. The input videos in our experiments contain a list of standard test sequences with 64 frames. We select the configuration of randomaccess_main. The default encoding test conditions are specified in [37] using different quantization parameters (QPs) (22, 27, 32, and 37). If no otherwise specified, the fast search in the HM software is applied. The experiment platform of this paper is based on Tile64, which is a member of TILERA many-core processor family and contains 64 processing cores [32].

B. Implementation

In order to avoid the impact of special processors, we do not utilize any processor-dependent optimizations. As shown in Fig. 11, in order to ease the costs of frequent thread creating and destroying, we construct a thread pool at the beginning of encoding process. Thread pool is especially well suited to the TILERA architecture since the pool can be spread among the cores, with each one running a single thread. When a thread finishes its work, it will stay in the thread pool. There are

TABLE II
BD-RATE PERFORMANCE OF GPM COMPARED WITH HM12.0

Resolution of Sequences	Sequences	GPM
832x480	Flowervase	52.29%
832x480	Keiba	53.82%
832x480	Mobisode	75.30%
1280x720	KristenAndSara	51.44%
1280x720	SlideEditing	23.52%
1280x720	SlideShow	43.03%
1920x1080	Tennis	62.16%
1920x1080	BasketballDrive	80.42%
1920x1080	Kimonol	52.13%
2560x1600	Traffic	47.01%
2560x1600	PeopleOnStreet	31.06%
2560x1600	Nebuta	50.22%
	Average	**50.9%**

five work queues, denoted as queue0, queue1, queue2, queue3, and queue4. We define five kinds of works: 1) ME for a particular CTU (work_CTU); 2) ME for a particular MER (work_ MER); 3) ME for a particular CIPU (work_CIPU); 4) ME for a particular PIPU (work_PIPU); and 5) ME for a particular PU which is not a CIPU or a PIPU (work_PU). Work_CTU is put into the queue0; work_MER is put into the queue1; work_PU is put into the queue2; work_PIPU is put into the queue3; and work_CIPU is put into the queue4. All the works are scheduled in a first-in-first-out manner in the work queues. The priorities of the queues are set as queue0 > queue1 > queue2 > queue3 > queue4. Idle threads in the thread pool try to fetch the works from the queues according to their priorities. Initially, we serialize all the work_CIPUs available in the queue4. A master thread keeps track of the state of the CTUs using a CM (Section III-A) and serializes the work_CTUs available in the queue0. All the threads have access to the CM located in the shared memory. When a thread has finished the work_CTU, it will update the CM. During running process, a work itself may generate and put other works into the queues. Specifically, work_CTU may generate work_MER and work_PIPU, which will be put into queue1 and queue3, respectively. Work_MER may generate work_PU and put it into queue2. Section III shows the rule of how a work generates other works.

When we implement LPM, there are no work_CTU, work_PIPU, work_CIPU, queue0, queue3, and queue4. Meanwhile, it is different from ILPM for a work_MER to generate work_PUs (Section III-A). When we implement DAGILPM, there are no work_PIPU, work_CIPU, queue3, and queue4 (Section III-B). Serial execution just uses one processing core.

C. Coding Efficiency Analysis

The coding efficiency of all the methods is compared in terms of combined Bjøntegaard delta bitrates (BD-rate) [38], [39], which is calculated by the average PSNR of different color components. Although there are other video quality assessment approaches [40], the PSNR-based video quality measurement is widely used during the development

TABLE III
BD-RATE PERFORMANCE OF DIFFERENT METHODS COMPARED WITH HM12.0. DAGILPM MEANS THE BD-RATE PERFORMANCE OF OUR PROPOSED METHOD WITHOUT USING CIPUS AND PIPUS. THE SIZE OF MER IS 8 × 8. BD-RATE PERFORMANCES OF LPM, DAGILPM, AND OUR PROPOSED METHOD COMPARED WITH HM12.0 ARE THE SAME

Resolution of Sequences	Sequences	LPM	DAGILPM	Proposed
832x480	Flowervase	0.3%	0.3%	0.3%
832x480	Keiba	0.0%	0.0%	0.0%
832x480	Mobisode	0.0%	0.0%	0.0%
1280x720	KristenAndSara	0.3%	0.3%	0.3%
1280x720	SlideEditing	-0.4%	-0.4%	-0.4%
1280x720	SlideShow	0.1%	0.1%	0.1%
1920x1080	Tennis	0.1%	0.1%	0.1%
1920x1080	BasketballDrive	0.1%	0.1%	0.1%
1920x1080	Kimonol	0.2%	0.2%	0.2%
2560x1600	Traffic	0.4%	0.4%	0.4%
2560x1600	PeopleOnStreet	0.2%	0.2%	0.2%
2560x1600	Nebuta	0.3%	0.3%	0.3%
	Average	**0.1%**	**0.1%**	**0.1%**

TABLE IV
BD-RATE PERFORMANCE OF DIFFERENT METHODS COMPARED WITH HM12.0. THE SIZE OF MER IS 16 × 16. BD-RATE PERFORMANCES OF LPM, DAGILPM, AND OUR PROPOSED METHOD COMPARED WITH HM12.0 ARE THE SAME

