A Modified IDCT and a New Strategy for Motion Compensation Based on VLIW

Wanli Ouyang, Chuangbai Xiao, Xiaomei Quan, Guang Liu, Wenqi Ju, Dandan Song

College of Computer Science and Technology, Beijing University of Technology, Beijing, China, 100022
Email: wanli210@bjut.emails.edu.cn

Abstract:

In this paper, the matrix decomposition of existing DCT algorithms is presented. Using a Very Long Instruction Word (VLIW) architectural view, existing state of the art algorithms are compared with an algorithm that is tailored for DCT. VLIW operation is further utilized to design a modified IDCT algorithm. Compared with existing algorithms, the new algorithm reduces the number of instruction cycles needed. In addition, we use the register character of VLIW for merging motion compensation (MC) with IDCT (DCT) in the video codec. This reduces the number of cycles required for MC to about 50% of the number required for the unimproved one. The approach is applicable to MPEG1/2/4, H.263 and H.264/AVC.

Key Words: very long instruction word (VLIW); discrete cosine transform (DCT); IDCT; fast algorithm; parallel algorithm; motion compensation; video compression

1. Introduction

Discrete cosine transforms (DCT) and inverse discrete cosine transforms (IDCT) have been the mainstay of transform-based digital signal processing of image data, especially video compression. In the Codec for MPEG1/2/4 and H.264, for inter mode, motion compensation (MC) is often coupled with IDCT (DCT). Since these are computational intensive processes, it is essential to design a fast algorithm to reduce their computing complexity.

Because of the wide application of DCT/IDCT in MPEG1/2/4, H.263 and JPEG, the fast 8*8 2-D DCT becomes a primary concern. Employing the row-column method (RCM) for the fast 8*8 2-D DCT algorithm, the 1-D 8 point algorithm can be applied to the 2-D 8*8 algorithm. As a result, 8 point DCT algorithms are widely studied [1~3, 9~10]. However, as stated in this paper, they are not optimal for Very Long Instruction Word (VLIW) architecture.

Based on VLIW, DCT and IDCT algorithms are given in [4~6]. Meanwhile, IDCT algorithms for video applications, such as [7], are gaining more attention. The algorithm in [4~5] needs fewer clock cycles than those in [3, 6]. However, VLIW operations are not fully utilized for IDCT in [4~5].

In this paper we present DCT algorithms in [3~5] in matrix form, which was not presented in [3~5]. Then, using a VLIW architectural view, the state of the art DCT algorithms are compared with an already existing VLIW tailored algorithm in [4~5]. After that, VLIW operations are further employed for a modified IDCT algorithm. In addition, the register character of VLIW is utilized to combine DCT (IDCT) with MC. This combination results in about 50% reduction in the number of the clock cycles needed for MC.

The rest of the paper is organized as follows. In Section 2, we first analyze existing algorithms with a VLIW view. Then we present a modified IDCT. Section 3 introduces the use of the VLIW's register character to combine IDCT (DCT) and MC. Section 4 gives the experimental results and their analysis. Finally, Section 5 we present the conclusions we have drawn.

2. A modified IDCT algorithm

2.1. Matrix analysis of 8 point DCT algorithms

For $0 \le n \le N-1$, the formula for 1-D DCT is:

$$y_n = \sum_{k=0}^{N-1} C_n x_k \cos\left[\frac{(2k+1)\pi n}{2N}\right],$$

where $C_0 = 1/\sqrt{N}$; $C_n = \sqrt{2/N}$, for $n \neq 0$.

Its inverse (IDCT) is:

$$x_k = \sum_{n=0}^{N-1} C_n y_n \cos\left[\frac{(2k+1)\pi n}{2N}\right].$$

The 8 point DCT matrix form is $Y = C_8 * X$, and IDCT matrix can be presented as $X = C_8^T Y$.

