Hindawi
Scientific Programming
Volume 2017, Article ID 1431574, 15 pages
https://doi.org/10.1155/2017/1431574

Research Article

A Highly Parallel and Scalable Motion Estimation Algorithm with GPU for HEVC

Yun-gang Xue, Hua-you Su, Ju Ren, Mei Wen, Chun-yuan Zhang, and Li-quan Xiao

School of Computer, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Yun-gang Xue; xueyungangyun@163.com

Received 13 March 2017; Revised 13 August 2017; Accepted 10 September 2017; Published 12 October 2017

Academic Editor: Christoph Kessler

We propose a highly parallel and scalable motion estimation algorithm, named *multilevel resolution motion estimation* (*MLRME* for short), by combining the advantages of local full search and downsampling. By subsampling a video frame, a large amount of computation is saved. While using the local full-search method, it can exploit massive parallelism and make full use of the powerful modern many-core accelerators, such as GPU and Intel Xeon Phi. We implanted the proposed MLRME into HM12.0, and the experimental results showed that the encoding quality of the MLRME method is close to that of the fast motion estimation in HEVC, which declines by less than 1.5%. We also implemented the MLRME with CUDA, which obtained 30–60x speed-up compared to the serial algorithm on single CPU. Specifically, the parallel implementation of MLRME on a GTX 460 GPU can meet the real-time coding requirement with about 25 fps for the 2560×1600 video format, while, for 832×480, the performance is more than 100 fps.

1. Introduction

Progress in science and technology is bringing about a growing demand for video quality and thus the development of the video coding methods. *High Efficiency Video Coding* (HEVC) in Bross et al. [1] is the state-of-the-art video coding standard based on H.264/AVC in Wiegand et al. [2]. HEVC doubles the data compression ratio and improves the compressed video quality at the cost of introducing approximately 4 times the computational complexity compared to that of H.264/AVC. Moreover, HEVC aims at higher resolution video sequences, leading to a rapid increase in the encoding complexity of video sequence. Thus the high computational complexity posts a challenge for real-time video coding with HEVC.

As a vital part of video encoding, *motion estimation* (ME) is the process of determining *motion vector* (MV) that describes the position transformation of a pixel block between the current frame and its adjacent frame, named reference frame, in a video sequence. Since a pixel block may be slightly different in the current frame and reference frame, its MV is defined by the displacement between the pixel block in the current block and its matching block in the reference frame. Here, the matching block is the most similar one to the current block.

ME is the most time-consuming component of HEVC, using more than 50% of the execution time in Purnachand et al. [3]. In order to decide the final partition modes of one *Coding Tree Unit* (CTU), ME needs to be invoked as many as 425 times in the *HEVC test model* (HM) in Wang et al. [4]. Therefore, the design of ME algorithms is a hotspot in academia and industry, and lots of ME algorithms have been proposed in recent years. The full-search method is the simplest and most straightforward way, which checks every position of searching windows in the reference frames. This method can get the global optimal result and thus the best encoding quality. However, its computational complexity is too huge to be applied to an encoding algorithm. Fast search algorithms greatly reduce the computational complexity with a slight decline in the encoding quality, and typical ones include the *three-step search* (TSS) of Koga et al. [5], the *four-step search* (FSS) method of Po and Ma [6], the *diamond search* (DS) method of Zhu and Ma [7], the *cross-diamond search* (CDS) method of Cheung and Po [8], the *hexagonal search* (HS) method of Hamosfakidis and Paker [9],

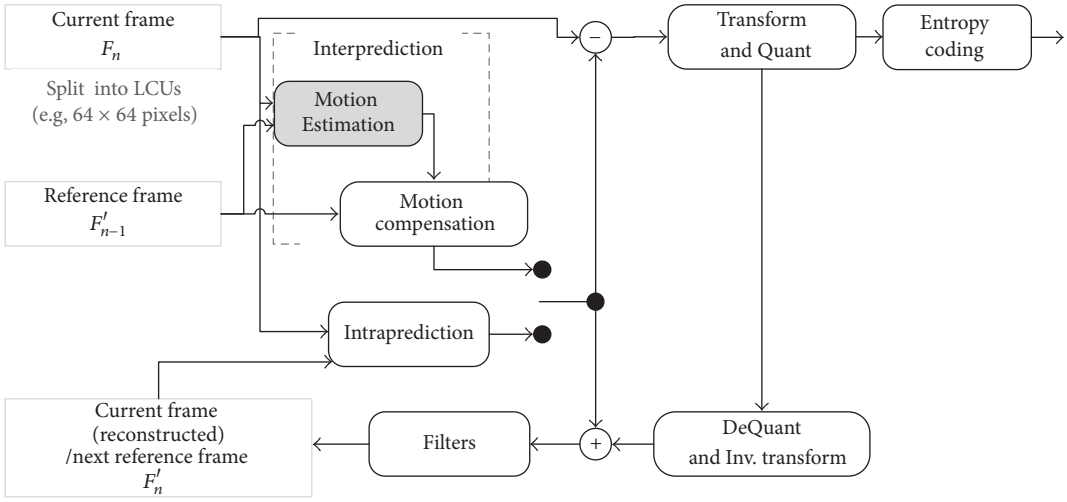


Figure 1: HEVC framework.

the *hybrid unsymmetrical multi-Hexagon-grid search* (UMHexagonS) method of Zhu et al. [10], the *enhanced hexagonal search* (EHS) method of Zhu et al. [11], the *enhanced predictive zonal search* (EPZS) method of Tourapis [12], and the *test zone search* (TZS) method of Purnachand et al. [3].

Multicore or many-core processors are now popular, and particularly many-core platforms give us a good chance to release the computing stress of many compute-intensive algorithms, including ME. Although the full-search method is highly parallel, its computational complexity is too huge for real applications even using the many-core processors. On the contrary, fast ME algorithms do not possess sufficient parallelism due to the complicated data dependencies, so they cannot make full use of the modern many-core processors, such as *graphics processing unit* (GPU).

To have a good trade-off among the encoding quality, computational complexity, and parallelism, an innovative ME algorithm, named *multilevel resolution motion estimation* (MLRME), is proposed in this paper. At the expense of a slight decline of the encoding quality, the new algorithm has abundant parallelism, which can make full use of the powerful computational capacity of modern many-core processors.

The rest of this paper is organized as follows: the second section presents a review of background and the related work. The third section details the proposed MLRME method and a related characteristics analysis. In the fourth section, we describe the parallel implementation and optimization on GPU. The experimental results and corresponding analysis are shown in the fifth section.

2. Background and Related Work

2.1. HEVC. The HEVC framework incorporates six important modules, including Intraprediction, Interprediction (motion estimation and motion compensation), Transform and Quant, DeQuant and Inverse Transform, Filters and Entropy Coding, shown in Figure 1. It is known that there exist plenty of similar pixel blocks within one frame and among adjacent frames, and the similar pixel blocks can be regarded as information redundancy in the digital video. These six modules work together to reduce the redundancy and compact the storage of the video sequence. The Prediction modules, including the Intraprediction and the Interprediction, help finding matching block, the most similar pixel block, for current pixel block. The Intraprediction works in the same frame, while the Interprediction works among the current frame and its adjacent frames, which are called reference frames. The Prediction modules can generate predictive data of current pixel blocks based on its matching block. The difference between the current pixel block and its predictive pixel block is usually much smaller than that of current pixel block because of the similarity. The Transform and Quant module aims to make the difference data smaller and smaller, which are as close as zero. The Entropy Coding module can compress the result data of the Transform and Quant module into an encoded bit-stream, which is rather short. The DeQuant and Inverse Transform module is the inverse process of the Quant and Transform module, which produces the difference data from the result data of Transform and Quant module. The sum data of difference data and the matching block are almost the same as the current pixel block but are slightly different because of the Quant module. The Filter module can improve the quality of the reconstructed data by smoothing out artifacts resulting from block method and quantization. The final data after filtering are the decoded video data, which can be used as the reference frame of the next frame.

