# An Efficient Matrix-Based DCT Splitter/Merger for MPEG-2-to-AVC/H.264 Transform Kernel Conversion

Yuh-Jue Chuang, *Member, IEEE*, and Ja-Ling Wu, *Senior Member, IEEE*

***Abstract*—An efficient matrix-based discrete cosine transform splitter/merger for realizing MPEG-2 to AVC/H.264 transform kernel conversion is presented in this paper. Due to its matrix-based nature, the proposed transform kernel converter can be readily implemented by specific multimedia instruction set available today. Both the computational complexity and numerical error behavior of the proposed approach are shown to be superior to that of the straightforward (full decoding + reencoding) one. In other words, the effectiveness of the proposed algorithm can be justified by examining both the execution speed and the reproduced image quality.**

***Index Terms*—Discrete cosine transform (DCT), DCT splitter/merger, MPEG-2-to-AVC/H.264 transform Kernel conversion.**

## I. Introduction

Multimedia signals, such as images and videos, are always compressed to conserve valuable system resources like storage space and network bandwidth [1]. The discrete cosine transform (DCT) [2] has well-known energy packing property and the $8 \times 8$ DCT has been adopted extensively by the major coding standards, such as JPEG, MPEG-1/2/4, and H.26x series. Currently, the latest AVC/H.264 [3] video coding standard provides the capability to predict samples locally down to $4 \times 4$ blocks before operating transform coding, an impressive compression gain has been obtained. However, due to wide spread DVDs and the trend of HDTV/DTV, most existing video contents are still encoded in MPEG-2 form. Therefore, the MPEG-2-to-AVC/H.264 transcoder [4]–[12] or vice versa, is believed to be an indispensable functional module of future video signal treatments. There are many issues [4] in MPEG-2-to-AVC/H.264 transcoding including motion vector compensation [6], [12], block size selection [7], transform kernel conversion [8], and transform-domain intra prediction [5]. In this work, we focus only on the transform kernel conversion between the two standards.

Manuscript received August 25, 2005; revised March 17, 2006. This work was supported in part by the National Science Council,R.O.C. under Contracts NSC95-2221-E-182-001 and NSC95-2221-E-182-037. This paper was recommended by Associate Editor J. F. Arnold.

Y.-J. Chuang is with the Communication and Multimedia Laboratory, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 333, R.O.C. She is also with the Graduate Institute of Health Care Management, Chang Gung University, Taoyuan, Taiwan 106, R.O.C. (e-mail: chuangyj@cmlab.csie.ntu.edu.tw).

J.-L. Wu is with the Communication and Multimedia Laboratory, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan 106, R.O.C. He is also with the Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei, Taiwan 106, R.O.C. (e-mail: wjl@cmlab.csie.ntu.edu.tw).

Color versions of Figs. 1 and 2 are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2006.887077

The simplest way to develop a different size transform kernel conversion for MPEG-2-to-AVC/H.264 transcoder is directly cascading an $8 \times 8$ inverse DCT (IDCT) with four $4 \times 4$ DCTs. Many fast algorithms have been proposed to efficiently split an $8 \times 8$ DCT into four adjacent $4 \times 4$ DCTs without performing the $8 \times 8$ IDCT and the four $4 \times 4$ DCTs [8]–[11]. However, such methods can not be directly applied to the MPEG-2-to-AVC/H.264 transcoder. Problems come mainly from the fact that the $4 \times 4$ DCT kernel used in AVC/H.264 has been modified for ease of implementation [3]. In this paper, we investigate the relation between the modified $4 \times 4$ DCT kernel and the conventional one, and then, show that with some simple post processes, the existing DCT splitting algorithms still can be applied to complete the MPEG-2-to-AVC/H.264 conversion. Furthermore, most of the modern processors have specific multimedia instruction sets to facilitate multimedia applications, such as the single-instruction-multiple-data (SIMD) execution model which enables multiple arithmetic or logic operations to be executed simultaneously, so as to improve the efficiency of the program. It follows that matrix-based method is more suitable for general-propose processors with SIMD instructions, and therefore, multiplication of matrices is chosen to be the vehicle for developing the proposed DCT Splitter/Merger.

