Acceleration of HEVC Transform and Inverse Transform on ARM NEON Platform

Hui Yong, Ronggang Wang, Wenmin Wang, Zhenyu Wang, Shengfu Dong, Bingjie Han and Wen Gao
School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School, Shenzhen, China
yonghuiyh@gmail.com, {rgwang, wangwm, wangzhenyu, dongsf}@pkusz.edu.cn, hbj515@sina.com, wgao@pku.edu.cn

***Abstract*—HEVC (High Efficiency Video Coding) has recently been published as the next generation video coding standard. Compared with previous standards, the coding efficiency is greatly improved at the cost of much higher codec complexity. On the other hand, ARM with SIMD (Single Instruction Multiple Data) instructions is widely deployed on mobile platform, which makes it feasible to accelerate multimedia applications by software. However, the accelerations are not straight forward, and the modules have to be carefully redesigned to utilize SIMD instructions efficiently. Transform and inverse transform modules are frequently called in HEVC codec. In this paper, we propose an acceleration method of the transform and inverse transform by utilizing the data level parallelism on NEON platform. Experiment results show that the speed of these modules has been increased considerably.**

Keywords—simd; neon; hevc; dct; idct

I. INTRODUCTION

With the development of multimedia application for mobile devices such as smart phone and tablet PC, the screen resolution for these devices has been increased substantially. People have more chances to enjoy high-definition video through the Internet and the demand for 4k×2k Quad Full High Definition (QFHD) resolution is becoming stronger. What's more, the transmission needs of high definition video are imposing severe challenges on network bandwidth. A desire for next generation video codecs with higher compression has drawn increased attention and the main goal of the HEVC (High Efficiency Video Coding), developed by the Joint Collaborative Team on Video Coding (JCT-VC), is to satisfy this demand. HEVC is issued recently, and a number of new coding tools are adopted. Transform with various block sizes is one of them [1].

The core of the coding layer in HEVC is the CTU (coding tree unit), which consists of a luma CTB (coding tree block) and the corresponding chroma CTBs and syntax elements. The basic unit for compression, termed CU (coding unit), is a 2N×2N square block, and one CU can be recursively split into four smaller CUs until the predefined minimum size is reached. One luma CB and ordinarily two chroma CBs, together with associated syntax, form a CU. Each CU contains one or multiple PUs (prediction units). And for residual coding, a CB can be recursively partitioned into TBs (transform blocks) [2].

The residual signal data, which would be processed by transform parts, is the difference between the original block and its prediction. Most operations performed on each element are similar, so they can be efficiently performed on a group of pixels in the packed form by using SIMD (Single Instruction Multiple Data) instructions, which can process multiple packed data in parallel with a single operation.

There are several related works on acceleration of H.264 [3] and VC-1 [4] decoder by utilizing SIMD capability of ARM NEON platform. In our previous work, we use the similar strategy to accelerate AVS decoder [5]. The transform tool in HEVC is more complicated than H.264 and AVS etc. Besides of 4×4 and 8×8, 16×16 and 32×32 block size TBs are supported in HEVC.

Hence we will focus on the acceleration of transform and inverse transform modules in HEVC by using ARM Cortex-Ax architecture and NEON SIMD engine efficiently. The rest of the paper is organized in five sections. Section II describes the Cortex™-A and NEON SIMD instructions. Section III gives a brief introduction of transform process in HEVC. Section IV presents our proposed acceleration method of transform in detail. Section V provides the acceleration results. At last, conclusion is drawn in Section VI.

II. ARM NEON INSTRUCTIONS

The ARM SIMD media extensions were introduced with the ARMv6 architecture, beginning with ARM1136. NEON technology is firstly introduced in the ARMv7 architecture and is available with ARM Cortex-A class processor [6], which has been accepted as one of the most remarkable microprocessors for multimedia application. Compared with the 32-bit wide SIMD in the ARMv6 architecture, NEON technology, designed to provide flexible and powerful acceleration for consumer multimedia applications, builds on the concept of SIMD with a dedicated module to provide 128-bit wide vector operations. It has 32 64-bit D registers (D0-D31) and 16 128-bit Q registers (Q0-Q15). One Q register is composed of two consecutive D registers as shown in Fig.1.

