

Highly Parallel Framework for HEVC Motion Estimation on Many-core Platform

Chenggang Yan^{*,+}, Yongdong Zhang^{*}, Feng Dai^{*} and Liang Li⁺

^{*}Advanced Computing Research Laboratory, Beijing Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, Chinese Academy of Sciences, No.6 Kexueyuan South Road, Zhongguancun, Beijing, 100190, China
{yanchenggang, zhyd, fdai}@ict.ac.cn

⁺University of Chinese Academy of Sciences, No.19A Yuquanlu, Beijing, 100049, China
lli@jdl.ac.cn

Abstract: As the next generation standard of video coding, High Efficiency Video Coding (HEVC) is expected to be more complex than H.264/AVC. Many-core platforms are good candidates for speeding up HEVC in the case that HEVC can provide sufficient parallelism. The local parallel method (LPM) is the most promising parallel proposal for HEVC motion estimation (ME), but it can't provide sufficient parallelism for many-core platforms. On the premise of keeping the data dependencies and coding efficiency the same as the LPM, we propose a highly parallel framework to exploit the implicit parallelism. Compared with the well-known LPM, experiments conducted on a 64-core system show that our proposed method achieves averagely more than 10 and 13 times speedup for 1920x1080 and 2560x1600 video sequences, respectively.

1. Introduction

Standards and techniques for video coding are being actively developed [1-4]. High Efficiency Video Coding (HEVC) is the state-of-the-art video coding standard [5-7]. Compared with H.264/AVC, HEVC aims to provide a doubling in coding efficiency [8-10]. The price to be paid for higher coding efficiency is higher computational complexity. HEVC encoders are expected to be several times more complex than H.264/AVC encoders[11]. Many-core platforms are good candidates for speeding up compression algorithms, but only in the case that compression algorithms can be effectively parallelized [12-15]. HEVC currently contains several proposals aimed at making it more “parallel-friendly” [7]. Parallelizing HEVC motion estimation (ME) is an important part of it [16].

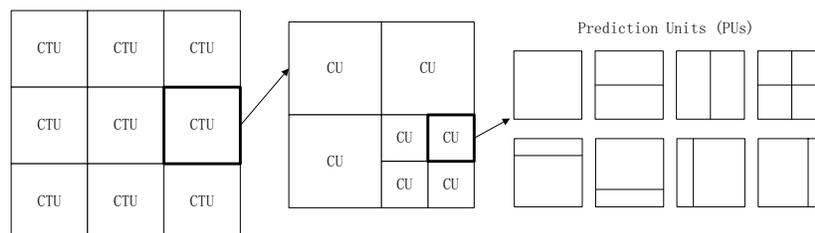


Figure 1: Flexible hierarchy of unit representation for motion estimation (ME)

HEVC provides a highly flexible hierarchy of unit representation for ME, which includes three block concepts [7]: coding tree unit (CTU), coding unit (CU) and prediction unit (PU) (figure 1). Hierarchy of unit representation has been proved effective [17-19]. Each frame is divided into CTUs, which can be recursively split into smaller

CUs by using a generic quadtree segmentation structure. CU can be further split into PUs, which have 8 partition modes used for ME. PUs are the basic units used for carrying the motion data related to ME.

Minhua Zhou [16] proposes an efficient local parallel method (LPM) for parallelizing HEVC ME. LPM divides each CTU into a number of non-overlapped parallel motion estimation regions (MERs). From MER to MER, ME is carried out sequentially. Within each MER, ME is carried out in parallel for all the PUs. LPM has been adopted into the HEVC working draft. In order to guarantee the coding efficiency, the maximum parallelism of LPM is equal or less than 8, which is not adequate for many-core platforms. It is urgently demanded to provide sufficient parallelism for HEVC ME.

LPM eliminates the data dependencies among PUs within the same MER. But the MERs and CTUs have to be processed sequentially. On the premise of keeping data dependencies the same as the LPM, we further analyze the dependencies in different levels of data granularity within the same frame:

- There exist independent PUs (IPUs), which have no data dependencies on other PUs within the same frame.
- 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.

