Implementing image compression algorithms on NeuroMatrix architecture: New approaches

By Sergey Mushkaev and Sergey Landyshev

This article discusses the possibilities the NM6403 processor opens up for static image compression. The authors present the Discrete Cosine Transform (DCT) algorithm and accuracy enhancement methods. In addition, the article briefly describes using the NM6403 processor for JPEG coding and using a vector co-processor for coding tasks.

Introduction

Successfully compressing images presents a complex computational challenge. Different architectures demand different methods to optimize compression, but all methods minimize arithmetic operations as one way to accomplish this optimization. The NM6403 processor helps minimize arithmetic operations by employing an algorithm that executes an optimal set of streaming operations over data blocks. Each block can be entirely processed at one processor step. Taking such an approach involves logical and arithmetic operations with small matrices and vectors. This method enables the NM6403 to organize parallel calculations of Fast Fourier Transform (FFT) with radix 16 and 32.[1]

Forward 2D DCT computation algorithm on NM6403 processor

Let's examine an image compression task involving direct DCT 8 x 8 calculus that does not require additional minimization. We can define these streaming operations starting from the forward DCT 8 x 8 formula, as defined in Equation 1:

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

$$u,v,x,y = 0,1,2,\ldots,7$$

$$w(u=0) = \frac{1}{\sqrt{2}};\ w(u \neq 0) = 1$$

$$w(v=0) = \frac{1}{\sqrt{2}};\ w(v \neq 0) = 1$$

Equation 1

In Equation 1 (x, y) are spatial coordinates and (u,v) are coordinates in the frequency domain. Equation 2 illustrates the results of putting Equation 1 into the equivalent matrix form of DCT:

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

Equation 2

In Equation 2 [X] is the initial 8 x 8 matrix, [Y] is the resulting 8 x 8 matrix of discrete cosine transforms, and ([C], $[C^T]$) are 8 x 8 matrices of cosine coefficients C_{ij}.

Thus, DCT computation consists of multiplying the initial matrix by two matrices [C] and $[C^T]$. We can carry this out in two ways:

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

Basing all computations on the fixed-point arithmetic causes variants (1) and (2), above, to differ widely from one another while the Vector Co-Processor (VCP) is implemented. The best choice depends on the digit capacity of [X], [C], and $[C^T]$ matrices. The structure of the VCP multiply unit makes the product of [C] and [X] accumulate with the same word length as that of the [X] multiplier. Therefore, the first variant imposes tighter constraints on input data word length. Variant (2) does not depend on input data digit capacity and so offers more universal and flexible cosine coefficients word length choice for the [C] and $[C^T]$ matrices. This increased flexibility makes variant (2) worthy of further consideration.

The whole process takes place in two stages:

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

Let's look at the simplest, most efficient DCT algorithm implementation for 8-bit input signed data. Setting the word length of $[C^T]$ and [Y] matrices to 32-bit prevents product overflow and simplifies further data processing after DCT. The transform accuracy depends on the number of significant bits in the [C] and $[C^T]$ matrices' coefficients. Using seven least significant bits and the sign achieves acceptable precision. Therefore using 8-bit word length of the matrix [C] suffices. Note that all elements of matrices are considered as signed, therefore input values X_{ij} range between –128 and 127, but this does not exclude the processing of the usual 8-bit images.

The cosine coefficients $C_{fp}(i, j)$ given in the floating-point format (index $_{fp}$) range between -0.491 and +0.491. They are converted to the fixed-point format (index $_{fix}$) according to $C_{fix}(i, j)=\mathrm{round}(C_{fp}(i, j)*2^8)$ formula, where 2^8 is the scale factor and round() is the round-off function returning the nearest integer. Using the same scale factor of 2^8 for [C] and $[C^T]$ matrices allows the accumulation of the result without overflow into the 32-bit data of the [T] and [Y] matrices. In addition, using the same scale factor makes it possible to scale down the products only once using the 16-bit right shift (arsh 16) of the [Y] matrix. Thus, as Figure 1 shows, the whole DCT computing process consists of two stages: (1), multiplying and (2), scaling down.

