SOFTWARE JPEG FOR A 32-BIT MCU WITH DUAL ISSUE

Tadashi Sakamoto and Tomohiro Hase
System LSI Development Center, Mitsubishi Electric Corporation
4-1, Mizuhara, Itami, Hyogo 664-8641, Japan

Abstract

This paper describes a new software solution for DCT-based JPEG image compression using a 32-bit MCU with parallel execution (dual issue).

Taking constraints of the MCU hardware resources into consideration, various fast algorithms for DCT are investigated. As a result, it is concluded that the optimal fast algorithm for MCUs capable of parallel execution differs from the optimal algorithm for MCUs which do not support parallel execution.

Comparing the number of JPEG image compression cycles using software on MCUs with and without parallel execution capability, it is concluded that the processing speed of the latter type of MCU is approximately 30% higher than the former type. When this former processing speed is applied to a 640 x 480 (VGA-size) YCbCr image in 4:2:2 format, compression can be performed in about 0.15 seconds (using a 100MHz MCU with parallel execution capability).

1. Introduction

Due to dramatic recent advances in microcontroller unit (MCU) technology, software solutions for image compression complying with the Joint Photographic Expert Group's (JPEG) standard [1] have become feasible [2]. At the same time, digital still cameras using JPEG image compression need to handle ever bigger images. Accordingly, faster JPEG software solutions are still required.

JPEG algorithms comprise two main processes, Huffman coding and DCT (Discrete Cosine Transform) [3]. The main function of Huffman coding is the table reference and conditional branch based on the value referred to, neither of which can be processed at high speed with conventional MCUs. In contrast, the DCT process is a series of straightforward operations with no branches, and a number of fast algorithms have already been developed [4-7]. Thus, great improvements in processing speed are attainable by improving the implementation of the algorithms based on the type of MCU used.

Parallel execution techniques, which are very common in PC and EWS processors, have begun to be applied to embedded MCUs [8]. As one of the techniques to realize faster DCTs, it is worth investigating the adoption of MCUs capable of parallel execution and optimal fast algorithms for DCT to suit the type of MCU.

To implement 8-point DCT using an MCU, it is necessary to determine the optimal solution, taking into consideration the constraints of the MCU hardware resources, such as whether the MCU is equipped with a multiplier and how many registers are available. In other words, no optimal solutions can be determined based only on the criteria. These criteria include the complexity of computation and the numbers of multiplication, addition and subtraction operations, described in papers related to fast algorithms for DCT [5, 6].

This paper focuses on a software solution for JPEG image compression using an embedded MCU with RISC (Reduced Instruction Set Computer) architecture. The authors' attention was directed to DCT as a critical factor in achieving high-speed processing, and the

Contributed Paper
Manuscript received September 4, 1998

processing speed of fast algorithms for DCT was evaluated taking into account the constraints of the MCU hardware resources. Overviews of DCT and the major fast algorithms are described in Chapter 2. Chapter 3 describes an evaluation of the processing speeds of these fast algorithms executed on a conventional MCU not supporting parallel execution. The evaluation discusses the effectiveness of the fast algorithms' technique for replacing multiplication operations with addition operations. In Chapter 4, the processing speed achieved by the dual issue of instructions under the limited conditions in Chapter 3 is evaluated. In Chapter 5, the performance of JPEG compression using an MCU equipped with dual issue capability is evaluated based on the discussions in Chapters 3 and 4.

2. Fast algorithms for DCT

This chapter describes the definition of DCT and the main fast algorithms for DCT used for the evaluation in this paper.

2.1 DCT

In the JPEG standard, two-dimensional DCT is executed for each block of 8 x 8 pixels.

One-dimensional DCT at N points is defined as follows.

$$F(u)=\sqrt{\frac{2}{N}}C(u)\sum_{x=0}^{N-1} f(x)\cos\left\{\frac{\pi u(2x+1)}{2N}\right\} \quad (1)$$

where

f(x): Input at each point
F(u): Transform value at N points

$$c(i)=\begin{cases}\frac{1}{\sqrt{2}} & (i=0)\\ 1 & (i\neq 0)\end{cases}$$

For two-dimensional DCT, the input image is divided into blocks of N x N pixels (8 x 8 pixels for JPEG). The two-dimensional DCT transforms the data of each f (x, y) into the N x N transform value, F (u, v).

