Performance Comparison of SIMD-based HEVC Decoders on Mobile Processor

Nguyen Van Dien, Eun-Seok Ryu
Department of Computer Engineering
Gachon University, Seongnam, Korea

***Abstract*—This paper implements SIMD mechanism on ARM-based mobile processor for HEVC Decoders. The experimental results evaluating the performance of HEVC Decoders with/without SIMD are presented. In summary, the decoding rate with SIMD is 4~5 times higher than decoding rate without SIMD. The results are up to 28 frames/s on multi-core ARM64 architecture with full HD video. The experiments also show the decoding performance comparison between the well-known open-source HEVC decodes such as Libde265 and openHEVCDecoder. In general, Libde265 achieved better decoding speedup than openHEVCDecoder in case of tile-based parallel processing. But it was less efficient in case of single threaded decoding because of the thread allocation on asymmetric multi-core processors. All experiments are conducted on Samsung Galaxy S6 with arm 64-bit architecture.**

Keywords—Performance comparison; Parallel processing; HEVC; SIMD; NEON; Libde265; OpenHEVC

I. INTRODUCTION

In 2013, the new video codec standard High Efficiency Video Coding (HEVC) was released by the International Telecommunication Union. HEVC claims to be about 50 percent more efficiency than the current H.264/MPEG-4 standard and reduce data loss [1]. However, the complexity of the algorithm and data structure of HEVC is several times than H.264/AVC[2]. That means the HEVC based video decoder requires more computing resource/power than its predecessor. It is a huge challenge for mobile device where battery life is the biggest factor for consideration. Thus, hardware and software optimizations are considered to overcome these issues. For example, using graphics processing unit (GPU) and Digital Signal Processor (DSP) for encoding/decoding to speed up the processing rate. Specially, software optimization also can contribute significantly in term of upgrading processing rate by using single instruction multiple data (SIMD) which supports data-level parallel processing mechanism. There are many experiments to evaluate the SIMD performance on HEVC[3], SIMD with HEVC deblocking filer[4]. Overall, these experiments showed that SIMD helps in reducing processing time. To examine the effect of SIMD-based HEVC decoder on mobile processor, this paper performs many experiments with/without SIMD instructions by using *openHEVCDecoder* and *Libde265* which are the most well-known open-source decoder for the HEVC. Also, this paper compares the efficiency of those software on mobile architecture with focusing on evaluating frame decoding rates. The rest of this paper is organized as follows. In section II, it summarizes the parallel processing mechanisms at data-level and tile-level in HEVC as well as the background of tile allocation with multi-core processors. In section III, it analyzes the SIMD-based HEVC decoder. Next in section IV, experimental environment and analyzing results are presented. In section V, it shows the future works and conclusion.

II. RELATED WORKS

A. ARM SIMD Instructions

SIMD has first been introduced for performing the same operation on multiple data points simultaneously. Hence, machine exploits data level parallelism, but not concurrency because there are many (parallel) computations but only a single instruction (process) at a given moment. SIMD have been glued widely in all architectures for accelerating multimedia and signal processing algorithm such as video encoding/decoding, gaming, 2D/3D graphics. On ARM architecture, SIMD is called NEON. It was firstly released in ARMv7 family. NEON owns its independent pipeline and register files which are built on concept of SIMD with dedicating module to provide 128-bit wide vector. NEON instructions perform "Packed SIMD" processing in which each register is considered as vectors of the same data like signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit and single precision floating point. The number of element inputs will be depended on the specified register size. For instance, if an operation on 16-bit integer elements stored on 128-bit register then there are 8 lanes simultaneously as Fig.1 This mean that during execution of one instruction the same operation will occur on 8 lanes in parallel.

