

Approaches to Implementation of Image Compression Algorithms on NeuroMatrix® Architecture

Sergey Mushkaev^a, Sergey Landyshev^a

^aResearch Center "Module", 3 Eight March 4th Street, Box 166, Moscow, 125190, Russia,
tel. +7-095-152-9698, fax. +7-095-152-4661,
e-mail: mushkaev@module.ru, <http://www.module.ru/>

Abstract

Possibilities of NM6403 processor in static image compression are discussed in this article. DCT (Discrete Cosine Transform) algorithm and methods of accuracy enhancement are presented. Short description of JPEG coder implementation on NM6403 processor and some aspects of the vector co-processor usage in coding tasks are also given.

Introduction

The question of the most effective implementation always appear in the tasks with high enough computational complexity. For different architectures optimal methods are unique, but in general the principle consists in minimization of arithmetic operations. In case of NM6403 processor it is partly fair. Here the main idea is to lead the algorithm to the optimal set of streaming operations over data blocks, where each block can be entirely processed at one processor step. Basically these are logical and arithmetic operations with small matrices and vectors. For example, that principle allows to effectively organize parallel calculations on NM6403 processor of FFT (Fast Fourier Transform) with radix 16 and 32 [1].

Forward 2D DCT computation algorithm on NM6403 processor

The direct DCT 8x8 calculus may be considered as an extreme case, when additional minimization is not required. Let us define these streaming operations starting from the forward DCT 8x8 formula.

The DCT 8x8 is defined as follows:

$$Y(u, v) = \frac{1}{4} w(u)w(v) \sum_{x=0}^7 \sum_{y=0}^7 f(x, y) \cos\left[\frac{\pi(2x+1)u}{16}\right] \cos\left[\frac{\pi(2y+1)v}{16}\right], \text{ where } u, v, x, y = 0, 1, 2, \dots, 7$$

x, y - spatial coordinates in the 8x8 block ,

u, v - coordinates in the frequency domain,

$w(u) = 1$ if $u = 0$; 1- otherwise,

$w(v) = 1/\sqrt{2}$ if $v = 0$; 1- otherwise.

In the equivalent matrix form of DCT this formula looks like:

$$[Y] = [C] \times [X] \times [C^T]$$

[X]- initial 8x8 matrix

[Y]- resulting 8x8 matrix of discrete cosine transform

[C], [C^T]- 8x8 matrices of cosine coefficients C_{ij}.

Thus, DCT computation consists of multiplying of the initial matrix by two matrices [C] and [C^T]. It can be carried out in two ways:

1. $[Y] = ([C] * [X]) * [C^T]$

2. $[Y] = [C] * ([X] * [C^T])$

Since all computations are based on the fixed-point arithmetic, these ways differ a lot from each other while vector co-processor is implemented. The best choice depends on digit capacity of

$[X]$, $[C]$ and $[C^T]$ matrices. The structure of vector co-processor (VCP) multiply unit is that the product of $[C]$ and $[X]$ is accumulated with the same word length as that of $[X]$ multiplier. Therefore, the first variant imposes tighter constraints on word length of input data. The second one does not depend on digit capacity of input data and therefore it is more universal and flexible in choice of cosine coefficients word length of $[C]$ and $[C^T]$ matrices. Due to this reason the last variant will be considered below.

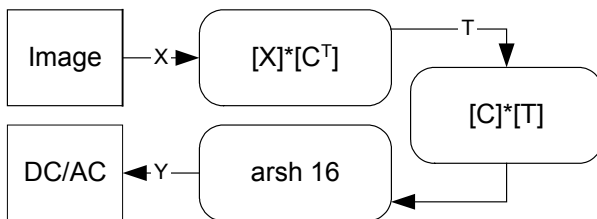
The whole process is divided into two stages:

1. Computing intermediate matrix $[T]$: $[T]=[X]*[C^T]$
2. Computing result matrix $[Y]$: $[Y]=[C]*[T]$

Let us illustrate the most simple and efficient way of DCT algorithm implementation for 8-bit input signed data. It is advantageous to set word length of $[C^T]$ and $[Y]$ matrices to 32 bit, it will prevent product overflow and simplify further data processing after DCT. The transform accuracy depends on the number of significant bits in the coefficients of $[C]$ and $[C^T]$ matrices. As will be shown below this usage of 7 least significant bits and the sign provides for acceptable precision. Therefore it is enough to use 8-bit word length of matrix $[C]$.