HEVC processes the frame as a series of basic pixel blocks rather than the whole frame. The basic pixel block is called LCU (largest coding unit), and its size is usually set as 64 × 64 in HEVC. The input video frame is firstly split into nonoverlapped LCUs, as shown in Figure 2(a). A LCU needs to be further partitioned during the encoding procedure. A highly flexible hierarchy of unit representation for LCU Sullivan et al. [13] is provided by HEVC, including three block concepts, CTU (coding tree unit), CU (coding unit), and PU (predictive unit). A LCU can be recursively subdivided into

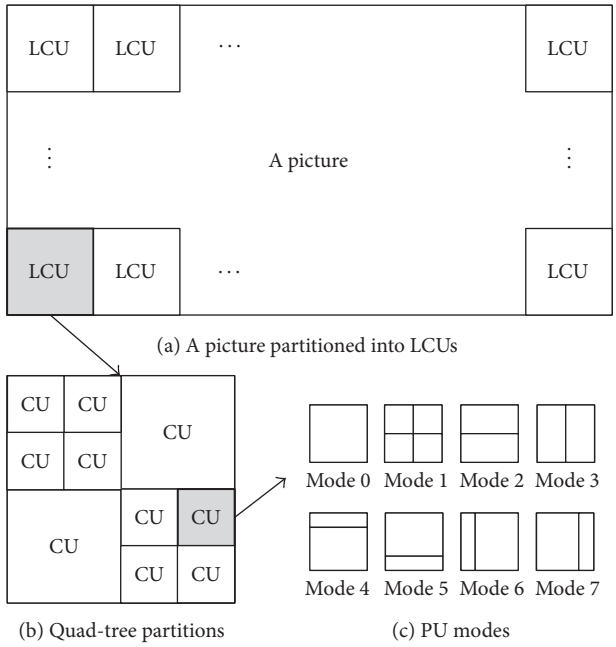


Figure 2: The partitioned process of a picture.

square CUs by a quad-tree till 8 × 8, as shown in Figure 2(b). Therefore, there are several sizes for a CU, including 64 × 64, 32 × 32, 16 × 16 and 8 × 8. A CU can also be further split by eight partition modes, described in Figure 2(c). Thus a CU can be divided into one, two, or four parts and each part is a PU, which is the basic unit delivering the motion information related to ME. It should be noticed that the partition of a CU is not recursive.

2.2. Motion Estimation in HEVC. For each PU of a LCU in HEVC, the ME algorithm aims to find out the matching block in a search window of the reference frame. In order to score a pixel block, the ME algorithm uses J_{MV} (Lagrangian cost function) and finds the pixel block with the smallest J_{MV} value as the matching block of current PU. J_{MV} is defined:

$$J_{\mathrm{MV}} = \mathrm{SAD}\,(\mathrm{MV}) + \lambda_M * R\,(\mathrm{MV} - \mathrm{PMV})\,, \tag{1}$$

where SAD (sum of absolute difference) is a widely used matching function. The SAD of two $N \times N$ blocks is defined as follows:

$$\mathrm{SAD}\,(O, R) = \sum_{i=1}^{N} \sum_{j=1}^{N} \left| O\,(i, j) - R\,(i, j) \right|. \tag{2}$$

λ_M is a Lagrangian factor, and MV (motion vector) is the displacement between the current block and its matching block. PMV (predicted motion vector) is a synthetic vector from the MVs of the neighboring PUs. R(MV − PMV) denotes the required bitrates to encode the value of (MV − PMV).

2.3. Related Work. The full-search algorithm, being one of the basic ME methods, searches all positions in a search region of the reference frame, where each position stands for a pixel block. Aiming to reduce the number of search points, researchers have proposed various fast ME methods, which only search a small part of positions in the search window with a certain pattern or search the positions which are likely to generate the optimal MV. Some good ones are adopted in H.264/AVC, such as UMHexagonS, EPZS, and TZS. The TZS algorithm is also adopted in the encoder standard of the HEVC reference software [14].

Though fast and practical on a single CPU, most fast ME algorithms, including HS, EPZS, and TZS, cannot satisfy the high parallelism demand of a many-core processor. Therefore, most of the parallel ME research work is on many-core concentrates on the full-search method, which is inherently highly parallel. In Chen and Hang [15]; Cheng et al. [16]; Lee and Oh [17]; Monteiro et al. [18], the parallel full-search method is implemented on the GPU platform, and about 10–100x speed-ups are, respectively, obtained compared with the serial full-search method on single core of a CPU. Although the speed-up of the full-search method is high on GPU platform, its performance advantage is not obvious compared with serial fast search method in HEVC or H.264/AVC on single core of a CPU. This is because the computational complexity of the full-search method is far higher than that of the fast method in HEVC/H.264.AVC.

The design of HEVC considers the parallelism to adapt to the multicore architecture, and three features for parallel processing are proposed, including tiles, entropy slice, and wavefront parallel processing. Therefore, quite a number of parallel approaches are proposed to accelerate the whole HEVC encoder on the multicore platform. In Wang et al. [4], the authors propose a slice parallelization approach to divide a frame into as many as the number of available processor cores, and obtain 9.8 speed-up on a 12-core processor. In Ahn et al. [19], the researchers implement a fast HEVC encoder based on the multicore platform using load-balanced algorithm, slice-level parallelization, frame-level interpolation filter, and SIMD implementation for those computationally intensive parts. About 10x speed-up is achieved compared to the HEVC test model software with acceptable loss of encoding quality. In Yan et al. [20], the researchers make use of the DAG- (directed acyclic graph-) based order to parallelize CTUs and employ the ILPM (Improved Local Parallel Method) within each CTU to exploit the CTU-level and PU-level parallelism and finally obtain 30–40x speed-up for 1920 × 1080/2560 × 1600 video sequences on a 64-core processor compared with that on one core of that.

As for the multicore and many-core systems with powerful single core, such as multicore CPU, Tile, and MIC, HEVC ME algorithm can get a fine speed-up by making full use of the various levels of parallelism, such as tiles, wavefront, slice-level, small-scale SIMD, CTUs, and PUs. However, these kinds of parallel approaches are difficult to be applied on GPU, because of the complicated control statements and limited degree of parallelism. In order to be well applied on many-core processor with thousand computing cores, an algorithm should have as high a degree of parallelism as

that of the platform. *Hierarchical Motion Estimation* (HME) is a good choice in fast ME algorithm, whose degree of parallelism is close to that of full-search method.

HME in Kuhn et al. [21]; Tedmori and Al-Najdawi [22]; Nijad [23]; Nam et al. [24] algorithm combines the advantages of large blocks in a high resolution frame with small blocks in a low resolution frame. In HME, the original frame is subsampled into multiple low resolution frames, and the full-search method is performed on all the resolution frames from low to high. The computational complexity of the HME is slightly higher than some of the other fast search methods but much less than full-search method. On the other hand, the encoding quality of HME is better than that of other fast algorithms. Moreover, the HME algorithm implies rich parallelism if the data dependency can be relaxed. We have applied HME in X264 and implemented parallel algorithm using CUDA on multiple GPUs, Chen et al. [25].

3. Multilevel Resolution Motion Estimation

The traditional fast search methods pay more attention to reducing the computational complexity rather than the high parallelism demand of many-core processors. This leads to three major problems: (1) strong data dependency exists among neighboring blocks, since the processing of the current block depends on the results of its previous adjacent blocks; (2) the memory access is unpredictable as a result of searching the most likely positions of search window; (3) the convergence rates of different blocks are inconsistent because of the strategy of terminating as early as possible. These troubles hinder the application of the traditional fast search methods on the many-core platform on account of the limited degree of parallelism.

In order to overcome these problems, we propose a scalable and parallel algorithm, taking the flexible block structure of HEVC into account, which can also be regarded as an improved HME method. The new ME algorithm is named MLRME and incorporates enough parallelism to meet the need of many-core processors. The main idea of MLRME is to relieve the data dependency among neighboring LCUs and exploit rich parallelism by the following means: (1) multilevel resolution frames are generated. (2) The full search method, rather than fast search methods, is applied on each resolution frame. (3) *Coarsely best MV* (cbMV), rather than the best MV of previous blocks in the original frame, is selected as the central point of search windows for the original resolution frame. (Here, the cbMV of a block refers to the best MV in the half-resolution frame rather than the best MV in the original resolution frame or quarter-resolution frame, although there exists the best MV in each resolution frame.) These means above greatly increase the degree of parallelism of MLRME but may cause the encoding quality to drop at the same time. In order to address this new problem, multiple cbMVs are adopted for each block. Therefore, the MLRME method can perform ME for all LCUs of the current picture in parallel while keeping good encoding quality by combining the methods of relaxing the data dependency and adopting multiple cbMVs.

3.1. Procedure. The MLRME method can contain multiple layers of different resolutions, and the number of layers impacts on the computational complexity and search accuracy. Given a certain resolution of the original frame, more layers in MLRME reduce the computational complexity at the cost of lower search accuracy. In our work, the resolution of the target videos varies from 832×480 to 2560×1600, and thus three levels of resolution are adopted in the MLRME. For higher resolution video, such as 3840×2160, we can add one more subsampling layer to obtain total four different resolutions. In the rest of this article, our discussion is mainly based on MLRME with three resolutions.

The procedure of MLRME is briefly shown in Figure 3. The left part of Figure 3 shows the subsampling process from bottom to top. The right part depicts the main search steps of MLRME from top to bottom. In order to facilitate the analysis, the proposed MLRME method is summarized into the following five steps.

The initial step is to generate the half-resolution frame and the quarter-resolution frame. The half-resolution frame is created by subsampling from the original frame, and the quarter-resolution frame is produced from the half-resolution frame in a similar way. Herein, a subsampled pixel in a lower resolution frame is generated from the corresponding higher resolution frame by using the arithmetic average value of four neighboring pixels, as shown in the left of Figure 3 from the bottom layer to top layer.

In the second step, two substep searches are performed to find the cbMV of each LCU in the lower resolution frames. The first substep is to locate the matching position in the search window of the quarter-resolution reference frame. The center of the search window can be calculated by mapping the center of the current LCU into the quarter-resolution frames. Then the size of the search window is reduced 4 times in the x and y direction, respectively, when compared with their size in the original frame. For the second substep, the cbMV can be calculated by searching in the corresponding search window of the half-resolution reference frame, whose center is indicated by the best MV in the quarter-resolution.

