PAPER

A Micro-Code-Based IME Engine for HEVC and Its Hardware Implementation

Leilei HUANG[†], ***Nonmember***, **Yibo FAN**[†a)], ***Member***, **Chenhao GU**[†], ***and*** **Xiaoyang ZENG**[†], ***Nonmembers***

SUMMARY High Efficiency Video Coding (HEVC) standard is now becoming one of the most widespread video coding standards in the world. As a successor of H.264 standard, it aims to provide a much superior encoding performance. To fulfill this goal, several new notations along with the corresponding computation processes are introduced by this standard. Among those computation processes, the integer motion estimation (IME) is one of bottlenecks due to the complex partitions of the inter prediction units (PU) and the large search window commonly adopted. Many algorithms have been proposed to address this issue and usually put emphasis on a large search window and great computation amount. However, the coding efforts should be related to the scenes. To be more specific, for relatively static videos, a small search window along with a simple search scheme should be adopted to reduce the time cost and power consumption. In view of this, a micro-code-based IME engine is proposed in this paper, which could be applied with search schemes of different complexity. To test the performance, three different search schemes based on this engine are designed and evaluated under HEVC test model (HM) 16.9, achieving a B-D rate increase of 0.55/−0.07/−0.14%. Compared with our previous work, the hardware implementation is optimized to reduce 64.2% of the SRAMs bits and 32.8% of the logic gate count. The final design could support 4K × 2K @139/85/37fps videos @500MHz.

key words: *high efficiency video coding (HEVC), integer motion estimation (IME), hardware implementation, very large scale integration (VLSI)*

1. Introduction

High Efficiency Video Coding (HEVC) is now becoming one of the most widespread video coding standards in the world. As a successor of H.264 standard, it aims to provide a much superior encoding performance.

To fulfill this goal, several new notations like the coding unit (CU), prediction unit (PU), transformation unit (TU) are introduced by this standard. To be more specific, the basic processing unit in HEVC is called the coding tree unit (CTU) which contains one luma coding tree block (CTB) and two chroma CTBs. The size of luma one can be set to 16×16, 32×32 or 64×64. In a general situation, a size of 64 × 64 would be taken to fully explore the performance of HEVC standard. The CTU plays the role of root node of the CU quad-tree while the CU plays the role of root node of the TU quad-tree and PU. In inter prediction, the size of PUs could vary from 4 × 4 to 64 × 64, including the asymmetric motion partitions (AMP). As shown in Fig. 1, 64×64 PUs can split into partitions of size 64×32, 32×64, 64×16, 64 × 48, 16 × 64, and 48 × 64. 32 × 32 PUs can split further into partitions of size 32 × 16, 16 × 32, 32 × 08, 32 × 24, 08 × 32, and 24 × 32. 16 × 16 PUs can split further into partitions of size 16×08, 08×16, 16×04, 16×12, 04×16, and 12 × 16.

Due to those complex partitions and the large search window usually adopted, the integer motion estimation (IME) has become one of the bottlenecks in the HEVC encoder. Many algorithms have been proposed to address this issue and usually put emphasis on a large search window and great computation complexity. For example, Yang *et al.* [1], Yoo *et al.* [2], and Shen *et al.* [3] employ early termination algorithms to reduce the number of search points. However, these methods are useful only for software based encoders due to the irregular processing flow. Sanchez *et al.* [4], Hu *et al.* [5] and Bao *et al.* [6] developed several hierarchical motion estimation algorithms to extend the search window.

Some different but more efficient methods are also adopted to further improve the efficiency of motion estimation (ME) such as affine motion estimation [7] and bit truncation [8].

However, the coding efforts should be related to the scenes. To be more specific, for relatively static videos, a small search window along with a simple search scheme should be adopted to reduce the time cost and power consumption.

In view of this, a micro-code-based IME engine is proposed in this paper, which could be applied with search schemes of different complexity. To test the performance, three different search schemes based on this engine are designed and evaluated under HEVC test model (HM) 16.9, achieving a B-D rate increase of 0.55/−0.07/−0.14%. Compared with our previous design [17], the hardware implementation is optimized to reduce 64.2% of the SRAMs bits and 32.8% of the logic gate count. The final design supports 4K × 2K @139/85/37fps videos @500MHz.

The rest of this paper is organized as follows. Sec-

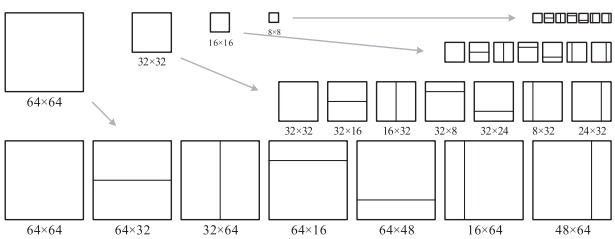


Fig. 1 Inter prediction partitions in HEVC.

Manuscript received December 13, 2018.
Manuscript revised May 20, 2019.
[†]The authors are with the State Kay Lab of ASIC & System, Fudan University, Shanghai, 200433 China.
a) E-mail: fanyibo@fudan.edu.cn (Corresponding author)
DOI: 10.1587/transele.2018ECP5077

tion 2 presents the background, motivations, and contributions of this paper. Section 3 introduces the proposed IME engine and the detailed programming interface. Section 4 gives the corresponding hardware implantation. Section 5 compares the search schemes and hardware implementations with other designs. Finally, Sect. 6 concludes this paper.

2. Background, Motivations and Contributions

In this section, some statistic behaviors and previous works are introduced, followed by motivations and contributions of this paper.

2.1 Statistic Behaviors

To evaluate the relationship between the necessary coding efforts and the coding scenes, some statistic information are collected under HM 16.9.

First, full searches with different search windows are applied to sequences including "BlowingBubbles", a small-resolution video with small motions in pictures, "Kimono", a large-resolution video with small motions, "BasketballDrill", a small-resolution video with large motions, and "BasketballDrive", a large-resolution video with large motions. It can be inferred from Table 1 that videos with a larger resolution and larger motions may need a wider search window, while for videos with a smaller resolution and smaller motions, a narrow search window would be enough.