In this work, we will show that with some simple post processes, the proposed DCT splitting algorithm can still be applied to complete the transform kernel conversion from MPEG-2 to AVC/H.264. This paper is structured as follows. Section II gives a short brief of the matrix-based DCT Splitter/Merger [8]. Section III details how, with some simple post processes, the proposed and other existing DCT splitting algorithms can be applied to complete the transform kernel conversion from MPEG-2 to AVC/H.264. Section IV concludes this write up.

## II. Matrix-Based 2-D DCT Splitter and Merger

We confine our attention first to the 1-D case. The normalized forward Type-II DCT (DCT-II) of a length-$N$ sequence $x$, given in [2], can be written in matrix form as

$$X = \left[T_N^{II}\right] x = [C_N][T_N]x \tag{1}$$

where $[C_N] = \mathrm{diag}(\varepsilon_0, \varepsilon_1, \ldots, \varepsilon_{N-1}), \varepsilon_k = \sqrt{1/N}$, for $k = 0$ and $\varepsilon_k = \sqrt{2/N}$, for $k \neq 0$. $[T_N]$ is an $N \times N$ matrix with elements $[T_N]_{ij} = \cos(\pi(2j+1)i/(2\cdot N)), i, j = 0, 1, \ldots, N-1$.

Assume that the DCT coefficients $Y$ and $Z$ of two consecutive $N/2$-point data sequences $y$ and $z$ are given. The problem to be addressed is how to efficiently compute the $N$-point DCT coefficients $X$ which stands for the DCT coefficients of $x = [y\ z]^t$. Then, the problem can be rewritten as how to efficiently compute $[S_N]$ where $X = [S_N][Y\ Z]^t$.

After absorbing the computation of $\varepsilon_k$ into the quantization stage, the matrix-based DCT splitting algorithm [8] can be derived as

$$[S_N] = \begin{bmatrix} \frac{1}{2}[\tilde{P}_{N/2}] & 0 \\ 0 & \frac{1}{2}[J_{N/2}] \end{bmatrix} \begin{bmatrix} [I_{N/2}] & [I_{N/2}] \\ [I_{N/2}] & -[I_{N/2}] \end{bmatrix} \times \begin{bmatrix} [I_{N/2}] & 0 \\ 0 & [M_{N/2}]^{-1}[K_{N/2}]^{-1} \end{bmatrix} [\tilde{P}_N], \quad (2)$$

where $[K_{N/2}] = [\tilde{P}_{N/2}][L_{N/2}][\tilde{P}_{N/2}]$, $[J_N] = [\tilde{P}_N]\begin{bmatrix} [I_{N/2}] & 0 \\ 0 & -[I_{N/2}] \end{bmatrix}$, $[\tilde{P}_N]$ is an $(N \times N)$ bit-reversal matrix, $[Q_N] = \operatorname{diag}\{\cos(\phi_0), \cdots, \cos(\phi_{N-1})\}$, $\phi_n = (n + 1/4)(2\pi/N)$, and $[L_N]$ is the lower $N \times N$ triangular matrix given by (5) in [8].

The 2-D DCT can be implemented by performing a series of 1-D transforms. Let $[G]$ be the $N \times N$ two-dimensional (2-D) DCT-II of the $N \times N$ data matrix $[g]$. The data matrix $[g]$ can be decomposed into four $(N/2) \times (N/2)$ data blocks, $[g_1]$, $[g_2]$, $[g_3]$ and $[g_4]$. Our goal is to split $[G]$ into four $(N/2) \times (N/2)$ matrices $[G_1]$, $[G_2]$, $[G_3]$ and $[G_4]$, which respectively are the DCTs of $[g_1]$, $[g_2]$, $[g_3]$ and $[g_4]$, without the need of using an $N \times N$ -IDCT and four $(N/2) \times (N/2)$-DCTs. Following the same derivation for 1-D DCT, the splitting algorithm can be written as

$$\begin{bmatrix} [G_1] & [G_2] \\ [G_3] & [G_4] \end{bmatrix} = [S_N][G][S_N]^{-1} \quad (3)$$

and the merging algorithm as

$$[G] = [S_N]^{-1} \begin{bmatrix} [G_1] & [G_2] \\ [G_3] & [G_4] \end{bmatrix} [S_N]. \quad (4)$$

From the analysis given in [8], the proposed algorithm is more efficient than the direct approach. The difference of performances between the proposed algorithm and the direct approach will become more evident when $N$ gets even larger. It is also shown that by integrating an effective algorithmic-level derivation and programming-level optimizations with SIMD instructions, the proposed approach achieves better performance when compared with the direct approach.