In the third step, the best MV for every partition of each LCU in full-resolution frame is worked out based on cbMVs. For a LCU with size 64×64 in HEVC, there are many hundreds of possible subblocks. Since most partitions of a LCU overlap with the others, there exits much repeated work when computing the SAD of each subblock. In order to avoid repeated computations, the SADs of smallest subblocks with size 4×4 are worked out first. Then, the SADs of other larger subblocks can be calculated based on the smaller ones. Figure 4 shows this process of using the blocks with size of 4×4 to calculate the SADs of blocks with size of 16×16. In HEVC, the SADs of all subblocks from sizes of 4×4 till 64×64 can be worked out by this way.

The fourth step computes the final CU partitions of each LCU, including the CU size and CU depth. After that, the optimal partition size and depth of each LCU can be selected by comparing the J_{MV} value of different partitions among various partitioned layers and selecting the smallest one. The

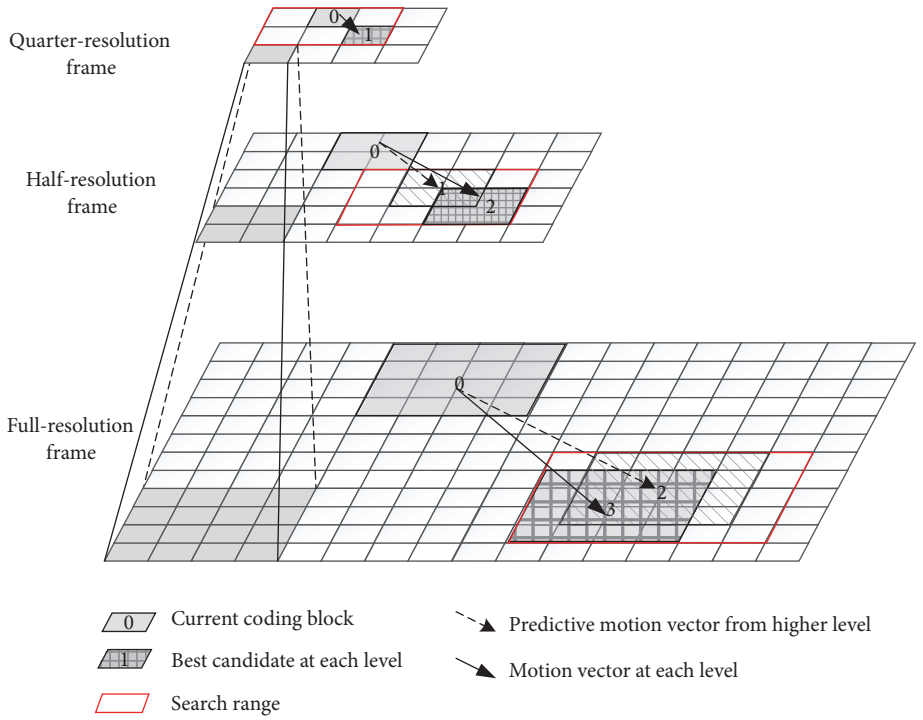


Figure 3: Process schematic of the proposed MLRME. The grey blocks in the left part of the figure show the subsampling process from bottom to top. The right part of the figure shows the main search steps of MLRME from top to bottom.

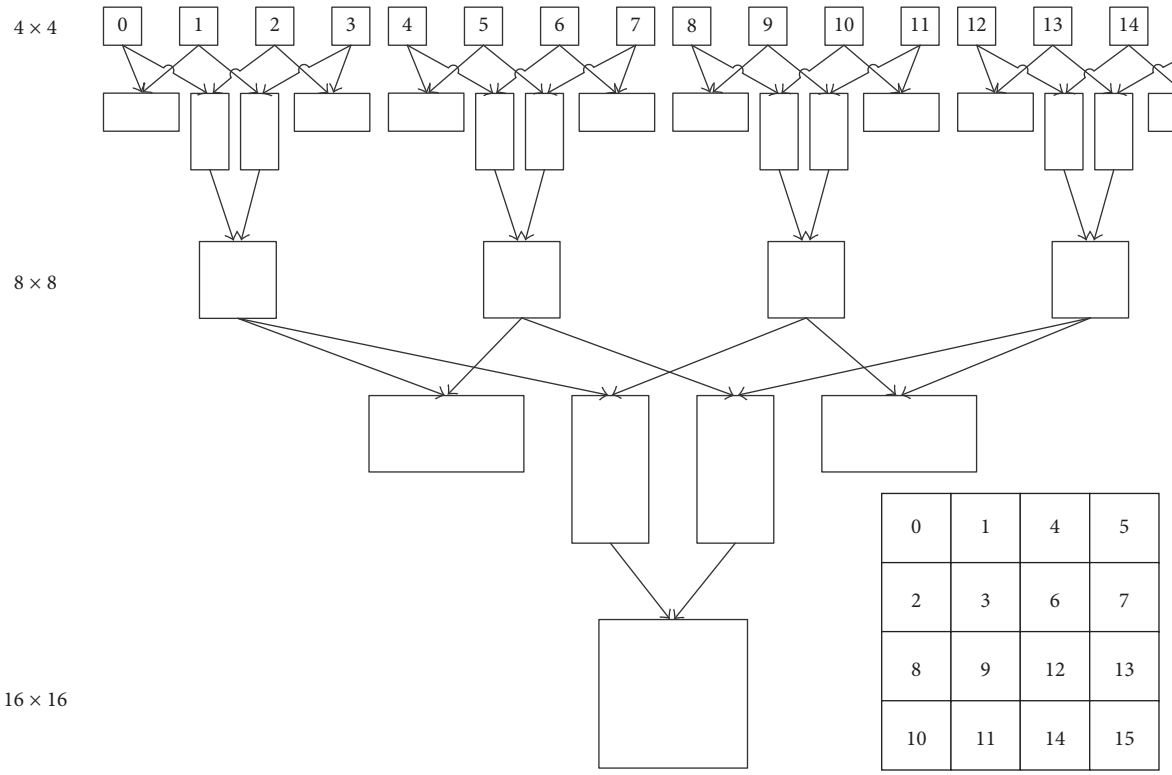


Figure 4: The processing of computing SAD from small blocks to large block.

number of final partitions is considerably smaller than the number of all possible partitions.

In the final step, the quarter-pixel precision MV is figured out for each PU based on its integer-pixel MV (the best MV in original frame). HEVC also adopts quarter-pixel precision MV, which leads to lower bitrates at the cost of higher computational complexity. The resolution of original frame is improved by interpolating half pixel with an 8-tap filter and quarter-pixel with a linear filter. Our MLRME method adopts this method to obtain the quarter-pixel precision MV of each finally partitioned block, which is selected in the fourth step.

3.2. Computational Complexity. In order to analyze the complexity of the MLRME method, some basic assumptions are needed: (1) three-level resolution pictures are used. (2) The full-search method is adopted to search in each level resolution frame to guarantee the best search result and parallelism. (3) Different search ranges are applied in different levels of resolution frames to control the number of search positions. Denote the resolution of input video by $W \times H$ and the search range of full-search algorithm by $M \times N$. Then, in MLRME, the search ranges are respectively $M/4 \times N/4$ in the quarter-resolution frame and the half-resolution frame, and 3×3 in the full-resolution frame. (4) For any block with size $m \times n$ (largest one with size 16×16 in h.264/AVC and 64×64 in HEVC), its size is, respectively, $m/4 \times n/4$ in the quarter-resolution frame and $m/2 \times n/2$ in half-resolution frame.

The computation work of MLRME includes three major parts. (The following equations represent the computation needed for one LCU.) The first part is subsampling to generate the quarter-resolution frame and half-resolution frame. The number of operations is presented in (3); the second part is coarse search in the subsampled frame to obtain cbMV. The complexity is exhibited in (4); the last part is to obtain the best MV at the full-resolution level. The complexity is shown as (5). Therefore, the total work of the MLRME method for one LCU approximates the sum of the three main parts, demonstrated in (6). Here the cost of refining the best MV by using subpixel is not included, which is identical with that in HEVC and only occupies a small proportion of the total computation.

$$\frac{(W \times H) + W/2 \times H/2}{(W \times H)/mn} = \frac{5mn}{4} \tag{3}$$

$$\left(\frac{M}{4} \times \frac{N}{4}\right) \times \left[2 \times \left(\frac{m}{4} \times \frac{n}{4}\right) - 1\right] + \left(\frac{M}{4} \times \frac{N}{4}\right) \times \left[2 \times \left(\frac{m}{2} \times \frac{n}{2}\right) - 1\right] = \frac{5MNmn}{128} - \frac{MN}{8} \tag{4}$$

$$(5 \times 3 \times 3) \times (2m \times n - 1) = 90mn - 45 \tag{5}$$

$$\frac{5mn}{4} + \left(\frac{5MNmn}{128} - \frac{MN}{8}\right) + (90mn - 45) \approx 91mn + \frac{5MNmn}{128}. \tag{6}$$

