Multimedia: An SIMD – Based Efficient 4x4 2 D Transform Method

Sang-Jun Yu, Chae-Bong Sohn, Seoung-Jun Oh, and Chang-Beom Ahn

VIA-Multimedia Center, Kwangwoon University, 447-1, Wolgye-Dong,
Nowon-Gu, 139-701, Seoul Korea
{yusj1004, bongbong, sjoh, cbahn}@viame.re.kr
http://www.viame.re.kr

Abstract. In this paper, we present an efficient scheme for the computation of 4x4 integer transform using SIMD instructions, which can be applied to discrete cosine transform (DCT) as well as Hadamard transform (HT) in MPEG-4 AVC/H.264, a video compression scheme for DMB. Even though it is designed for 64-bits SIMD operations, our method can easily be extended to 128-bits SIMD operations. On a 2.4G (B) Intel Pentium IV system, the proposed method can obtain 4.34x and 2.6x better performances for DCT and HT, respectively, than a 4x4 integer transform technique in an H.264 reference codec using 64-bits SIMD operations. We can still have 6.77x and 3.98x better performances using 128-bits SIMD operations, respectively.

1 Introduction

Transform coding is at the heart of the majority of video coding systems and standards such as H.26x and MPEG. Spatial image data can be transformed into a different representation, the transform domain since spatial image data are inherently difficult to compress without adversely affecting image quality: neighboring samples are highly correlated and the energy tends to be evenly distributed across the image. There are several desirable properties of a transform for compressions. It should compact the energy in the image into a small number of significant values, decorrelate the data and be suitable for practical implementation in software and hardware. The forward and inverse transforms are commonly used in 1D (Dimension) or 2D forms for image and video compression. The 1D version transforms a 1D array of samples into an a 1D array of coefficients, whereas the 2D version transforms a 2D array (block) of samples into a block of coefficients [1][2].

Most modern microprocessors have multimedia instructions to facilitate multimedia applications. For example, the single-instruction-multiple-data (SIMD) execution model was introduced in Intel architectures. MMX (Multimedia Extension), SSE (Streaming SIMD Extension) and SSE2 technologies can execute several computations in parallel with a single instruction. These instructions can make parallel processing of several data at the same time. In general, better performance can be achieved if the data pre-arranged for SIMD computation. However, this may not always be possible. Even if, referencing unaligned SIMD register data can incur a performance

O. Gervasi et al. (Eds.): ICCSA 2005, LNCS 3480, pp. 166 – 175, 2005.

penalty due to accesses to physical memory [3]. Also, when applying an SIMD operation, it may be happened that data packing/unpacking operation takes more time than arithmetic operations.

In this paper, we present an optimized SIMD method to carry out 2D DCT with an integer transform for a block of 4x4 in the processor supporting SIMD operation. The proposed method is multiplier free and can improve the speed by reducing the number of operations such as store, move, and load. Those operations dominate in 4x4 transform relatively to packing/unpacking process.

2 Transform Coding in H.264

Instead of using DCT/IDCT (Inverse DCT), MPEG-4 AVC/H.264 adopted as a video data compression technology in DMB (Digital Multimedia Broadcasting) service uses a 4x4 integer transform to convert spatial-domain signals into frequency-domain and vice versa. The "baseline" profile of H.264 uses three transforms depending on the type of residual data that is to bi coded: a transform for the 4x4 array of luma(luminance) DC(Direct Current) coefficients in intra macroblocks (predicted in 16x16 mode), a transform for the 2x2 array of chroma(chrominance) DC coefficients(in any macroblock) and a transform for all other 4x4 blocks in the residual data. Data within a macroblock are transmitted in the order shown in Fig.1. If the macroblock is coded in 16x16 Intra mode, then the block labeled "-1" is transmitted first, containing the DC coefficient of each 4x4 luma block. Next, the luma residual blocks "0~15" are transmitted in the order shown (with the DC coefficient set to zero in a 16x16 Intra macroblock). Blocks "16" and "17" contain a 2x2 array of DC coefficients from the Cb and Cr chroma components respectively. Finally, Chroma residual blocks "18~25" (with zero DC coefficients) are sent [4]-[6].