Resolution of Sequences	Sequences	LPM	DAGILPM	Proposed
832x480	Flowervase	0.9%	0.9%	0.9%
832x480	Keiba	0.5%	0.5%	0.5%
832x480	Mobisode	0.6%	0.6%	0.6%
1280x720	KristenAndSara	0.7%	0.7%	0.7%
1280x720	SlideEditing	0.1%	0.1%	0.1%
1280x720	SlideShow	0.5%	0.5%	0.5%
1920x1080	Tennis	0.5%	0.5%	0.5%
1920x1080	BasketballDrive	0.6%	0.6%	0.6%
1920x1080	Kimonol	0.5%	0.5%	0.5%
2560x1600	Traffic	1.0%	1.0%	1.0%
2560x1600	PeopleOnStreet	1.2%	1.2%	1.2%
2560x1600	Nebuta	0.5%	0.5%	0.5%
	Average	**0.6%**	**0.6%**	**0.6%**

of HEVC [41]. The average PSNR is calculated as

$$\text{PSNR}_{\text{avg}} = \frac{6 * \text{PSNR}_Y + \text{PSNR}_U + \text{PSNR}_V}{8}. \tag{16}$$

The BD-rate performances of all the methods compared with HM12.0 are shown in Tables II–IV. The positive number means coding efficiency loss. We find that GPM causes the BD-rates to a serious increase of 50.9% on average, while LPM, DAGILPM, and our proposed methods have little effect on coding efficiency. The BD-rate performances of LPM, DAGILPM, and our proposed method compared with HM12.0 are the same.

Fig. 12 shows the rate-distortion encoding efficiency with different methods for different video sequences. The curves for HM12.0, LPM, and our proposed method almost overlap in the figure, which show that the rate-distortion efficiency of LPM and our proposed method is almost as high as that of HM12.0. However, the encoding efficiency of GPM is much lower than

