AVS VIDEO DECODING ACCELERATION ON ARM CORTEX-A WITH NEON

Jie Wan[1], Ronggang Wang[1], Hao Lv[1], Lei Zhang[1], Wenmin Wang[1], Chenchen Gu[3], Quanzhan Zheng[3] and Wen Gao[2]

[1]School of Computer & Information Engineering, Peking University Shenzhen Graduate School
[2]National Engineering Laboratory of Video Technology, Peking University
[3] Tecent Technology（ShenZhen）Co., Ltd.
(e-mail: wanjie@sz.pku.edu.cn, rgwang@pkusz.edu.cn, {hlv, zhanglei}@sz.pku.edu.cn, wangwm@pkusz.edu.cn, {cicelygu, stevezheng}@tencent.com, wgao@pku.edu.cn)

ABSTRACT

Acceleration of Audio Video coding Standard (AVS) Jizhun profile decoder has been proposed for ARM Cortex-A with NEON. Data level parallelism is utilized to effectively use the SIMD capability of NEON. Key modules are redesigned to make them SIMD friendly. Our optimized C implementation is set as start point for the acceleration. Thanks to effective use of ARM Cortex-Ax architecture and NEON SIMD engine, AVS decoding has been accelerated considerably. Results are provided for the overall acceleration in decoding and individual acceleration for key modules of AVS decoder.

Index Terms— video decoder, AVS, ARM, NEON, SIMD, acceleration

1. INTRODUCTION

Video decoding in real-time on mobile phone is key to cater for the recent high demand for mobile multimedia applications. Nowadays, hand held devices are typically equipped with low power processor. On the other hand, with the advancement in the display system, hand held devices are also now capable of displaying video up to VGA resolution even to HD(720p) video. Video decoding is computationally intensive and involves a lot of data processing. In order to support decoding VGA (even 720p) video in real-time and display it smoothly, both high performance processor and video decoding acceleration techniques utilizing underlying processor architecture efficiently are essential.

Fortunately, the data processing operations of video decoding are mostly done at pixel level and similar operations are performed on each pixel in the entire macroblock partition with typical sizes of 16x16 pixels, 16x8 pixels and 8x8 pixels etc. Considering the same operations to be performed for entire macroblock partition, it can be efficiently performed for a group of pixels (e.g. 4, 8 or 16) in the packed form. SIMD (Single instruction multiple data) is one of the architecture which is evolved around this concept to perform the same operation on multiple data simultaneously. With a SIMD processor there are two improvements to data processing. For one the data is understood to be in blocks, and a number of values can be loaded all at once. Another advantage is that SIMD systems typically include only those instructions that can be applied to all of the data in one operation. In other words, if the SIMD system works by loading up eight data points at once, the operation being applied to the data will happen to all eight values at the same time. The first widely-deployed desktop SIMD was with Intel's MMX extensions to x86 architecture in 1996, followed in 1999 by SSE. Similarly NEON technology is a 128 bit SIMD (Single Instruction, Multiple Data) architecture extension for the ARM Cortex™-A series processors, designed to provide flexible and powerful acceleration for the low power mobile multimedia applications, delivering a significantly enhanced user experience.

On the other hand, implementing an algorithm with SIMD instructions usually requires human labor; most compilers don't generate SIMD instructions from a typical C program. And to implement video decoding efficiently on NEON, the algorithms should be re-implemented such that they can be made to use SIMD instructions efficiently. In this paper we will focus on the design and implementation of AVS video decoder which can take advantage of the NEON SIMD architecture on smartphone. The rest of the paper is organized in five sections. Section II will describe the Cortex™-A and NEON SIMD architecture, section III will give brief introduction of the AVS video decoder and the profiling information of main modules. Acceleration methods on typical modules would be described in section IV. Section V will provide the acceleration results and section VI will provide the summary.