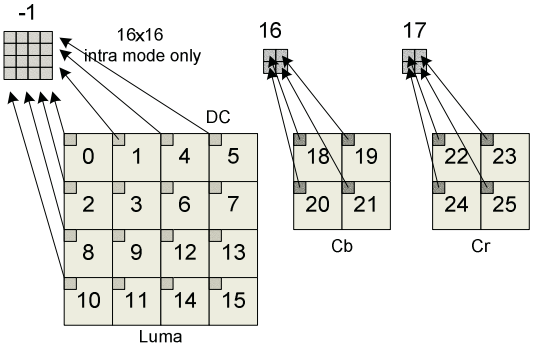


Fig. 1. Scanning order of residual blocks within macroblock

There are two types of 4x4 transforms for the residual coding in H.264. The one is DCT, and the other is HT [4].

2.1 Development from the 4x4 DCT

The 4x4 DCT of an input array is given by (1):

$$
\begin{aligned}
Y &= AXA^T \\
&= \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix}
\end{aligned} \tag{1}
$$

where, $a = 1/2$, $b = \sqrt{1/2}\cos\left(\pi/8\right)$, $c = \sqrt{1/2}\cos\left(3\pi/8\right)$.

This matrix multiplication can be factorized to the following equivalent form (2):

$$
\begin{aligned}
Y &= (CXC^T) \otimes E \\
&= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & d & -d & -1 \\ 1 & -1 & -1 & 1 \\ d & -1 & 1 & -d \end{bmatrix} \begin{bmatrix} & X & \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & d \\ 1 & d & -1 & -1 \\ 1 & -d & -1 & 1 \\ 1 & -1 & 1 & -d \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix}
\end{aligned} \tag{2}
$$

where CXC^T is a core 2D transform. E is a matrix of scaling factors and the symbol $\otimes$ indicates that each element of CXC^T is multiplied by the scaling factor in the same position in matrix E. The constant d is $c/b \approx 0.414$.

To simplify the implementation of the transform d is approximated by 0.5. To ensure that the transform remains orthogonal, b also needs to be modified, so that $a = 1/2$, $b = \sqrt{2/5}$, $d = 1/2$.

The final forward transform becomes (3):

$$
\begin{aligned}
Y &= (CXC^T) \otimes E \\
&= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} & X & \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \\ a^2 & ab/2 & a^2 & ab/2 \\ ab/2 & b^2/4 & ab/2 & b^2/4 \end{bmatrix}
\end{aligned} \tag{3}
$$

which is an approximation to the 4x4 DCT. Because of the change to factors d and b, the output of the new transform will not be identical to the 4x4 DCT [2]. Equation (3) shows the mathematical form of the forward integer transform in H.264 standard, where CXC^T is a core 2-D transform which can be calculated using matrix multiplica-

tions. Since both the first and the third matrices have only constant coefficients of "±1" and "±2", it is possible to implement the calculation by using only additions, subtractions and shifts. This "multiply-free" method is quit efficient and, thus, has been implemented in the H.264 reference codec [6].

2.2 4 x 4 Luma DC Coefficients Transform (HT)

In H.264, Hadamard transform is applied to the luminance DC terms in 16 x16 intra prediction mode. The transform matrix is shown in (4) and its fast implementation is shown in Fig. 5(a). HT seems to be a simplified version of (3) with replacing the coefficient 2 by 1. The inverse HT has the same form as (4) since the transform matrix for HT is orthogonal and symmetric [7][8].

$$Y_D = \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} \\ W_D \\ \\ \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \right) \tag{4}$$

where W_D is the block of 4x4 DC coefficients and Y_D is the block after transformation. Each output coefficient $Y_{D(i,j)}$ is divided by 2 with rounding.

3 Representative SIMD-Based Schemes

One way for speed-up is to implement a parallel method on a parallel architecture such as an SIMD machine.

Figure 2 shows a typical SIMD computation. Two sets of four packed data elements (X1, X2, X3, X4, and Y1, Y2, Y3, Y4) are operated in parallel, with the same operation being performed on each corresponding pair of data elements (X1 and Y1, X2 and Y2, X3 and Y3, X4 and Y4). The results of four parallel computations are sorted as a set of four packed data elements. Thus, we can achieve speed-up on the SIMD machine in the areas of graphics, speech recognition, image/video processing, and other scientific applications [9].

