A Fast Algorithm for DCT-Domain Inverse Motion Compensation

Neri Merhav* Vasudev Bhaskaran†

Abstract

One of the important tasks of a multiuser video network server is to composite compressed video streams from several sources into a single compressed video stream. A great deal of the computational load can be saved if this composition is performed directly in the compressed domain rather than using the brute-force approach of converting back to the uncompressed domain, compositing pixel-by-pixel in the spatial domain, and re-compressing the composite stream. We propose a fast algorithm that converts motion compensated compressed video into a sequence of DCT-domain blocks corresponding to the spatial domain blocks of the current frame alone, without prediction based on other frames, i.e., removing the inter-frame element of the compression-decompression. This operation enables video compositing in the DCT compressed domain as well as several compositing operations, e.g., scaling, overlapping, translation, filtering, etc. The proposed algorithm saves about 47% of the computations compared to the brute-force approach even without assuming sparseness of the DCT blocks. For typical sparse DCT blocks, where only the top-left 4×4 quadrant is nonzero, the reduction in computational complexity is about 68%.

*N. Merhav is with the HP Israel Science Center, Technion City, Haifa 32000, Israel.

†V. Bhaskaran is with the Visual Computing Department, HP Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304, U.S.A.

1. Introduction

A basic requirement from a modern video network is the ability to composite compressed video streams from several sources and to create a single compressed composite stream that combines them all in a specified fashion. A common example is that of video conferencing. While the compression and decompression of each stream is performed at the user site, the composition part, which requires powerful computation resources, is normally performed by a server located at an intermediate node. The server collects compressed video streams from all parties of the conferencing session, composites them in a manner specified by the user, and transmits to him/her the composite compressed video stream. The fact that both input and output streams of the server are in the compressed domain, the large computational cost of compression and decompression [1], and the enormous data rates associated with digital video, are all demanding development of fast algorithms for video composition and related tasks (e.g., scaling, translating, filtering) that operate directly in the compressed domain (see, e.g., [2], [3], [4] and references therein).

Several video compression standards, like MPEG and H.261, combine transform domain techniques, in particular, the discrete cosine transform (DCT), for removing spatial redundancy, with motion compensation (MC) methods for eliminating temporal redundancy. In this work, we focus on developing a fast algorithm for undoing the motion compensation operation in the DCT domain [2]. This algorithm receives as input DCT blocks of motion compensated compressed video, and provides DCT blocks of the corresponding spatial domain blocks of the current frame alone, without reference to past and future frames. This operation of canceling motion compensation enables video compositing in the DCT compressed domain as well as the above mentioned related online edition operations.

We further develop and improve on the algorithm proposed by Chang and Messerschmitt

[2] for inverse motion compensation. While in [2] computations are saved only if the DCT blocks are sufficiently sparse and if a large fraction of the reference blocks are aligned to the boundaries between the original blocks (at least in one direction), the improved algorithm proposed here reduces the computational complexity by 47% compared to the brute-force approach, even without any prior assumptions on sparseness or perfect alignment. If, in addition, DCT blocks are assumed sparse in the sense that only the top-left 4×4 subblocks are nonzero (as is typically the case), then computational complexity is reduced by 68%. By "computational complexity", in the context of this work, we mean the count of the basic arithmetic operations of the PA-RISC processor, namely, "shift", "add", or "shift and add" (SH1ADD, SH2ADD, and SH3ADD).

2. Preliminaries and Problem Description

The 8×8 2D-DCT transforms a block $\{x(n,m)\}_{n,m=0}^{7}$ in the spatial domain into a matrix of frequency components $\{X(k,l)\}_{k,l=0}^{7}$ according to the following equation

$$X(k,l) = \frac{c(k)}{2}\frac{c(l)}{2}\sum_{n=0}^{7}\sum_{m=0}^{7} x(n,m)\cos(\frac{2n+1}{16}\cdot k\pi)\cos(\frac{2m+1}{16}\cdot l\pi) \tag{1}$$

where $c(0) = 1/\sqrt{2}$ and $c(k) = 1$ for $k > 0$. The inverse transform is given by

$$x(n,m) = \sum_{k=0}^{7}\sum_{l=0}^{7} \frac{c(k)}{2}\frac{c(l)}{2} X(k,l)\cos(\frac{2n+1}{16}\cdot k\pi)\cos(\frac{2m+1}{16}\cdot l\pi). \tag{2}$$

