Ahn *et al. EURASIP Journal on Image and Video Processing* 2014, **2014**:16
http://jivp.eurasipjournals.com/content/2014/1/16

EURASIP Journal on Image and Video Processing
a SpringerOpen Journal

RESEARCH Open Access

Implementation of fast HEVC encoder based on SIMD and data-level parallelism

Yong-Jo Ahn[1], Tae-Jin Hwang[1], Dong-Gyu Sim[1*] and Woo-Jin Han[2]

Abstract

This paper presents several optimization algorithms for a High Efficiency Video Coding (HEVC) encoder based on single instruction multiple data (SIMD) operations and data-level parallelism. Based on the analysis of the computational complexity of HEVC encoder, we found that interpolation filter, cost function, and transform take around 68% of the total computation, on average. In this paper, several software optimization techniques, including frame-level interpolation filter and SIMD implementation for those computationally intensive parts, are presented for a fast HEVC encoder. In addition, we propose a slice-level parallelization and its load-balancing algorithm on multi-core platforms from the estimated computational load of each slice during the encoding process. The encoding speed of the proposed parallelized HEVC encoder is accelerated by approximately ten times compared to the HEVC reference model (HM) software, with minimal loss of coding efficiency.

Keywords: HEVC; HEVC encoder; SIMD implementation; Slice-level parallelism; Load balancing

1 Introduction

Along with the development of multimedia and hardware technologies, the demand for high-resolution video services with better quality has been increasing. These days, the demand for ultrahigh definition (UHD) video services is emerging, and its resolution is higher than that of full high definition (FHD), by a factor of 4 or more. Based on the market demands, ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG) have organized Joint Collaborative Team on Video Coding (JCT-VC) and standardized High Efficiency Video Coding (HEVC), whose target coding efficiency was twice better than that of H.264/AVC [1]. In the near future, HEVC is expected to be employed for many video applications, such as video broadcasting and video communications.

Historically, MPEG-x and H.26x video compression standards employ the macro-block (MB) as one basic processing unit [2], and its size is 16×16. However, HEVC supports larger sizes of the basic processing unit, called coding tree unit (CTU), from 8×8 to 64×64. A CTU is split into multiple coding units (CU), in a quad-tree fashion [3]. Along with the CU, the prediction unit (PU) and transform unit (TU) are defined, and their sizes and shapes are more diverse than the prior standard technologies [4,5]. On top of them, many advanced coding tools that improve prediction, transform, and loop filtering are employed to double the compression performance compared with H.264/AVC. However, the computation requirement of HEVC is known to be significantly higher than that of H.264/AVC because HEVC has more prediction modes, larger block size, longer interpolation filter, and so forth.

Typically, a huge number of rate-distortion (RD) cost computations are required to find the best mode from 64×64 to 8×8 block sizes in the encoder side for HEVC. With respect to applications, HEVC would be employed for ultrahigh-resolution video services. For such cases, fast video coders are required to process more data with a given processing power. Thus, parallelization techniques would be crucial, with multiple low-power processors or platforms. The single instruction multiple data (SIMD) implementation of the most time-consuming modules on HM 6.2 encoders was proposed [6]. This work implemented the cost functions, transformation, and interpolation filter with SIMD, and it reported that the average time saving obtained is approximately 50% to 80%, depending on the modules. Wavefront parallel

* Correspondence: dgsim@kw.ac.kr
[1]Department of Computer Engineering, Kwangwoon University, Wolgye-dong, Nowon-gu, Seoul 447-1, South Korea
Full list of author information is available at the end of the article

processing (WPP) for HEVC encoders and decoders was introduced [7]. For the decoder case, they achieved parallel speed-up by a factor of 3. The acceleration factor of the wavefront parallelism is in general saturated into 2 or 3 due to data communication overhead, epilog, and prolog parts. There are no works that incorporate all the parallel algorithms, with maximum harmonization for fast HEVC encoders. In this paper, we focus on load-balanced slice parallelization, with optimization implementation of HEVC. This paper presents several optimization techniques using SIMD operations for the RD cost computations and transforms for variable block sizes. In addition, motion estimation is also efficiently implemented with a frame-based processing to reduce the number of redundant filtering. For data-level parallelization, this paper demonstrates how to allocate encoding jobs to all the available cores through the use of complexity estimation. As a result, it is possible to achieve load-balanced slice parallelism in HEVC encoders to significantly reduce the average encoding time. With all the proposed techniques, the optimized HEVC encoder achieves a 90.1% average time saving within 3.0% Bjontegaard distortion (BD) rate increases compared to HM 9.0 reference software.

The paper is organized as follows. Section 2 presents a complexity analysis of HEVC encoder, and Section 3 introduces basic data-level parallelisms for video encoders. In Section 4, the SIMD optimization for cost functions and transform, as well as frame-level implementation of interpolation filter, is explained in detail. A slice-level parallelization technique with a load-balancing property is proposed in Section 5. Section 6 shows the performance and numerical analysis of the proposed techniques. Finally, Section 7 concludes the work, with further research topics.

2 HEVC and its complexity analysis

Figure 1 shows a block diagram of HM encoder [8]. The HEVC encoder consists of prediction, transformation, loop filters, and entropy coder, which are the same cores as the prior hybrid video coders. However, HEVC employs more diverse block sizes and types, named CU, PU, and TU. The CU sizes range from 8×8 to 64×64, when the CTU is set to 64×64. The CU structure is partitioned in a quad-tree fashion, and each CU can have one of several PU types inside it. This allows each CU to be predicted with diverse block sizes and shapes. In addition, advanced motion vector prediction (AMVP) [9] and block merging techniques [10,11] are adopted to effectively represent the estimated motion vectors. For residual coding, the transform block sizes and shapes are determined based on a rate-distortion optimization (RDO). The quad-tree transform is used for each CU, and it is independent of the PU type. Transform coefficients are quantized with a scalar quantizer. To improve coding efficiency, the rate-distortion optimized quantization (RDOQ) is often employed, during the quantization process. Finally, the reconstructed blocks are filtered with two-stage loop filters: the de-blocking filter (DBF) and sample adaptive offset (SAO).

As mentioned before, HEVC supports hierarchical block partitioning. Figure 2 shows an example of diverse CU and PU realizations in a slice, when the CTU size is set to 64×64. A slice is divided into multiple CTUs, and each CTU is again partitioned into multiple CUs. The quad-tree structure is effectively represented by

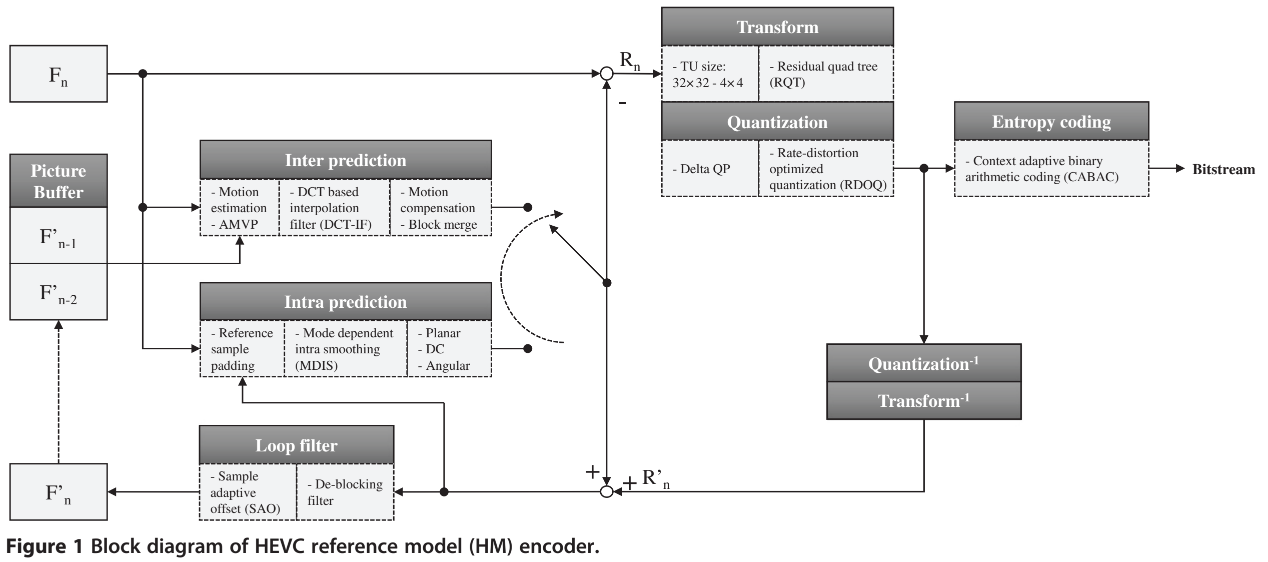


Figure 1 Block diagram of HEVC reference model (HM) encoder.

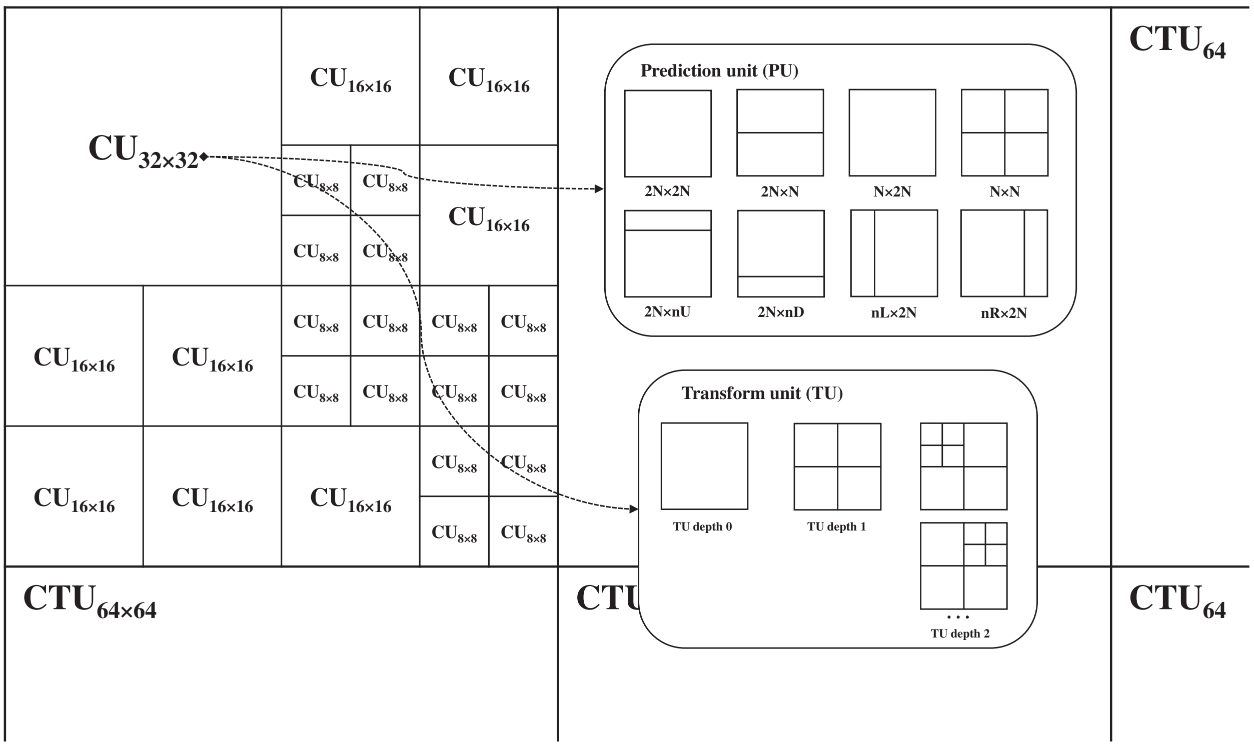


Figure 2 Example of CU, PU, and TU partitions for a CTU in HEVC.

hierarchical flag syntaxes. HEVC has one identical syntax for diverse CU block sizes. The size of a CU block can be derived with quad-tree split flags. In addition, one CU can be coded with one of the several PUs. For a current CU of 2 N × 2 N, the CU can have one of seven PU splitting types: 2 N × 2 N, 2 N × N, N × 2 N, or four asymmetric shapes (2 N × nU, 2 N × nD, nL × 2 N, and nR × 2 N) [12]. The residual signal is transformed in TU, which can be recursively split into multiple level quadrants. HEVC supports diverse TU sizes, from 4 × 4 up to 32 × 32, and the maximum TU size depends on the CU size. However, quadtree-based TU partitioning is conducted independent of PU partitioning.

