Bull. Nov. Comp. Center, Comp. Science, 21 (2004), 113–118

Real-time image rotation on the array microprocessor NM6403

Mikhail S. Tarkov

Abstract. An algorithm for the real-time image rotation on the array microprocessor NM6403 is proposed. For implementation of this image transformation on the microprocessor NM6403, decomposition of a rotation matrix is used. This decomposition reduces any pixel rotation to a sequence of unidimensional translations which can be effectively vectorized.

1. Introduction

When an image rotates, the pixel coordinates are transformed by the formula

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = R(\theta) \begin{pmatrix} x \\ y \end{pmatrix}, \qquad R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

Here θ is the rotation angle.

For implementation of this transformation on the array microprocessor NM6403, the following rotation matrix decomposition is used [1]:

$$R(\theta) = \begin{pmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \sin\theta & 1 \end{pmatrix} \begin{pmatrix} 1 & -\tan(\theta/2) \\ 0 & 1 \end{pmatrix}. \tag{1}$$

This decomposition reduces the image rotation to a sequence of unidimensional translations of the image rows along the direction x and translations of the image columns along the direction y. Since all translation values for all pixels of the same row or of the same column are equal to each other, then any translation can be readily vectorized. In this paper, we consider organization of parallel computations for the image rotation on the array microprocessor NM6403.

2. Architectural features of the microprocessor NM6403

The new 64-bit processor NM6403[2] is a high-performance microprocessor with the hardware implemented matrix-by-matrix, matrix-by-vector multiplication, vector addition and other vector operations support. This processor performs some operations over the data vectors for one clock cycle. Each data vector is a 64-bit word of packed integer data.

The kernel of the NM6403 architecture is an operating unit including a vector (array) coprocessor (VCP) and a masking device. The VCP consists of 64 × 64 cells and is capable to realize matrix operations on data with a variable number of components. The dimension of the VCP matrix is defined by programming special registers. The input 64-bit data word consists of $n \le 64$ vector components. Every input vector component is multiplied by the values packed into a 64-bit word; the products are added to a calculated partial sum. These multiplications and summation of values are realized for one step of a vector operation.

The vector arithmetic and logical unit (ALU) implements operations on pairs of vectors packed into 64-bit words. The data come to inputs of the VCP and the ALU through a programmable masking device. The masking device has three vector inputs: two inputs for the data vectors and one input for the mask vector. A vector command can include an operation for masking data inputs.

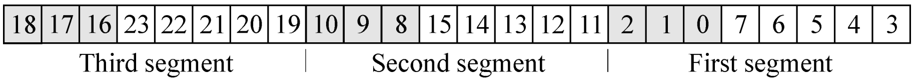
3. The image rotation algorithm on the processor NM6403

Before the rotation, the following translation tables are constructed: the table *Row* contains values of translations along the direction x, the value $Row[i]$, $i = 0, 1, \ldots, i_{\max} - 1$, is equal to the translation value of the i-th row, the table Col contains values of translations along the direction y, the value $\mathrm{Col}[j]$, $j = 0, 1, \ldots, j_{\max} - 1$, is equal to the translation value of the j-th column. The translations evaluated are rounded off. When the translation tables are ready, the image rotation procedure is called. The rotation procedure includes subprograms for translation along the direction x (the row translation) and for translation along the direction y (the column translation).

The row translation is realized by the following procedure. First, a contiguous 8-byte row segment is cyclically shifted to the right by $n < 8$ pixels across multiplication by a suitable permutation matrix loaded into the VCP. The permutation matrix for implementation of the right cyclic

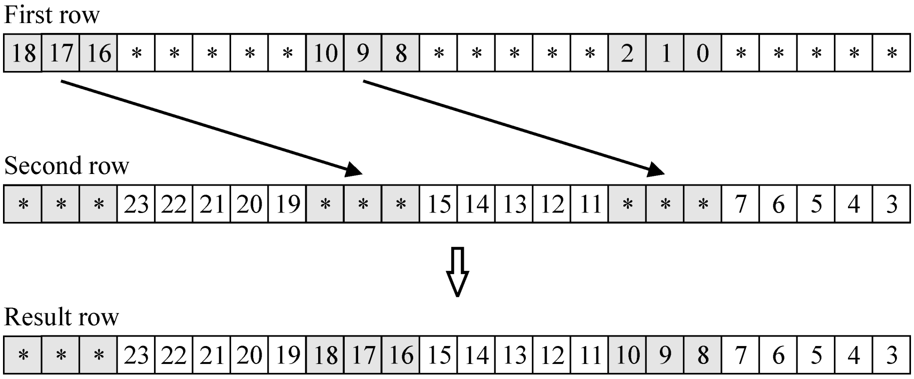
$$
\begin{array}{ccc}
\begin{pmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix} & \Rightarrow &
\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{pmatrix} \\
 & & \Downarrow \\
 & & \begin{pmatrix} x_2 & x_1 & x_0 & x_7 & x_6 & x_5 & x_4 & x_3 \end{pmatrix}
\end{array}
$$