In a matrix form, let $\boldsymbol{x} = \{x(n,m)\}_{n,m=0}^{7}$ and $\boldsymbol{X} = \{X(k,l)\}_{k,l=0}^{7}$. Define the 8-point DCT matrix $S = \{s(k,n)\}_{k,n=0}^{7}$, where

$$s(k,n) = \frac{c(k)}{2}\cos(\frac{2n+1}{16}\cdot k\pi). \tag{3}$$

Then,

$$\boldsymbol{X} = S\boldsymbol{x}S^t \tag{4}$$

where the superscript t denotes matrix transposition. Similarly, let the superscript $-t$ denote transposition of the inverse. Then,

$$\boldsymbol{x} = S^{-1}\boldsymbol{X}S^{-t} = S^{t}\boldsymbol{X}S \tag{5}$$

where the second equality follows from the unitarity of S.

Motion compensation of compressed video [6], [7] (see also [8]) means predicting each 8×8 spatial domain block $\boldsymbol{x}$ of the current frame by a corresponding reference block $\hat{\boldsymbol{x}}$ from a previous frame [1] and encoding the resulting prediction error block $\boldsymbol{e} = \boldsymbol{x} - \hat{\boldsymbol{x}}$ by using the DCT. The best matching reference block $\hat{\boldsymbol{x}}$ may not be aligned to the original 8×8 blocks of the reference frame. In general, the reference block may intersect with four neighboring spatial domain blocks, henceforth denoted $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$, and $\boldsymbol{x}_4$, that together form a 16×16 square, where $\boldsymbol{x}_1$ corresponds to northwest, $\boldsymbol{x}_2$ to northeast, $\boldsymbol{x}_3$ to southwest and $\boldsymbol{x}_4$ to southeast.

Our goal is to compute the DCT $\boldsymbol{X}$ of the current block $\boldsymbol{x} = \hat{\boldsymbol{x}} + \boldsymbol{e}$ from the given DCT $\boldsymbol{E}$ of the prediction error $\boldsymbol{e}$, and the DCT's $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$ of $\boldsymbol{x}_1, ..., \boldsymbol{x}_4$, respectively. Since $\boldsymbol{X} = \hat{\boldsymbol{X}} + \boldsymbol{E}$, $\hat{\boldsymbol{X}}$ being the DCT of $\hat{\boldsymbol{x}}$, the main problem that remains is that of calculating $\hat{\boldsymbol{X}}$ directly from $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$.

Let the intersection of the reference block $\hat{\boldsymbol{x}}$ with $\boldsymbol{x}_1$ form a $h \times w$ rectangle (i.e., h rows and w columns), where $1 \leq h \leq 8$ and $1 \leq w \leq 8$. This means that the intersections of $\hat{\boldsymbol{x}}$ with $\boldsymbol{x}_2$,$\boldsymbol{x}_3$, and $\boldsymbol{x}_4$ are rectangles of sizes $h \times (8-w)$, $(8-h) \times w$, and $(8-h) \times (8-w)$, respectively. Following Chang and Messerschmitt [2], it is readily seen that $\hat{\boldsymbol{x}}$ can be expressed as a superposition of appropriate windowed and shifted versions of $\boldsymbol{x}_1, ...\boldsymbol{x}_4$, i.e.,

$$\hat{\boldsymbol{x}} = \sum_{i=1}^{4} c_{i1}\boldsymbol{x}_i c_{i2}, \tag{6}$$

[1] In some of the frames (B-frames) blocks are estimated from both past and future reference blocks. For the sake of simplicity, we shall assume here that only the past is used (P-frames). The extension is straightforward.

where c_{ij}, $i = 1, ..., 4$, $j = 1, 2$, are sparse 8×8 matrices of zeroes and ones that perform window and shift operations accordingly. The basic idea behind the the work of Chang and Messerschmitt [2] is to use the distributive property of matrix multiplication w.r.t. the DCT. Specifically, since $S^t S = I$, eq. (6) may be rewritten as

$$\hat{\boldsymbol{x}} = \sum_{i=1}^{4} c_{i1} S^t S \boldsymbol{x}_i S^t S c_{i2}. \tag{7}$$

