An Efficient SIMD-based Quarter-Pixel Interpolation Method for H.264/AVC

Chae-Bong Sohn, and Hye-Jeong Cho

VIA-Multimedia Center, Kwangwoon University, 447-1, Wolgye-Dong, Nowon-Gu, 139-701, Seoul, Korea

Summary
Many media processors which support SIMD (Single Instruction Multiple Data) instructions have been widely used for digital signal processing and multimedia applications. In particular, a set of SIMD instructions is very effective in video applications which require both simple operations and short data types, mostly 8-bit or 16-bit samples. This paper proposes the fast quarter-pixel interpolation of the H.264/AVC, which can be implemented with the Intel SIMD instructions. The implementation of the proposed method is approximately six times faster than that of the JM software for the H.264/AVC quarter-pixel interpolation operation which needs multiple bi-linear and 6-tap filtering.

Key words:
Media processor, SIMD, H.264/AVC, quarter-pixel interpolation, video codec

1. Introduction

H.264/AVC, which is the latest video compression standard, has surfaced as one of the essential parts in new media such as Korea's T-DMB and S-DMB, Europe's DVB-H, and the Blu-ray Disc, the new generation DVD. H.264/AVC is superior to prior video compression standards such as H.261, H.263, MPEG-1, MPEG-2, and MPEG-4 in compression efficiency. In particular, H.264/AVC can have the same image quality to the MPEG-2 with only 50% of its bitrate. However, its advanced video compression technology to obtain such a high compression rates requires high computational complexity [1].

One of the techniques of the H.264/AVC to enhance the coding efficiency is the quarter-pixel motion estimation/compensation, which adopts the 6-tap FIR and bi-linear filters. The 6-tap filter's coefficient is (1, -5, 20, -5, 1) for interpolations of luminance and chrominance, and it takes about 25% of the entire decoding time at a decoder side [2].

Most of the microprocessors available today support multimedia instructions to accelerate their application programs. For instance, Intel's Xeon, Pentium IV, and AMD's Athlon64 all have the MMX, SSE, and SSE2 instructions of the SIMD (Single Instruction Multiple Data) model. These instructions can process multiple data in parallel in a single instruction. However, there are more cases in which it takes more time to pack or unpack data in their registers properly than to do computations in implementation with the SIMD instructions [3][4].

In this paper, an optimized SIMD algorithm for the interpolation of the H.264/AVC was proposed in the Intel processor supporting SIMD instructions. The proposed algorithm is based on the use of symmetry of the 6-tap FIR filter coefficient for parallel computation in the data level, resulting in significant reduction of multiplications.

2. Intel SIMD Technology

SIMD technology was introduced in the IA-32 architecture based on the MMX technology. MMX technology allows SIMD computations to be performed on packed byte, word, and double-word integers. The integers belong to a set of eight 64-bit registers called MMX registers. The Pentium Ⅲ processor extended the SIMD computation model by employing the Streaming SIMD Extensions (SSE). SSE enables SIMD computations to be performed on operands that support four packed single-precision floating-point data elements. The operands can be loaded in a memory or in a set of eight 128-bit XMM registers. SSE also extended SIMD computational capability by adding additional 64-bit MMX instructions. Fig. 1 shows a typical SIMD computation in which two sets of four packed data elements (X1, X2, X3, and X4, and Y1, Y2, Y3, and Y4) are operated in parallel, respectively. In the Pentium 4 processor, the SIMD computation model was enhanced by introducing Streaming SIMD Extensions 2 (SSE2) and Streaming SIMD Extensions 3 (SSE3). SSE2 is executed with operands loaded in either memory or in the XMM registers. The SIMD computation capability was improved to process packed double-precision floating-point data elements and 128-bit packed integers. The SSE supports 144 instructions, operating on two packed double-precision floating-point data elements or on 16 packed byte, 8 packed word, 4 double word, and 2 quad word integers. In the SSE3, SSE and SSE2 are extended by providing additional 13 instructions that can accelerate application performance in specific areas. These include

Manuscript received November 5, 2006.
Manuscript revised November 25, 2006.

video processing, complex arithmetic, and thread synchronization. SSE3 complements SSE and SSE2 with additional instructions that process SIMD data asymmetrically, facilitate horizontal computation, and avoid loading cache line splits [5].

