VC-1 Video Decoder Optimization on ARM Cortex-A8 with NEON

Chirag Pujara, Anurag Modi, Sandeep G, Shilpa Inamdar, Deepa Kolavil, Vidhu Tholath

Samsung India Software Operations Pvt. Ltd.
Bangalore, India

{chirag.p, anurag.modi, g.sandeep, shilpa.i, deepa.subin, vidhu.bennie}@samsung.com

Abstract— **Optimization of VC-1 Main profile decoder has been proposed using SIMD engine (NEON) of ARM Cortex-A8 processor. We have exploited data level parallelism to effectively use the SIMD capability of NEON. Ideas to make the algorithms SIMD friendly are also highlighted. With effective use of ARM Cortex- A8 architecture and NEON SIMD engine, the MCPS (Mega Cycles per second) requirement for decoding has reduced considerably as compared to our optimized C implementation. Results are provided for the overall performance gain in decoding and individual performance gain in various modules of VC-1 Main profile decoder.**

Keywords*— *Video codec*, *VC-1, Cortex-A8, NEON, SIMD, Optimization

I. INTRODUCTION

With the recent development of multimedia centric applications for the mobile devices, video codecs have become their integral part. These hand held devices use low power embedded processor for their operation. With the advancement in the display system, hand held devices are also now capable of displaying high resolution video. To support the higher resolution video encoding and decoding in real-time, it is essential to use underlying hardware architecture efficiently. Video codecs are computationally intensive and require a lot of data processing to encode and decode the data. These data processing operations are mostly carried out at pixel level (generally 8 bit) and similar operations are performed on each pixel in the entire Macroblock (MB) whose size is 16x16 pixels. These operations include scaling, multiply and accumulate (MAC), addition, saturation, rounding, shifts etc. Considering the same operations to be performed for entire MB, it can be efficiently performed for a group of pixels (e.g. 4, 8 or 16) in the packed form. Many architectures evolved around this concept to perform operations in parallel at the data level. SIMD (Single instruction multiple data) or widely known as sub word parallelism is one of the architectures which is suitable for video codec implementation. Intel MMX technology implements SIMD architecture which made multimedia centric PCs a great hit. Similarly ARM has designed ARM Cortex-A8 which has NEON SIMD engine which is specifically designed for the low power mobile applications. NEON instruction set has been designed keeping in mind the operations required in the implementation of multimedia codecs especially video codecs.

To implement video codec efficiently on NEON we need to design the algorithms such that they can use SIMD instructions efficiently. In literature, complexity of any newly proposed algorithm is judged by the number of operations required- this holds true for general purpose processors. On the other hand, for SIMD engines this assumption may not hold true. The algorithm which is suitable for SIMD implementation might be optimal (in the sense of cycles required to perform the operations) in spite of having more number of operations as compared to the serial algorithm. The very reason behind this is that, with SIMD architecture, many operations can be accomplished with single instruction.

In this paper we will focus on the design and implementation of VC-1 video decoder which can take advantage of the SIMD architecture. Though we have described the optimization of VC-1 decoder, the proposed ideas are generic in nature and they can be used for other codecs like MPEG-2, H.263, MPEG-4 and H.264. The rest of the paper is organized in five sections. Section II will describe the Cortex A-8 and NEON SIMD architecture, section III will give brief introduction to the VC-1 video decoder and its profiling information. Optimization ideas would be described in section IV. Section V will provide the optimization results and section VI will provide the summary.