Second, the best MVs of all PUs and their partitions are collected and analyzed in the unit of LCUs. According to the results, most of them are located in a diamond-shape or square-shape block, while some of them have a strong directionality. For example, Figs. 2 (a) and (b) give the best MVs of all PUs and their partitions in two certain LCUs (in "BasketballDrive"). The horizontal axis stands for the horizontal component of MVs; the vertical axis stands for the vertical component of MVs. It can be clearly seen that the shape of Fig. 2 (a) looks like a right slash, while the one in Fig. 2 (b) looks like a left slash. For such LCUs, a directional search scheme should be favored.

Table 1 Full search with different search windows.

Sequence	Resolution	Motion in Pictures	Search Window	B-D rate Increase
BlowingBubbles	416×240	small	[-8, 8)	0.25%
			[-16,16)	-0.04%
			[-24,24)	-0.06%
			[-32,32)	-0.28%
Kimono	1920×1080	small	[-8, 8)	1.11%
			[-16,16)	0.49%
			[-24,24)	0.20%
			[-32,32)	0.26%
BasketballDrill	832×480	large	[-8, 8)	5.00%
			[-16,16)	2.43%
			[-24,24)	1.48%
			[-32,32)	0.85%
BasketballDrive	1920×1080	large	[-8, 8)	9.58%
			[-16,16)	5.60%
			[-24,24)	3.70%
			[-32,32)	2.73%

Anchor is HM-16.9 with the default setting, encoder_lowdelay_P_main.cfg, but only one reference frame is allowed for each frame

2.2 Previous Works

However, existing algorithms usually put emphasis on a large search window and great computation amount while neglecting those statistic behaviors mentioned above.

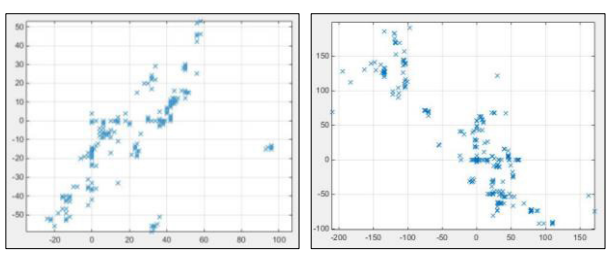
For example, Zhou *et al.* [9] propose a two-step search scheme with a search window of ±211 (horizontal) and ±106 (vertical). A rhombus search is applied for the coarse search step while a full search is adopted during the following refinement search. Dung *et al.* [10] propose a dual-search-windowing (DSW) algorithm which would introduce a secondary search window. Jiang *et al.* [11] adopts a search range of ±64 and always split LCUs into four quarter LCUs (QLCUs). In conclusion, a large search windows (±211 × ±106, dual window or ±64) and great computation amount followed (although the computation could be reduced by two-step search, dual search or CTU split, it is still large) are more favored. Of course, those designs are meaningful for large-resolution or large-motion videos.

But, for those videos with small resolutions or small motions, they may be inefficient. In other words, when dealing with those videos, it is possible to save a great power and bandwidth by reducing the complexity. Unfortunately, the above designs did not discuss about this topic.

2.3 Motivations

A programmable IME engine may be an appropriate choice to solve the above problems. To be more specific, when the target videos are relatively static or a fast encoding is required or the low-power mode is enabled, the engine could be programmed to execute some simple algorithms which costs lesser time and power; when the target videos have lots of fierce motions or the time cost and power consumption are not so cared, the engine could be programmed to execute some complex algorithms which ensures a good coding performance.

In our previous work [17], a hardware-oriented algo-



(a) (b)

Fig. 2 Distributions of the MVs in some certain LCUs.

rithm and the corresponding hardware implementation is proposed. By adopting the horizontal-vertical (H-V) reference SRAM and reference register array with eight updating directions, the implementation is quite suitable for transplanting different kinds of search algorithm.

However, the addressing logic of the previous work [17] is designed for a specific algorithm, and as a result, it lacks in flexibility. Due to this reason, this paper aims to put forward a flexible IME engine which could be easily programmed to execute algorithms of different complexity. The challenge mainly lies in how to provide enough flexibility and, at the same time, keep a simple programming interface.

Besides, the hardware cost of our previous implementation [17] is relatively high when compared with other designs. This paper also puts many efforts into reducing the hardware cost without affecting the coding performance as well as the throughput.

2.4 Contributions

The contributions of this paper are given as follows:

i. We proposed a micro-code-based IME engine.
ii. We optimized our previous hardware implementation.
iii. We evaluated three different search schemes based on this engine.

3. The Proposed IME Encoding Engine

As an engine, the proposed one is not designed for a certain algorithm. Instead, it executes IME searching in the unit of search steps and those search steps could form different search algorithms.

To be more specific, in each step, this engine searches all the PUs and their partitions contained in one quarter LCU (QLCU) according to the same settings including starting point, searching shape and down-sample rate; in different steps, those settings are configured to different values.

In this way, both simple searching algorithms and complex ones could be supported by the proposed design. The only difference is that the former one may consist of fewer steps and the latter one may consist of more steps, as the schemes listed in Sect. 5.1.

3.1 Starting Point

In a general case, IME may search from the center point of the search window, namely, point $(0, 0)$. However, by collecting some statistic information from previous LCUs or previous frames, a better starting point may be adopted. In view of this, the starting point could be directly configured in each step. However, this may need a closely-coupled CPU which updates the starting points every frame or even every LCU.

To make this process more automatically, the proposed engine could be configured to acquire the starting points from the MVs got in previous steps. This is usually adopted when previous steps execute a larger-region search and the current step executes a smaller-region search.

