A Fast Algorithm for DCT-Domain Inverse Motion Compensation Based on Shared Information in a Macroblock

Junehwa Song and Boon-Lock Yeo

***Abstract*—The ability to construct intracoded frame from motion-compensated intercoded frames directly in the compressed domain is important for efficient video manipulation and composition. In the context of motion-compensated discrete cosine transform (DCT)-based coding of video as in MPEG video, this problem of *DCT-domain inverse motion compensation* has been studied and, subsequently, improved faster algorithms were proposed. These schemes, however, treat each 8×8 block as a fundamental unit, and do not take into account the fact that in MPEG, a macroblock consists of several such blocks. In this paper, we show how shared information within a macroblock, such as a motion vector and common blocks, can be exploited to yield substantial speedup in computation. Compared to previous brute-force approaches, our algorithms yield about 44% improvement. Our technique is independent of the underlying computational or processor model, and thus can be implemented on top of any optimized solution. We demonstrate an improvement by about 19%, and 13.5% in the worst case, on top of the optimized solutions presented in existing literature.**

***Index Terms*—Compressed domain processing, DCT-domain inverse motion compensation, MPEG video, video composition, video processing.**

I. Introduction

MPEG [4], [5] has been established as a standard for efficient storage and transmission of video. However, the compression schemes based on a combination of discrete cosine transform (DCT) and motion compensation (MC) do not lead to easy manipulation and composition of the compressed video. In both applications of compressed domain processing and composition of compressed video streams from several sources in a network environment, it would be advantageous to convert the MC-DCT intercoded frames into DCT intracoded frames directly in the compressed domain. By converting video streams from one compressed format to the next, we gain in computational efficiency, and eliminate the need for decompression and possible coding delay. We will thus better preserve the quality of video. The computational efficiency also leads to higher throughput in dealing with the enormous data rates in a network environment. Techniques for the conversion of MC-DCT intercoded frames into DCT intracoded frames also form the basis for fast extraction of specially reduced images in MPEG-1 [6] and MPEG-2 video [7].

This problem of *DCT-domain inverse motion compensation*, i.e., the conversion of intercoded frames into intracoded frames directly in the DCT domain without the need for full decompression for MPEG video was studied in Chang and Messerschmitt [1], and subsequently in [2], [3]. The idea of the algorithm in [1] is to represent a target block as a summation of horizontally and/or vertically displaced anchor blocks. Then, the DCT values of the target block is constructed using the precomputed DCT values of the shifting matrices. The general setup is shown in Fig. 1. Here, $P_{\rm ref}$ is the current block of interest, $P_0, \cdots, P_3$ are the four original neighboring blocks from which $P_{\rm ref}$ is derived, and the motion vector is $(\Delta x, \Delta y)$. The shaded regions in $P_0, \cdots, P_3$ are moved by $(\Delta x, \Delta y)$.

We are thus interested in obtaining the DCT representation of block $P_{\rm ref}$ given the DCT representation of P_i and motion vector $(\Delta x, \Delta y)$. If we represent each block as an 8×8 matrix, then we can describe in the spatial domain through matrix multiplications

$$P_{\rm ref} = \sum_{i=0}^{3} S_{i1} P_i S_{i2} \tag{1}$$

where S_{ij} are matrices like

$$L_n = \begin{pmatrix} 0 & 0 \\ I_n & 0 \end{pmatrix} \text{ or } R_n = \begin{pmatrix} 0 & I_n \\ 0 & 0 \end{pmatrix}.$$

Each I_n is an identity matrix of size n. The pre-multiplication shifts the sub-block of interest vertically while post-multiplication shifts the sub-block horizontally.

There are four possible locations of the subblock of interest: upper-left, upper-right, lower-right and lower-left. The actions in terms of matrices are tabulated in Table I.

While the value of S_{ij} is clear from Table I with the given values of h_i and w_i, we will sometime write S_{ij} as a function of h_i and w_i. For example, $S_{01} = S_{01}(h_0, w_0) = S_{01}(h_0)$.

We denote the 2D-DCT of an 8×8 block A as

$$\text{DCT}(A) \stackrel{\text{def}}{=} \hat{A} = TAT^t \tag{2}$$

Manuscript received August 1, 1997; revised November 12, 1999. This work was supported in part by an IBM Cooperative Fellowship, and by NIST/ATP under Contract 70NANB5H1174.

J. Song is with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA (e-mail: junesong@us.ibm.com).

B.-L. Yeo was with IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA and with Microcomputer Research Labs, Intel Corporation, Santa Clara, CA 95052 USA. He is now with EXP.com, Menlo Park, CA 94025 USA (e-mail: byeo@exp.com).

Publisher Item Identifier S 1051-8215(00)06553-8.

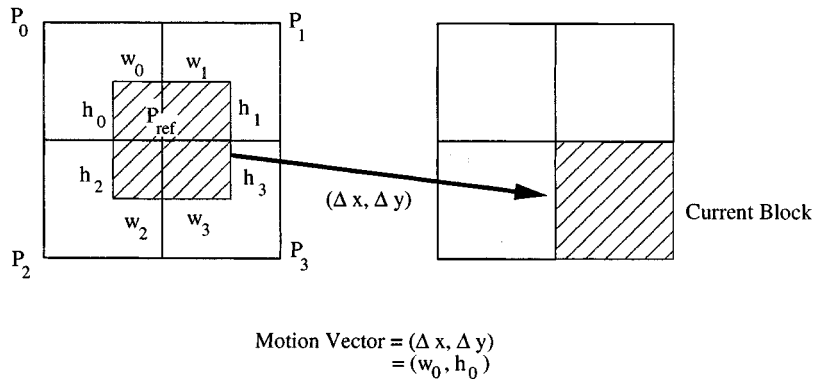

Fig. 1. Reference block ($P_{\rm ref}$), motion vectors and original blocks.