Fig.1. One Q register is composed of two consecutive D registers.

NEON supports multiple instructions such as addition, multiplication, rounding, shifting, and saturation etc. NEON instructions perform "Packed SIMD" processing: Registers are considered as vectors of elements of the same data type; Data types can be: signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit and

single precision floating point as shown in Fig.2; Instructions perform the same operation in all lanes as shown in Fig.3.

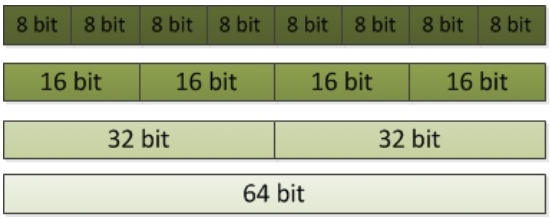

Fig.2. Data type of D register.

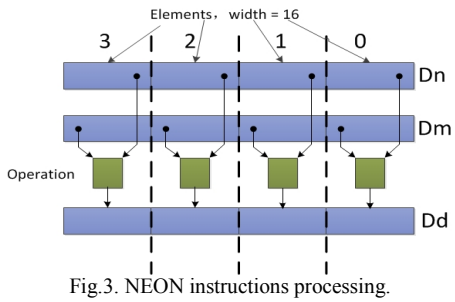

Fig.3. NEON instructions processing.

III. TRANSFORM AND INVERSE TRANSFORM IN HEVC

Similar with prior video codec standards, HEVC uses transform coding of the prediction residues to eliminate spatial redundancy.

In transform part, a CB can be recursively partitioned into TBs or not. The largest size of a TB is equal to the CB size. Unlike the asymmetric partition implemented on CBs into PBs, only square partitioning is specified, which means a block, if should be slit, can be recursively split into four quadrants. The TB sizes can be 4×4, 8×8, 16×16 and 32×32. To maximize the potential coding efficiency benefits, a TB can span across multiple PBs for inter-predicted CUs. And the splitting of the minimum TB size of 4×4 is not supported in the HEVC [7]. The partition of a CTB into CBs and CBs into TBs is shown in Fig.4.

The matrix for the length-16 transform is given as [8]:

$$H = \begin{bmatrix}
64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\
90 & 87 & 80 & 70 & 57 & 43 & 25 & 9 & -9 & -25 & -43 & -57 & -70 & -80 & -87 & 90 \\
89 & 75 & 50 & 18 & -18 & -50 & -75 & -89 & -89 & -75 & -50 & -18 & 18 & 50 & 75 & 89 \\
87 & 57 & 9 & -43 & -80 & -90 & -70 & -25 & 25 & 70 & 90 & 80 & 43 & -9 & -57 & -87 \\
83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 & 83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 \\
80 & 9 & -70 & -87 & -25 & 57 & 90 & 43 & -43 & -90 & -57 & 25 & 87 & 70 & -9 & -80 \\
75 & -18 & -89 & -50 & 50 & 89 & 18 & -75 & -75 & 18 & 89 & 50 & -50 & -89 & -18 & 75 \\
70 & -43 & -87 & 9 & 90 & 25 & -80 & -57 & 57 & 80 & -25 & -90 & -9 & 87 & 43 & -70 \\
64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 \\
57 & -80 & -25 & 90 & -9 & -87 & 43 & 70 & -70 & -43 & 87 & 9 & -90 & 25 & 80 & -57 \\
50 & -89 & 18 & 75 & -75 & -18 & 89 & -50 & -50 & 89 & -18 & -75 & 75 & 18 & -89 & 50 \\
43 & -90 & 57 & 25 & -87 & 70 & 9 & -80 & 80 & -9 & -70 & 87 & -25 & -57 & 90 & -43 \\
36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 & 36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 \\
25 & -70 & 90 & -80 & 43 & 9 & -57 & 87 & -87 & 57 & -9 & -43 & 80 & -90 & 70 & -25 \\
18 & -50 & 75 & -89 & 89 & -75 & 50 & -18 & -18 & 50 & -75 & 89 & -89 & 75 & -50 & 18 \\
9 & -25 & 43 & -57 & 70 & -80 & 87 & -90 & 90 & -87 & 80 & -70 & 57 & -43 & 25 & -9
\end{bmatrix}$$

