A FAST DCT DOMAIN INVERSE MOTION COMPENSATION ALGORITHM BASED ON SHARED INFORMATION IN A MACROBLOCK

Junehwa Song
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
junesong@watson.ibm.com

Boon-Lock Yeo
Microcomputer Research Labs, Intel Corporation
Santa Clara, CA 95052
boon-lock.yeo@intel.com

ABSTRACT

The ability to construct intra-coded frame from motion-compensated inter-frame coded directly in the compressed domain is important for efficient video manipulation and composition. In the context of motion-compensated DCT-based coding of video as in MPEG video, this problem of *DCT domain inverse motion compensation* has been studied in [1] and fast algorithm based on factorization of the DCT matrix is proposed in [2]. These schemes, however, treat each 8 × 8 block as a fundamental unit, and do not take into account the fact that in MPEG, a macrolock consists of several such blocks. In this paper, we show how shared information within a macroblock such as motion vector and common blocks can be exploited to yield substantial speedup in computation. Compared to the brute-force approach in [1], 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 [2] and [3].

1. INTRODUCTION

MPEG 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 inter-coded frames into DCT intra-coded frames directly in the compressed domain. By converting video streams from one compressed format to the next, we gain in computational efficiency, eliminate the need for decompression and compression and subsequent degradation in video quality, and allow for higher throughout in dealing with the enormous data rates in a network environment.

This problem of *DCT domain inverse motion compensation*, i.e., the conversion of inter-frame frames into intra-coded frames directly in the DCT domain without the need for full decompression, was studied by Chang and Messerschmitt [1]. The general setup is shown in Figure 1. Here, P_{ref} is the current block of interest, $P_0, \cdots, P_3$ are the four original neighboring blocks from which P_{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_{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

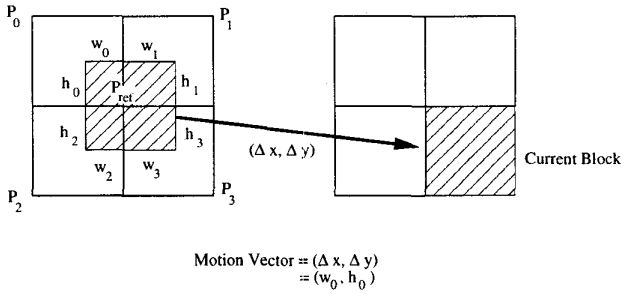

Figure 1. Reference block (P_{ref}), motion vectors and original blocks

can describe in the spatial domain through matrix multiplications:

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

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 1.

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	$\begin{pmatrix} 0 & 0 \\ I_{h_3} & 0 \end{pmatrix}$	$\begin{pmatrix} 0 & I_{w_3} \\ 0 & 0 \end{pmatrix}$

Table 1. Matrices S_{i1} and S_{i2}

While the value of S_{ij} is clear from Table 1 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

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

0-7803-5148-7/98/$10.00©1998 IEEE

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

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

where

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

Using the fact that $DCT(AB) = DCT(A)DCT(B)$, we can write

$$\widehat{P_{ref}} = \sum_{i=0}^{3} \widehat{S_{i1}}\widehat{P_i}\widehat{S_{i2}} \quad (5)$$

In [2], a fast algorithm for computing (5) is proposed. The algorithm is based on the factorization of the DCT matrix T presented in [4]. The factorization is as follows:

$$T = DPB_1B_2EA_1A_2A_3, \quad (6)$$

where D is a diagonal matrix consisting of real entries, P is a permutation matrix, A_i's and B_i's are sparse matrices with entries 1 and -1, and E is also a sparse matrix with real entries.

In this paper, we provide a novel technique to significantly speed up the DCT-domain inverse motion compensation. In Section 2., we present the key ideas of identifying and utilizing shared information including common motion vectors and blocks. Since motion compensation in MPEG stream is done on macroblock basis, we can take a few 8×8 blocks (for luminance component, there are 4 such blocks in each macroblock) as a unit and utilized the proposed idea. This results in about 44 % improvement over the brute force method proposed 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 3 and Section 4 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 works. 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 [5, 6]. We refer the readers to [7] for details on MPEG.