TABLE I
MATRICES S_{i1} AND S_{i2}

Subblock	Position	S_{i1}	S_{i2}
P_0	lower right	$\begin{pmatrix} 0 & I_{h_0} \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ I_{w_0} & 0 \end{pmatrix}$
P_1	lower left	$\begin{pmatrix} 0 & I_{h_1} \\ 0 & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & I_{w_1} \\ 0 & 0 \end{pmatrix}$
P_2	upper right	$\begin{pmatrix} 0 & 0 \\ I_{h_2} & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & 0 \\ I_{w_2} & 0 \end{pmatrix}$
P_3	upper left	$\begin{pmatrix} 0 & 0 \\ I_{h_3} & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & I_{w_3} \\ 0 & 0 \end{pmatrix}$

where T is the 8×8 DCT matrix with entries $t(i, j)$ (i denotes the ith row and j denotes the jth column) given by

$$t(i, j) = \frac{1}{2} k(i) \cos \frac{(2j+1)i\pi}{16} \tag{3}$$

where

$$k(i) = \begin{cases} \dfrac{1}{\sqrt{2}}, & i = 0 \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

Using the fact that

$$\text{DCT}(AB) = \text{DCT}(A)\text{DCT}(B) \tag{5}$$

we can write

$$\widehat{P_{\rm ref}} = \sum_{i=0}^{3} \widehat{S_{i1}} \widehat{P_i} \widehat{S_{i2}}. \tag{6}$$

The shifting matrix based technique shown in (6) was earlier proposed by Plompen *et al.* [8], [9]. They used the formulation to estimate the motion vector in the transform domain and showed that the transform-domain estimation resulted in a better image quality. Kou and Fjallbrant [10] also used similar method for the direct computation of DCT coefficients for a signal block taken from two adjacent blocks. The motivation of [10] was not processing of video, but rather processing of interpolation of speech samples coded in DCT blocks.

In [2], a fast algorithm for computing (6) is proposed. The algorithm is based on the factorization of the DCT matrix T presented in [11]. The factorization is as follows:

$$T = DPB_1B_2EA_1A_2A_3 \tag{7}$$

where

D	diagonal matrix consisting of real entries;
P	permutation matrix;
A_i's, B_i's	sparse matrices with entries 1 and -1;
E	sparse matrix with real entries.

Another fast algorithm is proposed in [3]. This algorithm approximates the DCT values of the shifting matrices by using a finite sum of powers of twos. Then, the matrix multiplications in (6) is implemented by basic integer operations such as *ADD* and *SHIFT*.

In this paper, we provide a novel technique to speed up the DCT-domain inverse motion compensation. The algorithms proposed in the literature so far construct the DCT-domain values of each target block separately [1]–[3]. However, motion compensation in MPEG stream is done on macroblock basis. This means that there is shared information in the predictions of multiple neighboring blocks. We take a few 8×8 blocks (for luminance component, there are four such blocks in each macroblock) as a unit and carefully rearrange the computation steps across correlated target blocks. By this, the hidden shared information is exposed and the overall process of computation is sped up by reusing thus identified shared information across the prediction of multiple blocks. This results in about 44% improvement over the brute-force method in [1].

One important aspect of the proposed method is that it is independent of the underlying computational (or processor) model. This means that the method can be used upon any fast algorithms and yield further speedup in the computation. We show in Section III that our method can be implemented on top of the already fast algorithms proposed in [2] and [3], and improve about 19% and 13.5% on top of these techniques. The percentage improvement numbers are derived based on the use of different computation models and calculation methods in each of the two work. In fact, the technique in [3] is faster than that of [2] by 72%. Thus, when our technique is applied on top of that in [3], using the same calculation methods of [3], we can improve by another 13.5%.

Another advantage of the method is in its flexibility. Since it is not hard-wired to any specific filtering or approximation technique, it can be easily combined with any specific filtering algorithms. While the focus of this paper is on reconstructing the full frames in DCT domains, the techniques apply to reconstruction of frames of reduced resolution [6], [7].

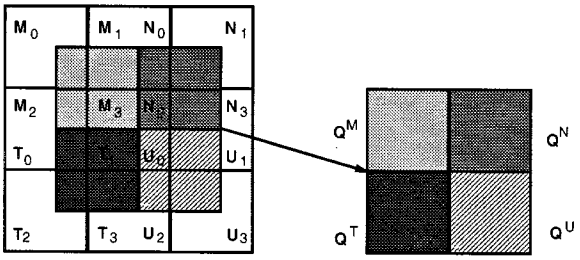

Fig. 2. Prediction of a 16×16 macroblock.

The rest of this paper is organized as follows. In Section II, we present the key ideas of identifying and utilizing shared information, including common motion vectors and blocks. We show in Section III that our method can be implemented on top of the two already fast algorithms proposed in [2], [3] and further improve their performance. Finally, we conclude the paper in Section IV. We refer the readers to [4], [5], [12], [13] for details on MPEG.

II. Macroblock-Based Prediction: Utilizing Shared Information

The DCT-domain inverse motion compensation in (6) constructs the DCT-domain values of each target block separately. The method requires the computation of the DCT domain values of its contributing blocks. In general, a target block is predicted from (up to) four anchor blocks. Therefore, the DCT-domain prediction of a target block requires the construction of DCT-domain values of (up to) four contributing blocks. However, in many cases, anchor blocks can be shared across multiple target blocks. This means that there is shared information in the predictions of multiple blocks. Therefore, careful rearrangement of computation steps across correlated target blocks can speed up the overall process of computation.

From (6), the computation of DCT-domain values of contributing blocks is a special case of pre- and post-multiplication of an 8×8 data block with two 8×8 matrices, where the matrices can be preprocessed. Given an 8×8 data block B and two 8×8 matrices A and C, the computation of the matrix multiplication of the type ABC will be called a *TM* operation. In the following, *TM* operation is taken as the unit to measure and compare the computational complexity of DCT-domain inverse motion compensation.

Fig. 2 shows the prediction of a 16×16 macroblock Q. Each of the four 8×8 target blocks Q^M, Q^N, Q^T, and Q^U is predicted from its four anchor blocks, M_i, N_i, T_i, and U_i, $i = 0, 1, 2, 3$, respectively. For the prediction of each target block, the DCT-domain values of the four contributing blocks need to be computed and added. Since the computation of the DCT-domain value of each contributing block requires a *TM* operation, 16 *TM* operations are required to predict the whole macroblock. However, note that the predictions of four target blocks are strongly correlated to each other and do not have to be computed independently. We will now state three observations below related to the setup of these 16 contributing 8×8 blocks. These three observations form the foundations of our fast algorithm, described in the subsequent sections.

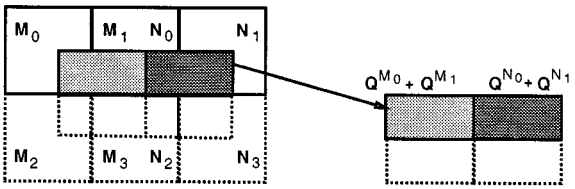

Fig. 3. Prediction of two horizontally neighboring regions.

