SIMD Acceleration for HEVC Encoding on DSP

Yongfei Zhang[1*], Rui Fan[2], Chao Zhang[1], Gang Wang[1], and Zhe Li[3]
[1] Beijing Key Lab of Digital Media, School of Computer Science and Engineering, Beihang University, Beijing, China, 100191
[2] China Academy of Electronic and Information Technology, Beijing, China, 100041
[3] Shandong Province Key Laboratory of Wisdom Mine Information Technology, Shandong University of Science and Technology, Qingdao China, 266590
*Corresponding Author: Yongfei Zhang (E-mail: yfzhang@buaa.edu.cn Tel: +86-1082314108)

***Abstract*— As the new generation video coding standard, High Efficient Video Coding (HEVC) significantly improves the video compression efficiency, which is however at the cost of a far more computational payload than the capacity of real-time video applications and general purpose processors. In this paper, we focus on the SIMD-based fast implementation of the HEVC encoder over modern TI Digital Signal Processors (DSPs). We first test the DSP-based HEVC encoder and indentify the most time-consuming encoding modules. Then SIMD instructions are exploited to improve the parallel computing capacity of these modules and thus speed up the encoder. The experimental results show that the proposed implementations can significantly improve the encoding speed of the DSP-based HEVC encoder, with a speedup ratio of 8.38-87.32 over the original C-based encoder and 1.59-6.56 over o3-optimization enabled encoder.**

***Index Terms*— HEVC, Encoder, DSP, SIMD**

I. INTRODUCTION

High Efficiency Video Coding (HEVC)[1], the latest video compression standard developed by the joint collaborative team on video coding (JCTVC), can significantly improve the coding performance compared its predecessor H.264/AVC[2], which is however achieved at a much improved computational cost of up to 2-10 times higher computational complexity, which makes it quite difficult to apply in real-time video applications[3].

Considering the high coding efficiency and pervasive applications of HEVC, low-complexity thus low-power-consumption HEVC encoder/decoder is urgently needed [3-6]. Besides plenty of algorithm-level optimizations for the encoding modules have been proposed to speed up either the encoder or the decoder [4-6], much attention has also been paid on the code/instruction level optimization to further and more significantly reduce the computational complexity[7-12].

Since HEVC decoder is much less computational intensive than the encoder, there are many literatures reporting the decoder implementations for HEVC, either on general purpose processor CPU/GPUs [7-10], FPGAs[11] or digital signal processors[12-13]. However, few research has been conducted on the code/instruction level optimization implementation for the more important and time-intensive HEVC encoders [14-17], most of which are on CPUs/GPUs.

Programmable processors such as multi-core digital signal processors are especially good at computational intensive tasks with very low power consumption, which make it very competitive and promising solution to help reduce the high computational burden and put it into real-time video applications. However, up to the best of our knowledge, [17] is the only work which addressed the encoder implementation on DSPs, which however only tackles the implementation of most simple SAD and SSE on DSPs and leave the most time-consuming encoding modules, such as inter and intra prediction, DCT/IDCT(Inverse/ Discrete Cosine Transform) and Sample Adaptive Offset(SAO), untouched.

In this paper, to address the problem of optimized implementation of HEVC video encoder on the powerful main-stream TI TMS320C6678 DSPs[18], various Single-instruction-multiple-data (SIMD) optimizations are explored to optimize the most time-consuming encoding modules of the DSP-based HEVC encoder. Up to the best of our knowledge, this is the first of this kind to comprehensively implement the encoder using SIMD. With our careful design, the encoding speed of the DSP-based HEVC encoder is significantly improved, which will help make possible to achieve real-time implementation of HEVC for practical video applications.

The rest of this paper is organized as follows. Section 2 analyses and identifies the most time-consuming encoding modules of the DSP-based HEVC encoder. After a brief introduction of the Single Instruction Multiple Data (SIMD) in the chosen DSP, the SIMD acceleration design and implementation for these modules are elaborated in Section 3. Experimental results are shown in Section 4. Finally, conclusions are drawn in Section 5.

II. COMPLEXITY ANALYSIS OF DSP-BASED HEVC ENCODER

In order to identify the most time-consuming modules in the DSP-based HEVC encoder, we analyze the execution time of our HM16.0 [19]-based embedded HEVC encoder on TI TMS320C6678 DSP. The first 100 frames of the video sequences in Class B (1080p) are encoded with a QP of 32 under Low Delay P main configuration. The rest of the configurations are kept unchanged [20]. The average execution time of the major modules of the DSP-based encoder is shown in Fig. 1. As can be seen, although it is slightly different from that in HM, the most time-consuming modules are Inter prediction (mainly half/quarter pixel

This work was partially supported by the National Key R&D Program of China (Grant No.2016YFC0801001), the NSFC Key Project (No. 61632001) and the National Natural Science Foundation of China (No 61502278, 61772054). This paper is partially done when Rui Fan and Zhe Li were with Beijing Key Lab of Digital Media, School of Computer Science and Engineering, Beihang University, Beijing, China, 100191.

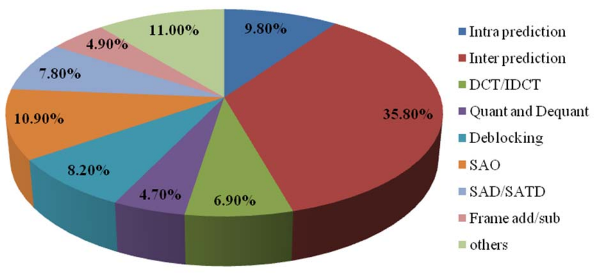


Fig. 1 Execution Time Analysis of DSP-based HEVC Encoder

Interpolation), Intra prediction, DCT/IDCT, Quantization/De-quantization, deblocking, SAO, SAD/SATD, Frame add/sub. As will be shown in Section III and IV, he computational performance of the first four modules can be effectively improved by using SIMD methods.