Based on LPM, we propose a highly parallel framework for HEVC ME. We generate a directed acyclic graph (DAG) [20] to capture the dependency relationships among neighboring CTUs. We use the DAG-based order to parallelize CTUs, which exploits the implicit CTU parallelism. In order to further increase the parallelism overhead, we process the IPUs at the beginning and at the end of processing a frame. Meanwhile, we adopt LPM within each CTU. The maximum parallelism of our proposed method reaches 120 and 160 for 1920x1080 and 2560x1600 video sequences, respectively. The data dependencies and coding efficiency stay the same as the LPM.

Our proposed parallel framework is suitable to many-core platform. Our testing platform is Tile64, which is a member of TILERA many-core processor family [21]. Experiments demonstrate that our proposed method significantly improves the performance overhead than LPM.

The remainder of the paper is organized as follows. Section 2 gives a review of the related work. Section 3 presents the proposed highly parallel framework for HEVC ME. The experiment results are elaborated in section 4. Finally, section 5 concludes the paper.

2. Related Work

PUs are the basic units used for 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 [7]. LPM [16] introduce the concept of MER and divides each CTU into a number of non-overlapped MERs. All the MERs are exact square shapes with the same size. LPM introduce new availability rule for PUs. Figure 2(a) shows an example of MER and available PUs for PU0, 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 treated as unavailable for each other. Within the same MER, ME is carried out in parallel for all the PUs. For example, PU8 and PU9 are all belong to 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 parallelism of LPM (MP_{LPM}) can be expressed as equation (1).

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

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 PU is as small as 8x4 or 4x8 luma samples, $S(PU)$ will reach the minimum. In order to guarantee the coding efficiency, M is commonly equal to 16 or 8 today [16]. So the MP_{LPM} is equal or less than 8. LPM can't provide sufficient parallelism for many-core platforms.

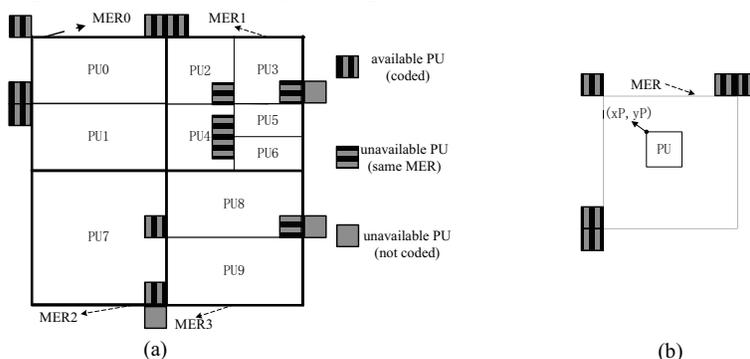


Figure 2: (a) Example of MER and available PUs for PU0, PU5 and PU9 (b) MER's neighboring left, left-down, upper, upper-left and upper-right PUs are available for PUs within the MER

3. Highly Parallel Framework for HEVC ME

In this section, on the premise of keeping data dependencies and coding efficiency the same as the LPM, we will firstly find the IPUs and analyze the data dependencies among neighboring CTUs. Then we will use the DAG to present the dependency relationships of CTUs and propose a highly parallel framework for HEVC ME.

3.1 Data Dependency Analysis

Independent PUs. On the basis of LPM, we find out IPUs. CTUs and MERs in a frame are processed sequentially in scan order. So only the MER's neighboring left, left-down, upper, upper-left and upper-right PUs are available for PUs within the MER (figure 2(b)). For example, PU9 has data dependencies on MER3's neighboring left and upper-left PUs (figure 2(a)). If PUs within the MER have no data dependencies on the MER's neighboring left, left-down, upper, upper-left and upper-right PUs, they will have no data dependencies on all the PUs within the frame. So the IPUs have two characteristics:

- The IPU's left boundary and MER's left boundary do not overlap.
- The IPU's upper boundary and MER's upper boundary do not overlap.

The neighboring PUs of the IPU are either not coded or belong to the same MER, which are unavailable for the IPU. For example, as shown in figure 2(a), PU5 meets requirements of IPU. PU5 and its neighboring left, left-down, upper and upper-left PUs are all belong to the MER1. Meanwhile, its neighboring upper-right PU isn't 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 IPU if the following condition satisfies:

$$xP \bmod M \neq 0 \quad \text{and} \quad yP \bmod M \neq 0 \quad (2)$$

where mod is modulus operation.