Here, N x N two-dimensional DCT is defined as follows.

$$F(u,v)=\frac{2}{N}C(u)C(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left\{\frac{\pi u(2x+1)}{2N}\right\}\cos\left\{\frac{\pi v(2y+1)}{2N}\right\} \quad (2)$$

Two-dimensional DCT can be implemented by a series of one-dimensional DCT processes [3]. A great improvement in computational speed can be obtained by this reduction to a one-dimensional transform. This is proven by the transformation of Equation (2) into Equation (3) as follows.

$$F(u,v)=\sqrt{\frac{2}{N}}C(v)\sum_{y=0}^{N-1}\left[\sqrt{\frac{2}{N}}C(u)\sum_{x=0}^{N-1} f(x,y)\cos\left\{\frac{\pi u(2x+1)}{2N}\right\}\right]\cdot\cos\left\{\frac{\pi v(2y+1)}{2N}\right\} \quad (3)$$

The inner summation of Equation (3) presents N-point one-dimensional DCT shown in Equation (1) and corresponds to the one-dimensional DCT in the X direction, whereas the outer summation of Equation (3) corresponds to the one-dimensional DCT in the Y direction. Therefore, two-dimensional DCT is implemented by means of the following steps.

1) Executing the multiply-add operations, treating eight pixels arranged in a row or column as in single calculation unit, and writing the results to the row or column, accordingly.
2) Executing an operation that repeats the one-dimensional DCT operation eight times in a single direction.
3) Finally, executing the same operation using the obtained results in the other direction.

2.2 Fast algorithms

Among the various fast algorithms for DCT currently proposed, this section describes the four major algorithms used in this evaluation.

The first fast algorithm was proposed by Chen [4]. Chen's algorithm is evaluated in this paper because it has an excellent regular structure and is also suitable for using multiply-add instructions. However, its disadvantage is that it requires as many as 16 multiplication operations.

Hou [5] proposed a recursive algorithm. Hou's algorithm is regular, except for the last stage, and needs 12 multiplications and 29 additions. Although the number of multiplication and addition operations is the same as that of other fast algorithms [9, 10], this algorithm has the advantage of its excellent regularity and the smaller number of constants necessary for the multiplication operations. Thus, this algorithm is selected for the evaluation described in this paper. To evaluate this algorithm, a non-recursive program was created to avoid the overhead of function calls.

The algorithm proposed by Loffler [6] involves only 11 multiplications. The number of addition is 29, which is the same as that of other algorithms, including Hou's.

The algorithm proposed by Arai [7] has the feature of simplifying the DCT processing by multiplying simple scalings with the DCT results. This algorithm requires only five multiplications and 29 additions. Since six results out of all the DCT results are multiplied with the simple scalings, the total number of multiplication operations required is the same as that of Loeffler's algorithm. However, the six multiplications can be omitted by multiplying the simple scalings by the quantization coefficient, utilizing the fact that JPEG executes quantization after DCT. In spite of this advantage, Arai's algorithm fails to avoid irregular data flows, resulting in problems with computational accuracy.

3. Evaluations of fast algorithms implemented on an MCU

3.1 Problems implementing fast algorithms on an MCU

When implementing a fast algorithm on an embedded MCU, the constraints of the MCU's hardware resources must be considered. Therefore, it is necessary to select an optimal fast algorithm to match the MCU's hardware resources.

The following are the MCU hardware resources which are critical for 8-point DCT.