II. ARCHITECTURE OF CORTEX-A8 WITH NEON

ARM Cortex–A8 has the new ARMV7 instruction set with a separate media processing engine known as NEON. The NEON vector instruction set adds very rich SIMD instruction set suitable for multimedia processing. Cortex-A8 has a superscalar architecture which supports the dual issue(with some restriction based on the underlying architecture) of instructions and has a 13 stage deep pipeline[4]. NEON adds thirty two 64 bit (know as Double word or D registers) vector registers to the general purpose ARM registers. These registers are called vector registers as operations are possible on packed vector of size 8, 16, 32 or 64 bit width. NEON also supports instructions that can operate on 128 bit data. In this case two consecutive D registers are combined together to make a Quad register (know as Q registers). Fig.1a shows how one can pack 8, 16, 32 and 64 bit data in one of these 32 D registers (D0-D31). Fig.1b shows the representation of Q register (Q0-Q15). NEON has a 10 stage deep pipe line which starts at the end of ARM pipe line. NEON instructions can also be dual issued with some restrictions. Cortex has two level of cache structure. L1 cache is of 16 KB (configurable up to 32 KB) each for both instruction and data while the L2

cache is unified and can be configured up to the size 1 MB. L2 cache improves the performance very significantly because of its size and the access speed as compared to the main memory. For NEON, access to L2 cache is non-blocking[3], which means it can access data directly from L2 in case of a cache miss in L1. Fig. 2 shows the architecture level diagram of Cortex A8 processor[4]. NEON supports various instructions for load/store, addition, multiplication, saturation and shifting which can operate on either D or Q registers. Detailed description of instruction set can be found in[4].

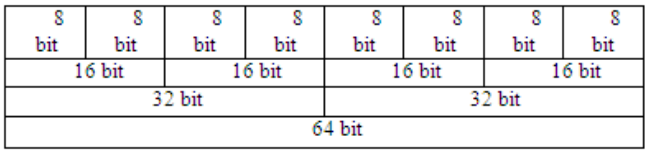


Fig. 1(a) Data packing in D register

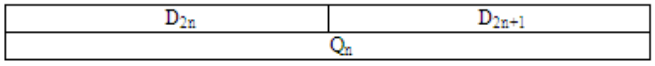


Fig. 1(b) Q register as a combination of two D registers

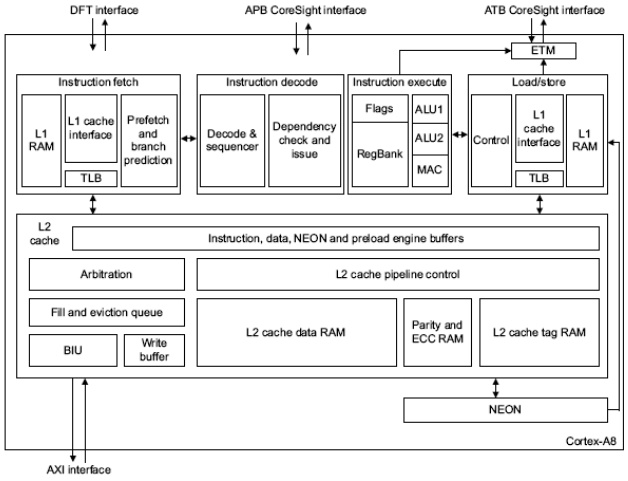


Fig.2 Cortex –A8 with NEON architecture diagram[4]