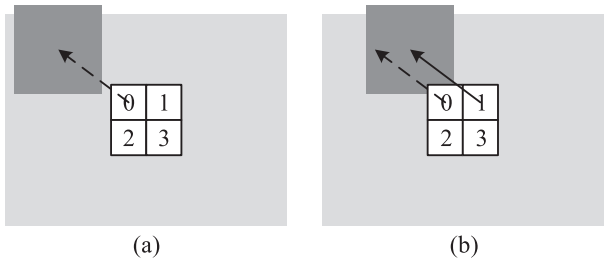


Fig. 3 Automatically acquiring of starting points.

Two examples are provided here to help the understanding.

i. As shown in Fig. 3 (a), the rectangle block colored with lighter gray are searched with a down-sample rate of 1/4. The dotted arrow denotes for the MV of the 32×32 PU got after this search.
To refine it, the IME engine may start from this MV and search the square block colored with darker gray without down sampling.
ii. As shown in Fig. 3 (b), the rectangle block colored with lighter gray are searched to get the best MVs of all PUs and their partitions contained in QLCU 0. The dotted arrow denotes for the MV of the 32×32 PU got after this search.
To get the best MVs of all PUs and their partitions contained in QLCU 1, the IME engine may start from this MV and search the square block colored with darker gray. Of course, the offset between QLCU 1 and QLCU 0 should be applied to the starting point as illustrated by the solid arrow.

3.2 Searching Shape

The basic searching shape is a hexagon. However, by changing the width, height and slopes of edges, it can be transformed into various shapes.

Several possible shapes are illustrated in Fig. 4. Among which, Fig. 4 (a) shows the basic shape, a hexagon.

However, by specifying smaller $\theta 0 \sim 3$, namely, smaller slopes of the four declining edges, a hexagon could be transformed into to a diamond shape as shown in Fig. 4 (b); or by specifying larger $\theta 0 \sim 3$ (90°), it could be transformed into a rectangle shape as shown in Fig. 4 (c). Even shapes with certain directions could be got by specifying an unequal $\theta 0 \sim 3$ as shown in Fig. 4 (d).

Different shapes are suitable for different scenes. For example, the directional shape described in Fig. 4 (d) may be suitable for LCUs which have similar MV distributions as the one shown in Fig. 2 (a).

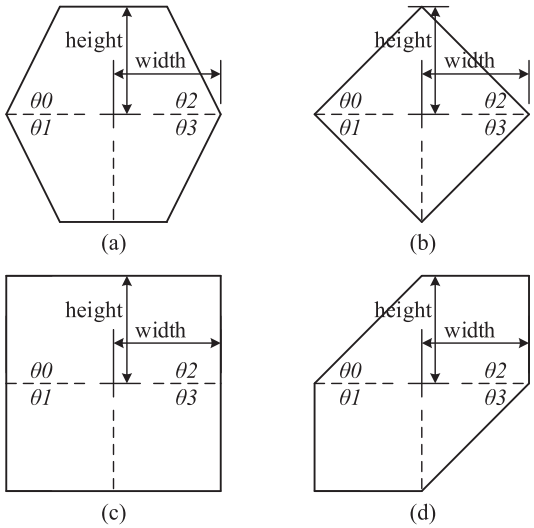


Fig. 4 Possible searching shapes.

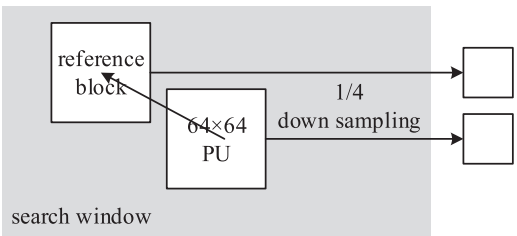


Fig. 5 Down-sample search.

3.3 Down Sampling

On one hand, a 32×32 instead of a 64×64 SAD tree and register array are adopted in this engine in view of the hardware cost. As a result, MVs of the 64 × 64 PU and its partitions cannot be directly got.

In a conventional way, this is done by iteration. Namely, the four 32×32 PUs in the 64×64 PU are searched successively, the related costs of each search are accumulated and stored with one large buffer. This approach may be accurate, but the hardware cost is relatively large considering of the buffer used.

In view of this, the proposed engine searches the 64×64 PU and its partitions by down sampling. As shown in Fig. 5, both the 64 × 64 PU and its reference block are reduced to a 32 × 32 block by 1/4 down sampling and processed with a 32 × 32 SAD engine. The best MVs and costs of the down-sampled 32 × 32 block and its partitions is taken as those of the 64 × 64 PU and its partitions. (Costs need to be scaled back). In a similar way, the best MVs and costs of the four 16 × 16 down-sampled blocks and their partitions are taken as those of the corresponding 32 × 32 PUs and their partitions, so as the 16 × 16 PUs, 8 × 8 PUs and their partitions.

On the other hand, hierarchical search strategies are favored by lots of IME designs, which usually consist of two different searches, a coarse search and a refined search. The former one is performed to cover a large region with an acceptable cycle cost, while the latter one is performed to achieve a refined result by searching around the MVs got in the coarse search. As a matter of fact, down sampling is one effective way to implement coarse search and it shows a good performance according to the B-D rate results listed in Table 3.

starting point	type	x, y / source	
searching shape	width	height	slope 0~3
down-sample rate	type	rate / QLCU id	

Fig. 6 Formats of the micro-codes.

3.4 Micro-Codes

The proposed engine is programmed through micro-codes. The bit length of the micro-code is 40, among which, 14 bits are used to describe the starting point, 23 bits are used to described the searching shape, 3 bits are used to described the down-sample rate. Detailed code format is listed in Fig. 6.

i. For the starting point, 1 bit is used to indicate whether it is got by directly configuration or automatically acquiring. For the former situation, another 15 bits are used to indicate the value of the starting point (x, y). X component occupies 7 bits covering a range of $[-64, 63]$, y component occupies 6 bits covering a range of $[-32, 31]$. For the latter situation, another 5 bits are used to indicate the acquiring sources. Supported sources includes MVs of 64×64 PU, four 32×32 PUs and sixteen 16 × 16 PUs.
ii. For the searching shape, 6 bits are used to indicate the width, covering a range of $[0, 63]$; 5 bits are used to indicate the height, covering a range of $[0, 32]$; 3 bits are used to indicate the slope of each θ. Supported slope includes infinite, 4, 2, 1, 1/2 and 1/4.
iii. For the down-sample rate, 1 bit are used to indicate whether it is down sampled or not. For the former situation, another 1 bit is used to indicate the down-sample rate, 1/4 or 1/16. For the latter situation, another 2 bits are used to indicate the QLCU to be searched.