The complexity of the full-search method is shown as

$$(M \times N) \times (2m \times n - 1) \approx 2MNmn. \tag{7}$$

The ratio of the MLRME method to the full-search method is shown as

$$\frac{91mn + 5MNmn/128}{2MNmn} = \frac{5}{256} + \frac{45}{MN}. \tag{8}$$

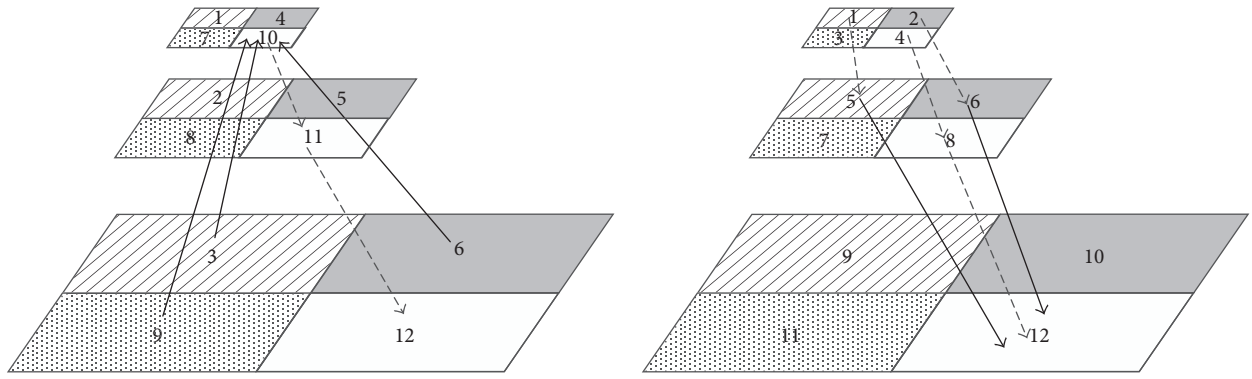
Given that the search window in H.264/AVC is 32×32, the ratio of computational complexity is 6.3%. If the search window in HEVC is 128×128, the ratio of computational complexity is about 3%. It is evident that a larger search window favors MLRME with respect to computational complexity.

3.3. Parallelism. The classic fast search methods are efficient and accurate by using the similarity of pixel values of adjacent blocks. The well-known reason is that the pixel values of adjacent blocks are close and the best MVs of adjacent blocks are good approximate. However, the classic fast search methods are unsuitable for parallel processing due to the data dependency among blocks. The processing of the current block relies on its previous neighboring blocks, including its left, upper-left, upper, and upper-right blocks in the original frame. As one of the fast search methods, HME inherits the same merits and faults. The rest of this subsection shows the data dependence in HME and the way to break the dependence among neighboring blocks in MLRME.

Figure 5 shows the dependency relationship during the procedure for block 12. Data dependency in HME can be found in Figure 5(a) while those in MLRME are shown in Figure 5(b). The number for each block indicates the processing order of these blocks in a serial program. The same color blocks represent the same blocks in different resolutions. They are always processed in the order from top to bottom, such as $1 \rightarrow 2 \rightarrow 3$, $4 \rightarrow 5 \rightarrow 6$, $7 \rightarrow 8 \rightarrow 9$, and $10 \rightarrow 11 \rightarrow 12$ in Figure 5(a), and $1 \rightarrow 5 \rightarrow 9$, $2 \rightarrow 6 \rightarrow 10$, $3 \rightarrow 7 \rightarrow 11$, and $4 \rightarrow 8 \rightarrow 12$ in Figure 5(b). These data dependencies are inevitable for the method with multiple resolution frames but have no effect on the parallelism among different blocks. In Figure 5, these data dependency relationships are shown by dashed lines.

For HME, in Figure 5(a), block 12 depends upon block 11, block 11 relies on block 10, and block 10 directly depends upon blocks 3, 6, and 9. Therefore, block 12 indirectly depends upon blocks 3, 6, and 9, which are the neighbors of block 12 in the original frame. Namely, block 12 cannot be processed until its neighbors, block 3, 6, and 9, have been processed. Taking the other dependencies into consideration, the processing order can only be strictly serial: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11 \rightarrow 12$. The parallelism of HME is thus limited by the data dependency among different blocks in the original frame, which are shown by solid lines.

The proposed MLRME algorithm successfully overcomes the shortage by only using cbMVs as the center of search windows for the current block in the original frame. The method does not use the correlation among adjacent blocks, which can improve the search efficiency. As is shown in



(a) Data dependency in HME

(b) Data dependency in our MLRME

Figure 5: Data dependency among blocks in HME method and the proposed MLRME. The number represents the processing sequence of all blocks in multiple resolution frames.

Table 1: Parallelism analysis of MLRME algorithm.

Step	Content	Max parallelism
1	Subsample into half-resolution image	$WH/4$
	Subsample into quarter-resolution image	$WH/16$
2	Search for the best MV in quarter-resolution image	$WHMN/(16n^2)$
	Search for the best MV in half-resolution image	$WHMN/(16n^2)$
3	Search for the best MV in full-resolution image	$45WH/n^2$
4	Decide the CU and PU partition of LCU	WH/n^2
5	Refine the best MV by using subpixel	$9WH/n^2$

Figure 5(b), block 12 depends upon blocks 5, 6, and 8, while blocks 5, 6, and 8, respectively, depend upon their subsampling blocks 1, 2, and 4. However, blocks 1, 2, and 4 no longer depend upon any other blocks. Thus the blocks in the same resolution frame can be processed in parallel from top to bottom, such as $(1, 2, 3, 4) \rightarrow (5, 6, 7, 8) \rightarrow (9, 10, 11, 12)$. The parallelism of the MLRME algorithm is exploited through this way. According to the steps of MLRME, the best MVs of all LCUs in the quarter-resolution frame can be found out first in parallel, and then the cbMVs of all LCUs in the half-resolution frame can be obtained based on the search results in quarter-resolution frame. Finally, the best MV of blocks in the full-resolution frame can be worked out in parallel based on cbMVs.

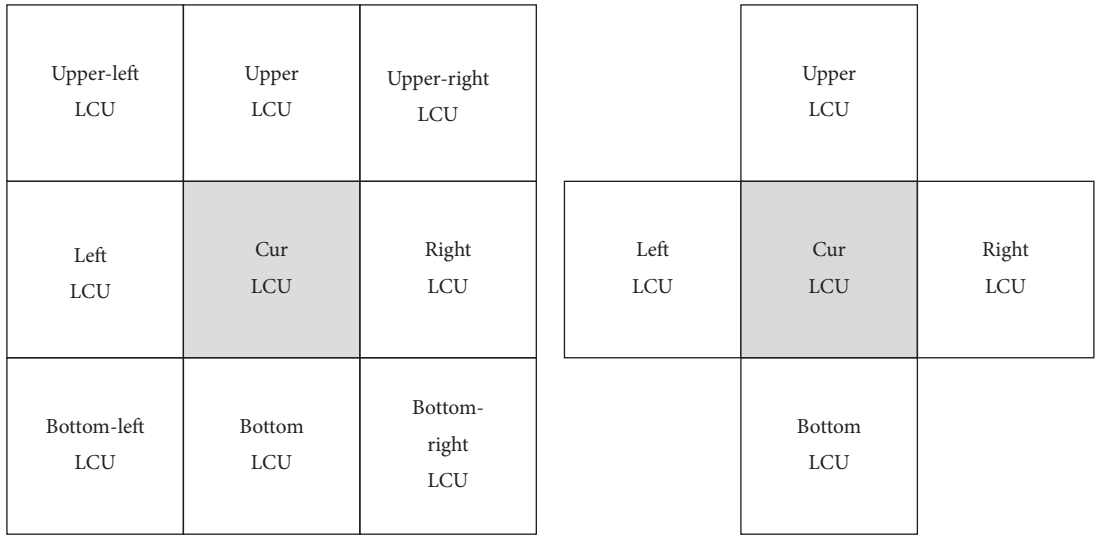
A quantitative analysis of the parallelism of MLRME is shown in Table 1 step by step; presuming that the resolution of the input video is $W \times H$, the LCU size is $n \times n$ and the search window is $M \times N$.

From Table 1, it can be concluded that the MLRME algorithm is suitable for a many-core processor, such as GPU with thousands of cores, because of high parallelism. The maximum parallelism of most steps can achieve $O(WH)$, which can satisfy the parallelism demand of many-core processor. Only the parallelism of step 4 is relatively low but can still reach hundreds.

3.4. Multiple Search Windows. As in the third step of the MLRME, the best MV for every final partition of LCU in the full-resolution frame is computed based on cbMVs. Since the cbMV may not be as accurate as the predictive MV in classic fast ME methods, more cbMVs are employed to improve the search results in MLRME.