First, although there are totally 16 contributing blocks for each macroblock, there are only 9 different anchor blocks, and five of them are shared among multiple target blocks, since the target blocks belong to the same macroblock and are predicted using the same motion vector. That is, the target blocks Q^M and Q^N share two anchor blocks, i.e., $M_1 = N_0$ and $M_3 = N_2$. Similarly, Q^M and Q^T share $M_2 = T_0$ and $M_3 = T_1$, etc.

Second, the vertical and horizontal displacements of participating subblock within an anchor block are pair-wise identical. Therefore, we have

$$S_{01} = S_{11}, \quad S_{21} = S_{31}, \quad S_{02} = S_{22}, \quad S_{12} = S_{32}.$$

Furthermore, we will make use of the following properties:

$$S_{01} + S_{31} = P^1, \quad S_{02} + S_{12} = P^0$$

for some permutation matrices P^0 and P^1.

Third, the vertical and horizontal displacements of each participating subblock within the anchor block are identical across the four target blocks, Q^M, Q^N, Q^T, and Q^U, i.e., $w^{M_i} = w^{N_i} = w^{T_i} = w^{U_i}$ and $h^{M_i} = h^{N_i} = h^{T_i} = h^{U_i}$ for $0 \le i \le 3$, where w's and h's superscribed by the anchor block (such as w^{M_i} and h^{M_i}) denote the horizontal and vertical displacements of anchor block.

In the following, a fast algorithm for DCT-domain inverse motion compensation is described based on the above three observations. We will show that in constructing the whole 16×16 target macroblock, the number of *TM* operations is reduced to nine, resulting in about 44% of improvement.

A. Computation of Contribution of Two Neighboring Regions

In Fig. 3, the step to predict upper regions of two horizontally neighboring target blocks Q^M and Q^N is shown in spatial domain. The upper regions of the target blocks can be computed as the addition of two contributing blocks, i.e., $Q^{M_0} + Q^{M_1}$ and $Q^{N_0} + Q^{N_1}$ respectively, from the anchor blocks M_0, M_1 (=N_0), and N_1.

The prediction of $Q^{M_0} + Q^{M_1}$ can be rewritten as follows:

$$\begin{aligned} Q^{M_0} + Q^{M_1} &= S_{01} M_0 S_{02} + S_{11} M_1 S_{12} && (8)\\ &= S_{01}(M_0 S_{02} + M_1 S_{12}) && (9)\\ &= S_{01}(M_0 S_{02} + M_1(P^0 - S_{02})) \\ &\quad (P^0 \stackrel{\text{def}}{=} S_{02} + S_{12}) && (10)\\ &= S_{01}(M_0 - M_1) S_{02} + S_{01} M_1 P^0. && (11) \end{aligned}$$

Assume that $\widehat{Q^{M_0}} + \widehat{Q^{M_1}}$ has already been computed using (11), and $\widehat{Q^{N_0}} + \widehat{Q^{N_1}}$ is currently sought for. The derivation of

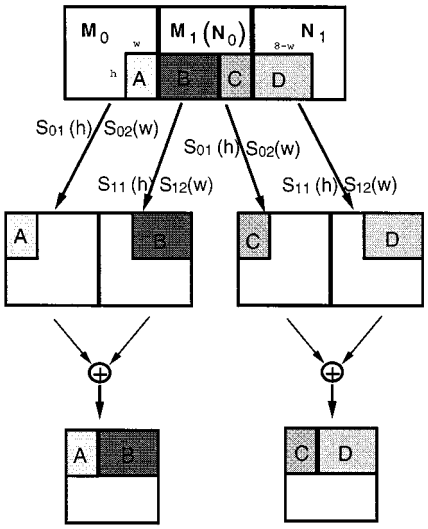

Fig. 4. Brute-force computation.

(10) can be also done as $S_{01}(M_0(P^0 - S_{12}) + M_1 S_{12})$. Therefore

$$Q^{M_0} + Q^{M_1} = S_{01}(M_1 - M_0)S_{12} + S_{01}M_0P^0. \quad (12)$$

The equation for the target block Q^N corresponding to (12) is

$$Q^{N_0} + Q^{N_1} = S_{01}(N_1 - N_0)S_{12} + S_{01}N_0P^0. \quad (13)$$

Note that the DCT-domain value for the second term of (13), i.e., $\widehat{S_{01}N_0P^0}$ has already been computed as $\widehat{S_{01}M_1P^0}$ using (11). Therefore, only $\widehat{S_{01}(N_1 - N_0)S_{12}}$ needs to be additionally computed to get $\widehat{Q^{N_0}} + \widehat{Q^{N_1}}$, resulting in three *TM* operations to compute both $\widehat{Q^{M_0}} + \widehat{Q^{M_1}}$ and $\widehat{Q^{N_0}} + \widehat{Q^{N_1}}$.

The idea behind this arrangement to identify shared information is illustrated in Figs. 4 and 5. In Fig. 4, the contributions of M_0, M_1, N_0 and N_1 are separately computed. The movement of subblocks labeled A through D through the effects of pre- and post-matrix multiplications of S_{ij} is illustrated in Fig. 4. The pair of matrix S_{i1} and S_{i2} associated with each arrow is used for pre- and post-multiplication of the anchor, respectively. In Fig. 5, the sharing of information in M_1 and N_0 is exploited and contrasted with the approach in Fig. 4. The common term in (11) and (13), $S_{01}M_1P^0 = S_{01}N_0P^0$, has the effect of horizontally flipping the location of subblock B and C and then vertically moving the two blocks by h.

A similar idea can be applied to any pair of regions which are vertically or horizontally neighboring. For example, for the prediction of $\widehat{Q^{M_0}} + \widehat{Q^{M_2}}$ and $\widehat{Q^{T_0}} + \widehat{Q^{T_2}}$

$$\begin{aligned} Q^{M_0} + Q^{M_2} &= S_{01}M_0S_{02} + S_{21}M_2S_{22} \\ &= (S_{01}M_0 + S_{21}M_2)S_{02} \\ &= (S_{01}M_0 + (P^1 - S_{01})M_2)S_{02} \\ &= S_{01}(M_0 - M_2)S_{02} + P^1M_2S_{02} \quad (14) \\ Q^{T_0} + Q^{T_2} &= ((P^1 - S_{21})T_0 + S_{21})T_2S_{02} \\ &= S_{21}(T_2 - T_0)S_{02} + P^1T_0S_{02}. \quad (15) \end{aligned}$$

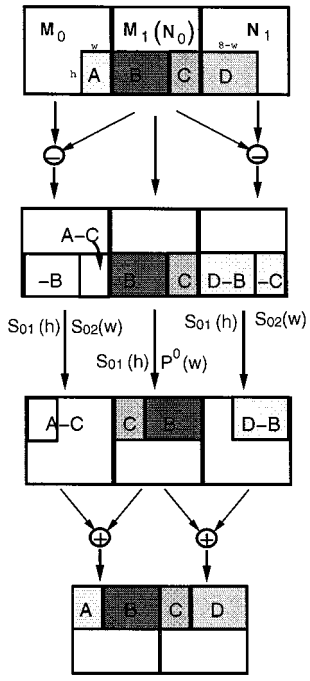

Fig. 5. Utilizing shared information.