4. The Proposed Hardware Implementation

In our previous work [17], a related implementation is proposed, which put forward several novel structures like the horizontal and vertical (H-V) reference SRAM, data compression based on 4 × 4 blocks and SAD with PU-level chip selections to support the 2-D data reuse, reduce the size of reference SRAMs and save the power consumption. The final performance in B-D rate loss is quite attractive, but the hardware cost is relatively large and the search algorithm lacks in flexibility since it is fixed by hardware.

In view of this, an optimized implementation is given

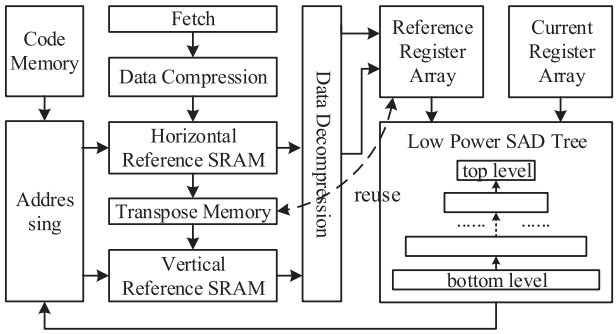


Fig. 7 Structure of the optimized implementation.

in this paper. As shown in Fig. 7, the optimized implementation still adopts the data compression and de-compression logic, the H-V SRAM and the low-power SAD tree, but the following optimizations are made.

i. the addressing logic is re-designed to support the micro-codes introduced in Sect. 3.4
ii. the updating logic of reference register array is simplified to reduce the hardware cost
iii. the reference register array is reused to remove the transpose memory
iv. pixel bit truncation is evaluated to reduce the hardware cost of H-V SRAMs, register arrays and the SAD tree

4.1 Addressing Logic

In our previous implementation [17], the addressing logic is designed for the specific algorithm proposed, which does show a good performance in B-D rate loss but lacks in flexibility.

In the optimized implementation, the addressing logic is re-designed to support the micro-codes introduced in Sect. 3.4, which could offer flexibilities in the starting point, searching shape and down-sampling rate. Once started, the addressing logic reads and executes the micro-codes stored in code SRAM one by one. Those micro-codes stored in code SRAM could be pre-programmed according to the scene or updated every frame or even every LCU if some statistic information could be collected and analyzed on the fly.

4.2 Updating Logic

In our previous implementation [17], 8 updating directions are supported by the reference register array as shown in Fig. 8 (a), where, squares denote for the reference register array; gray parts denote for the neighboring pixels to be updated; black dots stand for search points; arrows stand for search directions. All directions could be updated in one cycle thanks to the H-V reference SRAM adopted. Although this makes the lined-based search quite efficient, it leads to a complex updating logic in reference register array.

In the optimized implementation, although H-V reference SRAM is still adopted, only 3 directions are supported,

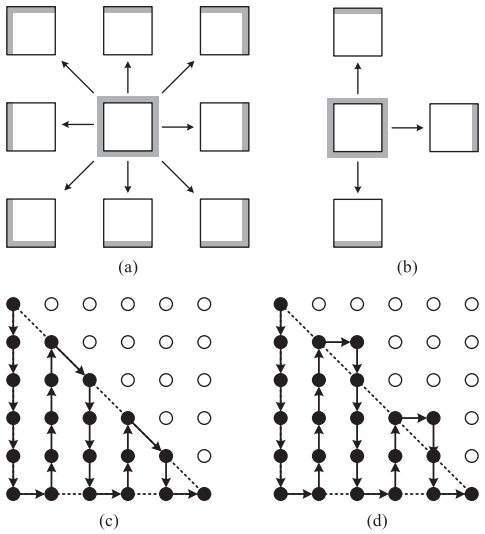


Fig. 8 Comparison between updating logics.

namely, shifting up, shifting down and shifting right, as shown in Fig. 8 (b). Due to this simplification, the search to shapes with hypotenuses would take more cycles, but it also traverses more points as shown by Figs. 8 (c) and (d) and saves a lot of hardware costs according to Table 6.

4.3 Transpose Memory

In our previous implementation [17], an SRAM-based transpose memory [12] is adopted to fill the vertical SRAM without occupying the bandwidth of external memories. However, it still has the following defects.

On one hand, its hardware cost is not low enough to be ignored. When compiled with ARM's memory compiler *rf_sp_hdd_svt_rvt_hvt*, one 32-word 8-bit SRAM occupied about 2K gate count. Since 32 SRAMs are needed, the total gate count is more than 64K.

On the other hand, as a single-port-SRAM-based structure, it could only provide a "half-duplex communication". To be more specific, the reading from the horizontal SRAM and writing to the vertical SRAM could not be launched at the same time.

In the optimized implementation, the reference register array is reused as the transpose memory, which not only reduce the hardware cost but also provide a "duplex communication". The detailed transpose process is shown in Fig. 9. To simplify the illustration of this process, a 4×4 transpose instead of a 32×32 transpose is adopted.