## III. Transform Kernel Conversion for MPEG-2-to-AVC/H.264 Transcoder

As above described in Section I, converting an MPEG-2 encoded bitstream into the AVC/H.264 format (and vice versa) can achieve interoperability between different multimedia systems. This gives rise to a strong need for implementing an MPEG-2-to-AVC/H.264 (or vice versa) transcoder. Performing a transcoder in the transform domain could be more efficient because the decoding and reencoding processes are not required. In this work, we focus only on the transform kernel conversion. The prescribed DCT Splitter/Merger can efficiently split an 8 × 8 DCT into four adjacent 4 × 4 DCTs; however, it can not be directly applied to conduct MPEG-2-to-AVC/H.264 transform kernel conversion. As prescribed, the reason is that the 4 × 4 DCT kernel used in AVC/H.264, we called it the modified 4-point DCT kernel, has been modified. In the following, we investigate the relation between the modified 4 × 4 DCT kernel and the conventional one, and then, show that with some simple post processes, the existing DCT splitting algorithms can still be applied to complete the transform kernel conversion from MPEG-2 to AVC/H.264.

### A. The Transform and Quantization Methods in AVC/H.264

The 4-point type-II DCT kernel can be represented in matrix form as

$$[T_4^{II}] = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ \beta & \gamma & -\gamma & -\beta \\ \alpha & -\alpha & -\alpha & \alpha \\ \gamma & -\beta & \beta & -\gamma \end{bmatrix} \quad (5)$$

where $\alpha = 1/2$, $\beta = \sqrt{1/2}\cos(\pi/8)$, and $\gamma = \sqrt{1/2}\sin(\pi/8)$.

Let $[T_4']$ denote the transform matrix of the modified 4-point DCT used in AVC/H.264, then

$$[T_4'] = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ \beta' & \gamma' & -\gamma' & -\beta' \\ \alpha' & -\alpha' & -\alpha' & \alpha' \\ \gamma' & -\beta' & \beta' & -\gamma' \end{bmatrix} \quad (6)$$

where $\alpha' = 1/2$, $\beta' = \sqrt{2/5}$ and $\gamma' = \sqrt{1/10}$.

$[T_4']$ is adopted by AVC/H.264 because it can be easily factorized as the product of a scaling matrix, $[DM_4]$, and an **integer** matrix, $[H_4]$, in which $[DM_4] = \operatorname{diag}\{\alpha', \beta'/2, \alpha', \beta'/2\}$ and

$$[H_4] = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix}. \quad (7)$$

In actual AVC/H.264 transform coding, $[DM_4]$ will be absorbed into the quantization stage and its inverse transform matrix is given by

$$[H_4]_{\text{inv}} = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & 1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix}.$$

Cooperating with the designed quantization and dequantization formulas used in AVC/H.264 [13], i.e.,

$$X_q(i,j) = \operatorname{sign}\{X(i,j)\}\left[|\left(X(i,j)|A(Q_M,i,j) + f2^{15+Q_E}\right) \gg (15+Q_E)\right] \quad (8)$$

and

$$X_r(i,j) = X_q(i,j)B(Q_M,i,j) \ll Q_E \quad (9)$$

a low complexity transform coding can be achieved, where $X(i,j)$ represent the 4 × 4 DCT coefficients, $X_q(i,j)$ and $X_r(i,j)$ are the corresponding quantization and reconstruction coefficients, and "$\gg$" and "$\ll$" are the bitwise right and left shift operators, respectively. $Q_M \equiv Q \bmod 6$ and $Q_E \equiv Q/6$ for the quantization step $Q$. The values of $A(Q_M,i,j)$ and