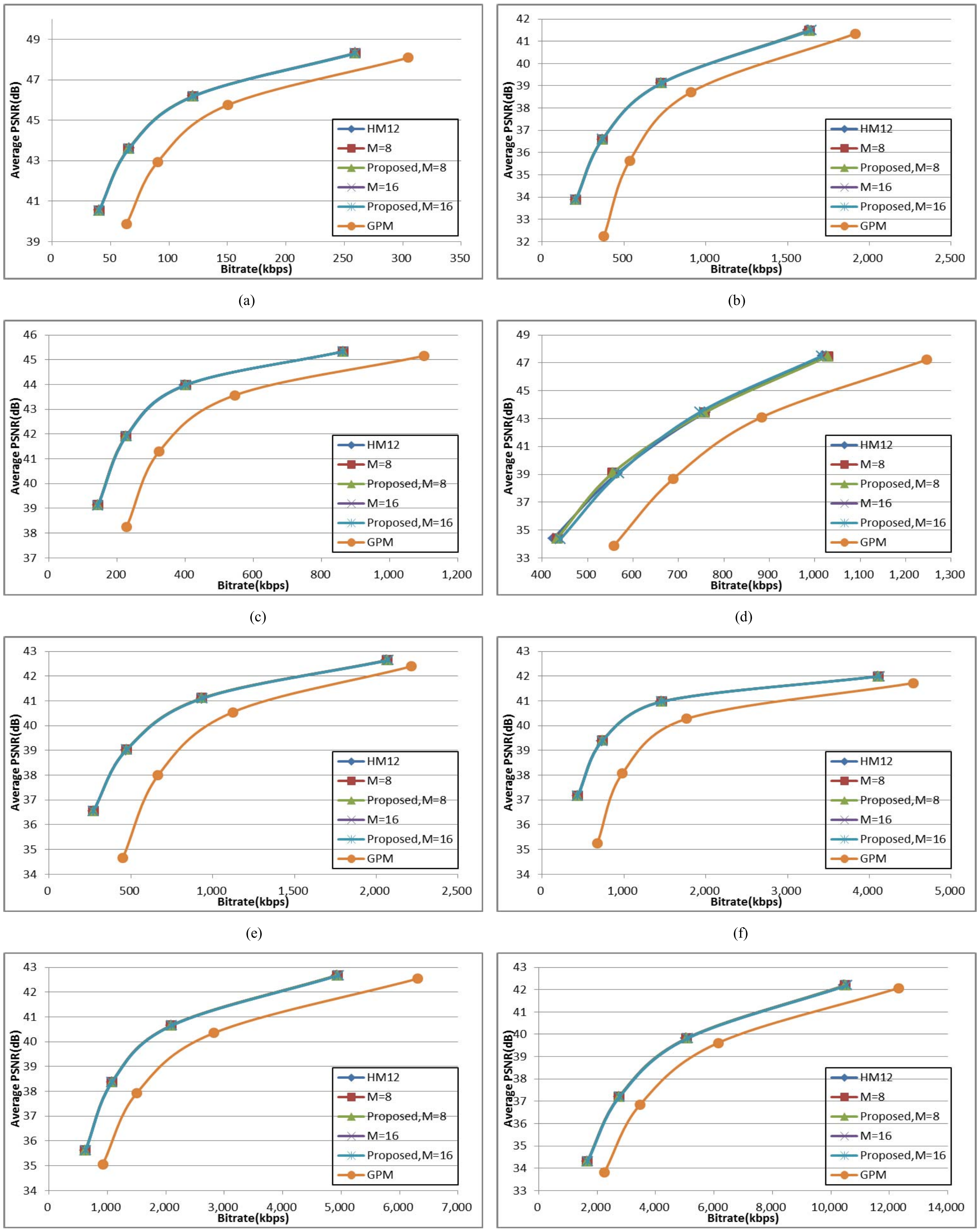


Fig. 12. Comparing the rate-distortion efficiency of different methods for different video sequences (QPs 22, 27, 32, and 37). M represents the length of MER. (a) Flowervase_832 × 480. (b) Keiba_832 × 4480. (c) KristenAndSara_1280 × 4720. (d) SlideEditing_1280 × 4720. (e) Tennis_1920 × 41080. (f) BasketballDrive_1920 × 41080. (g) Traffic_2560 × 41600. (h) PeopleOnStreet_2560 × 41600.

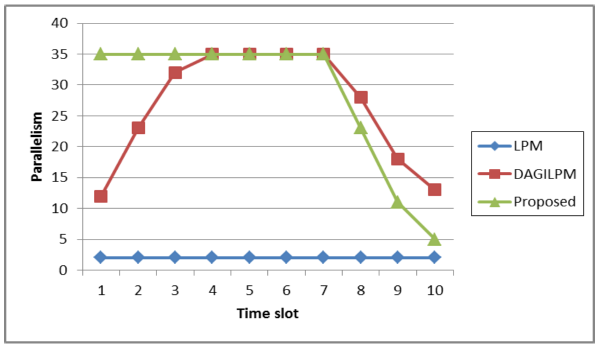
other methods because GPM takes no account of the data dependencies among the block partitions, which significantly increases the bit rates.

D. Parallelism Analysis

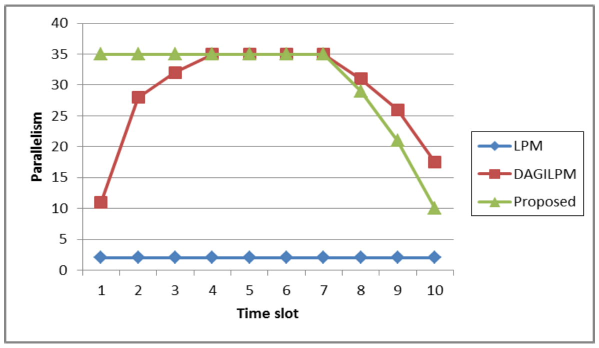
To compare the DP of our method with that of other methods for ME, time scale is normalized as shown in Fig. 13. Time slot is calculated as

$$\text{Time slot} = \frac{\text{Time interval(s)}}{\text{Total time(s)}} * 10 \tag{17}$$

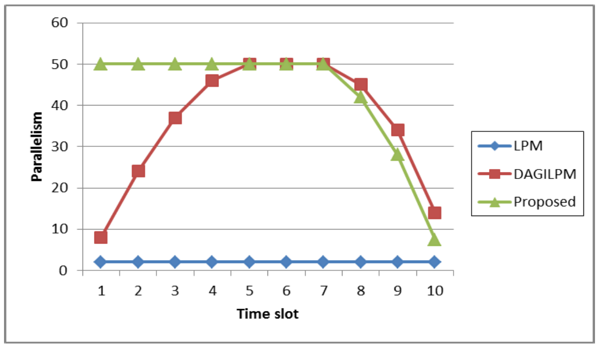
where the Total time(s) is the total consuming time of one frame, and the Time interval(s) starts from processing one frame to the Time slot. We averagely sample ten DPs in the



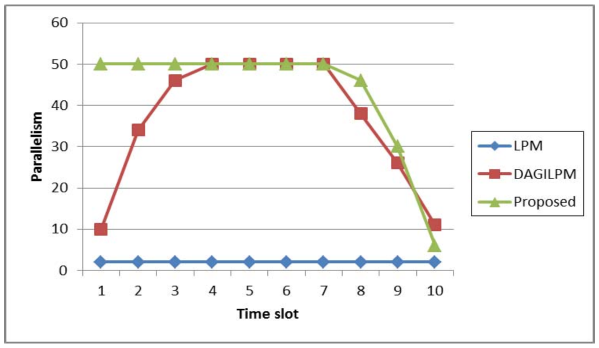
(a)



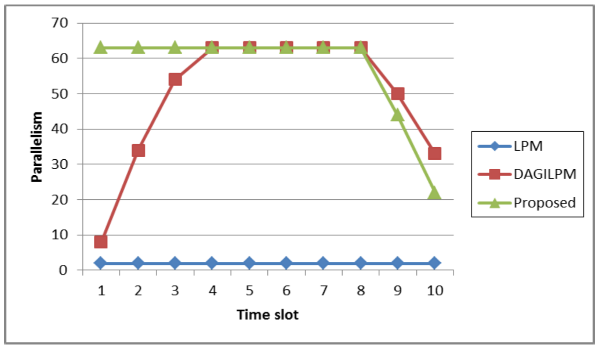
(b)