In this example, rows are marked with numbers; columns are marked with letters; blocks are marked with font type, normal, underline, and italic. Those rows, columns and blocks are counted from 0. For example, 1A~D (1A, 1B, 1C and 1D) denotes for the 1st row of the 0th 4×4 block; 0~3D (0D, 1D, 2D and 3D) denoted for the 3rd column of the 1st block; *2A~D* (*2A*, *2B*, *2C* and *2D*) denotes for the 2nd row of the 2nd block. It can be seen from Fig. 9 that

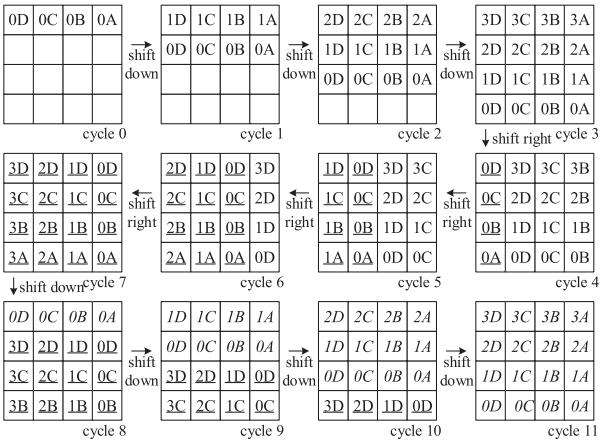


Fig. 9 Reference-register-array-based transpose memory.

i. At cycle 0, the 0th row, namely, pixel 0A~D are written into the register array after shifting down, so as the following three rows of the 0th block.
ii. At cycle 3, all pixels of the 0th block are stored in the register array.
iii. At cycle 4, the 0th row of 1th block, namely, pixel 0A~D are written into the register array after shifting right, so as the following three rows of the 1th block.
At the same time, the 0th column of the 0th block, namely, pixel 0~3A are read from the register array, so as the following three columns of the 0th block
iv. At cycle 7, all pixels of the 1th block are stored in the register array and all pixels of the 0th block are transposed.
v. At cycle 8, the 0th row of 2th block, namely, pixel *0A~D* are written into the register array after shifting down, so as the following three rows of the 2th block.
At the same time, the 0th column of the 1th block, namely, pixel 0~3A are read from the register array, so as the following three columns of the 1th block
vi. At cycle 11, all pixels of the 2th block are stored in the register array and all pixels of the 1th block are transposed.

The hardware cost of this reuse is merely nothing, because the shift-down and shift-right logics are already supported by the reference register array as shown in Fig. 8 (b).

However, extra time cost will be introduced because only after the transpose operation is done, the reference register array could be used to do the search. A detailed time cost is calculated as follows.

i. If the size of search window is 128×64, the total reference pixels would be $(128+64)\times(64+64) = 24576$ pixels. After the 4×4-block-based compression, it would be $24576/4 = 6144$ pixels. (Although the data compression is involved, the transpose process still works by applying some tricks).
ii. By taking advantages of the "duplex communication", the time cost of filling the vertical SRAM would be $6144/32 + 32 = 224$ cycles.
iii. In addition, 2/3 of the search windows between the horizontally neighbored LCUs could be reused. By taking advantages of this, the time cost could be reduced to $6144/3/32 + 32 = 96$ cycles.

Table 2 Hardware cost and B-D rate loss vs truncated bits.

Truncated Bits	Hardware Cost (SAD tree)	B-D Rate Loss of BasketballDrive	B-D Rate Loss of Average All
0	210.1 Kgate	/	/
1	166.4 Kgate	-0.10%	-0.08%
2	145.4 Kgate	0.13%	-0.07%
3	124.3 Kgate	-0.01%	0.01%
4	102.7 Kgate	0.51%	0.29%

Anchor of the proposed solution is HM-16.9 with the default setting, encoder_lowdelay_P_main.cfg, but only one reference frame is allowed for each frame

4.4 Pixel Bit Truncation

In our previous implementation [17], a 4×4 block based compression and decompression technology is adopted to reduce the size of reference SRAMs. Due to this technology, the bit depth of pixels is truncated to 7, which, of course, has been proved to introduce no significant loss.

In the optimized implementation, the truncation of bit depth is further evaluated in respect of the B-D rate loss and the hardware cost reduction. Detailed results are shown in Table 2. According to those results, B-D rate is not sensitive when the truncation bit is 3 and it saves 40.8% of the gate count of the SAD tree, which is an optimal choice from this paper's point of view.

5. Comparison

5.1 B-D Rate Performance

To test the proposed engine, three feasible search schemes based on the micro-codes are designed and evaluated under HM 16.9. The detailed search steps of each schemes are listed in Table 4, while the corresponding B-D rate performances are listed in Table 3.

According to Table 4,

i. Scheme A is the simplest one. It just searches a diamond shape with a width of 16 pixels and height of 16 pixels (W16 H16), and no feedback is taken.
ii. Scheme B is more complicated. It involves a "coarse" search (step 0) and a "refined" search (step 1–4). In the coarse search, it searches a rectangle shape (W48 H32). Although a 1/4 down sampling is adopted, this search range (W48 H32) is much larger than that of scheme A (W16 H16). In the refined search, it starts from the feedback points and searches a rectangle shape (W11 H11).
iii. Scheme C is the most complicated one. In comparison to scheme B, it keeps the refined search (step 4–7) unchanged but enhances the coarse search (step 0–3).

Table 3 Comparison in B-D rate increase.