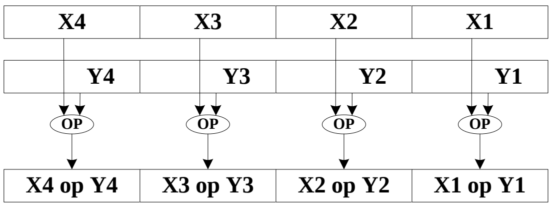


Fig. 1 Typical SIMD Operations.

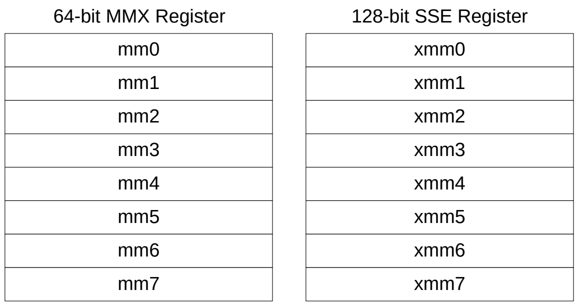


Fig. 2 SIMD instruction registers.

3. Overview of Quarter-Pixel Interpolation

In the H.264/AVC standard, a 6-tap FIR filter is at first applied for half-pixel interpolation in luminance components. Its coefficients are (1, -5, 20, 20, -5, -1) which can be considered as a low-pass filter. After that, a bi-linear filter is used for quarter-pixel estimation. For example, the half-pixels *aa, bb, b, cc, dd, h, j, m, ee, ff, s, gg,* and *hh,* as shown in Fig. 3, can be calculated as follows.

$$
\begin{aligned}
&b = ((E - 5*F + 20*G + 20*H - 5*I + J) +16)/32\\
&h = ((A - 5*C + 20*G + 20*M - 5*R + T) +16)/32\\
&m = ((B - 5*D + 20*H + 20*N - 5*S + U) +16)/32\\
&s = ((K - 5*L + 20*M + 20*N - 5*P + Q) +16)/32\\
&j = ((cc - 5*dd + 20*h1 + 20*m1 - 5*ee + ff) +512)/1024\\
&\quad \text{or } ((aa - 5*bb + 20*b1 + 20*s1 - 5*gg + hh) +512)/1024
\end{aligned} \tag{1}
$$

For interpolation at the half-pixel *j*, interpolations at the neighboring six points, *cc*, *dd*, *ee*, and *ff* or *aa, bb, gg*, and *hh* should be available in advance. On the other hand, for quarter-pixels *a, c, d, e, f, g, i, k, n, p* and *q*, their interpolations are denoted by

$$
\begin{aligned}
&a = (G+b+1)>>1 && c = (H+b+1)>>1\\
&d = (G+h+1)>>1 && n = (M+h+1)>>1\\
&f = (b+j+1)>>1 && i = (h+j+1)>>1\\
&k = (j+m+1)>>1 && q = (j+s+1)>>1\\
&e = (b+h+1)>>1 && g = (b+m+1)>>1\\
&p = (h+s+1)>>1 && r = (m+s+1)>>1
\end{aligned} \tag{2}
$$

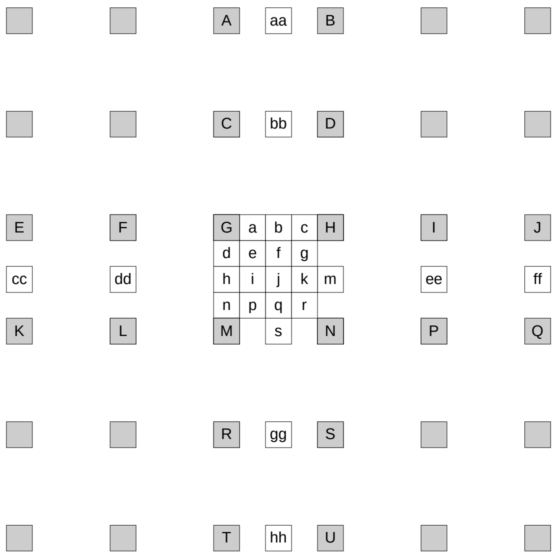


Fig. 3 Quarter-pixel interpolation for luminance.