Setting $r(k) = \cos(k\pi/16)$, C_8 is:

$$C_8 = \frac{1}{2}\begin{bmatrix} r(4) & r(4) & r(4) & r(4) & r(4) & r(4) & r(4) & r(4) \\ r(1) & r(3) & r(5) & r(7) & -r(7) & -r(5) & -r(3) & -r(1) \\ r(2) & r(6) & -r(6) & -r(2) & -r(2) & -r(6) & r(6) & r(2) \\ r(3) & -r(7) & -r(1) & -r(5) & r(5) & r(1) & r(7) & -r(3) \\ r(4) & -r(4) & -r(4) & r(4) & r(4) & -r(4) & -r(4) & r(4) \\ r(5) & -r(1) & r(7) & r(3) & -r(3) & -r(7) & r(1) & -r(5) \\ r(6) & -r(2) & r(2) & -r(6) & -r(6) & r(2) & -r(2) & r(6) \\ r(7) & -r(5) & r(3) & -r(1) & r(1) & -r(3) & r(5) & -r(7) \end{bmatrix}$$

Let P(8,1) be a matrix reordering rows of C_8 to: 0, 2, 4, 6, 1, 3, 5, 7; Let P(8,2) be a matrix reordering columns of C_8 to: 0, 1, 2, 3, 7, 6, 5, 4. Except for differences in row permutations, in most fast DCT algorithms (such as [1~5, 9~10]), C_8 is first transformed to:

$$M = P(8,1)C_8P(8,2)R_8$$

$$= \begin{bmatrix} r(4) & r(4) & r(4) & r(4) & & & & \\ r(4) & -r(4) & -r(4) & r(4) & & & & \\ r(2) & r(6) & -r(6) & -r(2) & & & & \\ r(6) & -r(2) & r(2) & -r(6) & & & & \\ & & & & r(1) & r(3) & r(5) & r(7) \\ & & & & r(3) & -r(7) & -r(1) & -r(5) \\ & & & & r(5) & -r(1) & r(7) & r(3) \\ & & & & r(7) & -r(5) & r(3) & -r(1) \end{bmatrix}$$

$$= \begin{bmatrix} M1 & \\ & M2 \end{bmatrix}(1), where R_8 = \begin{bmatrix} I_4 & I_4 \\ I_4 & -I_4 \end{bmatrix},$$

M1 in (1) is used for computing the even transformed coefficients; M2 in (1) is used for odd coefficients. For M1, algorithms in [3~5] all use the method below:

$$\begin{bmatrix} r(4) & r(4) & r(4) & r(4) \\ r(4) & -r(4) & -r(4) & r(4) \\ r(2) & r(6) & -r(6) & -r(2) \\ r(6) & -r(2) & r(2) & -r(6) \end{bmatrix} = r(4) * K4 \begin{bmatrix} 1 & & & 1 \\ & 1 & 1 & \\ & 1 & -1 & \\ 1 & & & -1 \end{bmatrix},$$

$$where \quad K4 = \begin{bmatrix} 1 & 1 & & \\ 1 & -1 & & \\ & & \sqrt{2}r(6) & \sqrt{2}r(2) \\ & & -\sqrt{2}r(2) & \sqrt{2}r(6) \end{bmatrix}.$$

Although K4 is further decomposed in [9], this decomposition seems redundant for 16-bit fixed point arithmetic when SIMD operation IFIR16 (see Fig.4) exists.

In [4~5], M2 is directly used for computation. The standard (generally used) form in [3] decomposes M2 as:

$$M2 = r(4) * \begin{bmatrix} 1 & & & 1 \\ & \sqrt{2} & & \\ & & \sqrt{2} & \\ -1 & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & 1 & \\ & -1 & & 1 \\ 1 & & -1 & \\ & 1 & & 1 \end{bmatrix} \begin{bmatrix} r(3) & & & r(5) \\ & r(1) & r(7) & \\ & -r(7) & r(1) & \\ -r(5) & & & r(3) \end{bmatrix}$$

With a VLIW architectural view, if M2 is directly used for odd coefficients, it will have no nested multiplication, have feasibility for the SIMD operation and have very good parallelism. This is the reason for improvement in [4~5]. Although Loeffler's algorithm in [3] requires fewer VLIW operations, the nested multiplication in M4 requires more clock cycles. As stated in [5], for floating point 8*8 DCT, Loeffler's algorithm requires 270 cycles as compared with 230 cycles for the algorithm in [5]. It seems obvious, therefore, that the more multiplication nested algorithm Lee [1] and Hou [2] proposed would be more inefficient for VLIW architecture. The architecture in [9~10] is too irregular for utilizing the SIMD operation.