Data dependencies among neighboring CTUs. In this section, we will analyze the data dependencies among neighboring CTUs. Figure 3(a) illustrates an example of 3x3 CTUs frame. The CTUs are processed according to their numbers which indicate the time stamp. The data dependencies among neighboring CTUs are caused by the PUs. When processing the current CTU, the PUs within the current CTU's 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 the its adjacent left-down CTU. Meanwhile, the PUs in the current CTU's 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 (figure 3(b)). The arrows indicate dependencies. When processing the current CTU, the left, upper, upper-left and upper-right CTUs should have been completed processed.

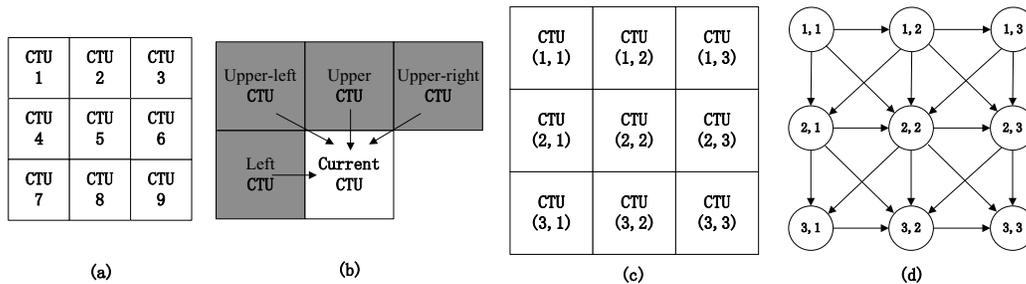


Figure 3: (a) An example of 3x3 CTUs frame. The CTUs are processed according to their numbers which indicate the time stamp. (b) Data dependencies among neighboring CTUs. The arrows indicate dependencies. (c) Each CTU in the frame is mapped into a point in a two-dimensional coordinates plane. (d) DAG for representing the dependency relationships of CTUs.

3.2 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 figure 3(c), we firstly map each CTU in the frame into a point in a two-dimensional coordinates plane as follows:

$$i = \text{ceil}\left(\frac{k}{W}\right) \quad (3)$$

$$j = k \bmod W \quad (4)$$

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 (figure 3(a)), W is the horizontal CTU number of the frame, ceil function returns the value of a number rounded upwards to the nearest integer.

Then we use a DAG to represent the execution flow of the CTUs and the precedence constraints among the CTUs [20]. As shown in figure 3(d), mark the DAG as $G=(V,E)$, which consists of a set of vertices V and edges E . Vertices are numbered according to the coordinate value of CTUs in the two-dimensional coordinates plane. For example, vertex

$v_{i,j}$ in figure 3(d) represents the CTU with coordinate value (i,j) in figure 3(c). 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. 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's important to record and update the in-degrees of all the vertices. We get the initial value of the in-degrees by the adjacency matrix. We generate the adjacency matrix A of the DAG as follows:

$$A_{(i,j),(m,n)} = \begin{cases} 1, & (v_{i,j}, v_{m,n}) \in E \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