In this section, the complexity of HEVC encoder is investigated, and critical modules can be identified based on the complexity analysis. In this work, HM 9.0 reference software [13] was used for HEVC encoder analysis. Note that it was used as the base software for our optimization. A HEVC encoder can be mainly modularized into five parts: entropy coding, intra prediction, inter prediction, transform quantization, and loop filter. The cycle analyzer, Intel® VTune™ Amplifier XE 2013 [14] on Intel® Core™ i7-3960 K processor, was employed to measure the number of cycles for each module, in cases of the random access (RA) and low-delay (LD) test configurations, under the common test conditions [15]. Note that class B (1,920 × 1,080) and class C (832 × 480) sequences were used.

Table 1 shows the percentages of computation cycles of all the key modules, for two configurations, RA and LD. We found that the inter prediction takes around 68.4% to 89.1% of the total cycles, whereas the intra prediction takes only 1.2% to 3.3%. For the transform quantization and entropy coding, the percentages of cycles are counted as 10.1% to 20.7% and 0.3% to 6.6%, respectively. Note that quantization is one of the key modules in terms of functionality, but its computational cycle is not significant for measuring it independently. In this paper, the total computational cycles of transform and quantization were measured. As shown in Table 1, the inter prediction, which consists of motion estimation and compensation, is the dominant module

Table 1 Percentages of computational cycles of HM 9.0 encoder

Module	RA (%)	LD (%)	Average (%)
Entropy coding	2.98	2.40	2.69
Intra prediction	2.25	1.95	2.10
Inter prediction	79.03	82.23	80.63
Transform quantization	14.48	12.50	13.49
In-loop filter (de-blocking filter)	0.08	0.08	0.08
In-loop filter (sample adaptive offset)	0.10	0.10	0.10
Others	1.28	0.93	1.10

in computational complexity. For HEVC inter prediction coding, a large number of coding modes for various size blocks are evaluated and coded by computing rate-distortion costs. In addition, the hierarchical block partition is traversed in a recursive fashion in HM.

Table 2 shows percentages of the cycles for intra prediction, inter prediction, and skip modes, depending on the CU sizes. Regardless of the CU sizes, the percentage of inter prediction coding is about 73.7% to 83.4%. For the CU of 8 × 8, the cycle percentage of the intra prediction coding is approximately 2 to 3 times higher than the others, because 4 × 4 and 8 × 8 intra prediction modes are tested for RDO.

Table 3 shows the percentiles of the numbers of cycles for the top four functions. The interpolation filter, sum of absolute differences (SAD), sum of absolute transformed differences (SATD), and discrete cosine transform (DCT)/inverse DCT (IDCT) take approximately 67% to 71% in the total cycles. The interpolation filter is the most complex function of HEVC encoders and is used for motion estimation and motion compensation. SAD and SATD are cost functions to calculate distortions between original and prediction blocks. In particular, SAD and SATD are the metrics for motion estimation of integer-pels and fractional-pels, respectively. In addition, SATD is also used for intra prediction. DCT/IDCT is applied to the residual signal from intra or inter prediction for data compaction in the DCT domain. We can say that optimization of the four functions is inevitable in order to accelerate HEVC encoder.

3 Data-level parallelization of video encoders

Data-level and function-level parallelization approaches are widely used for high-speed video codecs. In particular, function-level parallel processing is frequently used for hard-wired implementations. Note that function-level parallel processing is not easily implemented mainly due to difficulties of load balancing and longer development period. Data-level parallel processing is relatively easy to be employed for video encoders because the data processing flows are the same for all the data. The data-level parallelism for HEVC can be conducted in terms of CU-, slice-, and frame-level ones. In addition, HEVC contains a parallel tool, called tile, which divides a picture into multiple rectangles [16]. In tile partitioning, the number of CTUs adjacent to boundaries of tile partitions is less than that of slices. From this fact, tile partitioning can yield slightly lower coding loss in compression efficiency compared to an implementation with the same number of slices [17].

For parallel implementations, we need to consider several factors, such as throughput and core scalability, as well as coding efficiency. Note that the core scalability means how much we need to change an implementation, depending on an increasing or decreasing number of cores. In addition, the throughput can be improved with parallel processing as compared with the single processing unit. However, many video coding algorithms, in general, have dependencies among neighboring coding units, neighboring frame, earlier-coded syntaxes, and so on. At the same time, we need to consider the coding efficiency degradation from the parallelization. Even though the throughput can be improved with parallel processing, it is not desirable that the coding efficiency is significantly degraded. Regarding the core scalability, it is better to employ a scalable parallelization method that can be easily extended for an increasing number of cores. If not, we are required to change the implementation, depending on the number of cores.

The 2D wavefront algorithm [18] has been used for the parallelization of video coding in CTU level. This coding tool does not impact the coding gain, but there is a limitation in the parallelization factor, even with many cores, due to coding dependence. Frame-level parallelization can be also used for non-reference bidirectional pictures; however, it depends on the encoding of reference structures.

The slice-level parallelism is widely used because we can assume any dependencies among multiple slices. However, we need to realize that the coding gain can be degraded

Table 2 Percentages of computational cycles, depending on CU sizes and modes

Size	Mode	RA (%)	LD (%)	Average (%)	Ratio in each CU size (%)
64 × 64	Intra	2.1	1.0	1.6	5.6
	Inter	19.0	31.9	25.5	82.3
	Skip	3.9	3.4	3.7	12.1
32 × 32	Intra	1.9	0.7	1.3	4.5
	Inter	25.0	27.4	26.2	83.4
	Skip	4.5	3.2	3.9	12.2
16 × 16	Intra	2.3	0.2	1.3	4.4
	Inter	17.0	12.5	14.8	82.9
	Skip	3.2	1.7	2.5	12.7
8 × 8	Intra	2.4	0.4	1.4	13.5
	Inter	8.7	4.9	6.8	73.7
	Skip	1.7	0.6	1.2	12.8

Table 3 Percentages of computational cycles of top four functions

Module	RA (%)	LD (%)	Average (%)
Interpolation filter	35.74	36.00	35.87
SATD	12.99	18.57	15.78
SAD	15.33	13.26	14.30
Transform/inverse transform	3.52	3.08	3.30
Total (%)	67.58	70.91	69.25

with increased number of slices. In this paper, we evaluated bitrate increases in terms of the number of slices in the HM encoder software. Figure 3 shows the bitrate increases in terms of the number of slices, for four sequence classes. In our evaluation, class A (2,560 × 1,600), class B (1,920 × 1,080), class C (832 × 480), and class D (416 × 240) of the HEVC common test sequences [15] were used, under HEVC common test conditions. As shown in Figure 3, we can see the bitrate increases in terms of the number of slices. The bitrate increase becomes less significant as video resolution increases. In general, multiple slices are widely used for larger sequences due to parallel processing and error resilience. Regarding many commercial applications, the slice-level parallel processing is one of the good approaches for such large resolution videos.

As mentioned before, the slice-level parallelism has relatively high coding losses of around 2% to 4% compared to tile-level parallelism and wavefront processing [19]. However, slice-level parallelism has an advantage that the slice partitioning is more flexible and accurate for picture partitioning, by adjusting the number of CTUs, compared to the tile partitioning. Note that the tiles within the same row and column should use the same tile width and height, respectively. Slice-level parallelism of a fine-grained load balancing can yield additional encoding speed-up compared to the tile levels. WPP has the advantage that the loss of parallelization is relatively small compared to other parallelization methods. However, the acceleration factor of WPP is not so high compared to slice- or tile-level parallelism because WPP has prolog and epilog so that parts of the cores are inactivated. It is not easy to utilize all the cores with WPP on average. In our work, slice-level parallelism was chosen for the acceleration of parallelization. In addition, slice partitioning is widely used for the packetizing of bitstreams for error resiliencies, in practical video encoders and services.

There are two main criteria to divide a picture into multiple slices. One is an equal bitrate, and the other is the same number of CTUs for all the slices. The first one cannot be easily employed for parallel encoding because we cannot define the target bit prior to actual encoding. For the second method, we can easily use the same number of CTUs at a time.

4 Optimization for fast HEVC encoder

In this section, two software optimization methods, frame-level processing and SIMD implementation, for three most complex functions at the function-level are presented. The proposed software optimization methods have several advantages to accelerate HEVC encoders without any bitrate increase.