III. VC-1 DECODER

VC-1 is standardized by the Society of Motion Picture and Television Engineers (SMPTE) which is based on the Microsoft's WMV-9 (Windows Media Video) video codec. VC-1 supports three profiles, baseline for low complexity implementation, main profile for more coding gain with more complexity and advanced profile with the support for Interlace coding over and above the Main profile. Here we will focus on the Main profile. Important innovations that distinguish VC-1 from other standards are adaptive block size transform, 16-bit implementation of transform, multiple precision modes for motion compensation, uniform and non-uniform quantization, loop-filtering and overlap smoothing. With the use of these advanced coding tools, VC-1 can provide a very high coding gain similar to H.264/AVC with lesser computational complexity[1]. VC-1 supports variable size transforms (8x8, 8x4, 4x8 and 4x4). The larger size transform is good to capture the similarities in smoother spatial regions. The smaller size transforms are more suited for the areas with discontinuities because they produce less ringing effects. The transform is designed to be implemented on 16 bit fix point arithmetic which is very much suitable for SIMD operation. Motion Compensation (MC) supports 16x16 and 8x8 block sizes with the provision to measure the motion up to quarter pel. It uses different modes for motion compensation as per the motion vector resolution, block size and the interpolation filter type. Possible four modes are 1. Mixed block size (16x16, 8x8), ¼ pixel, bicubic, 2. 16x16, ¼ pixel, bicubic, 3. 16x16, ½ pixel, bicubic and 4. 16x16, ½ pixel, bilinear. These four modes provides enough flexibility to generate the reference block for the MC at the same time reduced complexity by not providing all the combinations of the above three parameters. Deblocking filter is kept in the motion estimation loop which is not merely a post-processing step. Loop filter will remove the visible 'blocky' artifacts to generate better reference frame. In VC-1 deblocking operation is performed on the same size that of the transform. Loop filter is highly conditional algorithm which makes it complex for implementation on processors. To reduce the complexity of this algorithm, VC-1 checks the filtering requirement only once for the 4 pixel wide boundary. Deblocking filter can not distinguish between the true edges across the blocks or the one introduced by the quantization. VC-1 uses the overlap smoothing to solve this problem. Overlap smoothing is a filter whose support is more than the transform block size. It will do sharpening of the edge in pre-processing and smoothing of the edge in post processing stage. It is not highly conditional like the deblocking operation, thus it can be used in low complexity baseline profile implementation where deblocking is switched off by the encoder. VC-1 supports both dead-zone and regular uniform quantization. The encoder uses dead-zone quantizer at large step sizes and uses uniform quantizer at lower step sizes. Extensive detail of all the algorithms of VC-1 can be found in[1].

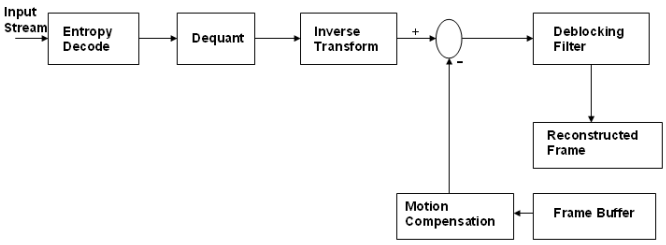


Fig. 3 VC-1 Main profile decoder block-diagram

A. *Profile Information*

Profiling is the first step for the optimization of any software. Profile will provide us the information about the contribution of all the modules in the percentage load (in terms of MCPS) and the coverage of those modules. As per the typical profile of any video decoder motion compensation (MC), inverse transform (IT), variable length decoding (VLD), deblocking filter (DB), de-quantization (DQ), reconstruction would contribute significantly in the computational

complexity. Fig. 4 shows the profiling data for VC-1 decoder for Foreman (QVGA, 384 Kbps, 30 fps) stream. From the profile we can see that the MC consumes as much as half the cycles required for total decoding. Other modules are DB, VLD+DQ (Merged function), IT and RECON as per the computational complexity. This order may change based on the bitstream, bit-rate and the spatial resolution.

We will discuss the modules which we have redesigned to use SIMD operations and the ones which are directly fit for SIMD implementation in section IV. Modules which are highly serial and more conditional like VLD and bit stream parsing are not suitable for SIMD implementation and should be implemented using ARMV7 assembly instructions.