 Copyright 2004

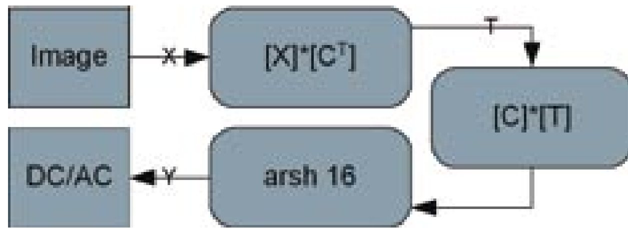

Figure 1

Figure 2 details the matrix multiplying process and the internal fragmentation of the packed data (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).

At the first stage, the $[T]=[X]*[C^T]$ formula produces the intermediate matrix [T]. 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 (See

[X]

63						15 8	7 0
X(0,7)	X(0,6)	X(0,5)	X(0,4)	X(0,3)	X(0,2)	X(0,1)	X(0,0)
X(1,7)	X(1,6)	X(1,5)	X(1,4)	X(1,3)	X(1,2)	X(1,1)	X(1,0)
X(7,7)	X(7,6)	X(7,5)	X(7,4)	X(7,3)	X(7,2)	X(7,1)	X(7,0)

Stage 1

X

$[C^T]$

63 32	31 0					63 32	31 0
CT(0,1)	CT(0,0)	CT(0,3)	CT(0,2)	CT(0,5)	CT(0,4)	CT(0,7)	CT(0,6)
CT(1,1)	CT(1,0)	CT(1,3)	CT(1,2)	CT(1,5)	CT(1,4)	CT(1,7)	CT(1,6)

=

$[T]=[X]*[C^T]$

[C]

C(0,7)	C(0,6)	C(0,5)	C(0,4)	C(0,3)	C(0,2)	C(0,1)	C(0,0)
C(1,7)	C(1,6)	C(1,5)	C(1,4)	C(1,3)	C(1,2)	C(1,1)	C(1,0)
C(7,7)	C(7,6)	C(7,5)	C(7,4)	C(7,3)	C(7,2)	C(7,1)	C(7,0)

Stage 2

X

T(0,1)	T(0,0)	T(0,3)	T(0,2)	T(0,5)	T(0,4)	T(0,7)	T(0,6)
T(1,1)	T(1,0)	T(1,3)	T(1,2)	T(1,5)	T(1,4)	T(1,7)	T(1,6)

=

[Y]=[C]*[T]

Y(0,1)	Y(0,0)	Y(0,3)	Y(0,2)	Y(0,5)	Y(0,4)	Y(0,7)	Y(0,6)
Y(1,1)	Y(1,0)	Y(1,3)	Y(1,2)	Y(1,5)	Y(1,4)	Y(1,7)	Y(1,6)