2. MACROBLOCK BASED PREDICTION: UTILIZING SHARED INFORMATION

The DCT domain inverse motion compensation in (5) 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 (5), 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.

Figure 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. Consider the following three observations.

First, although there are totally 16 contributing blocks, there are only 9 different anchor blocks and 5 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 $M_2 = N_2$. 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 sub-block within an anchor block are pairwise 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 sub-block 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 9, resulting in about 44 % of improvement.

2.1. Computation of contribution of two neighboring regions

In Figure 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:

$$\begin{aligned} Q^{M_0} + Q^{M_1} &= S_{01}M_0S_{02} + S_{11}M_1S_{12} \\ &= S_{01}(M_0S_{02} + M_1S_{12}) \\ &= S_{01}(M_0S_{02} + M_1(P^0 - S_{02})) \\ &= S_{01}(M_0 - M_1)S_{02} + S_{01}M_1P^0, \end{aligned} \quad (7)$$

where $P^0 \stackrel{def}{=} S_{02} + S_{12}$. Assume that $\widehat{Q^{M_0}} + \widehat{Q^{M_1}}$ has already been computed using (7) and $\widehat{Q^{N_0}} + \widehat{Q^{N_1}}$ is currently sought for.

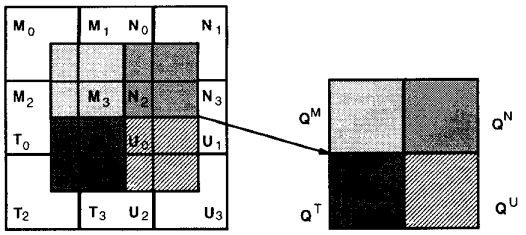

Figure 2. Prediction of a 16 × 16 macroblock

The corresponding equation for the target block Q^N is:

$$Q^{N_0} + Q^{N_1} = S_{01}(N_1 - N_0)S_{12} + S_{01}N_0P^0. \quad (8)$$

Note that the DCT domain value for the second term of (8), i.e., $\widehat{S_{01}N_0P^0}$ has already been computed as $\widehat{S_{01}M_1P^0}$ using (7). Therefore, only $\widehat{S_{01}}(\widehat{N_1} - \widehat{N_0})\widehat{S_{12}}$ needs to be additionally computed to get $\widehat{Q^{N_0}} + \widehat{Q^{N_1}}$, resulting in 3 *TM* operations to compute both $\widehat{Q^{M_0}} + \widehat{Q^{M_1}}$ and $\widehat{Q^{N_0}} + \widehat{Q^{N_1}}$.

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 $\widehat{Q^{T_0}} + \widehat{Q^{T_2}}$,

$$Q^{M_0} + Q^{M_2} = S_{01}(M_0 - M_2)S_{02} + P^1M_2S_{02} \quad (9)$$

$$Q^{T_0} + Q^{T_2} = S_{21}(T_2 - T_0)S_{02} + P^1T_0S_{02}. \quad (10)$$

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

2.2. Prediction of two neighboring blocks

We will now show 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:

$$\begin{aligned} Q^M &= \sum_{i=0}^{3} Q^{M_i} \\ &= S_{01}(M_0 - M_1)S_{02} + S_{01}M_1P^0 + S_{21}(M_2 - M_3)S_{02} + \\ &\quad S_{21}M_3P^0 \\ &= S_{01}(M_0 - M_1 - M_2 + M_3)S_{02} + \\ &\quad S_{01}(M_1 - M_3)P^0 + \\ &\quad P^1(M_2 - M_3)S_{02} + P^1M_3P^0 \end{aligned} \quad (11)$$

Consider the prediction of $\widehat{Q^N}$ assuming that $\widehat{Q^M}$ has already been predicted using (11). The equivalent equation for Q^N is:

$$\begin{aligned} Q^N &= \sum_{i=0}^{3} Q^{N_i} \\ &= S_{01}(N_1 - N_0 - N_3 + N_2)S_{12} + S_{01}(N_0 - N_2)P^0 + \\ &\quad P^1(N_3 - N_2)S_{12} + P^1N_2P^0 \end{aligned} \quad (12)$$

Again, the second and the fourth terms of (12) are the same as those of (11) 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 $\widehat{Q^N}$.