$$\text{s.t. } 1 \leq i, m \leq H \quad 1 \leq j, n \leq W$$

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

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

$$D_{m,n} = \sum_{i=1}^H \sum_{j=1}^W A_{(i,j),(m,n)} \quad (6)$$

$$\text{s.t. } 1 \leq m \leq H \quad 1 \leq n \leq W$$

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

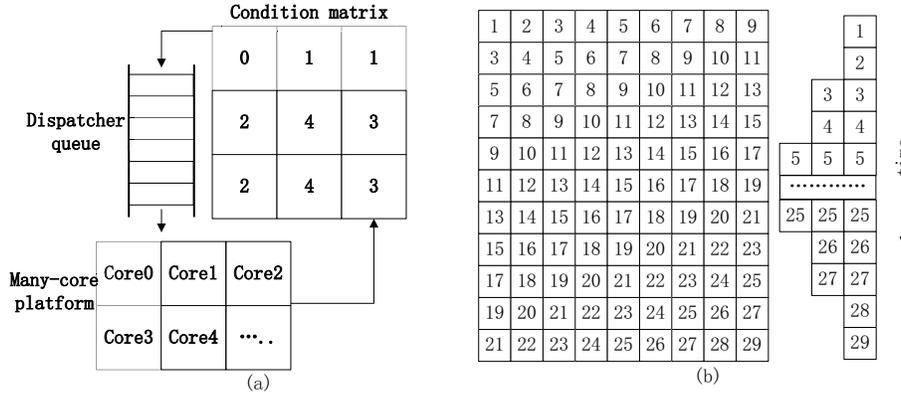


Figure 4: (a) DAG-based order to parallelize the CTUs. (b) An example of DAG-based order to parallelize the CTUs for a frame.

3.3 Highly Parallel Framework

After getting the initial value of the in-degrees, we use a DAG-based order to parallelize the CTUs as shown in figure 4(a). Condition matrix (CM) is a 2D matrix, which is designed to record the number of related CTUs for each CTU. The initial value of CM is identical with D . When some values in CM are zero, the corresponding CTUs can be processed in parallel. When a CTU with coordinate (i, j) in CM is processed, the values of coordinates $(i+1, j)$, $(i+1, j-1)$, $(i, j+1)$ and $(i+1, j+1)$ in CM will minus one operation. Furthermore, dispatcher queue (DQ) is a waiting queue, whose elements are the coordinates of available CTUs. The pseudo code of the DAG-based order is expressed as follows:

- (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 values in the 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 platform
 - (4) Step 4: Update CM. When a CTU with coordinate (i, j) in CM is processed, the values of coordinates $(i+1, j)$, $(i+1, j-1)$, $(i, j+1)$ and $(i+1, j+1)$ in CM will minus one operation.
 - (5) Step 5: Repeat above steps 2~4 until each frame is over

The processing times of CTUs are different from each other. In order to analyze the maximum parallelism conveniently, we suppose all the processing times of CTUs are the same. An example of DAG-based order to parallelize the CTUs for a frame is illustrated in figure 4(b). Rectangles represent CTUs, which are processed according to their numbers. CTUs with the same numbers are processed concurrently. The maximum parallelism of CTU (MP_{CTU}) can be calculated as follows:

$$MP_{CTU} = \min(\text{ceil}(\frac{W}{2}), H) \quad (7)$$

Just like LPM, when a CTU is being processed, the MERs within the CTU are being processed sequentially. The PUs within the same MER are being processed in parallel. So the maximum parallelism of our proposed highly parallel framework (MP_{HPF}) can be summarized as follows:

$$MP_{HPF} = MP_{CTU} \times MP_{LPM} \quad (8)$$

Table 1 compares the maximum parallelism of LPM with that of our method. The size of CTU is usually specified by 64x64 [16]. The maximum parallelism of our method is much more than that of LPM. When the resolution of MER is 16x16, the maximum parallelism of our proposed method reaches 120 and 160 for 1920x1080 and 2560x1600 video sequences, respectively.

Resolution of MER	Resolution of CTU	MP_{LPM}	MP_{HPF}
8x8	832x480	2	14
	1280x720	2	20
	1920x1080	2	30
	2560x1600	2	40
16x16	832x480	8	56
	1280x720	8	80
	1920x1080	8	120
	2560x1600	8	160

Table 1: The maximum parallelism of LPM and our method

In order to further increase the average parallelism overhead, we process the IPUs at the beginning and at the end of processing a frame. From figure 4(b), we find that the parallelism of CTU is low at the beginning and at the end of processing a frame. The parallelism starts at one and increases one at two time stamps, which is not believed to be adequate for many-core platform. There are many idle cores at the beginning and at the

end of processing a frame, which influences the overhead performance. The average parallelism (AP) can be calculated as follows:

$$AP = \left(\frac{1 + MP_{CTU}}{2}\right) \times MP_{LPM} \quad (9)$$

From section 3.1, we find that IPUs have no data dependencies on other PUs within the frame. When the parallelism is smaller than MP_{HPF} , we use the idle cores to process the IPUs. We make the non-idle cores equal to MP_{HPF} until all the IPUs have been processed, which further increases the average parallelism overhead.

