

# SIMD OPTIMIZATION OF THE H.264/SVC DECODER WITH EFFICIENT DATA STRUCTURE

*Joohyun Lee, Gwanggil Jeon, Sangjun Park, Taeyoung Jung, and Jechang Jeong*

Department of Electrical and Computer Engineering, Hanyang University,  
17 Haengdang, Seongdong, Seoul 133-791, Korea  
{saint81, windcap315, sjp21c, master, jjeong}@ece.hanyang.ac.kr

## ABSTRACT

H.264/scalable video coding (SVC) is a new compression technique that can adapt to various network environments and applications. However, despite its outstanding performance, H.264/SVC has considerable complexity burden on decoding, especially in inverse transform and interpolation for motion compensation. We first analyze the Joint Scalable Video Model (JSVM) decoder to identify time-consuming modules of H.264/SVC. Based on the analysis, we present the single instruction multiple data (SIMD) Optimization methods for 4×4 inverse transform and interpolation for motion compensation with an efficient data structure. On a 2.4 GHz Intel Pentium IV processor, the proposed methods reduce considerably the computation time for inverse transform and interpolation for motion compensation, compared to JSVM software.

*Index Terms*— Optimization methods, Video coding

## 1. INTRODUCTION

Recently, in order to integrate to a ubiquitous environment, video coding systems should adapt to a variety of network environments and applications. From these needs, Joint Video Team (JVT), which was formed by Motion Picture Experts Group (MPEG) and ITU Video Coding Experts Group (VCEG) jointly, has standardized the scalable extension of H.264/AVC called H.264/scalable video coding (SVC) [1].

Based on H.264/AVC coding techniques, H.264/SVC can correspond to diverse applications over heterogeneous networks with only one bit-stream using temporal, spatial, and quality scalability techniques, particularly in scenarios where system resources and network conditions are not known in advance.

The H.264/SVC also uses block-based motion compensation (MC) and transform coding structure like H.264. In addition, H.264/SVC employs various kinds of scalability techniques that provide scalable bit-streams while preserving similar coding efficiency compared with previous coding techniques [2]. The flexible coding performance of

H.264/SVC, however, causes large complexity burden, especially in inverse transform and interpolation for MC.

Since the complexity of multimedia applications is increasing, most microprocessors support parallel processing with single instruction multiple data (SIMD) architecture to enhance speed of image and video processing [3]. By using SIMD technology, such as MMX/SSE/SSE2/SSE3, multiple arithmetic operations or data can be processed in parallel simultaneously without any decrease in coding efficiency.

To optimize the H.264/SVC decoder, we use not only SIMD instruction but also an efficient data structure that avoids unnecessary rearrangement of the data. By using these algorithms, we optimize the 4×4 inverse transform and interpolation for MC. On a 2.4 GHz Intel Pentium IV processor, the proposed algorithms can reduce computation time up to 79.8 % for inverse transform and up to 81 % for interpolation in MC compared to JSVM software.

The rest of the paper is organized as follows. Overview of H.264/SVC is presented in Section 2. We describe time-consuming modules of the H.264/SVC decoder in Section 3. Our optimized H.264/SVC implementation is presented in Section 4. Experimental results and analyses are shown in Section 5. Finally, conclusions are given in Section 6.

## 2. OVERVIEW OF H.264/SVC

The scalable extension of H.264/AVC is to extend the hybrid video coding approach of H.264/AVC with a layered video coding. The general structure of the H.264/SVC encoder with three spatial layers is depicted in Fig. 1.

The input video sequences are decimated to get lower spatial resolution video in each layer. Each spatial layer follows the basic concepts of motion compensated prediction and intra prediction as in H.264/AVC, and the corresponding network abstraction layer (NAL) units contain motion information and texture data. The redundancy between adjacent layers is removed by three kinds of inter-layer prediction techniques that reuse information from lower spatial resolution (texture, motion, and residual information).