Figure 1. Cyclic shift of 8-pixel row to the right by 3 pixels



18	17	16	23	22	21	20	19	10	9	8	15	14	13	12	11	2	1	0	7	6	5	4	3

Figure 2. Result of the right cyclic shift



First row

18	17	16	*	*	*	*	*	10	9	8	*	*	*	*	*	2	1	0	*	*	*	*	*

Second row

*	*	*	23	22	21	20	19	*	*	*	15	14	13	12	11	*	*	*	7	6	5	4	3

Result row

*	*	*	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3

Figure 3. The row translated to right by 3 bytes

shift by 3 bytes is shown in Figure 1. As a result of this shift, we have, for example, a row consisting of three 8-byte segments and shown in Figure 2.

The row resulted from the right cyclic shift is used for distinguishing two rows on the vector ALU by masking. First, row has 3 bytes shifted to the left (distinguished in Figure 2 by shadowing) and the second row has 5 bytes shifted to the right. Next, the first row is translated to the right by one 64-bit word and added to the second row. As a result, we have the row translated to the right by $n < 8$ pixels (Figure 3).

In Figure 3, the rows realized by a masking row from Figure 2, and the result of their summation are shown. In the resulting row, the left-three bytes (marked with stars in Figure 3) are empty, and the right-three bytes are lost. Analogously, the left row translation by $n < 8$ bytes is implemented.

If $n \geq 8$, then we evaluate the number of 8-byte words $N = \lfloor \frac{n}{8} \rfloor$, $\lfloor x \rfloor$ is an integer part of x, and the number of bytes $n' = n - 8N$, $n' < 8$, for the row translation. Next, we translate the row by n' bytes, as described above, and after that, by N 8-byte words by means of simple copying.

Column translation. Since the vector instructions in the processor NM6403 process 8-byte words only, then to translate a byte column we need to delete bytes from 8-byte words by masking. The column is translated by a given number of bytes by simple copying. The image rotation can be realized in the two ways:

1. Translation along the rows + translation along the columns + translation along the rows;

2. Translation along the columns + translation along the rows + translation along the columns.

The second way corresponds to the permutation x and y in expression (1). We choose the first way, because it has the smallest rotation time. The algorithm is capable of rotating the images both to the positive angles (counterclockwise) and to the negative angles (clockwise).

4. Experiments

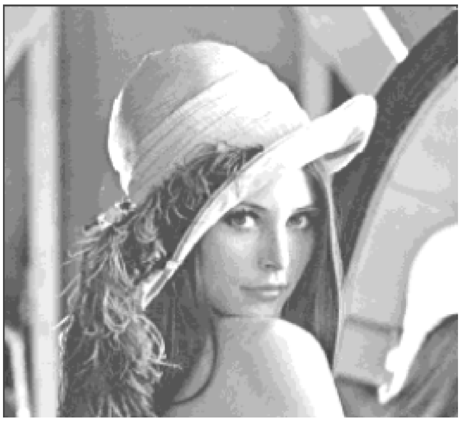


Figure 4. Original image

In the table below, the data for the dependence of the rotation time on the rotation angle value for the image "Lena" (Figure 4) of the size of 240 × 264 pixels is shown. Here the time for creating translation tables is not taken into consideration. This time is in the interval from 24 to 28 milliseconds. The translation tables does not use much memory and can be evaluated beforehand. The data of the table show that the rotation time is not greater than 22 milliseconds and is weakly dependent upon the rotation angle in the range from 4 to 150 degrees. A relatively fast increase of the rotation time in the range from 1 to 4 degrees is due to increase in the number of translated rows and columns.

Dependence of rotation time t (msec) on rotation angle θ (degrees)

θ	1	2	3	4	5	10	15	20	30	45	60	75	90	105	135	150
t	13	16	17	18	18	19	20	20	20	21	21	22	22	22	21	20

In Figures 5–7, the results of three sequential rotation steps of the image shown in Figure 4 are presented for the rotation angle of 30 degrees. Figures 7 and 8 show an increasing of a number of the lost pixels, when the rotation angle increases. These losses are caused by the translation of pixels behind the boundaries of the image under processing. It is possible to avoid losses by expanding the boundaries of the field under processing, but with the increase in the rotation time. Another way to decrease losses is to realize the necessary rotation angle by means of n-multiple rotation of the image by the angle $\frac{\theta}{n}$ with n-multiple increase of the rotation time and an increase in distortions caused by rounding the new pixel coordinates.