$B(Q_M, i, j)$, at the position $(i, j)$, $i, j = \{0, 1, 2, 3\}$, for the transform coefficients inside the block are given by

$$A(Q_M, i, j) = \begin{cases} M(Q_M, 0), & \text{when both } i \text{ and } j \text{ are odd} \\ M(Q_M, 1), & \text{when both } i \text{ and } j \text{ are even} \\ M(Q_M, 2), & \text{otherwise} \end{cases} \tag{10}$$

and

$$B(Q_M, i, j) = \begin{cases} S(Q_M, 0,) & \text{when both } i \text{ and } j \text{ are odd} \\ S(Q_M, 1), & \text{when both } i \text{ and } j \text{ are even} \\ S(Q_M, 2), & \text{otherwise} \end{cases} \tag{11}$$

with

$$M = \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix} \text{ and } S = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}. \tag{12}$$

### *B. Proposed Transform and Quantization Methods for MPEG-2-to-AVC/H.264 Transform Kernel Conversion*

According to (2) and (3), an $8 \times 8$ DCT-II block $[G]$ can be split into four $4 \times 4$ DCT-II blocks $[G_1]$, $[G_2]$, $[G_3]$ and $[G_4]$ and vice versa, where $[S_8]$ is the DCT Splitter and is equivalent to $\begin{bmatrix} [T_4^{II}] & 0 \\ 0 & [T_4^{II}] \end{bmatrix} [T_8^{II}]^{-1}$. Since $[S_8]$ can not be directly applied to conduct MPEG-2-to-AVC/H.264 transform kernel conversion, we readdress the problem as how to compute $[S_8']$ such that

$$[S_8'] = \begin{bmatrix} [T_4'] & 0 \\ 0 & [T_4'] \end{bmatrix} [T_8^{II}]^{-1}. \tag{13}$$

For reusing existing DCT Splitting algorithms, the relationship between $[T_4^{II}]$ and $[T_4']$ plays the key role. Equation (13) can be reformulated as $[S_8'] = [B_8][S_8]$ where

$$[B_8] = \begin{bmatrix} [T_4'] & 0 \\ 0 & [T_4'] \end{bmatrix} \begin{bmatrix} [T_4^{II}]^{-1} & 0 \\ 0 & [T_4^{II}]^{-1} \end{bmatrix}. \tag{14}$$

The definition of $[B_8]$ is general enough for being applied to all existing algorithms which split a length-8 DCT into two length-4 DCTs. $[B_8]$ can also be seen as the postprocess for adjusting the outputs of $[S_8]$.

Because $[T_4^{II}]^{-1}$ can be calculated as

$$[T_4^{II}]^{-1} = \begin{bmatrix} 1/2 & \beta & \alpha & \gamma \\ 1/2 & \gamma & -\alpha & -\beta \\ 1/2 & -\gamma & -\alpha & \beta \\ 1/2 & -\beta & \alpha & -\gamma \end{bmatrix} \tag{15}$$

and from (14), the complexity of $[B_8]$ depends on the complexity of the multiplication of $[T_4'][T_4^{II}]^{-1}$, which can be represented as

$$[T_4'] [T_4^{II}]^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2(\beta\beta' + \gamma\gamma') & 0 & 2(\beta'\gamma - \gamma'\beta) \\ 0 & 0 & 1 & 0 \\ 0 & 2(\beta\gamma' - \gamma\beta') & 0 & 2(\beta\beta' + \gamma\gamma') \end{bmatrix}. \tag{16}$$

After substituting the corresponding variables defined in (5) and (6), $2(\beta\beta' + \gamma\gamma')$ and $2(\beta\gamma' - \gamma\beta')$ can be written as $(\sqrt{4/5}\cos(\pi/8) + \sqrt{1/5}\sin(\pi/8))$ and $(\sqrt{1/5}\cos(\pi/8) - \sqrt{4/5}\sin(\pi/8)$, respectively. It is easy to verify that

$$\sqrt{\frac{4}{5}} \approx \sin(63.43^\circ) \text{ and } \sqrt{\frac{1}{5}} \approx \cos(63.43^\circ).$$