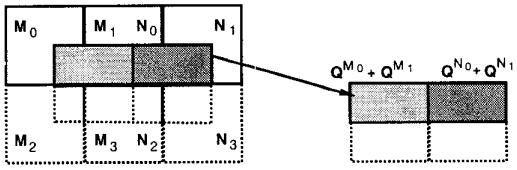

Figure 3. Prediction of two horizontally neighboring regions

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.

2.3. Prediction of the whole macroblock

Now, consider the prediction of the whole macroblock composed of Q^M, Q^N, Q^T, and Q^U. The equivalent equations for Q^T and Q^U are:

$$\begin{aligned} Q^T &= \sum_{i=0}^{3} Q^{T_i} \\ &= S_{21}(T_2 - T_3 - T_0 + T_1)S_{02} + S_{21}(T_3 - T_1)P^0 + \\ &\quad P^1(T_0 - T_1)S_{02} + P^1T_1P^0 \end{aligned} \quad (13)$$

$$\begin{aligned} Q^U &= \sum_{i=0}^{3} Q^{U_i} \\ &= S_{21}(U_3 - U_2 - U_1 + U_0)S_{12} + S_{21}(U_2 - U_0)P^0 + \\ &\quad P^1(U_1 - U_0)S_{12} + P^1U_0P^0 \end{aligned} \quad (14)$$

As previously described, both $\widehat{Q^M}$ and $\widehat{Q^N}$ can be computed by 6 *TM* operations. Now compare (13) and (14) with (11) and (12). The third and the fourth terms of (13) are the same as those of (11) since $T_0 = M_2$ and $T_1 = M_3$. The second and the fourth terms of (14) are the same as those of (13) since $T_1 = U_0$ and $T_3 = U_2$. Finally, the third term of (14) is the same as that of (12) 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.

3. FAST COMPUTATION BASED ON FACTORIZATION OF DCT MATRIX

A fast algorithm for DCT domain inverse motion compensation has been proposed in [2]. The proposed idea of utilizing shared information can also be applied to the algorithm to obtain further speedup.

The speed up 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

$$J_l = U_l(EA_1A_2A_3)^t$$

$$K_l = L_l(EA_1A_2A_3)^t, \quad 1 \le l \le 8,$$

where matrices E, A_1, A_2, A_3 are the components of the factorization in (6), and, $U_h \stackrel{def}{=} S_{11}(h) = S_{21}(h)$ and $L_w \stackrel{def}{=} S_{12}(w) = S_{32}(w)$. 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, $1 \le l \le 8$,

$$\begin{aligned}\widehat{S_{ij}} &= TL_lT^t = TL_l(A_1A_2A_3E)^tB_2^tB_1^tP^tD^t \\ &= TK_lB_2^tB_1^tP^tD^t, \quad \text{or} \qquad (15)\\ \widehat{S_{ij}} &= TL_lT^t = DPB_1B_2EA_1A_2A_3L_lT^t \\ &= DPB_1B_2K_l^tT^t. \qquad (16)\end{aligned}$$

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):

$$\widehat{Q^M} = (\widehat{S_{01}}\widehat{M_0} + \widehat{S_{21}}\widehat{M_2})\widehat{S_{02}} + (\widehat{S_{11}}\widehat{M_1} + \widehat{S_{31}}\widehat{M_3})\widehat{S_{32}} \qquad (17)$$

$$\begin{aligned}&= T[\ (J_hB_2^tB_1^tP^tD\widehat{M_0} + K_{8-h}B_2^tB_1^tP^tD\widehat{M_2})DPB_1B_2J_w^t \\ &+(J_hB_2^tB_1^tP^tD\widehat{M_1} + K_{8-h}B_2^tB_1^tP^tD\widehat{M_3})DPB_1B_2K_{8-w}^t\ ]\ T^t \qquad (18)\end{aligned}$$