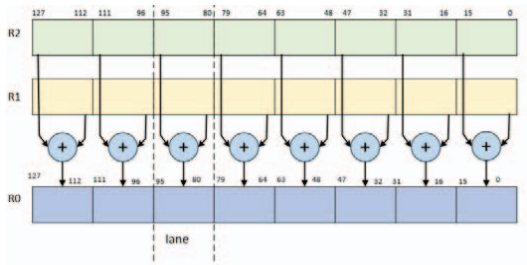


Fig. 1. 8-way 16-bit unsigned integer add operation

B. *Tile-based HEVC Decoder for Mobile Processor*

In HEVC, each picture can be partitioned into rectangular regions called tile. The main purpose of tiles is to enable the use of parallel processing architectures for encoding and decoding. All tiles within a picture are independent from each other except for potential dependencies regarding cross-tile border in-loop filtering. Each tile contains a rectangular arranged group of Coding Tree Units (CTUs) that may have dependencies on CTUs of other tiles. HEVC supports tile-level parallel processing by using multiple threads at encoder or decoder. Thus, this approach can scale up to multi-core architectures such as big/little asymmetric multicore processors in mobile device. In which, each core will be assigned to decode the specified tile [5][6] as shown in Fig 2.

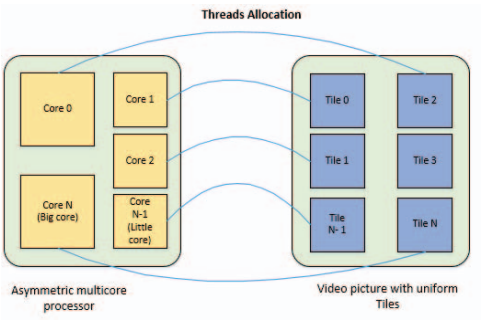


Fig. 2. Tile-based HEVC decoding on asymmetric multi-core processors.

III. HEVC DECODER BASED ON SIMD

This section describes the usage of SIMD on HEVC decoder. In HEVC decoding architecture as shown in Fig. 3, SIMD can be applied to all processing steps except for bitstream parsing. This includes inter/intra prediction, inverse transform (IT), In-loop filer. Similar with the concept of a macroblock in prior video coding standards, a CTU represents the basic processing unit in HEVC so all decoding steps will be analyzed at CTU level where performing on small intermediate buffer for performance reason.

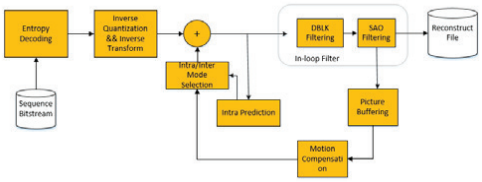


Fig. 3. HEVC Decoding Block Diagram

A. *SIMD with Inter Prediction*

Inter prediction exploits the motion data among pictures as shown in Fig. 4a. It makes use of the temporal correlation between pictures to derive a motion-compensated prediction for a block of images pixels. During the inter-prediction process, the prediction block (PB) must be created from previous pictures indexed by the reference indicates. The associated motion vectors specify a translation in these pictures with quarter-sample precision. In case, the horizontal or vertical vector component points to a fractional position, interpolation is required. HEVC specifies that the interpolation is performed using longer tap filter (7-8 tap filter for luma sub-pixels and 4-tap filter for chroma sub-pixels) compared to shorter filter in H.264/AVC (6-tap filter for luma sub-pixels and bilinear filter for chroma). In terms of memory bandwidth, utilizing longer tap filter increase the amount data that needs to be fetched from reference memory. This can be efficiently done by SIMD load instruction with multiple data pointed loaded. Similarly, the longer tap filters increase the number of arithmetic operations required to obtain the interpolated sample. For instance, to derive a horizontally interpolated sample seven to eight multiplication and additions must be performed. If also the vertical position is fractional, a second filter iteration is applied on the horizontally interpolated samples. The interpolation process is parallel for each sample and is well suited for SIMD instructions. The basic SIMD implementation is straightforward, simply multiplying and adding vector eight times either in horizontal or vertical direction can improve performance with NEON for most of 8-bit and 16-bit interpolation modes. SIMD can process 4, 8 or even 16 adjacent samples at a time depending on the block width.