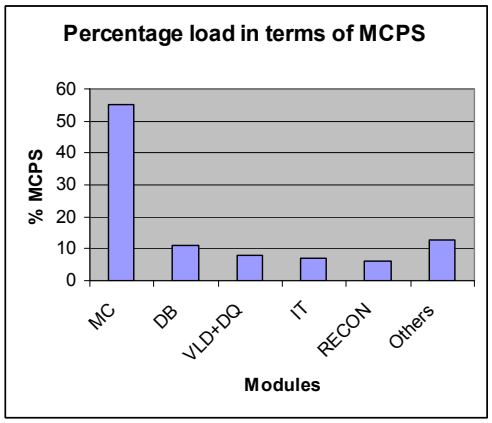


Fig. 4 Contribution of various modules in total load

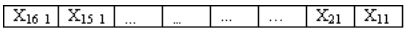
IV. OPTIMIZATION IDEAS

A. Motion Compensation

Motion compensation (MC) generates the predicted macroblock (MB) using the reference frame and the motion vectors. If the motion vectors are pointing to a fractional pel location then interpolation is required for generating pixels at non integer pixel grid (half pel or quarter pel). VC-1 uses three filters, one (bicubic) for ¼ pixel and two (bicubic, bilinear) for ½ pixel interpolation as mentioned in section III. Eq.1 and 2 show the 4-tap bicubic filtering operation for ½ pel and ¼ pel interpolation respectively. Here ¼ pixel and ½ pixel are interpolated values located at quarter and half pixel grid respectively, x_{nm} denotes the n^{th} pixel of m^{th} row (here 0< n< 4 for 4-tap filtering). To illustrate the use of SIMD operations, we will take the example of the vertical interpolation to generate quarter pel pixels for a MB using bicubic interpolation. To interpolate the entire MB, we need to apply this filter on all the columns of the MB. To perform filtering using SIMD operations (for eight pixels simultaneously), we apply the filtering on first four rows (as 4-tap filter is used) as shown in Fig.5. Here we can load all sixteen pixels in a row in one Q register and operate on eight at a time. Similarly first 4 rows can be loaded in four different Q registers. Now all multiplication, addition, shifts and saturation operations can be done in parallel using NEON SIMD operations to compute eight (not sixteen because intermediate result require sixteen bit precision) interpolated pixels in parallel. Right shift operation for the division by 64 and saturation both can be performed simultaneously using VQSHRN vector instruction [4]. Here Y_{mn} denotes the interpolated pixels at the quarter pixel grid.

$$1/4\,pixel = uint8\left(\frac{-4x_{11}+53x_{21}+18x_{31}-3x_{41}}{64}\right) \quad (1)$$

$$1/2\,pixel = uint8\left(\frac{-x_{11}+9x_{21}+9x_{31}-x_{41}}{16}\right) \quad (2)$$



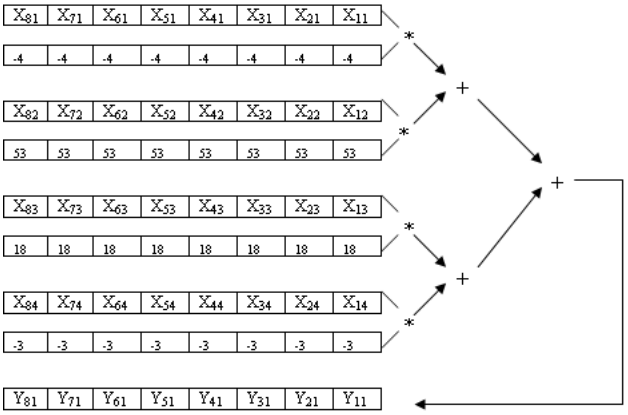


Fig.5 SIMD operations for Vertical Interpolation

To implement horizontal interpolation utilizing SIMD operations, data needs to be arranged in a specific way. Fig. 6 shows the data arrangement for filtering eight pixels of the first row with SIMD operations. In this case x_{nm} denotes the nth pixel in m^{th} row. VEXT instruction can be used to arrange the data in the required format [5]. The rest of the implementation is similar as the vertical interpolation. We implemented bicubic and bilinear filtering for ½ pixel interpolation in the same way and the method can be easily deduced from the implementation of the bicubic filtering for ¼ pixel interpolation.

Fig.6 Data arrangement for Horizontal Filtering

This way we can interpolate eight pixels in parallel using the SIMD instructions. SIMD operations are very effective for the interpolation. The same can be verified from the experimental results. For the rest of the modules input data would be loaded as a vector as in the case of vertical interpolation.