Note, that all elements of matrices are considered as signed, therefore input values X_{ij} have the range between -128 and 127, but it does not exclude the processing of usual 8-bit images.

The cosine coefficients $C_{fp}(i,j)$ given in the floating-point format (index $_{fp}$) have the range between -0.491 and +0.491. They are converted to the fixed-point format (index $_{fix}$) according to $C_{fix}(i,j)=round(C_{fp}(i,j)*2^8)$ formula, where 2^8 – is the scale factor, $round()$ – is the round-off function returning the nearest integer. Using the same scale factor - 2^8 for $[C]$ and $[C^T]$ matrices allows to accumulate the result without overflow in 32-bit data of $[T]$ and $[Y]$ matrices and to scale down the products only once using 16 bit right shift (arsh 16) of $[Y]$ matrix. Thus the whole process of the DCT computing consists of the two stages of multiplying and one scaling down:



The matrix multiplying process and the internal fragmentation of packed data are shown in details in Fig. 1 (the bold lines mark the boundary of 64-bit words, and the thin lines point to the boundary of components packed into 64-bit words)

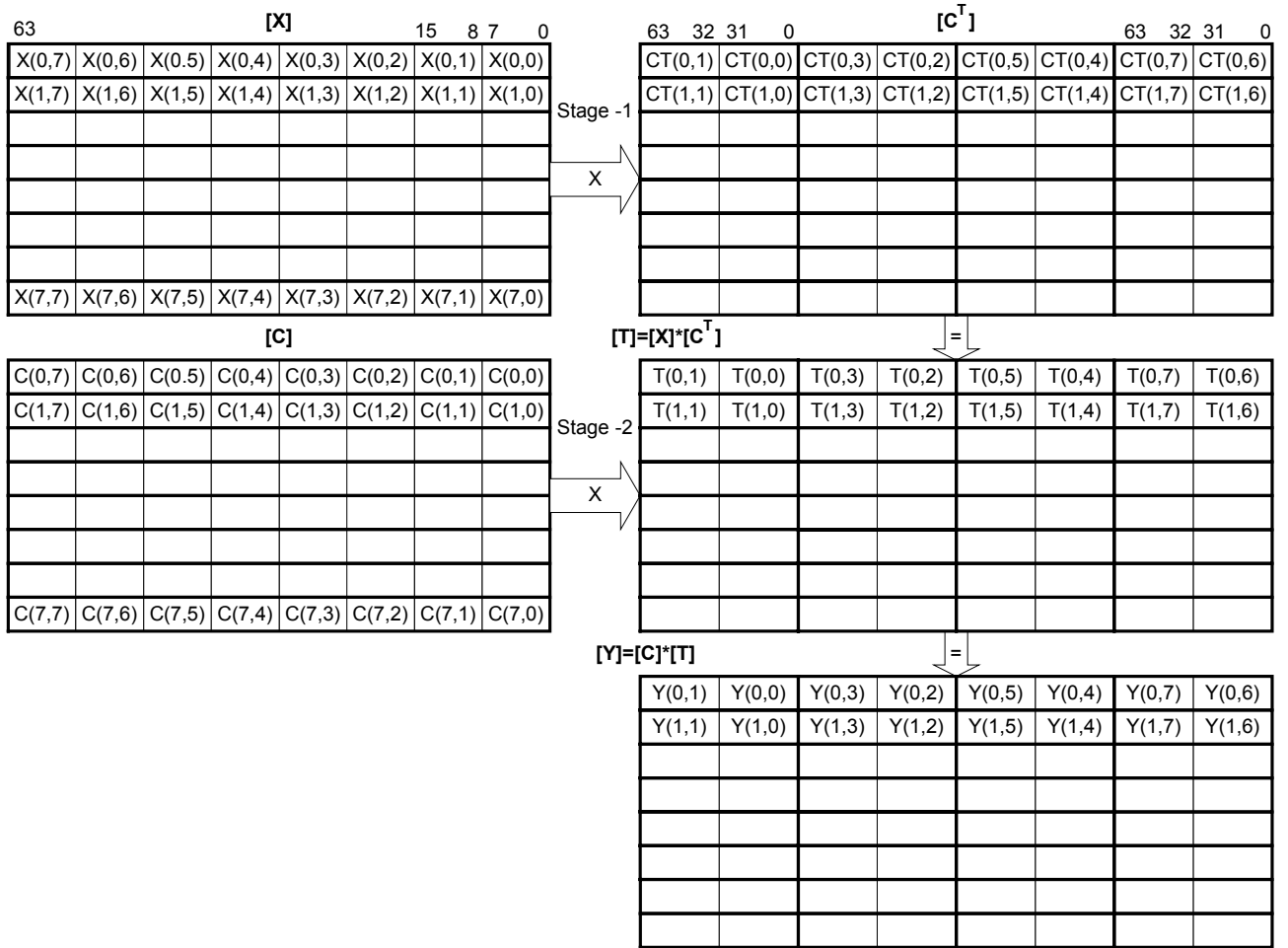


Fig. 1 The matrix multiplying scheme of the DCT 8x8 computation and the internal structure of packed data.

At the first stage intermediate matrix $[T]$ is obtained according to $[T] = [X] * [C^T]$ formula. The cosine coefficients of $[C^T]$ are loaded to the VCP multiply unit in two columns, and eight vectors of $[X]$ come in series to the «X» input of the multiply unit (Fig. 1,2). At each clock the matrix of weighted coefficients of VCP unit are multiplied by one vector-string