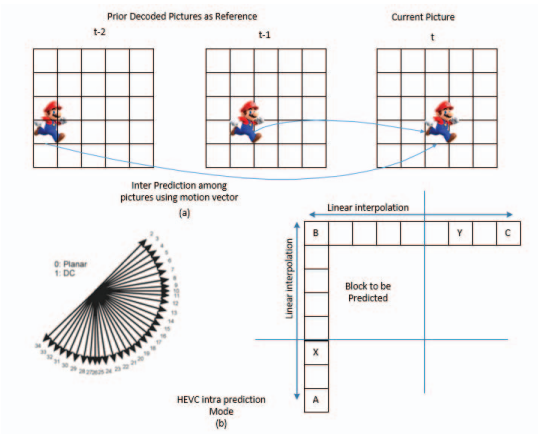


Fig. 4. HEVC Inter/Intra Prediction

B. *SIMD with Intra Prediction*

Intra Prediction in HEVC is quite similar with H.264/AVC. Different with inter prediction, samples of intra prediction are predicted from reconstructed samples of neighboring blocks. To predict different directional samples, HEVC extends up to 35 modes supported as Fig. 4b. The 33 angular modes are

designed to model different directional structures typically present in pictures. Whereas planar and DC prediction modes provide predictions for image areas with smooth and gradually changing content. Planar and DC predictions are also useful for prediction blocks with no high frequency components. For all modes, the derivation of prediction value is independent and therefore well suited for SIMD acceleration. The derivation process of the samples can be accelerated with SIMD. For example, assume that predicted sample values *p* are obtained by projecting the location of the sample *p* to the reference sample array applying the selected prediction direction and interpolating a value for sample at 1/32 sample position accuracy in angular prediction. Interpolation is performed linearly utilizing the two closest reference samples in the direction of prediction as Fig. 4b. In detail, sample prediction for vertical/horizontal mode is given by following equation:

$$P = (u*a + v*b + 16) >> 5 \quad (1)$$

Where u and v are two pre-computed constants, a and b are two pre-fetched reference sample values. The equation (1) suggests that the angular prediction requires a total of five operations including two multiplications, two additions and one shift per one predicted sample. The arithmetic operations can be efficiently using SIMD instruction by predicting multiple samples in parallel using 128-bit vector NEON

C. SIMD with Inverse Transform

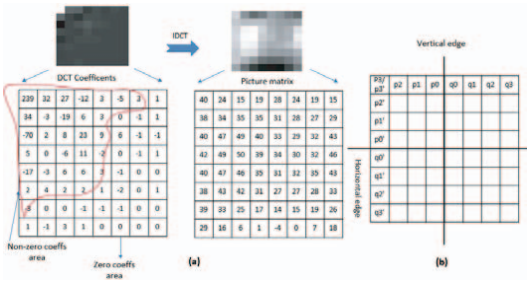


Fig. 5. Inverse Transform and Debocking Filter in HEVC

The HEVC specifies two-dimensional (2-D) core transforms that are finite precision approximations to inverse discrete cosine transform (IDCT) for all transform sizes of 4x4, 8x8, 16x16 and 32x32. Supporting up to 32x32 block size, the transform is much more computationally complex compared to previous standards. The two-dimensional (2-D) transform is performed by a serial of two one-dimensional (1-D) transform on the columns and on the rows of transform block (TB). The computation of a 1-D column transform consists of a series of a matrix-vector multiplication followed by adding/subtracting the partial results. Hence, IT has been well-suited with SIMD acceleration. The SIMD implementation of the inverse transform can be either performed inside one column or row transform or using SIMD on multiple columns/rows, or a combination of the two. Similar with the most hybrid video codecs, the coefficient scan pattern of HEVC concentrates the coefficients in the top left corner as shown in Fig. 5a. Because of the larger transforms in HEVC compared with HEVC/AVC, more columns and rows contain only zero coefficients for which the computation can be dropped as this would result back to zero. These optimizations can speed up the calculated process.

D. SIMD with In-loop Filter

The HEVC standard specifies two in-loop filters, a deblocking filter (DF) and a sample adaptive offset (SAO). At decoder side, the in-loop filters are applied after the inverse quantization and before saving the picture in the decoded picture buffer. DF is applied first then SAO used. DF and SAO attenuate discontinuities at the prediction, inverse transform and ringing artifacts respectively.

The DF can be accelerated using SIMD by considering multiple edge parts simultaneously in both vertical and horizontal edge in Fig 5b. If the computation precision of the DF is within 16-bit, eight computation lanes are available for NEON implementation with 128- bit vector in width. The SAO filter has two different modes, the edge offset (EO) mode and the band offset modes. In both modes, the entire CTB is considered and the deblocked samples are used as input. Both EO and BO uses four transmitted sample offsets for sample classification. SIMD can be applied for the entire SAO process by considering multiple samples at the same time, because each sample can be derived in parallel.