III. Proposed Scheme

To improve the video coding speed, several critical modules will be written in linear assembly language using the powerful SIMD instructions available in TI TMS320C6678 DSPs. The main idea is to improve the parallelism of data processing by using SIMD instructions.

In this section, we first give a brief introduction of the Single Instruction Multiple Data instruction sets available in the TI TMS320C6678 DSP. And we then elaborate the SIMD acceleration design and implementation for these time-consuming encoding modules identified in Section II.

A. *C66x CorePac*[21-22]

The TMS320C6678 is an eight-core, high-performance DSP with both fixed-point and floating-point precision capabilities. C66x CorePac is based on a Very Long Instruction Word (VLIW) architecture, which differs from Reduced Instruction Set Computing (RISC) or Complex Instruction Set Computing (CISC) architectures by having multiple execution units which can execute several instructions in parallel. The C66x CorePac has two identical data paths, A and B, each with four unique functional units (M, L, S, and D). The M unit performs multiplication operations, while the L and S units handle addition, subtraction, logical, branching, and bitwise operations. The D unit is responsible for load/store and address calculations. All the functional units provide vector-processing capabilities using the SIMD instruction set included in the C66x CorePac. The SIMD instructions can operate on up to 128-bit vectors providing data-level parallelism within each core. With L, M, and S units on the two data paths, each core can perform eight single-precision or two double-precision multiply-add operations in one cycle. The TMS320C6678 also provides thread-level parallelism by scheduling application on the eight available cores, which is has been implemented in our DSP-based HEVC encoder and thus out of the scope of this paper.

B. *Inter Prediction*

As is well known [1], inter prediction is the most powerful encoding techniques to remove the inter-frame redundancy. However, as shown in Fig. 1, inter prediction is responsible for about one thirds of the computational burden of the whole HEVC encoder. Furthermore, the major computation of inter prediction lies in the interpolation for quarter-pixel motion estimation (ME) and compensation (MC). Thus, we mainly focus on the SIMD implementation here.

In HEVC, since quarter-sample precision is used for ME/MC, 7-tap or 8-tap filters are used for interpolation of fractional-sample positions of the luminance component, (compared to six-tap filtering of half-sample positions followed by linear interpolation for quarter-sample positions in H.264/MPEG-4 AVC)[1], which make the implementation different and more complicated.

Taking the design for the luminance component as an example, the interpolation filter in HEVC is composed of two steps, namely horizontal filtering first and then vertical filtering. Please refer to Ref. [1] for more details of the interpolation as well as filter coefficients.

Since storage and access of the source data, i.e., the integer pixels for horizontal and vertical interpolation are quite different, the implementation of the horizontal and vertical interpolation will be elaborated separately as below.

Some of the notations used are defined as below. Let A-H denote the value of luminance component of the eight integer pixels used for interpolation while C0-C7 8-tap interpolation filter coefficients. Ai is the pixel in row i and column A. It is the intermediate result after horizontal interpolation filter, and Ii is 14 bit precision to ensure the accuracy. X is the final filtered pixel through vertical interpolation whose precision is the same to the raw data.

1) Horizontal Interpolation

The proposed implementation of luma horizontal interpolation filter is shown in Fig. 2. Firstly, eight reference frame pixels on the horizontal direction are loaded into a register pair which is composed of two adjacent 32-bit registers. Then the interpolation filter is divided into two ways, and each way completes four-pixel interpolation filters. Finally, the two-way sums results are added together, taking into consideration the interpolation offset and the 14-bit luma horizontal filter result is obtained. SIMD instructions involved as in Fig. 2 are specified as follows.

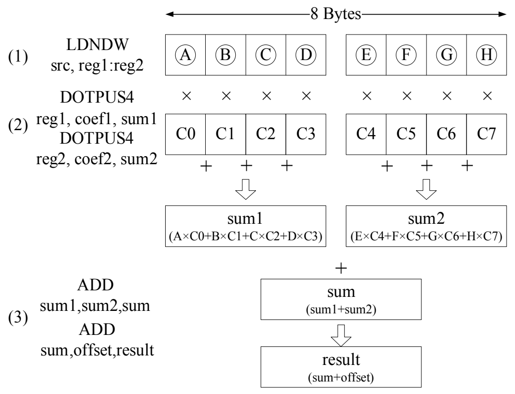


Fig. 2. Horizontal interpolation filter for inter prediction

The DOTPUS4 instruction set supports dot-product between four sets of packed 8-bit values. The source operands of DOTPUS4 are two 32-bit registers. One is used to store four reference frame pixels ranging between 0 and 255. And the luma interpolation filter coefficients which are in the range from -128 to 127 are stored in the other one register.

2) Vertical Interpolation

The intermediate result after horizontal interpolation is one of the inputs in vertical interpolation filter. The essential difference between horizontal and vertical interpolation filter is whether the logically adjacent points are also continuous in space. The dot-product operations similar to DOTPUS4 are unfit for vertical interpolation filter. This is because, to use dot-product operations, we need spend too much cost on transforming logically adjacent points to the corresponding data organization format which is very time consuming. Therefore, DMPY2, a 4-way SIMD multiplication between signed packed 16-bit quantities, is adopted to speed up the process of vertical interpolation filter.

As shown in Fig. 3, 64-bit intermediate result (four points) from horizontal interpolation filter is loaded from L2 into registers. Eight rows of intermediate results are prepared since the number of luma interpolation filter coefficients is 8. Each row of intermediate results multiplies its corresponding interpolation filter coefficient. Each element in one row occupies 16 bits storage of one register. After the multiplication procedure, the products should be accumulated together by employing DADD a couple of times, which is a 2-way SIMD addition instruction.