2.2. A modified IDCT algorithm

In [4~5], VLIW operations are efficiently utilized for DCT. For IDCT in [4~5], however, VLIW operations can be further applied.

2.2.1. Data arrangement

For most image processing applications, pixels are presented by 8-bit unsigned integers; thus the input and output data are often stored using 16-bit integer to meet the demand of both accuracy and memory conservation. Generally, VLIW DSP is a 32-bit processor, and the input and output data of 8*8 2D IDCT are stored in an array that has 64 data. So, 32 32-bit elements can be used to store 64 16-bit input or output data.

2.2.2 1D IDCT algorithm

For 8 point IDCT:

$(O_0, O_1, O_2, O_3, O_4, O_5, O_6, O_7)^T$

$= C_8^T(X_1, X_2, X_3, X_4, X_5, X_6, X_7)^T$

The matrix form for proposed IDCT algorithm, the same as that in [4~5], is:

$$C_8^T = \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix} \begin{bmatrix} 1 & & & & & & & +1 \\ & 1 & & & & & +1 & \\ & & 1 & & & +1 & & \\ & & & 1 & +1 & & & \\ & & & 1 & -1 & & & \\ & & 1 & & & -1 & & \\ & 1 & & & & & -1 & \\ 1 & & & & & & & -1 \end{bmatrix} \begin{bmatrix} 1 & & & 1 & & & & \\ & 1 & 1 & & & & & \\ & 1 & -1 & & & & & \\ 1 & & & -1 & & & & \\ & & & & & & & 1 \\ & & & & & & 1 & \\ & & & & & 1 & & \\ & & & & 1 & & & \end{bmatrix}$$

$$* \begin{bmatrix} r(4) & r(4) & & & & & & \\ r(4) & -r(4) & & & & & & \\ & & r(6) & -r(2) & & & & \\ & & r(2) & r(6) & & & & \\ & & & & r(1) & r(3) & r(5) & r(7) \\ & & & & r(3) & -r(7) & -r(1) & -r(5) \\ & & & & r(5) & -r(1) & r(7) & r(3) \\ & & & & r(7) & -r(5) & r(3) & -r(1) \end{bmatrix} \begin{bmatrix} 1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

With X_0, X_2, X_3, X_4 stored in the upper 16 bits, Fig 1 gives the data flow diagram in [4~5], Fig.2 describes the modified IDCT algorithm; Fig.3 illustrates Fig 1 and Fig.2; and Fig.4 gives the operation used in Fig.3. C_i is scaled by 2^{15} in Fig.3, which is the same as stated in [5], while C_i in [4] is scaled by $2^{14}\sqrt{2}$. The reason for scaling C_i by 2^{15} is that all signed C_i are less than 1 and some of them are greater than 0.5 (see (1) in 2.1), and they are to be stored using 16 bits. This method of scaling C_i by 2^{15} can also be applied for DCT in [4~5], which improves precision.

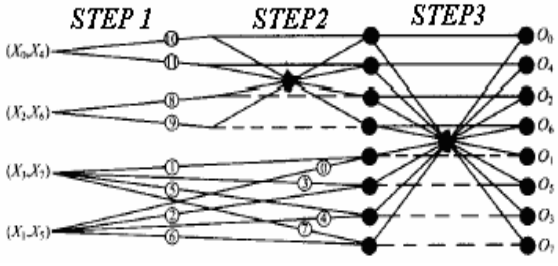


Fig. 1: 1D IDCT algorithm in [4][5]