2. INSTRUCTION PROCESSING WITH NEON

The ARM SIMD media extensions were introduced with the ARMv6 architecture, beginning with ARM1136 and

978-1-4673-2193-8/12/$31.00 ©2012 IEEE

continuing through ARM1176, ARM11, Cortex-A5, Cortex-A8 and Cortex-A9. These SIMD extensions increase the processing capability of ARM processor-based SoC without materially increasing the power consumption. The SIMD extensions are optimized for a broad range of software applications including video and audio codecs, where the extensions increase performance by up to 75% or more. NEON technology is introduced in the ARMv7 architecture and is only available with ARM Cortex-A class processors. NEON technology builds on the concept of SIMD with a dedicated module to provide 128-bit wide vector operations, compared to the 32bit wide SIMD in the ARMv6 architecture.

NEON technology is cleanly architected and works seamlessly with its own independent pipeline and register file. NEON technology is a 128 bit SIMD architecture extension for the ARM Cortex-A series processors, designed to provide flexible and powerful acceleration for consumer multimedia applications, delivering a significantly enhanced user experience. It has 32 D registers (D0-D31), 64-bits wide and Q registers(Q1-Q15) are composed of two consecutive D registers as showed in Fig.1(a). 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, single precision floating point as showed in Fig.1 (b). 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, single precision floating point; Instructions perform the same operation in all lanes as showed in Fig.2.

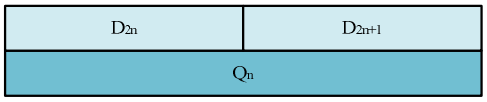

Fig.1(a). Q register is composed of two consecutive D registers

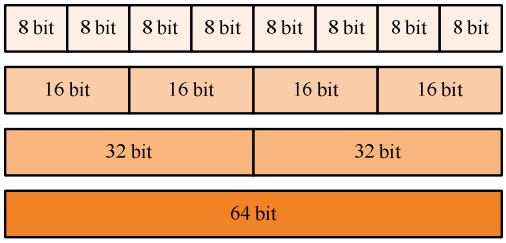

Fig.1(b). Data type of D register

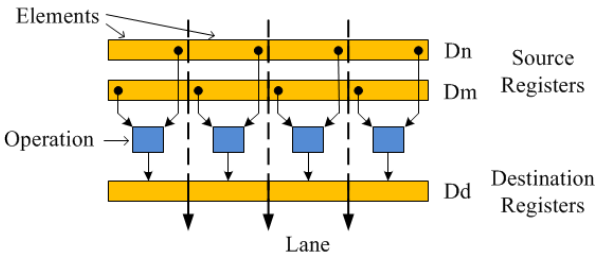

Fig.2. NEON instructions processing

3. AVS DECODER

AVS video standard is developed by the Audio Video Coding Standard Working Group of China (AVS working group in short), which was approved by the Chinese Science and Technology Department of Ministry of Information Industry in June 2002. The mandate of the AVS working group is to establish the China's national standards for compression, manipulation and digital right management in digital audio and video multimedia equipment and systems. AVS working group has finished the first version of AVS video standard in December 2003[1], and it has been approved as a national standard in 2006. This version mainly targets at high definition and high quality digital broadcasting, digital storage media and other related applications. It can be also applied in existing standard definition applications[4].

As a kind of high performance video coding standard, AVS has adopted a series of advanced technologies to reach the high-efficient compression ratio. The key technologies include 8x8 integer transform, entropy coding, intra prediction, inter prediction, 1/4 pixel interpolation, loop filter, etc.. AVS encoder employs temporary redundancy and spatial redundancy to improve compression performance, so the process of decoding is usually made up of two parts: decoding residual data and reconstructing prediction data. Decoder first parses header information from the video data stream, which contains sequence header, picture header, and slice header information through entropy decoding. Then it begins to decode the stream macroblock by macroblock. As shown in Fig.3, the main steps are: reading the luma coefficients and chroma coefficients of one macroblock through VLD; doing inverse zig-zag scanning, inverse quantization, and inverse integer cosine transform, from which the decoder gets the residual data. The predictors are obtained from intra prediction or inter prediction according to the macroblock type. Then, the macroblock is reconstructed with the residual data and predictors. At last, De-blocking is performed on the reconstructed data to get the final decoded pixels.