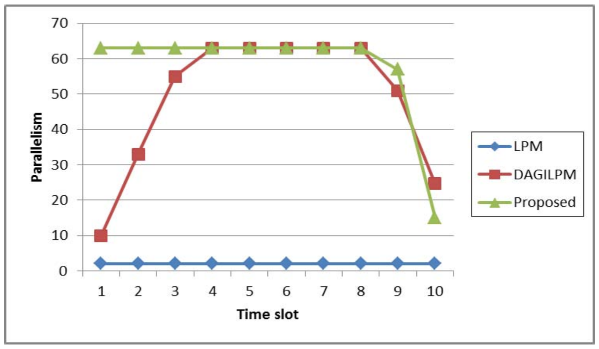
(c)



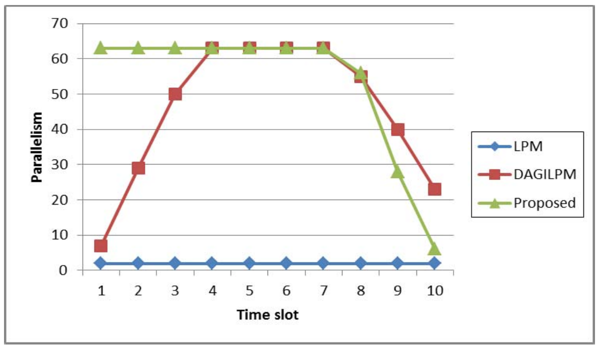
(d)



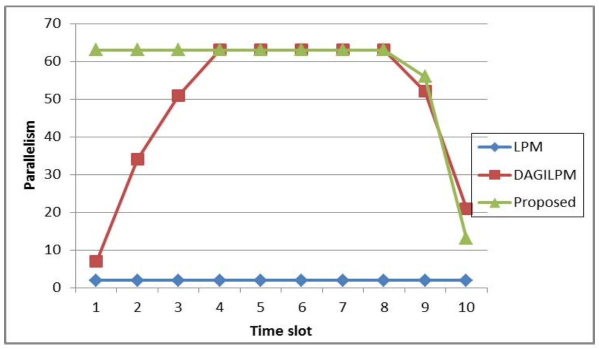
(e)



(f)



(g)



(h)

Fig. 13. Comparing the DP for ME among LPM, DAGILPM, and our proposed method. QP $= 32$, $M = 8$. DAGILPM means the DP of our proposed method without using CIPUs and PIPUs. The DP of our proposed method is much higher than that of LPM and DAGILPM. (a) Flowervase_832 $\times$ 480. (b) Keiba_832$\times$480. (c) KristenAndSara_1280$\times$720. (d) SlideEditing_1280$\times$720. (e) Tennis_1920$\times$1080. (f) BasketballDrive_1920$\times$1080. (g) Traffic_2560$\times$1600. (h) PeopleOnStreet_2560 $\times$ 1600.

timeline for every frame. Then, we calculate the ADP for every time slot.

As shown in Fig. 13, DAGILPM means the DP of our proposed method without using CIPUs and PIPUs. The DP of DAGILPM and our proposed method is much higher than that of LPM. The DP of DAGILPM is low at the beginning and at the end of processing a frame. Our proposed method is less affected and maintains a higher DP. Most of the time, the DP of our approach is equal to the maximum DP of DAGILPM or the number of processing cores. We also find that the DP of our proposed method is low at the end of processing the frame, because there are not enough

TABLE V
AVERAGE SPEEDUP OF OUR PROPOSED METHOD COMPARED WITH SERIAL EXECUTION USING 64 CORES FOR ME, WHOSE MOTION SEARCH IS FAST SEARCH M REPRESENTS THE LENGTH OF MER

Resolution	QP=27 M=8	QP=27 M=16	QP=32 M=8	QP=32 M=16
832x480	11.14	16.66	12.08	21.17
1280x720	11.69	25.98	14.01	33.08
1920x1080	15.40	29.25	14.77	34.82
2560x1600	17.82	42.33	22.02	47.36

TABLE VI
AVERAGE SPEEDUP OF OUR PROPOSED METHOD COMPARED WITH SERIAL EXECUTION USING 64 CORES FOR ME, WHOSE MOTION SEARCH IS FULL SEARCH. M REPRESENTS THE LENGTH OF MER

Resolution	QP=27 M=8	QP=27 M=16	QP=32 M=8	QP=32 M=16
832x480	13.18	17.29	13.19	22.67
1280x720	13.26	29.14	14.33	34.36
1920x1080	17.77	34.92	16.88	42.39
2560x1600	20.70	46.92	26.03	54.10

TABLE VII
PERCENTAGE OF TIME SPENT ON THE ME IN THE ENCODER USING A SINGLE CORE

Resolution of Sequences	Sequences	
832x480	Flowervase	73.90%
832x480	Keiba	66.34%
832x480	Mobisode	79.00%
1280x720	KristenAndSara	60.69%
1280x720	SlideEditing	68.77%
1280x720	SlideShow	67.63%
1920x1080	Tennis	75.31%
1920x1080	BasketballDrive	75.90%
1920x1080	Kimonol	63.74%
2560x1600	Traffic	69.80%
2560x1600	PeopleOnStreet	68.91%
2560x1600	Nebuta	72.93%
	Average	**70.24%**

CIPUs and PIPUs at the end of processing a frame. The DP of our proposed method and DAGILPM dependents on the resolution of the frame. As the resolution of frames becomes smaller, the DP of DAGILPM and our proposed method is reduced.