(a) Whether the MCU is equipped with a multiplier
(b) Whether instructions related to multiply-add and accumulator operations are supported
(c) The number of general-purpose registers
(d) The memory access speed

In the case of an MCU without a multiplier, for example, the number of cycles necessary for executing a multiplication operation is at least 10 times the number necessary for an addition operation. Therefore, an algorithm with a reduced number of multiplication operations is advantageous even if the algorithm needs a larger number of addition operations. On the other hand, recent MCUs equipped with multipliers can execute multiplication operations using the same number of cycles required for additions. This means that utilizing the registers effectively by making the most of the smaller number of points of the 8-point DCT is more important than reducing the number of multiplications. That is to say, in order to minimize the use of memory by utilizing the registers effectively, the amount of data to be retained during computing and the number of constants necessary for multiplication are more important than reducing the number of multiplication operations, as far as the processing speed is concerned.

3.2 Hardware resources of the MCU

The evaluation in this paper was performed using a 32-bit MCU with RISC architecture, equipped with the hardware resources listed in Table 1. This is considered to be appropriate since recent RISC type MCUs are designed to support the implementation of simple functions for digital signal processing. The MCU used for this evaluation was capable of executing at least 16-bit x 16-bit multiplication and multiply-add instructions in a single cycle, just as it could execute addition instructions. It was also capable of executing the rounding and data transfer instructions for the accumulator in a single cycle. Although there are 16 registers in the 32-bit width, for the evaluation 15 registers were used for coding programs and one register was assigned as a stack pointer. For the memory access speed, it was assumed that data could be read in a single cycle with no cache misses.

Table 1 MCU hardware resources for evaluation

Multiplier	16-bit x 16-bit or more
Instructions for multiplier-related operations	Instructions for multiply-add and data transfer between accumulator and general purpose registers Execution of multiply-add operation in one cycle
General purpose register	16 registers in the 32-bit width. One register for stack pointer
Cache	Mounted
Memory access speed	Data can be read in one cycle

3.3 Results of fast algorithm evaluations

The four main algorithms mentioned in Section 2.2 were implemented using assembly language and the resulting processing speeds were evaluated, taking into account the constrains of the hardware resources described in Section 3.2. As explained in Section 2.1, the two-dimensional DCT can be divided into a series of one-dimensional DCTs, and the evaluation was based on the 8-point one-dimensional DCT. The numbers of steps each fast algorithm required to execute an 8-point one-dimensional DCT are listed in Table 2.

As seen in Table 2, the algorithm proposed by Chen is effective for DCT when using an MCU equipped with a multiplier and capable of processing multiply-add instructions. Since this algorithm needs only one multiplication operation to per DCT result, no deterioration of computing accuracy occurs. The reasons why Chen's algorithm is more efficient than the other algorithms are:

(a) The four multiply-add operations (making a total of seven addition and multiplication operations) appearing in Chen's algorithm can be executed with only four multiply-add instructions, and thus it appears that no additions are necessary.

(b) In contrast, to compute `a + b` and `a - b` which are often used in algorithms other than Chen's, it is necessary to transfer the value of "a" to another register, and thus three instructions are necessary to execute what appear to be only two operations (i.e., one addition and one subtraction).

Table 2 Number of steps necessary for 8-point one-dimensional DCT

Fast algorithm	Number of steps
Chen	76
Hou	99
Loeffler	117
Arai	78

4. Realizing DCT using parallel execution

This chapter discusses whether the selection of optimal fast algorithms for DCT is affected when parallel execution, a new computing method, is added under the hardware resource constraints described in Chapter 3.

4.1 Definition of parallel execution

The evaluation was performed based on the assumptions that parallel execution would not be possible for all the instruction combinations, but that it would be possible to execute instructions such as multiplication, accumulator-related operations, load,

store, and branch operations only once in a single step (executable on one side only). This took into account the size limitations of the hardware required to realize the embedded MCU at a low price. The conditions applying for parallel execution are as follows.

1) For instructions such as arithmetic operations, logical operations, transfer, shift and null operation, two instructions could be executed in parallel.
2) Instructions such as multiplication, accumulator-related operations, load, store and branch are executed only on one side. These instructions could be executed in parallel only when they occur in combination with other instructions compatible with parallel execution.
3) When two instructions executed in parallel use the same register, the copy of the register value is assigned to both of these instructions in parallel.

