HIGH DEFINITION IEEE AVS DECODER ON ARM NEON PLATFORM

Ronggang Wang, Jie Wan, Wenmin Wang, Zhenyu Wang, Shengfu Dong, Wen Gao
School of Electronic and Computer Engineering, Peking University Shenzhen Graduate School

ABSTRACT

Nowadays, mobile devices are capable of displaying video up to HD resolution. In this paper, we propose two acceleration strategies for Audio Video coding Standard (AVS) software decoder on multi-core ARM NEON platform. Firstly, data level parallelism is utilized to effectively use the SIMD capability of NEON and key modules are redesigned to make them SIMD friendly. Secondly, a macroblock level wavefront parallelism is designed based on the decoding dependencies among macroblocks to utilize the processing capability of multiple cores. Experiment results show that AVS (IEEE 1857) HD video stream can be decoded in real-time by applying the proposed two acceleration strategies.

Index Terms— Video decoder, parallel decoding, IEEE 1857, AVS, ARM, NEON, SIMD

1. INTRODUCTION

With the progresses in the display system, mobile devices are now capable of displaying video up to HD video. Video decoding in real-time on mobile devices is key to cater for the recent high demand for mobile multimedia applications. AVS video standard is developed by the Audio Video Coding Standard Working Group of China. The coding performance of AVS is nearly doubled than that of MPEG-2. Recently it is approved as IEEE 1857 standard. However, real-time decoding of AVS HD video bit-stream is still a challenge work for mobile platform with low-power ARM processors. In order to support decoding HD 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, modern mobile devices are widely equipped with ARM Cortex™-A series processors, and NEON instructions are supported to provide flexible and powerful acceleration for consumer multimedia applications.

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. NEON instructions are evolved around this concept to perform the same operation on multiple data simultaneously, and video decoding can be considerably accelerated by skillfully using NEON instructions. Meanwhile, ARM processors with multiple cores are immerging in the market, and video decoding can be further accelerated by smartly utilizing multiple cores.

There are several works on acceleration of H.264 [1] and VC-1 [2] decoder by utilizing SIMD capability of ARM NEON platform. In our previous work, we use the similar strategy to accelerate AVS decoder [3], but HD video can't still be decoded in real-time only by this kind of acceleration. In this paper, we accelerate the IEEE AVS decoder by both of the following two strategies. Firstly, data level parallelism is exploited to effectively use the SIMD capability of NEON. Then, a macroblock level wavefront parallelism is designed to utilize the multiple cores of ARM processor. Consequently, AVS HD video stream can be decoded in real-time on the multi-core ARM NEON platform.

The rest of the paper is organized in five sections. Section 2 gives the background information of ARM NEON SIMD architecture and AVS decoding process. Section 3 describes the re-designs of key modules in AVS decoder to utilize SIMD capability of NEON, and section 4 presents our proposed macroblock level wavefront parallelism to exploit multiple cores. Experiment results are shown in Section 5 and this paper is concluded in section 6.

2. ARM NEON AND AVS DECODING

The ARM SIMD media extensions were introduced with the ARMv6 architecture, beginning with ARM1136 and 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 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 architecture has 32 D registers (D0-D31), 64-bits wide and Q registers(Q1-Q15) are composed of two consecutive D registers as shown 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 shown 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; Instructions perform the same operation in all lanes as shown in Fig.2.