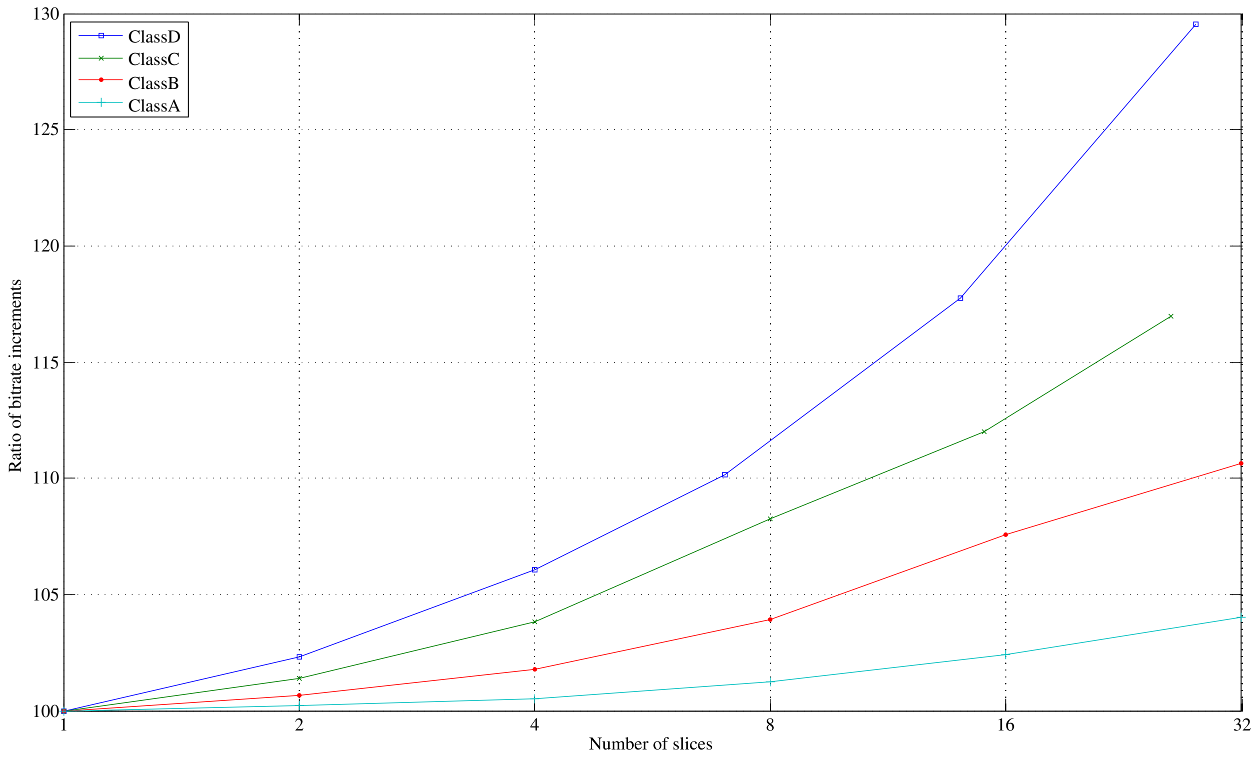


Figure 3 Bitrate increment in terms of the number of slices, for four sequence classes. HEVC common test sequences [15]; class A, 2,560 × 1,600; class B, 1,920 × 1,080; class C, 832 × 480; class D, 416 × 240.

4.1 Frame-level interpolation filter in HEVC encoder

The HEVC DCT-based interpolation filter (DCT-IF), which is used for obtaining fractional sample positions, is the most complex function, especially with motion estimation in encoders. Instead of using 6-tap and bilinear interpolation filters of H.264/AVC, HEVC adopts 8(7)-tap DCT-IF for luminance components, and 4-tap DCT-IF for chrominance components [20]. Furthermore, all of the fractional position samples are derived by increasing the number of filter taps without intermediate rounding operations which can reduce potential rounding errors compared to H.264/AVC. In order to determine the optimal CU size and coding modes, HM encoder uses a recursive scheme for the RD optimization process. In particular, the PU-level interpolation filter causes iterative memory accesses for the same positions redundantly. Excessive memory accesses significantly increase encoding time due to the limit of memory bandwidth. Actually, the DCT-IF occupies approximately 30% to 35% of the total cycles in the HM encoder. We adopt a frame-level interpolation filter to reduce redundant memory accesses. The frame-level interpolation filter avoids redundant execution that occurs in the RD optimization process and enables parallel process with independency among neighboring blocks. However, it requires the additional amount of memory for 15 factional samples per integer sample in an entire frame. In addition, SIMD instructions and multi-thread processes using OpenMP and GPU can be easily used for fast encoding.

4.2 SIMD implementation of cost function and transformation

SAD, SATD, and DCT are the most complex functions in the HEVC encoder, except for DCT-IF. Several cost functions are used to decide the best coding mode and its associated parameters. SAD and SATD are the two main metrics to find integer and quarter-pel motion vectors in the motion estimation process, respectively. SAD takes around 10% to 12% of the total cycles in HEVC encoder, and SATD takes 15% to 16%. These two cost functions are defined by

$$\text{SAD} = \sum_{i,j}^{I,J} ||O(i,j) - P(i,j)|| \tag{1}$$

$$\text{SATD} = \left(\sum_{i,j}^{I,J} ||H(i,j)|| \right) / 2 \tag{2}$$

where i and j are the pixel indices, and their ranges are determined by a block size. $O(i,j)$ and $P(i,j)$ are the original and predicted pixel values, respectively. Note that

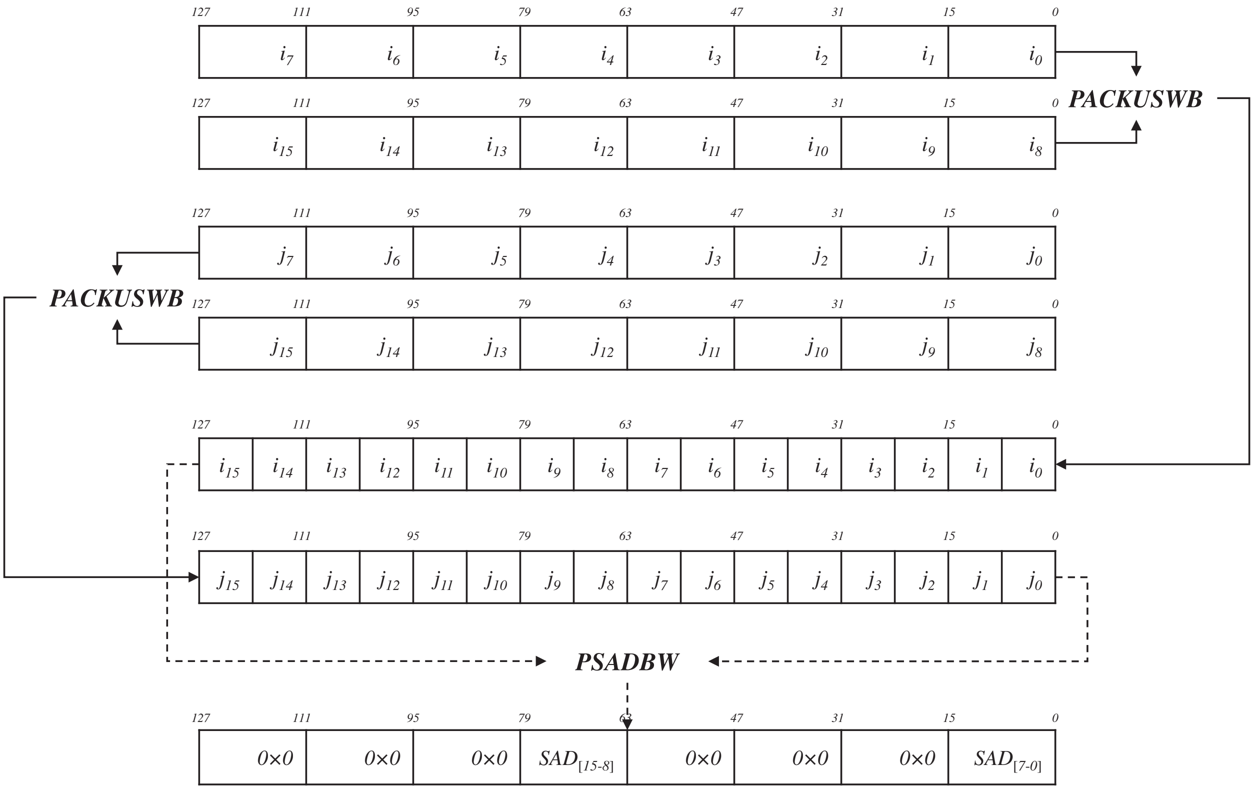

127
111
95
79
63
47
31
15
0
i_7 i_6 i_5 i_4 i_3 i_2 i_1 i_0
i_{15} i_{14} i_{13} i_{12} i_{11} i_{10} i_9 i_8
PACKUSWB
j_7 j_6 j_5 j_4 j_3 j_2 j_1 j_0
j_{15} j_{14} j_{13} j_{12} j_{11} j_{10} j_9 j_8
PACKUSWB
i_{15} … i_0
j_{15} … j_0
PSADBW
0×0
$SAD_{[15\text{-}8]}$
$SAD_{[7\text{-}0]}$

Figure 4 Computation steps of SAD using SIMD instructions.