x_i of matrix $[X]$, and as a result the pair of numbers of $[T]$ matrix: $T(i,j)$ and $T(i,j+1)$ is carried out (Fig. 2). Thus the whole multiplication of two 8x8 matrices requires 4 reloads of the VCP multiply unit and it is performed during $4*8=32$ multiply steps. Taking into account that the cosine coefficients of $[C^T]$ are constant values, for the DCT processing of whole images we can regroup calculations to use the VCP multiply unit contents without periodical VCP reloading. So the average calculation time of each element of matrix $[T]$ would be about 0.5 clock. Effectiveness of multiply-add operations is achieved by using SIMD instruction - rep 32 operating with blocks of 32 64-bit words.

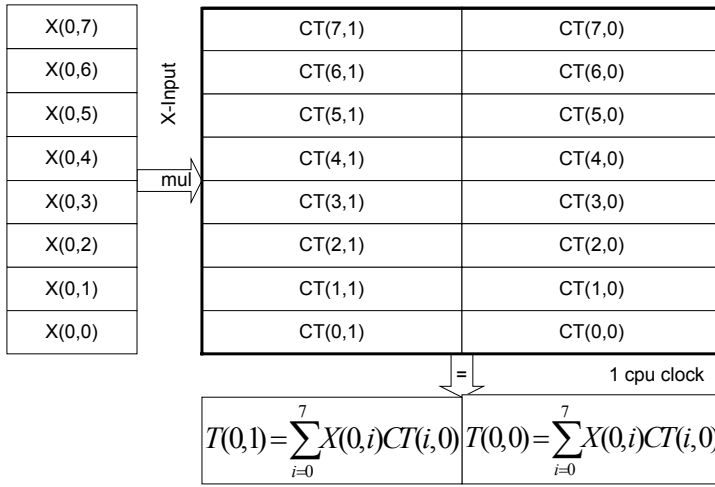


Fig. 2 Stage 1.The multiplication of the first vector-string of [X] by two left columns of [C^T].

At the second stage matrix [Y] is computed according to [Y]=[C]*[T] formula. The elements of intermediate matrix [T] are also loaded to the VCP unit in two columns, and vectors of matrix [C] come in series to the “X” input of the VCP multiply unit (Fig. 3). This process is analogous to the previous one except for the size of data blocks coming to the VCP multiplier input. Each multiply cycle includes only 8 vector-matrix multiplications (rep 8 instruction is used), after that VCP matrix content is updated with the next portion of matrix [T]. Since the VCP reloading takes 32 CPU clocks and we have only 8 vectors to multiply against the VCP reloading background, then the total cycle time would be equal to 32 clocks, therefore each Y(i,j) element would be calculated four times longer – i.e. about 2 clocks.