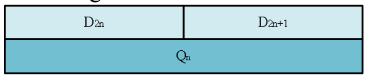


Fig.1(a). Q register is composed of two consecutive D registers

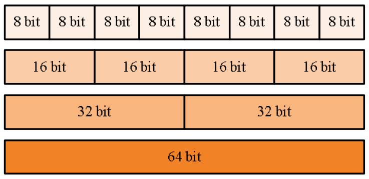


Fig.1(b). Data type of D register

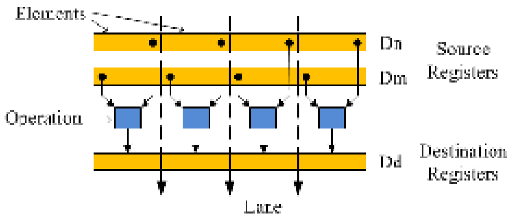


Fig.2. NEON instructions processing

AVS video standard is developed by the Audio Video Coding Standard Working Group of China (AVS working group in short) [4], and it is approved as a China national standard in 2006. Recently AVS is approved as IEEE 1857 standard. AVS adopts a series of advanced technologies to reach the high compression performance, such as 8x8 integer transform, 2DVLC entropy coding, spatial intra prediction, fractional-pel motion compensation, and de-blocking etc.. The decoding process 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 by entropy decoding. Then each picture/slice is decoded 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. The residue data is recovered by the above steps. The predictors are obtained from intra prediction or inter prediction according to the macroblock header information. Then, the macroblock is reconstructed by adding the residual data to the predictors. At last, De-blocking is performed on the reconstructed data to get the final decoded pixels.

Fig.4 shows the profile of our AVS Video Decoder written in C codes for River (1080p) bit-stream by android-ndk-profiler. 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 own the features of performing the same operation on multiple pixels simultaneously, and are suitable for SIMD acceleration.

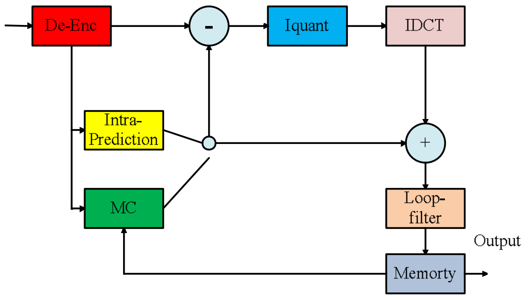


Fig.3 AVS decoder block-diagram

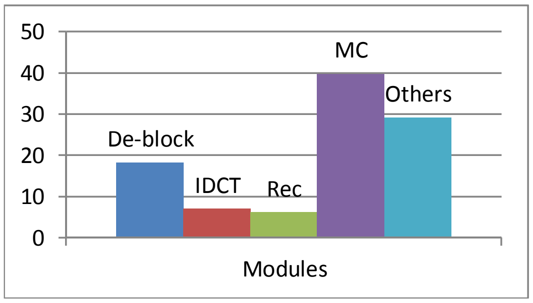


Fig.4. Time ratio of various modules in AVS decoder

3. DECODING ACCELERATION BY ARM NEON

3.1. De-blocking

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 in parallel. De-blocking is a conditional operation, and we have to calculate whether one pixel is satisfied with filtering condition. NEON has comparison instructions such as VCLT[5] , which is to compare the first operand to the second operand vector by vector. When the comparison condition is true, the bits of the destination vector are all set to 1 (otherwise 0), and this destination vector is 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 [5] instructions can help us to set the output value as either the original input values or filtered values. 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 [5] instruction to arrange these pixels, Fig.5 shows how this instruction works.

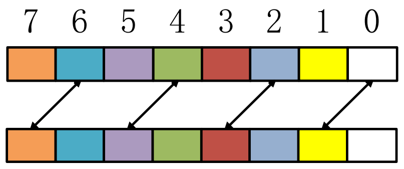


Fig.5. D registers for VTRN.8 operation

3.2. Motion compensation

Motion Compensation uses the reference frame and motion vector to generate the predictors of each macroblock partition. When 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 [4] are adopted as the interpolation filters. To generate horizontal or vertical half pixels, the four tap filter of {-1, 5, 5, -1} is used. To generate horizontal or vertical quarter pixels, four tap filter of {-1, 5, 5, -1}/8 is first used to generate the middle results of half pixels, and then another four tap filter of {1, 7, 7, 1}/16 is further used to generate the final result. To simplify this process, a 5 tap filter of {-1, -2, 96, 42, -7}/128 (or {-7, 42, 96, -2, -1}/128) 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 apply this filter on all the columns of the MB.