Using trigonometric equalities, we have

$$\begin{aligned} 2(\beta\beta' + \gamma\gamma') &= \sin(63.43^\circ)\cos\frac{\pi}{8} + \cos(63.43^\circ)\sin\frac{\pi}{8} \\ &= \sin(85.93^\circ) \end{aligned} \tag{17}$$

and

$$\begin{aligned} 2(\beta\gamma' - \gamma\beta') &= \cos(63.43^\circ)\cos\frac{\pi}{8} - \sin(63.43^\circ)\sin\frac{\pi}{8} \\ &= \cos(85.93^\circ). \end{aligned} \tag{18}$$

Consequently

$$\begin{aligned} [T_4'] [T_4^{II}]^{-1} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \sin(85.93^\circ) & 0 & -\cos(85.93^\circ) \\ 0 & 0 & 1 & 0 \\ 0 & \cos(85.93^\circ) & 0 & \sin(85.93^\circ) \end{bmatrix} \\ &= [A_4^1] [A_4^2] \end{aligned} \tag{19}$$

where

$$[A_4^1] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(85.93^\circ) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \cos(85.93^\circ) \end{bmatrix}$$

and

$$[A_4^2] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \tan(85.93^\circ) & 0 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & \tan(85.93^\circ) \end{bmatrix}.$$

The computation of $[A_4^1]$ can also be eliminated by absorbing it into the quantization stage. This implies that $[B_8]$ can be simplified to $\begin{bmatrix} [A_4^2] & 0 \\ 0 & [A_4^2] \end{bmatrix}$, in which $\tan(85.93^\circ)$ can be precalculated and stored. Let $[f] = [f_0, f_1, \ldots, f_7]^t$ and $[F] =$

$[F_0, F_1, \ldots, F_7]^t$ be two length-8 vectors. The multiplication of $[B_8]$ and $[f]$ or, equivalently, the postprocess, can be realized as

$$\begin{aligned}[F] &= [B_8][f] \\ &= \begin{bmatrix} [f_0\ (f_1 \tan 85.93^\circ - f_3)\ f_2\ (f_1 + f_3 \tan 85.93^\circ)]^t \\ [f_4\ (f_5 \tan 85.93^\circ - f_7)\ f_6\ (f_5 + f_7 \tan 85.93^\circ)]^t \end{bmatrix}\end{aligned} \tag{20}$$

Equation (20) implies that only 4 multiplications (denoted as 4M) and 4 additions (denoted as 4A) or one more SIMD instruction (PMADDWD) [8] are required to accomplish the postprocess.

For absorbing the $[A_4^1]$ into quantization, the quantization factor in (8), $A(Q_M, i, j)$, and quantization table $M$ need to be redefined as

$$\begin{aligned} &A_p(Q_M, i, j) \\ &= \begin{cases} M_p(Q_M, 0) & \text{for } (i,j) = (0,0),(0,2),(2,0),(2,2) \\ M_p(Q_M, 1) & \text{for } (i,j) = (1,1),(1,3),(3,1),(3,3) \\ M_p(Q_M, 2) & \text{for } (i,j) = (1,2),(1,4),(3,2),(3,4) \\ M_p(Q_M, 3) & \text{for } (i,j) = (2,1),(0,2),(2,0),(2,2) \end{cases}\end{aligned} \tag{21}$$

and

$$M_p = 4 \times \begin{bmatrix} 13107 & 179720 & 905 & 13108 \\ 11916 & 166886 & 840 & 11650 \\ 10082 & 146031 & 735 & 10485 \\ 9362 & 129788 & 654 & 9118 \\ 8192 & 116820 & 588 & 8388 \\ 7282 & 101580 & 512 & 7233 \end{bmatrix}. \tag{22}$$

However, the values of elements in $M_p$ are too large to allow 16-bit arithmetic implementation. Thus, we modified the proposed quantization table, $M_p$, and the quantization formula, $X_q$ in (8), again as (23), shown at the bottom of the page, and

$$M_p = \begin{bmatrix} 13107 & 44933 & 905 & 13108 \\ 11916 & 41725 & 840 & 11650 \\ 10082 & 36508 & 735 & 10485 \\ 9362 & 32450 & 654 & 9118 \\ 8192 & 29208 & 588 & 8388 \\ 7282 & 25398 & 512 & 7233 \end{bmatrix} \tag{24}$$

where $X(i,j)$ in (23) is the output of our proposed transform kernel converter. Then, the original decoder, without any modification, can decode the bitstreams generated by the proposed transform kernel converter, directly.