The matrices for blocks with smaller sizes can be derived from this matrix by using the first entries in the corresponding rows. For example, the matrix for size 4×4 is derived by using the first 4 entries in the rows 0, 4, 8 and 12 [7].

Two-dimensional DCT (discrete cosine transforms) are calculated by applying one-dimensional transforms separately in the horizontal and vertical directions. Large transform sizes, i.e. 16×16 and 32×32, are involved to improve compression efficiency. Moreover, an alternative integer transform derived from a DST (discrete sine transform), is applied to the intra luma residual blocks with size 4×4 [7].

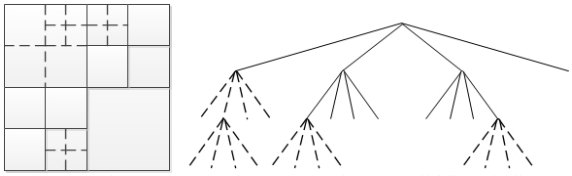

Fig.4. Partition of a CTB into CBs and TBs. Solid lines indicate CB boundaries and dotted lines indicate TB boundaries.

IV. ACCLERATION OF TRANSFORM AND INVERSE TRANSFORM WITH NEON

In this part, we will propose an acceleration method of transform and inverse transform with NEON SIMD architecture. As instructed in previous parts, NEON technology supports instructions up to 128-bit width. Since the prediction residual coefficients to be transformed are with 16-bit width, 4 coefficients stored in one 64-bit D register can be processed in parallel with a single instruction. In order to avoid overflowing, 32-bit space is required to store the temporary value. Hence the 128-bit Q register is required to store four intermediate variables. Thus most instructions in these parts apply to 4 coefficients at most in parallel.

Herein we take the 16×16 TU as an example to illustrate the acceleration method as follows. High acceleration efficiency should be achieved by minimizing memory access and arithmetic operations, especially multiplication.

For 16×16 transform, in order to minimize arithmetic operations, partial butterfly computation is applied to obtain the intermediate variables as shown in Fig.5. One dimensional transform is applied separately in horizontal and vertical direction almost in the same way, except for different shift bits.

To perform this partial butterfly computation with SIMD instructions, we need to load the residual coefficients (i.e. src(0)-src(15) described in Fig.5) of the first 4 rows into 16 D registers at first. Each D register contains 4 adjacent coefficients in the same row and since the coefficients in one column are operated in the same way, we need to arrange these data by VTRN [6] instruction. Since the supported bit width is 128 at most, 4 coefficients can be loaded in parallel. Hence 4 rows of results can be achieved in parallel and the same process is iterated for 4 times to obtain all the 16 columns. Fig.6 shows the way the instruction VTRN works. Now the coefficients of 16 columns in first 4 rows are kept in 16 D registers (i.e. D16-D31), and they are pushed onto the stack. Since the extended registers provided by NEON architecture are limited, some of the prediction residual coefficients and the intermediate variables are pushed onto the stack to save registers for other instructions. Memory access is a time-consuming process, so a good compromise should be made between storing data in

stack and recalculation to minimize the time consumption. Herein we can perform all the multiplication, addition, rounding and saturation instructions in parallel using SIMD operations. The intermediate variables (i.e. E, EE, EO, EEE, EEO) can be obtained through instructions such as VADDL and VSUB [6].