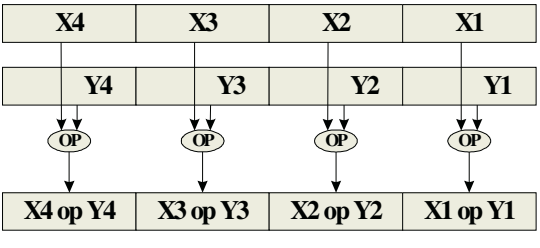


Fig. 2. Typical SIMD Operation

There exist three types of integer transform methods with SIMD operation: a 2D matrix multiplication method, a butterfly expression method and a parallel processing with four matrix [10]-[12].

In the 2D matrix multiplication method separability, which is an important property of the 2-D DCT, is used. Using that property 2-D DCT can be obtained by first performing 1-D DCTs of the rows of x_{ij} followed by 1-D DCTs of the columns of, where x_{ij} and c_{ij} are the (i,j)th elements in matrix X and C in (5), respectively [7].

$$XC = \begin{bmatrix} x_{00} & x_{01} & x_{02} & x_{03} \\ x_{10} & x_{11} & x_{12} & x_{13} \\ x_{20} & x_{21} & x_{22} & x_{23} \\ x_{30} & x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \tag{5}$$

If there is no multiplier in a processor when the 2D matrix multiplication method is used, it takes too much time on computation for a transform since a multiplication operation must be replaced with other several operations.

The second method is code rescheduling to exploit parallelism in the 2D DCT. Parallelism in the 2D DCT approached from two directions as shown in Fig. 3: within a single 4x4 DCT and among four 4x4 DCTs. While data are accessed by rows within the matrix in the former, data from the four matrices are interleaved to enable efficient use of the MMX instructions in the latter. The single DCT approach has some disadvantages; 1) The input matrix must be transposed in order to operate on several rows in parallel, and 2) packed shift instructions must be used to prevent overflow. The four DCTs approach has also some problems as following; 1) The input data from the four matrices must be interleaved before computing transform, and 2) more temporal use of memory is needed since it requires more register usage [8].

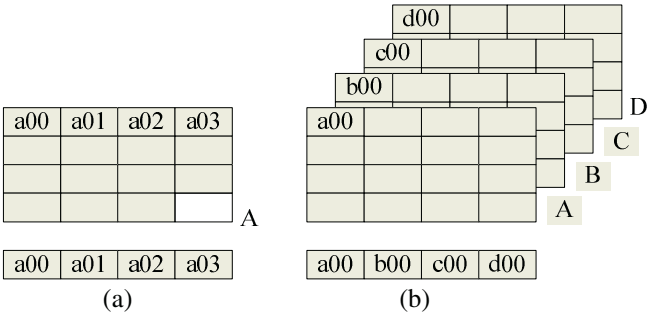
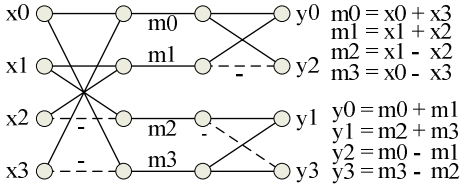
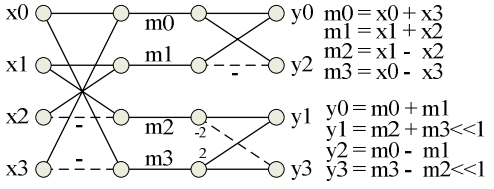


Fig. 3. Single DCT vs. Four DCTs: parallelism (a) within single matrix and (b) among four matrices [11]

The third method is to use the butterfly expression shown in Fig. 4. Even though it is easy to implement, it is hard to further optimize parallel processing in SIMD instructions because the operations on each operand are different.



(a) Butterfly Expression of the Hadamard Transform



(b) Butterfly Expression of the Integer Transform

Fig. 4. Butterfly expression of the Transform in H.264

4 An SIMD-Based 4x4 2D Transform Method

As we mention before, each element of the DCT matrix in H.264 takes one of four values: ± 1 and ± 2 [6]. Therefore, we can compute DCT coefficients without any multiplication, that is, need only addition, subtraction and shift commands to calculate them. Zhou and his colleagues proposed a new method [3]. They showed that the original form of the integer transform was the 4x4 matrix multiplications, which could be implemented by using SIMD instructions, providing better performance than the multiplication-free method implemented in the H.264 reference code.