4.2 Problems using parallel execution

When implementing the fast algorithms evaluated in Chapter 3 based on the conditions for parallel execution described in Section 4.1, the following two computations are affected.

1) Computing a + b and a - b

When computing a + b and a - b, which are frequently used in fast algorithms, the value of " a " must be transferred once to another register when no parallel execution is used, so three instructions are required, as mentioned in Section 3.3. In contrast, when a + b and a - b are executed in parallel, a copy of the register value is given to both instructions as described in Section 4.1, and a + b and a - b are executed in one step as b + a || a - b (where || indicates the parallel execution of instructions on the left and right). Thus, parallel execution can reduce the number of steps necessary for computing a + b and a - b from three steps to one.

2) Multiply-add computations

In contrast to 1), instructions such as multiplication, multiply-add and accumulator-related operations can be executed only once in one step. For the effective use of these instructions, therefore, instructions such as arithmetic operations and transfers must be executed simultaneously.

4.3 Results of fast algorithm evaluations using parallel execution

The fast algorithms in Section 2.2 were implemented in assembly language in the same manner as described in Section 3.3 and their processing speeds were evaluated, taking into account the influence of parallel execution described in Section 4.2. The numbers of steps each fast algorithm required to perform 8-point one-dimensional DCT are given in Table 3.

As seen in Table 3, the algorithms proposed by Arai and Chen are advantageous for DCT using an MCU with dual issue. The reason why Arai's algorithm shows better performance than Chen's, in contrast to the results given in Section 3.3, include its highly efficient a + b and a - b computations. In addition, Arai's algorithm requires only four constants for multiplication and a small number of registers.

The advantages of Arai's algorithm are quite effective in realizing two-dimensional DCT by using one-dimensional DCT. To implement two-dimensional DCT, as mentioned in Section 2.1, a total of sixteen one-dimensional DCTs are performed in both vertical and horizontal directions (eight each). The features of Arai's algorithm allow the four constants necessary for multiplication to be pre-assigned to the registers and these values shared for the sixteen one-dimensional DCTs. As a result, the number of steps necessary to execute Arai's algorithm in Table 3 could be reduced from 48 steps to 45.

The comparison results of Table 2 and Table 3 are shown in Figure 1. DCT with an MCU which did not support parallel execution but was equipped with a multiplier provided efficient performance using Chen's algorithm.

Once the parallel execution function was added, however, Chen's algorithm showed the lowest ratio of reduction of number of steps. This was due to the fact that Chen's algorithm cannot take full advantage of the effectiveness of parallel execution because of a bottleneck of multiply-add instructions.

A comparison of the results given in Table 2 showed that the DCT speed could be increased by approximately, 40% by the addition of parallel execution capability.

Table 3 Number of steps required for 8-point one-dimensional DCT (using parallel execution)