E. Speedup Analysis

Fig. 14 shows the speedup of all the methods compared with serial execution for ME using 64 cores. Serial execution just uses one processing core. DAGILPM means the speedup of our proposed method without using CIPUs and PIPUs compared with serial execution. Fig. 15 shows the speedup of LPM and our proposed method compared with serial execution for ME with different number of cores. The speedup of LPM (S_{LPM}), DAGILPM (S_{DAGILPM}), and our proposed method (S_{Proposed})

TABLE VIII
SPEEDUP OF OUR PROPOSED METHOD COMPARED WITH SERIAL EXECUTION FOR THE ENCODER USING 64 CORES, QP = 32. M REPRESENTS THE LENGTH OF MER

Resolution of Sequences	Sequences	M=8	M=16
832x480	Flowervase	2.45	2.56
832x480	Keiba	2.06	2.17
832x480	Mobisode	2.69	2.90
1280x720	KristenAndSara	1.93	1.96
1280x720	SlideEditing	2.28	2.34
1280x720	SlideShow	2.21	2.25
1920x1080	Tennis	2.64	2.73
1920x1080	BasketballDrive	2.68	2.79
1920x1080	Kimonol	2.05	2.09
2560x1600	Traffic	2.38	2.41
2560x1600	PeopleOnStreet	2.32	2.36
2560x1600	Nebuta	2.58	2.61
	Average	**2.36**	**2.43**

can be calculated as

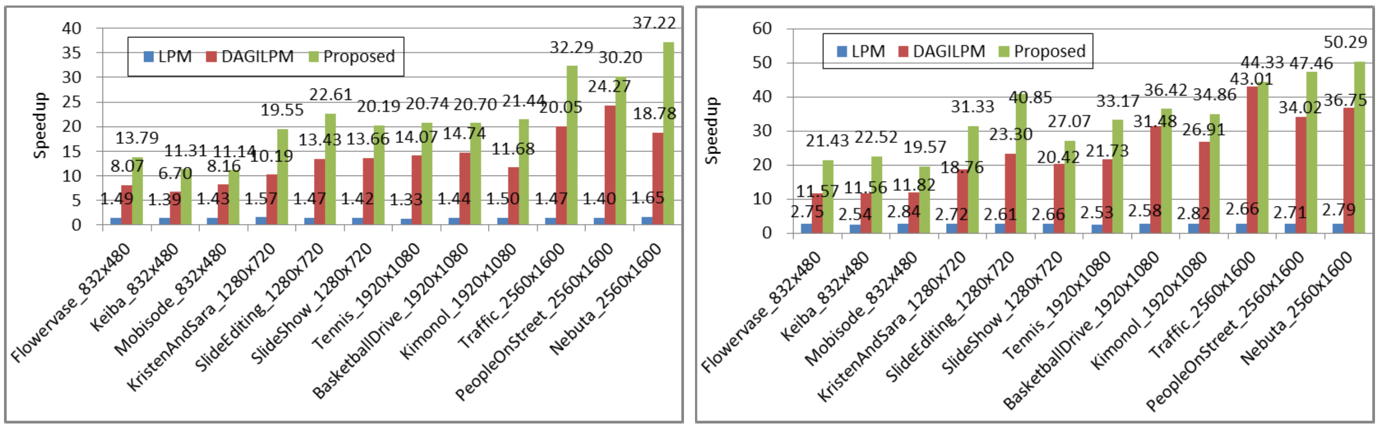
$$S_{\text{LPM}} = \frac{T_{\text{serial}}}{T_{\text{LPM}}} \tag{18}$$

$$S_{\text{DAGILPM}} = \frac{T_{\text{serial}}}{T_{\text{DAGILPM}}} \tag{19}$$

$$S_{\text{Proposed}} = \frac{T_{\text{serial}}}{T_{\text{Proposed}}} \tag{20}$$

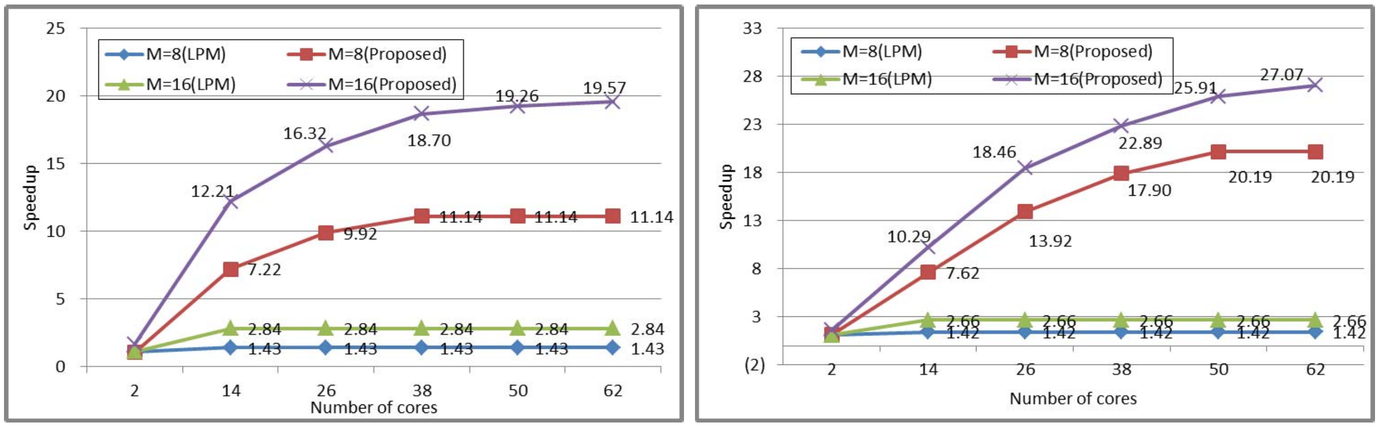
where T_{serial}, T_{LPM}, T_{DAGILPM}, and T_{Proposed} are, respectively, the ME time of serial execution, LPM, DAGILPM, and our proposed method. Tables V and VI show the average speedup of our proposed method compared with serial execution for ME using 64 cores. The numbers are the average ratios of serial execution running time of ME to that using our proposed method. Table VII shows the percentage of time spent on the ME in the encoder using a single core. Table VIII shows the speedup of our proposed method compared with serial execution for the encoder using 64 cores. From Figs. 14 and 15 and Tables V–VIII. We have the following five major observations.