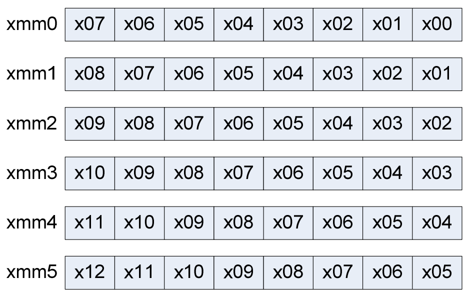
All the interpolation points can be classified into layer 0~3, according to their mutual dependencies. Points in the layer-0 represent the integer pixels that are originally available as the input. Points at which the interpolation depends on the layer-0 pixels are classified as the layer-1. In the same way, layer-2 and 3 can be defined. Obviously, the input pixels *A*, *B*, *C*, *D*,…, *U* belong to the layer-0 and the half-pixels, *b*, *h*, *m*, and *s* are grouped to the layer-1. The quarter-pixels, *a*, *c*, *d*, *e*, *g*, *n*, *p*, *r*, and *j* are considered to be in the layer-2. Finally, *f*, *i*, *k*, and *q* longs to the layer-3. Theoretically, all the three layers of computations should be performed for interpolation in motion estimation (At an encoder with 1/4 pixel accuracy). On the other hand, the layer-3 computation is necessary in motion compensation (decoder) for a full compliant to the H.264/AVC baseline specification, depending on motion vector (MV) for each block [6],[7].

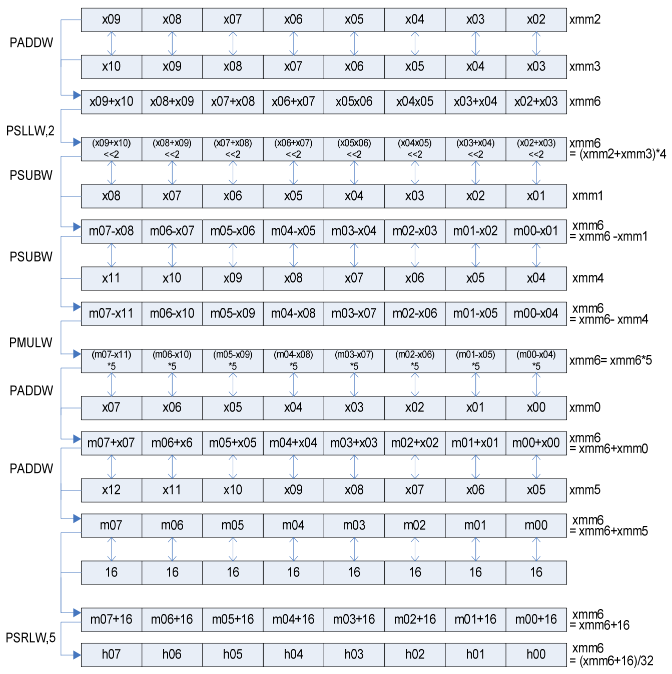
4. Proposed SIMD-based Quarter-Pixel Interpolation Method

In this paper, a new computation for the quarter-pixel interpolation of the H.264/AVC is proposed for SIMD architecture. The proposed algorithm is derived by minimizing the number of memory access. The formulas to compute half-pixel interpolations are proposed by using the symmetry of the 6-tap FIR filter coefficients, resulting in significant reduction of the multiplications.