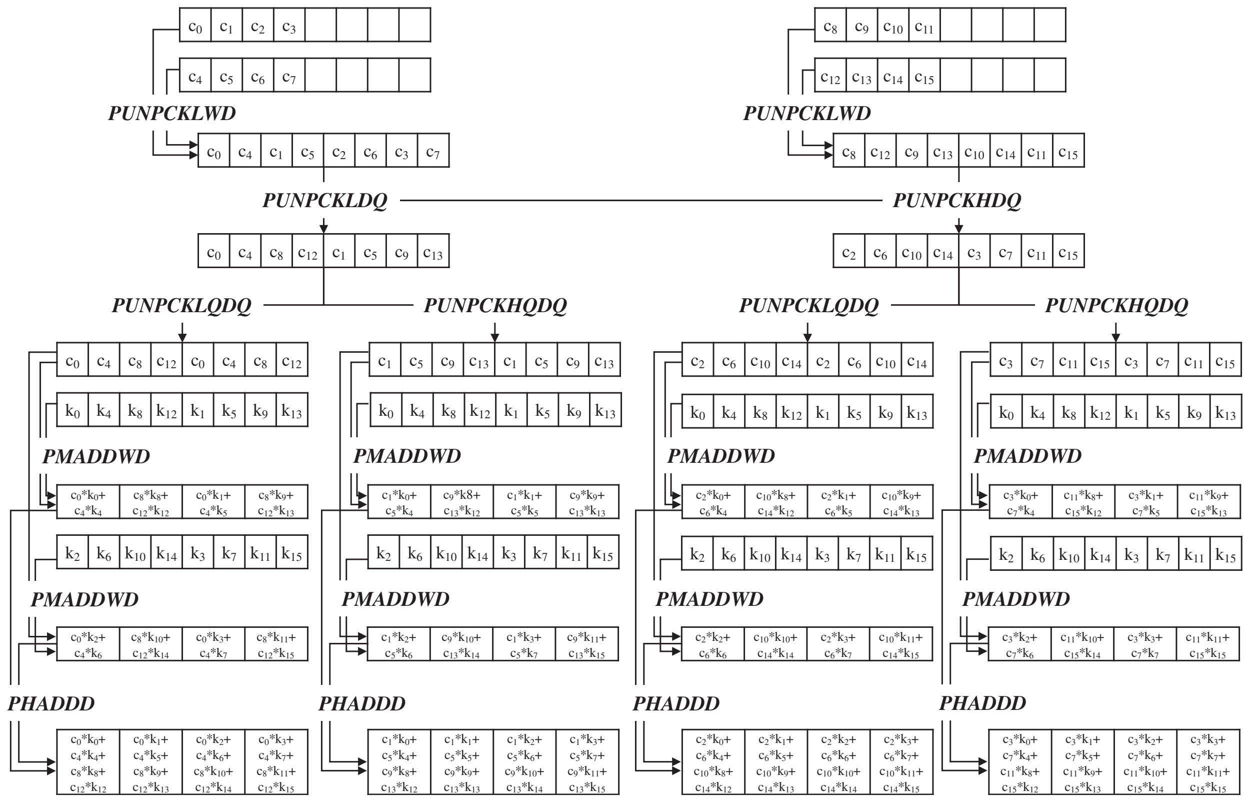


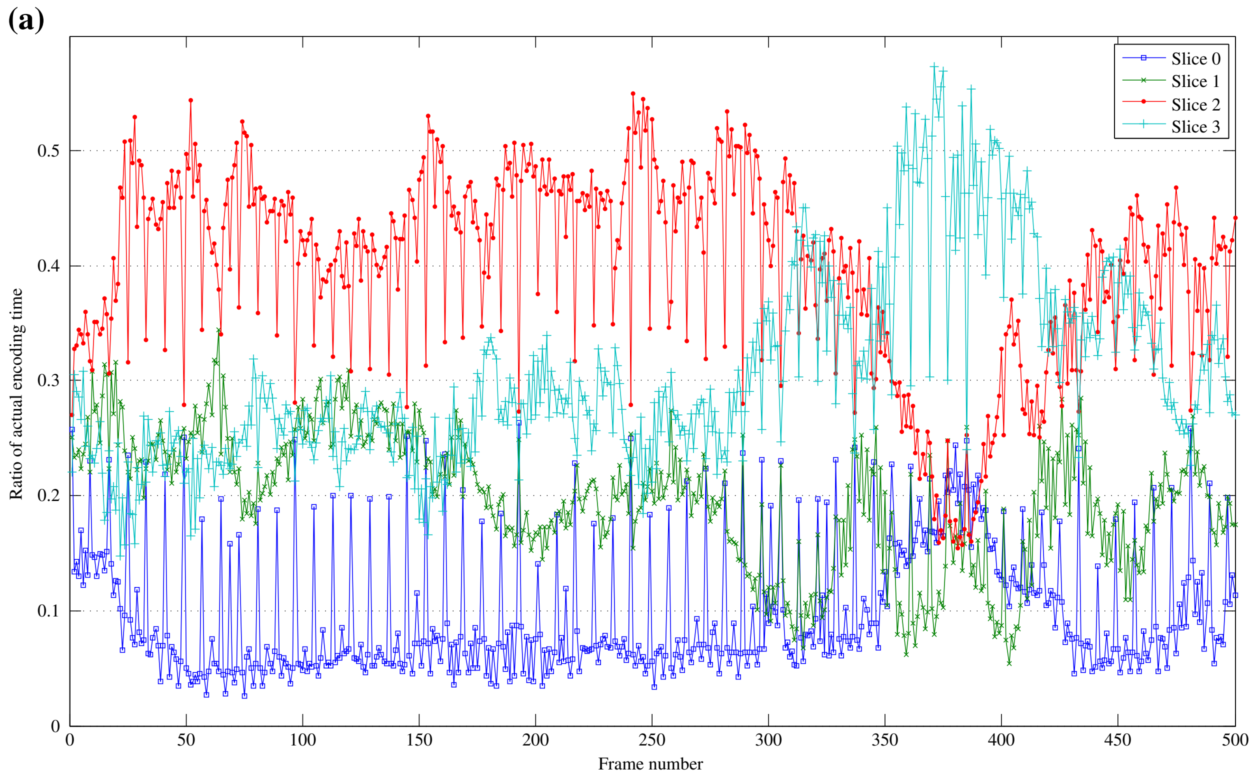
Figure 5 Computation steps of HEVC inverse transform using SIMD instructions.

$H(i,j)$ is the Hadamard transformation of the prediction error, $O(i,j) - P(i,j)$ [8]. Because only addition and subtraction operations are involved for the cost functions, SATD can yield an accurate cost in the transform domain with relatively small complexity compared to DCT. Since both apply the same operations on multiple data, vector instructions are quite useful to reduce the required clock cycles. This work uses SSE2 and SSE3 instructions defined in Intel SIMD architecture, which are widely employed for many DSP processors [21]. In the case of the SAD operation, we employed *PSADBW* (packed sum of absolute differences), *PACKUSWB* (packed with unsigned saturation), and *PADDD* (add packed double word integers) instructions. Sixteen SAD values can be computed by *PSADBW* instruction at once. Figure 4 shows how to compute SAD with SIMD instructions. Data packing is conducted with sixteen 16-bit original pixels (i_x) and sixteen 16-bit reference pixels (j_x) using *PACKUSWB* instruction. For 8-bit internal bit depth, the data packing is conducted to form 16-bit short data. For 10-bit internal bit depth, the data packing process is not required. Sixteen original pixels and reference pixels are packed into two 128-bit registers, and *PSADBW* is performed. The computed SAD from i_0-j_0 to i_7-j_7 is stored in the lower 16 bits, and the SAD from i_8-j_8 to i_{15}-j_{15} is stored at bit position 64 to 79. Acceleration of 4×4 to 64×64 SAD computations can be achieved using the aforementioned instructions based on instruction-level parallelism. The 4×4 and 8×8 SATD operations are implemented using interleaving instructions, such as *PUNPCKLQDQ* (unpack low-order quad-words), *PUNPCKHWD* (unpack high-order words), *PUNPCKLWD* (unpack low-order words), and arithmetic instructions, such as *PADDW* (add packed word integers), *PSUBW* (subtract packed word integers), and *PABSW* (packed absolute value).

Not only the cost function but also the forward transform and inverse transform can be implemented with SIMD instructions. The forward transform and inverse transform in HEVC are implemented by partial butterfly or matrix multiplication. In this paper, the matrix multiplication is chosen due to its simplicity and regularity. Transform and inverse transform are accelerated using interleaving instructions such as *PUNPCKLDQ* (unpack

Table 4 Normalized complexity for variable CU size and mode

CU size	Skip	Inter	Intra
64×64	109	760	52
32×32	42	280	16
16×16	9	71	3
8×8	2	19	1



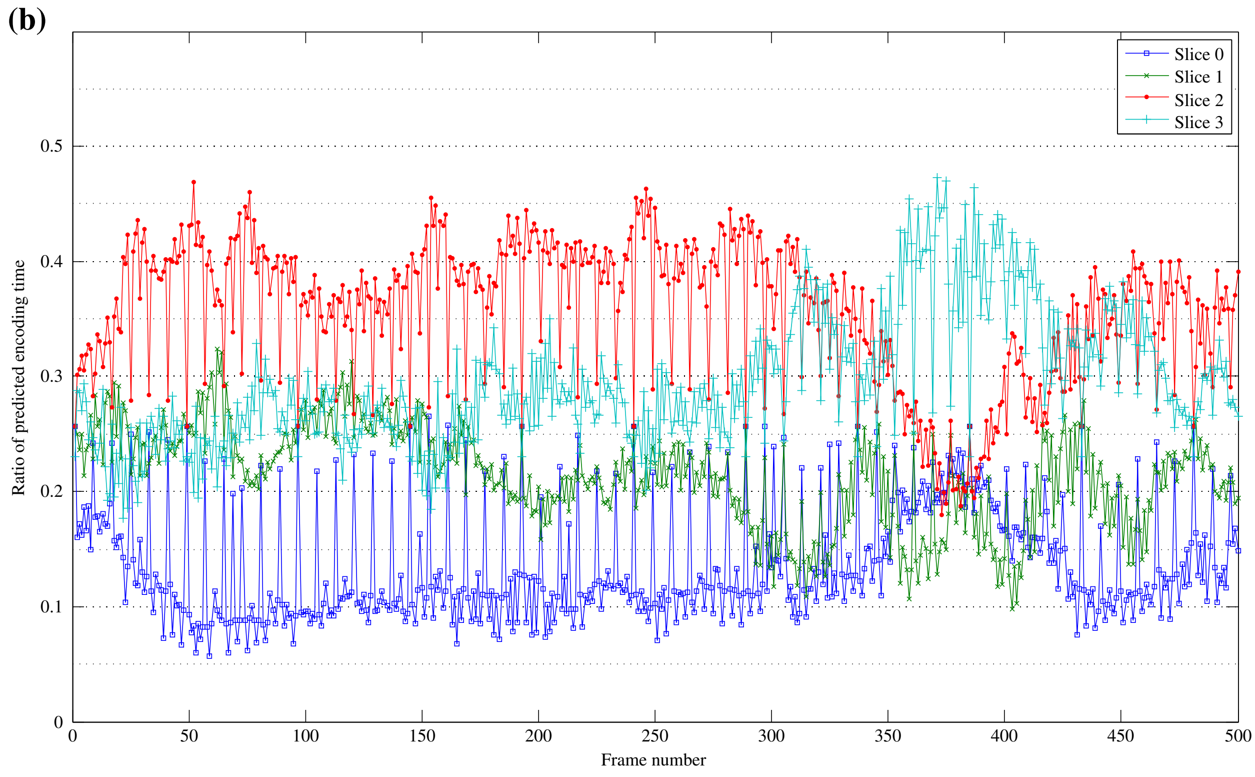


Figure 6 Ratios of actual encoding time and predicted complexity for BasketballDrive sequence. (a) Ratios of actual encoding time for the four slices. **(b)** Ratios of predicted complexity for four slices.

low-order double words), *PUNPCKHDQ* (unpack high-order double words), *PUNPCKLQDQ*, *PUNPCKHQDQ* (unpack high-order quad words), and arithmetic instructions such as *PMADDWD* (packed multiply and add), *PADDD*, and shift instruction such as *PSRAD* (packed shift right arithmetic). For HEVC forward and backward transformation, we need to consider the data range and the center value in computing matrix multiplications, unlike SAD and SATD implementations. Figure 5 shows how to compute the HEVC inverse transform using SIMD instructions. Data packing is conducted with sixteen 16-bit coefficients (c_x) using *PUNPCKLWD* instruction. The 16 coefficients are packed into two 128-bit registers. For reordering coefficients, the packed coefficient signals are repacked using *PUNPCKLQDQ and PUNPCKHQDQ* instructions. Repacked coefficients and the kernel (k_x) of the inverse transform are multiplied for eight 16-bit data in 128-bit registers. Then, the results of multiplications are added into 128-bit registers using *PMADDWD* instruction. Finally, the results of *PMADDWD* are added into the 128-bit destination register to compute inverse-transformed residuals using *PHADD* instruction. Input data for transformation range from −255 to 255. As a result, the data should be represented by at least 9 bits. Data ranges of coefficients of HEVC transform kernels depend on the size of the transform kernels. However, they can be represented in 8 bits for the 32 × 32 kernel because they range from −90 to 90 [22]. For computation of one transform coefficient, the required number of addition and multiplication operations is as many as the size of the transform kernel along the horizontal and vertical directions. A downscale should be employed to keep 16 bits in every operation for each direction. To avoid overflow and underflow, four 32-bit data should be packed into the 128-bit integer register of SSE2. In addition, the transform matrix is transposed in advance to reduce memory read/write operations.

5 Proposed slice-level parallelism with load balance

To reduce the computational load of the RD optimization, early termination and mode competition algorithms have been adopted in HM reference software [23-25]. However, these fast encoding algorithms cause different encoding complexities among different slices. To maximize parallelism of the data-level task partition, an accurate load balance for slice parallelization is required. Several works [26,27] have been conducted to achieve accurate load balance for slice parallelization. In Zhang's algorithm [26], the adaptive data partitioning for MPEG-2 video encoders was proposed by adjusting computational loads based on the complexity of a previously encoded frame of the same picture type. In Jung's algorithm [27], the adaptive slice partition algorithm was proposed to use early-decided coding mode for macro-blocks in H.264/AVC. In the conventional algorithm, a quantitative model was designed to estimate the computational load associated with each candidate MB mode group. However, in order to apply slice-level parallelism to a HEVC encoder, we need to focus on CTU structures, variable block sizes, and coding modes. In this section, a complexity estimation model and adaptive slice partition algorithm to achieve load-balanced slice parallelization are proposed.