The second terms of the two equations are the same since $M_2 = T_0$.

B. Prediction of Two Neighboring Blocks

We will now examine how the prediction of two horizontally or vertically neighboring blocks can be facilitated.

The prediction of the whole block, i.e., $\sum_{i=0}^{3} Q^{M_i}$, can be rewritten as follows:

$$\begin{aligned} Q^M &= \sum_{i=0}^{3} Q^{M_i} \quad (16) \\ &= S_{01}(M_0 - M_1)S_{02} + S_{01}M_1P^0 \\ &\quad + S_{21}(M_2 - M_3)S_{02} + S_{21}M_3P^0 \quad (17) \\ &= S_{01}(M_0 - M_1)S_{02} + S_{01}M_1P^0 + (P^1 - S_{01}) \\ &\quad \cdot (M_2 - M_3)S_{02} + (P^1 - S_{01})M_3P^0 \quad (18) \\ &= S_{01}(M_0 - M_1 - M_2 + M_3)S_{02} + S_{01}(M_1 - M_3) \\ &\quad \cdot P^0 + P^1(M_2 - M_3)S_{02} + P^1M_3P^0. \quad (19) \end{aligned}$$

Consider the prediction of $\widehat{Q^N}$, assuming that $\widehat{Q^M}$ has already been predicted using (19). For this, we first note that, by changing the orders of the terms in the computation, the derivation of Q^M in (19) can be rearranged as follows:

$$\begin{aligned} Q^M &= S_{01}(M_1 - M_0)S_{12} + S_{01}M_0P^0 \\ &\quad + S_{21}(M_3 - M_2)S_{12} + S_{21}M_2P^0 \\ &= S_{01}(M_1 - M_0)S_{12} + S_{01}M_0P^0 \\ &\quad + (P^1 - S_{01})(M_3 - M_2)S_{12} + (P^1 - S_{01})M_2P^0 \\ &= S_{01}(M_1 - M_0 - M_3 + M_2)S_{12} + S_{01}(M_0 - M_2) \cdot P^0 \\ &\quad + P^1(M_3 - M_2)S_{12} + P^1M_2P^0. \quad (20) \end{aligned}$$

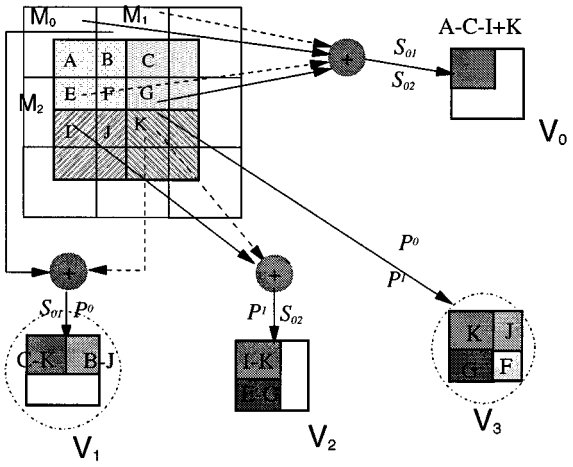

Fig. 6. Steps to unveil shared information while computing Q^M: in Figs. 6–9, a dotted arrow denotes a subtraction of an anchor block and a straight arrow represents an addition. Also, the shared terms are marked by dotted circles.

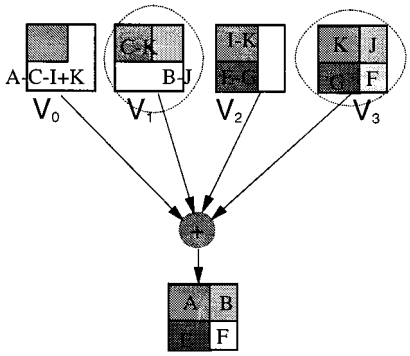

Fig. 7. Computation of Q^M.

The equation for Q^N equivalent to (20) is

$$\begin{aligned} Q^N &= \sum_{i=0}^{3} Q^{N_i} \\ &= S_{01}(N_1 - N_0 - N_3 + N_2)S_{12} + S_{01}(N_0 - N_2) \cdot P^0 \\ &\quad + P^1(N_3 - N_2)S_{12} + P^1 N_2 P^0. \end{aligned} \tag{21}$$

Again, the second and the fourth terms of (21) are the same as those of (19) and can be reused, since $N_0 = M_1$ and $N_2 = M_3$. Therefore, $\widehat{S_{01}}(\widehat{N_1} - \widehat{N_0} - \widehat{N_3} + \widehat{N_2})\widehat{S_{12}}$ and $\widehat{P^1}(\widehat{N_3} - \widehat{N_2})\widehat{S_{12}}$, corresponding to the first and the third terms need to be additionally computed, resulting in 6 *TM* operations for both $\widehat{Q^M}$ and $\widehat{Q^N}$.

Figs. 6–9 illustrate how the shared computation steps are unveiled in the computation of Q^M and Q^N. In the figures, the 16 subblocks which contributes to the prediction of a macroblock (see Fig. 2) is labeled by A through L. A dotted arrow denotes the subtraction of an anchor block while a straight arrow denotes the addition of a corresponding anchor block. Fig. 6 shows the computation of the four terms in (19). The first term in Fig. 6 is computed by first taking $M_0 - M_1 - M_2 + M_3$ and then by shifting it to the upper left corner (pre- and post-multiplication by S_{01} and S_{02}). The result is represented by V_0. The other three terms are also computed by appropriately adding or subtracting

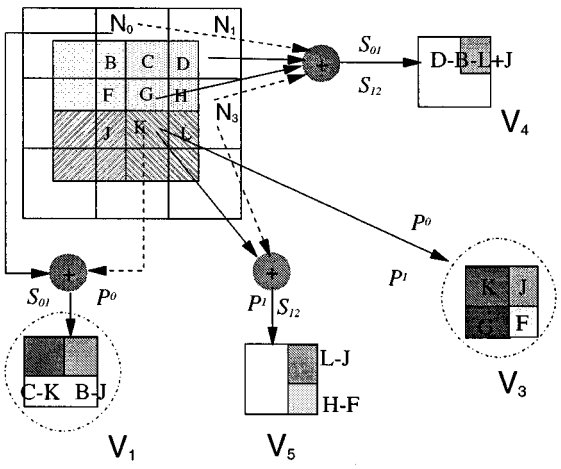

Fig. 8. Steps to unveil shared information while computing Q^N.