IV. HEVC DECODER PORTING AND EXPERIMENTS

A. Implementation

In this section, the efficiency of SIMD and tile-based parallel mechanism in HEVC are analyzed. For this purpose, OpenHEVC[7] Decoder and Libde265[8] that have been published on Github for developer are used. However, these versions do not support arm 64-bits core yet, thus, porting the software to mobile platforms is needed. The Fig. 6 shows the porting processes for both openHEVCDecoder and Libde265.

OpenHEVCDecoder version was introduced in ffmpeg[9] version 2.1. The following changes have been carried out to accomplish a functional decoder running on the arm 64-bits architecture.

- Modifying macros in *config.h* to control compilation
- Changing object compiling in *Android.mk*
- Porting libraries including *libavcodec, libavutil* supported for arm 64-bits architecture from ffmpeg to openHEVCDecoder

Different with openHEVCDecoder, porting Libde265 uses the customized toolchains which can be created by NDK for arm 64-bits architecture. This toolchain is invoking with the "*configure*" script of Libde265 source code that expects a cross-compiler in the CC environment variable.

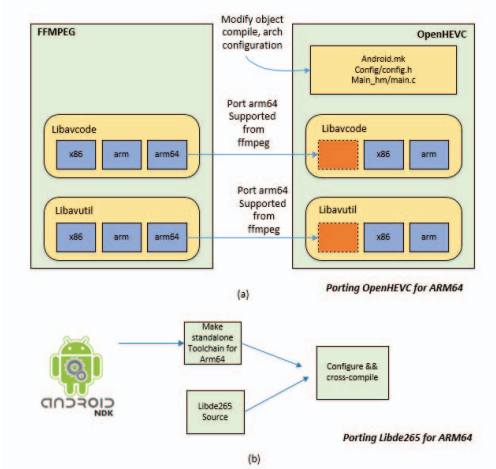


Fig. 6. openHEVC and Libde265 port to arm 64-bits architecture

B. Experiments with JCT-VC Common Test Condition

Fig. 7 demonstrates the processing model conducted in this paper. In which, the decoder has been embedded by using openHEVCDecoder and Libde265 decoder, composed of 2 modes: with SIMD and without SIMD. The stream was read from bitstream file. Then, this stream was input of both openHEVCDecoder and Libde265 to decode. During the decoding process, this paper measures the starting time and ending time of each decoder to compare speed performance. As mentioned in section II, all test cases have been implemented on Samsung Galaxy S6 with the asymmetric multicore processors with 4 big cores running at 2.1 GHz and 4 little cores running at 1.5 GHz.

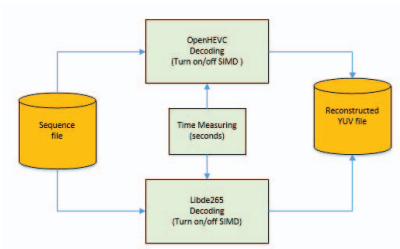


Fig. 7. Testing model used to measure the decoder performance

The same test model has done with openHEVCDecoder and Libde265, and then run with the test conditions that are defined in JVC-VC (common test condition; CTC). The test conditions can be summarized as:

- Four quantization values used: 22, 27, 32 and 37
- A total number of 7 different sequences are decoded. These sequences are divided into 2 classes (class A+ with 3840x2048 of resolution and class B with 1920x1080 of resolution) that represent different use-cases and video characteristics. This experiment uses Class A+ sequences instead of Class A for considering ultra-high-definition (UHD) service.
- Tests are conducted for two different coding structures: Random Access (RA), Low Delay with B pictures in case of single thread and Low Delay with B pictures only in case of multiple threads using tiles.

C. Performance Evaluation of HEVC Decoders

As mentioned above, the decoding efficiency of SIMD in HEVC decoders is analyzed in this section. For this purpose, all test cases were conducted with two modes: with/without SIMD. Also, performance comparison of openHEVCDecoder and Libde265 decoder was compared. Table I, II show the decoding performance of openHEVCDecoder and Libde265 respectively in case of none parallel processing using. It can be seen from these tables that using SIMD will increase the decoding speed from 4 to 5 times. And the decoding speed achieved by openHEVCDecoder is better than Libde265 in general. The highest rate was given by openHEVCDecoder is 12.5 frames/s comparing with 11 frames/s in case of libde265.