Equation (5) shows a typical 4x4 matrix multiplication. On the other hand, the butterfly expression has the multiplication-free method implemented in the H.264 reference code. Equation (5) shows a typical 4x4 matrix multiplication. On the other hand, the butterfly expression has the multiplication-free property. However, that expression may be hard to be implemented in parallel form using the SIMD instructions as mentioned before because the operations required in the butterfly are not identical.

To solve the problems in Zhou's and the butterfly expression, a transpose process is taken for 4x4 input matrix before 1D DCT, and the butterfly operation is applied for four input elements, that is, a row vector, simultaneously to utilize a series of SIMD instructions. From this point we assume that the 64-bits operations are used. The transpose process followed by butterfly operations in parallel applied in horizontal and vertical directions makes 2D DCT for the 4x4 input as schematically shown in Fig. 5.

The proposed method consists of four steps: the transpose step using the SIMD instructions shown in Fig. 6, the butterfly step for the row-direction 1D 4x4 integer transform using the series of SIMD operations shown in Fig. 8, the transpose step for the 1D output with the SIMD instructions, and the butterfly step for the column-direction 1D 4x4 integer transform using the series of SIMD operations.

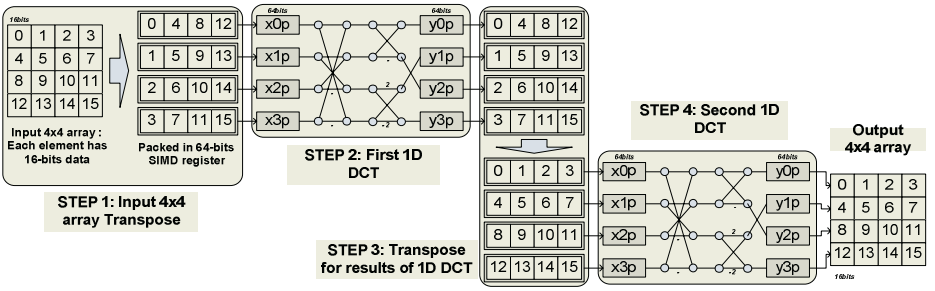


Fig. 5. Block diagram of the proposed method at 64-bits SIMD operation

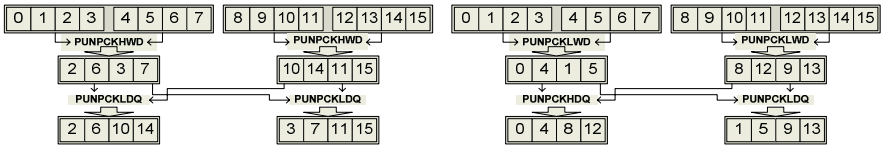


Fig. 6. 4x4 block transpose with SIMD instructions

With the SIMD instruction, PUNPCKx can be used to optimize the transpose process. The PUNPCKx instructions perform an interleave merge of the data elements of the destination and source operations into the destination register. The following example merges the two operations into the destination registers with interleaving. For example, PUNPCKLWD instruction interleaves the two low-order words of the *source1* operand (1_1 and 1_0) and the two low-order words of the *source2* operand (2_1 and 2_0) and writes them to the destination operand. PUNPCKHWD instruction interleaves the two high-order words of the *source1* operand (1_3 and 1_2) and the two high-order words of the *source2* operand (2_3 and 2_2) and writes them to the destination operand. One of the destination registers will have the combination illustrated in Fig. 7 [13].

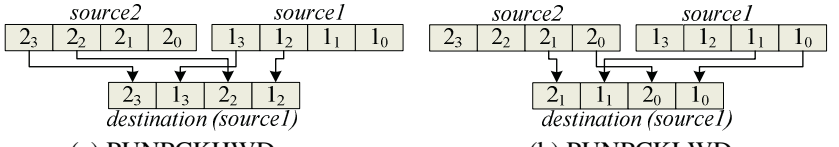


Fig. 7. Operation of the PUNPCKx instruction

We explain the butterfly step for the 1D 4x4 integer transform using the series of SIMD operations shown in Fig. 8. Let four elements in the first row be x00, x10, x20 and x30. They are loaded into x0p which is a 64-bits register. By the same token, other three rows can be loaded into registers x1p, x2p and x3p. The 1D DCT integer transform process is followed using the butterfly structure in Fig. 4(b).