Figure 2

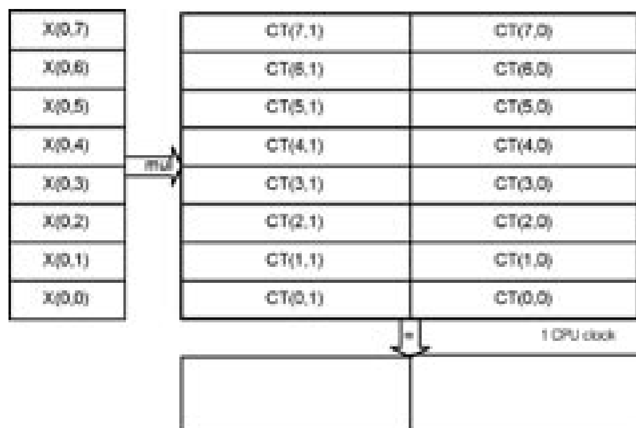

Figure 3

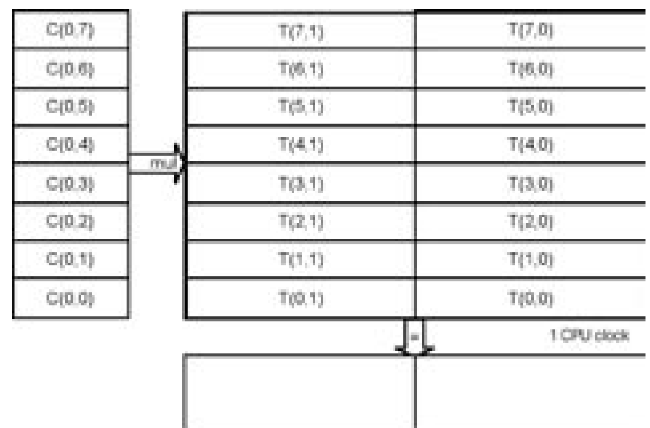

Figure 4

Copyright 2004

Figures 2, 3). At each clock the matrix of weighted coefficients of the VCP unit are multiplied by one vector-string x_i of matrix [X] and a resulting pair of [T] matrix numbers. Figure 3 shows this calculation carried out for T(i, j) and T(i, j+1). Thus the whole multiplication of two 8 x 8 matrices requires four reloads of the VCP multiply unit, and takes place 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 clocks. Using Single Instruction Multiple Data (SIMD) instruction (rep 32 operating with blocks of 32 64-bit words) produces effective multiply-add operations.

At the second stage, the formula [Y]=[C]*[T] computes matrix [Y]. Intermediate matrix [T] elements 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 (Figure 4). This process is analogous to the previous one except for the size of the data blocks coming to the VCP multiplier input. Each multiply cycle includes only eight vector-matrix multiplications (rep 8 instruction is used), then VCP matrix content is updated with the next portion of matrix [T]. Since the VCP reloading takes 32 CPU clocks and we have only eight vectors to multiply against the VCP reloading background, 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.

The resulting summary evaluation time of multiplying $[Y]=[C]*[X]*[C^T]$ is:

$$64 * 0.5 + 64 * 2 = 160\ clocks$$

Equation 3

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's important to note that the scaling-down stage may be replaced from DCT to the functions such as quantization or Z-reordering with no time penalty. In that case average DCT performance can be accepted as three clocks per pixel.

DCT accuracy analysis and enhancement methods

Several factors play important simultaneous roles in algorithm development:

- Input and output data
- Accuracy
- Performance requirements

Although these factors are mutually exclusive, our analysis showed that a special initialization of the cosine coefficients and negligible variations of calculations could enhance accuracy without reducing performance.

The DCT accuracy depends on fixed-point notation of the coefficients C_{ij} and final results processing, The word length, the scaling factor M, and the round function C_{fix}=round(C_{fp}*M) define the fixed-point format. The final processing consists of scaling down with product rounding. The cosine coefficients C_{ij} range from -0.491 to +0.491, therefore if we apply the scaling factor $M=2^m$ we can form fixed-point numbers in m-bit words. Rounding to the nearest integer shows the best results on the vector core. Using hardware supported multiply-add operations makes this rounding type easy to implement in parallel. Rounding a fixed-point number requires adding the constant $1/2*2^{2m}$ (0.5 in the floating-point notation) to the [C]*[T] product (Figure 5):

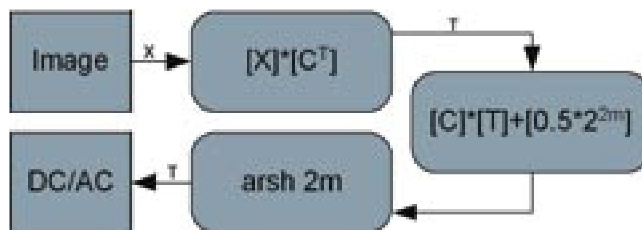

Figure 5

Figure 5 shows how DCT accuracy analysis depends on digit capacity *m* when rounding takes places to the nearest integer. In this context the digit capacity means the number of significant bits in some n-bit word, where n ≥ m. For simplicity all coefficients of [C] and $[C^T]$ matrices were formed with the common scaling factor 2m, and the absence of overflow was assumed everywhere.