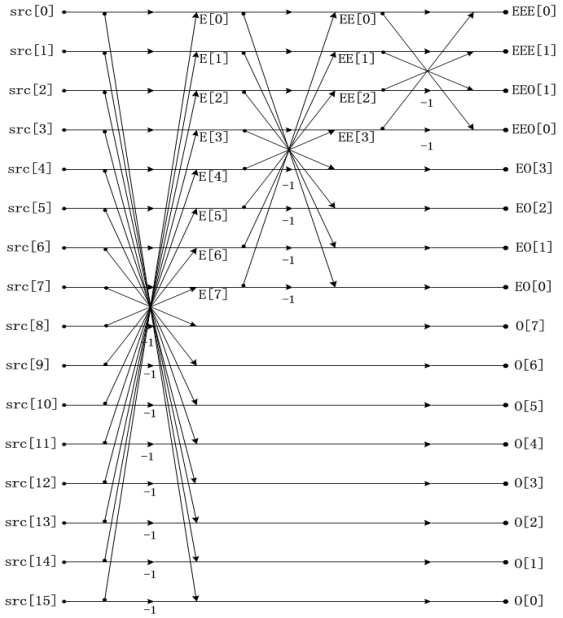

Fig.5. Process of partial butterfly computation of DCT size=16×16.

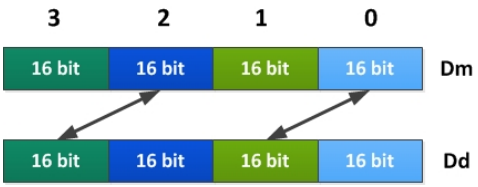

Fig.6. D registers VTRN.16 operation

After partial butterfly transform, destination results of different rows are achieved differently. The computation for results in even rows with index 0, 4, 8 and 12 are in the same way, even rows with index 2, 6, 10 and 14 are in the other same way, and the odd rows are calculated in the third same way. Hence we will introduce the process in three parts.

For even rows with index 0, 4, 8 and 12, Fig.7 shows the method to calculate them.

Fig.7. Computation process for results in rows 0, 4, 8 and 12.

For even rows with index 2, 6, 10 and 14, Fig.8 shows the method to calculate them. Intermediate variables (i.e. EO) should be popped off the stack at first and stored in 4 quadword registers Q0, Q1, Q2 and Q3. Then multiplication and addition instructions could be performed in parallel to get the results.

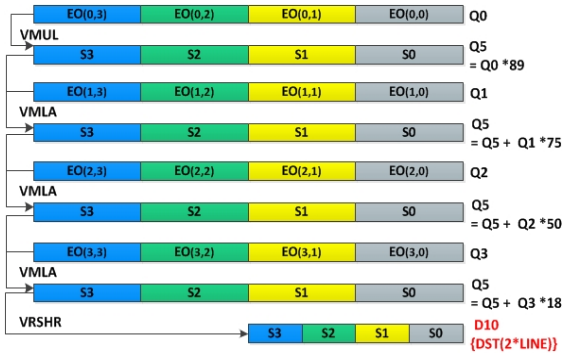

Fig.8. Computation process for results in rows 0, 4, 8 and 12.

For odd numbered rows, Fig.9 shows the method to calculate them. We need to pop the residual coefficients off the stack at the beginning and store the intermediate values with 32-bit width in 8 quadword registers (i.e. Q0-Q7). It should be noticed that in NEON technology registers capable of multiplication on 16-bit width scalar quantity are restricted to the first 8 D registers (i.e. D0-D7) and registers capable of multiplication on 32-bit width scalar quantity are restricted to the first 16 D registers (i.e. D0-D15). Taking the parameters of the transform matrix into account, the Q registers capable of multiplication are not enough to calculate the results straightly.