Fast algorithm	Number of steps
Chen	53
Hou	65
Loeffler	77
Arai	48

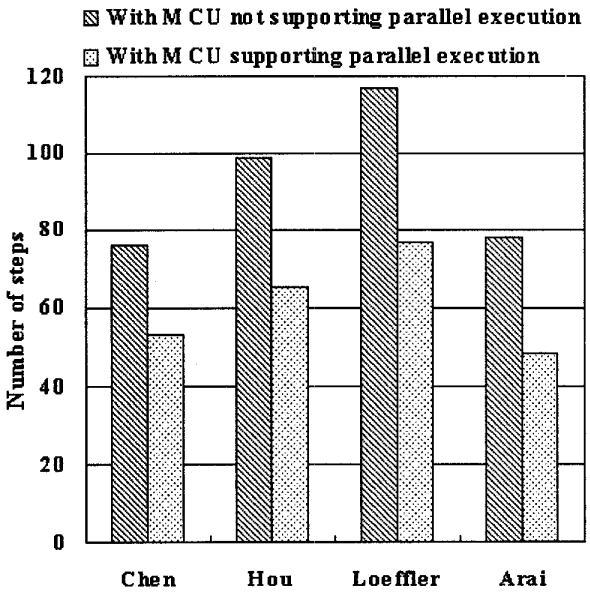
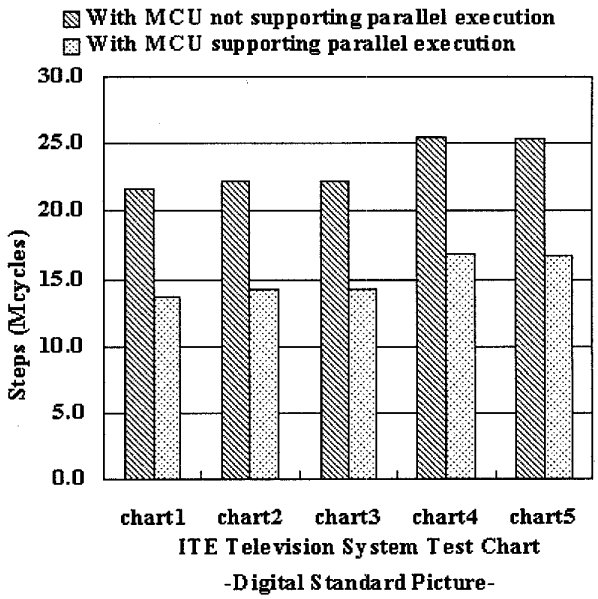


Figure 1 Effects of parallel execution on DCT

5. Evaluation of the JPEG software solution using parallel execution

Figure 2 shows the result of speed performance obtained using MCUs with and without parallel execution capability, when running JPEG programs. Arai's and Chen's algorithms were adopted to implement two-dimensional DCT: Arai's for the former type of MCU (parallel) and Chen's for the latter type (non-parallel). These algorithms were highly evaluated for their respective type of MCU in Chapters 3 and 4. The digital standard picture for television system tests of the Institute of Television Engineers [11] was used for the evaluation. Original RGB data with a pixel size of 755 x 483 was converted into data with a pixel size of 752 x 480 with a sampling ratio of Y:Cb:Cr = 4:2:2, and the number of cycles from YCbCr to JPEG file generation was measured.



Data:
1. ITE Color Matching Chart (a girl with a carnation)
2. ITE Picture (a girl with a hair band)
3. ITE Picture (weather forecast)
4. Villages of Switzerland
5. Tulips

Image size: 752 x 480
Image format: YCbCr
ITE: The Institute of Television Engineers of Japan

Figure 2 Performance comparison of execution time

The processing speeds of JPEG programs on a conventional MCU without parallel execution and on an MCU with parallel execution were compared. As a result, the processing speed of the former type of MCU was approximately 22-26 megacycles, whereas the data could be compressed on the latter type of MCU within

approximately 14-17 megacycles. This means that JPEG program on the latter type of MCU can be realized 35% faster than on the former type.

This processing speed is equivalent to the speed at which a VGA-size image with a sampling ratio of Y:Cb:Cr = 4:2:2 could be compressed within 0.15 seconds (when a 100MHz MCU with dual issue function is used).

6. Conclusion

In order to achieve high-speed JPEG programs, fast algorithms for DCT were evaluated using embedded MCUs with dual issue.

To realize an 8-point DCT which is used in JPEG on an embedded MCU, an optimal solution must be determined, which takes into account the constraints of the MCU's hardware resources, including whether it has a multiplier and the number of registers available. Therefore, in this paper the performance of each algorithm was evaluated quantitatively based on the number of steps actually required given the MCU hardware resource constraints, instead of the numbers of multiplication and addition operations which have conventionally been used as criteria for evaluating the performance of fast algorithms.

To evaluate fast algorithms, four major fast algorithms were implemented in assembly language and the number of steps actually required was measured using MCUs with and without parallel execution capability.

As a result, the fast algorithm by Chen, which is characterized for its capability for effective utilization of multiply-add instructions, was found to be most suitable for 8-point DCT when a conventional MCU with no parallel execution was used. In contrast, when using an MCU with parallel execution capability, the algorithm proposed by Arai was found to be the most advantageous. It was therefore shown that optimal fast algorithms for DCT differ according to the constraints and functions of the MCU used.