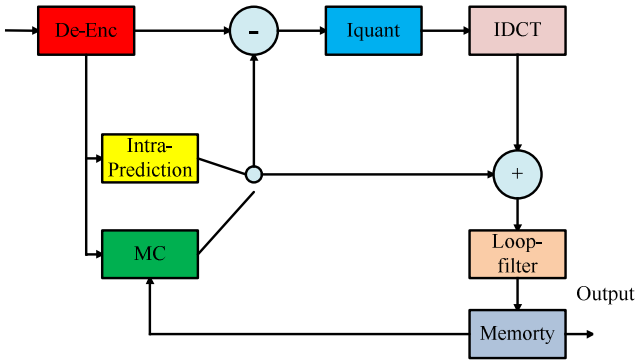

Fig.3 AVS Video Decoder block-diagram

Fig.4 shows the profile of AVS Video Decoder for News(480p) bit-stream. We can see that about 40% of decoding time is consumed by motion compensation module, and nearly 30% of decoding time is consumed by modules of de-blocking, IDCT, and Reconstruction. These modules have the features of performing the same operation on multiple data simultaneously, and are suitable for SIMD acceleration.

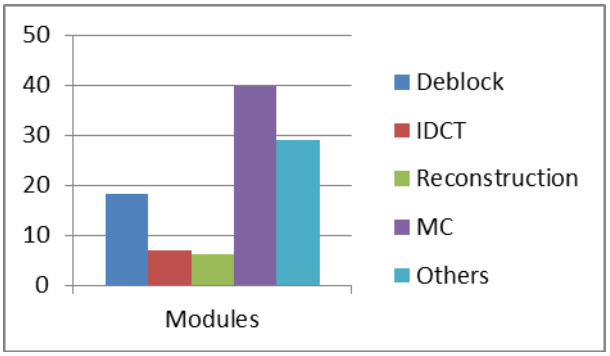

Fig.4 Contribution of various modules in total load

4. ACCELERATION METHODS

4.1. De-blocking filter

Since Block operations (DCT and motion compensation) tend to bring block artifacts at the block boundaries, de-blocking filter is utilized to reduce this phenomenon. Reduction in block artifacts will help to improve reference picture and hence the performance of motion estimation. De-blocking operation is performed on each block boundary, which is a time-consuming process. In AVS decoder, de-blocking is performed on 8x8 block boundaries. The length of D register is 64-bits, which is 8 pixels width, so we can deal with one boundary at a time. De-blocking is a conditional operation, and we have to calculate whether one pixel is satisfied with the de-blocking condition. For SIMD instructions those deal with data in parallel, they can't export single pixel alone. NEON has compare instructions such as VCLT[6] , which is to compare the first operand to the second operand vector by vector in parallel, if the condition is true, the bits of the destination vector are all set to 1 (otherwise 0), this destination vector is as a flag register. By this instruction we can know which vector needs to do de-blocking operation. According to the flag register, VBIT and VBIF[6] instructions can help us to set the output value as either the original input value or filtered value. De-blocking operations are performed both in vertical and horizontal directions. However, de-blocking in vertical direction is not as convenient as that in horizontal direction. Since the pixels to be filtered in vertical direction are from the same row, we have to load the pixels of each row then use VTRN[6] instruction to arrange these pixels, Fig.5 shows how this instruction works. In order to use SIMD instructions to perform de-blocking operation, we have to calculate all the pixels, no matter whether one pixel is satisfied with the condition, even though there is just one of them is required to do de-blocking, which will limit the acceleration of de-blocking operation.