Next, by premultiplying both sides of (7) by S, and postmultiplying by S^t, one obtains

$$\hat{\boldsymbol{X}} = \sum_{i=1}^{4} C_{i1} \boldsymbol{X}_i C_{i2}. \tag{8}$$

where C_{ij} is the DCT of c_{ij}. Chang and Messerscmitt [2] proposed to precompute the fixed matrices C_{ij} for every possible combination of w and h, and to compute $\hat{\boldsymbol{X}}$ directly in the DCT domain using eq. (8). Although most of the matrices C_{ij} are not sparse, computations can still be saved on the basis of typical sparseness of $\{\boldsymbol{X}_i\}$, and due to the fact the reference block might be aligned in one direction (either $w = 8$ or $h = 8$), which means that the right-hand side of eq. (8) contains two terms only, or in both directions ($w = h = 8$), in which case $\hat{\boldsymbol{x}} = \boldsymbol{x}_1$ and hence no computations at all are needed.

3. The Proposed Algorithm

We now demonstrate that the computation of $\hat{\boldsymbol{X}}$ can be done even more efficiently by utilizing two main facts. First, we observe that some of the matrices c_{ij} are equal to each other for every given w and h. Specifically,

$$c_{11} = c_{21} = U_h \triangleq \begin{pmatrix} 0 & I_h \\ 0 & 0 \end{pmatrix}$$

$$c_{12} = c_{32} = L_w \triangleq \begin{pmatrix} 0 & 0 \\ I_w & 0 \end{pmatrix}$$

where I_h and I_w are identity matrices of dimension $h \times h$ and $w \times w$, respectively. Similarly,

$$c_{31} = c_{41} = L_{8-h},$$

and

$$c_{22} = c_{42} = U_{8-w}.$$

The second observation that helps in saving computations is that rather than fully precomputing C_{ij}, it might be more efficient to leave these matrices factorized into relatively sparse matrices. In particular, similarly as in [5], we shall use a factorization of S that corresponds to the fastest existing algorithm for 8-point DCT due to Arai, Agui, and Nakajima [9] (see also [10]). According to this factorization, S is represented as follows.

$$S = DPB_1B_2MA_1A_2A_3 \tag{9}$$

where D is a diagonal matrix given by

$$D = \mathrm{diag}\{0.3536, 0.2549, 0.2706, 0.3007, 0.3536, 0.4500, 0.6533, 1.2814\}, \tag{10}$$

P is a permutation matrix given by

$$P = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

and the remaining matrices are defined as follows:

$$B_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{pmatrix}$$

$$
B_2 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 1
\end{pmatrix}
$$

$$
M = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.7071 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.9239 & 0 & -0.3827 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.7071 & 0 & 0 \\
0 & 0 & 0 & 0 & -0.3827 & 0 & 0.9239 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

$$
A_1 = \begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

$$
A_2 = \begin{pmatrix}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
$$

$$
A_3 = \begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & -1
\end{pmatrix}
$$

The best way we have found to use the two observations mentioned above is the following:

First, we precompute the fixed matrices

$$
J_i \triangleq U_i (M A_1 A_2 A_3)^t, \qquad i = 1, 2, ..., 8
$$

and

$$K_i \triangleq L_i(MA_1A_2A_3)^t, \qquad i = 1, 2, ..., 8$$

These matrices are very structured and therefore, premultiplication by K_i or J_i can be implemented very efficiently as we shall demonstrate shortly. Next, we compute $\hat{\boldsymbol{X}}$ by using the expression

$$\begin{aligned} \hat{\boldsymbol{X}} &= S[J_h B_2^t B_1^t P^t D(\boldsymbol{X}_1 DPB_1B_2J_w^t + \boldsymbol{X}_2 DPB_1B_2K_{8-w}^t) + \\ & K_{8-h} B_2^t B_1^t P^t D(\boldsymbol{X}_3 DPB_1B_2J_w^t + \boldsymbol{X}_4 DPB_1B_2K_{8-w}^t)]S^t \end{aligned} \tag{11}$$

which can easily be obtained from eqs. (7) and (9), or by its dual form

$$\begin{aligned} \hat{\boldsymbol{X}} &= S[(J_h B_2^t B_1^t P^t D\boldsymbol{X}_1 + K_{8-h} B_2^t B_1^t P^t D\boldsymbol{X}_3) DPB_1B_2J_w^t + \\ & (J_h B_2^t B_1^t P^t D\boldsymbol{X}_2 + K_{8-h} B_2^t B_1^t P^t D\boldsymbol{X}_4) DPB_1B_2K_{8-w}^t]S^t, \end{aligned} \tag{12}$$

depending on which one of these expressions requires less computations for the given w and h.