5.1 Complexity estimation model

In this section, a load-balancing technique for a slice parallelization is proposed by allocating the proper number of CTUs for each core after estimating the computational load for one slice. For this purpose, this work introduces a model of computational load for CU-level RD optimization process, in terms of diverse coding tools such as CU size, skip mode, AMVP, and many intra prediction directions. Table 4 shows normalized computational complexities by setting the complexity of 8 × 8 intra prediction to 1, as shown in Table 2. The normalized computational complexities for variable CU sizes and modes are computed by

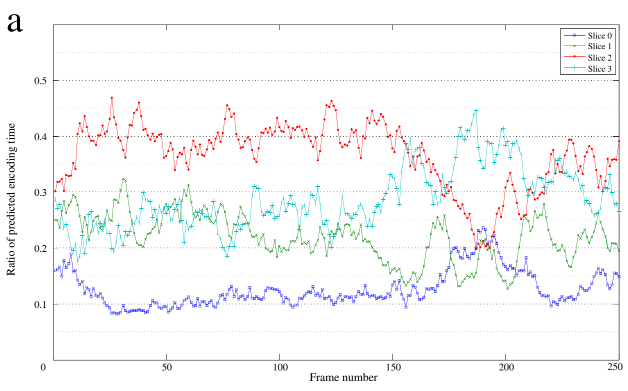
$$R(s,m) = r(s,m) \times 2^{w(\mathrm{CTU})/w(s)} \tag{3}$$

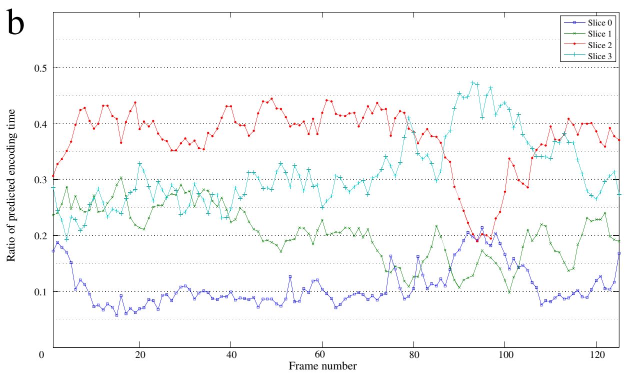
$$\mathrm{CEM}(s,m) = R(s,m) \times \mathrm{NF} \tag{4}$$

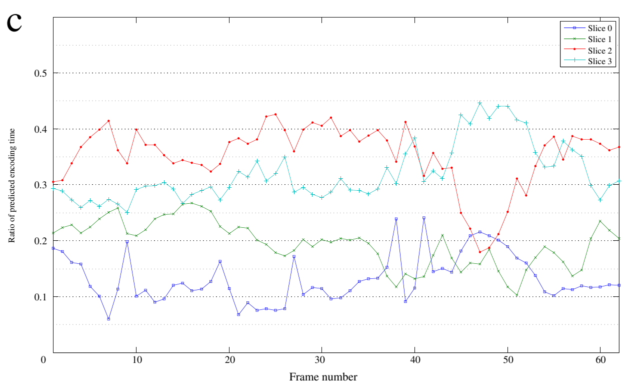
where $R(s,m)$ and $r(s,m)$ represent the complexity per unit and the complexity ratio of each CU size and mode, respectively. $w(s)$ and $\mathrm{CEM}(s,m)$ are the width of CU size and the complexity estimation model in Table 4, respectively. Note that NF is a normalization factor for

Table 5 Pearson product moment correlations of the actual and predicted times

Class	Sequence name	Pearson product moment correlation
Class A (2,560 × 1,600)	Traffic	0.9495
	PeopleOnStreet	0.9083
Class B (1,920 × 1,080)	Kimono	0.9859
	ParkScene	0.9689
	Cactus	0.9382
	BasketballDrive	0.9456
	BQTerrace	0.9093
Class C (832 × 480)	BasketballDrill	0.9568
	BQMall	0.9723
	PartyScene	0.9326
	RaceHorses	0.9484
Average		0.9469







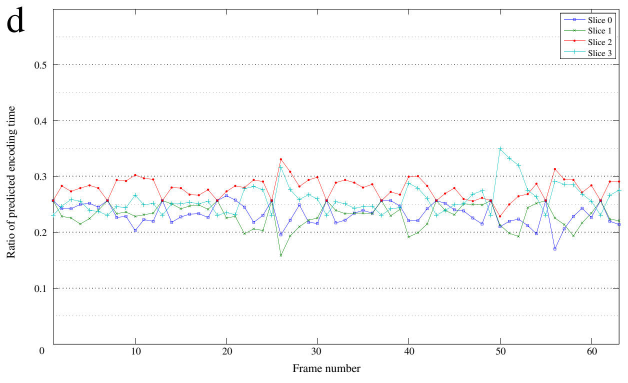
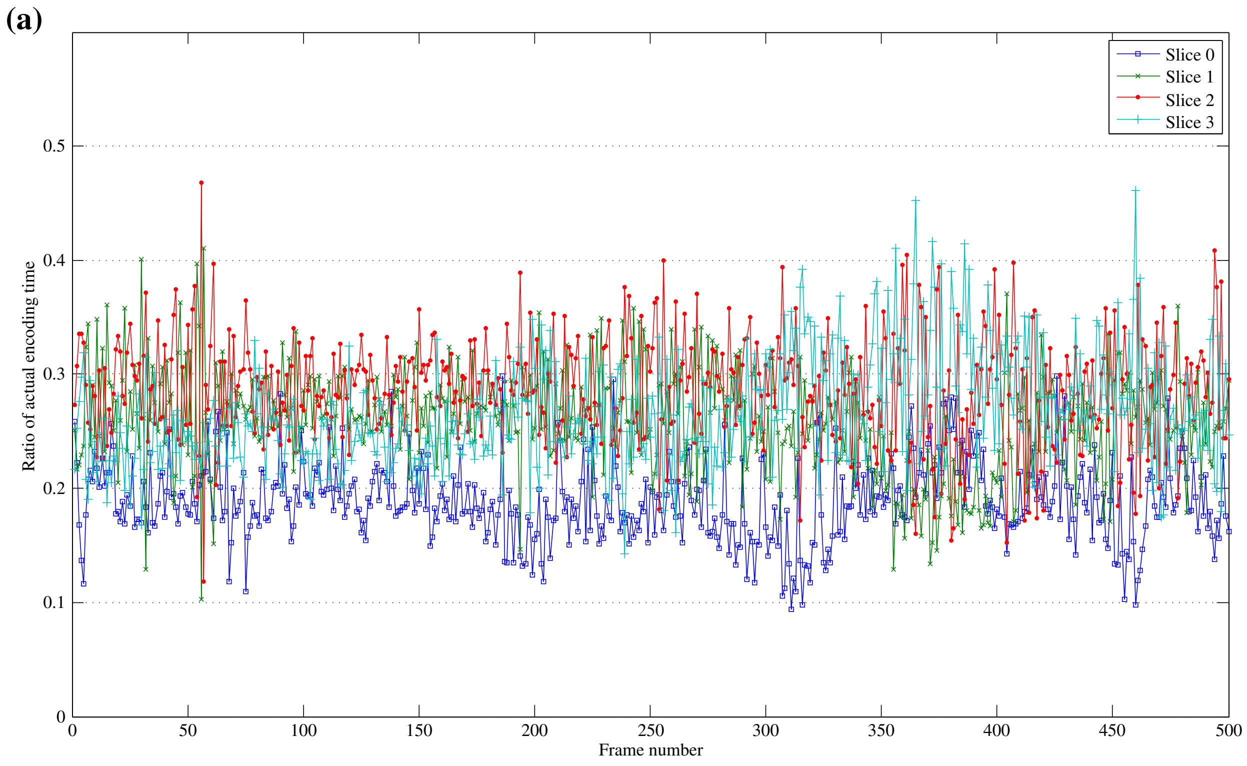


Figure 7 Ratios of predicted complexity for each temporal layer. (a) Temporal layer 3. **(b)** Temporal layer 2. **(c)** Temporal layer 1. **(d)** Temporal layer 0.



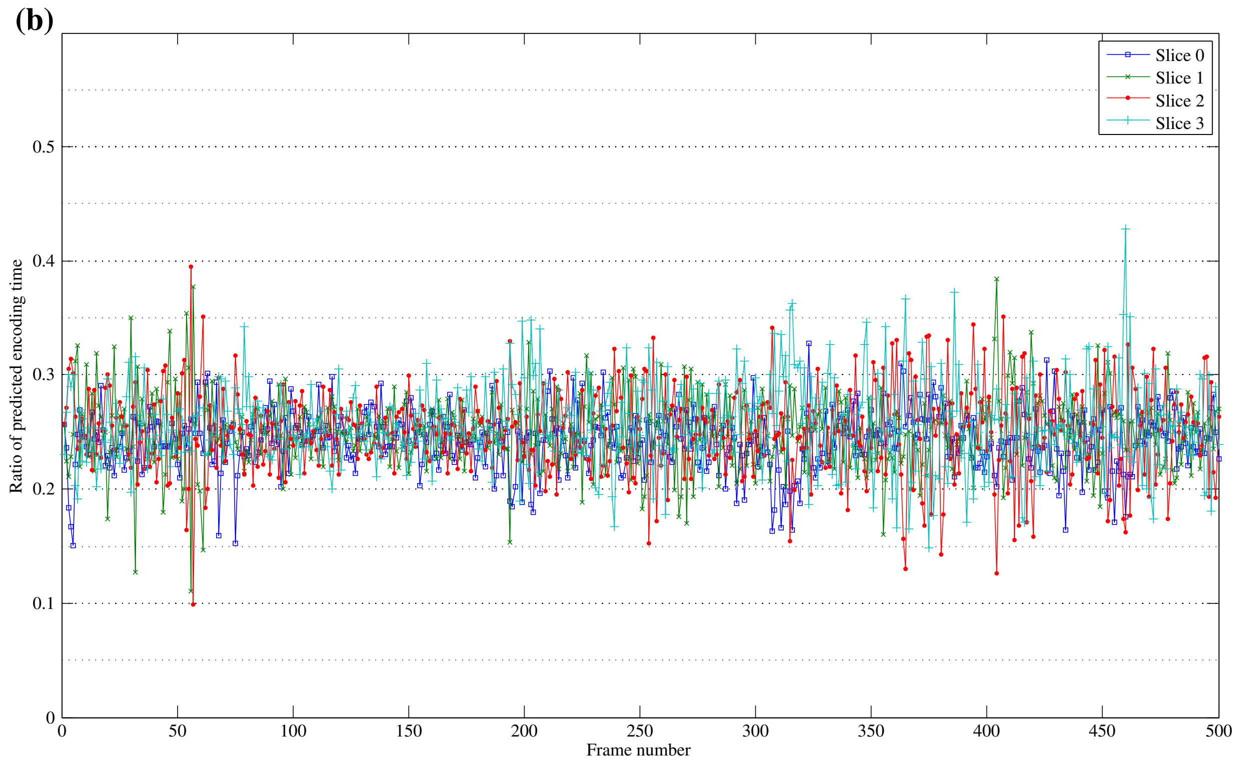


Figure 8 Ratios of actual encoding time and predicted complexity for BasketballDrive sequence. (a) Ratios of actual encoding time for the four slices. **(b)** Ratios of predicted complexity for the four slices.