The second experiment evaluates the impact of tile-based parallel processing with SIMD. Here, this experiment uses 6 tiles bitstream of RA coding structure. The corresponding decoding efficiency gains are summarized in Table III. In summary, SIMD continue to make a different frame rate significantly in range of 4-5 times higher on average between with SIMD and without SIMD in table III. However, the comparison result between Libde265 and openHEVCDecoder is reverse. In this case, the decoding speed achieved by Libde265 is much better than openHEVCDecoder. The highest rate obtained by Libde265 and openHEVCDecoder are about 20 and 15 frames/s, respectively in case of 6 uniform tiles. The difference comes from their schedule of thread allocation which is shown in table IV. It can be seen from table IV that even using same number of processing thread, while openHEVCDecoder trend to allocate on all cores including big and little cores then Libde265 trend to make the big cores become busier than little core. Here, big core is much more powerful with frequency of 2.1 GHz comparing to 1.5 GHz of little core.

Fig. 8 demonstrates the evaluating performance of tile-based parallel processing with SIMD using Libde265 in case of higher number of tiles used. In general, the results show that the higher number of tiles, the higher throughput gain. The highest decoding speed is 28 frames/s with 12 tiles and quantization parameter equal 37. Nevertheless, the results get worse after 18 tiles used because of synchronization between threads, memory access. The more thread number, the more time for synchronization. On other hand, although tiles are completely independent from each other in term of prediction, transform and entropy coding, in-loop filter may be required to apply across tile boundaries to optionally prevent tile border artifacts. Due to this disadvantage, exploiting tile-level parallelism is only advisable when the number of tiles per frame is strictly limited.

TABLE I. DECODING RATE BY OPENHEVC WITHOUT TILES

Class	Sequence	QP=22				QP=27				QP=32				QP=37			
		RA		LDB		RA		LDB		RA		LDB		RA		LDB	
		Neon	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*
A+	People	1.47	0.41	1.38	0.36	2.21	0.53	1.85	0.48	2.31	0.63	2.21	0.6	2.63	0.68	2.59	0.67
	Traffic	2.14	0.56	1.92	0.52	2.68	0.67	2.63	0.66	3.13	0.76	3.19	0.78	3.57	0.85	3.75	0.9
	Overall	**1.81**	**0.48**	**1.65**	**0.44**	**2.44**	**0.6**	**2.24**	**0.60**	**2.72**	**0.69**	**2.70**	**0.69**	**3.10**	**0.76**	**3.17**	**0.78**
B	Basket	6.49	1.58	5.81	1.41	8.77	2.08	7.94	1.88	10.2	2.38	9.8	2.28	12.8	2.56	12.8	2.56
	BQT	5.77	1.44	5.41	1.37	9.09	2.21	8.33	2.07	11.3	2.62	10.9	2.56	12.2	2.76	12.7	2.84
	Cactus	7.58	1.98	6.94	1.75	13.2	2.84	10.6	2.63	15.1	3.23	12.5	3.11	14.7	3.52	14.7	3.45
	Kimono	8.57	1.75	6.32	1.63	10.4	2.11	9.6	1.92	10.9	2.47	11.4	2.29	11.4	2.58	10.9	2.58
	Park	7.27	2.05	7.27	1.67	10.4	2.20	8.28	2.09	10.4	2.42	10	2.42	11.4	2.67	12	2.76
	Overall	**7.14**	**1.76**	**6.35**	**1.57**	**10.4**	**2.29**	**8.96**	**2.12**	**11.6**	**2.63**	**10.9**	**2.63**	**12.5**	**2.82**	**12.6**	**2.84**