## C. Computational Complexity Analyses

*1) Algorithmic-Level Analysis:* First, the number of multiplications and additions required for the proposed approach will be counted in comparison with the direct approach. As shown in [8], we need 10M and 29A to get $[S_8]$. Hence, splitting an 8 × 8 DCT-II into four 4 × 4 DCT-IIs can be implemented by performing 1-D split/merge algorithm along each row and then each column. Our algorithm needs $16 \times (10M + 29A) = 160M + 464A$, in total. As described in (20), we need to perform four more multiplications and additions for adjusting the coefficients along row and column sides, respectively. Thus, the proposed algorithm totally needs $168M$ and $472A$ for splitting an 8 × 8 DCT-II into four 4 × 4 modified DCTs blocks adopted in AVC/H.264. Comparing with the straightforward approach, it takes one 8 × 8 IDCT-II, and then takes four forward 4 × 4 modified DCTs for splitting an 8 × 8 DCT-II block into four 4 × 4 AVC/H.264 DCT blocks. There are $176M + 464A$ for one 8 × 8 inverse DCT-II and $48A$ for four 4 × 4 modified DCTs. The amount of computation are $176M + 512A$.

*2) Programming-Level Analysis:* Follow the similar analysis given in [8], 80 PMADDWDs and 80 PADDDs are needed for splitting an 8 × 8 DCT-II into four 4 × 4 DCT-IIs in the programming-level and two more PMADDDs are required for modifying the outputs to be applied to an AVC/H.264 decoder. Comparing with the straightforward approach, 144 PMADDWDs, and 96 PADDDs are needed.

In Table I, the numbers of operations needed in different approaches for performing the proposed and the straightforward approaches are tabulated. It follows that both arithmetic and programming level complexities of the proposed algorithm are lower than that of the straightforward approach.

TABLE I
NUMBERS OF OPERATIONS REQUIRED FOR SPLITTING AN 8 × 8 DCT INTO FOUR 4 × 4 IN DIFFERENT APPROACHES

| Approaches | Algorithmic-Level Analysis | Programming-Level Analysis |
|---|---|---|
| The Proposed Approach | $168M + 472A$ | 82 PMADDWDs + 80 PADDDs |
| Straightforward Approach | $176M + 512A$ | 144 PMADDWDs + 96 PADDDs |

## D. Error Behavior of the Straightforward and the Proposed Approaches

The straightforward approach for splitting a 2-D 8 × 8 DCT-II into four 4 × 4 modified DCTs needs to do coefficient roundings after conducting the 8 × 8 IDCT-II and the four 4 × 4 modified DCTs, respectively. However, for intra-coded blocks only one rounding step is required in the proposed approach. Intuitively, image quality of the proposed approach would be better than that of the straightforward approach, if intra-coded block is considered. In this section, we will analyse the error behavior of the straightforward and the proposed approaches and discuss what is the effect of the errors. 64-bits word-length rounding operation is adopted for both direct and proposed approaches. Since transform kernel

$$X_{q\text{-}p} = \begin{cases} \operatorname{sign}(X(i,j)) \left[ |X(i,j)|\, 4 \times A_p(Q_M, i, j) + f 2^{13+Q_E} \right] \gg (13 + Q_E), & \text{for } (i,j) = \{(1,2),(1,4),(3,2),(3,4)\} \\ \operatorname{sign}(X(i,j)) \left[ |X(i,j)|\, 4 \times A_p(Q_M, i, j) + f 2^{15+Q_E} \right] \gg (15 + Q_E), & \text{otherwise} \end{cases} \tag{23}$$