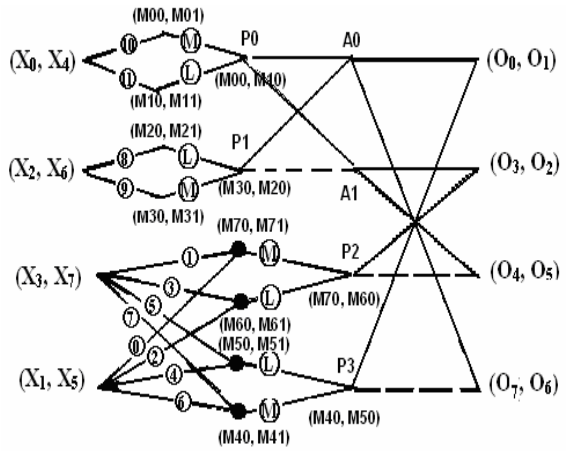
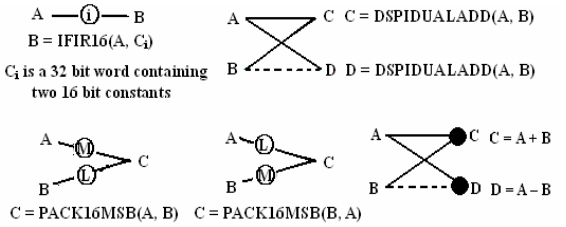


Fig.2: The proposed modified algorithm



Ci	Upper	Lower	Ci	Upper	Lower	Ci	Upper	Lower
0	*r(7)*	*r(3)*	4	*r(3)*	*-r(1)*	8	*r(6)*	*-r(2)*
1	*-r(5)*	*-r(1)*	5	*r(7)*	*-r(5)*	9	*r(2)*	*r(6)*
2	*r(5)*	*r(7)*	6	*r(1)*	*r(5)*	10	*r(4)*	*r(4)*
3	*-r(1)*	*r(3)*	7	*r(3)*	*r(7)*	11	*r(4)*	*-r(4)*

Upper: Upper 16 bits of Ci, Lower: lower 16 bits of Ci
The upper and lower 16 bits value should all be scaled 215 times

Fig.3: Illustration of Fig.1 and Fig.2

C = (C0, C1) A = (A0, A1) B=(B0, B1) A0, B0, C0 : the upper 16-bit element of A, B, C A1, B1, C1 : the lower 16-bit element of A, B, C	
C = DSPIDUALSUB(A, B)	**C = DSPIDUALADD(A, B)**
C0 = A0 – B0 C1 = A1 – B1	C0 = A0 + B0 C1 = A1 + B1
C = PACK16MSB(A,B)	**C = IFIR16(A, B)**
C0 = A0 C1 = B0	C = A0×B0 + A1×B1

Fig.4 VLIW operations used in Fig.3

Tab 1 compares the number of operations needed for the 8 point IDCT algorithm in [4~5] with our modified algorithm. The modified algorithm uses VLIW SIMD operation DSPIDUALADD instead of the 32-bit SISD iadd. By utilizing more VLIW operations, total operations are reduced by 2. The topology structure of the modified algorithm corresponds better to the DCT in [4~5].

Table.1: Number operations for 8 point IDCT

algorithm	iadd &isub	pack1 6msb	ifir	dspidual add/sub	Total
[4][5]	16	0	12	0	28
proposed	4	4	12	6	26

2.2.3 2D implementation of modified IDCT algorithm

The RCM can easily be applicable to 2-D implementation. Data should be packed before horizontal and vertical computation as depicted in Fig.2. If the output is stored in 16 bits, (O_3, O_2) and (O_7, O_6) need to be rotated before storage, and (O_0, O_1) and (O_4, O_5) simply need to be stored. In contrast, the algorithm in [4~5] requires that the eight data of 1D IDCT be packed using 4 operations before storage. Thus, 2 operations are saved for storing every 8 16-bit output data.

2.2.4 Conclusion for modified algorithm

For 8*8 2D-DCT's implementation, we have 16 operations saved before storage as stated in 2.2.3. We also have 2 x 16 = 32 operations saved during computation as expressed in Tab 1. In total our method saved 48 operations. In addition, the modified algorithm does no harm to the parallelism.