fixed-point operation. The complexity of the lth CTU is defined by

$$\mathrm{CC}_i(l) = \sum_{s\in S}\sum_{m\in M} \mathrm{CEM}(s,m) \times \mathrm{CHK}(s,m|l) \tag{5}$$

$$\mathrm{CHK}(s,m|l) = \begin{cases} 1, \text{if selected}(s,m|l) \\ 0, \text{otherwise} \end{cases} \tag{6}$$

where CHK(s,m|l) represents the selected mode for the CTU. S and M are defined by {64 × 64, 32 × 32, 16 × 16, 8 × 8} and {Skip, Inter, Intra}, respectively. The predicted complexity for each slice is computed by summation of complexity for CTU and is defined by

$$\mathrm{SC}_i(k) = \sum_{l=0}^{L(k)-l} \mathrm{CC}_i(l) \tag{7}$$

The proposed estimated complexity for a slice is evaluated with the Pearson product moment correlation with HEVC common test sequences [15]. The HEVC common test sequences are the most widely used video sequences for tool experiments of HEVC standard. In our evaluation, we used three classes of sequences (class A, class B, and class C). The size of class A is 2,560 × 1,600, and it consists of two sequences (Traffic and PeopleOnStreet). The sequences are formed by cropping 4 K videos for storage and evaluation time. Class B (1,920 × 1,080) is for full HD sequences and class C (832 × 480) is widely used for mobile devices. For this evaluation, we employed these sequences, and they are coded with multiple quantization parameters (QPs) of 22, 27, 32, and 37, under the HEVC common test condition [15]. The QP values were selected to cover many video applications, such as broadcasting, video communication, and high-quality media players, by considering the quality of reconstructed videos and bitrates. Figure 6 shows the ratio of an actual encoding time and the predicted complexity for the 'BasketballDrive' sequence under random access setting with the proposed model for four slices. Table 5 shows the Pearson product moment correlation of the actual and predicted times for the test sequences. Note that the correlation coefficients can vary, depending on the coding parameters. In this paper, the correlation coefficients are computed from HM software under the common test conditions that are widely used for practical HEVC encoder. As shown in Table 5, the correlation coefficient is about 0.95, and it is quite high in predicting computational complexity, in the case of slice partitioning.

Table 6 Test sequences

Class	Sequence number	Sequence name	Frame count	Frame rate
Class B (1,920 × 1,080)	S01	Kimono	240	24
	S02	ParkScene	240	24
	S03	Cactus	500	50
	S04	BasketballDrive	500	50
	S05	BQTerrace	600	60
Class C (832 × 480)	S06	BasketballDrill	500	50
	S07	BQMall	600	60
	S08	PartyScene	500	50
	S09	RaceHorses	300	30

5.2 Adaptive slice partitioning using characteristics of temporal layers

In the current frame, the number of CTU for each slice is adaptively determined based on the complexity of the co-located slice in previously coded frames. In the proposed algorithm, the hierarchical temporal coding structure is considered to select the coded frame for the complexity prediction of each slice. Figure 7 shows the ratio of the predicted complexity of each slice in terms of temporal layers for the BasketballDrive sequence. In this evaluation, four temporal layers are used for the hierarchical coding structure, with four reference frames, based on the common test conditions. HEVC adopts a temporal layer coding structure to improve coding efficiency and temporal scalability, and each temporal layer has a different quantization parameter. The ratio of skip mode and the characteristics of CU split appear differently over the different temporal layers, and the same temporal layer has a high similarity of the actual encoding time and the predicted encoding time. As shown in Figures 6 and 7, the complexity of each slice, without considering the temporal layers, has a large fluctuation compared to that with considered temporal layers. In order to compare the complexity fluctuation of each slice, the statistical variance of complexities of each slice was measured. In the case in Figure 6, the variances of the complexity ratio of each slice are 29.23, 27.83, 61.25, and 57.87, respectively. The cases classified by the temporal layer, as shown in Figure 7, have the low variances of the complexity ratio of each slice of (a) 10.74, 17.85, 33.80, and 31.10; (b) 13.45, 23.85, 33.34, and 38.89; (c) 18.02, 16.29, 30.51, and 25.62; and (d) 3.78, 4.61, 3.30, and 6.61. The number of CTU in each slice is adjusted by an offset based on the predicted complexity of the slice in the same temporal layer of the hierarchical structure. The number of CTU in a slice, $L(k)$, and the offset to control the number of CTU in a slice, *offset* (k), are defined by

$$L_i^j(k) = \frac{\mathrm{CTU}_{\mathrm{inFrame}}}{N} + \mathrm{offset}_i^j(k) \tag{8}$$

Table 7 HM 9.0 vs. optimized HEVC encoder software

Sequence		RA			LD		
		SIMD (A)	Frame-level IF (B)	A + B	SIMD (A)	Frame-level IF (B)	A + B
B	S01	14.13	17.79	31.92	15.74	19.44	35.18
	S02	12.38	20.18	32.56	14.78	21.10	35.88
	S03	14.09	19.56	33.65	16.23	20.26	36.49
	S04	15.16	16.85	32.01	17.62	17.12	34.74
	S05	11.93	20.35	32.28	13.59	21.58	35.17
C	S06	14.33	18.51	32.84	16.49	19.60	35.99
	S07	13.84	20.90	34.74	16.02	20.95	36.97
	S08	11.88	18.49	30.37	13.44	19.94	33.38
	S09	14.67	15.03	29.70	17.23	15.54	32.77
Average (B)		13.54	18.95	32.48	15.59	19.90	35.49
Average (C)		13.68	18.23	31.91	15.80	18.98	34.78

$$\text{offset}_i^j(k) = \text{offset}_{i-1}^j(k) + \left(\frac{1}{N} - \frac{\text{SC}_{i-1}^j(k)}{\sum_{n=0}^{N-1} \text{SC}_{i-1}^j(n)}\right) \times \text{CTU}_{\text{inFrame}} \qquad (9)$$

where $L(k)$ is the number of CTU in the kth slice, i is the frame index, j is the temporal layer index, and k is the slice index. Also, N is the number of slices in a frame, and $\text{CTU}_{\text{inFrame}}$ is the number of CTUs in the frame. In Equation 9, the CTU offset for each slice is set to the additional number of CTUs. The proposed algorithm adopts the adaptive slice partitioning method, with the difference between the ideal complexity for each slice, and the ratio of predicted complexity, which achieves the speed-up of slice-level parallelism. Figure 8 shows the actual encoding time and predicted encoding time using the proposed load-balanced slice parallelization. This shows that the complexity load is quite well balanced compared to that shown in Figure 6. In addition, the maximum difference between the ratios of actual encoding time and the predicted one is 0.09363, and the minimum difference is 4×10^{-5}.

6 Experimental results

In this section, we show the performance of the proposed optimization techniques for HEVC encoder in terms of Bjontegaard distortion-bitrate (BD-BR) [28], Bjontegaard distortion peak signal-to-noise ratio (BD-PSNR) [28], and average time saving (ATS). In order to evaluate the efficiency of the proposed methods, HM 9.0 reference software was utilized. A PC equipped with an Intel® Core™ i7-3930 K CPU (six cores, 12 threads are supported with hyper-threading) and 16 GB memory was used for this evaluation. Intel® C++ 64-bit compiler XE 13.0 and VTune analyzer (performance monitoring tool) were used in a

Table 8 HM 9.0 vs. slice parallelization using OpenMP

Sequence		RA			LD		
		BD-BR (%)	BD-PSNR (dB)	ATS (%)	BD-BR (%)	BD-PSNR (dB)	ATS (%)
B	S01	1.79	−0.05	70.25	1.49	−0.05	70.60
	S02	1.00	−0.03	71.53	0.89	−0.03	70.61
	S03	1.38	−0.03	71.08	1.29	−0.03	71.60
	S04	2.06	−0.05	68.03	1.53	−0.04	68.65
	S05	1.39	−0.02	70.27	1.35	−0.03	70.23
C	S06	3.41	−0.14	68.95	2.60	−0.10	69.26
	S07	3.78	−0.14	66.98	2.89	−0.11	66.98
	S08	1.58	−0.07	68.04	1.45	−0.06	69.61
	S09	2.96	−0.11	68.53	2.03	−0.08	68.75
Average (B)		1.52	−0.04	70.23	1.31	−0.03	70.34
Average (C)		2.93	−0.12	68.13	2.24	−0.09	68.65

Windows 7 64-bit operating system. The encoding configuration of this evaluation is set as follows:

(a) According to HEVC common test condition [15]
(b) Profile: HEVC main profile (MP) [1]
(c) Level: Level 4.1 [1]
(d) Encoding structure: RA and LD
(e) QP value: 22, 27, 32, 37
(f) Test sequences: HEVC common test sequences (classes B and C) in Table 6

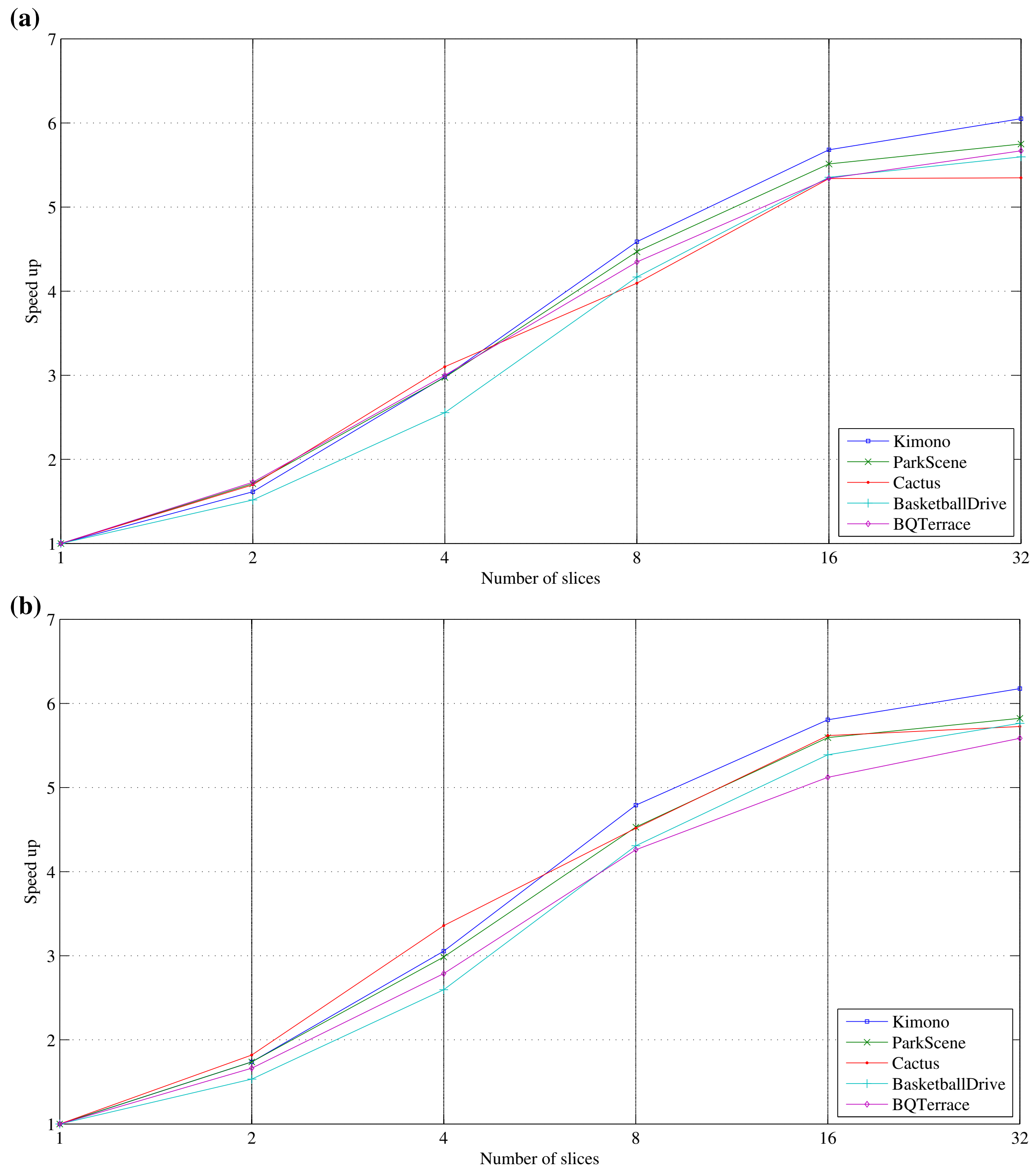


Figure 9 Average speed-up in terms of the number of slices for class B. (a) Random access case. **(b)** Low-delay case.