4. Experimental Results

4.1 Input Stream and Environment Conditions

To compare our proposed method with LPM [16], we adopt an encoder migrated from HEVC reference software HM7.0 [22] 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 [22] using different quantization parameters (QPs). 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 [21]. In order to avoid the impact of special platform, we do not utilize any platform-dependent optimizations.

4.2 Speedup Analysis

Figure 5 shows the speedup of all the methods compared to serial execution using 64 cores. Figure 6 shows the speedup of all the methods compared to serial execution with different number of cores. The speedup of LPM (S_{LPM}) and HPF (S_{HPF}) can be calculated as follows:

$$S_{LPM} = \frac{T_{serial}}{T_{LPM}} \quad (10)$$

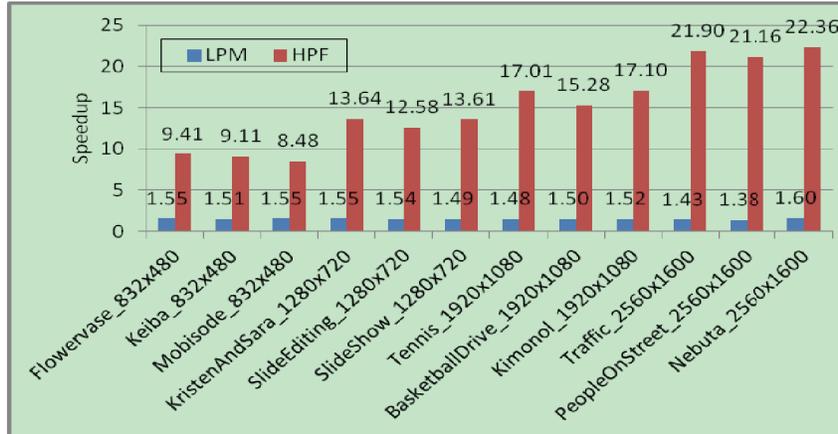
$$S_{HPF} = \frac{T_{serial}}{T_{HPF}} \quad (11)$$

where T_{serial} , T_{LPM} and T_{HPF} are respectively the ME time of serial execution, LPM and HPF. Table 2 shows the average speedup of our proposed method compared to LPM using 64 cores. The numbers are the average ratios of LPM running time of ME to that using our proposed method. From figure 5,6 and table 2. We get four major observations:

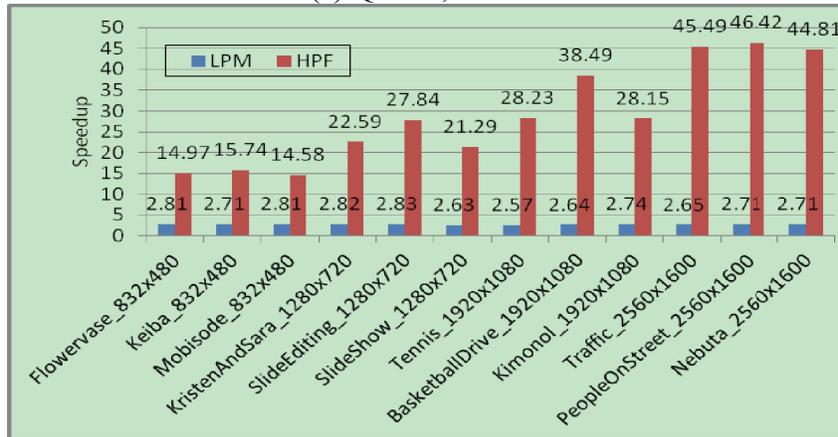
- As the size of MER (M) increases (figure 5), our method and LPM both speed up much more quickly than serial processing. This is mainly because the parallelism of our method and LPM increase as shown in equations (1) and (8).
- As the resolution of frame increases (figure 5), the speedup of LPM is nearly unchanged because the parallelism of LPM stays the same. On the contrary, the speedup of our method increases because the parallelism of our method increases as shown in equation (7) and (8).
- As the number of cores increases (figure 6), 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

parallelism of LPM is not sufficient for the increasing number of cores. We also find that when the number of cores is more than the parallelisms of our method, the speedup of our method will not increase as well. In general, our method can use more cores than LPM.