The reconstruction quality of base representation can be improved using progressive refinement slices. Moreover, the

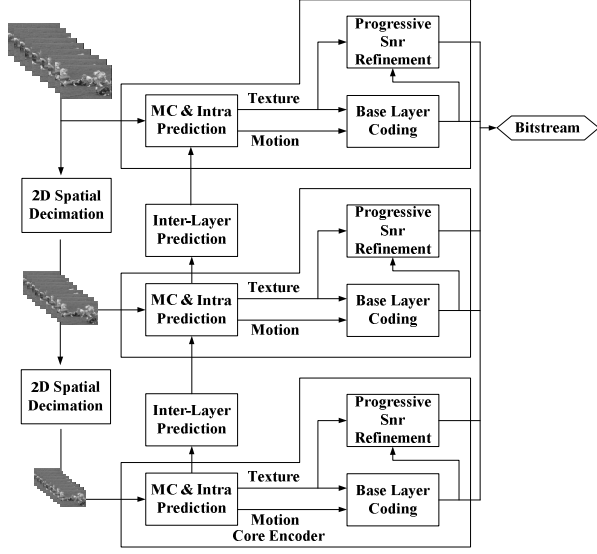


Fig. 1. General structure of the H.264/SVC encoder.

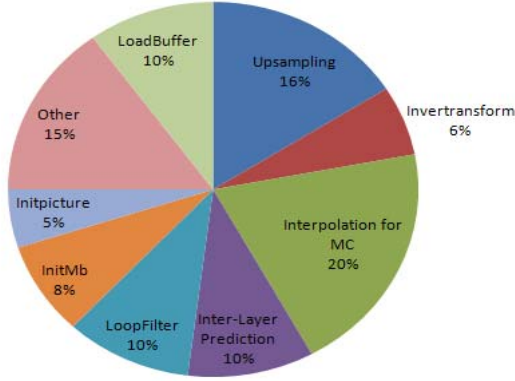


Fig. 2. Complexity description of the H.264/SVC decoder.

corresponding NAL units can be arbitrarily truncated in order to support quality scalability or flexible bit-rate adaptation. An important feature of the H.264/SVC design is that scalability is provided at a bit-stream level [4].

### 3. COMPLEXITY ANALYSIS OF THE H.264/SVC DECODER

We used the Intel VTune Performance Analyzer as the profiling tool to evaluate the software performance and obtain the complexity profile of the H.264/SVC decoder [5]. Fig. 2 shows the execution time of the H.264/SVC decoder measured on Intel Core 2 Duo 2.4 GHz Pentium IV processor with 1 GB RAM memory in Microsoft Windows XP. In the test, JSVM version 9.6 was used as the reference software of H.264/SVC with 3 layers of CIF (352×288), SD (720×480), and HD (1920×1088) resolution on IBBBBBBBP structure. We also used inter-layer predictions,

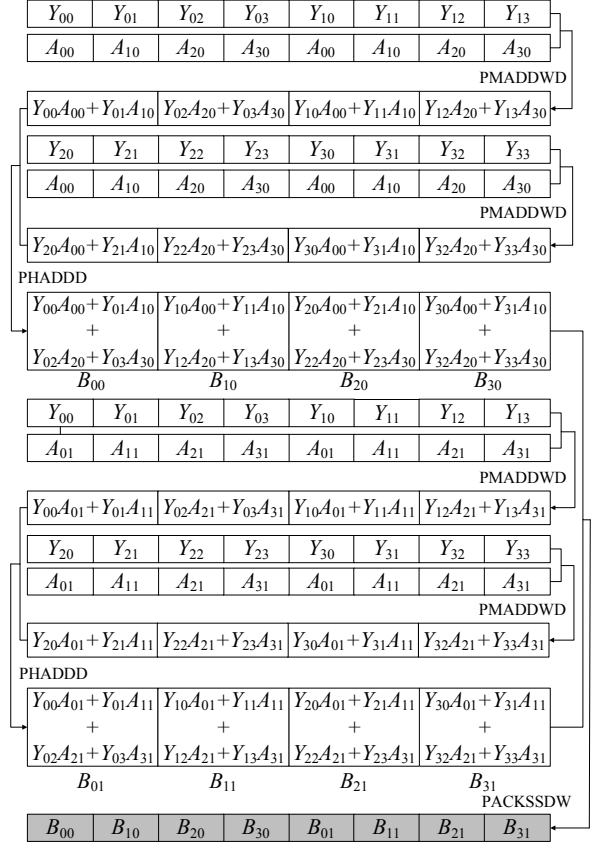


Fig. 3. Inverse transform for row direction.

context adaptive variable length coding (CAVLC), and deblocking filter.