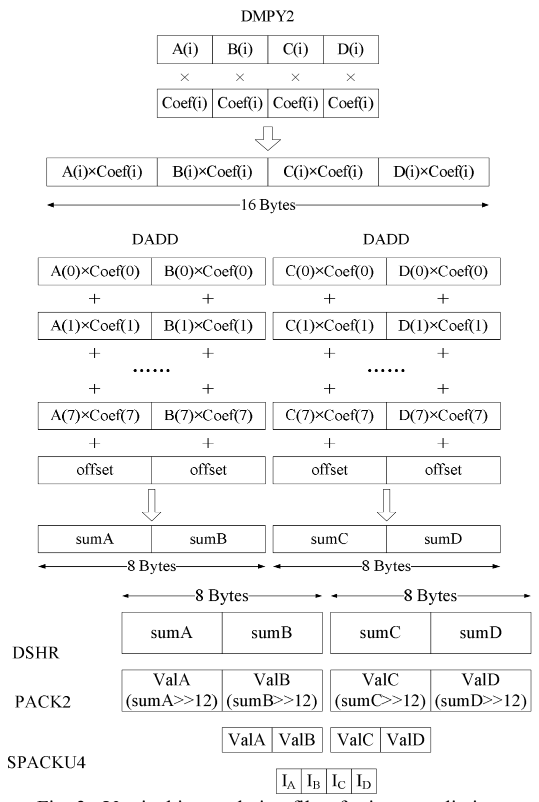

DMPY2
A(i) B(i) C(i) D(i)
Coef(i) Coef(i) Coef(i) Coef(i)
A(i)×Coef(i) B(i)×Coef(i) C(i)×Coef(i) D(i)×Coef(i)
16 Bytes
DADD
A(0)×Coef(0) B(0)×Coef(0) C(0)×Coef(0) D(0)×Coef(0)
A(7)×Coef(7) B(7)×Coef(7) C(7)×Coef(7) D(7)×Coef(7)
offset
sumA sumB sumC sumD
8 Bytes
DSHR
PACK2
ValA (sumA>>12) ValB (sumB>>12) ValC (sumC>>12) ValD (sumD>>12)
SPACKU4
I_A I_B I_C I_D

Fig. 3. Vertical interpolation filter for inter prediction

The vertical interpolation data will be obtained until a shift operation (DSHR) is done. However, extra processing steps are performed as the eventual format is 8 bits per pixel, which is different from the procedure of horizontal interpolation filter. To maintain the consistency of the data format, we pack 2 Bytes data respectively into upper and lower register halves and clip the data from 0 to 255 by SPACKU4 instruction.

C. *Intra Prediction*

Similar to that in H.264/AVC, intra prediction in HEVC are also based on spatial sample prediction while it supports a total of 35 intra prediction modes, including planar intra prediction (mode 0), DC prediction (mode 1) and angular prediction modes with different directionalities (mode 2 to mode 34). Mode 2 to 17 are horizontal directionalities, while mode 18 to 34 represent vertical directionalities [1]. The intra sample prediction process in HEVC is performed by extrapolating sample values from the reconstructed reference samples in adjacent Pus utilizing a given directionality. In order to simplify the process, all sample locations within one prediction block are projected to a single reference row or column depending on the directionality of the selected prediction mode (utilizing the left reference column for angular modes 2 to 17 and the above reference row for angular modes 18 to 34).

Fig. 4 is an example of the projection procedure for mode 16 and 20. To reduce the code redundancy, the horizontal directionality modes is realized through the corresponding vertical directionality mode by matrix transposition, because mode 18-i and mode 18+i (i=0,1,…,16) are symmetrical with respect to the main diagonal of the matrix (mode 18).

$$P_{x,y} = ((32-\omega)\cdot \mathrm{Ref[pos]} + \omega\cdot \mathrm{Ref[pos+1]} + 16) >> 5$$

$$\boldsymbol{P}_{2N\times 2N}{}^{\mathrm{T}} = \begin{bmatrix} \boldsymbol{P}_{N\times N}(1,1) & \boldsymbol{P}_{N\times N}(1,2) \\ \boldsymbol{P}_{N\times N}(2,1) & \boldsymbol{P}_{N\times N}(2,2) \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} [\boldsymbol{P}_{N\times N}(1,1)]^{\mathrm{T}} & [\boldsymbol{P}_{N\times N}(2,1)]^{\mathrm{T}} \\ [\boldsymbol{P}_{N\times N}(1,2)]^{\mathrm{T}} & [\boldsymbol{P}_{N\times N}(2,2)]^{\mathrm{T}} \end{bmatrix}$$

(2)

From Eqn. (2), we discover that the transposition of 2N×2N matrix can be realized through transpositions of its internal four N×N matrices. In other words, as long as matrix transposition of 4×4 size is achieved, we can obtain matrix transpositions of arbitrary $2^n\times 2^n$ size (n>1). The matrix transposition of 4×4 size is enough for HEVC standard, because the minimal PU size for intra prediction is 4×4. The

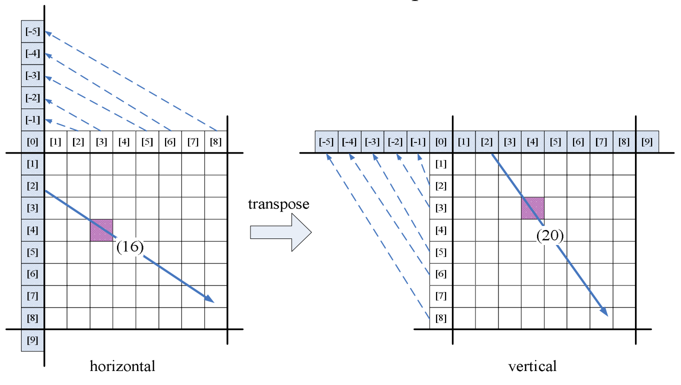


Fig. 4. Illustration of the projection procedure for mode 16 and 20.