In this work, we define the average load balance saving (ALS) to measure the prediction accuracy of the proposed load balance performance against an anchor. ALS is calculated by

$$ALS(\%) = \frac{\mathrm{AML}_{\mathrm{anchor}} - \mathrm{AML}_{\mathrm{proposed}}}{\mathrm{AML}_{\mathrm{anchor}}} \times 100(\%) \qquad (10)$$

where $\mathrm{AML}_{\mathrm{anchor}}$ is the average of the maximum ratio of complexity load over all the slices for the anchor, and $\mathrm{AML}_{\mathrm{proposed}}$ is the average of the maximum ratio of complexity load over all the slices for the proposed algorithm. In addition, BD-BR (%) for bitrate increase, BD-PSNR (dB) for objective quality decrease, and ATS (%) for average time saving were evaluated. Note that the ATS is defined by

$$\mathrm{ATS}(\%) = \frac{\mathrm{Etime}_{\mathrm{anchor}} - \mathrm{Etime}_{\mathrm{proposed}}}{\mathrm{Etime}_{\mathrm{anchor}}} \times 100(\%) \qquad (11)$$

where $\mathrm{Etime}_{\mathrm{anchor}}$ is the encoding time of the anchor encoder and $\mathrm{Etime}_{\mathrm{proposed}}$ is the proposed method.

Firstly, the ATS comparison between the anchor and the proposed software optimizations will be shown. Secondly, the coding efficiency of the slice parallelism using OpenMP will be presented for the four-slice case. Thirdly, the coding efficiency of the proposed load-balanced slice parallelism will be presented. Finally, the coding efficiency of the overall proposed encoder based on software optimization and parallelization will be evaluated, comparing to the HM 9.0 reference encoder.

Table 7 shows ATS performances of SIMD implementation for cost functions and the frame-level interpolation filter on HM 9.0. As mentioned in Section 2, interpolation filter, SAD, SATD, transform, and inverse-transform are the main sources of computation load of HEVC encoder, and they take 35.87%, 14.30%, 15.78%, and 3.30% of the overall encoder time, respectively. In Table 7, the ATS gain of the developed SIMD implementations is found to be from 13.54% to 15.80%; and the ATS gain of frame-level interpolation is from 18.23% to 19.90%. With the optimized SIMD implementation, the computational complexities of SAD, SATD, and transform/inverse-transform reduce by approximately up to 50%. In addition, the amount of complexity reduction with the frame-level interpolation filter is about 60% to 70%. Through software optimization, the total ATS gain of the developed optimization method is 31.91% to 35.49%, without any loss in coding efficiency.

Table 8 shows performance evaluation results of the slice parallelization using OpenMP for the four slices. Compared to cases without parallelization, four-slice parallelization for an entire frame with HM 9.0 encoder yields an ATS gain of 70.06% with only 1.52% BD-BR increase and 0.037 dB BD-PSNR reduction for class B and an ATS gain of 69.21% with only 3.36% BD-BR increase and 0.128 dB BD-PSNR reduction for class C, respectively.

Figure 9 illustrates average speed-up factors of the slice parallelization, in terms of the number of slices (2, 4, 8, 16, and 32). As shown in Figure 9, the speed-up also increases up to 6, as the number of slices increases. We can see that the increasing trend becomes slow, from the eight-slice case. The speed-up factor according to the number of slices does not linearly increase due to data communication; memory accesses; context switching overhead; complexity imbalance over the slices; and frame-level sequential processes, such as DBF, SAO, and entropy coding. Note that we used an Intel processor that has six cores, with hyper-threading technology. The hyper-threading technology can somehow reduce the context switching overhead to improve parallel performance. However, the speed-up factor is saturated to nearby 5.5 to 6.0 due to the communication and other overheads [29,30]. For class B sequence, the speed-up factors are

Table 9 BD-BR, ATS, and ALS for slice and load-balanced slice parallelization

Sequence		RA			LD		
		BD-BR (%)	ATS (%)	ALS (%)	BD-BR (%)	ATS (%)	ALS (%)
B	S01	−0.01	13.44	16.33	−0.04	12.03	11.72
	S02	0.01	11.31	14.94	−0.02	12.44	13.25
	S03	0.05	0.46	2.95	−0.01	−0.16	14.86
	S04	0.16	18.05	22.58	0.05	17.82	20.83
	S05	−0.01	10.90	11.16	−0.08	14.63	16.18
C	S06	0.16	5.64	6.29	0.10	6.50	7.88
	S07	0.33	17.71	15.99	0.23	19.13	18.55
	S08	0.18	8.01	10.17	0.08	8.02	8.89
	S09	−0.01	8.72	10.62	0.02	8.90	9.65
Average (B)		0.04	10.83	13.59	−0.02	11.35	15.37
Average (C)		0.17	10.02	10.77	0.11	10.64	11.24

Table 10 HM 9.0 vs. the proposed accelerated and parallelized HEVC encoder

Sequence		RA			LD		
		BD-BR (%)	BD-PSNR (dB)	ATS (%)	BD-BR (%)	BD-PSNR (dB)	ATS (%)
B	S01	3.88	−0.12	90.13	3.29	−0.10	89.34
	S02	3.45	−0.11	91.33	3.56	−0.11	90.47
	S03	4.81	−0.10	89.12	3.96	−0.09	88.77
	S04	4.34	−0.10	88.26	3.08	−0.07	87.93
	S05	4.52	−0.07	91.28	3.32	−0.06	90.19
C	S06	5.44	−0.22	86.86	3.84	−0.15	86.72
	S07	7.46	−0.28	88.92	5.41	−0.21	88.31
	S08	4.30	−0.18	86.75	3.65	−0.15	86.10
	S09	6.10	−0.23	85.44	3.76	−0.15	85.46
Average (B)		4.20	−0.10	90.02	3.44	−0.09	89.34
Average (C)		5.83	−0.23	86.99	4.17	−0.17	86.65

higher than those for any other sequences, in cases of four to eight slices parallelization. Based on the speed-up factors for Class B, we can predict that speed-up factors with eight or more slice parallelization for ultrahigh-resolution sequences such as 4 K (3,840 × 2,160) can be higher than those for lower resolution videos.

Table 9 shows the performance comparison between slice parallelism with OpenMP and the proposed load-balanced slice parallelism. A frame is partitioned into four slices for fair evaluation; and two fast encoding algorithms, CFM [23] and ECU [24], adopted for HM, are employed for the evaluation of the proposed load-balanced parallelization. In Table 9, the proposed load-balanced algorithm achieves 10.5% ATS gain, on average (minimum −0.16% and maximum 19.13%), by adaptively controlling the number of CTUs in a slice. Moreover, the ratio of the maximum complexity load is highly reduced in ALS gain by 12.89% to 14.94%. Note that the bottleneck of parallelization is the maximum computational load for one, over all the slices, and it is crucial to reduce the ratio of the maximum complexity load for overall performance. In our implementation, we found that the ALS reduces by load balancing; as a result, the overall encoding speed is moderately improved. The average ratio of encoding time saving is 0.15%, which is relatively small, for sequences whose complexity load gap over multiple slices, for example, 'Cactus' sequence, is small before the load balancing. However, the amount of encoding time saving is about 12% to 18% for sequences whose complexity load gap among slices, such as BasketballDrive and 'Kimono' sequences, is large, without load balancing. In terms of coding gain, it is confirmed that BD-BR and BD-PSNR losses are quite small compared to uniform slice partition.

BD-BR, BD-PSNR, and ATS of the proposed fast HEVC encoder against HM 9.0 encoder are shown in Table 10. A frame is partitioned into four slices. The proposed fast HEVC encoder yields 89.34% to 90.02% in ATS compared to HM 9.0 encoder with only 3.44% to 4.20% BD-BR increase and 0.09 to 0.10 dB BD-PSNR decrease for class B sequences. For class C sequences, we found that BD-BR increases by 4.17% to 5.83% and BD-PSNR decreases by 0.17 to 0.23 dB. We found that coding loss for lower resolution videos is moderate higher than that for higher resolution ones. In addition, we evaluated BD-BR, DB-PSNR, and ATS of the proposed HEVC encoder, on top of HM9.0, for class A (2,560 × 1,600). Note that class A consists of two sequences (Traffic and PeopleOnStreet). As shown in Table 11, we found that BD-BR increases by 2.29% to 3.72%, and BD-PSNR decreases by 0.09 to 0.17 dB, on top of HM9.0, with ATS gains of 85.48% to 91.14%. Figure 10 illustrates the RD comparison of HM 9.0 and the proposed accelerated and parallelized HEVC encoder. Thinking about the

Table 11 BD-BR, BD-PSNR, and ATS of the proposed HEVC encoder for class A (2,560 × 1,600)

Sequence	RA			LD		
	BD-BR (%)	BD-PSNR (dB)	ATS (%)	BD-BR (%)	BD-PSNR (dB)	ATS (%)
Traffic	3.66	−0.12	91.14	3.72	−0.17	85.48
PeopleOnStreet	2.80	−0.09	90.43	2.29	−0.11	85.80
Average	3.23	−0.11	90.79	3.01	−0.14	85.64