As shown in Fig. 2, interpolation and inverse transform are intensive time-consuming modules that can be optimized by SIMD structure.

## 4. PROPOSED METHOD

In this paper, to optimize the 2-D inverse transform and interpolation for MC, we propose not only SIMD method but also efficient data structures. There is no need to further rearrange their data positions during the processing. These save unnecessary processing time compared with conventional methods [6].

### 4.1. 4×4 Inverse Transform

2-D inverse transform of H.264/SVC is defined by

$$X = A^T Y A = A^T (Y A) = A^T B \quad (1)$$

$$= \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \begin{bmatrix} y_{00} & y_{01} & y_{02} & y_{03} \\ y_{10} & y_{11} & y_{12} & y_{13} \\ y_{20} & y_{21} & y_{22} & y_{23} \\ y_{30} & y_{31} & y_{32} & y_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1/2 & -1/2 & -1 \\ 1 & -1 & -1 & 1 \\ 1/2 & -1 & 1 & -1/2 \end{bmatrix}$$

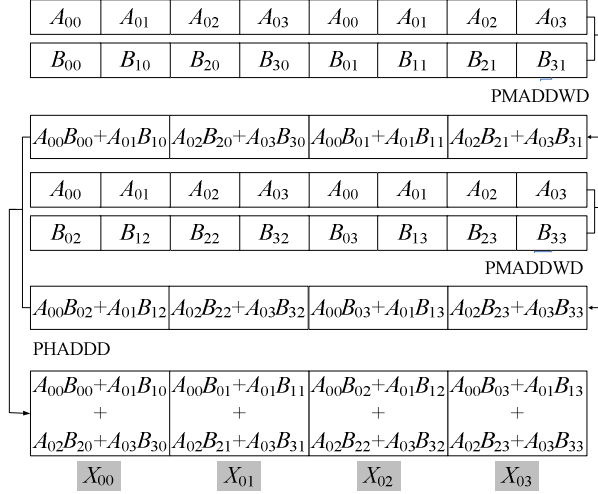


Fig. 4. Inverse transform for column direction.

In the conventional methods, data re-arrangement such as transpose and shuffle instruction is needed in order to implement (1) using SIMD instructions. However, the proposed SIMD data structure does not require any re-arrangement to implement (1). The implementation of 2-D inverse transform mainly consists of two steps using (1).

- Step 1. Multiply the last two matrices  $Y$  and  $A$  (1-D row inverse transform)
- Step 2. Multiply the first two matrices  $A^T$  and  $B$  (1-D column inverse transform)

Fig. 3 illustrates the first step of inverse transform. To avoid extra data re-arrangement, we multiply the last two matrices instead of multiplying the first two matrices.

In the process, we load the matrix  $Y$  in two 128-bit SSE registers. To process multiplication with matrix  $A$ , we load the first two columns of matrix  $A$  in another two 128-bit SSE registers. We prepare adequate manipulation for computation in advance because all elements of matrix  $A$  are constant. After first multiplication processing using PMADDWD instruction, we obtain the upper half of data of  $B$  matrix,  $B_{00}, B_{10}, B_{20}, B_{30}, B_{01}, B_{11}, B_{21},$  and  $B_{31}$ , as shown in Fig. 3. Similar processing is applied to obtain the other half of data of  $B$  matrix,  $B_{02}, B_{12}, B_{22}, B_{32}, B_{03}, B_{13}, B_{23},$  and  $B_{33}$ , respectively.

In the next step, with the results of step 1, we multiply the two matrices  $A^T$  and  $B$ . Similar to inverse transform method for row direction, we get the first row of result matrix,  $X_{00}, X_{01}, X_{02},$  and  $X_{03}$ , as shown in Fig. 4. Similar processing is applied to obtain the rest of the result matrix.