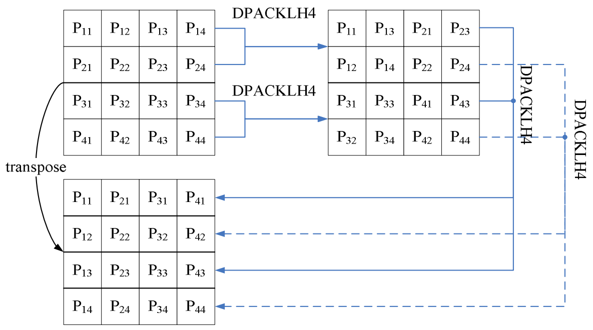


Fig. 5. Matrix transpositions of PU of size 4×4

matrix transposition process of 4×4 size can be accelerated by two steps of DPACKLH4 instruction, which is illustrated in Fig. 5. DPACKLH4 is a 2-Way SIMD instruction. It packs high bytes of four half-words to packed 8-bit, and low bytes of the same four half-words into packed 8-bits.

D. DCT/IDCT

The transform matrices are an approximation of mathematical DCT matrices, which are rounded to 8-bit integer accuracy including sign flag. The matrices are optimized for maximizing orthogonality. Smaller size transform matrices are embedded in larger size transform matrices. This simplifies implementation, since a 32×32 matrix, can do 4×4, 8×8, 16×16, and 32×32 transform. The transform process is listed as follows

$$Y = 2^{-12-\log_2 N}(C_N \times X \times C_N{}^T) \tag{3}$$

where Y is the final transform coefficient matrix and X is the prediction residual matrix. C_N represents the transform matrix of size N×N. In HM, the transform is performed by partial butterfly structure for low complexity. However, when it is implemented with SIMD on C66x DSP platform, a large amount of data reorganization though PACK instruction series will become the bottleneck of acceleration performance. Consequently, the transform process can be reformed as

$$\begin{aligned} Y &= 2^{-7+\log_2 N} \cdot 2^{-6-\log_2 N} \cdot \{C_N \times [2^{1-\log_2 N} \cdot (C_N \times X^T)]^T\} \\ &= 2^{-7+\log_2 N} \cdot Y' \end{aligned} \tag{4}$$

$2^{-12-\log2N}$ can be divided into $2^{-7+\log2N} \cdot 2^{-6-\log2N} \cdot 2^{1-\log2N}$, so the results of first transform stage will be right shifted by $\log_2 N$-1 and results of second transform stage will be right shifted by 6+$\log_2 N$, to ensure all the intermediate calculations are within 16 bit. Y' is the outputs of HEVC forward transform modules. Its values are all 15 bits plus one sign bit. It also be noted the calculation of $2^{-7+\log2N}$ is combined into the following quantization process and this item is calculated as “iTransformShift” in HM encoder.

It is discovered that the first and second transform stage is the same substantially, which both satisfy C_N×A^T. In the first stage, A is X while A is [2(C_N×X^T)] in the second stage. The (i, j)th element of C_N×A^T denoted by m_{ij} is computed as in (5).

$$m_{ij} = \sum_{k=1}^{N} c_{ik} \cdot a_{jk} \tag{5}$$

where c_{ik} is the (i, k)th element in CN and a_{jk} is the (j, k)th

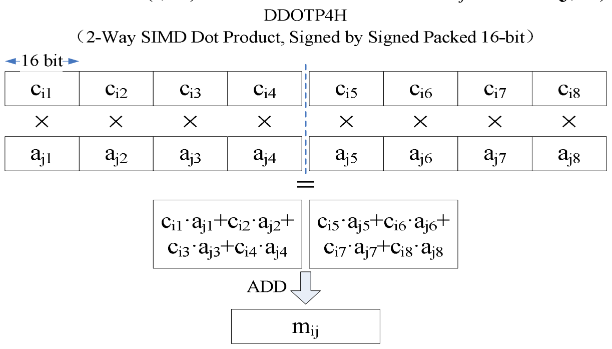

DDOTP4H
（2-Way SIMD Dot Product, Signed by Signed Packed 16-bit）
16 bit
ci1 ci2 ci3 ci4 ci5 ci6 ci7 ci8
aj1 aj2 aj3 aj4 aj5 aj6 aj7 aj8
ci1·aj1+ci2·aj2+ci3·aj3+ci4·aj4
ci5·aj5+ci6·aj6+ci7·aj7+ci8·aj8
ADD
mij

Fig. 6. DCT sketch for 8×8 block

element in A. This dot product process can be optimized by DDOTP4H instruction. In the case of 8×8 block shown in Fig. 6, DDOTP4H can simultaneously calculate 2-way dot product, which takes the same cycles of multiply instruction, such as MPY. Then the two parts of DDOTP4H output are added to obtain the final m_{ij}. And IDCT can be implemented similarly.

E. Quant and Dequant

The traditional scalar quantization method has been employed in HEVC as the default quantization means, as

$$q_{i,j} = floor\left(\frac{c_{i,j}}{Q_{\text{step}}} + f\right) \tag{6}$$

where $c_{i,j}$ and $q_{i,j}$ denote the DCT coefficient and the quantized coefficient at *(i, j)*, respectively. Q_{step} indicates the quantization step size. *floor()* is a round down function. *f* is used to control the round. Summing up the above, the ultimate quantization formula of HEVC can be expressed by Eqn. (7) and (8), whose derivational process is

$$|q_{i,j}| = (|d_{i,j}| \cdot \text{MF} + f\,') >> (\text{qbits} + \text{T\_shift}) \tag{7}$$

$$\text{sign}(q_{i,j}) = \text{sign}(d_{i,j}) \tag{8}$$

where qbits=14+floor(QP/6). T_shift. MF is a constant integer determined by QP. f’ is a constant influenced by frame type.