Class	Sequence	Ye [14]	Medhat [15]	Jou [16]	Fan [17]	Scheme A	Scheme B	Scheme C	Scheme C*
A	Traffic	/	/	/	-0.8%	-0.2%	-0.4%	-0.4%	-0.5%
	PeopleOnStreet	/	/	/	-1.1%	-0.7%	-0.4%	-0.4%	-0.5%
	Nebuta	/	/	/	-0.3%	-0.1%	-0.1%	0.0%	0.0%
	SteamLocomotive	/	/	/	0.3%	0.0%	0.1%	0.0%	0.0%
B	Kimono	/	/	/	-0.3%	-0.1%	0.1%	0.1%	0.0%
	ParkScene	/	/	/	-0.7%	-0.1%	0.0%	0.0%	0.0%
	Cactus	/	/	/	-0.7%	0.9%	0.2%	0.0%	-0.2%
	BasketballDrive	/	/	/	0.3%	12.1%	0.9%	-0.5%	-0.7%
	BQTerrace	/	/	/	-1.6%	-0.4%	-0.1%	-0.4%	-0.7%
C	BasketballDrill	/	/	/	-0.5%	2.1%	-0.5%	-0.6%	-0.7%
	BQMall	/	/	/	-1.3%	-0.6%	-0.4%	-0.1%	-0.1%
	PartyScene	/	/	/	-0.1%	-0.5%	-0.5%	-0.4%	-0.5%
	RaceHorsesC	/	/	/	0.2%	0.6%	0.5%	0.5%	0.7%
D	BasketballPass	/	/	/	-0.3%	0.1%	0.3%	-0.3%	-0.1%
	BQSquare	/	/	/	-1.5%	-0.4%	-0.3%	-0.2%	-0.4%
	BlowingBubbles	/	/	/	-0.5%	-1.0%	-0.4%	-0.4%	-0.9%
	RaceHorses	/	/	/	-1.2%	-0.6%	-0.1%	-0.4%	-0.6%
E	FourPeople	/	/	/	-0.2%	-0.2%	-0.5%	-0.1%	-0.4%
	Johnny	/	/	/	-1.2%	-0.2%	-0.3%	0.4%	-0.2%
	KristenAndSara	/	/	/	-1.1%	0.3%	0.3%	0.3%	-0.1%
A	Average A	/	/	/	-0.4701%	-0.2531%	-0.1750%	-0.2094%	-0.2667%
All	Average All	1.9%	0.92%	4.04%	-0.4983%	0.5494%	-0.0719%	-0.1356%	-0.2943%

Anchor of the proposed solution is HM-16.9 with the default setting, encoder_lowdelay_P_main.cfg, but only one reference frame is allowed for each frame (20 frames tested for each sequence); Anchor of Ye [14] is TZ search in HM-11.0; Anchor of Medhat [15] is full search in x265; Anchor of Jou [16] is EPZS search in HM-5.0; Anchor of Fan [17] is TZ search in HM-15.0;

Table 4 Searching steps of three proposed schemes.

Scheme	Step	Starting Point	Searching Shape	Down Sampling
A	0	(0, 0)	W16 H16 diamond	No (QLCU0)
	1			No (QLCU1)
	2			No (QLCU2)
	3			No (QLCU3)
B	0	(0, 0)	W48 H32 rectangle	Yes (1/4)
	1	best MV of QLCU	W11 H11 rectangle	No (QLCU0)
	2			No (QLCU1)
	3			No (QLCU2)
	4			No (QLCU3)
C & C*	0	(0, 0)	W48 H32 rectangle	Yes (1/4)
	1	(0, 1)		
	2	(1, 0)		
	3	(1, 1)		
	4	best MV of QLCU	W11 H11 rectangle	No (QLCU0)
	5			No (QLCU1)
	6			No (QLCU2)
	7			No (QLCU3)

In column "Searching Shape", "W" stands for width, "H" stands for the height, which share the same meaning as in Fig. 4.

Due to the 1/4 down sampling, a starting point of $(0, 0)$ only covers points in even lines and even rows Considering of this, scheme C searches starting points $(0, 0)$, $(0, 1)$, $(1, 0)$ and $(1, 1)$ to cover all points.

iv. Scheme C* uses the same steps as scheme C, but no bit truncation is applied.

According to Tables 3 and 4, it can be concluded that scheme A is quite simple, which may be sufficient for small-resolution or small-motion videos. However, for sequences like BasketballDrive and BasketballDrill which have large resolutions and large motions, it would introduce a great B-D rate loss (12.1% and 2.1%) due to the small search range it adopts. As for scheme B, it makes up the deficiency of scheme A by involving a large-range coarse search. As a result, the B-D rate loss of BasketballDrive and "Average All" is reduced to 0.9% and −0.0719%. Then scheme C further reduces them to −0.5% and −0.1356% by enhancing the coarse search. Of course, scheme C* is the best one since no bit truncation is involved.

Table 5 Re-descripted steps of Zhou et al.'s work [9].

Algo.	Step	Starting Point	Searching Shape	Down Sampling
Zhou [9]	0	(0, 0)	W208 H104 diamond	Yes (1/16)
	1	best MV of LCU	W3 H2 rectangle	No (QLCU0)
	2			No (QLCU1)
	3			No (QLCU2)
	4			No (QLCU3)
	5	best MV of QLCU	W3 H2 rectangle	No (QLCU0)
	6			No (QLCU1)
	7			No (QLCU2)
	8			No (QLCU3)

In column "Searching Shape", "W" stands for width, "H" stands for the height, which share the same meaning as in Fig. 4.

As a matter of fact, both of the algorithms proposed by Zhou *et al.* [9] and Jiang *et al.* [11] could be implemented on the proposed engine. As an example, the algorithm proposed by Zhou *et al.* [9] is re-descripted by the steps listed in Table 5. It should be pointed out Zhou *et al.*'s work [9] also sends MVPs to the refined step. This may work because Zhou *et al.*'s work [9] is targeted for H.264 standard, in which, the processing unit (macro-block) is as small as 16×16. In other words, only one 16×16, two 16×8, two 8×16, and four 8×8 blocks need to be evaluated in the

Table 7 Overall comparison.