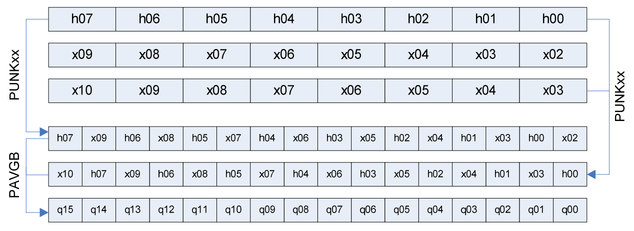
$$\begin{aligned}
b &= ((E-5*F+20*G+20*H-5*I+J)+16)/32 \\
&= (((E+J)-(F+I)*5+(G+H)*20)+16)/32 \\
&= (((G+H)*4-F-I)*5+E+J+16)/32 \\
h &= ((A-5*C+20*G+20*M-5*R+T)+16)/32 \\
&= (((A+T)-(C+R)*5+(G+M)*20)+16)/32 \\
&= (((G+M)*4-C-R)*5+A+T+16)/32
\end{aligned} \tag{3}$$

Fig.4 shows of the proposed quarter-pixel interpolations in the horizontal direction with SIMD instructions. In order to carry out interpolations using SIMD instructions, the pixel data should be loaded in registers, as shown in Fig. 4(a). Those data could exceed the bites in computing the 6-tap filtering, which called for converting the byte data into word units with the PUNPCKxx instructions [5]. Fig. 4(b) shows the 6-tap filtering to compute the equation (3) for eight half-pixel values. In Fig. 4(c), the half-pixel and integer-pixel values are computed with the PAVGB instruction along with quarter-pixel ones. Then the PUNPCKxx instructions are adopted to store the integer-, half- and quarter-pixel values at the proper place of the register.


xmm0 x07 x06 x05 x04 x03 x02 x01 x00
xmm1 x08 x07 x06 x05 x04 x03 x02 x01
xmm2 x09 x08 x07 x06 x05 x04 x03 x02
xmm3 x10 x09 x08 x07 x06 x05 x04 x03
xmm4 x11 x10 x09 x08 x07 x06 x05 x04
xmm5 x12 x11 x10 x09 x08 x07 x06 x05
(a) Data arrangement for the horizontal interpolations



(b) A method to make half-pixel by using the 6-tap filtering following the equation (3)



(c) A method to make quarter-pixel by using half- and integer-pixel

Fig. 4 An SIMD data flow for the horizontal quarter-pixel interpolations (a)Data arrangement for the horizontal interpolations, (b)half-pixel interpolation by using the 6-tap filtering, (c)quarter-pixel interpolation with half- and integer-pixels. (h: Half-pixel value, q: Quarter-pixel value).

The quarter-pixel interpolations for the second vertical direction are obtained by computing the integer- and half-pixel information out of the completed information after the quarter-pixel interpolations for horizontal direction. Fig. 5 shows a case of vertical quarter-pixel interpolation using SIMD along with the way to store the integer- and half-pixel from the horizontal direction in the register. If you perform the same computation as Fig. 4(b) in the stored register, you can get half-pixel for the vertical direction. The quarter-pixel can be obtained by performing Fig. 4(c) in the horizontal direction. Interpolations for both the directions will leave quarter-pixel for the diagonal element and half-pixel just as shown in Fig. 6(a). Fig. 6(b) presents a structure to compute the values for the half-pixel and diagonal elements, which can be obtained by using the PUNPCKxx and PAVGB instructions.