The NM6403 development team tested all 8 x 8 blocks of a 256 x 256 image dubbed *Lena*. The team estimated accuracy according to the similarity of the DCT results achieved using fixed-point and floating-point arithmetic. Estimating accuracy involved the following criteria:

- The root-mean-square error shown in Equation 4.
  Results of the DCT transforms were placed as X being DC coefficients (block brightness index), and AC coefficients (spatial frequencies). A separate AC/DC analysis simplifies results interpretation.

$$RMSE(X) = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(x_{fix}(i) - x_{fp}(i))^2}$$

Equation 4

- The signal-to-noise ratio (Equation 5) is between the initial image and the reconstructed image according to the [SrcImage=>DCT=>Quant=>Dequant=>IDCT=>RecImage] scheme.

$$PSNR(X) = 20 \cdot \log\left(\frac{255}{RMSE(X)}\right)$$

Equation 5

Figure 6 shows the difference between floating-point and fixed-point DCT output. DC and AC coefficients are compared for their dependence on the digit capacity *m* of the fixed-point cosine coefficients. At the *m* = 9 bit, a sudden drop in RMSE of the DC curve occurs, also shown in Figure 6.

The characters of notation for some of the C_{ij} coefficients involved in the DC calculation increased accuracy. These coefficients are located in the first column of matrix $[C^T]$ and in the first row of matrix [C]. All of these coefficients are equal to 0.35355339059327. In the fixed-point format this value is written in a binary form as *0.0101 1010 1000 0010 0111…* . As we can see, five zeros follow the ninth digit after the decimal point. This value demonstrates that the accuracy of the fixed–point representation of 0.35355… value using only 9 bits (0101 1010 1b) equals the accuracy achieved using 14 bits (0101 1010 1000 00b). In the 8-bit case, a rounding ambiguity causes the large RMSE error. Rounding of the 8-bit down to 0101 1010 is as correct as rounding up to 0101 1011. If we use only one rounding scheme, then coefficients C_{ij} will introduce some systematic error to

Copyright 2004

DC. Eliminating this defect requires making cosine coefficients using the probabilistic round-off that is equivalent to interchange rounding up and rounding down of $C^T(i, 0)$ and $C(0, j)$. As the plots in Figure 7 and Figure 8 show, such correction significantly increases DCT accuracy at low digit-capacities (8-bits and less).