By evaluating the speed performance of JPEG programs on MCUs with and without parallel execution capability, it was proved that a JPEG program could be processed 35% faster on the latter type of MCU than on the former type. By applying this performance to compression, VGA-size images with a sampling ratio of Y:Cb:Cr=4:2:2 could be compressed within 0.15 seconds when using a 100MHz MCU.

Acknowledgment

The authors would like to thank Dr. S. Iwade for his support and encouragement of this work.

References

[1] Joint Photographic Experts Group, ISO/IEC JTC 1/SC 29/WG 10, Digital compression and coding of continuous-tone still images - part 1: Requirements and guidelines, 1994.

[2] T. Sakamoto and T. Hase, "JPEG Software Solution for a 32-bit MCU", IEEE Transactions on Consumer Electronics, Vol. 43, No. 3, pp. 410-417, August 1997.

[3] K. R. Rao and P. Yip, Discrete Cosine Transform - Algorithm, Advantages, Applications, Academic Press, San Diego, California, 1990.

[4] W. A. Chen, C. Harrison, and S. C. Fralick, "A Fast Computational Algorithm for the Discrete Cosine Transform", IEEE Transactions on Communications, Vol. COM-25, No. 9, pp. 1004-1011, September 1977.

[5] H. S. Hou, "A Fast Recursive Algorithm for Computing the Discrete Cosine Transform", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-35, No. 10, pp. 1455-1461, October 1987.

[6] C. Loeffler, A. Ligtenberg, and G. S. Moschytz, "Practical Fast 1-D DCT Algorithms with 11

Multiplications", Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP-1989, pp. 988-991, 1989.

[7] Y. Arai, T. Agui, and M. Nakajima, "A Fast DCT-SQ Acheme for Images", Transactions of IEICE, Vol. E 71, No.11, pp. 1095-1097, November 1988.

[8] T. Shimizu, "M32Rx/D – A Single Chip Microcontroller with A High Capacity 4MB Internal DRAM", Hot Chips 10, pp. 37-48, August 1998.

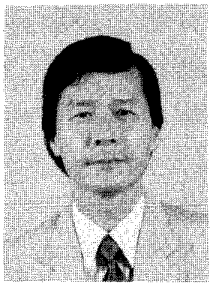
[9] M. Vetterli, H. Nussbaumer, "Simple FFT and DCT Algorithms with Reduced Number of Operations", Signal Processing (North Holland), Vol. 6, No. 4, pp. 267-278, August 1984.

[10] B. Lee, "A New Algorithm to Compute the Discrete Cosine Transform", IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. ASSP-32, No. 6, pp. 1243-1245, December 1984.

[11] The Institute of Television Engineers of Japan, ITE television System Test Charts -Digital Standard Pictures -, May 1985.

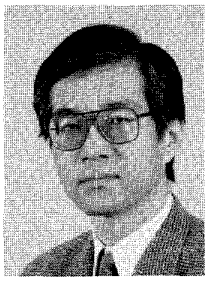
Biographies

Tadashi Sakamoto



Born in 1959 in Kyoto, Japan. Received a B.S. degree in Mathematics from Kyoto University, Japan in 1983 and an M.S. degree in Computer Science from University of Denver, USA in 1990. In 1983, joined the LSI Laboratory of Mitsubishi Electric Corporation, and engaged in research into CAD and the development of software development tools. In 1997, joined the Microprocessor Division of the ULSI Laboratory and is currently engaged in the development of microprocessor application software.

Tomohiro Hase



Born in 1953 in Gifu, Japan. Received a B.S. degree from Fukui University in 1977, an M.S. degree in 1979 and a Ph.D. degree in 1995 both from Shizuoka University, Japan, all in Electronics Engineering. In 1979, joined the Image Electronics Division of Industrial Electronics and Systems Laboratory of Mitsubishi Electric Corporation, and engaged in research into signal processing for video cameras, VCRs, high-definition television and large screen display systems. In 1997, joined the Microprocessor Division of the ULSI Laboratory and is currently engaged in the development of microprocessor application software.