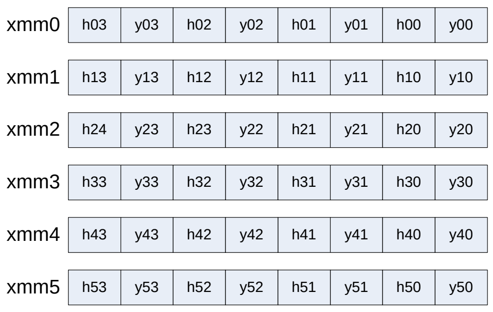
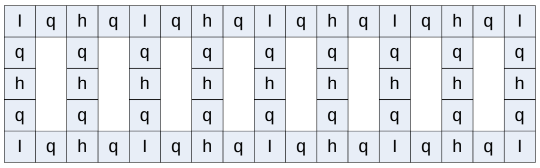
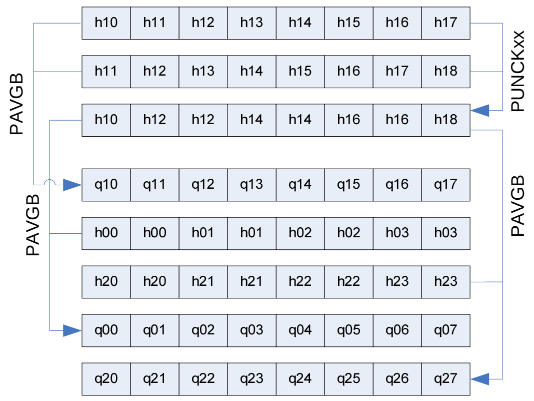

xmm0 h03 y03 h02 y02 h01 y01 h00 y00
xmm1 h13 y13 h12 y12 h11 y11 h10 y10
xmm2 h24 y23 h23 y22 h21 y21 h20 y20
xmm3 h33 y33 h32 y32 h31 y31 h30 y30
xmm4 h43 y43 h42 y42 h41 y41 h40 y40
xmm5 h53 y53 h52 y52 h51 y51 h50 y50

Fig. 5 Data arrangement to implement vertical quarter-pixel interpolations. h : Half-pixel value y : integer-pixel value



(a) Half- and quarter-pixel created after the horizontal and vertical interpolations


h10 h11 h12 h13 h14 h15 h16 h17
h11 h12 h13 h14 h15 h16 h17 h18
h10 h12 h12 h14 h14 h16 h16 h18
q10 q11 q12 q13 q14 q15 q16 q17
h00 h00 h01 h01 h02 h02 h03 h03
h20 h20 h21 h21 h22 h22 h23 h23
q00 q01 q02 q03 q04 q05 q06 q07
q20 q21 q22 q23 q24 q25 q26 q27
PAVGB
PUNCKxx
PAVGB
PAVGB

(b) A method to make quarter-pixel for the diagonal element and half-pixel.

Fig. 6 Data arrangement to implement vertical quarter-pixel interpolations. (a) Half- and quarter-pixel created after the horizontal and vertical interpolations (b) A method to make quarter-pixel for the diagonal element and half-pixel. I : integer-pixel value, h: Half-pixel value, q: Quarter-pixel value.

5. Experiment Results

The processor used in this experiment is the Intel Pentium IV (2.4GHz), supporting 64-bit MMX and 128-bit SSE2 instructions. VTune performance analyzer was used for performance profiling and actual execution time is used for performance evaluation [8].

Table 1 presents the numbers of instruction clock ticks of the JM and the proposed methods for the H.264/AVC interpolation. The proposed SIMD-based interpolation gives benefit for loading and storing of data, multiplication, calculation and logic computation. However, extra computation like packing and unpacking of data is required for the proposed algorithm but it is not significant in overall performance.

Table 1: Comparison of clock ticks for each module

Method / Instruction	JM	SIMD	
		Horizontal Direction	Vertical Direction
Memory Move	2304	398	178
Logic and Arithmetic	768	146	166
Multiply	1536	72	72
Data pack/unpack	0	224	84
Total Clock tick	4608	840	500
Speedup	1.0000	5.4857	9.216

Table 2 shows the comparison of actual execution time of quarter-pixel interpolation module of the reference encoder, and proposed algorithm using the SSE2(128-bit). We created 100,000 8x8 block generated with a random number generator, and the average execution time is measured with the generated data. Each input video used in our experiment consists of 300 frames. As shown in Table 3, the proposed method is about 6 times faster than the JM reference software for an H.264/AVC quarter-pixel interpolation.