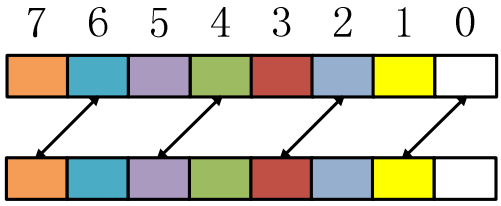

Fig.5. D registers VTRN.8 operation

4.2. Motion Compensation

Motion Compensation uses the reference frame and motion vector to generate the predictors of one macroblock partition. If the motion vector points to a fractional pixel position, interpolation filter is used to estimate the pixel value at this position. In AVS, two steps four taps filters [2] are adopted as the interpolation filters. To generate horizontal or vertical half pixels, the four tap filter of {-1, 5, 5, -1} (see Eq.1) is used. To generate horizontal or vertical quarter pixels, four tap filter of {-1, 5, 5, -1} is first used to generate the middle results of two half pixels respectively, and then four tap filter of {1, 7, 7, 1} is further used to generate the final result. To simplify this process, a 5 tap filter of {-1, -2, 96, 42, -7} (or {-7, 42, 96, -2, -1}) is performed on neighbor five integer pixels to get the final result directly. To illustrate how to use the SIMD instructions to generate the fractional pixel for a macroblock partition, we take the interpolation process of vertical and horizontal sub-pixels as example. To interpolate the entire partition, we need to apply this filter on all the columns of the MB.

For vertical interpolation, different integer pixels of each row are referenced. For 8x8 blocks, we can load 8 integer pixels in the same row in a D register and 4 rows can be loaded in 4 different D registers, as showed in Fig.6. Now all the multiplication, addition, rounding, and saturation instructions can be done in parallel using SIMD operations. For rounding and saturation operations we can use VQRSHRUN[6] instruction in NEON, the Q means saturation, the first R means rounding and the U means transforming signed data to unsigned data.

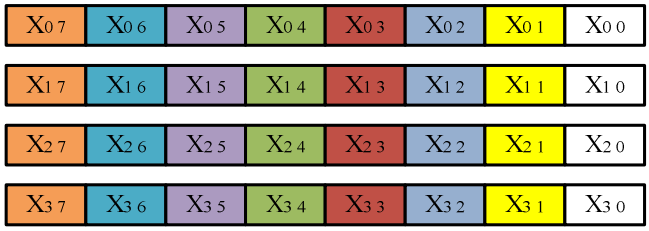

Fig.6 Data arrangement for vertical interpolation

For horizontal interpolation, the 4 integer pixels in Eq.1 is in the same row. We need to arrange the pixels so that we can use SIMD instructions efficiently. NEON has a VEXT[6] instruction, through which we can arrange the pixels in one row. Fig.7 shows how this instruction works. Fig.8 shows how to load 16 integer pixels in a Q register and use VEXT instruction to arrange it into 4 different D registers. The rest operations are same as those of vertical interpolation.

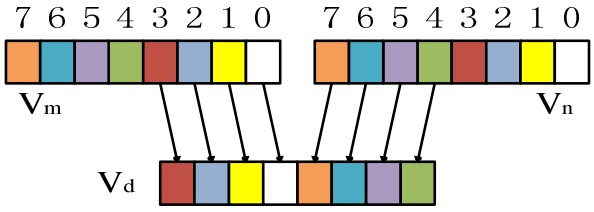

Fig.7 VEXT.8 instruction

By this way, we can interpolate 8 pixels in parallel. For the horizontal and vertical quarter-pixel interpolation, we can use the Q register to retention the half pixel for next stage of interpolation.

Fig.8 Data arrangement for horizontal interpolation

4.3. Inverse Transform

AVS performs 8x8 integer transform which can implemented using additions and shifts and does not require multiplication for its implementations. Intermediate results are restricted to 16-bit length. Addition, shift and multiplication can be easily implemented with SIMD operations. In order to prevent overflowing, we need 32-bit to store the temporary value, and restricted to 16-bit value in the end. So here we can use 8 D registers one time with 4 coefficients per register, and operate in parallel. And after we perform 2 times of the 8x4 transform, the row transform is finished. Here the middle results shall be restricted into 16-bit value. Then we transpose the matrix in place using VZIP instruction of NEON, and the result of the row transform can be directly used as input of the column transform. The 8x4 transform used here is almost the same as the row transform. After 2 times of 8x4 transform, the column transform is finished too. And the result should be changed to 16 bits value again. The fast algorithm of AVS IDCT [3] is used to accelerate the transform. And by simplifying the butterfly algorithm and performing addition, shift and multiplication in parallel, the speed is improved further.