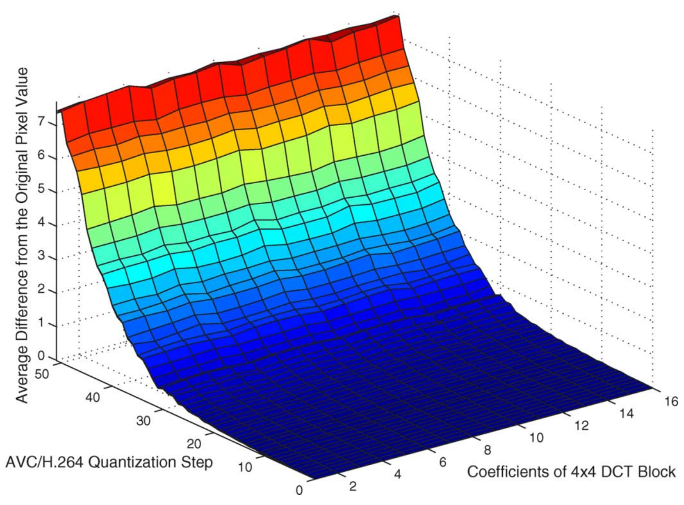

Fig. 1. Average difference between the original block and the reconstructed ones produced by the straightforward and the proposed approaches. When the AVC/H.264 quantization step becomes larger, the average difference increases.

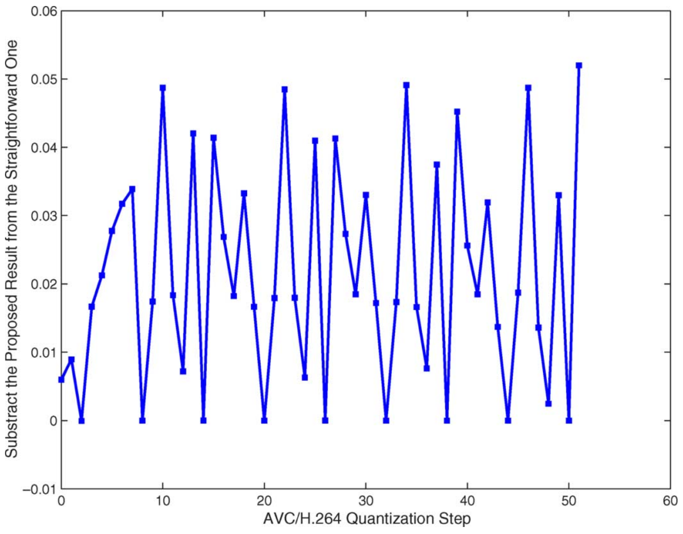

Fig. 2. Subtracting the average difference of the proposed approach from that of the straightforward one for observing the relation between the AVC/H.264 Quantization Step and the difference between the straightforward and the proposed approaches. Most values are positive means that the difference produced by the straightforward approach is larger than that of the proposed approach.

conversion is our focus, instead of transcoder, only intra-coded blocks are considered in the following analyses.

The analysis unit of the proposed error model is a $4 \times 4$ block. We have randomly chosen $10^4$ $4 \times 4$ blocks from 18 standard test images of size $512 \times 512$ as the testing benchmark and examined the error behavior of the straightforward and the proposed approaches with different quantization steps $(\mathrm{Q} = 0 - 51)$ defined in AVC/H.264 on each coefficient inside the $4 \times 4$ block. For each block and each quantization step, $Q$, we first calculated the differences between the original block and the reconstructed ones produced by the straightforward and the proposed approaches, and also recorded the absolute values of them. Then, we computed the average of each of the differences from $1.8 \times 10^5$ blocks and conducted the corresponding error behavior analyses.

Fig. 1 presents the average differences between the original pixel values and the ones reconstructed by using the straightforward and the proposed approaches, where the $x, y$, and $z$ axes are the position $(i, j)$ of a $4 \times 4$ block, AVC/H.264 quantization step and the average difference from the original pixel value, respectively. Note that the position $(i, j)$ represents the value $(4 \times i + j)$ in the $x$ axis. Because the trend of the differences of the coefficients for the same quantization step are similar, we average the differences. Since the results shown in Fig. 1 for the two approaches are too close to be discriminated, we subtract the results of the proposed approach from that of the straightforward one and show their differences in Fig. 2. Except for the quantization steps $Q_m = (2+6\times m)$ where $m = 0, 1, \cdots, 8$, we find that all the differences are positive, and this means that the difference produced by the straightforward approach is always larger than that of the proposed approach. According to the definition of $Q_M$, $Q_E$ and (10), quantization factor doubles with $Q$ increase of 6. Due to the specific design of the third row in the quantization factor table $M$ and the dequantization factor table $S$ for $Q_m$, the difference between our approach and the straightforward one are close to zero, as shown in Fig. 2, in those $Q_m$ positions.