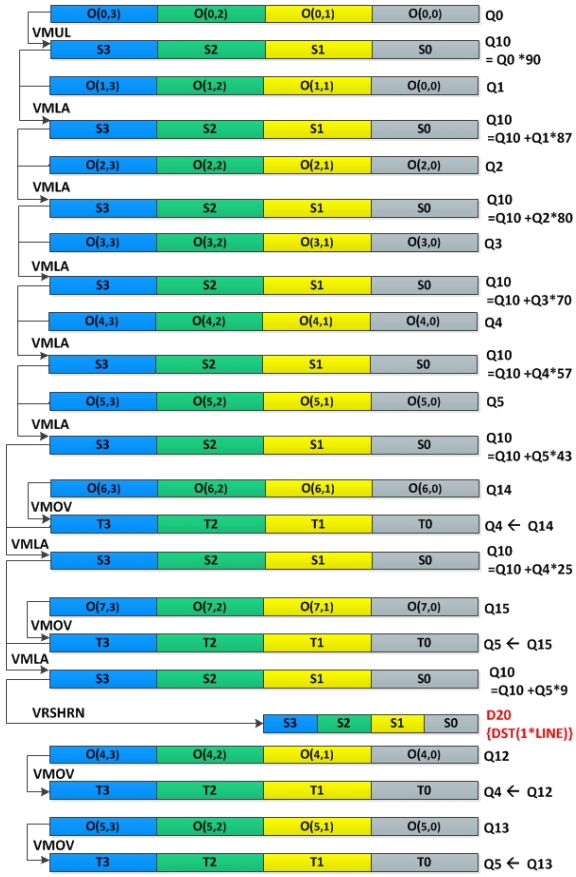

Fig.9. Computation process for results in odd numbered rows.

Many NEON data processing instructions are available in normal, wide, narrow, long and saturating variants. Both the source address and the destination address are 16-bit pointers and the intermediate data are stored in 32-bit, so VRSHRN [6] instruction is used to make sure that results are right shifted, rounded and stored in 16-bit. The first R in VRSHRN instruction means round.

Now results of the first 4 columns have been obtained and the similar process needs to be iterated for another 3 times to get the remaining results (i.e. results of columns from 4 to 15).

IDCT of 16×16 TU is just the inverse process of DCT. The last results of IDCT might be out of the range from -32768 to 32767 after the right shift operation, so we need VQRSHRN [6] instruction to saturate the results to the appropriate range. The Q in VQRSHRN instruction means saturation.

For 32×32 TUs, registers are not enough to load all the coefficients in first four rows and store the intermediate results of the first addition operation. So registers arrangement should be made much more elaborately and results should be stored in the accurate location of memory instantly to save registers for the following instructions.

V. ACCELERATION RESULTS

The transform modules of the HEVC C++ codes are used as the base for the acceleration. The HEVC reference software HM10.0 [9] is implanted to the mobile phone operating system of Android. The Android VM allows the application's source code to call methods implemented in native code through the JNI (Java Native Interface). Android NDK r8e is used to compile the C++ codes in HEVC reference software and ARM NEON codes of the accelerated modules. Eclipse Juno is used to compile the Java codes. LG P920 mobile phone, which has a Cortex-A9 MPCore processor with NEON running at 1GHz, is used as the platform to test our acceleration methods. We conduct experiments of transform parts in HM10.0 encoder under the common test conditions defined in JCTVC-F900 [10]. The configuration of low delay P high efficiency (LPHE) is used.

Table I shows the comparison of execution time of five different functions in HM10.0 encoder and our proposed method using NEON. The sequence Foreman, which consists of 150 frames, is used in this experiment and the results indicate that at least 2 times speedup has been achieved. Then we test performance of the whole transform module on five sequences respectively and the seconds consumed are given in Table II.