TABLE II. DECODING RATE BY LIBDE265 WITHOUT TILES

Class	Sequence	QP=22				QP=27				QP=32				QP=37			
		RA		LDB		RA		LDB		RA		LDB		RA		LDB	
		Neon	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*
A+	People	1.06	0.33	0.94	0.3	1.57	0.46	1.42	0.44	1.99	0.59	1.93	0.61	2.31	0.67	2.41	0.75
	Traffic	1.74	0.5	1.57	0.46	2.37	0.66	2.32	0.66	2.86	0.8	3.05	0.85	3.37	0.94	3.89	1.09
	Overall	**7.14**	**0.42**	**1.26**	**0.38**	**1.97**	**0.56**	**1.87**	**0.55**	**2.43**	**0.7**	**2.49**	**0.73**	**2.84**	**0.81**	**3.15**	**0.92**
B	Basket	4.97	1.38	4.4	1.29	7.22	2.0	6.45	1.91	8.71	2.48	8.57	2.37	9.92	2.63	10.4	2.98
	BQT	3.88	1.18	3.59	1.07	7.54	2.18	6.82	2.00	9.91	2.66	9.79	2.88	11.1	2.93	12.1	3.22
	Cactus	5.69	1.74	5.91	1.57	9.76	2.74	9.31	2.82	13.3	3.34	12.3	3.63	13.8	3.8	15.2	4.28
	Kimono	6.40	1.79	5.47	1.51	8.45	2.25	7.28	2.03	10.3	2.45	9.7	2.51	10.2	3.06	10.9	3.01
	Park	6.19	1.63	5.18	1.45	7.64	2.07	6.95	2.20	8.8	2.4	9.1	2.52	10.2	2.82	11.5	3.16
	Overall	**5.43**	**1.54**	**4.91**	**1.38**	**8.12**	**2.25**	**7.36**	**2.19**	**10.2**	**2.67**	**9.9**	**2.78**	**11.1**	**3.05**	**12**	**3.33**

TABLE III. DECODING RATE BY OPENHEVC & LIBDE265 WITH 6 UNIFORM TILES

Class	Sequence	QP=22				QP=27				QP=32				QP=37			
		OpenHevc		Libde265		OpenHevc		Libde265		OpenHevc		Libde265		OpenHevc		Libde265	
		Neon	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*	*Neon*	*None*
A+	People	2.68	0.64	3.17	0.93	3.19	0.87	4.29	1.35	3.49	1.07	4.27	1.68	4.05	1.15	4.94	1.99
	Traffic	3.41	0.95	4.52	1.37	4.55	1.11	5.87	1.88	4.35	1.21	6.57	2.14	4.76	1.31	7.6	2.45
	Overall	**3.04**	**0.8**	**3.85**	**1.15**	**3.87**	**0.99**	**5.08**	**1.62**	**3.92**	**1.14**	**5.42**	**1.91**	**4.41**	**1.23**	**6.27**	**2.22**
B	Basket	8.62	2.73	11.6	3.58	11.1	3.6	15.9	5.04	12.8	4.13	15.7	5.87	14.3	4.31	20.2	8.58
	BQT	11.5	2.34	10.9	2.99	11.5	3.77	16	5	14.3	4.38	18.7	6.57	15	4.58	19.5	9.37
	Cactus	9.62	3.09	13.3	4.35	12.8	4.07	18.5	6.2	13.9	5.49	21.3	8.83	16.7	4.67	22.7	11
	Kimono	9.6	3.24	13.5	4.42	11.4	3.53	15.5	4.04	14.1	4.21	17.7	6.26	14.1	4.36	19.3	8.98
	Park	10	3.43	13.2	4.36	12	3.64	15.7	5.72	13.3	4.29	18.2	6.86	14.1	4	18	8.82
	Overall	**9.87**	**2.97**	**12.5**	**3.94**	**11.8**	**3.72**	**16.3**	**5.2**	**13.7**	**4.5**	**18.3**	**6.88**	**14.8**	**4.39**	**19.9**	**9.35**