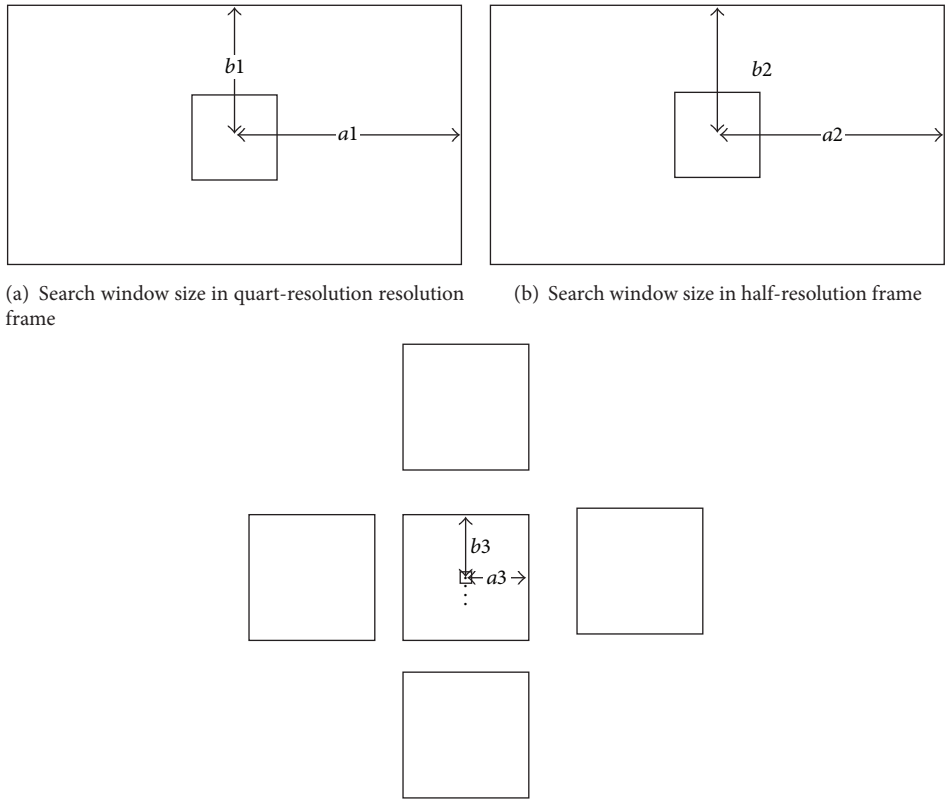
As shown in Figure 6(a), there are at most nine directly related cbMVs for each LCU. A cbMV corresponds to a search window. It is obvious that using all nine cbMVs brings the best result as well as the most computational complexity. In the MLRME method, five candidates, including cbMVs of the current LCU and its upper, left, and right neighboring LCUs, are used, as shown in Figure 6(b). The reasons for choosing five candidates, rather than nine ones, are as follows: (1) five candidates require about half of the computational complexity. (2) These five candidates can achieve very close results with nine candidates. It is known that the search windows provided by neighboring candidates are overlapped with each other in most cases. For example, in Figure 6(a), the search window provided by cbMV of upper-left LCU is overlapped with search windows provided by cbMVs of the upper LCU and left LCU. On the other hand, the corner candidates are farther away from the center LCU than the direct adjacent candidates. Therefore, the corner candidates are not adopted for the sake of reducing the computational complexity. The efficiency of the MLRME with five search windows is showed in Section 5.2.

Given the search sizes in each resolution frame, we can work out the total search range for each block. According to the MLRME method, we assume that the search sizes in each resolution frame are given as follows: (1) the quarter-resolution frame has one search window with size $2a_1 \times 2b_1$; (2) the half-resolution frame also owns one search window



(a) All candidates (b) Our selected candidates

FIGURE 6: The candidates from subsampled frames. (a) All available candidates in subsampled frame. (b) The selected candidates in MLRME.



(a) Search window size in quart-resolution resolution frame (b) Search window size in half-resolution frame

(c) Search window size in full-resolution frame

FIGURE 7: The sizes of search windows in each layer.

with size $2a_2 \times 2b_2$; (3) the full-resolution frame has five search windows with the same size $2a_3 \times 2b_3$, as shown in Figure 7; (4) the distance between two LCUs in the original frame is n. Since one pixel in the quarter-resolution and half-resolution frames, respectively, represents four pixels and two pixels in the full-resolution frame, the total search range can reach $h_t \times v_t$, when mapping back to the full-resolution frame:

$$\begin{aligned} h_t &= 2 \times \left(4a_1 + 2a_2 + a_3 + n\right) = 326 \\ v_t &= 2 \times \left(4b_1 + 2b_2 + b_3 + n\right) = 326. \end{aligned} \tag{9}$$

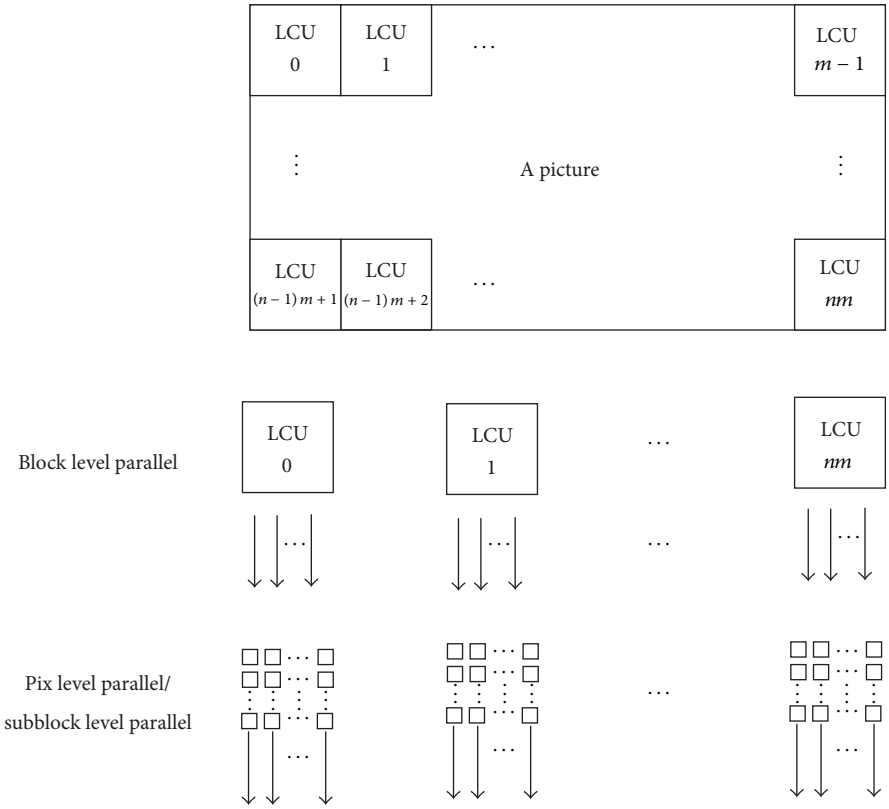


Figure 8: The parallel model.

According to the assumptions in Section 3.2, the values of the parameters are assigned as follows:

$a_1 = 16, b_1 = 16; a_2 = 16, b_2 = 16; a_3 = 3, b_3 = 3; n = 64.$

For a full-HD video with resolution 1920×1080, the final search range can, respectively, cover 1/6 and 1/3 of the full frame in horizontal and vertical directions. Therefore, the MLRME method can capture the fast moving objects, which pass the video by more than 6 frames in horizontal directions and 3 frames in vertical directions.

In the classic fast ME methods, the position of search window can inherit from the best MVs of previous adjacent blocks, so its search range can reach farther from the original position of the current block. Namely, the search range of MLRME is smaller than classic method in some cases. Despite all that, Section 5.1 shows MLRME achieves close encoding quality to the ME method in HEVC test model. Actually, the search range of MLRME is large enough to capture overwhelming majority of moving objects, because such fast moving objects which can pass the video from one side to the other within 6 frames are few in video sequences.

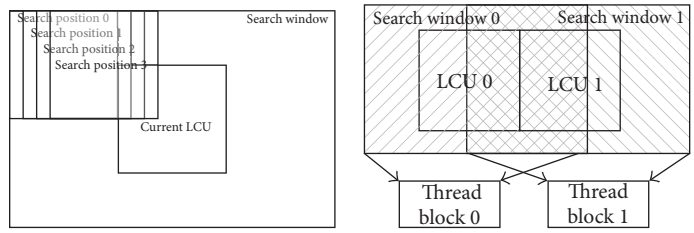
4. CUDA Implementation and Optimization of MLRME

In this section, we implement MLRME for GPU and employ some optimizations to the parallel program.

Parallel programs based on CUDA include serial codes and parallel kernels. The serial codes run on the host side, and the CUDA kernels run on GPUs in the *single-instruction-multiple-thread* (SIMT) style NVIDIA [26]. The GPU threads can be regarded as having two layers, the first layer is made up of blocks. The threads belonging to different blocks cannot communicate with each other. The other layer consists of threads in each block, which can share data through high speed shared memory.

4.1. The Parallel Model. In the basic CUDA implementation of MLRME, a kernel is created for each step, which is simple and clear. There are five kernels. For each step, output data mainly is the SAD value and motion vector of each CU size. Motion vector of each CU size in lower resolution frame is the input data in higher resolution frame. By this way, the search start position in higher resolution frame is decided by the output motion vector in lower resolution frame.