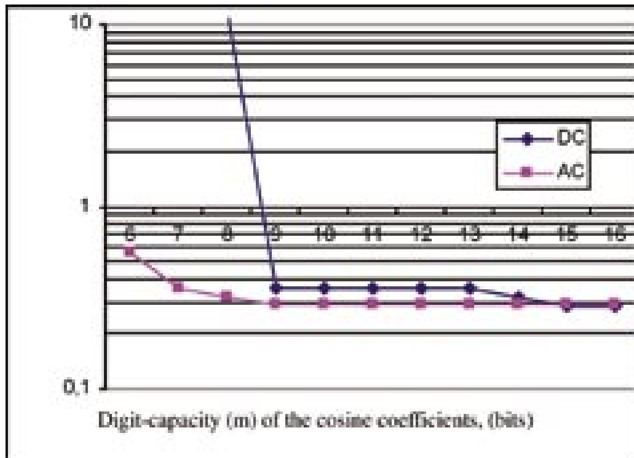

Figure 6

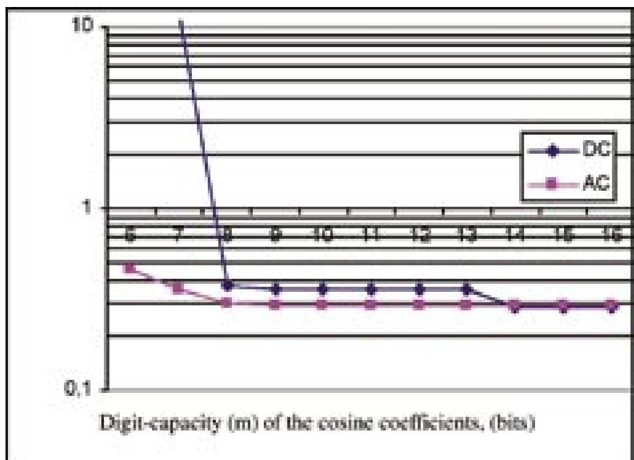

Figure 7

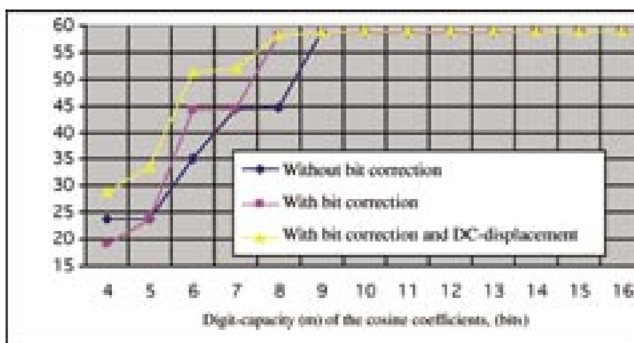

Figure 8

Most tasks require unsigned image data processing

Applying the DC-displacement mechanism can increase accuracy for these tasks (see Figures 8 and 9). With this mechanism the 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 typical 8-bit images it corresponds to a range shift 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 DC element modification, such as that built into the JPEG standard. The scheme shown in Figure 10 lets us process any 8-bit images in MPEG, H.263, and similar standards.

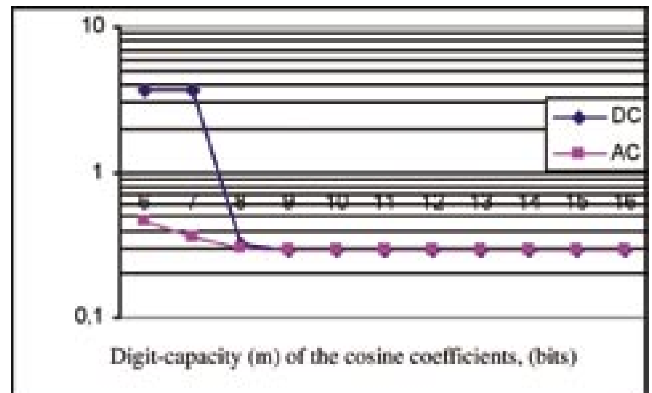

Figure 9

In practice, performing DC-displacement according to the scheme in Figure 10 allows more precise 12-bit fixed-point format use of the cosine coefficients $[C^T]$ without overflow. Table 1 compares fixed-point and floating-point processing accuracy. This table shows the dependence of signal-to-noise ratio on the reconstructed image *Lena* according to quantization factor Q.

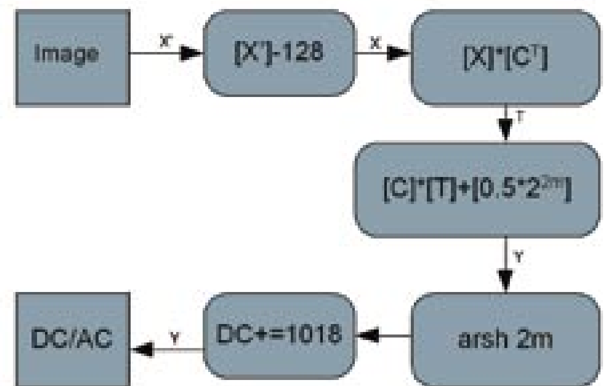

Figure 10

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

JPEG codec implementation on NeuroMatrix architecture

Figure 11 illustrates JPEG coding and NM6403 processor time distribution between functional blocks. Average processing time

Copyright 2004

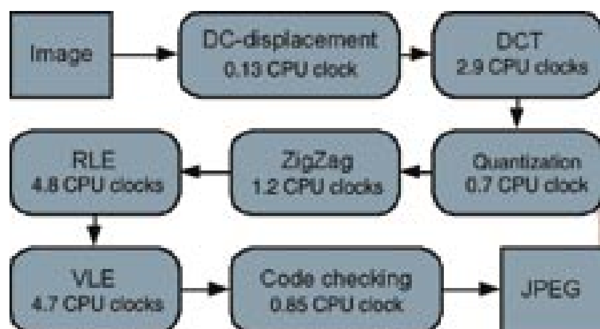

Figure 11

in clocks per pixel is shown for each block. For the last three blocks the time performance is relative, depending strongly on the quantization table and the initial image. The summary average processing time of one pixel is about 15 to 16 clocks, corresponding (at the NM6403's 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 since these procedures do not depend on input data. Other procedures are conducted with Run Length Encoding (RLE) and Variable Length Encoding (VLE). These input-data dependent functions have jump conditional operations and handle variable length code words making full vector processing impossible. However, some processing pieces may be picked out for the VCP in order to accelerate the subsequent scalar core computing. In particular, vector preprocessing in RLE and code checking reduces scalar instructions and minimizes conditional jumps.