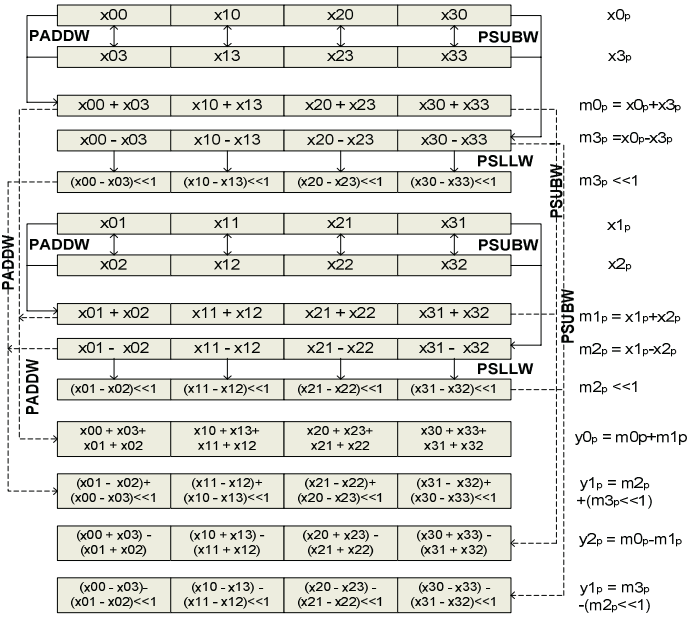


Fig. 8. Example of the 4x4 integer transform "core" implementation

The operations of addition, subtraction and shift in Fig. 4(b) are replaced with PADDW, PSUBW and PSLLW, respectively, supported by IA-32 Intel architecture [13]. Up to now we explain the SIMD-based 4x4 integer transform method on the 64-bits machine. That method can be expended to be executed on 128-bits machine. After expansion two consecutive 4x4 blocks can be simultaneously transformed. Figure 9 shows an additional process for expansion.

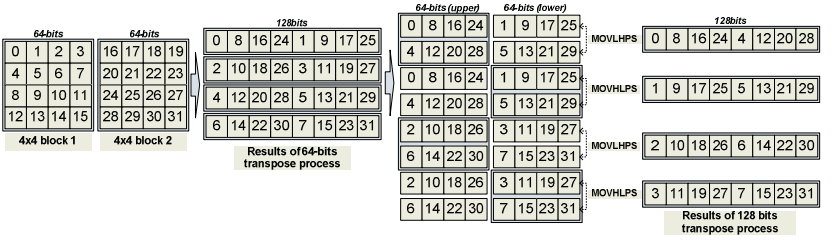


Fig. 9. 128-bits SIMD instructions for transpose

For the additional process two SIMD instructions, MOVHLPS and MOVLHPS, are used. Using the instruction of MOVHLPS (MOVLHPS) the upper (lower) 64-bits of the *source2* are loaded into the lower (upper) 64-bits of the 128-bit *destination*, and the upper(lower) 64-bits of *source1* are unchanged.

The proposed method can be applied to compute the 4x4 HT in H.264 with removing the shift instruction of PSLLW in Fig. 8.

5 Experimental Results

The reference code for the 4x4 integer transform in H.264, Zhou's and our methods are executed on a 2.4G(B) Intel Pentium IV PC. The 2.4G (B) Intel Pentium IV processor supports both 64-bits MMX and 128-bits SSE2 operations. A performance analysis tool Vtune [14] is used to collect the detailed characteristics of those methods.

Tables 1 and 2 show both execution times and a comparison of four different 4x4 integer transforms and HTs, respectively. Each execution time for the 64-bits MMX and the 128-bits implementations is measured after inputting 2 million and 1 million data, respectively.