1) As the length of MER (M) increases (Fig. 14), LPM, DAGILPM, and our method all speed up much more quickly than serial processing. This is mainly because the DP of LPM, DAGILPM and our method increase as shown in (1) and (11).
2) As the resolution of frame increases (Fig. 14), the speedup of LPM is nearly unchanged because the DP of LPM stays the same. On the contrary, the speedup of DAGILPM and our method increases because the DP of DAGILPM and our method increases as shown in (10) and (11).
3) As the number of cores increases (Fig. 15), our method speeds up more quickly than LPM because our method fully utilizes the increasing number of cores. When the number of cores is more than 14, the speedup of LPM is unchanged because the DP of LPM is not sufficient for the increasing number of cores. We also find that when the number of cores is more than the DP of our method, the speedup of our method will not increase as well. In general, our method can use more number of cores than LPM.



(a) (b)

Fig. 14. Speedup of LPM, DAGILPM, and our proposed method compared with serial execution for ME using 64 cores. DAGILPM means the speedup of our proposed method without using CIPUs and PIPUs compared with serial execution using 64 cores. (a) QP = 32, $M = 8$. (b) QP = 32, $M = 16$.



(a) (b)

Fig. 15. Speedup of LPM and our proposed method compared with serial execution for ME with different number of cores. M represents the length of MER. (a) Mobisode_832 × 480, QP = 32. (b) SlideShow_1280 × 720, QP = 32.

4) HEVC ME is the most computationally expensive operation in the HEVC encoder (Table VII). After the speedup of ME, compared with serial execution, our proposed method achieves averagely more than two times speedup for the encoder (Table VIII). One of our future work directions is to find efficient parallel methods for other processing stages in the encoder and to find an efficient parallel framework for HEVC encoder.
5) Our proposed method accelerates a lot more than LPM and DAGILPM. Compared with serial execution, our proposed method achieves averagely more than 30 times speedup for ME.

V. Conclusion

Efficient parallelization of HEVC ME on many-core processors is challenging. In general, there are two ways to parallelize ME on many-core processors: 1) GPM and 2) LPM. The GPM provides a high DP but has nonignorable coding efficiency loss; LPM has ignorable coding efficiency loss but the DP of LPM is not adequate for many-core processors. Based on LPM, we propose an efficient parallel framework for HEVC ME. After analyzing the data dependencies among neighboring CTUs, we generate a DAG to capture the dependency relationships of CTUs. We use the DAG-based order to parallelize CTUs and adopt ILPM within each CTU (DAGILPM), which exploits the implicit DP. Then, we process the CIPUs and PIPUs at the beginning and at the end of processing a frame, which further increases the DP. Experiments conducted on a Tile64 processor demonstrate that our method accelerates more than LPM. Meanwhile, the coding efficiency of our method stays the same as LPM, which is much better than GPM.

We also find that the DP of our proposed method is low at the end of processing the frame, which will influence the performance. One of our future work directions is to keep a high DP. Meanwhile, we will also try to find efficient parallel methods for other processing stages in the encoder and to find an efficient parallel framework for HEVC encoder.

References

[1] C. Yan, Y. Zhang, F. Dai, and L. Li, "Highly parallel framework for HEVC motion estimation on many-core platform," in *Proc. Data Compress. Conf.*, Mar. 2013, pp. 63–72.

[2] 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.

[3] R. Sjöberg *et al.*, "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.

[4] 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.

[5] C. Yeo, Y. H. Tan, and Z. Li, "Dynamic range analysis in High Efficiency Video Coding residual coding and reconstruction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 7, pp. 1131–1136, Jul. 2013.

[6] J. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "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.

[7] A. Krutz, A. Glantz, M. Tok, M. Esche, and T. Sikora, "Adaptive global motion temporal filtering for High Efficiency Video Coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1802–1812, Dec. 2012.

[8] A. Abou-Elailah, F. Dufaux, J. Farah, M. Cagnazzo, and B. Pesquet-Popescu, "Fusion of global and local motion estimation for distributed video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 1, pp. 158–172, Jan. 2013.