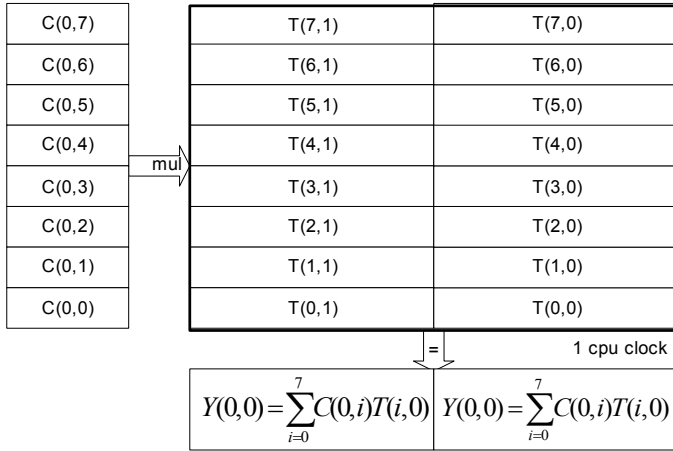


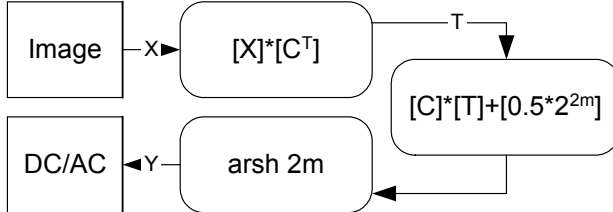
Fig. 3. Stage 2. Multiplication of the first vector-string of matrix [C] by two left columns of matrix [T].

In the result the summary evaluation time of [Y]=[C]×[X]×[C^T] multiplying is: 64*0.5+64*2=160 clocks. The real value is about 190 clocks or less depending on image size. It is also necessary to add 32 clocks for scaling down. It is important, that the scaling down stage may be replaced without time expense from DCT to the following functions, like quantization or Z-reordering. In that case average DCT performance can be accepted as 3 clocks per pixel.

Analysis of the DCT accuracy and methods of its enhancement.

While the algorithm is being developed it is important to take into account several factors at the same time: input and output data, accuracy and necessary performance. All of them are mutually exclusive, but as the analysis showed, a special initialization of the cosine coefficients and negligible variations of calculations can essentially enhance the accuracy without performance reducing.

The DCT accuracy depends on fixed-point notation of the coefficients C_{ij} and the way of final processing of results. The fixed-point format is defined by the word length, the scaling factor $-M$ and the round function $-C_{fix} = \text{round}(C_{fp} * M)$. The final processing consists of scaling down with products rounding. The cosine coefficients C_{ij} have the $(-0.491 \dots +0.491)$ range, therefore if we apply the scaling factor $M=2^m$ we can form fixed-point numbers in m -bit words. The best rounding type as in the conversion to the fixed-point format as in the final processing is rounding to the nearest integer, because this scheme showed the best results on the vector core and may be easily implemented in parallel, using hardware supported multiply-add operations. In order to round a fixed-point number it is necessary to add the constant $1/2 * 2^{2m}$ (0.5 in the floating-point notation) to the $[C]*[T]$ product as shown below:



The analysis of the DCT accuracy depending on the digit-capacity $-m$ was made according to the shown scheme. In this context the digit-capacity means the number of significant bits in some n -bit word, where $n \geq m$. For simplicity all coefficients of $[C]$ and $[C^T]$ matrices were formed with the common scaling factor 2^m , and the absence of overflow was assumed everywhere.

The tests were executed over all 8×8 blocks of "Lena" 256×256 image. The accuracy was estimated by similarity of DCT results received using the fixed and the floating-point arithmetic. The accuracy was estimated by means of the following most informative criteria:

1. $RMSE(X) = \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} (x_{fix}(i) - x_{fp}(i))^2}$ - root-mean-square error. Results of DCT transforms

were placed as X : DC (block brightness index) and AC (spatial frequencies). A separate AC/DC analysis simplifies interpretation of the result.

2. $PSNR(X) = 20 \cdot \log\left(\frac{255}{RMSE(X)}\right)$ - signal to noise ratio between the initial image and the

reconstructed image according to $[SrcImage \Rightarrow DCT \Rightarrow Quant \Rightarrow Dequant \Rightarrow IDCT \Rightarrow RecImage]$ scheme.