5. ACCELERATION RESULTS

The optimized C code is used as the base for the acceleration. We use the LG P920 mobile phone as the platform to test our acceleration methods. This phone has Cortex-A9 MPCore processor with NEON running at 1GHz. Visual Studio 10 is used to compile the C codes and RVDS is used to compile the assemble codes of the accelerated modules in AVS Decoder.

Table.1 Modules Performance Improvement

Sequences		Before Optimization	After Optimization
Foreman	MC	1.46	0.34
	Deblock	0.41	0.14
	IT	0.41	0.24
	RCN	0.27	0.02
News	MC	4.18	0.9
	Deblock	1.89	0.57
	IT	0.73	0.25
	RCN	0.67	0.18
Mobile-calendar	MC	5.28	1.37
	Deblock	1.32	0.45
	IT	1.04	0.44
	RCN	0.91	0.13

Table.1 shows the seconds consumed by key modules which we dealt with before and after acceleration for Foreman(CIF-1Mbps), Mobile-calendar(480p-2Mbps) and News(480p-2Mbps) test bit-streams respectively. News has vary less motion and almost the same background, and Foreman has moderate motion and texture while Mobile-calendar has moderate motion and vary high texture. Table.2 shows the ratio of acceleration in terms of decoded frames per seconds (FPS) after applying the acceleration methods.

Table.2 Overall Performance Improvement Ratio In FPS

Sequences	Improvement Ratio
Foreman	2.09
News	2.11
Mobile-calendar	2.11

From the Tables we can see that the improvement ratio is more than 2 multiples. Motion Compensation is the main time-consuming module, chroma motion compensation consumes almost half the time of Motion Compensation, our implement of the chroma motion compensation is based on 4x4 blocks, the improvement is not as much as 8x8 blocks.

Reconstruction process is running in high parallel, so the optimization is significant. Infected by numerous judgment operations, de-blocking process acceleration is limited.

6. SUMMARY

AVS Video Decoder with SIMD instructions on ARM processor with NEON is implemented. Algorithms of the key modules are redesigned in order to utilize the SIMD architecture efficiently. The acceleration results compared with C code are shown that the AVS decoding speed can be doubled by efficient utilizing NEON instructions.

7. ACKNOWLEDGE

This work was supported by the grant of National Basic Research Program of China (973 Program) 2009CB320907, the grant of Key Projects in the National Science & Technology Pillar Program 2011BAH08B03, Tencent and the grant of Shenzhen Basic Research Program of JC201104210117A.

8. REFERENCES

[1] Draft of Advanced Audio Video Coding – Part 2:Video, AVS_N1063, 2003.

[2] R.G. Wang, C. Huang; J.T. Li, and Y.F. Shen, "Sub-pixel motion compensation interpolation filter in AVS[video coding]," *Multimedia and Expo, 2004. ICME '04. 2004 IEEE International Conference on*, vol.1, no., pp. 93- 96 Vol.1, 27-30 June 2004.

[3] L. Yu, S.J. Chen, J.P. Wang, "Overview of AVS Video Coding Standard," *Signal Processing: Image Communication,* Vol. 24, Issue 4, pp 247-262, 2009.4

[4] J.A. Zheng, "Research on AVS Video Decoding and Optimization of Decoding Key Module [D]," Xiamen university master's academic dissertation, 2007.

[5] T. Ebrahimi, and C. Horne, "Mpeg-4 Natural Video Coding – An Overview," *Signal Processing Image Communication.* Tutorial Issue on the MPEG-4 Standard, (15): 365-385, 2000.

[6] ARMV7-A architecture reference manual. Available on Request from ARM

[7] Tero. Rintaluoma, "SIMD Performance in Software Based Mobile Video Coding," *Embedded Computer Systems (SAMOS), 2010 International Conference*.