MMX-based DCT and MC Algorithms for Real-Time Pure Software MPEG Decoding

Yi-Shin Tung, Chia-Chiang Ho, and Ja-Ling Wu, Senior Member IEEE

Communication and Multimedia Lab.,
Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan

E-Mail: tung@cmlab.csie.ntu.edu.tw

Abstract

To overcome the difficulties of computation-intensive multimedia applications, the development groups of major CPU manufactories, such as Intel™ and Digital™, have decided to include new instruction sets into their CPU families to increase their multimedia handling ability. The newly introduced instruction set is basically in a Single Instruction Multiple Data (SIMD) Stream operation type. For the practical purpose (e.g. the trade off between the complexity of hardware implementation and the so-obtained performance improvement), they use a reduced SIMD instruction set instead of the full one. Taking Intel as an example, the new instruction set is composed of 57 operations called the MultiMedia eXtension (MMX) instruction set. Nowadays, how to fully utilize the power of the embedded instruction set for providing various multimedia applications becomes an interesting and important issue. In this paper, we demonstrate an efficient realization, based on the new MMX instruction set, of the block Inverse Discrete Cosine Transform (IDCT) and Motion Compensation (MC) which are kernel components of the block-based decoding standards, such as MPEG-1, MPEG-2, H.261 and H.263. The convincing results show that: with the add of proper SIMD instruction set, the pure software solution for complicated multimedia applications (such as real-time MPEG video decoding) becomes feasible.

1. Introduction

Recently, multimedia and virtual reality applications make PC-based products more attractive and more related to human life. However, both of these applications need huge computations. This trend of demand makes CPU manufactories embed new SIMD instruction sets to their CPU families so as to make more complicated applications feasible. These new architectures will benefit to the applications of audio, 2D graphics and video, especially.

Nowadays, the most widespreadly used multimedia application is the video coding. The International Telecommunication Union (ITU-T) has defined several international standards that compromise between perceptual quality and bitrates, such as MPEG-1, MPEG-2, H.261 and H.263. Most of these standards are built under the block-based MC-DCT coding schemes (c.f. Fig. 1). In the decoding process of these schemes, block IDCT and MC are performed heavily. Before MMX technology is released, using original x86 CPU to realize a pure software solution for real-time video decoding is almost impossible. In this paper, based on Intel's new MMX instruction set, we have implemented some modified block decoding algorithms which can be used to evaluate the achievable performance improvement of the existed scale (non-MMX) algorithms by using this new instruction set.

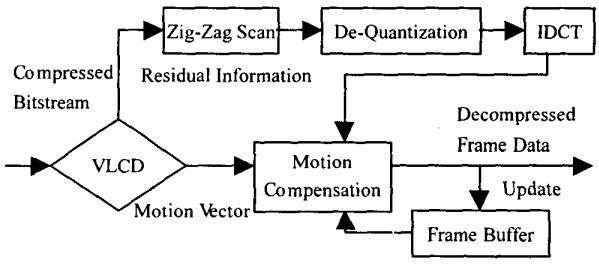

Fig. 1: Block-Based Decoding Diagram. This diagram shows the decoding process of the block-based coding. The compressed bitsream is first decoded by VLCD. The decoded symbols are demultiplexed into the motion vector and the residual information. The latter one passes zig-zag-scan, dequantization and IDCT to restore back to spatial domain information. This information together with the reference obtained, based on the motion vector, from the previous frame buffer composing the new frame. This new frame is treated both as the decompressed frame data and the updating data of the frame buffer.

0-7695-0253-9/99 $10.00 © 1999 IEEE

More importantly, they have been embedded into the block based coding process to show that with the add of MMX technology, pure software solution becomes feasible.