3. A strategy for combining motion compensation with IDCT (DCT)

In the generally used video codec, such as inter mode in MPEG2/4 and H.264, IDCT/DCT and motion compensation (MC) are always coupled. For IDCT i.e., part of the input of MC comes from IDCT; another part comes from the previously referenced frame(s). Usually (JM [11] i.e.), IDCT and MC are implemented as below:

MV_Decode:

Decodes the "MV"; uses "MV" to get the "address of the referenced block(s)" in the referenced frame.

IDCT:

1. Inverse transforms the input data, with output data stored in "IDCT output registers".

2. Stores "IDCT output registers" in an "array".

MC:

1. Uses "address of the referenced block" in MV_Decode as an input parameter to load the referenced block data.

2. Uses the "array" in IDCT as an input parameter to load the output from the "array" to registers.

3. Does motion compensation.

From above, the "array" in IDCT and MC is used for passing parameter between IDCT and MC, and the "array" is stored in memory. However, memory access is operation-consuming for VLIW architecture. In this method, redundant operations will appear for storing data in registers into memory and loading data from memory to registers. Moreover, memory access tends to cause more Nops and data cache misses. Cache misses will cause CPUs to stall while waiting for data to be prepared. The same problem exists for

DCT and MC in the motion estimation stage. Obviously, the implementation above is inefficient.

Usually, for VLIW architecture, there are enough registers to be used for passing parameters. In the next section, a strategy utilizing this register character for combining modified IDCT with MC into one function is introduced.

3.1. Combine motion compensation with modified IDCT

Combining IDCT and MC together, the proposed strategy is:

MV_Decode: (Remain unchanged)

Decodes the "MV"; uses "MV" to get the "address of the referenced block" in the referenced frame.

IDCT_and_MC: (IDCT and MC combined)

1. *Inverse transforms the input data and store the output in "IDCT output registers".*
2. *Uses "address of the referenced block" in MV_Decode as an input parameter to obtain the referenced block data.*
3. *Directly use IDCT's output in "IDCT output registers" to do motion compensation.*

From the above, it can be seen that the formerly separate IDCT and MC are combined into one. Registers, instead of arrays, are now used to pass parameters. As a result, the formerly needed memory operations (Step 2. in both IDCT and MC) for passing parameters of IDCT's output now are eliminated. The strategy has no extra requirement on algorithms for IDCT and MC except that they are coupled together, which is common for most applications such as codec in MPEG1/2/4, H.263/H.264 etc.

Although [5] implemented the combination with the referenced frame data stored in 16 bits, it is not applicable for every situation. What is proposed in this section is a strategy that combines MC and IDCT based on VLIW register character. This is a generalized concept that provides the theoretical basis for the implementation reported in [5].

4. Simulation results and analysis

The measure taken to analyze performance is the number of clock (instruction) cycles required to perform certain 8*8 2D IDCT and MC functions on TM1300. Two methods are used:

1. Count the number of cycles needed in .S file compiled by Trimedia SDE2.2;

2. Use the Trimedia performance analysis tool: tmsim and tmprof ([5] can be referenced for details).

4.1. Results using method 1

Table.2 Comparison using method 1
Row: Row transform used modified algorithm
col: Column transform used modified algorithm
M: Motion compensation combined

	cycles	Nops	%(Cycles)	Saved(cycles)
[4][5]	163	52	100	0
Row-col	157	74	96.31	6
Row	157	38	96.31	6
Row-col-M	185	57		

Tab 2 compares the algorithm in [4~5] to various forms of modified algorithm. The 163 cycles for algorithm in [4~5] is in accordance with what has been stated in [5] (160-170 Cycles).

Carefully analyzing the Row-col method, as stated in 2.2.4, in all 48 operations are reduced. In Table.2, 6 cycles (6*5 = 30 operations) are saved. Adding 22 increased Nops, 52 operations are reduced, which is similar to the 48 operation reduction above.

It is interesting that if column transform is kept the same as that in [4~5] ("Row" in Tab 2), the cycles required are the same compared with the implementation in which both row and column transform have been used in the modified algorithm. However, for combining with MC, implementation in which both row and column are used leads to a better result because it has more Nops left for MC to insert.