4. Implementation Issues and Computational Complexity

We now demonstrate how to implement fast multiplication by J_i and K_i, which is the bottle neck of the computation load. As an example, we shall examine J_6. The other matrices are handled in a similar fashion. The matrix J_6 is the following:

$$J_6 = \begin{pmatrix} 1 & -1 & -a & 0 & b & a & c & 0 \\ 1 & 1 & -a & -1 & b & 0 & c & 0 \\ 1 & 1 & -a & -1 & -b & 0 & -c & 0 \\ 1 & -1 & -a & 0 & -b & -a & -c & 0 \\ 1 & -1 & a & 0 & c & -a & -b & 0 \\ 1 & 1 & a & 1 & c & 0 & -b & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

where $a = 0.7071$, $b = 0.9239$, and $c = 0.3827$. To compute $u = J_6 v$, where $u = (u_1, ..., u_8)^t$ and $v = (v_1, ..., v_8)^t$, we calculate according to the following steps:

$$y_1 = v_1 + v_2 \tag{13}$$

$$y_2 = v_1 - v_2 \tag{14}$$

$$y_3 = av_3 \tag{15}$$

$$y_4 = av_6 \tag{16}$$

$$y_5 = y_1 - y_3 \tag{17}$$

$$y_6 = y_5 - v_4 \tag{18}$$

$$y_7 = y_3 - y_4 \tag{19}$$

$$y_8 = y_3 + y_4 \tag{20}$$

$$y_9 = (b + c)(v_5 + v_7) \tag{21}$$

$$y_{10} = cv_5 \tag{22}$$

$$y_{11} = bv_7 \tag{23}$$

$$y_{12} = y_9 - y_{10} - y_{11} \tag{24}$$

$$y_{13} = y_{10} - y_{11} \tag{25}$$

$$u_1 = y_2 - y_7 + y_{12} \tag{26}$$

$$u_2 = y_6 + y_{12} \tag{27}$$

$$u_3 = y_6 - y_{12} \tag{28}$$

$$u_4 = y_2 - y_8 - y_{12} \tag{29}$$

$$u_5 = y_2 + y_7 + y_{13} \tag{30}$$

$$u_6 = y_1 + y_3 + v_4 + y_{13} - v_8 \tag{31}$$

$$u_7 = 0 \tag{32}$$

$$u_8 \quad = \quad 0. \tag{33}$$

This implementation requires 5 multiplications and 22 additions. In this work, however, we are more interested in the number of basic arithmetic operations on the PA-RISC processor (see also [5]). As explained in the introduction, here the term "operation" corresponds to the elementary arithmetic computation of the PA-RISC processor which is either "shift", "add", or "shift and add" (SH1ADD, SH2ADD, and SH3ADD). For example, the computation $z = 1.375x + 1.125y$ is implemented as follows: First, we compute $u = x + 0.5x$ (SH1ADD), then $v = x + 0.25u$ (SH2ADD), afterwards $w = v + y$ (ADD), and finally, $z = w + 0.125y$ (SH3ADD). Thus, overall 4 basic operations are needed in this example.

Let us now return to our problem and calculate the total number of required operations. By developing similar implementation schemes of matrix multiplication for all matrices $J_1, ..., J_8$, we find that the numbers $\{N_i\}$ of operations required to multiply by $\{J_i\}$, $1 \leq i \leq 8$, are given by $N_1 = 18$, $N_2 = 24$, $N_3 = 38$, $N_4 = 39$, $N_5 = 40$, $N_6 = 43$, $N_7 = 44$, and $N_8 = 46$. Since the matrix K_i has a structure similar to that of J_i for every $1 \leq i \leq 8$, multiplication by K_i costs also N_i operations.