Feature	Sinangil [13] JSTSP'13	Ye [14] VCIPC'14	Medhat [15] ICECS'14	Jou [16] TCSVT'15	Fan [17] TCSVT'18	this work
technology	65 nm	FPGA	FPGA	65 nm	65 nm	65 nm
search method	/	PCTS	FCSA	PEPZS	/	micro-code-based scheme A/B/C
cycle cost (cyc/LCU)	1628	3255	8952	2197	8228	2146/3516/8004
working frequency	200 MHz	200 MHz	550 MHz	270 MHz	500 MHz	500 MHz
supported resolution	4K×2K@60fps	4K×2K@30fps	4K×2K@30fps	4K×2K@60fps	4K×2K@30fps	4K×2K@139/85/37fps
supported block size	16×16 to 64×64	/	8×8 to 32×32	all in HEVC	all in HEVC	all in HEVC
maximum search range	[-64,64]	/	[-16,16]	[-64,64]	[-64,64]	[-64,64] H [-32,32] V
size of reference SRAM	36 KB*	/	9 KB*	36 KB*	18 KB	6.75 KB
size of SAD tree	/	32×32	32×32	16×16	32×32	32×32
gate count of normalized SAD tree	/	/	/	27 K	37 K **	30 K **
B-D rate increase	**12%**	**1.9%**	**0.92%**	**4.04%**	**-0.50%**	**0.55/-0.07/-0.14%**

Anchor of the proposed solution is HM-16.9 with default setting of encoder_lowdelay_P_main.cfg, but only one reference frame is allowed for each frame; Anchor of Ye [14] is TZ search in HM-11.0; Anchor of Medhat [15] is full search in x265; Anchor of Jou [16] is EPZS search in HM-5.0; Anchor of Fan [17] is TZ search in HM-15.0; * those are estimated data, estimated by (search range + 64)^2; ** one 16×16 version is implemented to help the comparison

Table 6 Comparisons in area cost.

Modules	Before Opt	After Opt.	Reduction Rate (%)
H-V reference SRAM	18 KB	6.75 KB	62.5%
transpose memory	1 KB	removed	100.0%
code memory	\	352 b	\
SAD tree	166.4 Kgate	124.3 Kgate	25.3%
addressing logic	3.3 Kgate	2.8 Kgate	15.2%
updating logic	53.0 Kgate	17.9 Kgate	66.2%
reference register array	56.5 Kgate	40.4 Kgate	28.5%
current register array	56.5 Kgate	40.4 Kgate	28.5%
overall	19 KB / 335.7 Kgate	6.79 KB / 225.7 Kgate	64.2% / 32.8%

target frequency is 500MHz;

refined step. But as mentioned in Sect. 1, this is not a feasible way for HEVC standard because of the vast combination of PU sizes and their partitions. As a result, MVPs are replaced by the best MVs of QLCUs in step 5–8. By executing these steps under HM-16.9, the B-D rate loss of "BasketballDrive" and "Average All" and is 2.88% and 3.09% respectively. The cycle cost is $576+(32+(208\times2-1)\times(104\times2-1)/16)+(32+(3\times2-1)\times(2\times2-1))\times8=6353$, where the first item 576 is the cycle cost to do transpose, the second item is the cycle cost of step 0 and the third item is the cycle cost of step 1–8. (Please refer to Sect. 5.2 for the calculation method.) According to this result, Zhou *et al.*'s work [9] is not so appropriate for HEVC standard, which may largely come from the high down-sampling rate it adopts.

5.2 Throughput

The cycle cost of scheme A could be estimated by $96+(32+(16\times2-1)^2/2)\times4=2146$, where

i. 96 is the cycle cost of transposition.
ii. 32 is cycle cost to load original pixels and the corresponding reference pixels of the first search point.
iii. $(16\times2-1)^2/2$ is the total search points in the "W16 H16 Diamond" shape
iv. ×4 is because the same step is executed on 4 QLCUs.

This throughput guarantees a real-time process of 4K × 2K @ 139 fps videos under a working frequency of 500MHz.

In a similar way, the cycle cost of scheme B could be estimated by $96+(32+(48\times2-1)\times(32\times2-1)/4)+(32+(11\times2-1)^2)\times4=3516$, corresponding to 4K × 2K @ 85 fps videos under 500MHz; the cycle cost of scheme C could be estimated by $96+(32+(48\times2-1)\times(32\times2-1)/4\times4)+(32+(11\times2-1)^2)\times4=8004$, corresponding to 4K × 2K @ 37 fps videos under 500MHz.

5.3 Area Cost and Power Consumption

As shown in Table 6, the area cost is reduced a lot compared with our previous implementation [17]. Most of the modules take benefits from the truncation of pixel bits, while the H-V reference SRAM is also reduced due to the adoption of a rectangle search window, the updating logic is also reduced due to the removal of extra directions. Of course, the transpose memory could be removed thanks to the reuse of reference register array.

5.4 Overall Comparison

The overall comparison is given in Table 7.

In terms of B-D rate increase, the proposed search scheme B and C(*) achieves a negative B-D rate loss. Although it is not as good as that of Fan et al.'s design [17], it provides much more flexibility in starting points, search shapes and down-sample rates.

In terms of throughput, the proposed search scheme A costs only 2146 cycles, which is similar to Jou et al.'s design [16]. However, the corresponding B-D rate increase of the proposed design is 0.55%, while that of Jou et al.'s [16] is 4.04%.

In terms of area cost, the SAD tree after normalization is about the same size with Jou et al.'s design [16] while the reference SRAM achieves the smallest area.

In terms of flexibility, the proposed implementation could be programmed with micro-codes to execute search schemes of different complexity.

6. Conclusion

The integer motion estimation (IME) is one of bottlenecks in HEVC encoding due to the complex partitions of the inter prediction units (PU) and the large search window commonly adopted. Many algorithms have been proposed to address this issue and usually put emphasis on a large search window and great computation amount. However, the necessary coding efforts is closely related to the scene. To be more specific, for relatively static videos, a small search window along with a simple search scheme should be adopted to reduce the time cost and power consumption. In view of this, a micro-code-based IME engine is proposed in this paper, which could be applied with search schemes of different complexity. To test the performance, three different search schemes based on this engine are designed and evaluated under HM 16.9, achieving a B-D rate increase of 0.55/−0.07/−0.14%. Compared with our previous design [17], the hardware implementation is optimized to reduce 64.2% of the SRAMs bits and 32.8% of the logic gate count, which supports 4K × 2K @139/85/37fps videos @500MHz.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 61674041, in part by Alibaba Innovative Research (AIR) Program, in part by IBM Faculty Award, in part by the Innovation Program of Shanghai Municipal Education Commission, in part by the pioneering project of academy for engineering and technology and Fudan-CIOMP joint fund.