2. The MMX Instruction Set

In order not to change the existed architectures too much, MMX operations make good use of floating point registers instead of introducing new ones. The MMX instructions treat the registered 64 bits either as eight 8-bit data (packed byte), or four 16-bit data (packed word), or two 32-bit data (packed doubleword), or one 64-bit data (quadword). That is, these new instructions could process 64 bits of data at the same time. For example, one new MMX operation could add four 16-bit data to another four in one instruction. The 57 new MMX instructions could be classified into the following categories with respect to their functionality: addition and subtraction, shifting, logical, multiplication, comparison, packing and unpacking, and data transferring. Interesting readers can refer the details of these instructions and the ways for increasing the efficiency of their usage to [1],[2] and [3].

In the essence, MMX is the subset of traditional SIMD instructions. This type of instruction operates on several data streams simultaneously. That means if several data streams need to be undergone by the same operation, SIMD instruction could process them at the same time. It will improve the speed performance as if there were several processing units operated simultaneously. However, unlike general SIMD machines, MMX uses floating-point registers to express integer quantities, and that such quantities come in several 64-bit formats, such as 8 bytes or 2 doublewords. That is to say, the level of parallelism depends on how many elements you can pack into one register. All the processing data must be arranged to have continuous storage addresses, and all the computing elements are represented in fixed-point format. Both of these are limitations, and will introduce some additional overheads when applying MMX. The overheads include data arrangement and precision maintenance. So, how to use MMX with minimum overhead becomes an important issue because this treatment affects the performance of your code heavily.

3. The Block-Based Coding Scheme

Fig. 1 depicts the common decoder diagram of "Block-Based" video decoding standards. In the decoding process of block based coding, the Inverse Quantization (IQ) and the Inverse Discrete Cosine Transform (IDCT) modules follow directly after the Variable Length Code Decoder (VLCD). And finally, the Motion Compensation (MC) is applied.

4. The MMX-Based Block MC-DCT Decoding

In order to take advantage of the MMX technology, parallelizing the traditional algorithms is of interest. In this section, we exploiting the power of MMX technology and applying it to parallelize the IDCT and the MC modules so as to meet the crucial real-time constraint of full motion video decoding. Real-time MPEG-1 and MPEG-2 software decoders are our major targets.

4.1. The Block Inverse Discrete Cosine Transform

In block based decoding process, the 8x8 2D IDCT plays a dominating role. As mentioned above, parallelizing the traditional algorithm is needed for accelerating by MMX. Unfortunately, the 8x8 block IDCT is of less parallelism if the direct 2D approach[4] is applied. Therefore, the separable row-column 1D approach is adopted in our work. In this approach, an 8x8 2D IDCT is realized by calculating 16 8-point 1D IDCT's. Among all 1D IDCT fast algorithms, the AAN algorithm[5] is the most widely used and is also our selection.

Pixel is the smallest unit in the digitized video. With regard to perceptual quantity, 8-bit quantity is necessary to represent the intensities of pixels. However, more precision is needed during the computation. In the 8x8 block DCT/IDCT process, each one of the 64(=2^6) elements does contribution to every transformed coefficients, so at least 14-bit (8+6) precision is needed.

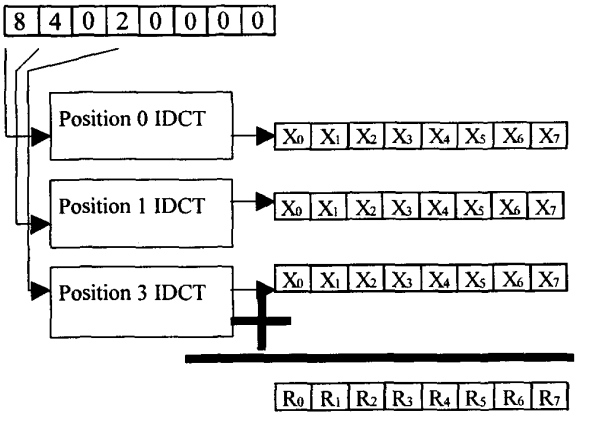

Fig. 2: An illustrative example for the "1D position functions" and the pattern-based IDCT. A 1D IDCT vector (8,4,0,2,0,0,0,0) is transformed to their corresponding vector (R_0, R_1, R_2, R_3, R_4, R_5, R_6, R_7). In the pattern based approach, the input vector is decomposed to V_0 (8,0,0,0,0,0,0,0), V_1 (0,4,0,0,0,0,0,0) and V_3 (0,0,0,2,0,0,0,0). Each vector, according to the position of its non-zero value, is calculated by the proposed Position Function. All of the transformed vectors are then added to produce the final result.