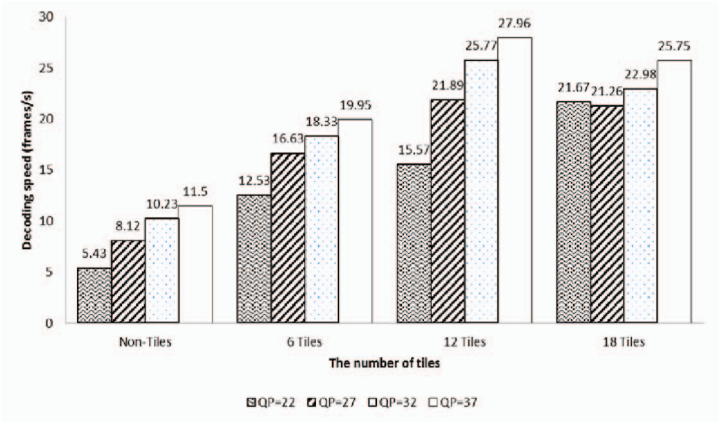
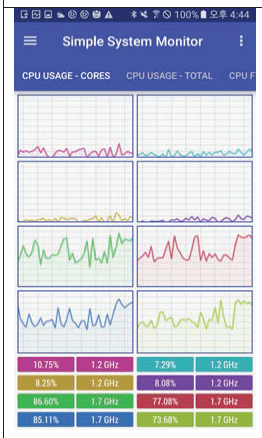
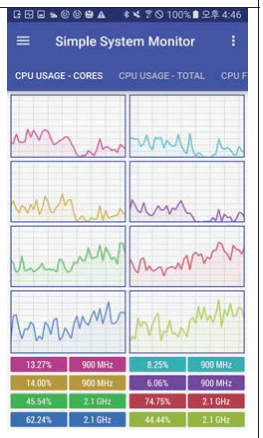


Fig. 8. The decoding speed of Tile-based HEVC deccoder base on Libde265

TABLE IV. THREAD ALLOCATION BY LIBDE265 AND OPENHEVCDECODER

Libde265	OpenHEVCDecoder
	

V. CONCLUTION

This paper presents the SIMD-based HEVC decoder on mobile platform by using both openHEVCDecoder and Libde265. It concludes that the HEVC decoding time with SIMD instructions scaled up 4~5 times on mobile processor. The maximum decoding frame rate of the proposed tile-based parallel processing was increased to 28 frames/s on Libde265. In term of decoding speed, the experimental results show that Libde265 is better than openHEVCDecoder in case tile-based parallel processing but less efficient in case of none tiles used. For further study, we will continue to examine the effect of SIMD on each module in detail including: inter/intra prediction, IT, DF, SAO. Furthermore, we will compare openHEVCDecoder and Libde265 HEVC decoder in functionality level as well as their effects on performance level.

ACKNOWLEDGMENT

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2015R1C1A1A02037743).

REFERENCES

[1] Thomas Wiegand, Gary I.Sullivan, Woo-Jin Han and Jens-Rainer Ohm, “Overview of the High Efficiency Video Coding (HEVC) Standard” IEEE Transactions on circuits and systems for video technology, vol. 22, No. 12, December 2012.

[2] Frank Bossen, Benjamin Bross, Karsten Suhringn and David Flynn “HEVC Complexity and Implementation Analysis” IEEE Transaction on Circuits and systems for video technology, vol.22, No. 12, December 2012.

[3] Chi Ching Chi, Mauricio Alvarez-Mesa, Benjamin Bross, Ben Juurlink and Thomas Schierl “SIMD Acceleration for HEVC Decoding” IEEE Transactions on circuits and systems for video technology, Vol.25, No.5, May 2015.

[4] Hui Yong, Ronggang Wang, Wenmin Wang, Zhenyu Wang, Shengfu, Bingjie Han and Wen Gao “Acceleration of HEVC Transform and Inverse Transform on ARM NEON Platform” in Intelligent Signal Processing and Communications Systems (ISPACS), pp 169-173, 2013.

[5] Yeongil Ryu, Hyun-Joon Roh, Shin Jin Kang, Soo-Kyun Kim, Eun-Seok Ryu, “Non-Uniform HEVC Tile Partitioning Method for Asymmetric Multicores”, Proceedings of The 11th Asia Pacific International Conference on Information Science and Technology(APIC-IST 2016), pp. 229-231, Jun. 27, 2016.

[6] Hyun-Joon Roh, Sung Won Han, Eun-Seok Ryu, “Prediction Complexity-Based HEVC Parallel Processing for Asymmetric Multicores”, Springer, Multimedia Tools and Applications (MTAP), 2017.

[7] OpenHEVCDecoder; Link: https://github.com/OpenHEVC

[8] Libde265, Link: https://github.com/strukturag/libde265

[9] ffmpeg, Link: https://www.ffmpeg.org/