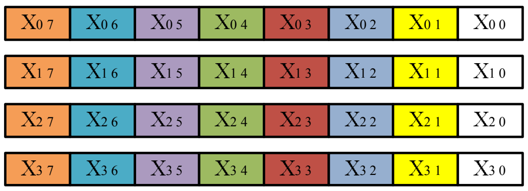

X07 X06 X05 X04 X03 X02 X01 X00
X17 X16 X15 X14 X13 X12 X11 X10
X27 X26 X25 X24 X23 X22 X21 X20
X37 X36 X35 X34 X33 X32 X31 X30

Fig.6 Data arrangement for vertical interpolation

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 shown 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[5] instruction in NEON, the Q means saturation, the first R means rounding and the U means transforming signed data to unsigned data. 8 pixels can be interpolated 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.

3.3. Inverse transform

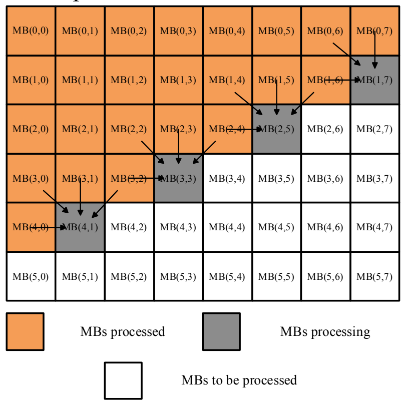
8x8 integer transform in AVS can be implemented by only using additions and shifts. In order to prevent overflowing, 32-bit registers are needed to store the temporary values, and the temporary values clipped to 16-bit values in the end. We use 8 D registers with 4 coefficients per register, and the 8x4 coefficients are operated in parallel. After we perform twice 8x4 transforms, the row transform is finished. The results of row transform are clipped into 16-bit values. Then the row transformed matrix is transposed in place using VZIP instruction of NEON. Consequently, the results of the row transform can be directly used as inputs of the column transform. The 8x4 column transform used here is almost the same as the row transform. After twice 8x4 transforms are performed, the results are also clipped to 16 bits values. The fast algorithm of AVS IDCT [4] is used to accelerate the transform. And by simplifying the butterfly algorithm and performing addition, shift and multiplication in parallel, the calculations are accelerated further.

4. MB-LEVEL WAVEFRONT PARALLELISM

There are some dependencies among macroblocks decoding, for example, the motion vector of current macroblock is related to the motion information of above and/or left macroblocks due to motion vector prediction method in AVS. According to the conditional dependencies, we proposed a macroblock level wavefront parallelism to make full use of the parallel relationship among macroblocks.

In AVS, 2DVLC and context adaptive arithmetic coding are adopted as entropy coding tools. There're dependencies for neighbor MB's parsing, especially for arithmetic doing, one macroblock's entropy decoding is related to all its previous macroblocks in the same slice. So the entropy decoding can't be parallelized in MB level. In the recently issued standard of High Efficiency Video Coding (HEVC), there is a Wavefront Parallel Processing (WPP) parallelism by relaxing the context dependency between macroblock rows[6]. But in one hand WPP will involve somewhat performance loss; in the other hand WPP can't be supported by AVS bit-stream. So WPP can't be used in AVS decoding. The dependencies among neighbor MBs for motion vector prediction, intra prediction, and the de-blocking modules are shown as arrows in Figure.7. Every macroblock is related to its above and left neighbors.

Based on the above dependencies, we divide AVS decoder into two parts: parsing and reconstructing. Parsing part reads the bit stream and parses it into syntax element values. Reconstructing part performs motion vector prediction, motion compensation, reconstruction and de-blocking. For MBs in one slice, parsing part runs sequentially and reconstructing part works in parallel. During reconstruction, current macroblock needs the information from left, top-left, top and top-right macroblocks. Processing MBs in a diagonal wavefront manner satisfies all the dependencies and allows to exploit parallelism among MBs. When decoding a slice, a main thread is responsible to parse all the macroblocks, other child threads are set up and each thread is responsible for the reconstructing of one row macroblocks. After the parsing the macroblocks of a row by the main thread, a child thread will start to reconstruct the macroblocks of this row, meantime, main thread proceeds to parse the next row of macroblocks. If the syntax information of the top, top-left, top-right and left macroblocks is available by entropy decoding, the macroblocks in other rows can be reconstructed in parallel. From Fig.6 we know that MB(1,7), MB(2,5), MB(3,3) and MB(4,1) can be reconstructed in parallel.