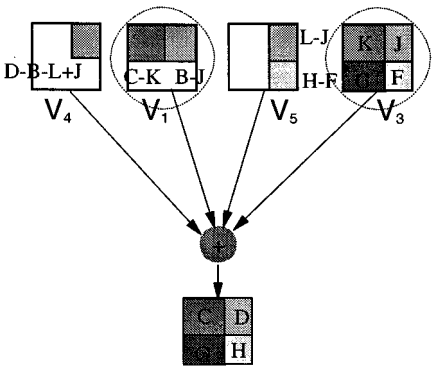

Fig. 9. Computation of Q^N.

anchor blocks, M_i, $0 \le i \le 3$, and then multiplying shift matrices or permutation matrices. They are marked by V_1, V_2, and V_3. Similarly, Fig. 8 shows the computation of the terms in Q^N, as written in (21). Note that the multiplication of the permutation matrix P^0 or P^1 has the effect of the horizontal or vertical flipping of subblocks, and results in the common terms V_1 and V_3, as marked by the dotted circles. Therefore, in computing Q^M and Q^N, as shown in Figs. 7 and 9, those two terms are computed only once and shared.

Similar idea can also be applied in the prediction of two vertically neighboring blocks such as Q^M and Q^T or Q^N and Q^U and the same amount of improvement on the computation can be achieved. The difference lies only in the arrangement of the equations, which can be easily deduced.

C. Prediction of the Whole Macroblock

Now, consider the prediction of the whole macroblock composed of Q^M, Q^N, Q^T, and Q^U. Similar to the derivation of (20), we can further rearrange the steps of the derivation of Q^M differently and get the corresponding equivalent equations for Q^T and Q^U as follows:

$$\begin{aligned} Q^T &= \sum_{i=0}^{3} Q^{T_i} \\ &= S_{21}(T_2 - T_3 - T_0 + T_1)S_{02} + S_{21}(T_3 - T_1) \cdot P^0 \\ &\quad + P^1(T_0 - T_1)S_{02} + P^1 T_1 P^0 \end{aligned} \tag{22}$$

$$Q^U = \sum_{i=0}^{3} Q^{U_i}$$
$$= S_{21}(U_3 - U_2 - U_1 + U_0)S_{12} + S_{21}(U_2 - U_0)\cdot P^0 + P^1(U_1 - U_0)S_{12} + P^1 U_0 P^0. \quad (23)$$

As previously described, both $\widehat{Q^M}$ and $\widehat{Q^N}$ can be computed by six *TM* operations. Now compare (22) and (23) with (19) and (21). The third and the fourth terms of (22) are the same as those of (19), since $T_0 = M_2$ and $T_1 = M_3$. The second and the fourth terms of (23) are the same as those of (22), since $T_1 = U_0$ and $T_3 = U_2$. Finally, the third term of (23) is the same as that of (21), since $N_2 = U_0$ and $N_3 = U_1$. Therefore, two additional *TM* operations are required to compute $\widehat{Q^T}$ and only one for $\widehat{Q^U}$, resulting in a total of $9(=4+2+2+1)$ *TM* operations for the whole macroblock. This yields a gain of $(1 - 7/16) \times 100 \approx 44\%$.

D. Special Cases: Vertically or Horizontally Aligned Blocks

Now let us consider special cases when the contributing subblocks are vertically or horizontally aligned with the corresponding anchor blocks, i.e., $h = 0$ or $w = 0$. In these cases, each target block is composed from two anchor blocks, and two vertically or horizontally neighboring target blocks share an anchor block. We now analyze the case when blocks are vertically aligned and show that the construction of two horizontally neighboring blocks can be done with three *TM* operations. The situation for horizontally aligned blocks is similar.

When Q^{M_i} is vertically aligned, we have $h^{M_i} = 8$, and thus $S_{01} = S_{11} = I$. We rewrite (8) as

$$Q^{M_0} + Q^{M_1} = M_0 S_{02} + M_1 S_{12} \quad (24)$$
$$= (M_0 - M_1)S_{02} + M_1 P^0. \quad (25)$$

Similarly, we have $h^{N_i} = 8$, and we rewrite (13) as

$$Q^{N_0} + Q^{N_1} = (N_1 - N_0)S_{12} + M_1 P^0. \quad (26)$$

Thus, to compute two blocks, it will take three *TM* operations using shared information versus four *TM* operations using the straightforward approach. This yields a gain of 25%. Note that in this case, T_i and U_i do not contribute to the computation of Q^M.

III. Fast Computations

We will now present in detail how our proposed fast techniques based on shared information can be applied to existing fast algorithms for DCT-domain inverse motion compensation. In particular, we will focus on two techniques proposed in [2] and [3]. Each of the techniques has its own method of computing the performance, and we will use the same computing method for comparing the improvement with each technique.

A. Factorization of DCT Matrix

The speedup of the algorithm in [2] has been achieved based on two observations. First, they utilize the fact that vertical (horizontal) shifting of two horizontally (vertically) neighboring anchor blocks are the same, i.e., $S_{01} = S_{11}$, $S_{21} = S_{31}$, $S_{02} = S_{22}$, $S_{12} = S_{32}$. Second, instead of fully pre-computing $\widehat{S_{ij}}$, $0 \le i \le 3$, $1 \le j \le 2$, they factorize those matrices into relatively sparse matrices.