## IV. Conclusion

The $8 \times 8$ DCT is adopted extensively by the major coding standards, such as JPEG, MPEG-1/2/4, and H.26x series. However, instead of using the popular $8 \times 8$ DCT, the latest AVC/H.264 employs $4 \times 4$ DCT as the transform kernel and has better quality v.s. bandwidth performance. This paper focuses on the DCT kernel conversions of different sizes. Many fast algorithms have been proposed to efficiently split one $N \times N$ DCT block into four $N/2 \times N/2$ DCT blocks and vice versa. MPEG-2 to AVC/H.264 transcoder is believed to be an important functional module for future video signal treatments. When $N = 8$, our method can be applied to conduct MPEG-2-to-AVC/H.264 transform kernel conversion. However, the existing methods for splitting an $8 \times 8$ DCT block into four $4 \times 4$ DCT blocks can not be directly used because the $4 \times 4$ DCT kernel of AVC/H.264 has been modified. We proposed an efficient postprocessing method which can be combined with any existing splitting DCT approach in the literature to achieve an efficient DCT splitting. Since most of the modern processors enable multiple arithmetic or logic operations to be executed at the same time for improving the speed performance of multimedia applications, the proposed matrix-based DCT Splitter (or Merger), which can split an $N \times N$ DCT block into four $N/2 \times N/2$ or two $N \times N/2$ (or $N/2 \times N$) modified DCT blocks (or merger small size blocks into a large size one), is designed for more suitable being realized on processors with SIMD instructions than the other approaches.

## References

[1] S. F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," *IEEE J. Sel. Areas Commun.*, vol. 13, no. 1, pp. 1–11, Jan. 1995.

[2] K. R. Rao and P. Yip, *Discrete Cosine Transform: Algorithms, Advantages, and Applications*. New York: Academic, 1990.

[3] *ITU-T Rec. Final Draft Int. Standard of Joint Video Specification*, ITU-T Rec. H.264—ISO/IEC 11496-10, Advanced Video Coding, 2003.

[4] H. Kalva, "Issues in H.264/MPEG-2 video transcoding," in *Proc. IEEE Int. Conf. Consum. Commun. Netw.*, Las Vegas, NV, Jan. 2004, pp. 657–659.

[5] C. Chen, P. H. Wu, and H. Chen, "MPEG-2 to H.264 transcoding," in *Proc. Picture Coding Symp.*, San Francisco, Dec. 2004, pp. 215–220.

[6] Z. Zhou, S. Sun, S. Lei, and M. T. Sun, "Motion information and coding mode reuse for MPEG-2 to H.264 transcoding," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2005, vol. 2, pp. 1230–1233.

[7] G. Chen, Y. Zhang, S. Lin, and F. Dia, "Efficient block size selection for MPEG-2 to H.264 transcoding," in *Proc. 12th Ann. ACM Int. Conf. Multimedia*, Oct. 2004, pp. 300–303.

[8] Y.-J. Chuang and J.-L. Wu, "An efficient matrix-based 2-D DCT splitter and merger for SIMD instructions," *IEICE Trans. Inf. Syst.*, vol. E88-D, no. 7, pp. 1569–1577, Jul. 2005.

[9] A. N. Skodras, "Direct transform to transform computation," *IEEE Signal Process. Lett.*, vol. 6, no. 8, pp. 202–204, Aug. 1999.

[10] R. Dugad and N. Ahuja, "A fast scheme for image size change in the compressed domain," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, no. 4, pp. 900–913, Apr. 2001.

[11] K. T. Fung and W. C. Siu, "DCT-based video downscaling transcoder using split and merge technique," *IEEE Trans. Image Process.*, vol. 15, no. 2, pp. 394–403, Feb. 2006.

[12] M. Kucukgoz and M. Sun, "Early-stop and motion vector reusing for MPEG-2 to H.264 transcoding," *SPIE Vis. Commun. Image Process.*, 2004.

[13] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in H.264/AVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 598–603, Jul. 2003.