MB(0,0) MB(0,1) MB(0,2) MB(0,3) MB(0,4) MB(0,5) MB(0,6) MB(0,7)
MB(1,0) MB(1,1) MB(1,2) MB(1,3) MB(1,4) MB(1,5) MB(1,6) MB(1,7)
MB(2,0) MB(2,1) MB(2,2) MB(2,3) MB(2,4) MB(2,5) MB(2,6) MB(2,7)
MB(3,0) MB(3,1) MB(3,2) MB(3,3) MB(3,4) MB(3,5) MB(3,6) MB(3,7)
MB(4,0) MB(4,1) MB(4,2) MB(4,3) MB(4,4) MB(4,5) MB(4,6) MB(4,7)
MB(5,0) MB(5,1) MB(5,2) MB(5,3) MB(5,4) MB(5,5) MB(5,6) MB(5,7)
MBs processed
MBs processing
MBs to be processed
Fig.7 Dependencies among macroblocks

5. EXPERIMENT RESULTS

Our optimized AVS decoder written in C codes is used as the base for the acceleration. We use the Samsung galaxy S3 mobile phone as the test platform for our experiment. It adopts Samsung Exynos 4210 processor with 4-core ARM NEON architecture running at 1GHz. Visual Studio 10 is used to compile the C codes and RVDS is used to compile the assemble codes of the redesigned modules in Section 3.

Table.1. Decoding fps of 6Mbps bit-streams for various versions of AVS decoder

Sequences	Version 1 (fps)	Version 2 (fps)	Version 3 (fps)	Speedup (ratio)
crowdrun	6.66	15.92	29.94	4.50
kimono	6.76	15.69	28.92	4.28
parkscene	7.00	16.11	28.91	4.13
riverbed	7.04	15.43	29.07	4.13
sunflower	6.80	16.23	31.25	4.60
average	6.85	15.88	29.62	4.33

Five typical 1080P HD videos are used as test sequences. They are encoded to 6Mbps and 10Mbps bit-streams by AVS Jizhun profile reference software, and the frame rate is set as 25 fps. Table 1 and Table 2 give the number of decoded frames per second (fps) by various versions of AVS decoder for 6Mbps bit-streams and 10Mbps bit-streams respectively. The version 1 is the AVS decoder written in C codes, the version 2 is the accelerated version 1 by the strategy described in Section 3, and the version 3 is the accelerated version 2 by the strategy presented in Section 4.

Table.2. Decoding fps of 10Mbps bit-streams for various versions of AVS decoder

Sequences	Version 1 (fps)	Version 2 (fps)	Version 3 (fps)	Speedup (ratio)
crowdrun	6.09	14.16	28.43	4.67
kimono	5.91	13.48	27.96	4.73
parkscene	6.28	14.12	26.37	4.20
riverbed	6.11	13.30	25.04	4.10
sunflower	5.86	13.81	26.60	4.53
average	6.05	13.77	26.88	4.45

We can see from Table 1 & 2 that AVS decoding speed is about doubled by utilizing ARM NEON instructions, and doubled further by utilizing MB-level wavefront parallelism. The version 3 of AVS decoder can decode typical HD (1080p) video streams in real-time.

6. CONCLUSIONS

HD AVS decoder on ARM processor with NEON is implemented both by utilizing the SIMD architecture and a MB-level wavefront parallelism. These strategies can also be used to accelerate other similar video decoding processes, such as H.264/AVC and HEVC etc..

7. ACKNOWLEDGE

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 JCYJ20120614503016 23.

8. REFERENCES

[1] 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.
[2] 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
[3] 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.
[4] L. Yu, S.J. Chen, and J.P. Wang, "Overview of AVS Video Coding Standard," *Signal Processing: Image Communication,* Vol. 24, Issue 4, pp 247-262, 2009.4.
[5] ARMV7-A architecture reference manual, at http:// *infocenter.arm.com.*
[6] C.C. Ching, M. Alvarez-Mesa, B. Juurlink, G. Clare, F. Henry, S. Pateux, and T. Schierl, "Parallel Scalability and Efficiency of HEVC Parallelization Approaches," *Circuits and Systems for Video Technology, IEEE Transactions on* , vol.22, no.12, pp.1827,1838, Dec. 2012.