Existed DCT/IDCT implementations usually use 32-bit precision because scalar registers are of 32 bit long in traditional Intel architecture. However, to get more performance gain from using MMX technology, shorter bit quantity is preferred (because one floating point register could contain more individual processing data). Facing this dilemma, we decided to use 16-bit precision for all transformed (both DC and AC) coefficients, intermediate data and output elements, as a trade-off between the calculation error and the execution speed.

In our implementation, we use 64-bit (quad-word) MMX registers to perform four 8-point 1-D IDCT's at the same time such that the cost of computing four IDCT's is the same as that of one in traditional approaches. To maintain maximum precision of data but without overflow, the algorithm needs to control the precision point in dynamic sense, during each step of the IDCT butterfly. The overheads of such implementation include:

(i) Data movements

We need to transpose data after completing the column 1-D IDCT's.

The output data must be transposed again after completing the row 1D IDCT's.

(ii)Bit Shift operations

In order to maintain appropriate dynamic range (precision), extra shift operations are needed during each step of the IDCT butterfly.

The first generation of evolution is a general-purpose IDCT kernel which can be substituted into any program in which the 2D IDCT is performed. We name this first version "General Purpose MMX IDCT". In this basic functional module, the decoding steps of IDCT are as follows.

General Purpose MMX IDCT

1: pre-multiplication of IDCT
2: 1D column IDCT's
3: Transpose the resultant matrix in Step 2
4: 1D Row IDCT's
5: Transpose the resultant matrix again

In this implementation, there is an obvious overhead, transposing the matrix twice, which make it not so efficient. We will make some modifications to reduce this overhead.

In the existed standards, the IDCT module is always following the VLCD module. In the process of VLCD, we can get transformed coefficients one by one. At that time, we can transpose the 8x8 block immediately, instead of transposing it after the step of pre-multiplication. As a result, the input to our IDCT module is a data array lined up with 64 transposed coefficients, where each element is of 16-bit precision. The corresponding output has the same precision, but without the transposition. Combining the processes of VLCD with IQ, the steps of modified decoding process are as follows and we called this algorithm "Transposed MMX IDCT."

Transposed MMX IDCT

1. VLCD (zigzag scan) puts IDCT domain elements in transposed order
2. Inverse quantization (quant table and quant scale)
3. Pre-multiplication (transposed)
4. 1D row IDCT's
5. Transpose the resultant matrix obtained in Step 4
6. 1D column IDCT's

In our implementation, we reserve a local space to store the output of the VLCD and then feed the block of output to the IDCT module. The output of the IDCT is then combined with the delta data (i.e. the residue data reconstructed by motion compensation) to reconstruct the original image. At this stage, we have combined VLCD, IQ and IDCT together. This will benefit global optimization further.

Intel has suggested a similar flowchart for performing 2D IDCT in their web site [6]; moreover, they also proposed a 3D method for eliminating the corresponding transposition. In theory, it works but it is impractical in reality. If four 2D IDCT's are merged together as described in [6], memory bitstream-dependent branch points will be generated. That is, this merging process introduces a lot of branch instructions which in turn introduce some difficulties for deep pipeline programming. Furthermore, in the prescribed 3D approach, the merged four blocks must be treated in the same way, this will inhibit the applicability of some block-dependent techniques, such as the 'Pattern Based MMX IDCT', presented later in this section. Therefore, we try to seek for some other possible solutions.