## 4.2. Interpolation for Motion Compensation

In the H.264/SVC decoder, the sub-pixel values at half sample positions are derived by applying a 6-tap filter with tap values [1, -5, 30, 20, -5, 1]. The filter coefficients are

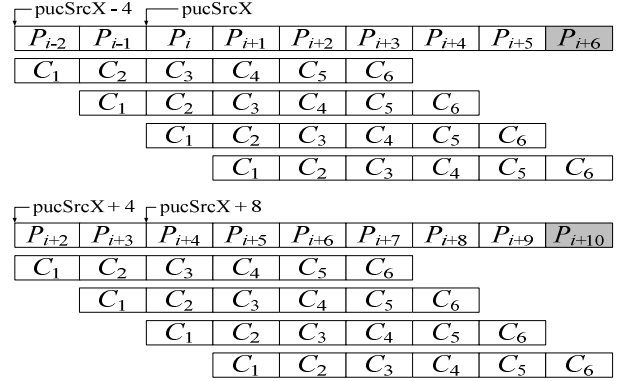


Fig. 5. FIR implementation of row direction.

constants and pre-stored. The interpolation process conducts vector product between the coefficient vector and the image data vector using 128-bit SSE registers. Since the data type of pixel is short (2 Bytes) in JSVM, eight pixels can be processed at the same time.

The scheme for row interpolation is shown in Fig. 5. Data vector is denoted as  $\text{pucSrcX}$ , and the current pixel value is denoted as  $P_i$ , the index of pixel is denoted as  $i$ , and 6-tap filter coefficients are denoted as  $c$ . Since the first two coefficients are multiplied to  $P_{i-2}$  and  $P_{i-1}$ , we load eight pixels which are shifted two pixels to the left into a 128-bit register.

Since the filter coefficients must be shifted along the row direction, we can load four different copies of the filter coefficients with different shift phases, and then perform the PMADDWD instruction on data vector and the four copies of the filter coefficients to get the first four pixels. Note that while calculating the fourth pixel, we shift the data vector by one pixel to the left since the coefficient  $C_6$  (equals to 1) cannot match the data vector. This is why we colored the pixel in gray to distinguish it from other pixels. The remaining four pixels can be interpolated as shown in the bottom of Fig. 5. We first use PMADDWD instruction on data vector with coefficient vector. After we use PHADD instruction to add the result in horizontal direction, we use PHADD instruction again on the previous result. Then, dividing four values by the sum of the coefficients, we obtain four interpolated pixels. Similar processing is applied to obtain the other four pixels. Each of the four interpolated pixels is packed by PACKSSDW instruction to fit in 128-bit register, which includes eight interpolated pixels. By using these two diagrams, we manage the eight pixels simultaneously.

The scheme for column interpolation is shown in Fig. 6. To implement interpolation for column direction, we use not only SIMD methods but also efficient data structures. There is no need to further rearrange and shift phases during the processing. To process the eight pixels simultaneously six 128-bit registers are loaded. First register is loaded with data vector that was shifted up by two rows in a similar way as in the row direction interpolation. And second register is

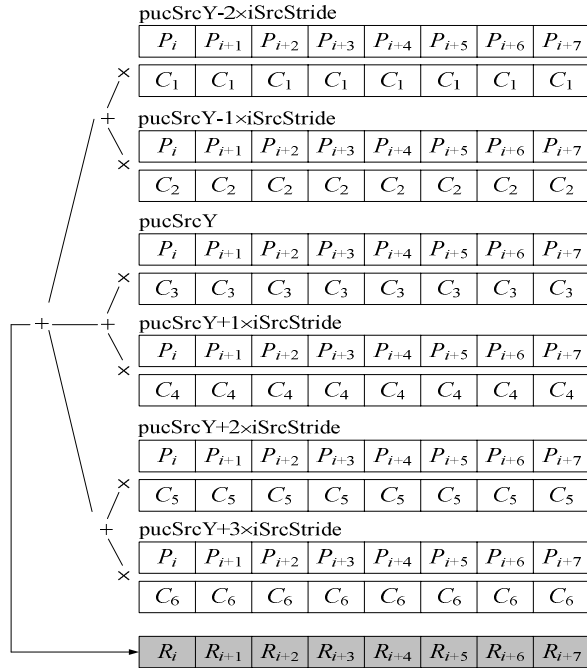


Fig. 6. FIR implementation of column direction.