[9] 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.

[10] J. Vanne, M. Viitanen, T. D. Hämäläinen, 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.

[11] 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.

[12] Y. Zhang, C. Yan, F. Dai, and Y. Ma, "Efficient parallel framework for H.264/AVC deblocking filter on many-core platform," *IEEE Trans. Multimedia*, vol. 14, no. 3, pp. 510–524, Jun. 2012.

[13] C. C. Chi *et al.*, "Parallel scalability and efficiency of HEVC parallelization approaches," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1827–1838, Dec. 2012.

[14] C. Yan, Y. Zhang, F. Dai, X. Wang, L. Li, and Q. Dai, "Parallel deblocking filter for HEVC on many-core processor," *Electron. Lett.*, vol. 50, no. 5, pp. 367–368, Feb. 2014.

[15] N. Ramzan, H. Park, and E. Izquierdo, "Video streaming over P2P networks: Challenges and opportunities," *Signal Process., Image Commun.*, vol. 27, no. 5, pp. 401–411, May 2012.

[16] S. Shi, C.-H. Hsu, K. Nahrstedt, and R. Campbell, "Using graphics rendering contexts to enhance the real-time video coding for mobile cloud gaming," in *Proc. 19th ACM Int. Conf. Multimedia*, Nov. 2011, pp. 103–112.

[17] Z. Gu, W. Lin, B.-S. Lee, and C. T. Lau, "Rotated orthogonal transform (ROT) for motion-compensation residual coding," *IEEE Trans. Image Process.*, vol. 21, no. 12, pp. 4770–4781, Dec. 2012.

[18] V. Sze and A. P. Chandrakasan, "A highly parallel and scalable CABAC decoder for next generation video coding," *IEEE J. Solid-State Circuits*, vol. 47, no. 1, pp. 8–22, Jan. 2012.

[19] N.-M. Cheung, X. Fan, O. C. Au, and M.-C. Kung, "Video coding on multicore graphics processors," *IEEE Signal Process. Mag.*, vol. 27, no. 2, pp. 79–89, Mar. 2010.

[20] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt, "Prefetch-aware shared resource management for multi-core systems," in *Proc. Int. Symp. High-Perform. Comput. Archit.*, Jun. 2011, pp. 141–152.

[21] M. Annavaram, "A case for guarded power gating for multi-core processors," in *Proc. Int. Symp. Comput. Archit.*, Feb. 2011, pp. 291–300.

[22] E. Bini *et al.*, "Resource management on multicore systems: The ACTORS approach," *IEEE Micro*, vol. 31, no. 3, pp. 72–81, May/Jun. 2011.

[23] ClearSpeed. *The CSX600 Processor*. [Online]. Available: http://www.clearspeed.com

[24] M. Zhou, *AHG10: Configurable and CU-group Level Parallel Merge/skip*, document JCTVC-H0082, Feb. 2012.

[25] Q. Yu, L. Zhao, and S. Ma, "Parallel AMVP candidate list construction for HEVC," in *Proc. Vis. Commun. Image Process.*, 2012, pp. 1–6.

[26] Y.-L. Huang, Y.-C. Shen, and J.-L. Wu, "Scalable computation for spatially scalable video coding using NVIDIA CUDA and multi-core CPU," in *Proc. ACM Int. Conf. Multimedia*, Oct. 2009, pp. 361–370.

[27] E. Marth and G. Marcus, "Parallelization of the x264 encoder using openCL," in *Proc. ACM SIGGRAPH*, Jul. 2010, pp. 1–72.

[28] 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.

[29] I.-K. Kim, J. Min, T. Lee, W.-J. Han, and J. Park, "Block partitioning structure in the HEVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1697–1706, Dec. 2012.

[30] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA, USA: Addison-Wesley, 1974, p. 52.

[31] N.-M. Cheung, O. C. Au, M.-C. Kung, P. H. W. Wong, and C. H. Liu, "Highly parallel rate-distortion optimized intra-mode decision on multicore graphics processors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 11, pp. 1692–1703, Nov. 2009.

[32] C. Yan, Y. Zhang, F. Dai, J. Zhang, L. Li, and Q. Dai, "Efficient parallel HEVC intra-prediction on many-core processor," *Electron. Lett.*, vol. 50, no. 11, pp. 805–806, May 2014.

[33] C. Yan *et al.*, "A highly parallel framework for HEVC coding unit partitioning tree decision on many-core processors," *IEEE Signal Process. Lett.*, vol. 21, no. 5, pp. 573–576, May 2014.

[34] J. E. Fowler, S. Mun, and E. W. Tramel, "Block-based compressed sensing of images and video," *Found. Trends Signal Process.*, vol. 4, no. 4, pp. 297–416, Mar. 2012.

[35] R. Joshi, Y. Reznik, and M. Karczewicz, *Simplified Transforms for Extended Block Sizes*, document VCEG Contribution VCEG-AL19, Jul. 2009.

[36] M. Karczewicz *et al.*, "A hybrid video coder based on extended macroblock sizes, improved interpolation, and flexible motion representation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 12, pp. 1698–1708, Dec. 2010.

[37] F. Bossen, *Common Test Conditions and Software Reference Configurations*, document JCTVC-N1006, Jul. 2013.