In both cases compiling a binary vector from a parsed sequence of 64 elements ($X_{i,i}$=0…63) is key to vector preprocessing. The binary vector B is a 64-bit word, where each bit is determined by a Boolean function of the appropriate argument in the sequence (Equation 6):

$$b_i = f(x_i)$$

Equation 6

For example, the following simplest Boolean functions may be easily implemented on the NM6403 vector core (See Equation 7).

Executing these functions on a vector co-processor enables 2, 4, 8, 16…resulting bits (depending on x_i digit capacity) to be generated at one time. Using a few accumulations of these bits in a 64-bit word we can completely compile the binary vector. More complicated functions, such as Equation 8, could also be implemented.

Composing elementary logical and arithmetic operations can derive all these functions.

For instance, the code checking procedure includes looking for the 0xFF symbol in the binary stream generated by Huffman encoding (VLC). According to the JPEG standard the 0xFF symbol is reserved as an auxiliary code word and should be replaced with 0xFF00 at each of its occurrences in the bit stream. An analysis on the scalar core of the binary vector (Equation 9) makes finding 0xFF occurrences easy.

The NM6403 processor's relatively fast computing of the binary vector (8 to 64 clocks per 64-bit binary vector depending on x_i digit capacity and f(x) function complexity) makes computing the binary vector a useful searching tool.

The RLE procedure can also benefit from binary vector use. The RLE procedure calls for finding continuous chains of identical symbols in some sequence and replacing them with a pair of <Counter of symbol repeats, Symbol> values, to reduce data redundancy. 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), and LEVEL is a nonzero symbol following a chain of zeros if length exceeds 15.

For example, assume that there is a sequence X resulting from executing the DCT and Z-reorder procedures. Let X consist mainly of zeros, and let 11 nonzero symbols (a, b, c, d, e, f, g, h, i, j, k) be placed at the 0, 2, 4, 5, 6, 13, 19, 25, 26, 34, 53 positions accordingly (Equation 10).

Using I = 0...63 to produce the binary vector we get the result, shown in Equation 11. Inverting the binary vector Y[00] = Not(B) gives the result in Equation 12. Now running an iterative cycle from i = 1 to 15 Y[i] = (Y[i-1]>>1) and (Y[i-1]) yields Equation 13. Summing the bits in each column forms the vector *S* of 64 numbers shown in Equation 14. 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 shown in Equation 15.

Conclusion

DCT analysis based on fixed-point arithmetic shows that 8-bit representation of the cosine coefficients C_{ij} produces accuracy comparable to that of floating-point arithmetic. The signal-to-noise ratio of the reconstructed image using 8-bit cosine coefficients yields precision of about 58.2dB (Figure 9), while the DCT accuracy using the 14-bit and higher digit capacities achieves the limit of PSNR = 58.9dB. At the same time the computation scheme with different digit capacities of the [C] and [C^T] cosine coefficients, 8-bit and 12-bit respectively (PSNR = 58.6dB), makes the difference between fixed-point and floating-point arithmetic even less significant. If we use a quantization factor of 4 or more the difference disappears altogether. Performing DCT calculation from fast algorithms using fixed-point arithmetic reduces multiplication steps resulting in smaller error accumulation. In addition this method frees the VCP for calculations.

We demonstrated how the NM6403 processor's vector unit speeds coding tasks. The flexible programmable structure of the VCP[2] and the ability to process variable word length data brings more advantages to the implementation of complex algorithms.