In the rest of this subsection, three approaches are presented to further improve the speedup performance of the IQ and IDCT modules. Notice that all these approaches can be realized by MMX technology very efficiently.

4.1.1. Advanced Multiplication

In the encoding process, each element of the DCT coefficients will be quantized to a value with smaller range by the product of quant-scale and quant-table; therefore, in the decoding process this operation must be reversed to reconstruct the original data. That is, in VLCD each IDCT domain element will be multiplied by the product of quant-scale and quant-table. In this way, we can pre-compute and store this multiplication result. Owing to the infrequent change of the quant-table, we can build a pre-multiplication table and use quant-scale as the index to retrieve the stored pre-multiplication value. This process saves one multiplication when decoding each IDCT element. We call this mechanism "Advanced Multiplication".

4.1.2. Pattern Based MMX IDCT

DCT-based coding provides compression ability because most of the data energy concentrates on the upper left (i.e. low frequency) corner, and therefore, most of the other elements are composed of zeros. This means that in the VLCD process, only approximately one-quarter among the 64 coefficients are not zero. On the other hand, DCT is a linear operator; therefore, according to the individual input pattern, we can do some input-pruning to reduce the required operations. This method is called the "Pattern-Based Approach". We modify our MMX IDCT so that the Pattern-Based approach is included in the first step of performing the row 1D-DCT's. In the IDCT butterfly, each singular input position (i.e. assuming all other input positions have inputs zero) processes its corresponding pruning butterfly. Using different pruned butterflies which are input pattern dependent, we can define a set of functions which record the relationships between the positions of the non-zero input and the corresponding contributions to all output positions. We called them the "1D position functions". After IDCT's pre-multiplication step, we feed the multiplied values to their corresponding position functions to calculate the corresponding output values and send them to all 8 transform domain coefficients. Fig. 2 illustrates one 1D case as an example. The new decoding steps are as follows.

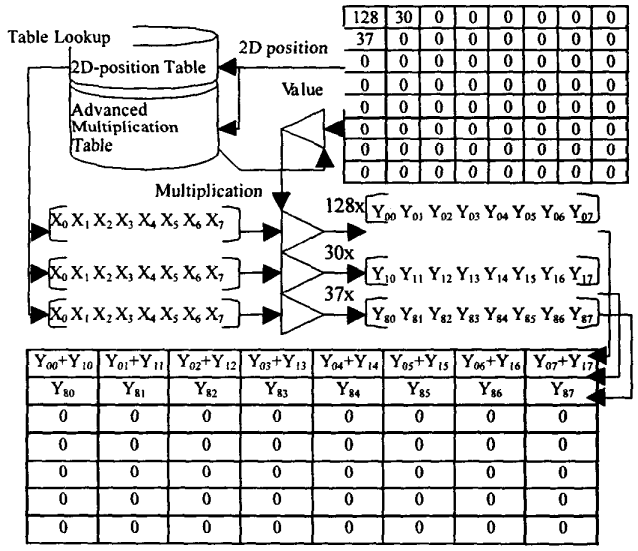

Fig. 3: An illustrative example for the "2D Position Table". Assuming the data block has only three nonzero terms. Look up the table to retrieve the corresponding unit vector based on their positions. And then multiplying table lookup outputs by the corresponding dequantized values. All of these vectors are then summed up to obtain the final result. The above process includes the row IDCT and the pre-multiplication operation of the column IDCT. This is why we called it "2D Position Table".

Pattern Based MMX IDCT

1: VLCD (zigzag scan) puts IDCT domain element in **nontransposed** order
2: ***Advance Multiplication***
3: Pre-multiplication
4: ***1D position i function*** (1D row IDCT)
5: 1D column IDCT's

In the aforementioned Pattern-Based implementation, we found that the coefficients at the same position while with different values contribute to all 8 transformed elements in a linear way. So, one can use Table-Lookup's to form the unit vector, multiplying it with the actual coefficient value to produce the 1D-IDCT outputs. Because the MMX instruction can multiply four 16-bit data in one instruction, this approach could speed up the first step of the row IDCT largely. Actually, we pre-compute the unit vectors of every 2D singular input position and compose them to build a 64x8 vector table according to the 2D IDCT equation. The table is called "2D Position Table", in our implementation, which records the relationship between the unit vector for each 2D singular input position and its corresponding contributions to all output positions of the 1D row IDCT. Moreover, the second pass, (i.e. the column-IDCT's) can be implemented using the same technique. Fig. 3 illustrates one 2D IDCT example by this way.