The sign values of the quantized coefficients can be calculated rapidly through SIMD. An example is given in Fig. 7, where four sign values are computed simultaneously through DSHR2, NOT, DSHRU2 and OR instruction.

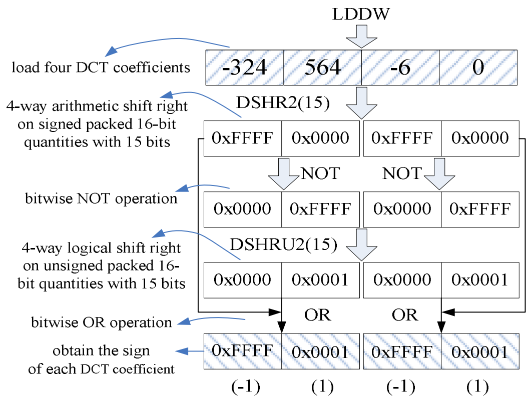
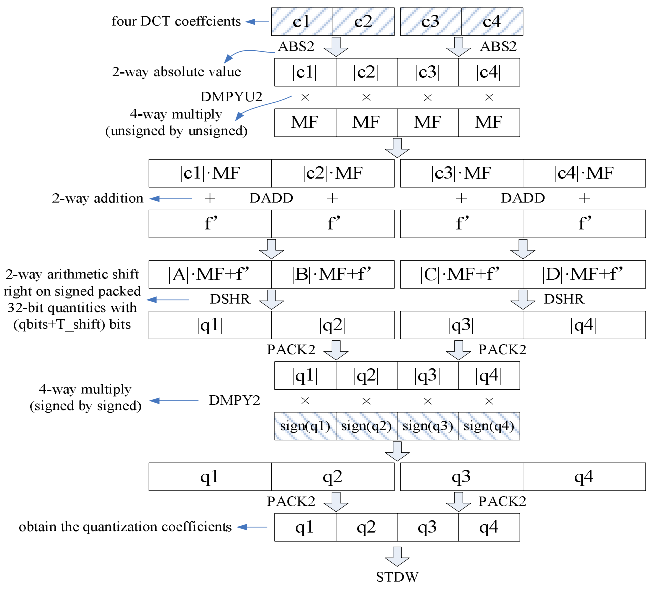

LDDW
load four DCT coefficients
-324 564 -6 0
4-way arithmetic shift right on signed packed 16-bit quantities with 15 bits
DSHR2(15)
0xFFFF 0x0000 0xFFFF 0x0000
NOT
NOT
bitwise NOT operation
0x0000 0xFFFF 0x0000 0xFFFF
4-way logical shift right on unsigned packed 16-bit quantities with 15 bits
DSHRU2(15)
0x0000 0x0001 0x0000 0x0001
bitwise OR operation
OR
OR
obtain the sign of each DCT coefficient
0xFFFF 0x0001 0xFFFF 0x0001
(-1) (1) (-1) (1)

Fig. 7. 4-way sign calculation example



four DCT coeffcients
c1 c2 c3 c4
ABS2
2-way absolute value
|c1| |c2| |c3| |c4|
DMPYU2
4-way multiply (unsigned by unsigned)
MF MF MF MF
|c1|·MF |c2|·MF |c3|·MF |c4|·MF
2-way addition
DADD
f' f' f' f'
2-way arithmetic shift right on signed packed 32-bit quantities with (qbits+T_shift) bits
|A|·MF+f' |B|·MF+f' |C|·MF+f' |D|·MF+f'
DSHR
|q1| |q2| |q3| |q4|
PACK2
4-way multiply (signed by signed)
DMPY2
sign(q1) sign(q2) sign(q3) sign(q4)
q1 q2 q3 q4
obtain the quantization coefficients
STDW

Fig. 8. SIMD optimization of quantization module

Four absolute values of the quantized coefficients will be calculated in parallel by employing SIMD, which is shown in Fig. 8. And the final quantized coefficients can be obtained by multiplying the absolute values and sign values of the quantized coefficients with DMPY2 instruction.

F. Sample Adaptive Offset

The key idea of SAO is to reduce the sample distortion by first classifying reconstructed samples into different categories, obtaining an offset for each category, and then adding the offset to each sample of the category [1]. Edge offset (EO) is based on comparison between current samples and neighboring samples. In order to keep reasonable balance between complexity and coding efficiency, EO uses four 1-D directional patterns for sample classification: horizontal, vertical, 135° diagonal and 45° diagonal, which is shown in Fig. 18. As illustrated in Fig. 9, for each EO pattern, there are five categories based on the relationship between a, b and c. In order to figure out which category the current point c belongs to, we should calculate the sign values of c-a and c-b.

Different from the results of sign bit for the quantization module in Fig. 8, there are three cases for SAO, namely 1 (greater than 0), -1 (less than 0) and 0 (equal to 0). Therefore, we redesign the computational process and give an example in Fig. 10. Four sign values are computed using parallel instructions, such as DSHR2, DSHRU2, DSUB2 and OR, whose functions can be seen clearly in Fig. 11.

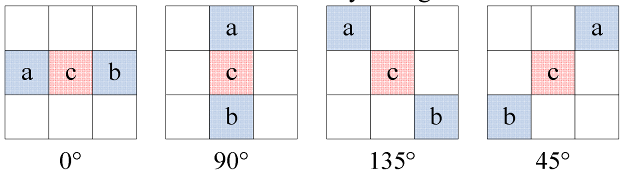


Fig. 9 Four 1-D directional patterns for EO sample classification: Horizontal (EO class=0), vertical(EO class= 0),135°Diagonal (EO class=0) and 45° diagonal(EO class=0).