References:

[1] V.Kashkarov, S.Mushkaev, "Parallel Execution of FFT Algorithms on NeuroMatrix Architecture"; available at www.module.ru

[2] NeuroMatrix architecture documentation is available at www.module.ru

Sergey Mushkaev *was born in Moscow, Russia, in 1977. In 2000, he received B.S. and M.S. degrees in electronics engineering and automatics of physical installations from Moscow Engineering Physics Institute State University (MEPhI), Moscow, Russia. Since 1999 Sergey has been working at the Moscow Research Center "Module" where he is involved in the development of various DSP algorithms for the NM6403 processor. Sergey's research activities are focused on digi-*

 Copyright 2004

tal image sequence processing and still and moving image compression applied to NeuroMatrix architecture.

Sergey Landyshev *was born in Moscow, Russia, in 1970. In 2001, Sergey received B.S. and M.S. degrees in computer-based complexes, systems, and networks from Moscow Engineering Physics Institute State University (MEPhI), Moscow, Russia. In 2002, Sergey joined the Research Center "Module" in Moscow as a software developer. Sergey's research interests focus on vectorization of image compression algorithms and cryptographic methods on the NeuroMatrix processor. He has also worked on developing system level and image processing software.*

For further information, contact Sergey Mushkaev at:

Research 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
E-mail: landysh@module.ru
Web site: www.module.ru

$$f(x_i) = \begin{Bmatrix} 0, x_i \geq 0 \\ 1, x_i < 0 \end{Bmatrix}, f(x_i) = \begin{Bmatrix} 1, x_i \geq 0 \\ 0, x_i < 0 \end{Bmatrix}, f(x_i) = \begin{Bmatrix} 0, x_i = 0 \\ 1, x_i > 0 \end{Bmatrix}, f(x_i) = \begin{Bmatrix} 0, x_i - even \\ 1, x_i - odd \end{Bmatrix} \ldots$$

Equation 7

$$f(x_i) = \begin{Bmatrix} 0, x_i = C \\ 1, x_i \neq C \end{Bmatrix}$$

Equation 8

$$f(x_i) = \begin{Bmatrix} 1, x_i = FF \\ 0, x_i \neq FF \end{Bmatrix}$$

Equation 9

$$b_i = f(x_i) = \begin{Bmatrix} 0, x_i = 0 \\ 1, x_i \neq 0 \end{Bmatrix}, i = 0 \ldots 63$$

$$X = 0000000000k00000000000000000\,j0000000ih00000g00000f000000edc0b0a$$

$$x63 \ldots x_1, x_0$$

Equation 10

Giving

$$B = 000000000010000000000000000000100000001100000100000100000001110101$$

$$b_{63 \ldots b_1, b_0}$$

Equation 11

$$Y[00] = 111111111101111111111111111111011111111001111101111101111111000101C$$

Equation 12

Y[01]=0111111111101111111111111111111011111110011111011111011111111000101
Y[02]= 0011111111100111111111111111111001111110001111001111001111111000000
Y[03]= 0001111111110001111111111111111000111110000111000111000111111000000
Y[04]= 0000111111110000111111111111110000111100000011000011000011111000000
Y[05]= 0000011111110000011111111111100000111000000010000010000011000000
Y[06]= 0000001111110000001111111111110000001100000000000000000000001000000
Y[07]= 0000000111110000000111111111110000000100000000000000000000000000000
Y[08]= 0000000011110000000011111111110000000000000000000000000000000000000
Y[09]= 0000000001110000000001111111110000000000000000000000000000000000000
Y[10]= 0000000000100000000000111111110000000000000000000000000000000000000
Y[11]= 0000000000000000000000011111110000000000000000000000000000000000000
Y[12]= 0000000000000000000000001111110000000000000000000000000000000000000
Y[13]= 0000000000000000000000000111110000000000000000000000000000000000000
Y[14]= 0000000000000000000000000011110000000000000000000000000000000000000
Y[15]= 0000000000000000000000000001110000000000000000000000000000000000000

Equation 13

$$S = 123456789A0123456789ABCDEFFFFF01234567001234501234501234560001011$$

$$X = 0000000000k00000000000000000\,j0000000ih00000g00000f000000edc0b0a$$

Equation 14

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

Equation 15

 Copyright 2004