[38] G. Bjontegard, *Calculation of Average PSNR Differences Between RD-Curves*, document VCEG-M33, Austin, TX, USA, Apr. 2001.

[39] G. Bjontegard, *Improvement of BD-PSNR Model*, document VCEG-AI11, Berlin, Germany, Jul. 2008.

[40] K. Seshadrinathan, R. Soundararajan, A. C. Bovik, and L. K. Cormack, "Study of subjective and objective quality assessment of video," *IEEE Trans. Image Process.*, vol. 19, no. 6, pp. 1427–1441, Jun. 2010.

[41] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of PSNR in image/video quality assessment," *Electron. Lett.*, vol. 44, no. 13, pp. 800–801, Jun. 2008.

Chenggang Yan received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2013.

He is a Post-Doctoral Research Fellow with the Department of Automation, Tsinghua University, Beijing. His research interests include parallel computing, video coding, computational photography, computer vision, and multimedia communication.

Yongdong Zhang (M'08–SM'13) received the Ph.D. degree in electronic engineering from Tianjin University, Tianjin, China, in 2002.

He is a Professor with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China. He has authored over 100 refereed journal and conference papers. His research interests include multimedia content analysis and understanding, multimedia content security, video encoding, and streaming media technology.

Prof. Zhang serves as an Editorial Board Member of *Multimedia Systems Journal* and *Neurocomputing*. He received the Best Paper Awards in PCM 2013, ICIMCS 2013, and ICME 2010, and the Best Paper Candidate in ICME 2011.

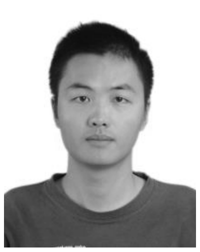
Jizheng Xu (SM'10) received the B.S. and M.S. degrees in computer science from University of Science and Technology of China, Hefei, China, and the Ph.D. degree in electrical engineering from Shanghai Jiao Tong University, Shanghai, China.

He joined Microsoft Research Asia, Beijing, China, in 2003, where he is currently a Lead Researcher. He has been an active contributor to ISO/MPEG and ITU-T video coding standards. He has over 30 technical proposals adopted by H.264/AVC, H.264/AVC scalable extension, High Efficiency Video Coding (HEVC), HEVC range extension and HEVC screen content coding standards. He chaired and co-chaired the ad hoc group of exploration on wavelet video coding in MPEG, as well as various technical ad hoc groups in JCT-VC, e.g., screen content coding, parsing robustness, and lossless coding. He has authored or co-authored over 80 conference and journal refereed papers. He co-organized and co-chaired special sessions on scalable video coding, directional transform, and high-quality video coding at various conferences. He holds 30 U.S. granted or pending patents in image and video coding. His research interests include image and video representation, media compression, and communication.

Dr. Xu was the 2014 Special Session Co-Chair of the IEEE International Conference on Multimedia and Expo.

Feng Dai was born in 1979. He received the M.S. and Ph.D. degrees from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2008.

He is an Associate Professor with the Multimedia Computing Group, Advanced Research Laboratory, Institute of Computing Technology, Chinese Academy of Sciences, Beijing. His research interests include video coding, transmission, and processing.



Jun Zhang received the B.E. degree from Xidian University, Xi'an, China, in 2009. He is currently working toward the Ph.D. degree with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.

His research interests include video coding and computer vision.

Qionghai Dai (SM'05) received the B.S. degree in mathematics from Shanxi Normal University, Xi'an, China, in 1987, and the M.E. and Ph.D. degrees in computer science and automation from Northeastern University, Shenyang, China, in 1994 and 1996, respectively.

He has been a Faculty Member with Tsinghua University, Beijing, China, since 1997. He is currently a Cheung Kong Professor with Tsinghua University and the Director of the Broadband Networks and Digital Media Laboratory. His research interests include signal processing and computer vision and graphics.

Feng Wu (M'99–SM'06–F'13) received the B.S. degree in electrical engineering from Xidian University, Xi'an, China, in 1992, and the M.S. and Ph.D. degrees in computer science from Harbin Institute of Technology, Harbin, China, in 1996 and 1999, respectively.

He was a Principle Researcher and Research Manager with Microsoft Research Asia, Beijing, China. He is currently a Professor with University of Science and Technology of China, Hefei, China. He has authored or co-authored over 200 high-quality papers (including several dozens of the IEEE transaction papers) and top conference papers on MOBICOM, SIGIR, CVPR, and ACM MM. He holds 77 granted U.S. patents. His 15 techniques have been adopted into international video coding standards. His current research interests include image and video compression, media communication, and media analysis and synthesis.

Prof. Wu was an Associate Editor of IEEE TRANSACTIONS ON CIRCUITS AND SYSTEM FOR VIDEO TECHNOLOGY, IEEE TRANSACTIONS ON MULTIMEDIA, and several other international journals. He also was the TPC Chair in MMSP 2011, VCIP 2010, and PCM 2009, and Special Sessions Chair in ICME 2010 and ISCAS 2013. He received the IEEE Circuits and Systems Society 2012 Best Associate Editor Award. As a co-author, he received the Best Paper Award in IEEE T-CSVT 2009, PCM 2008, and SPIE VCIP 2007.