The experiments of inverse transform module are conducted in HM10.0 decoder. Since the operation CLIP3 in HM10.0, consists of two *if* statements, which is time consuming and the corresponding instruction in NEON is much more time saving, the time consumed on CLIP3 is not counted in. Table III shows the actual execution seconds consumed by different functions of inverse transform on sequence of Foreman and Table IV shows the time consumption of the whole inverse transform module on five sequences respectively. The frame resolutions of these sequences BUS, INTERVIEW, HARBOUR, NEWS, FORMAN, RUSHHOUR and SUNFLOWER are 352*288, 704*576, 1280*720, 704*480, 352*288, 1920*1080, 1920*1080, respectively. Because of the high efficiency of HEVC, we give the ratio of different sizes in the last two 1080p sequences (i.e. RUSHHOUR and SUNFLOWER) in Table V and Table VI.

TABLE I. COMPARISON OF EXECUTION TIME OF FIVE DIFFERENT TRANSFORM FUNCTIONS IN SEQUENCE FOREMAN

Function	Time (sec)		Speedup
	HM10.0	HM 10.0 With NEON	
Fast DST	1.245004	0.394097	3.15913
4x4 DCT	8.929107	3.988412	2.23876
8x8 DCT	14.323542	5.154061	2.77907
16x16 DCT	21.742545	8.742509	2.48699
32x32 DCT	22.913228	10.073966	2.27449

TABLE II. COMPARISON OF EXECUTION TIME OF TRANSFORM MODULE IN FIVE DIFFERENT SEQUENCES

Sequence	Time (sec)		Speedup
	HM 10.0	HM 10.0 With NEON	
BUS	81.510647	35.159296	2.31832
INTERVIEW	292.979588	127.168451	2.30387
HARBOUR	332.373747	142.639278	2.33017
NEWS	242.572488	105.409731	2.30123
FOREMAN	71.637034	30.946671	2.31485
RUSHHOUR	811.564390	361.2785579	2.24636
SUNFLOWER	573.258879	256.968929	2.23084

TABLE III. COMPARISON OF EXECUTION TIME OF FIVE DIFFERENT INVERSE TRANSFORM FUNCTIONS IN SEQUENCE FOREMAN

Function	Time (sec)		Speedup
	HM10.0	HM10.0 with NEON	
Fast IDST	0.031435	0.010615	2.96138
4x4 IDCT	0.037216	0.011949	3.11457
8x8 IDCT	0.096371	0.016147	5.96835
16x16 IDCT	0.114515	0.018545	6.17498
32x32 IDCT	0.132919	0.022302	5.95996

TABLE IV. COMPARISON OF EXECUTION TIME OF INVERSE TRANSFORM MODULE IN FIVE DIFFERENT SEQUENCES

Sequence	Time (sec)		Speedup
	HM 10.0	HM 10.0 With NEON	
BUS	0.474526	0.107917	4.39714
INTERVIEW	2.205797	0.446847	4.93636
HARBOUR	2.452085	0.463453	5.29090
NEWS	1.969601	0.384094	5.12791
FOREMAN	0.562166	0.123711	4.54418
RUSHHOUR	7.406415	1.319671	5.61231
SUNFLOWER	5.742934	1.029635	5.57764

Compared to forward transform, inverse transform with NEON achieves much higher speedup, especially the IDCT with sizes 8×8, 16×16 and 32×32. We choose the DCT 8×8 and IDCT 8×8 to investigate this matter further and as is shown in Table I and Table III, the speedup of IDCT 8×8 is about 2.15 times of DCT. In this comparison experiment, DCT and IDCT

of both C++ and NEON codes are supplied with same data input. On one hand, time consumed by IDCT 8×8 in C++ is about 1.26 times of DCT 8×8 in C++. On the other hand, time consumed by IDCT 8×8 in SIMD is about 0.57 times of DCT 8×8 in SIMD. Through the comparison of SIMD instructions used in DCT 8×8 and IDCT 8×8, we find all the instructions relevant to time-consuming multiplication (i.e. VMUL, VMLA and VMLS [6]) in IDCT 8×8 apply to scalar quantities with 16-bit width, while it is 32-bit width in DCT. And through experiment, we find the time consumption of the instruction VMUL.S32 (S32 means 32 bit signed integer) is 2 or 3 times of VMULL.S16. This should be one main reason for higher speedup of IDCT compared to DCT for large block transforms.