Table Lookup MMX IDCT

1: VLCD (zigzag scan) puts IDCT domain element in **nontransposed** order
2: ***Advance Multiplication***
3: Pre-multiplication
4: ***2D position j Table Lookup*** (1D row IDCT)
5: 1D column IDCT's

4.2. Motion Compensation

In this subsection, we will focus on how to implement MMX-based fast MC algorithm for decoding MPEG encoded bitstreams.

In view of implementation, motion compensation includes three major functions: pixel averaging, data movement, and delta data adding.

4.2.1. Pixel Averaging:

Pixel values are typically stored as unsigned 8-bit quantities. If pixel averaging is needed, one might question that 8-bit space is not sufficient. However, there is a shortcut to do pixel averaging with some unnoticeable precision loss. For constructing an averaged pixel, related pixel values are accumulated in 8-bit quantities. But, each value can be divided by two (or four) before accumulation, by using shift instructions. So, when averaging two pixels, the least significant bit of each pixel is lost; similarly, when

Table 1: The simulation results of embedding our MMX-based IDCT and MC kernels into the original Non-MMX based MPEG-1 decoder. The values of this table are evaluated on Pentium 200MHz CPU, with MMX technology.

		Sequence 1:			Sequence 2:		
		F/ms	Avg (msec)	Speed up(%)	F/ms	Avg (msec)	Speed up(%)
Original	I	2650/113	23.45		2610/114	22.89	
	P	7000/377	18.57		6520/376	17.34	
	B	15130/953	15.88		15210/958	15.88	
	T	24780/1443	17.17		24340/1448	16.81	
IDCT enhanced	I	2170/114	19.04	18.8	2270/114	19.91	13
	P	6690/379	17.65	5	5610/378	14.84	14.4
	B	13980/956	14.62	7.9	13610/965	13.64	14.1
	T	22840/1449	15.76	8.20	21490/1457	14.75	12.3
MC enhanced	I	2310/114	20.26	13.6	2220/114	19.47	15
	P	7420/379	19.58	-5.50	6720/375	17.92	-3.3
	B	14010/957	14.64	7.8	14180/956	14.83	6.6
	T	23740/1450	16.37	4.7	23120/1445	16.00	4.8
IDCT & MC enhanced	I	1910/114	16.75	28.6	2130/114	18.68	18.40
	P	6080/379	16.04	13.6	5670/375	15.12	12.8
	B	11980/957	12.52	21.2	12300/955	12.88	18.90
	T	19970/1450	13.77	19.8	20100/1444	13.92	17.2

averaging four pixels, the least two significant bits of each pixel are lost. After accumulating, a compensation value (that is 1) can be added back to minimize the overall accuracy loss. By using MMX 64-bit registers, up-to eight pixels can be handled at the same time.

In actual implementation, two code copies for constructing eight averaging pixels are interleaved to increase the percentage of instruction pairing (which is an important technique for Pentium instruction optimization).

4.2.2. Data Movement:

Data movement also benefits from MMX technology. MMX registers are alias of floating-point registers, so, if MMX instructions are used for data movement, scalar registers can be relaxed for other usage (such as memory addressing, loop counter, etc.). Furthermore, registers used by MMX technology are 64-bit long, and one can move eight 8-bit data at the same time. Loop-unrolling technique[3] is then of more practical value in this situation.