The matrices to be precomputed are

$$J_l = U_l(EA_1A_2A_3)^t$$
$$K_l = L_l(EA_1A_2A_3)^t, \qquad 1 \le l \le 8$$

where matrices E, A_1, A_2, and A_3 are the components of the factorization in (7), and $U_h \stackrel{\text{def}}{=} S_{11}(h) = S_{21}(h)$ and $L_w \stackrel{\text{def}}{=} S_{12}(w) = S_{32}(w)$. Then, $\widehat{S_{ij}}$, $0 \le i \le 3$ and $1 \le j \le 2$, can be factorized as follows. In the case of $S_{ij} = L_l$ for a certain l, $1 \le l \le 8$

$$\widehat{S_{ij}} = TL_lT^t$$
$$= TL_l(A_1A_2A_3E)^t B_2^t B_1^t P^t D^t$$
$$= TK_l B_2^t B_1^t P^t D^t, \text{ or} \quad (27)$$
$$\widehat{S_{ij}} = TL_lT^t$$
$$= DPB_1B_2EA_1A_2A_3L_lT^t$$
$$= DPB_1B_2K_l^tT^t. \quad (28)$$

Now, the prediction of a target block Q^M from anchor blocks M_i, $0 \le i \le 3$ is computed by the following (or its dual):

$$\widehat{Q^M}$$
$$= (\widehat{S_{01}}\widehat{M_0} + \widehat{S_{21}}\widehat{M_2})\widehat{S_{02}} + (\widehat{S_{11}}\widehat{M_1} + \widehat{S_{31}}\widehat{M_3})\widehat{S_{32}} \quad (29)$$
$$= T[(J_hB_2^tB_1^tP^tD\widehat{M_0} + K_{8-h}B_2^tB_1^tP^tD\widehat{M_2})DPB_1B_2J_w^t + (J_hB_2^tB_1^tP^tD\widehat{M_1} + K_{8-h}B_2^tB_1^tP^tD\widehat{M_3}) \cdot DPB_1B_2K_{8-w}^t]T^t. \quad (30)$$

Note that this method does not utilize the shared information across neighboring target blocks. Therefore, we can still speed up by rearranging the computation steps. First, consider the prediction of two horizontally neighboring blocks Q^M and Q^N in Fig. 2. The equations for the predictions of Q^M and Q^N in spatial domain can be rewritten as follows:

$$Q^M = [S_{01}(M_0 - M_1) + S_{21}(M_2 - M_3)]S_{02} + [S_{01}M_1 + S_{21}M_3]P^0 \quad (31)$$
$$Q^N = [S_{01}(N_1 - N_0) + S_{21}(N_3 - N_2)]S_{12} + [S_{01}N_0 + S_{21}N_2]P^0. \quad (32)$$

The DCT-domain versions of (31) and (32) have similar structures to (29). Therefore, we can apply the same factorization as in (30). The only difference is the permutation matrix P^0 in (31) and (32). This problem is solved as follows:

$$TP^0T^t = DPB_1B_2(EA_1A_2A_3)P^0T^t$$
$$= DPB_1B_2(EA_1A_2A_3I)P^0T^t$$
$$= DPB_1B_2(EA_1A_2A_3L_8)P^0T^t$$
$$= DPB_1B_2K_8^tP^0T^t. \quad (33)$$

Similarly

$$TP^0T^t = TP^{0t}K_8B_2^tB_1^tP^tD. \tag{34}$$

Therefore, we can apply the factorizations in (29) and (30), as shown in (35)–(37), at the bottom of the page. Note that one of the two terms inside the DCT matrices T and T^t can be reused in predicting $\widehat{Q^N}$. Therefore, disregarding the cost of DCT transformation, we can save $1/4 = 25\%$ of computation per block. Now let us analyze the computational cost following the analysis method in [2] and precisely compare two algorithms. In [2], the performance is measured by counting the number of basic arithmetic operations in the PA-RISC processor, such as ADD, SHIFT, and SHIFT-ADD. The worst-case analysis in the paper can be summarized as follows. The factorized versions of TS_{ij} and $S_{ij}T^t$, such as $J_hB_2^tB_1^tP^tD$, are denoted as S'_{ij} and S''_{ij}, respectively. We refer the readers to [2] for more details about the actual derivations.

1) Cost of a 2-D DCT: 672 arithmetic operations. The 2-D DCT used in [2] is based on the fastest known algorithm for 8-point DCT due to Arai *et al.* [11] and built upon the work of Winograd [14]. See also [15] for in-depth treatment of fast DCT algorithms.
2) Worst-case cost of pre- or post-multiplication by S'_{ij} or S''_{ij}: $39 \times 8 + 32 \times 2 = 376$ arithmetic operations.
3) Worst-case cost of predicting a block: six multiplications by S'_{ij} or S''_{ij} + three matrix additions + one 2D-DCT $=376 \times 6 + 3 \times 64 + 672 = 3120$ arithmetic operations.

Note that in (30), three matrix additions are used. In [2], the total number of arithmetic operations did not include these matrix additions. However, we include the cost of matrix additions ($64 \times 3 = 192$) to compare the two methods in a more precise way so as to count the effect of extra additions required in our method, i.e., $\widehat{M_0} - \widehat{M_1}$, $\widehat{M_2} - \widehat{M_3}$, etc., as in (37), shown at the bottom of the page. The purpose of following [2]'s scheme of counting operations is for direct comparisons of our proposed scheme with [2]'s scheme. It is reasonable to assume that the counting model chosen in [2] also applies to other architecture.

Now consider the effect of permutation matrix as in (33), (34), and (37). We denote the factorized versions of TP^0 and P^0T^t in (33) and (34) as P'^0 and P''^0, respectively. That is, $P'^0 = DPB_1B_2K_8^tP^0$ and $P''^0 = P^0K_8B_2^tB_1^tP^tD$. In [2], the cost of multiplying a permutation matrix was ignored, since it causes only changes in the order of matrix components. Therefore, the effect of multiplying matrix P^0 can be ignored as well. From [2], the number of operations for $DPB_1B_2K_8^t$ is 432 arithmetic operations, which is the cost of multiplying P'^0 or P''^0. Now, the total number of operations to compute $\widehat{Q^M}$ and $\widehat{Q^N}$ in the worst case when utilizing the shared information as in (37), shown at the bottom of the page, is 5360, as follows.

1) $\widehat{Q^M}$: five multiplications by S'_{ij} or S''_{ij} + one multiplication by P'^0 or P''^0 + three matrix additions + one 2D DCT: $376 \times 5 + 432 + 64 \times 3 + 672 = 3176$.
2) $\widehat{Q^N}$: three multiplications by S'_{ij} or S''_{ij} + two matrix additions + one 2D DCT: $376 \times 3 + 64 \times 2 + 672 = 1928$.
3) four extra matrix additions ($\widehat{M_0} - \widehat{M_1}$, $\widehat{M_2} - \widehat{M_3}$, $\widehat{N_1} - \widehat{N_0}$, and $\widehat{N_3} - \widehat{N_2}$): $64 \times 4 = 256$.

Therefore, to predict one block, $2680 (= 5360/2)$ operations are used, which is $(3120 - 2680)/3120 = 14.1\%$ improvement over the method in [2] in the worst case.