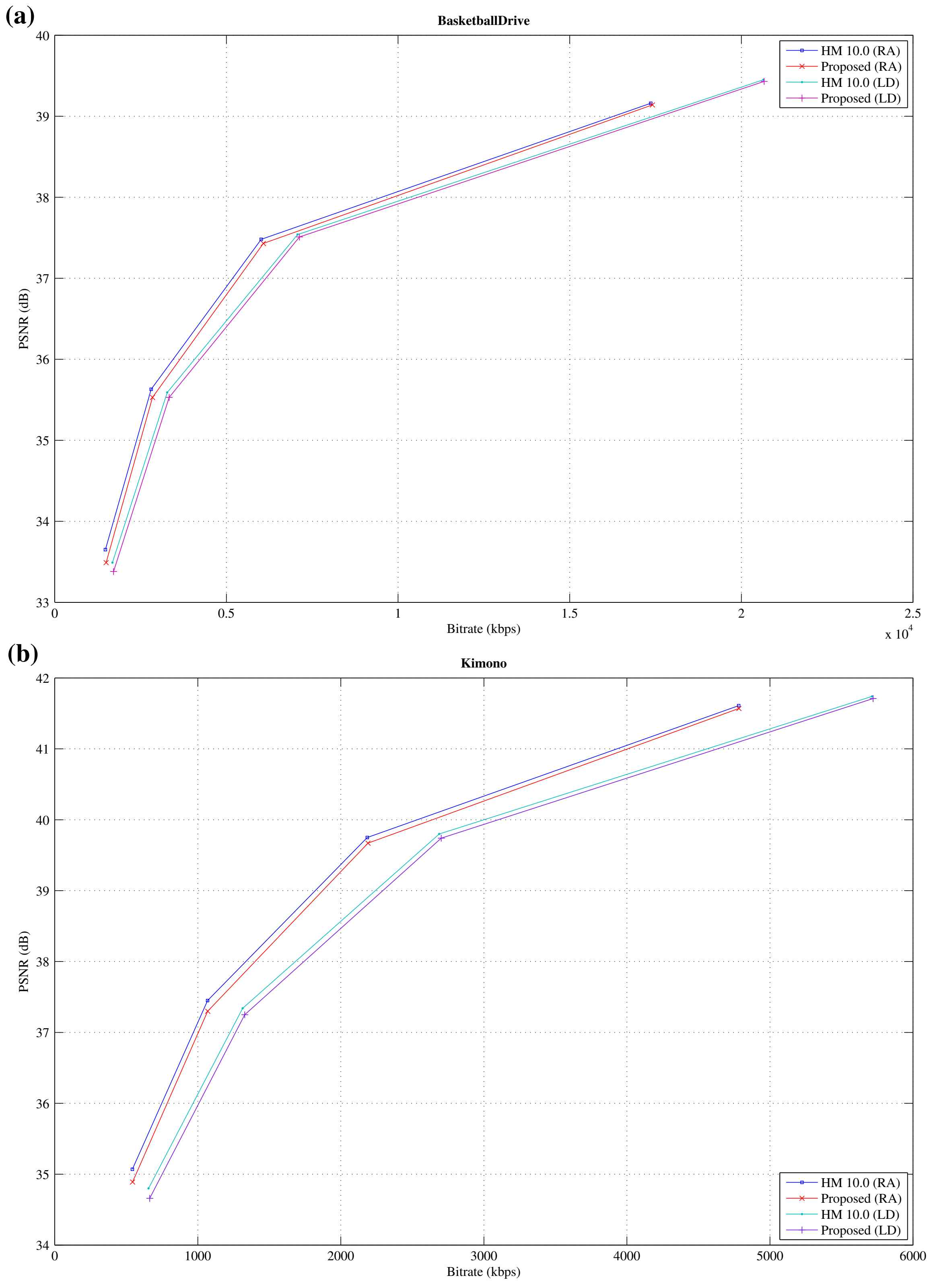


Figure 10 RD comparison of HM 9.0 and the proposed accelerated and parallelized HEVC encoder. (a) BaksetballDrive sequence. **(b)** Kimono sequence.

90.0% ATS performance gain, the RD performance loss, as shown in Table 10, is quite negligible, even containing the loss from the fast encoding algorithm and slice partitioning.

7 Conclusions

In this paper, the computational complexity of the HM 9.0 encoder was analyzed for acceleration and parallelization of the HEVC encoder. We identified five key modules for the HM 9.0 encoder, requiring dominant computing cycles. Based on the complexity analysis, two software optimization methods were used for acceleration: the frame-level interpolation filter and SIMD implementation. In addition, load-balanced slice parallelization is proposed. Software optimization methods achieve 33.56% of the average time saving, with any coding loss. In addition, load balancing for the slice parallelization method achieves about 10% of average time saving compared to uniform slice partition. The overall average time saving of the proposed HEVC encoder yields approximately 90% compared to HM 9.0 with acceptable coding loss. HEVC encoder with the proposed methods can compress full HD videos at approximately 1 fps speed in a commercial PC environment, without any hardware acceleration.

Further study will be focused on additional software optimization, fast encoding algorithm, and tile-level parallel processing for real-time encoder of HEVC.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
This research was partly supported by the IT R&D program of MSIP/KEIT [10039199, A Study on Core Technologies of Perceptual Quality based Scalable 3D Video Codecs], the MSIP (Ministry of Science, ICT & Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2013-H0301-13-1011) supervised by the NIPA (National IT Industry Promotion Agency), and the grant from the Seoul R&BD Programs (SS110004M0229111).

Author details
[1]Department of Computer Engineering, Kwangwoon University, Wolgye-dong, Nowon-gu, Seoul 447-1, South Korea. [2]Department of Software Design and Management, Gachon University, Seongnam, Gyeonggi 461-701, South Korea.

Received: 18 June 2013 Accepted: 5 March 2014
Published: 26 March 2014

References

1. B Bross, W-J Han, GJ Sullivan, JR Ohm, T Wiegand, *High Efficiency Video Coding (HEVC) text specification draft 9, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-K1003*, 2012
2. ITU-T and ISO/IEC JTC 1, Advanced video coding for generic audiovisual services, ITU-T Rec. H.264/and ISO/IEC 14496–10 (MPEG-4 AVC), versions 1-16, 2003-2012
3. H Samet, The quadtree and related hierarchical data structures. ACM Comput Surv (CSUR) **16**(2), 187–260 (1984)
4. W-J Han, J Min, I-K Kim, E Alshina, A Alshin, T Lee, J Chen, V Seregin, S Lee, YM Hong, MS Cheon, N Shlyakhov, K McCann, T Davies, JH Park, Improved video compression efficiency through flexible unit representation and corresponding extension of coding tools. Circuits Syst Video Technol, IEEE Trans **20**(12), 1709–1720 (2010)
5. T Wiegand, J-R Ohm, GJ Sullivan, W-J Han, R Joshi, TK Tan, K Ugur, Special section on the joint call for proposals on High Efficiency Video Coding (HEVC) standardization. Circuits Syst Video Technol, IEEE Trans **20**(12), 1661–1666 (2010)
6. K Chen, Y Duan, L Yan, J Sun, Z Guo, *Efficient SIMD optimization of HEVC encoder over X86 processors, in Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC)* (Asia-Pacific, Hollywood, CA, 2012), pp. 1–4
7. G Clare, F Henry, S Pateux, *Wavefront parallel processing for HEVC encoding and decoding, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-F274*, 2011
8. I-K Kim, K McCann, K Sugimoto, B Bross, W-J Han, *HM9: High Efficiency Video Coding (HEVC) test model 9 encoder Description, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-K1002*, 2012
9. K McCann, WJ Han, IK Kim, JH Min, E Alshina, A Alshin, T Lee, J Chen, V Seregin, S Lee, YM Hong, MS Cheon, N Shlyakhov, Samsung's response to the call for proposals on video compression technology, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-A124. (2010)
10. R De Forni, D Taubman, On the benefits of leaf merging in quad-tree motion models. IEEE Int Conf Image Process **2005**, 858–861 (2005)
11. J Jung, B Bross, P Chen, W-J Han, *Description of core experiment 9: MV coding and skip/merge operations, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-D609*, 2011
12. Y Yuan, X Zheng, X Peng, J Xu, IK Kim, L Liu, Y Wang, X Cao, C Lai, J Zheng, Y He, H Yu, CE2: non-square quadtree transform for symmetric and asymmetric motion partition, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-F412. (2011)
13. Joint Collaborative Team on Video Coding, (JCT-VC) of ITU-T SG 16 WP 3 and ISO/IEC JTC 1/SC 29/WG 11, HM-9.0 reference software. (2014)
14. VTune™Amplifier XE 2013 from Intel. (2014). http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/
15. F Bossen, *Common HM test conditions and software reference configuration, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-K1100*, 2012
16. GJ Sullivan, JR Ohm, WJ Han, T Wiegand, Overview of the High Efficiency Video Coding (HEVC) standard. IEEE Transactions on Circuits and Systems for Video Technology **22**(12), 1649–1668 (2012)
17. A Fuldseth, M Horowitz, S Xu, A Segall, M Zhou, *Tiles, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-F335*, 2011
18. F Henry, S Pateux, *Wavefront parallel processing, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-E196*, 2011
19. CC Chi, M Alvarez-Mesa, B Juurlink, G Clare, F Henry, S Pateux, T Schierl, Parallel scalability and efficiency of HEVC parallelization approaches. IEEE Transactions on Circuits and Systems for Video Technology **22**(12), 1827–1838 (2012)
20. A Alshin, E Alshina, JH Park, WJ Han, *DCT based interpolation filter for motion compensation in HEVC, in Proceedings of the SPIE 8499 Applications of Digital Image Processing XXXV* (San Diego, CA, 2012)
21. Intel, Intel 64 and IA-32 architectures software developer manuals. (2014). http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
22. M Budagavi, V Sze, *Unified forward + inverse transform architecture for HEVC, in 19th IEEE International Conference on Image Processing (ICIP), 30 September 30 2012 to 3 October* (Orlando, Florida, USA, 2012), pp. 209–212
23. RH Gweon, Y-L Lee, J Lim, *Early termination of CU encoding to reduce HEVC complexity, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-F045*, 2011
24. K Choi, ES Jang, *Coding tree pruning based CU early termination, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-F092*, 2011
25. J Yang, J Kim, K Won, H Lee, B Jeon, *Early skip detection for HEVC, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC) document JCTVC-G543*, 2011
26. N Zhang, C-H Wu, Study on adaptive job assignment for multiprocessor implementation of MPEG2 video encoding. IEEE Trans. Ind. Electron **44**(5), 726–734 (1997)
27. B Jung, B Jeon, Adaptive slice-level parallelism for H.264/AVC encoding using pre macroblock mode selection. J Vis Commun Image. Representation **19**(8), 558–572 (2008)
28. G Bjontegaard, *Document VCEG-M33: calculation of average PSNR differences between RD-curves, ITU-T VCEG Meeting* (Austin, Texas, USA, 2001)

29. X Tian, Y-K Chen, M Girkar, S Ge, R Lienhart, S Shah, *Exploring the use of hyper-threading technology for multimedia applications with Intel® OpenMP compiler, in Proceedings of International Symposium on Parallel and Distributed Processing 2003* (Nice, France, 2003)
30. S Sankaraiah, LH Shuan, C Eswaran, J Abdullah, Performance optimization of video coding process on multi-core platform using GOP level parallelism. Int J Parallel Program Springer , 1–17 (2013)

doi:10.1186/1687-5281-2014-16
Cite this article as: Ahn *et al.*: Implementation of fast HEVC encoder based on SIMD and data-level parallelism. *EURASIP Journal on Image and Video Processing* 2014 **2014**:16.