TABLE V. THE RATIO OF DEFFERENT SIZES OF TRANSFORM AND INVERSE TRANSFORM PARTS IN SEQUENCES RUSHHOUR

The ratio of different sizes in transform		The ratio of different sizes in inverse transform	
FastForwardDst	0.108388	FastInverseDst	0.040795
4×4	0.626433	Inverse 4×4	0.199058
8×8	0.204715	Inverse 8×8	0.339196
16×16	0.052266	Inverse 16×16	0.308532
32×32	0.008196	Inverse 32×32	0.112416

TABLE VI. THE RATIO OF DEFFERENT SIZES OF TRANSFORM AND INVERSE TRANSFORM PARTS IN SEQUENCES SUNFLOWER

The ratio of different sizes in transform		The ratio of different sizes in inverse transform	
FastForwardDst	0.118217	FastInverseDst	0.069038
4×4	0.640180	Inverse 4×4	0.235510
8×8	0.191152	Inverse 8×8	0.320238
16×16	0.043985	Inverse 16×16	0.283816
32×32	0.006463	Inverse 32×32	0.091397

VI. CONCLUSIONS

The acceleration of transform and inverse transform modules in HEVC using NEON SIMD instructions is proposed and implemented on ARM processor. According to the results, the transform module using NEON SIMD instructions is about 2.3 times faster and the inverse transform module has higher speedup. The similar acceleration strategy can be used in other modules in HEVC. It is expected that the speed of the HEVC codec will be increased considerably.

ACKNOWLEDGMENT

This work was partly supported by the grant of Key Projects in the National Science & Technology Pillar Program 2011BAH08B03, and Shenzhen Basic Research Program of JC201104210117A, JC201105170732A, and JCYJ2012061450301623.

REFERENCES

[1] B. Bross, W.J. Han, J.R. Ohm, G. J. Sullivan, Y.K. Wang, T. Wiegand, “High Efficiency Video Coding (HEVC) text specification draft 10 (for FDIS & Last Call)”, JCTVC-L1003_v34, Geneva, CH, Jan. 2013.

[2] M. Winken, P. Helle, D. Marpe, H. Schwarz, T. Wiegand, “Transform coding in the hevc test model”, 18th IEEE International Conference on Image Processing, vol., no., pp.3693,3696, 11-14 Sept. 2011.

[3] C. Pujara, A. Modi, G. Sandeep, S. Inamdar, D. Kolavil, and V .Tholath, "H.264 Video Decoder Optimization on ARM Cortex-A8 with NEON," India Conference (INDICON), 2009 Annual IEEE , vol., no., pp.1-4, 18-20 Dec. 2009.

[4] C. Pujara, A. Modi, G. Sandeep, S. Inamdar, D. Kolavil, and V .Tholath, "VC-1 video decoder optimization on ARM Cortex-A8 with NEON," Communications (NCC), 2010 National Conference on , vol., no., pp.1-5, 29-31 Jan. 2010.

[5] J. Wan, R.G. Wang, H. Lv, L. Zhang, W.M. Wang, C.C. Gu, Q.Z. Zheng, and W. Gao, "AVS video decoding acceleration on ARM Cortex-A with NEON," Signal Processing, Communication and Computing (ICSPCC), 2012 IEEE International Conference on , vol., no., pp.290-294, 12-15 Aug. 2012.

[6] ARM Corp., RealView® Compilation Tools Version 4.0 Assembler Guide, DUI 0204J, Dec. 2010.

[7] 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. 1648–1667, Dec. 2012.

[8] A. Fuldseth, G. Bjøntegaard, M. Sadafale, M. Budagavi, “CE10: Core transform design for HEVC”, JCTVC-F446, Torino, IT, Jul, 2011.

[9] https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/

[10] F. Bossen, “Common test conditions and software reference configurations”, JCTVC-F900, Torino, Italy, Jul. 2011.