4.2. Results using method 2

Tab 3 compares the algorithm in Section 2 with that in [4~5]. Tab 4 gives the result of combining the modified IDCT with motion compensation. In Tab 3 and 4, function IDCT and MC (for motion compensation) have been optimized using the available methods provided in [5].

According to Tab 4, implemented separately, IDCT and MC require 1576+913=2489 instruction cycles (ICs) in all. The combined IDCT_and_MC, in contrast, requires 2019 ICs, about 80% of 2489. Considering MC alone, the proposed strategy only needs (2019-1576) = 443 ICs, 48.5% of the 913 ICs (uncombined one). In addition, the cycles caused by data cache misses are greatly reduced from 679+14= 693 to 84 by utilizing registers rather than memory to pass parameters.

Table.3 Results on Section 2

IDCT in [4][5]		Modified IDCT	
TIC	%	TIC	%
1610	10	1576	97.88

Table.4 Results on Section 3
TIC: Total Instruction Cycles
D$ Cycles: Cycles caused by Data Cache Miss

	function	TIC	D$ Cycles
Before combined	IDCT	1576	14
	MC	913	679
After combined	IDCT_and_MC	2019	84

5. Conclusions

Focusing on the RCM 8*8 2D DCT/IDCT, this paper presents DCT algorithms in [3~5] in matrix form, which was not presented in [3~5]. A modified IDCT algorithm to [4~5] is then proposed. As shown by simulation results, the proposed IDCT requires fewer ICs by reducing 48 operations needed for 8*8 2D IDCT while maintaining parallelism. Then the register character of VLIW is utilized to combine IDCT with motion compensation (MC). Employing registers rather than memory to pass parameters, the method results in less than half instruction cycles needed for MC compared with the uncombined one. The method in Section 3 can be broadly used for functions that are coupled together with parameters being passed by array, i.e. DCT and MC.

6. Acknowledgement

The authors would like to thank the anonymous reviewers for their useful comments and Doctor Rhoda Perozzi for English advice.

This work was supported by Scientific Research Common Program of Beijing Municipal Commission of Education (KM200510005012).

7. References

[1] Byeong Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Trans on Acoustics, Speech and Signal Processing*, pp. 1243 – 1245, 1984.

[2] H. S. Hou, "A fast recursive algorithm for computing the discrete cosine transform," *IEEE Trans on Acoustics, Speech and Signal Processing*, pp. 1455-1461, 1987.

[3] Christoph Loeffler, et al, "Practical fast 1-D DCT algorithms with 11 multiplications," *IEEE Trans on Acoustics, Speech, and Signal Processing*, pp.988 - 991, 1989.

[4] Li ji, Li Xue Ming, "New DCT computation algorithm for VLIW architecture," *IEEE 6th International Conference on Signal Processing*, pp.1074-1077, 2002.

[5] Trimedia Technology Inc.Philips TriMedia™ SDE Documentation 2.2, Book 2, Book4 &TriMedia Databooks, 2000.

[6] Sohm O. P., Bull D.R., Canagarajah C. N., "Fast 2D-DCT implementations for VLIW processors," *IEEE 3rd Workshop on Multimedia Signal Processing*, pp, 655-660, 1999.

[7] Murata E., Ikekawa M., Kuroda C., "Fast 2D IDCT implementation with multimedia instructions for a software MPEG2 decoder," *Proceedings of the IEEE Inc on Acoustics, Speech, and Signal Processing*, 3105 - 3108 vol.5, 1998.

[8] Schutten R. J., De Haan G., "Real-time 2-3 pull-down elimination applying motion estimation/compensation in a programmable device," *IEEE Trans on Consumer Electronics*, pp. 930 – 938, 1998.

[9] Feig E., Winograd S., "Fast algorithms for the discrete cosine transform," *IEEE Trans on Signal Processing*, pp, 2174 – 2193, 1992.

[10] Y. Arai, T. Agui, and M. Nakajima, "A fast DCT-SQ scheme for images," *Trans IEICE*, pp, 1095, 1988.

[11] Joint Video Team (JVT) software JM9.3