Table 2: Comparison of excution time and performance of quarter-pixel interpolation for the 8x8 block

speedup / method	Time (sec)		Speedup	
	Horizontal	Vertical	Horizontal	Vertical
JM	0.292839	0.329502	1	1
SIMD	0.055824	0.036922	5.2458	8.9243

According to Table 1, the proposed algorithm would be approximately 5.4 times and 9.2 times faster than the JM for each direction module. Table 2 shows that the actual execution time is 5.2 and 8.9 times faster than the JM for each directional module. As shown in Table 3, the execution time is measured in terms of image size. Each input video used in our experiment consists of 300 frames. As shown in Table 3, the proposed method is about 6 times faster than the JM reference software for an H.264/AVC quarter-pixel interpolation.

Table 3: Comparison of implementation time and performance of quarter-pixel interpolation for video sequence

speedup / method	Time (sec)		Speedup
	JM	SIMD	
QCIF(176x144)	0.03095	0.00496	6.235
QVGA(320x240)	0.09366	0.01417	6.608
CIF(352x288)	0.13725	0.01862	7.370

6. Conclusions

This paper presents a optimized method to quarter-pixel interpolations used in H.264/AVC by using SIMD instructions. The proposed method resulted in about 6 times of performance improvement in quarter-pixel interpolations compared to the JM. From our experiments, We have shown that proposed method is suitable for quarter-pixel interpolations at H.264/AVC codecs in SIMD processors. With increasingly more real-time applications in H.264 encoders, it is possible that the proposed SIMD based quarter-pixel interpolation will be indispensable.

Acknowledgments

The present Research has been conducted by the Research Grant of Kwangwoon University in 2006

References

[1] J. Ostermann, J. Bormans, PList, D. Marpe, M. Narroshke, F. Pereira, T. Stockhammer, and T. Wedi, “Video coding with H.264/AVC: tool, performance, and complexity,” IEEE Circuit and Syst. Mag. Vol. 4, pp. 7-28, 2004

[2] M. Horowitz, A. Joch, F.kossentini, and A. Hallapuro, “H.264/AVC baseline profile decoder complexity analysis,” *IEEE Trans Circ and Syst. Video Tech,* vol. 13, n0 7, pp 715-727, 2003

[3] Richard G., "The Software Optimization Cookbook", Intel Press, 2002

[4] Intel Corp., "Intel Pentium 4 and Intel Xeon Processor Optimization - Reference Manual", Order Number: 248966-05, 2002

[5] Intel Corp., " IA-32 Intel® Architecture Optimization Reference Manual", Order Number: 248966-013US, April 2006

[6] T. Wiegand, G.J. Sullivan, G. Bjontegaard and A. Lutha, “Overview of the H.264/AVC Video Coding Standard,” *IEEE Trans. Circuits Syst. Video Technol*, vol 13, no.7, pp.560-576, July 2003

[7] Wen-Nung Lie, Han-Ching Yeh, Lin, T.C.-I, Chien-Fa Chen ”Hardware-efficient computing architecture for motion compensation interpolation in H.264 video coding” *IEEE international Symposium on Circuit and system,* vol. 3, pp.2136-2139, May 2005

[8] Intel Corp., Intel® VTune^{TM}Performance Analyzer, Version 7.0, 2003

Chae-Bong Sohn received the B.S., M.S., and Ph.D. degree in electronics engineering form Kwangwoon University, Seoul, Korea in 1993, 1995, and 2006, respectively. From 2000 to 2005, he was a full-time lecturer, Hanyang Women’s College, Department of Internet Informations Engineering. He is currently a full-time lecturer, Department of Electronics and Communications Engineering. Kwangwoon University, Seoul, Korea. His research interests include image compression, transcoding, digital broadcasting systems.

Hye-Jeong Cho received the B.S. degree in 2004 form the Department of Internet Informations Engineering, Hanyang Women’s College, Seoul, Korea. She is currently pursuing the joint M.S. and Ph.D. degree in electronics engineering from Kwangwoon University, Seoul, Korea. Her research interests include video coding, digital broadcasting systems.