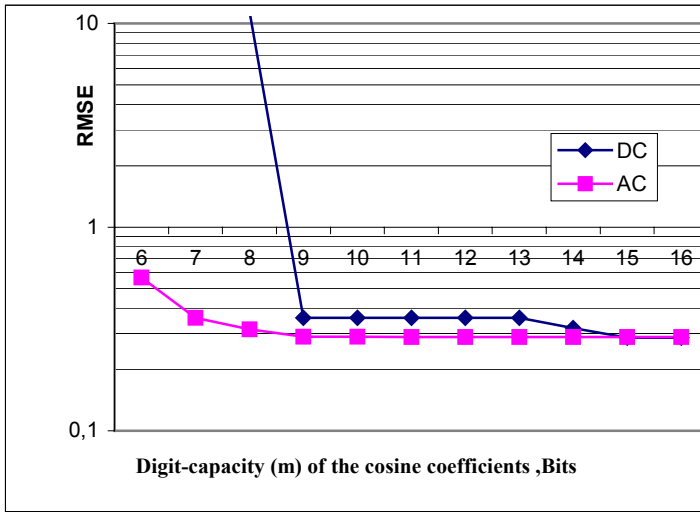


Fig. 4 RMSE of DC and AC coefficients

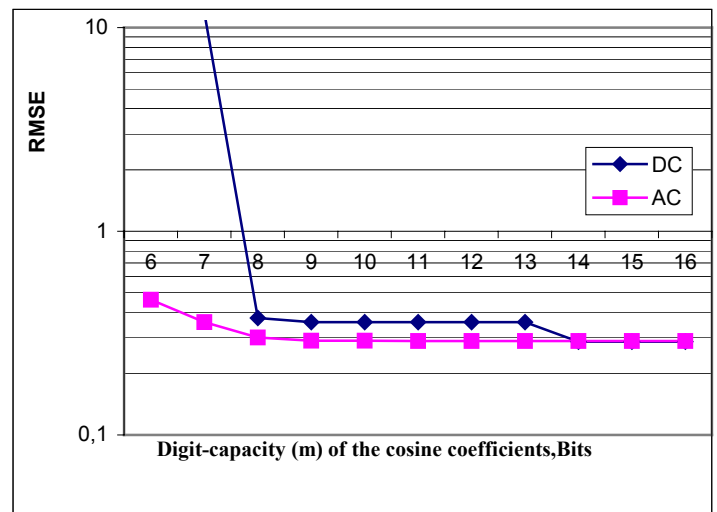


Fig. 5 RMSE of DC and AC coefficients with bit correction of the cosine coefficients $C(i,j)$

Fig. 4 shows the difference between float and fixed-point DCT output. DC and AC coefficients are compared in dependence on digit capacity m of the fixed-point cosine coefficients. There is the sudden RMSE drop of the DC curve at $m = 9$ bit.

Increase of accuracy is caused by the character of notation of some coefficients C_{ij} , involved in DC calculation. These coefficients are located in the first column of matrix $[C^T]$ and in the first row of matrix $[C]$, all of them are equal to 0,35355339059327. In the fixed-point format this value is written in a binary form: 0.0101 1010 1000 0010 0111... As we can see, 5 zeros follow the 9-th digit after the point. It is evident that the accuracy of fixed-point representation of 0.35355.. value using only 9 bits (0101 1010 1b) is the same as in case of 14 bits (0101 1010 1000 00b). In case of 8-bit using the big RMSE error is caused by a rounding ambiguity. Rounding of the 8-th bit to the smaller direction (to 0101 1010) is equally correct as rounding to the larger one (to 0101 1011). It is obvious that if only one way of rounding is used, then coefficients C_{ij} will introduce some systematic error to DC. This defect may be eliminated by making cosine coefficients using the probabilistic round-off, what is equivalent to interchange rounding-up and rounding-down of $C^T(i, 0)$ and $C(0, j)$. As the plots in Fig. 5 and Fig. 8 show such correction enhances the DCT accuracy at low digit-capacities (8 bits and less) very much.

In most tasks it is required to process unsigned image data. In these cases some additional enhancement of accuracy is possible (see Fig. 7, Fig. 8) by applying the mechanism of DC-displacement. This mechanism consists in input data transform from the $[0..2^m-1]$ range to the $[-2^{m-1}.. +2^{m-1}-1]$ range by subtracting the 2^{m-1} -constant. For usual 8-bit images it corresponds to the shift of range from $[0..255]$ to $[-128..127]$. As a result the DCT output will differ only in DC coefficients for some constant value. The DCT results become absolutely correct after appropriate modification of the DC element. Note that such mechanism is built into JPEG standard. The scheme shown in Fig. 6 lets us also to process any 8-bit images in other standards, like MPEG, H.263 and so on.