4.2.3. Delta Data Adding:

There are two problems for delta data adding. The first problem is type mismatching. Pixel values in reconstructed reference MacroBlock(MB) are stored as 8-bit quantities, but the delta data calculated by IDCT are stored as 16-bit quantities. The second problem is the so-called saturation problem. After adding, the resulting pixel values may be out of the range to be represented as 8-bit quantities. So a saturation table is needed to map out-of-range values into valid values (say, between 0 and 255).

With MMX technology, we can handle the first problem more efficiently by using MMX's packing and unpacking instructions[2,3]. That is, we can unpack 8-bit quantities to 16-bit quantities, adding them with delta data, and then packing the result back into 8-bit quantities. Since MMX instructions are SIMD in nature, up-to four pixels can be processed at the same time instead of one. And MMX technology also provides instructions to do arithmetic operations with saturation. So, the second problem does not exist any more.

Since delta data adding is significantly simplified by using MMX technology, loop unrolling is practical and is adopted in each routine so that further optimization (instruction pairing [2,3]) can be achieved.

5. Performance Evaluation

We have applied the prescribed MMX-based IDCT and MC algorithms to MPEG-1 and MPEG-2 decoding processes. These two standards are the most popular and widely used block-based codecs in video compression. Along with the increase in CPU power, these decoders will be the primary equipment for future PC.

5.1. The MPEG-1 Video Decoder

For comparison purposes, we adopt the latest commercial available MPEG-1 related product: Cyberlink VCDPlayer™ as our non-MMX counterpart. In this product, the pattern-based IDCT and the fast MC have been applied. Replace the IDCT and MC modules of it by our General Purpose MMX IDCT and the MMX-based MC, respectively. And then measuring the execution time of the Non-MMX and the MMX decoders, in which the input bitstreams are taken from commercial VCD discs. The speedup of the IDCT module is about 8% to 15% and that of the MC's is about 5% to 8%. Some simulation

Table 2: The simulation results of embedding our IDCT kernels into the original Non-MMX MPEG-2 decoder. We skip the Pattern Based MMX IDCT, because we are convinced that the Table-lookup MMX IDCT is faster. The values in this table are evaluated on Pentium 200 MHz CPU, with MMX technology. The last sequence (i.e., Susi) is tested under the assumption that the entire MPEG bitstream was cached into the memory.

Bitstream / Codec versions	Pana	Sea	Sub	Heli	Susi
NO IDCT , IQ	19.17	19.07	18.22	18.22	21.05
General Purpose MMX IDCT	N/A	N/A	N/A	N/A	N/A
Transposed MMX IDCT	15.67	15.88	15.32	15.23	17.87
ADVM,MMX IDCT	15.76	15.95	15.46	15.22	17.90
Pattern Based MMX IDCT	N/A	N/A	N/A	N/A	N/A
Table-lookup MMX IDCT	16.69	16.76	16.15	16.09	19.24

Table 3: This table represents the frame rates of decoding 8 Mbps MPEG-2 encoded files on Pentium 200MHz CPU with MMX. The speedup is about 40 to 50% between the two different versions. Notice that different sequences result in different performance gains and "MMX implementation" means Table-Lookup MMX IDCT and MMX MC modules are included.

Sequence	Sea	Susie
Non-MMX implementation	14 fps	13.7 fps
MMX implementation	21 fps	19.2 fps

Table 4: The relative and absolute decoding performance of MMX-based MPEG-2 decoder

(A) The values shown in this table represent the decoder performance for decoding the 8Mbits MPEG2 encoded video sequences on Pentium 200MHz CPU with MMX.

	Scale ver. (sec)	MMX ver. (sec)	Speedup
IDCT	40.66	9.29	4.37
MC	98.34	44.69	2.2
Total time	215.61	123.83	1.74

(B) The values shown in this table represent the decoder performance for decoding the 8Mbits MPEG2 encoded video sequences on Pentium II 266MHz CPU with MMX.