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 Figure 2. The equations for the predictions of Q^M and Q^N in spatial domain can be rewritten as follows:

$$\begin{aligned}Q^M &= [S_{01}(M_0 - M_1) + S_{21}(M_2 - M_3)]S_{02} + \\ &\quad [S_{01}M_1 + S_{21}M_3]P^0 \qquad (19)\\ Q^N &= [S_{01}(N_1 - N_0) + S_{21}(N_3 - N_2)]S_{12} + \\ &\quad [S_{01}N_0 + S_{21}N_2]P^0 \qquad (20)\end{aligned}$$

The DCT domain versions of (19) and (20) have similar structures to (17). Therefore, we can apply the same factorization as in (18). The only difference is the permutation matrix P^0 in (19) and (20). This problem is solved as follows:

$$\begin{aligned}TP^0T^t &= DPB_1B_2(EA_1A_2A_3)P^0T^t \\ &= DPB_1B_2(EA_1A_2A_3I)P^0T^t \\ &= DPB_1B_2(EA_1A_2A_3L_8)P^0T^t \\ &= DPB_1B_2K_8^tP^0T^t. \qquad (21)\end{aligned}$$

Similarly,

$$TP^0T^t = TK_8B_2^tB_1^tP^tD \qquad (22)$$

Therefore, by applying the factorizations in (17) and (18), we have

$$\begin{aligned}\widehat{Q^M} = [\ &\{J_hB_2^tB_1^tP^tD(\widehat{M_0} - \widehat{M_1}) \\ &+K_{8-h}B_2^tB_1^tP^tD(\widehat{M_2} - \widehat{M_3})\}DPB_1B_2J_w^t \\ &\overbrace{+\ \{J_hB_2^tB_1^tP^tD\widehat{M_1} + K_{8-h}B_2^tB_1^tP^tD\widehat{M_3}\}DPB_1B_2K_8^tP^0}^{\text{shared}}\ ]\ T^t\end{aligned} \qquad (23)$$

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. We will now 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 PA-RISC processor such as ADD, SHIFT, and SHIFT-ADD. The worst case analysis in the paper can be summarized as follows. In summary, 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.

- Cost of a 2D DCT: 672 arithmetic operations
- Worst-case cost of pre- or post-multiplication by S'_{ij} or S''_{ij}: $39 \times 8 + 32 \times 2 = 376$ arithmetic operations
- 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 (18), 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., $\widehat{Q^{M_0}} - \widehat{Q^{M_1}}$, $\widehat{Q^{M_2}} - \widehat{Q^{M_3}}$, etc, as shown in (23). Now consider the effect of permutation matrix as in (21), (22), and (23). We denote the factorized versions of TP^0 and P^0T^t in (21) and (22) as P'^0 and P''^0, respectively. That is, $P'^0 = DPB_1B_2K_8^tP^0$ $P''^0 = P^0K_8B_2^tB_1^tP^tD$. 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. 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 (23) is 5360 as shown below.

- $\widehat{Q^M}$: five multiplications by S'_{ij} or S''_{ij} + one multiplication by P'^0 or P''^0 + three matrix additions + one 2D DCT: $376 \times 5 + 432 + 64 \times 3 + 672 = 3176$
- $\widehat{Q^N}$: three multiplications by S'_{ij} or S''_{ij} + two matrix additions + one 2D DCT: $3763 + 64 \times 2 + 672 = 1928$
- 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 the (11), (12), (13), and (14) (which are for the predictions of Q^M, Q^N, Q^T, and Q^U in Figure 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,

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

Now for the prediction of $\widehat{Q^M}, \widehat{Q^N}, \widehat{Q^T}$ and $\widehat{Q^U}$, the factorization of (17) and (19) can be applied to the following rearranged

equations and intermediate results for the shared terms can be reused.