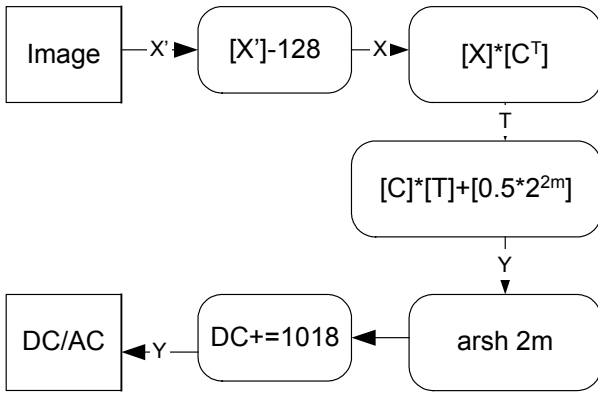


Fig. 6 Alternative scheme of the DCT calculation with the DC-displacement mechanism

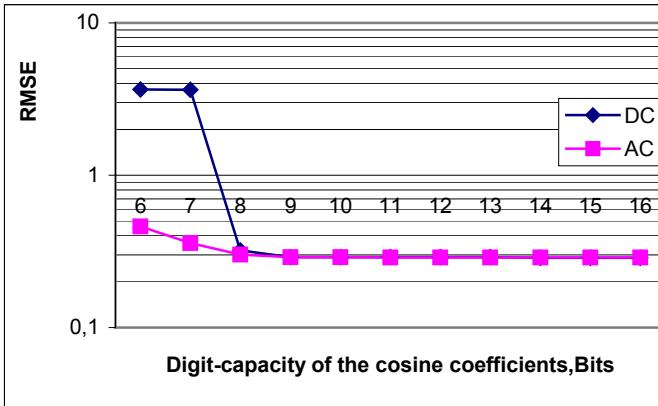


Fig. 7 RMSE of DC/AC coefficients using bit correction and the DC-displacement mechanism

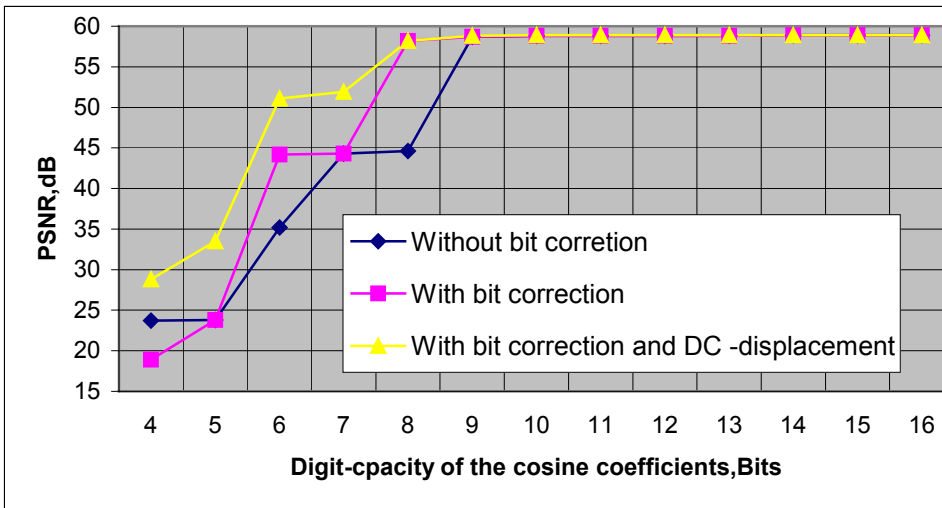


Fig. 8 Signal to noise ratio between the initial image and the reconstructed image according to $[SrcImage \Rightarrow DCT_{fix} \Rightarrow IDCT_{fp} \Rightarrow ReclImage]$ scheme.

In practice realization of the mechanism of DC-displacement according to the scheme given in Fig. 6 allows to use the more precise 12-bit fixed-point format of the cosine coefficients $[C^T]$ without overflow. Table 1 provides results to compare accuracy between fixed-point and floating-point processing. It represents the dependence of signal to noise ratio of reconstructed image “Lena 256x256” on quantization factor Q .

Q	PSNR,dB		
	DCT _{fix} -IDCT _{fix}	DCT _{fix} -IDCT _{fp}	DCT _{fp} -IDCT _{fp}
1	52.84	58.65	58.9
2	48.41	49.63	49.65
4	45.74	46.38	46.39
8	42.13	42.37	42.37
16	38.03	38.12	38.12
32	33.78	33.82	37.82
64	29.80	29.84	29.84

Table 1. PSNR between the initial and the reconstructed image according to [SrcImage=>DCT=>Quant=>Dequant=>IDCT=>ReImage] depending on the quantization factor Q.

Short description of the JPEG codec implementation on NeuroMatrix architecture .