As before, if we consider the prediction of the whole macroblock, we can find out more shared terms. To simplify the notations, we rewrite (19) and (21)–(23) (which are for the predictions of Q^M, Q^N, Q^T, and Q^U in Fig. 2) by renaming matrices

$$\begin{aligned}
Q^M &= S_{01}Z_0S_{02} + S_{01}Y_0P^0 + P^1Y_1S_{02} + P^1X_0P^0 \\
Q^N &= S_{01}Z_1S_{12} + S_{01}Y_0P^0 + P^1Y_2S_{12} + P^1X_0P^0 \\
Q^T &= S_{21}Z_2S_{02} + S_{21}Y_3P^0 + P^1Y_1S_{02} + P^1X_0P^0 \\
Q^U &= S_{21}Z_3S_{12} + S_{21}Y_3P^0 + P^1Y_2S_{12} + P^1X_0P^0,
\end{aligned}$$

where

$$\begin{aligned}
Z_0 &= M_0 - M_1 - M_2 + M_3 \\
Z_1 &= N_1 - N_0 - N_3 + N_2 \\
Z_2 &= T_2 - T_3 - T_0 + T_1 \\
Z_3 &= U_3 - U_2 - U_1 + U_0 \\
Y_0 &= M_1 - M_3 = N_0 - N_2 \\
Y_1 &= M_2 - M_3 = T_0 - T_1 \\
Y_2 &= N_3 - N_2 = U_1 - U_0 \\
Y_3 &= T_3 - T_1 = U_2 - U_0 \\
X_0 &= M_3 = N_2 = T_1 = U_0.
\end{aligned}$$

Now for the prediction of $\widehat{Q^M}$, $\widehat{Q^N}$, $\widehat{Q^T}$ and $\widehat{Q^U}$, the factorization of (29) and (31) can be applied to the following rearranged equations and intermediate results for the shared terms can be reused:

$$Q^M = [S_{01}Z_0 + P^1Y_1]S_{02} + [S_{01}Y_0 + P^1X_0]P^0 \tag{38}$$

$$Q^N = [S_{01}Z_1 + P^1Y_2]S_{12} + \overbrace{[S_{01}Y_0 + P^1X_0]P^0}^{\text{shared}} \tag{39}$$

$$Q^T = [S_{21}Z_2 + \overbrace{P^1Y_1}^{\text{shared}}]S_{02} + [S_{21}Y_3 + \overbrace{P^1X_0}^{\text{shared}}]P^0 \tag{40}$$

$$Q^U = [S_{21}Z_3 + \overbrace{P^1Y_2}^{\text{shared}}]S_{12} + \overbrace{[S_{21}Y_3 + P^1X_0]P^0}^{\text{shared}}. \tag{41}$$

$$\widehat{Q^M} = \{\widehat{S_{01}}(\widehat{M_0} - \widehat{M_1}) + \widehat{S_{21}}(\widehat{M_2} - \widehat{M_3})\}\widehat{S_{02}} + \{\widehat{S_{01}}\widehat{M_1} + \widehat{S_{21}}\widehat{M_3}\}\widehat{P^0} \tag{35}$$

$$= T[\{J_hB_2^tB_1^tP^tD(\widehat{M_0} - \widehat{M_1}) + K_{8-h}B_2^tB_1^tP^tD(\widehat{M_2} - \widehat{M_3})\}DPB_1B_2J_w^t \tag{36}$$

$$+ \overbrace{\{J_hB_2^tB_1^tP^tD\widehat{M_1} + K_{8-h}B_2^tB_1^tP^tD\widehat{M_3}\}DPB_1B_2K_8^tP^0}^{\text{shared}}]T^t \tag{37}$$

Now consider the extra matrix additions to compute, Y_i, $0 \le i \le 3$, and Z_j, $0 \le j \le 3$. Computation of each Y_i takes one matrix addition. Once Y_i's are available, Z_0 and Z_1 can be computed with five matrix additions by arranging the computation as follows:

$$Z_0 = M_0 - Y_1 - (Y_0 + X_0)$$
$$Z_1 = N_1 - Y_2 - (Y_0 + X_0).$$

Similarly, Z_2 and Z_3 can be computed with five matrix additions. Therefore, the total number of extra matrix additions is 14. The number of arithmetic operations required can be counted as follows.

1) Cost to compute $\widehat{Q_M}$: three multiplications by S'_{ij} or S''_{ij} + three multiplications by P'^0 or P''^0 + three matrix additions + one 2D-DCT $=376 \times 3 + (376 + 56) \times 3 + 64 \times 3 + 672 = 3288$.
2) Cost to compute $\widehat{Q_N}$: two multiplications by S'_{ij} or S''_{ij} + one multiplication by P'^0 or P''^0 + two matrix additions + one 2D-DCT $=376 \times 2 + (376 + 56) + 64 \times 2 + 672 = 1984$.
3) Cost to compute $\widehat{Q_T}$: three multiplications by S'_{ij} or S''_{ij} + one multiplication by P'^0 or P''^0 + three matrix additions + one 2D-DCT $=(376 \times 3) + (376 + 56) + 64 \times 3 + 672 = 2424$.
4) Cost to compute $\widehat{Q_U}$: two multiplications by S'_{ij} or S''_{ij} + two matrix additions + one 2D-DCT $=376 \times 2 + 64 \times 2 + 672 = 1552$.
5) Cost for extra matrix additions (to compute $\widehat{X_i}$, $\widehat{Y_i}$, $\widehat{Z_i}$): 14 matrix additions $=64 \times 14 = 896$.

Therefore, the total cost to compute four blocks $=3288+1984+2424+1552+896 = 10\,144$, which amounts to $10\,144/4 = 2536$ for one block. Compared to the algorithm in [2], we obtain an improvement of $(1 - 2536/3120) \times 100 = 18.72\%$ in the worst case.

Now, let us again consider the special cases when we have vertically or horizontally aligned blocks. As shown in Section II-D, the chance of speedup, in these cases, is less, since number of shared anchor blocks are reduced. However, we can still uncover some shared information and gain some amount of speedup. From [2], the cost in the worst case for these cases is 1152 operations. To demonstrate the improvement using shared information, we first write (25) in terms of the above factorizations

$$\widehat{Q^M} = [(\widehat{M_0} - \widehat{M_1})DPB_1B_2J_w^t + \widehat{M_1}DPB_1B_2K_8^tP_0]T^t. \tag{42}$$

A similar equation holds for (26)

$$\widehat{Q^N} = [(\widehat{N_1} - \widehat{N_0})DPB_1B_2K_{8-w}^t + \overbrace{\widehat{M_1}DPB_1B_2K_8^tP_0}^{\text{shared}}]T^t. \tag{43}$$

In the worst case, the total number of operations for (42) becomes: one multiplication by S'_{ij} + one multiplication by P'^0 + two matrix additions + one 1D DCT, giving $376+432+64\times2+336 = 1272$. For (43), taking into account the shared information used, we have: one multiplication by S'_{ij} + two matrix additions + one 1-D DCT, giving $376+64\times2+336 = 840$. Therefore, the total cost to compute two blocks $=1272+840 = 2112$, which amounts to $2112/2 = 1056$ operations per block. Compared to the algorithm in [2], we obtain an improvement of $(1 - 1056/1152) \times 100 = 8.3\%$.