$$
\begin{aligned}
Q^M &= [S_{01}Z_0 + P^1Y_1]S_{02} + [S_{01}Y_0 + P^1X_0]P^0 \\
Q^N &= [S_{01}Z_1 + P^1Y_2]S_{12} + \overbrace{[S_{01}Y_0 + P^1X_0]P^0}^{\text{shared}} \\
Q^T &= [S_{21}Z_2 + \overbrace{P^1Y_1}^{\text{shared}}]S_{02} + [S_{21}Y_3 + \overbrace{P^1X_0}^{\text{shared}}]P^0 \\
Q^U &= [S_{21}Z_3 + \overbrace{P^1Y_2}^{\text{shared}}]S_{12} + \overbrace{[S_{21}Y_3 + P^1X_0]P^0}^{\text{shared}}
\end{aligned}
$$

Now consider the extra matrix additions to compute, $Y_i, 0 \leq i \leq 3$ and Z_j, $0 \leq j \leq 3$. Computation of each Y_i takes one matrix addition. Once Y_i's are available, Z_0 and Z_1 can be computed with 5 matrix additions by arranging the computation as follows:

$$
\begin{aligned}
Z_0 &= M_0 - Y_1 - (Y_0 + X_0) \\
Z_1 &= N_1 - Y_2 - (Y_0 + X_0).
\end{aligned}
$$

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

- 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 = $376 \times 3 + (376+56) \times 3 + 64 \times 3 + 672 = 3288$
- 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 = $376 \times 2 + (376+56) + 64 \times 2 + 672 = 1984$
- 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 = $(376 \times 3) + (376+56) + 64 \times 3 + 672 = 2424$
- Cost to compute $\widehat{Q_U}$: two multiplications by S'_{ij} or S''_{ij} + two matrix additions + one 2D-DCT = $376 \times 2 + 64 \times 2 + 672 = 1552$
- 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 = 10144, which amounts to 10144/4 = 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.

4. 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 [4]. 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 (5) is approximated by a finite sum of powers of twos with a maximum distortion of 1/32. The matrix multiplications in (5) 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 *SHIFT*'s.

As before, we can apply our propose technique by utilizing shared information as in (19) and (20). The DCT values of the permutation matrices, $DCT(P^0)$ or $DCT(P^1)$ are also represented as summations of power of 2. 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 arithmetic operations. In the end, the number of arithmetic operations required to compute both blocks in (19) and (20) is 1401. Therefore, for one block, it requires 700.5 operations, which means 13.5% (1 - 700.1/810) improvement. Note that the applying (11), (12), (13), and (14) 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.

5. CONCLUSIONS

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 even 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.

ACKNOWLEDGEMENTS

This work was performed when the second author was with IBM T. J. Watson Research Center. This work is funded in part by an IBM Cooperative Fellowship, and NIST/ATP under Contract Number 70NANB5H1174.

REFERENCES

[1] S. F. Chang and D. G. Messerschmitt, "Manipulation and compositing of MC-DCT compressed video", *IEEE Journal on Selected Areas in Communications: Special Issue on Intelligent Signal Processing*, vol. 13, pp. 1–11, Jan. 1995.

[2] N. Merhav and V. Bhaskaran, "A fast algorithm for DCT domain inverse motion compensation", *in IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*, vol. 4, pp. 2307–2310, May 1996.

[3] P. A. A. Assuncao and M. Ghanbari, "Transcoding of MPEG-2 video in the frequency domain", *in ICASSP 1997*, pp. 2633–2636, 1997.

[4] Y. Arai, T. Agui, and M. Nakajima, "A fast DCT-SQ scheme for images", *The Transactions of the IEICE*, vol. E71, pp. 1095–1097, Nov. 1988.

[5] B. L. Yeo and B. Liu, "Rapid scene analysis on compressed videos", *IEEE Transactions on Circuits and Systems For Video Technology*, vol. 5, pp. 533–544, Dec. 1995.

[6] J. Song and B. L. Yeo, "Spatially Reduced Image Extraction from MPEG-2 Video: Fast Algorithms and Applications", *in Proceedings, SPIE Storage and Retrieval for Still Image and Video Databases VI*, vol. SPIE 3312, pp. 93–107, Jan. 1998.

[7] J. L. Mitchell, W. B. Pennebaker, C. E. Foog, and D. J. Le Gall, *MPEG Video Compression Standard*, Chapman and Hall, 1996.