Table 1. Comparison of different methods for the 4x4 integer transforms

Speedup / methods	Times (sec)	H.264 reference code	Proposed method (64-bits)
H.264 reference code	0.817	1.000	0.230
Proposed method (64-bits)	0.188	4.341	1.000
Proposed method (128-bits)	0.121	6.776	1.561

Table 2. Comparison of different methods for the 4x4 Hadamard transform

Speedup / methods	Times (sec)	H.264 reference code	Proposed method (64-bits)
H.264 reference code	0.458	1.000	0.384
Proposed method (64-bits)	0.176	2.602	1.000
Proposed method (128-bits)	0.115	3.983	1.530

As shown in the Table 3, the proposed methods do not use any multiplication operation. While Zhou's, which is the representative for the 128-bits SIMD implementation, is faster than the reference code by 4.2 times according to [3], the 64-bits based proposed method is as fast as Zhou's and the 128-bits based one is 1.5 times faster than Zhou's method.

Table 3. The number of operation in different methods

methods / operations	Reference code	Proposed (64-bits)	Proposed (128-bits)
mov	522	114	71
add, sub, shift	102	26	14
multiplication	0	0	0
Else	76	25	22
total clocktic	700	165	107
Speedup	1.000	4.242	6.542

6 Conclusion

In this paper, we proposed the SIMD-based fast methods for the 4x4 integer transform as well as HT in MPEG-4 Part10 AVC/H.264. The experimental results showed that the proposed 64-bits and 128-bits SIMD-based 4x4 integer transform methods were 4.3 and 6.7 times faster than the H.264 reference code, respectively. Furthermore, the proposed 64-bits and 128-bits SIMD-based 4x4 HT methods were 2.6 and 3.98 times faster than the H.264 reference code, respectively

Acknowledgment

This work was supported by grant from IT-SoC Association, grant No. R01-2002-000-00179-0 from the Basic Research Program of the Korea Science & Engineering Foundation and Research Grant of Kwangwoon University in 2004

References

1. Iain E. G. Richardson, *Video Codec Design-Developing Image and Video Compression Systems*, John Wiley & Sons Ltd, England, 2002
2. Iain E.G Richardson, *H.264 and MPEG-4 VIDEO COMPRESSION*, WILEY, 2003
3. X. Zhou, Eric Q. Li and Yen-Kuang Chen, "Implementation of H.264 Decoder on General-Purpose Processors with Media Instructions," *Proceeding of SPIE conference on Image and Video Communication and Processing*, Vol. 5022, Jan. 2003, pp. 224-235
4. T. Wiegand, G. J Sullivan, G. Bjontegaard and A.Lutha, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. on CSVT*, Vol. 13, No. 7, July 2003 pp. 560-576.
5. Henrique S. Malvar, Antti Hallapuro, Marta Karczewicz, and Louis Kerofsky, " Low-Complexity Transform and Quantization in H.264/AVC", *IEEE Transaction on Circuits and Systems for Video Technology*, Vol. 13, No.7, July 2003, pp. 598-603
6. Iain E., G Richardson, "H.264 White paper - White Papers describing aspects of the new H.264/MPEG-4 Part 10 standard", May 12, 2004
7. Vasudev B. and Konstanstinos k., *Image and Video Compression Standards*, Kluwer Academic Publishers, Norwell, Massachusetts 02061 USA, 2000
8. Tu-Chih Wang. Yu-Wen Hwang, Hung-Chi Fang, and Liang-Gee Chen, "Parallel 4x4 2D Transform and Inverse Transform Architecture for MPEG-4 AVC/H.264," *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium* on , Vol. 2 , 25-28 May 2003 p:II-800 - II-803
9. Intel Corp.,"IA-32 Intel® Architecture Optimization Reference Manual," AP - 248966-010
10. Intel Corp., "Using Streaming SIMD Extensions in a Fast DCT Algorithm for MPEG Encoding," version 1.2, Jan. 1999
11. Intel Corp., "Using Streaming SIMD Extensions 2(SSE2) to Implement an Inverse Discrete Cosine Transform," version 2.0, July 2000
12. Intel Corp., "Streaming SIMD Extensions - Matrix Multiplication," AP-930, June 1999
13. Intel Corp., "Intel Pentium 4 and Intel Xeon Processor Optimization - Reference Manual," Order Number: 248966-05, 2002
14. Intel Corp., Intel® VTune TM Performance Analyzer, Version 7.0, 2003.