B. Approximation of Shifting Matrices

Another fast algorithm has been proposed in [3]. It achieves 72% improvement over the algorithm in [2] and 81% over the DCT/IDCT method based on the fastest existing 8-point DCT [11]. The idea of utilizing shared information can also be used on top of this fast algorithm and can further speed up by 13.5%.

The basic idea of the algorithm in [3] is as follows. Each entry of the DCT values of the shifting matrices in (6) is approximated by a finite sum of powers in two's with a maximum distortion of 1/32. The matrix multiplications in (6) can then be implemented by basic integer operations such as *ADD* and *SHIFT*. The data representation and the order of computation are optimized to reduce the number of *SHIFT* operations. It is shown that a DCT block can be constructed with only 810 arithmetic operations of *ADD*'s and *SHIFTS*'s.

As before, we can apply our proposed technique by utilizing shared information as in (31) and (32). The DCT values of the permuation matrices, $DCT(P^0)$ or $DCT(P^1)$ are also represented as summations of power of two. With some lineup of the computation steps, the pre- or post-multiplication by $DCT(P^0)$ or $DCT(P^1)$ can be performed with between 82 and 116 arithmatic operations. In the end, the number of arithmetic operations required to compute both blocks in (31) and (32) is 1401. Therefore, for one block, it requires 700.5 operations, which means 13.5% (1–700.1/810) improvement. Note that applying (19) and (21)–(23) to utilize more common terms by considering four neighboring blocks does not help improve the performance any further, since the cost of extra matrix additions (64 additions per matrix addition) gets significant compared to the cost of the original algorithm which independently constructs each block.

IV. Conclusion

In this paper, we provide a novel technique to significantly speed up the DCT-domain inverse motion compensation based on the exploitation of shared information such as motion vectors and common block within a macro-block. Our technique results in about 44% improvement over the brute-force method proposed in [1]. A key advantage of our proposed technique is that it is independent of the underlying computational or processor model, and thus can be implemented on top of any optimized solution. We have shown that our method can further improve upon the optimized results of [2] and [3] by about 19% and 13.5%, respectively.

Acknowledgment

The authors thank the reviewers for their valuable comments.

References

[1] S. F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video," *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1–11, Jan. 1995.

[2] N. Merhav and V. Bhaskaran, "A fast algorithm for DCT domain inverse motion compensation," in *Proc. IEEE Int. Conf. Acoustics, Speech, Signal Processing*, vol. 4, May 1996, pp. 2307–2310.
[3] P. A. A. Assuncao and M. Ghanbari, "Transcoding of MPEG-2 video in the frequency domain," in *ICASSP 1997*, 1997, pp. 2633–2636.
[4] *Coding of Moving Pictures and Associated Audio for Digital Storage Media up to 1.5 bits/s*, ISO/IEC JTC1 CD 11172, 1992.
[5] *Generic Coding of Moving Pictures and Associated Audio*, ISO/IEC JTC1 CD 13 818, 1994.
[6] B. L. Yeo and B. Liu, "Rapid scene analysis on compressed videos," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 533–544, Dec. 1995.
[7] J. Song and B. L. Yeo, "Fast extraction of spatially reduced image sequences from MPEG-2 compressed video," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 1100–1114, Oct. 1999.
[8] R. H. J. M. Plompen, B. F. Schuurink, and J. Biemond, "A new motion compensated transform domain coding scheme," in *Proc. ICASSP'85*, vol. 1, 1985, pp. 371–374.
[9] R. H. J. M. Plompen, J. G. P. Groenveld, D. E. Boekee, and F. Booman, "The performance of a hybrid video-conferencing coder using displacement estimation in the transform domain," in *ICASSP'86*, 1986, pp. 4.8.1–4.8.4.
[10] W. Kou and T. Fjallbrant, "A direct computation of DCT coefficients for a signal block taken from two adjacent blocks," *IEEE Trans. Signal Processing*, vol. 39, pp. 1692–1695, July 1991.
[11] Y. Arai, T. Agui, and M. Nakajima, "A fast DCT-SQ scheme for images," *Trans. IEICE*, vol. E71, pp. 1095–1097, Nov. 1988.
[12] J. L. Mitchell, W. B. Pennebaker, C. E. Foog, and D. J. Le Gall, *MPEG Video Compression Standard*. London, U.K.: Chapman and Hall, 1996.
[13] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2*. London, U.K.: Chapman and Hall, 1997.
[14] S. Winograd, "On computing the discrete Fourier transform," *Math. of Comput.*, vol. 23, pp. 175–199, Jan. 1978.
[15] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression Standard*. New York: Van Nostrand, 1993.
[16] J. Song, "Structured composite multimedia documents: Design and presentation in a distributed environment," Ph.D. dissertation, Department of Computer Science, Univ. Maryland at College Park, , College Park, MD, 1997.

Junehwa Song received the B.S. degree from Seoul National University, Seoul, Korea, in 1988, the M.S. degree from the State University of New York in 1990, and the Ph.D degree from the University of Maryland at College Park in 1997, and all in computer science.

He is currently a Research Staff Member at IBM T.J. Watson Research Center, Yorktown Heights, NY, where his focus has been on issues of distributed multimedia systems and internet intermediaries. He was awarded an IBM graduate fellowship during 1995-1997.

Boon-Lock Yeo received the Ph.D. degree in electrical engineering from Princeton University, Princeton, NJ.

He is currently the Senior Vice President of Technology and Engineering at EXP.com, Menlo Park, CA, the online marketplace for expert advice and services, where he leads the development of the site and creation of novel communication technologies, such as transaction-based chat and phone. Prior to this, he led R&D efforts on media technology at the research labs of Intel Corporation, Santa Clara, CA, and IBM, Yorktown Heights, NY. He has published over 30 technical papers and holds five U.S. patents (and over 30 pending applications).

Dr. Yeo previously served as an Associate Editor for the IEEE TRANSACTIONS ON IMAGE PROCESSING and is a recipient of the Best Paper Award.