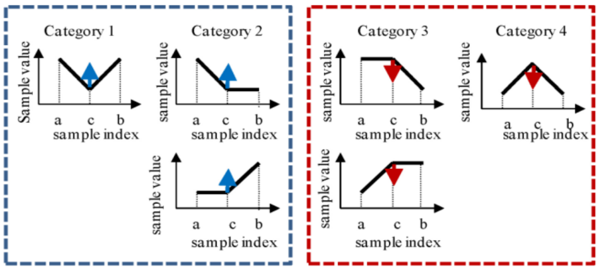
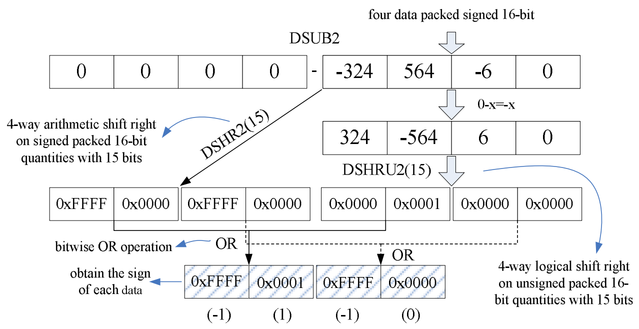
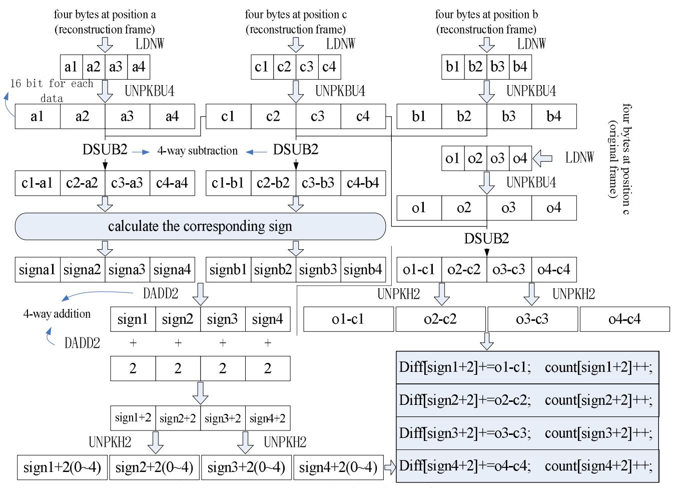


Fig. 10. EO classification



four data packed signed 16-bit
DSUB2
0 0 0 0 - -324 564 -6 0
0-x=-x
324 -564 6 0
4-way arithmetic shift right on signed packed 16-bit quantities with 15 bits
DSHR2(15)
DSHRU2(15)
0xFFFF 0x0000 0xFFFF 0x0000 0x0000 0x0001 0x0000 0x0000
bitwise OR operation
OR
obtain the sign of each data
0xFFFF 0x0001 0xFFFF 0x0000
(-1) (1) (-1) (0)
4-way logical shift right on unsigned packed 16-bit quantities with 15 bits

Fig. 11. 4-way sign calculation (3 conditions)



four bytes at position a (reconstruction frame)
four bytes at position c (reconstruction frame)
four bytes at position b (reconstruction frame)
LDNW
a1 a2 a3 a4
c1 c2 c3 c4
b1 b2 b3 b4
16 bit for each data
UNPKBU4
DSUB2
4-way subtraction
c1-a1 c2-a2 c3-a3 c4-a4
c1-b1 c2-b2 c3-b3 c4-b4
o1 o2 o3 o4
four bytes at position c (original frame)
calculate the corresponding sign
signa1 signa2 signa3 signa4
signb1 signb2 signb3 signb4
o1-c1 o2-c2 o3-c3 o4-c4
DADD2
UNPKH2
4-way addition
sign1 sign2 sign3 sign4
2 2 2 2
sign1+2 sign2+2 sign3+2 sign4+2
sign1+2(0~4) sign2+2(0~4) sign3+2(0~4) sign4+2(0~4)
Diff[sign1+2]+=o1-c1; count[sign1+2]++;
Diff[sign2+2]+=o2-c2; count[sign2+2]++;
Diff[sign3+2]+=o3-c3; count[sign3+2]++;
Diff[sign4+2]+=o4-c4; count[sign4+2]++;

Fig. 12. SIMD optimization of SAO

Fig. 12 shows the overall calculation procedure for one EO pattern. *count* records the number of each category and *diff* records the cumulative sum of the difference between the original frame and reconstruction frame. It can be easily seen that 4-position EO statistics are computed at the same time. The sign calculation module is employed to obtain the sign information of *c-a* and *c-b*. *signa*1~*signb*4 in Fig. 12 are in the range of -1 to 1. As a result, *sign*1~*sign*4 change from -2 to 2, and they are used to distinguish between different categories. Finally, the number of each category and the difference between the original frame and reconstruction frame of each category are updated, respectively.

G. Distortion Calculation

Various distortion calculations are used to perform mode or parameter decisions, and SAD (Sum of absolute difference)

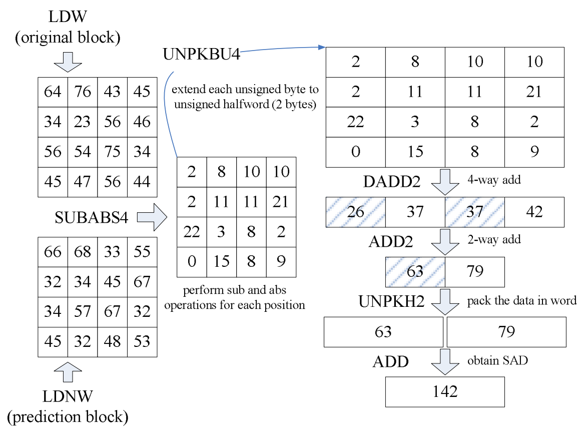


Fig. 13. SAD calculation with SIMD for a 4×4 block

and SATD (Sum of Absolute Transformed Difference) are the two most used distortion. SAD and SATD are computed as

$$SAD = \sum_{i,j} \left| Diff(i,j) \right| \tag{10}$$

$$SATD = \left(\sum_{i,j} \left| DiffT(i,j) \right|\right)/2 \tag{11}$$

Fig. 14 is an example of SAD calculation implemented with SIMD for a 4×4 block.