loaded with data vector, which was shifted down by one row. The other registers are shown in Fig. 6. Each of the six registers is loaded with the same six coefficients. Finally, we get the interpolated values,  $R_i$ ,  $R_{i+1}$ ,  $R_{i+2}$ ,  $R_{i+3}$ ,  $R_{i+4}$ ,  $R_{i+5}$ ,  $R_{i+6}$ , and  $R_{i+7}$ , respectively.

## 5. EXPERIMENTAL RESULTS

All experiments were carried out with Microsoft Windows XP on a 2.4 GHz Intel Pentium IV processor with 1 GB RAM memory. We compared our methods with implementations of the JSVM reference software. In these simulations, we checked the average execution time of  $4 \times 4$  inverse transforms and interpolation iterated 32,640 times with HD sequences of  $1920 \times 1088$  in each implementation.

Table 1 compares the execution time of  $4 \times 4$  inverse transform for each implementation. We also measured execution time using SIMD instruction with original JSVM structure. From the results, Table 1 verified that it is unfortunately not proper to keep the structure of the original JSVM code for implementing SIMD structure. In addition, simulation results describe that our algorithm achieves computation time saving up to 79.84 % for inverse transform compared to the JSVM software, as opposed original JSVM structure with SIMD instruction.

Table 2 compares the execution time of interpolation for MC. As can be seen in Table 2, the proposed method clearly reduces the execution time by about 81.01 % compared to that of JSVM code. In other words, the proposed method outperforms the conventional JSVM code by 5.3 times in computation complexity.

Table 1. Execution time of  $4 \times 4$  Inverse Transforms

Comparison	Time (ms)	Time saving (%)
Methods		
JSVM code	0.749	-
JSVM structure with SIMD instruction	0.252	66.35
Proposed method	0.151	79.84

Table 2. Execution time of Interpolation for MC

Comparison	Time (ms)	Time saving (%)
Methods		
JSVM code	79	-
Proposed method	15	81.01

## 6. CONCLUSION

As compression techniques become more and more complex, processes need much higher computational complexity than most conventional methods. H.264/SVC includes a number of new features that can provide scalable bit-streams while preserving similar coding efficiency compared with previous coding techniques. However, these features require much more computation complexity than most existing standards such as MPEG-4 and H.264.

In this paper, we proposed SIMD-based optimization algorithms for  $4 \times 4$  inverse transform and interpolation in MC with efficient data structure. The experimental results showed that the proposed  $4 \times 4$  inverse transform and interpolation algorithms were 5 times and 5.3 times faster than JSVM code of H.264/SVC, respectively.

## Acknowledgments

This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD) (KRF-2004-005-D00164)

## 7. REFERENCES

- [1] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the Scalable H.264/MPEG4-AVC Extension," in *Proc. ICIP 2006*, Atlanta, GA, USA, October 2006.
- [2] H. Schwarz, D. Marpe, and T. Wiegand, "Basic concepts for supporting spatial and SNR scalability in the scalable H.264/MPEG4-AVC extension," in *Proc. IWSSIP 2005*, Chalkida, Greece, September 2005.
- [3] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 2: Instruction Set Reference A-Z*, 253666-022US, November 2006.
- [4] J. Reichel, H. Schwarz, and M. Wien, "Joint Scalable Video Model JSVM-9," Doc. JVT-V202, Marrakech, Morocco, January 2007.
- [5] Intel Corporation, Intel VTune Performance Analyzer, <http://www.intel.com/software/products/vtune/>.
- [6] Y.K. Chen, Eric Q. Li, X. Zhou, and St. Ge, "Implementation of H.264 Encoder and Decoder on Personal Computers," in *Journal of Visual Communications and Image Representations*, vol. 17, no. 2, pp. 509-532, April 2006.