B. Deblocking Filter

Deblocking filter is used to reduce the 'blocky' artifacts at the block boundaries introduced by block operations like transform and quantization. Reduction in blocky artifacts will help to generate better reference picture, which will in turn improve the motion estimation. Deblocking is also one of the major contributors in the cycle consumption. Deblocking will be performed at the block boundaries as per the transform block size which can be one of the four(8x8, 8x4, 4x8 and 4x4). For P and B pictures, the block boundaries can occur at every 4^{th}, 8^{th}, 12^{th} etc pixel row or column depending on the transform size used. For I pictures filtering occurs at every 8^{th}, 16^{th}, 24^{th} etc pixel row and column as only 8x8 transform size is used. As NEON registers can hold 128 bit data and can perform operations over that using vector instructions, it is advantageous to deblock four 4x4 or two 8x4, 4x8 blocks together. This may require some modifications to be made at the algorithm implementation level. This idea can improve the cache efficiency as well as can reduce the number of cycles required. Fig.7 shows the data loading for deblocking of horizontal edge. As show in fig.7, all the p0 pixels can be loaded in to one D register, similarly all p1, p2, …, p7 pixels are loaded in D registers. Here we have combined two 4x4 blocks. NEON instructions cannot set any flags based on their output. Not to filter the natural edges present in original image, deblocking will be performed only if the edge is likely to be occurred due to the quantization or motion compensation. To ensure this the gradient across the block boundaries and the motion vector information is used. Due to various condition checks involved, loop filtering is a computationally expensive process. To reduce the complexity, determination of whether to filter across an edge or not is made only once every four pixel and the same decision is used for all four pixels. In order to use the SIMD operations, we need to use the NEON instructions (Vector comparison instructions like VCLE, VCGT[4]) that can set the bits of the destination register (flag register) based on the comparison between its two source operands. This flag register can be used as an input to choose between the filtered value or the original non filtered value using vector bit select instructions (VBIT, VBSL, VBIF[4]). The shortcut for determination of filtering operation may not be useful for the SIMD implementation because even though the filtering is required for one pixel boundary, all eight edges need to be filtered first and then based on the status of the status registers filtered or non-filtered pixels would be chosen. Due to the limitation posed by non-conditional instructions in NEON, the optimization level may not be comparable to the one achieved for interpolation in case of MC. This is also reflected in the optimization results.

C. Quantization/De-quantization

Quantization reduces non-significant high frequency coefficients to make long run of zeros which will be compactly represented by the entropy coding. De-quantization re-scales the quantized coefficients which is the inverse of quantization process. Four 4x4 blocks can be combined similarly as in the case of deblocking implementation to effectively use the NEON registers. De-quantization can be performed either in zigzag or raster scan order. Former has the advantage of terminating the DQ calculation at the last non zero coefficient but this accessing order is not suitable for SIMD operations. To implement with SIMD instructions we follow the raster scan order. This will let us load eight quantized coefficients together and rescale them in parallel. It may be noted that, in such an implementation, scaling operation will be done on all coefficients including zero valued coefficients. Hence actual gain depends on the number of non-zero coefficients in a block. Due to this unnecessary calculations associated with raster scan order we merged the de-quantization computation with the VLD. Here we perform the de-quantization at the same time of decoding the symbol, if it is found to be non-zero. This approach showed better results as compared to the SIMD implementation.

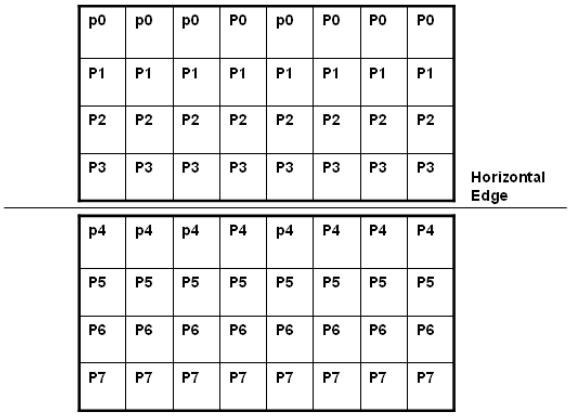


Fig. 7 Horizontal Deblocking

D. Transform

VC-1 supports variable block size (8x8, 8x4, 4x8, 4x4) integer transform which can be implemented using only additions and shifts and does not require multiplication for its implementations. Fig.8 shows the block sizes supported for the transform. Intermediate results can be represented in sixteen bit precision. Due to this, we can perform two 8x4 or 4x8 block and four 4x4 block transforms together. Additions and shifts can be easily implemented with SIMD operations. Here we can load all eight coefficients together and can operate in parallel. After computing the row transform, we can transpose the matrix in place using VZIP instruction [4] of NEON. Hence result of the row transform can be directly used as input for the column transform. This will reduce the interactions with the memory as the need of intermediate load/store has been eliminated. This can boost the performance.