When counting the operations in the implementation of eq. (11) or (12), we will use the fact that multiplications by D and D^{-1} can be ignored because these can be absorbed in the MPEG quantizer and dequantizer, respectively. The matrices P and P^{-1} cause only changes in the order of the components so they can be ignored as well. Thus, for a general position reference block (i.e., $1 \leq w \leq 7$, $1 \leq h \leq 7$), we have the following:

1. Six multiplications by B_1 or B_1^t: $6 \times 32 = 192$ operations.

2. Six multiplications by B_2 or B_2^t: $6 \times 32 = 192$ operations.

3. Two multiplications by J_w and K_{8-w}, and one by J_h and K_{8-h}, or vice versa: $8 \cdot (N_h + N_{8-h} + N_w + N_{8-w} + \min\{N_h + N_{8-h}, N_w + N_{8-w}\}$ operations.

4. One 2D-DCT (using eq. (9)): $42 \times 16 = 672$ operations.

Total: $1056 + 8 \cdot (N_h + N_{8-h} + N_w + N_{8-w} + \min\{N_h + N_{8-h}, N_w + N_{8-w}\})$ operations.

Note that we have not counted additions of the products in eqs. (11) and (12) because the different summands are nonzero on disjoint subsets of indices of matrix elements. When the reference block is aligned in the vertical direction only, i.e., $h = 8$ and $1 \leq w \leq 7$, then $K_{8-h} = K_0 = L_0(MA_1A_2A_3)^t = 0$, and therefore eqs. (11) and (12) contain two terms only. Furthermore, since $J_h = J_8 = U_8(MA_1A_2A_3)^t = (MA_1A_2A_3)^t$, eq. (11) degenerates to

$$\hat{\boldsymbol{X}} = (\boldsymbol{X}_1 DPB_1B_2J_w^t + \boldsymbol{X}_2 DPB_1B_2K_{8-w}^t)S^t \tag{34}$$

which requires the following steps:

1. Two multiplications by B_1: $2 \times 32 = 64$ operations.
2. Two multiplications by B_2: $2 \times 32 = 64$ operations.
3. One multiplication by J_w and one by K_{8-w}: $8(N_w + N_{8-w})$ operations.
4. One multiplication by S^t: $8 \times 42 = 336$ operations.

Total: $464 + 8(N_w + N_{8-w})$ operations.

Similarly, for the horizontally aligned case, where $w = 8$ and $1 \leq h \leq 7$, the number of computations is $464+8(N_h+N_{8-h})$. As mentioned earlier, when $w = h = 8$ no computations are required at all since $\hat{\boldsymbol{X}} = \boldsymbol{X}_1$ and hence already given.

By using the above expressions, we find that the number of computations for the worst case values of h and w is 2928, and the average number, assuming a uniform distribution on the pairs $\{(w, h) : 1 \leq w \leq 8, 1 \leq h \leq 8\}$, is 2300.5. On the other hand, the brute-force approach of performing IDCT to $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$, cutting the appropriate reference block in the spatial domain, and transforming it back, requires a total of 4320 operations. This means that the reduction in computational complexity, in comparison to the brute-force method, is 32% for the worst case and 46.8% for the average.

So far we have not assumed that the input DCT matrices are sparse. Typically, a considerable percentage of the DCT blocks have only a few nonzero elements, normally, those corresponding to low spatial frequencies in both directions. For simplicity, we shall refer to a DCT block as *sparse* if only the top left 4×4 quadrant (corresponding to low frequencies) is nonzero.

We have redesigned the implementation of multiplication by J_i and K_i, $1 \leq i \leq 8$, when $\boldsymbol{X}_1, ..., \boldsymbol{X}_4$ are assumed sparse in the above sense, and found that the number of computations is $672 + 8 \cdot (N'_w + N'_{8-w} + N'_h + N'_{8-h})$ for $1 \leq w \leq 7$ and $1 \leq h \leq 7$, $336 + 4 \cdot (N'_w + N'_{8-w})$ for $h = 8$ and $1 \leq w \leq 7$, $336 + 4 \cdot (N'_h + N'_{8-h})$ for $w = 8$ and $1 \leq h \leq 7$, and zero when $w = h = 8$, where $N'_1 = 15$, $N'_2 = 20$, $N'_3 = 26$, $N'_4 = 33$, $N'_5 = 36$, $N'_6 = 40$, $N'_7 = 41$, and, $N'_8 = 42$. This means that the are 1728 computations in the worst case and 1397.2 on the average, corresponding to reductions of 60% and 68%, respectively, compared to the brute force approach.

For comparison with earlier results, Chang and Messerschmitt [2] have shown computation savings only if the DCT matrices are sparse enough and if a large percentage of the reference blocks are aligned at least in one direction. Specifically, these authors introduced three parameters: the reciprocal of the fraction of nonzero coefficients β, the fraction α_1 of reference blocks aligned in one direction, and the fraction α_2 of completely unaligned reference blocks.