H. Frame Subtraction and Addition

The residual data is obtained through the subtraction of the original and the prediction data. Similarly, the reconstruction data can be achieved by adding the decoded residual data and the prediction data. These two processes can be also optimized by the linear assembly instructions. Fig. 14 shows the frame subtraction and addition implementation.

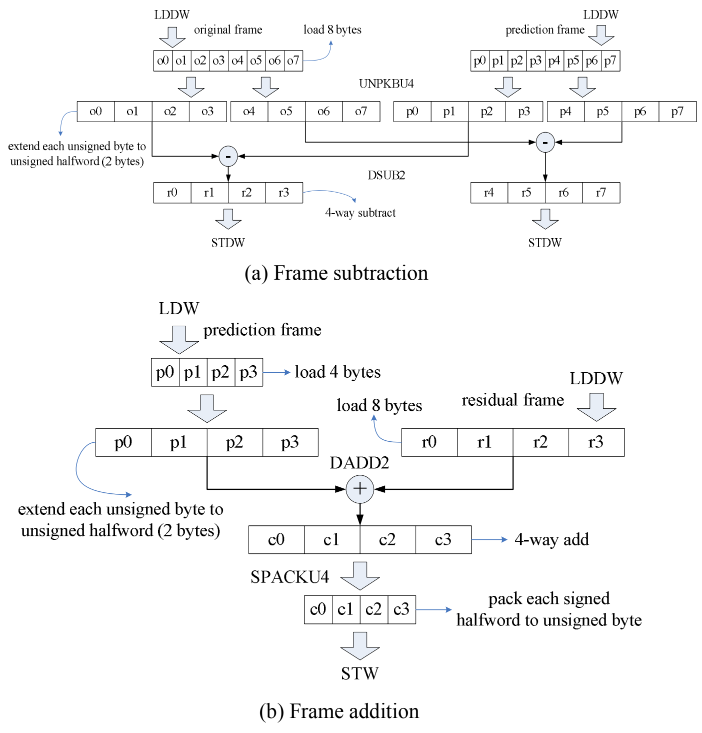


Fig. 14. Frame subtraction and addition with SIMD optimization

IV. Performance Evaluation

In this section, the proposed SIMD implementation in our HEVC encoder on TI TMS320C6678 DSP is evaluated. In order to assess the performance of our proposed SIMD optimization, we illustrate the actual cycles of each HEVC modules or three comparable schemes and speed-up ratio is computed to evaluate the optimization performance.

(1) C: the original DSP-based HEVC encoder in C;

(2)C(-o3): the DSP-based HEVC encoder with -o3 level optimization enabled to accelerate the modules, which enables SPLOOP (C66x hardware loop buffer), file-level optimization, function-level optimization and restricts high-level optimization.

(3)SIMD(-o3): the proposed DSP-based HEVC encoder with SIMD optimizations -o3 level optimization enabled.

Table I. SIMD Performance Evaluation

encoding modules	block sizes	C	C(-o3)	SIMD (-o3)	Speedup over C	Speedup over C(-o3)
Intra	4×4	339	68	16	21.19	4.25
	8×8	1387	237	54	25.69	4.39
	16×16	5595	845	188	29.76	4.49
	32×32	22459	3245	683	32.88	4.75
	64×64	89979	12621	2560	35.15	4.93
Inter (horizontal interpolatio)	8×8	8690	1239	501	17.35	2.47
	16×16	24898	2841	853	29.19	3.33
	32×32	81122	8177	2197	36.92	3.72
	64×64	288802	27343	6805	42.44	4.02
Inter (vertical interpolatio)	8×8	6271	936	303	20.70	3.09
	16×16	24239	2857	859	28.22	3.33
	32×32	95503	9752	2375	40.21	4.11
	64×64	379343	38227	7866	48.23	4.86
DCT	4×4	670	179	80	**8.38**	2.24
	8×8	2608	534	216	12.07	2.47
	16×16	16432	2568	1077	15.26	2.38
	32×32	82722	11612	5900	14.02	1.97
quant	4×4	1129	454	124	9.10	3.66
	8×8	4393	1702	363	12.10	4.69
	16×16	17443	6694	1215	14.36	5.51
	32×32	74164	26662	4064	18.25	**6.56**
SAO	64×64	50078	6919	3255	15.38	2.13
SAD	8×8	805	73	39	20.64	1.87
	16×16	2974	293	71	41.89	4.13
	32×32	11594	1201	229	50.63	5.24
	64×64	47050	4993	795	59.18	6.28
frame sub	8×8	1524	43	27	56.44	**1.59**
	16×16	5724	140	76	75.32	1.84
	32×32	22188	726	271	81.87	2.68
	64×64	87372	3648	1046	83.53	3.49
frame add	8×8	2314	114	36	64.28	3.17
	16×16	8834	409	108	81.80	3.79
	32×32	34546	1763	399	86.58	4.42
	64×64	136658	7738	1565	**87.32**	4.94

Table I shows the actual cycles of each HEVC modules for the three comparable schemes, respectively. It is clear that the original DSP-based HEVC encoder in C consumes lots of cycles and cannot be put into practice in real-time applications. With –o3 level optimization, several times or even dozens of times of speed up can be obtained. On the basis of C language with –o3 level optimization, our proposed optimization employing linear assembly language (SIMD) achieves further acceleration with several times, which benefits from parallel data processing. Especially for functions of vertical interpolation, quantization and frame addition, the speed-up ratio might reach 4.86~6.56 over C(-o3) and 87.32 over the original C-based encoder. Besides, in most cases, the larger the block size is, the higher the speed-up ratio is. This is because larger block can arrange instruction execution for more subsequent loops at the same moment. Its structure of software pipeline is more reasonable, which has higher pipeline efficiency. The speedup ratios of 16×16 and 32×32 DCT are outliers. The main reason is that the number of internal register pairs is limited to store transform matrix for large block, and it will waste lots of cycles to read the related transform coefficients. For blocks of small sizes, the transform matrix can be stored in the internal register pairs during the transform process; as a result, the speed up ratio is higher.