	Scale ver. (sec)	MMX ver. (sec)	Speedup
IDCT	34.05	10.47	3.25
MC	64.56	42.87	1.51
Total time	153.40	111.62	1.37

results are shown in Table 1. Combined with MMX implementations of IDCT and MC, the total speedup of MPEG-1 video decoder is about 15% to 25%.

5.2. The MPEG-2 Video Decoder

Similarly, we use Cyberlink DVDPlayer™ as our non-MMX MPEG-2 decoder counterpart. As in the MPEG-1 case, embedding all of our MMX-based IDCT and MC modules into Cyberlink DVDPlayer to compose the MMX MPEG-2 decoder. Table 2 shows the decoding frame-rates of the refined versions in which all the aforementioned MMX IDCT modules are included. Note that, for comparison purpose, the frame-rates of the original MPEG-2 decoder without IDCT and IQ modules are included in the first row. Table 3 briefly reveals the performance improvement when both MC and IDCT are implemented by using MMX technology. It is clear that different video sequences may have different performance gains. Table 4 shows the absolute and relative decoding performance of the MMX MPEG-2 decoding on two different Pentium CPU's. Briefly, the total speedup of MPEG-2 video decoder is about 40 to 50%, if MMX technology is applied.

6. Conclusion

Intel's MMX technology has been criticized; critics claimed that 16 bits data precision could not fit the necessity of most 2D, 3D graphics and high dimensional audio data. Indeed, MMX is not always usable and its tolerance for computation error depends on the precision requirement of the application. But the programmers can compromise the precision according to the requirements. As what we have done in our implementation, 2D video could get a balance between visual quality and speed-up when we decided to use 16 bits quantity. Actually, not only in Intel processors but also in Alpha processors, a similar SIMD instruction set called "MVI"[7] has been built-in. It will achieve the same kind of speed-up for block-based decoding applications. The inclusion of the new SIMD instruction sets is an important evolution in PC's CPU. Intel has announced that they will include new MMX2 instruction sets based on the floating type operation (this will help to solve the precision problem) to their next generation CPU's and provide more powerful computation ability for more and more multimedia applications. The work done in this paper provides some guidelines on how to speedup video decoding by using MMX technology. Moreover, the so-obtained results also show that, with the advance of CPU power, real time pure software solution for complicated multimedia application, such as MPEG decoding, becomes feasible.

By the reduced SIMD instruction sets, most of the video processing speed will be well improved. We will use these technologies to complete even more computation-intensive tasks, such as video encoder, video editor in the near future. These works may lead digital world more valuable.

References:

[1] Alex Peleg, Sam Wilkie, and Uri Weiser, "Intel MMX for Multimedia PCs", Communications of the ACM, Jan. 1997/Vol 40, No 1.

[2] Rohan Coelho and Maher Hawash, DirectX, RDX, RSX, and MMX™ Technology, A Jumpstart Guide to High Performance APIs, Addison Wesley Developers Press, ISBN 0-201-30944-0.

[3] D. Bistry, et. al., The Complete Guide to MMX™ Technology, Intel Corporation, McGraw-Hill, Inc., 1997

[4] Yung-Pin Lee, Thou-Ho Chen, Liang-Gee Chen, Mei-Juan Chen and Chung-Wei Ku, "A Cost-Effective Architecture for 8x8 Two-Dimensional DCT/IDCT Using Direct Method", IEEE Transaction on Circuits and Systems for Video Technology, Vol. 7, No. 3, June 1997.

[5] Yukihiro ARAI, Takeshi AGUI and Masayuki NAKAJIMA, A Fast DCT-SQ Scheme for Images," The Transactions of The IEICE, Vol. E 71, No. 11, pp. 1095-1097.

[6] Intel Technology Application Note for MMX (http://developer.intel.com/drg/mmx/appnotes/ap528.htm)

[7] Digital Semiconductor Alpha 21164PC Microprocessor Product Brief, Hardware Manual and Data Sheet, available from web site (http://www.digital.com/semiconductor/).