In each kernel, a LCU is processed by a thread block in every step of the MLRME algorithm. Figure 8 shows our basic parallel strategy. The number of thread blocks remains the same as the number of LCUs in a frame for all steps of the MLRME method. In fact, the number of LCUs is constant in a video sequence with the same resolution. The basic parallel implementation of each LCU is shown as the lower part of Figure 8, where all the threads in the same block deal with all partitions of a LCU. The number of threads per block may be adjusted according to the computation content of each step.



(a) Overlapped data access among threads

(b) Overlapped data access among blocks

Figure 9: Overlapped data access modes.

4.2. Optimizations. In order to utilize the computation power of a GPU efficiently, we have employed some optimizations to the parallel MLRME, including the following.

(1) Take full advantage of the shared memory of the GPU memory hierarchy. Many intermediate results and repeated access data can be shared between threads to reduce the overheads of global memory access. It should be noticed that the input/output data, only accessed once, do not need to be carried into the shared memory. For example, in the second step of searching for cbMV, one thread calculates the SAD value of one search position. When computing the SADs in the same search windows, there are many repeatedly accessed data, which are contained in multiple nearby search positions, as shown in Figure 9(a). Namely, every pixel value will be accessed more than once by different threads, and these threads can be in the same thread block or in the different thread blocks. In the case of the same thread block, pixel data should be transported from GPU's global memory to the shared memory for reducing the access cost. Therefore, data of one search window for one LCU are loaded into the shared memory of the corresponding thread block, as shown in Figure 9(b).

(2) Reduce the overhead of kernel launching by merging small kernels. There is another advantage: avoiding intermediate results to be written to and read from global memory. These optimizations greatly enhance the performance. For example, we merge kernels $1, 2, 3$ of basic CUDA implementation of MLRME into one big kernel of optimization of MLRME. Namely, in the optimization of MLRME, three kernels finally remain in the optimization of MLRME. Kernel 1 of optimization of MLRME includes steps $1, 2, 3$. Kernel 2 includes step 4. Kernel 3 includes step 5 While merging kernels $1, 2, 3$, we have tested different thread configuration in a thread block and selected the optimal one: 8×8. The performance with different thread number of each thread block, including 4×4, 8×8, and 16×16, is tested. The result shows the 8×8 thread block configuration leads to best performance. Table 1 shows the maximum parallelism of each step, which is rather rich in our programming, especially for high resolution video. Therefore, we merged steps $1, 2, 3$ into one kernel which is easy, since we just take care of the smallest maximum parallelism step. In fact, the smallest maximum parallelism step can meet our demand. When we merge the small kernel together, it seems natural because of rich parallels and the same search range.

Table 2: Test video sequences.

Resolution	Sequence 1	Sequence 2	Sequence 3
832×480	BasketballDrill (BD)	BQMall (BQ)	Keiba (Kb)
1280×720	KristernAndSara (KAS)	FourPeople (FP)	SlideEditing (SE)
1920×1080	Kimonol (Kmn)	ParkScene (PS)	Tennis (Tnn)
2560×1600	PeopleOnStreet (POS)	Traffic (Tff)	—

5. Experimental Results

The experimental hardware environments are described as follows. Three GPUs are used to evaluate the performance of the CUDA program of the proposed MLRME algorithm, including NVIDIA Geforce GTX 460, C2050, and K40c. These three GPUs consist of 336, 448, and 2880 streaming processors, respectively. The host is equipped with an Intel i7 2600 CPU with 4 cores, and the Ubuntu 14.04 operating system is used. For NVIDIA GPU programming, CUDA IDE 6.5 is used. The video sets involve four resolutions, and each resolution includes two or three different video sequences, as shown in Table 2. The video sequences are downloaded from the official test set of HM. When encoding, the frames are set as one I frame followed by nine P frames for each ten frames.

5.1. Encoding Quality. In order to evaluate the encoding quality of the MLRME algorithm, we implant it in HM12.0 and compare with the average PSNR of the fast search method in HM12.0. PSNR of Huynh-Thu and Ghanbari [27] is one of the most widely used objective measurements to evaluate the encoding quality of different encoders. Figure 10 shows the average PSNR with different methods for different video sequences, where we use 5 candidates of cbMVs. The subpicture represents the results with different quantization parameters (QP) (increasing the QP leads to fewer bitrates at the cost of losing more information). A higher PSNR value represents better quality of video coding. We can see that the proposed MLRME method with five or nine search windows causes no distinct decline of the video coding quality compared with the fast method in HEVC. The decrease of the average PSNR is less than 1.5%.

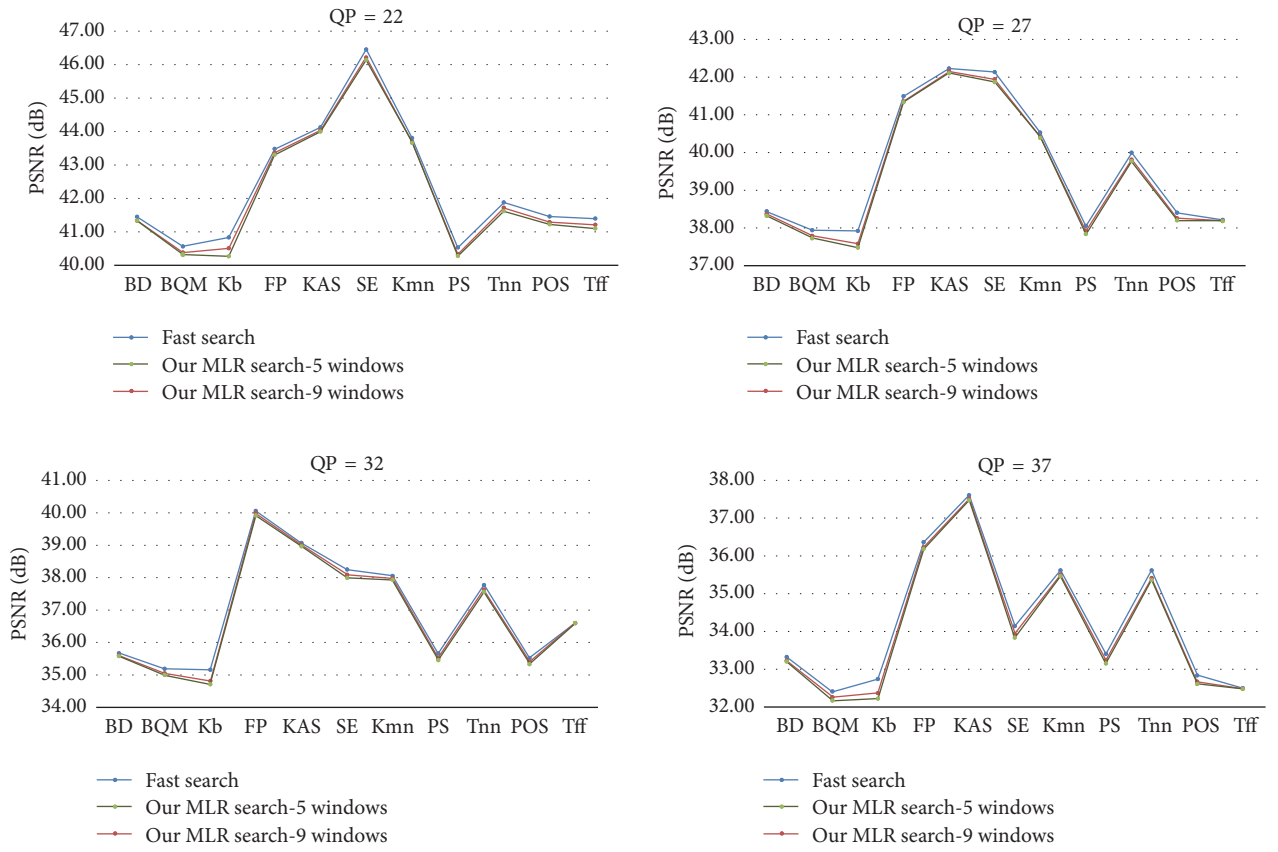


Figure 10: PSNR of the fast search and the MLRME search with 5 candidates for QP values of 22, 27, 32, and 37.

5.2. The Efficiency of Multiple Search Windows Method. The efficiency of multiple windows includes two aspects: encoding quality and computational complexity. In Figure 10, the PSNR of our MLRME with nine or five search windows has been showed, where the PSNR values of them are very close. It demonstrates that the five search windows can achieve almost as good quality as that with the nine search windows. On the other hand, it is quite clear that the computational complexity of our MLRME with five search windows is smaller than that with nine search windows. In order to get the exact value of its computational complexity, we test the running time of their serial codes on CPU, and the results are displayed in Figure 11. The experimental results show that our MLRME with 5 searching windows can save 20 percent of running time in comparison with that with 9 searching windows on a single core of the CPU.