References

[1] L. Yang, K. Yu, J. Li, and S. Li, "An effective variable block-size early termination algorithm for H.264 video coding," IEEE Trans. Circuits Syst. Video Technol., vol.15, no.6, pp.784–788, June 2005.
[2] H.-M. Yoo and J.-W. Suh, "Fast coding unit decision algorithm based on inter and intra prediction unit termination for HEVC," Proc. IEEE ICCE, pp.300–301, Jan. 2013,
[3] L. Shen, Z. Liu, X. Zhang, W. Zhao, and Z. Zhang, "An effective CU size decision method for HEVC encoders," IEEE Trans. Multimed., vol.15, no.2, pp.465–470, Feb. 2013.
[4] G. Sanchez, M. Porto, and L. Agostini, "A hardware friedly motion estimation algorithm for the emergent HEVC standard and its low power hardware design," 2013 IEEE International Conference on Image Processing, pp.1991–1994, Melbourne, VIC, 2013.
[5] L. Hu, J. Gu, G. He, and W. He, "A hardware-friendly hierarchical HEVC motion estimation algorithm for UHD applications," 2017 IEEE International Symposium on Circuits and Systems (ISCAS), pp.1–4, Baltimore, MD, 2017.
[6] X. Bao, D. Zhou, P. Liu, and S. Goto, "An advanced hierarchical motion estimation scheme with lossless frame recompression and early-level termination for beyond high-definition video coding," IEEE Trans. Multimed., vol.14, no.2, pp.237–249, April 2012.
[7] C. Tsutake and T. Yoshida, "Block-matching-based implementation of affine motion estimation for HEVC," IEICE Trans. Inf. & Syst., vol.E101-D, no.4, pp.1151–1158, April 2018.
[8] S. Lee, M.-C. Hong, and J.-K. Wee, "Low-hardware-cost motion estimation with large search range for VLSI multimedia processors," IEICE Trans. Inf. & Syst., vol.E88-D, no.9, pp.2177–2182, Sept. 2005.
[9] D. Zhou, J. Zhou, G. He, and S. Goto, "A 1.59 Gpixel/s motion estimation processor with −211 to +211 search range for UHDTV video encoder," IEEE J. Solid-State Circuits, vol.49, no.4, pp.827–837, April 2014.
[10] L.-R. Dung and M.-C. Lin, "Wide-range motion estimation architecture with dual search windows or high resolution video coding," IEICE Trans. Fundamentals, vol.E91-A, no.12, pp.3638–3650, Dec. 2008.
[11] Q. Jiang, L. Huang, Y. Fan, and X. Zeng, "Quarter LCU based integer motion estimation algorithm for HEVC," 2016 IEEE International Conference on Image Processing (ICIP), pp.2018–2021, Phoenix, AZ, 2016.
[12] Q. Shang, Y. Fan, W. Shen, S. Shen, and X. Zeng, "Single-port SRAM-based transpose memory with diagonal data mapping for large size 2-D DCT/IDCT," IEEE Trans. Very Large Scale Integration (VLSI) Systems, vol.22, no.11, pp.2422–2426, Nov. 2014.
[13] M.E. Sinangil, V. Sze, M. Zhou, and A.P. Chandrakasan, "Cost and coding efficient motion estimation design considerations for high efficiency video coding (HEVC) standard," IEEE J. Sel. Topics Signal Process., vol.7, no.6, pp.1017–1028, Dec. 2013.
[14] X. Ye, D. Ding, and L. Yu, "A hardware-oriented IME algorithm and its implementation for HEVC," 2014 IEEE Visual Communications and Image Processing Conference, pp.205–208, Dec. 2014.
[15] A. Medhat, A. Shalaby, M.S. Sayed, M. Elsabrouty, and F. Mehdipour, "Fast center search algorithm with hardware implementation for motion estimation in HEVC encoder," 2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp.155–158, Dec. 2014.
[16] S.-Y. Jou, S.-J. Chang, and T.-S. Chang, "Fast motion estimation algorithm and design for real time QFHD high efficiency video coding," IEEE Trans. Circuits Syst. Video Technol., vol.25, no.9, pp.1533–1544, Sept. 2015.
[17] Y. Fan, L. Huang, B. Hao, and X. Zeng, "A hardware-oriented IME algorithm for HEVC and its hardware implementation," IEEE Trans. Circuits Syst. Video Technol., vol.28, no.8, pp.2048–2057, Aug. 2018.

Leilei Huang received the B.S. and M.S. degree in Microelectronics and Solid Electronics from Fudan University, Shanghai, China, in 2014 and 2017. He is currently pursuing toward the Ph.D. degree in Microelectronics and Solid Electronics from Fudan University. His research interests include VLSI design, algorithms and corresponding VLSI architectures for multimedia signal processing.

Yibo Fan received the B.E. degree in electronics and engineering from Zhejiang University, China in 2003, M.S. degree in microelectronics from Fudan University, China in 2006, and Ph.D. degree in engineering from Waseda University, Japan in 2009. From 2009 to 2010, he worked as an assistant professor in Shanghai Jiaotong University, and currently, he is the assistant professor in the college of microelectronics of Fudan University. His research interesting includes image processing, video coding and associated VLSI architecture.

Chenhao Gu received the B.S. degree in Microelectronics Science and Engineering from Fudan University, Shanghai, China, in 2017. Currently he is pursuing the M.S. degree in Microelectronics and Solid Electronics from Fudan University. His research interests include algorithms and VLSI architectures for video codecs.

Xiaoyang Zeng received the B.S. degree from Xiangtan University, Xiangtan, China, in 1992, and the Ph.D. degree from Changchun Institute of Optics, Fine Mechanics, and Physics, Chinese Academy of Sciences, Changchun, China, in 2001. From 2001 to 2003, he was a Postdoctoral Researcher with Fudan University, Shanghai, China. Then, he joined the State Key Lab of ASIC and System, Fudan University, as an Associate Professor, where he is currently a Full Professor and the Director. His research interests include information security chip design, system-on-chip platforms, and VLSI implementation of digital signal processing and communication system.