V. Conclusions

In this paper, we have studied the problem of fast implementation of the HEVC encoder over modern TI Digital Signal Processors (DSPs). First, we identified the most time-consuming encoding modules of the DSP-based HEVC encoder. And then the DSP SIMD instructions were exploited to improve the parallel computing capacity of these modules and thus speed up the encoder. Experimental results have demonstrated that the proposed implementations can significantly improve the encoding speed of the DSP-based HEVC encoder.

References

[1] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649-1668, Sep. 2012.

[2] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560-576, Jul. 2003.

[3] F. Bossen, B. Bross, K. Suhring, and D. Flynn, "HEVC Complexity and Implementation Analysis," *IEEE Trans.* Circuits *Syst. Video Technol.*, vol. 22, no. 12, pp. 1685-1696, Oct. 2012.

[4] R. Fan, Y. Zhang, B. Li, "Motion classification-based fast motion estimation for HEVC", *IEEE Trans. on Multimedia*. vol. 19, no. 5, pp. 893-907, May. 2017.

[5] B. Min, Ray C. C. Cheung, "A Fast CU Size Decision Algorithm for the HEVC Intra Encoder," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 5, pp. 892-896, May. 2015.

[6] Y. Zhang, H. Wang and Z. Li. "Fast Coding Unit Depth Decision Algorithm for Interframe Coding in HEVC," *IEEE Data Compression Conference*, Utah, US, Mar. 2013, pp.53-62.

[7] Y. Z. Duan, J. Sun, L. J. Yan, K. J. Chen, and Z. M. Guo, "Novel Efficient HEVC Decoding Solution on General-Purpose Processors," *IEEE Trans. on Multimedia*, vol. 16, no. 7, pp. 1915-1928, Nov. 2014.

[8] C.C. Chi, M. A.-Mesa, B. Bross, B. Juurlink, and Thomas Schierl, "SIMD Acceleration for HEVC Decoding," *IEEE Trans.* Circuits *Syst. Video Technol.*, vol. 25, no. 5, pp. 841-855, May 2015.

[9] D. F. de Souza;, A. Ilic, N. Roma; L. Sousa, "GHEVC: An Efficient HEVC Decoder for Graphics Processing Units," *IEEE Trans. on Multimedia,* vol. 19, no. 3, pp. 459-474, Mar. 2017.

[10] W. Xiao, B. Li, J. Xu, G. Shi, F. Wu, "HEVC Encoding Optimization Using Multicore CPUs and GPUs," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 11, pp. 1830 – 1843, Nov. 2015.

[11] M. Abeydeera, M. Karunaratne, G. Karunaratne, K. D. Silva, and A.Pasqual, "4K Real Time HEVC Decoder on FPGA," *IEEE Trans.* Circuits *Syst. Video Technol.*, vol. 26, no. 1, pp. 236-249, Jan. 2016.

[12] F. Pescador, M. Chavarrias, M. J. Garrido, E. Juarez, and C. Sanz, "Complexity analysis of an HEVC decoder based on a digital signal processor," *IEEE Trans. on Consumer Electronics*, vol. 59, no. 2, pp. 391-399, May 2013.

[13] M. Chavarrías, F. Pescador, M. J. Garrido, et al., "A multicore DSP HEVC decoder using an actor-based dataflow model and OpenMP," *IEEE Trans. on Consumer Electronics*, vol. 61, no. 2, pp. 236-244, May 2015.

[14] Y.-J. Ahn, T.-J. Hwang, D.-G. Sim, and W.-J. Han, "Implementation of fast HEVC encoder based on SIMD and data-level parallelism," *EURASIP Journal on Image and Video Processing*, pp. 1-19, Dec. 2014.

[15] S. Radicke, J. Hahn; Q. Wang; C. Grecos, "A Parallel HEVC Intra Prediction Algorithm for Heterogeneous CPU+GPU Platforms" *IEEE Trans. on Broadcasting*, vol. 62. No. 1, pp.103-119, Jan. 2016.

[16] H. Igarashi; F. Takano; T. Moriyoshi, "Highly parallel transformation and quantization for HEVC encoder on GPUs," *2016 Visual Communications and Image Processing*, pp. 1-4, 2016.

[17] H. Kibeya; N. Bahri; M. Ali Ben Ayed; N. Masmoudi, "SAD and SSE implementation for HEVC encoder on DSP TMS320C6678," *Int'l Image Processing, Applications and Systems (IPAS)*, pp.1-6, 2016.

[18] "TMS320C6678 Multicore Fixed and Floating-Point Digital Signal Processor (Rev. E)," SPRS691E, Mar. 2014. [Online]. Available: http://www.ti.com/lit/ds/symlink/tms320c6678.pdf.

[19] "HM Reference Software 16.0 [Online]." Available: http://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware.

[20] F. Bossen, "Common test conditions and software reference configurations," JCTVC-I1100, Joint Collaborative Team on Video Coding meeting, Geneva, May 2012.

[21] B. Ramesh, A. Bhardwaj, J. Richardson, A. D. George, H. Lam, "Optimization and evaluation of image- and signal-processing kernels on the TI C6678 multi-core DSP", 2014 IEEE High Performance Extreme Computing Conference, Sept. 2014.

[22] "C66x CPU and Instruction Set Reference Guide," SPRUGH7, Nov. 2010. [Online]. Available: http://www.ti.com/lit/ug/sprugh7/sprugh7.pdf.