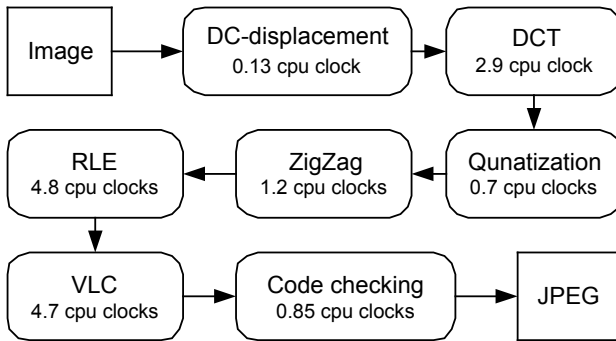


Fig. 9 Diagram of the JPEG coding

Fig. 9 illustrates the diagram of JPEG coding and NM6403 processor time distribution between functional blocks. Average processing time in clocks per pixel is shown for each block. For the last three blocks the time performance is relative enough because it strongly depends on the quantization table and the initial image. The summary average time processing of one pixel is about 15-16 clocks what corresponds (at NM6403 clock rate of 40 MHz) to 25 frames per second for black and white CIF images.

The whole preprocessing including DC-displacement, DCT, quantization, Z-reordering and result clipping is vectorized very well because these procedures do not depend on input data. Other procedures are conducted with RLE (Run Length Encoding) and VLC (Variable Length Encoding). These are input data depending functions having jump conditional operations and handling variable length code-words, therefore full vector processing becomes impossible. However some processing pieces may be picked out for vector co-processor in order to accelerate the following computing on the scalar core.

In particular, using vector preprocessing in RLE and code checking allows to essentially reduce scalar instructions and minimize conditional jumps.

In the both cases the essence of vector preprocessing is compiling a binary vector from a parsed sequence of 64 elements $\{X_i, i=0..63\}$. The binary vector B is a 64-bit word, each bit of which is determined by Boolean function of the appropriate argument in the sequence: $b_i = f(x_i)$. For example, the following simplest Boolean functions may be easy implemented on NM6403 vector core:

$$f(x_i) = \begin{cases} 0, x_i \geq 0 \\ 1, x_i < 0 \end{cases}, f(x_i) = \begin{cases} 1, x_i \geq 0 \\ 0, x_i < 0 \end{cases}, f(x_i) = \begin{cases} 0, x_i = 0 \\ 1, x_i > 0 \end{cases}, f(x_i) = \begin{cases} 0, x_i - \text{even} \\ 1, x_i - \text{odd} \end{cases} \text{ and so on.}$$

Since these functions are executed on vector co-processor, so 2-4-8-16... result bits (depending on x_i digit-capacity) can be generated at one time. Using a few accumulations of these bits in 64-bit word

we can entirely compile the binary vector. More complicated functions, like $f(x_i) = \begin{cases} 0, x_i = C \\ 1, x_i \neq C \end{cases}$

could be also implemented by composing from elementary logical and arithmetic operations.

In particular, in the code checking procedure 0xFF symbol is checked for presence in the binary stream generated by Huffman encoding (VLC). According to JPEG standard 0xFF symbol is reserved as an auxiliary code-word and it should be replaced with 0xFF00 code-word at each its occurrence in the bitstream. The analysis on the scalar core of the binary vector produced by

$$f(x_i) = \begin{cases} 1, x_i = FF \\ 0, x_i \neq FF \end{cases} \text{ function allows us to easily find the location of 0xFF code-word. Relatively}$$

fast computing of binary vector on NM6403 processor (8 - 64 clocks per a 64-bit binary vector depending on x_i digit-capacity and the $f(x)$ function complexity) allows us to apply this mechanism in the searching tasks.

The other way of the binary vector using is applied in RLE (Run Length Encoding) procedure. RLE procedure is called to find continuous chains of identical symbols in some sequence and to replace them with a pair <Counter of symbol repeats, Symbol>, so data redundancy is reduced. During image processing the chains of zeros are searched in blocks of 64 symbols and then they are replaced with a <RUN,LEVEL> pair, where RUN is the chain length (in the range between 0 and 15), LEVEL- is a nonzero symbol following a chain or zeros if length exceeds 15.

Let us illustrate search of such chains and their lengths on an example.