- Our proposed method accelerates a lot more than LPM (table 2). Compared with LPM, our proposed method achieves averagely more than 10 and 13 times speedup for 1920x1080 and 2560x1600 video sequences, respectively.

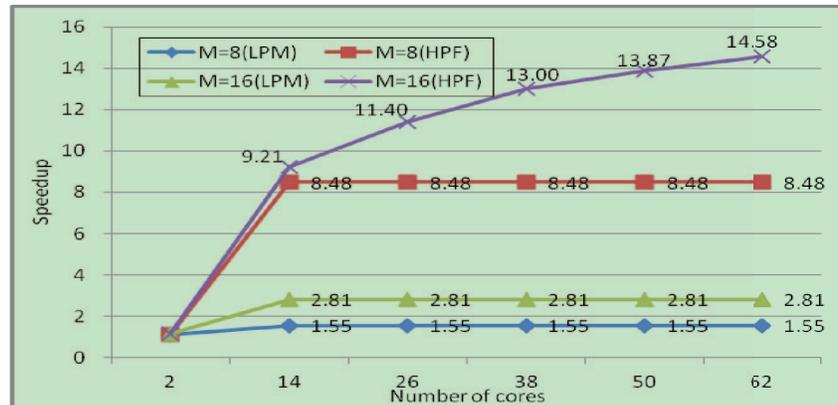


(a) QP=32, M=8

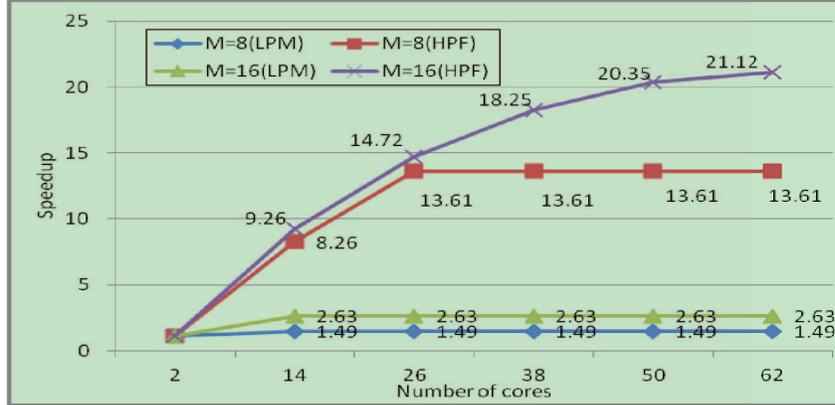


(b) QP=32, M=16

Figure 5: The speedup of LPM and HPF compared to serial execution using 64cores.



(a) *Mobisode_832x480*, QP=32



(b) *SlideShow_1280x720*, QP=32

Figure 6: The speedup of LPM and HPF compared to serial execution with different number of cores

Resolution	QP=27		QP=32	
	$M=8$	$M=16$	$M=8$	$M=16$
832x480	6.19	4.83	5.86	5.44
1280x720	8.28	8.86	8.70	8.66
1920x1080	11.08	12.05	10.98	11.93
2560x1600	13.61	13.98	14.84	16.94

Table 2: The average speedup of our proposed method compared to LPM using 64 cores.

5. Conclusions

As the state-of-the-art parallel HEVC ME method, LPM provides insufficient parallelism for many-core platforms. Based on LPM, we propose a highly parallel framework for HEVC ME. After analyzing the data dependencies among neighboring CTUs, we generate a DAG to capture the dependency relationships of CTUs and use the DAG-based order to parallelize CTUs, which exploits the implicit parallelism. Then we process the IPU at the beginning and at the end of processing a frame, which further increases the parallelism overhead. Experiments conducted on Tile64 platform demonstrate that our method accelerates more than the state-of-the-art parallel HEVC ME method.

Acknowledgement. This work is supported by National Nature Science Foundation of China (61272323, 61102101), National Key Technology Research and Development Program of China (2012BAH06B01), Beijing New Star Project on Science & Technology (2007B071), Co-building Program of Beijing Municipal Education Commission

References

- [1] Eduardo Martínez-Enríquez, Antonio Ortega, “Lifting Transforms on Graphs for Video Coding,” in *Proc. DCC*, pp. 73-82, Mar. 2011.
- [2] Yan Ye, Giovanni Motta, Marta Karczewicz, “Enhanced Adaptive Interpolation Filters for Video Coding,” in *Proc. DCC*, pp. 435-444, Mar. 2010.