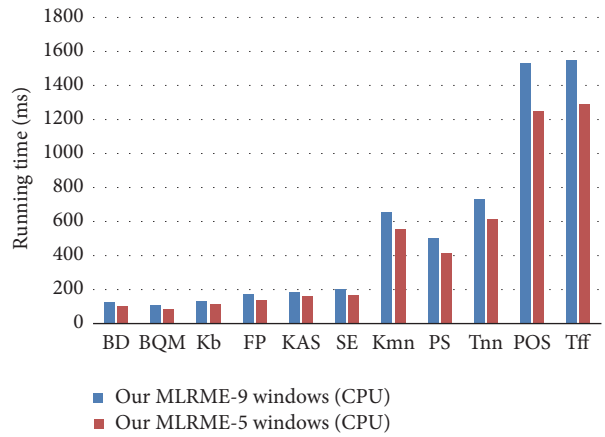


Figure 11: The running time of our MLRME with five search windows compared with that with nine search windows on a single core of the CPU.

5.3. Parallel Performance on GPU. As previously mentioned, for each LCU in a frame, the ME algorithm needs to be invoked hundreds of times in the HEVC test model (HM), so much of complicated invoking ME impedes the effective measurement of its precise execution time in HM. Therefore, we extract the single ME out from the whole test model HM when we need to measure the running time of ME. In this article, we adopt two steps to evaluate the parallel performance on GPU and can indirectly compare the performance of MLRME on GPU with the fast ME in HEVC on single core of CPU. The first step is to measure the serial execution time of HM12.0 with MLRME as the ME algorithm and compare the performance with the original HM12.0; both are based on a single core of the CPU. The second step is to compare the performance of full-search method and MLRME on GPU and their serial performance on a single core of the CPU.

Table 3: Encoding time (in seconds).

Method	Fast ME	MLRME	Increase
BD	90.636	101.556	12%
BQM	76.814	87.796	14%
Kb	102.336	114.269	12%
KAS	137.717	140.691	2%
FP	144.456	159.166	10%
SE	150.681	166.794	11%
Kmn	544.253	565.013	4%
PS	403.182	423.771	5%
Tnn	602.348	625.887	4%
POS	1176.273	1251.465	6%
Tff	1205.733	1289.681	7%

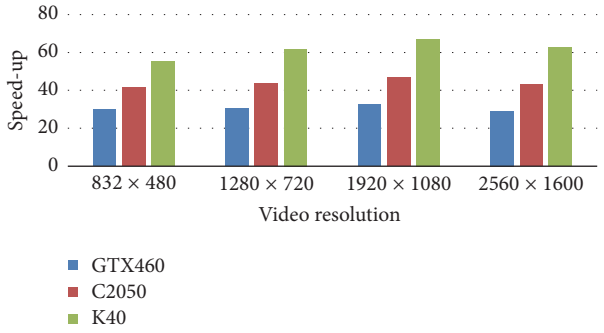


Figure 12: The average speed-up of video sequences with different resolutions on three GPUs compared with MLRME on a single core, including data transfer time.

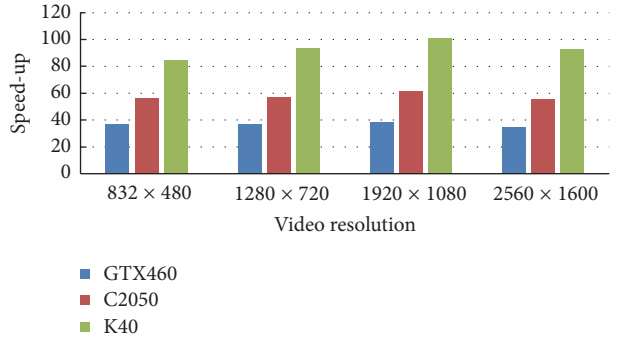


Figure 13: The average speed-up of video sequences with different resolutions on three GPUs compared with MLRME on a single core, not including data transfer time.

At first, we evaluate the execution time of HM12.0 with the fast search method and the MLRME method. Table 3 shows the execution time of encoding 10 frames on a single core of the CPU. We can see that the execution time of the whole encoding algorithm with the serial MLRME method is longer than that of the fast search method, increased by 2%–14%. The major reasons include the following: (1) the complexity of the MLRME method is higher than the fast ME in HM12.0; (2) the heuristic search and early termination mechanism of the fast ME help reducing the search time. The experimental results show the computational complexity of our MLRME is higher than that of the fast method in HM12.0, but the difference is not very big.

5.4. The Proportion of Computation and Communication. In order to evaluate the parallel efficiency of MLRME, the CUDA program, respectively, executes on the three GPUs mentioned above. For each test video sequence, the average speed-up on these three GPUs is obtained, while the performance on a single core of the CPU is used as the baseline. For each resolution, we test two or three video sequences, respectively, on these three GPUs and a single core of the CPU. The results are shown in Figures 12 and 13, where Figure 12 includes data transfer time, but Figure 13 does not include data transfer time. We find that the MLRME method can achieve about 30–60x speed-up on a GPU compared to that on a single CPU core when considering the data transfer time. If ignoring data transfer time, the speed-up can reach 35–100x. Figure 13 also shows that the speed-up increases from GTX 460 to K40c, since the number of streaming processors increases. This suggests the MLRME method is highly parallel and scalable. According to the result of Figure 12 and considering that the fast ME in HEVC is faster than MLRME on single core of CPU, our parallel MLRME on GPU can achieve about 25–50x speed-up. In fact, the parallel MLRME on these GPUs can reach the processing rate with about 25 fps for all test video sequences.

We can also conclude that the speed-up of different resolutions is different on the same GPU. The speed-up increases from 832 × 480 to 1920 × 1080, since the number of LCUs can provide increasing parallelism to satisfy the parallelism demand of the GPU. Then, the speed-up decreases from 1920 × 1080 to 2560 × 1600, and we believe that this anomaly is due to remaining inefficiencies in data access, which will be an issue for future improvements.

The parallel full-search method has been implemented and optimized on the GPU platform in many previous researches, and about 10–100x speed-ups are, respectively, obtained compared with the serial full-search method on single core of a CPU. Therefore, we do some experiments to compare with traditional full-search algorithm. We test four kinds of programs, including full search on a single core of the CPU, full search on GPU K40, our MLR method on a single core of the CPU, and our MLR method on GPU K40. For all the four programs, we test them with our test video sequences, and the data transfer time is not considered. The performance of full-search algorithm on single CPU core is regarded as the baseline; we can obtain the speed-up of the other three program, as is shown in Figure 14. We can find that the speed-up of the full search on GPU is 60–150x, and the speed-up of our MLR method on a single core of the CPU can also reach about 40x. That means the performance of full-search method on GPU is just about twice the performance of our MLR method on a single core of the CPU. We can also notice that the performance of our MLR

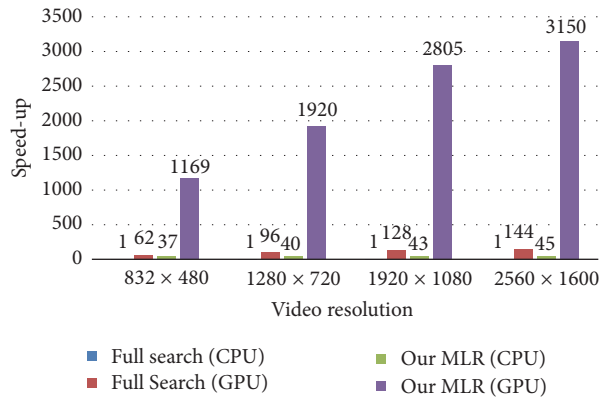


Figure 14: The speed-up of full search on GPU K40, our MLR on a single core of the CPU, our MLR on GPU K40 with different resolutions, compared with the performance of full search on a single core of the CPU, not including data transfer time.

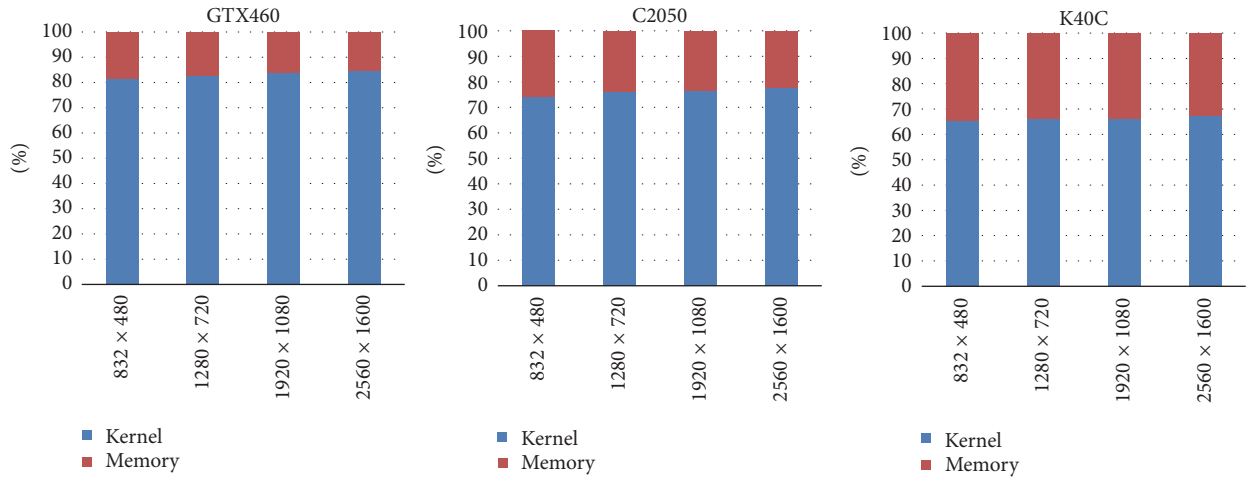


Figure 15: The proportion of kernel computation and data communication with different resolutions on different GPUs.

method on GPU is the highest and excellent, whose speed-up can reach 1000–3000x compared to the baseline. Namely, the performance of our MLR method on GPU is over 20x compared to the performance of the full-search method on GPU. That is why we try our best to improve the parallelism of the fast search algorithm, and we can obtain very high speed-up.