Let us consider first the worst case situation in terms of block alignment, i.e., $\alpha_1 = 0$

and $\alpha_2 = 1$. Our above definition of sparseness corresponds to $\beta = 4$. Chang et al. provide exact formulas for the number multiplications and additions associated with their approach in terms of α_1, α_2, β and the block size N ($N = 8$ in JPEG and MPEG). According to these formulas, their approach require in this case 16 mutiplications per pixel and 19 additions per pixel. To compare with PA-RISC processor operations, let us assume that on the average every multiplication requires upto 4 SHIFTs and 3 ADDs and that SHIFTs and ADDs can be done simultaneously. This means that a conservative estimate of the total number of operations per block is $(16 \times 3 + 19) \times 64 = 4288$, which is much larger than 1728 operations (see above) in the proposed approach under the same circumstances.

As another point of comparison, note that a uniform distribution over w and h in our case corresponds to $\alpha_1 = 14/64 = 0.219$ and $\alpha_2 = 49/64 = 0.766$, which is more pessimistic than the upper curve in Fig. 5 of [2], where $\alpha_1 = 0.2$ and $\alpha_2 = 0.1$. Nevertheless, for $\beta = 1$ we are able to speedup the computations by a factor of $4320/2300.5 = 1.87$ compared to 0.6 in [2], and for $\beta = 4$ our speedup is $4320/1397.2 = 3.13$ compared to approximately 2.0 in [2]. Furthermore, if we assume $\alpha_1 = 0.2$ and $\alpha_2 = 0.1$ as in [2] we obtain speedup factors of 9.06 for $\beta = 1$ and about 15 for $\beta = 4$, which means an improvement by an order of magnitude compared to [2].

5. Conclusion

We proposed a fast algorithm that converts motion compensated compressed video into a sequence of DCT-domain blocks corresponding to the spatial domain blocks of the current frame. The heart of the algorithm is in computing the DCT of a possibly unaligned reference block from the DCT's of the blocks with which it intersects. This operation might be applicable in other tasks as well, e.g., translation and filtering. The proposed algorithm saves about 47% of the computations compared to the brute-force approach even without assuming sparseness of the DCT blocks. For typical sparse DCT blocks, where only the top-left 4×4 quadrant is nonzero, the reduction in computational complexity is about 68%. The computational efficiency was also shown to outperform that of a scheme proposed earlier by Chang and Messerschmitt.

References

[1] R. B. Lee et al., "Achieving Realtime Software MPEG Decompression on a Multimedia-Enhanced PA-RISC Processor," *. Proc. Hewlett-Packard Image and Data Compression Conference*, Palo Alto, California, May 1994.

[2] S.-F. Chang and D. G. Messerschmitt, "A New Approach to Decoding and Compositing Motion-Compensated DCT Based Images," *Proc. ICASSP '93*, pp. V.421-V.424, Minneapolis, April 1993.

[3] W. Kou and T. Fjalbrant, "A Direct Computation of DCT Coefficients for a Signal Block Taken from Two Adjacent Blocks," *IEEE Trans. Signal Proc.*, Vol. SP-39, pp. 1692-1695, July 1991.

[4] J. B. Lee and B. G. Lee, "Transform Domain Filtering Based on Pipelining Structure," *IEEE Trans. Signal Proc.*, Vol. SP-40, pp. 2061-2064, August 1992.

[5] N. Merhav and V. Bhaskaran, "A Transform Domain Aproach to Spatial Domain Image Scaling," HPL Technical Report #HPL-94-116, December 1994.

[6] Coding of Moving and Associated Audio. Committee Draft of Standard ISO11172: ISO/MPEG 90/176, December 1990.

[7] Video Codec for Audio Visual Services at px64 Kbits/s. CCITT Recommendation H.261, 1990.

[8] D. le Gall, "MPEG: A Video Compression Standard for Multimedia Applications," *Commun. of the ACM*, Vol. 34, No. 4, pp. 47-58, April 1991.

[9] Y. Arai, T. Agui, and M. Nakajima, "A Fast DCT-SQ Scheme for Images," *Trans. of the IEICE*, E 71(11):1095, November 1988.

[10] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, 1993.