- [3] Jiangtao Wen, Mou Xiao, Jianwen Chen, Pin Tao, Chao Wang, “Fast Rate Distortion Optimized Quantization for H.264/AVC,” in *Proc. DCC*, pp. 557, Mar. 2010.
- [4] Charles D. Creusere, “Motion-compensated video compression with reduced complexity encoding for remote transmission,” *Sig. Proc.: Image Comm.* vol. 16, pp. 627-642, Apr. 2001
- [5] G. J. Sullivan, J.-R. Ohm, “Recent Developments in Standardization of High Efficiency Video Coding (HEVC),” *Proc. SPIE*, vol. 7798, pp. 30–36, Aug. 2010
- [6] R. Joshi, Y. Reznik, and M. Karczewicz, “Efficient large size transforms for high performance video coding,” *Proc. SPIE*, vol. 7798, pp. 1–7, Aug. 2010
- [7] G. J. Sullivan et al., “Overview of the High Efficiency Video Coding (HEVC) Standard,” *IEEE Trans. Circuits Syst. Video Technol.*, to appear, 2012.
- [8] B. Li, G. J. Sullivan, and J. Xu, “Compression performance of high efficiency video coding (HEVC) working draft 4,” in *Proc. ISCAS*, pp. 886-889, May 2012.
- [9] B. Li, G. J. Sullivan, and J. Xu, “Comparison of compression performance of HEVC draft 6 with AVC high profile,” *document JCTVC-I0409*, Apr. 2012.
- [10] B. Li, G. J. Sullivan, and J. Xu, “Comparison of compression performance of HEVC draft 7 with AVC high profile,” *document JCTVC-J0236*, July 2012.
- [11] F. Bossen et al., “HEVC Complexity and Implementation Analysis,” *IEEE Trans. Circuits Syst. Video Technol.*, to appear, 2012.
- [12] Yongdong Zhang, Chenggang Yan, Feng Dai and Yike Ma, “Efficient parallel framework for H.264/AVC deblocking filter on many-core platform,” *IEEE Transactions on Multimedia*, vol. 14, pp. 510-524, June 2012.
- [13] Yi Pang, Lifeng Sun, Jiangtao Wen, Fengyan Zhang, Weidong Hu, Wei Feng, Shiqiang Yang, “A Framework for Heuristic Scheduling for Parallel Processing on Multicore Architecture: A Case Study With Multiview Video Coding,” *IEEE Trans. Circuits Syst. Video Techn.* vol. 19, pp.1658-1666, Nov. 2009.
- [14] Xiulian Peng, Jizheng Xu, Feng Wu, “Highly parallel image coding for many cores,” in *Proc. ISCAS*, pp. 105-108. May 2011.
- [15] Xiulian Peng, Jizheng Xu, You Zhou, Feng Wu, “Highly Parallel Line-Based Image Coding for Many Cores,” *IEEE Transactions on Image Processing*, vol. 21, pp. 196-206, Jan. 2012
- [16] Minhua Zhou, “AHG10: Configurable and CU-group level parallel merge/skip,” *JCTVC-H0082*, Feb. 2012
- [17] James E. Fowler et al., “Block-Based Compressed Sensing of Images and Video,” *Foundations and Trends in Signal Processing* vol. 4, pp. 297-416, Mar. 2012.
- [18] R. Joshi, Y. Reznik, and M. Karczewicz, “Simplified Transforms for Extended Block Sizes,” *VCEG Contribution VCEG-AL19*, July 2009
- [19] 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, pp. 1698 – 1708, Dec. 2010
- [20] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, “The Design and Analysis of Computer Algorithms. Reading, ” *MA: Addison-Wesley*, pp. 52, 1974
- [21] S. Bell, B. Edwards, J. Amann, et al, “TILE64-Processor: A 64-core SoC with mesh,” *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 88-598, Feb. 2008
- [22] F. Bossen, “Common test conditions and software reference configurations,” *JCTVC-I1100*, Apr. 2012