Let assume that we have some sequence X after executing the DCT and Z-reorder procedures. And let it consist mainly of zeros, and 11 nonzero symbols (a,b,c,d,e,f,g,h,i,j,k) placed at 0,2,4,5,6,13,19,25,26,34,53 positions accordingly:

$X =$ 0000000000k0000000000000000000j000000ih00000g00000f000000edc0b0a
 $x_{63} \dots \dots x_1, x_0$

Produce the binary vector according to the formula : $b_i = f(x_i) = \begin{cases} 0, x_i = 0 \\ 1, x_i \neq 0 \end{cases}, i=0..63$

$B =$ 000000000010000000000000000000010000000110000010000010000001110101
 $b_{63} \dots \dots b_1, b_0$

Invert the binary vector: $Y[00] = \text{Not}(B)$

$Y[00] =$ 111111111101111111111111111111101111111001111101111101111110001010

Run an iterative cycle from $i=1$ to 15

$$Y[i] = (Y[i-1] \gg 1) \text{ AND } (Y[i-1])$$

$Y[01] =$ 0111111111011111111111111111110111111100111110111110111111000101
 $Y[02] =$ 0011111111100111111111111111110011111100011110011110011111000000
 $Y[03] =$ 0001111111100011111111111111110001111100001110001110001111000000
 $Y[04] =$ 0000111111100001111111111111110000111100000110000110000111000000
 $Y[05] =$ 000001111110000011111111111111100000111000001100000100000110000000
 $Y[06] =$ 00000011111000000111111111111110000001100000000000000001000000
 $Y[07] =$ 00000001111000000011111111111110000001000000000000000000000000
 $Y[08] =$ 00000000111000000001111111111100000000000000000000000000000000
 $Y[09] =$ 00000000011000000000111111111100000000000000000000000000000000
 $Y[10] =$ 00000000001000000000111111111100000000000000000000000000000000
 $Y[11] =$ 00000000000000000000111111111100000000000000000000000000000000
 $Y[12] =$ 00000000000000000000111111110000000000000000000000000000000000
 $Y[13] =$ 00000000000000000000111111000000000000000000000000000000000000
 $Y[14] =$ 00000000000000000000111110000000000000000000000000000000000000
 $Y[15] =$ 00000000000000000000111100000000000000000000000000000000000000

Let sum bits in each column and form vector S of 64 numbers:

S= 0123456789A0123456789ABCDEF0123456700123450123450123456000101

X= 0000000000k0000000000000000000j0000000ih00000g00000f000000edc0b0a

Every returned sum s_i indicates the amount of continuous zeros standing before x_i , and the incremented sum points to the location of the next symbol (LEVEL). Taking displacements s_i+1 from S vector and making parallel crossings in both S and X vectors on them we will have the required <RUN,LEVEL> set:

<a,?><b,1><c,1><d,0><e,0><f,6><g,5><h,5><i,0><j,7><0,15><a,2><?,10>

Resume

The DCT analysis based on fixed-point arithmetic shows that 8-bit representation of the cosine coefficients C_{ij} has accuracy comparable with floating-point arithmetic. The signal to noise ratio of the reconstructed image using 8-bit cosine coefficients precision is about 58.2dB (Fig.7), while the DCT accuracy using the 14-bit and more digit-capacities achieves the limit of - PSNR=58.9dB. At the same time computation scheme with different digit-capacities of the $[C]$ and $[C^T]$ cosine coefficients - 8 and 12 bit accordingly (PSNR=58.6dB) makes the difference between fixed-point arithmetic and floating-point arithmetic even less significant, and if we use quantizing factor 4 or more the difference disappears. A distinctive feature of the direct way of DCT calculation from fast algorithms with the fixed-point arithmetic consists in a minimal quantity of stages of multiplication, that on one hand results in smaller error accumulation, and on the other allows to build calculations on the vector co-processor.

Applicability of the vector unit of NM6403 processor in coding tasks was demonstrated. The flexible programmable structure of the vector co-processor and the potential to operate data with variable word length data give us additional advantages in implementation of complex algorithms.

References:

- [1] *V.Kashkarov, S.Mushkaev* "Parallel Execution of FFT Algorithms on NeuroMatrix Architecture" <http://www.module.ru/>
- [2] *V.Kashkarov* "Fast Hadamard Transform Application Repor" <http://www.module.ru/>
- [3] *Seehyun Kim and Wonyong Sung*, "Fixed-Point Error Analysis and Word Length Optimization of 8x8 IDCT Architectures"
- [4] NeuroMatrix architecture documentation is available at <http://www.module.ru/>