We collect separately the computing time and data transmission time on the three GPUs. Figure 15 shows the proportion of computing kernel and data communication with different resolutions on GTX 460, C2050, and K40c. The proportion of data transmission increases with the increasing GPU performance because the time of computing part becomes shorter and shorter. For the K40c GPU, the proportion of data transmission reaches more than 30%. Therefore it is important to optimize data transmission as far as possible in a system with the work mode of CPU-GPU, when an algorithm has obtained high enough parallelism. In our experiments, we simply transfer data from CPU to GPU, then compute it on GPU, and finally return result data back to CPU. This is an unwise method. We can do some research on overlapping of data transmission and computing in the future work. It becomes more and more significant and valuable after overcoming the problem of parallelism.

6. Conclusion

Based on HME, we have proposed a parallel and scalable ME algorithm, named the MLRME method. The crucial improvements put emphasis on relaxing the data dependency among neighboring LCUs and adopting multiple search windows to keep quality by just selecting search candidates from a lower resolution picture. We presented our proposed MLRME method in detail and analyzed its computational complexity and parallelism step by step. We can see two important advantages of the MLRME algorithm: acceptable complexity and high parallelism, which are suitable for the many-core architecture. We also implanted MLRME into HM12.0 to evaluate the encoding quality compared to the fast search method of HM12.0. The results on several test sequences showed that the loss of PSNR of the picture coded by MLRME is less than 1.5% compared to the results with the fast ME in HEVC. We demonstrated the efficiency of our MLRME with five search windows when compared with that

with nine search windows. Moreover, we implemented and optimized MLRME on GPU to demonstrate its high parallelism and scalability. The experiments showed that MLRME can achieve about 30–60x speed-up on GPU compared to its serial method on a single core of the CPU. Moreover, it is faster than the fast ME algorithm in HEVC on single core of CPU by a factor of 25 to 50, and the parallel MLRME implementation on a GTX 460 GPU can meet the real-time coding requirement of about 25 fps for the 2560×1600 video format, while, for 832×480, the performance is more than 100 fps. Therefore, the MLRME method can meet the requirement of real-time processing on GPU. We also separately collected the computing time and data transmission time on the three GPU platforms. The proportion of data transmission cost increases when the GPU performance enhanced, which demonstrates the importance of optimizing data transmission on CPU-GPU platform, when an algorithm has achieved high enough parallelism.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors gratefully acknowledge support from National Natural Science Foundation of China under NSFC no. 61502509, 61402504, and 61272145; National High Technology Research and Development Program of China (The 863 Program) under Grant no. 2012AA012706; Research Fund for the Doctoral Program of Higher Education of China under SRFDP Grant no. 20124307130004.

References

[1] B. Bross, W. Han, J. Ohm, G. Sullivan, Y. K. Wang, and T. Wiegand, "High efficiency video coding (hevc) text specification draft 10," Standard Draft, No. JCTVC-L1003. Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, 2013.

[2] T. Wiegand, G. Sullivan, and A. LuthraW, Draft itu-t recommendation and final draft international standard of joint video specification (itu-t rec. h.264/iso/iec 14 496-10 avc. Standard Draft, No. JVTG050. Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, 2003.

[3] N. Purnachand, L. N. Alves, and A. Navarro, "Fast motion estimation algorithm for HEVC," in *Proceedings of the 2012 IEEE 2nd International Conference on Consumer Electronics - Berlin, ICCE 2012*, pp. 34–37, Berlin, Germany, September 2012.

[4] X. Wang, S. Li, M. Chen, and J. Yang, "Paralleling variable block size motion estimation of HEVC on CPU plus GPU platform," in *Proceedings of the 2013 IEEE International Conference on Multimedia and Expo Workshops, ICMEW 2013*, San Jose, Calif, USA, July 2013.

[5] T. Koga, K. linuma, A. Hirano, and T. Ishiguro, "Motion compensated inter frame coding for video conferencing," in *Proceedings of the NTC 81*, vol. 4, pp. 961–965, 1981.

[6] L.-M. Po and W.-C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 313–317, 1996.

[7] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no. 2, pp. 287–290, 2000.

[8] C.-H. Cheung and L.-M. Po, "A novel cross-diamond search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 12, pp. 1168–1177, 2002.

[9] A. Hamosfakidis and Y. Paker, "A novel hexagonal search algorithm for fast block matching motion estimation," *Eurasip Journal on Applied Signal Processing*, vol. 2002, no. 6, pp. 595–600, 2002.

[10] C. Zhu, X. Lin, and L.-P. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 5, pp. 349–355, 2002.

[11] C. Zhu, X. Lin, and L. P. Chau, "An enhanced hexagonal search algorithm for block motion estimation," in *Proceedings of the ISCAS 2003. International Symposium on Circuits and Systems*, pp. II-392–II-395, Bangkok, Thailand, 2003.

[12] A. M. Tourapis, "Enhanced predictive zonal search for single and multiple frame motion estimation," in *Proceedings of the Viual Communications and Image Processing 2002*, pp. 1069–1079, FL, USA, January 2002.

[13] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[14] Hevc reference software, https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-12.0/ .

[15] W.-N. Chen and H.-M. Hang, "H.264/AVC motion estimation implmentation on compute unified device architecture (CUDA)," in *Proceedings of the 2008 IEEE International Conference on Multimedia and Expo, ICME 2008*, pp. 697–700, Hanover, Germany, June 2008.

[16] R. Cheng, E. Yang, and T. Liu, "Speeding up motion estimation algorithms on CUDA technology," in *Proceedings of the 2nd Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics, PrimeAsia 2010*, pp. 93–96, Shanghai, China, September 2010.

[17] D.-K. Lee and S.-J. Oh, "Variable block size motion estimation implementation on compute unified device architecture (CUDA)," in *Proceedings of the 2013 IEEE International Conference on Consumer Electronics, ICCE 2013*, pp. 633-634, Las Vegas, NV, USA, January 2013.

[18] E. Monteiro, B. Vizzotto, C. Diniz, B. Zatt, and S. Bampi, "Applying CUDA architecture to accelerate full search block matching algorithm for high performance motion estimation in video encoding," in *Proceedings of the 23rd International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2011*, pp. 128–135, bra, October 2011.

[19] Y.-J. Ahn, T.-J. Hwang, D.-G. Sim, and W.-J. Han, "Implementation of fast HEVC encoder based on SIMD and data-level parallelism," *Eurasip Journal on Image and Video Processing*, vol. 2014, article no. 16, 2014.

[20] C. Yan, Y. Zhang, J. Xu et al., "Efficient parallel framework for HEVC motion estimation on many-core processor," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 12, pp. 2077–2089, 2014.

[21] P. M. Kuhn, G. Diebel, S. Herrmann et al., "Complexity and PSNR-comparison of several fast motion estimation algorithms for MPEG-4," in *Proceedings of the Applications of Digital Image Processing XXI*, pp. 486–499, San Diego, Calif, USA, July 1998.

[22] S. Tedmori and N. Al-Najdawi, "Hierarchical stochastic fast search motion estimation algorithm," *IET Computer Vision*, vol. 6, no. 1, pp. 21–28, 2012.

[23] A. N. Nijad, "A novel hierarchical search algorithm for video compression," in *Proceedings of the International Conference on Advances in Computer and Electrical Engineering (ICACEE)*, pp. 46–50, Manila, Philippines, 2012.

[24] K. M. Nam, J. S. Kin, R. H. Pari, and Y. S. Shin, "A fast hierarchical motion vector estimation algorithm using mean pyramid," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 4, pp. 344–351, 1995.

[25] D. Chen, H. Su, W. Mei, L. Wang, and C. Zhang, "Scalable Parallel Motion Estimation on Muti-GPU system," in *Proceedings of the 2nd International Symposium on Computer, Communication, Control and Automation*, China, Feburary 2013.

[26] NVIDIA, *Nvidia Cuda Compute Unified Device Architecture-Programming Guide Version 2.0. Guidebook*, NVIDIA Corporation, 2003.

[27] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of PSNR in image/video quality assessment," *Electronics Letters*, vol. 44, no. 13, pp. 800-801, 2008.