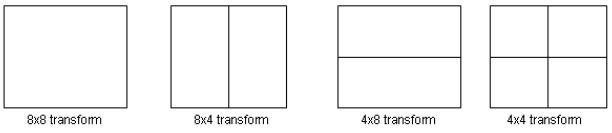


Fig. 8 Variable block sizes for the transform

V. OPTIMIZATION RESULTS

We have used our VC-1 main profile decoder implementation which is optimized for C as the base code for optimization. The OMAP 3530 based development board is used for porting and performance measurement of optimization ideas. This board has ARM Cortex-A8 with NEON running at 500 Mhz. Codesourcery compiler tool chain has been used for cross compiling the C codes. RVDS 3.1 development suite has been used to develop and debug the hand coded assemblies for the time critical modules described in section IV.

Table I shows the percentage of load contribution in terms of MCPS by core modules before and after the optimization for Foreman (QVGA-384Kbps), Akiyo (QVGA-384Kbps) and Stefan (QVGA-384 Kbps) test streams. Results are shown for the modules which are implemented with NEON SIMD instructions. Akiyo has very less motion and almost the same background, Foreman has moderate motion and texture while Stefan has high content of motion and texture. Table II shows the ratio of improvement in speed in terms of decoded frames per second (FPS) after applying all the proposed optimization ideas. The result in the table I confirms the effectiveness of the proposed optimization ideas. We can see that the maximum optimization has been achieved for Foreman sequence due to the higher contribution of MC module as shown in Table I. The reason for lesser performance improvement for the Akiyo test vector can be attributed to the fact that it has inherently lesser contribution from MC as compared to Foreman and Stefan sequences. Hence the contribution from the MC optimization reflects less in the performance improvement.

TABLE I
MODULE WISE PERFORMANCE IMPROVEMENT IN PERCENTAGE MCPS

Sequences		**Before Optimization**	**After Optimization**
Akiyo	MC	29.62	1.85
	DB	11	6.4
	IT	10	1.54
	RCN	9.16	0.66
Foreman	MC	55	4
	DB	11	6.22
	IT	7	1.02
	RCN	6	0.37
Stefan	MC	52	3.63
	DB	13	8
	IT	6	0.89
	RCN	5	0.37

TABLE II
OVERALL PERFORMANCE IMPROVEMENT RATIO IN FPS

Sequences	**Improvement Ratio**
Akiyo	2.10
Foreman	3.05
Stefan	2.75

VI. SUMMARY

Optimized implementation of VC-1 decoder modules with SIMD operations is described for ARM Cortex–A8 processor with NEON. Required modifications in algorithm implementations are proposed for effective utilization of SIMD architecture. Brief overview of VC-1 decoder and Cortex-A8 architecture has been included for the completeness. Finally the optimization results are shown which corroborate the proposed optimization ideas.

REFERENCES

[1] Sridhar Srinivasan, Shankar L. Regunathan "An Overview of VC-1", *Visual Communications and Image processing,* Proc. Of SPIE, Vol. 5960, pp. 720-728 (2005)

[2] Sridhar Srinivasan, Pohsiang Hsu, Tom Holcomb et al., "Windows Media Video 9: overview and applications", *Signal Processing: Image Communication*, Vol. 19, Issue 9, Oct. 2004, pp. 851-875

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

[4] ARM Cortex-A8 technical reference manual. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0344h/DDI0344H_cortex_a8_r3p0_trm.pdf

[5] ARM Openmax-DL libraries [Online]. Available: http://www.design-reuse.com/news/exit/?id=14688&url=http://www.arm.com/products/esd/openmax_home.html