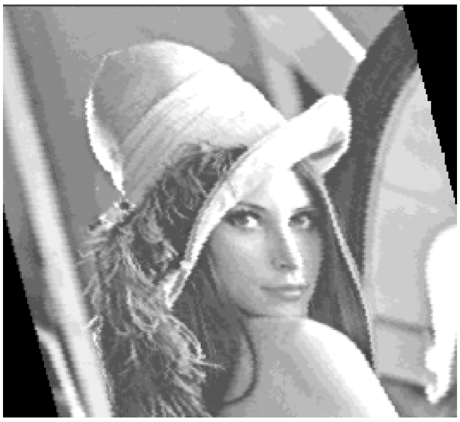


Figure 5. Translation along rows

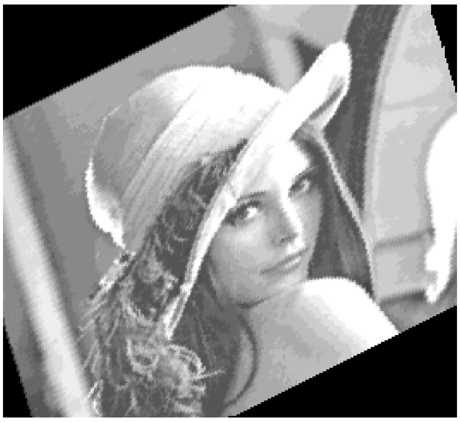


Figure 6. Translation along columns

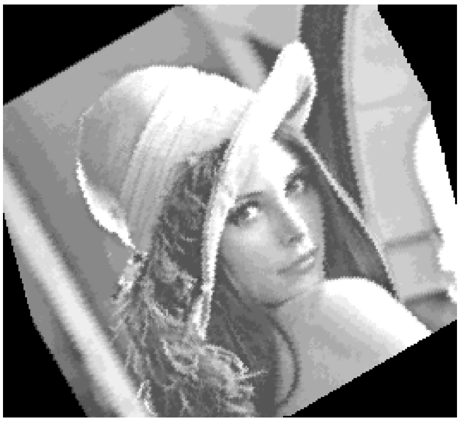


Figure 7. Translation along rows

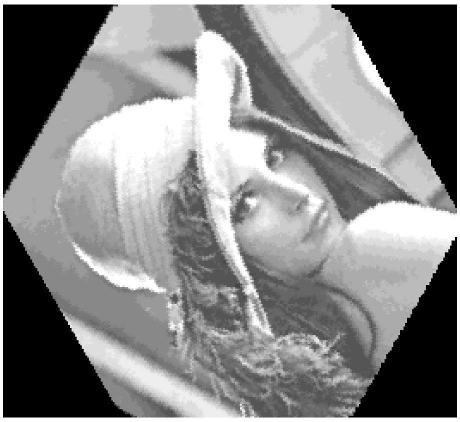


Figure 8. Rotation by 60 degrees

5. About the image rotation on a multiprocessor system

Representation (1) of the rotation matrix as a product of translation matrices (i.e., reduction of any pixel rotation to a sequence of translations) both simplifies the vector realization of the rotation on the processor NM6403 and allows one to implement the image rotation in the parallel mode on a system of processors interconnected to the line structure. For multiprocessor implementation of the image rotation, it is necessary to distribute the image rows uniformly among the processors (as shown in Figure 8 with the dashed lines). It makes possible to execute the row translations along x direction independent of each other.

For execution of translations along y direction, it is necessary to organize the exchange of the column data fragments between the neighboring processors. In Figure 9, a change in boundaries between the neighboring data

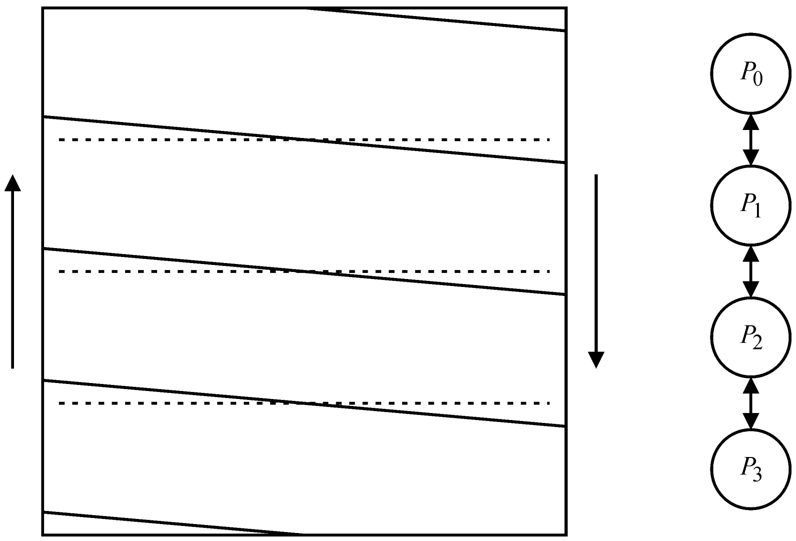


Figure 9. Distribution of image rows among processors

fragments is shown by the sloping lines for the column translations along the direction y, when the image rotates at a negative angle.

6. Conclusion

An algorithm for the real time implementation of an image rotation on the array microprocessor NM6403 is proposed. For this image transformation, the decomposition of the rotation matrix is used. The decomposition reduces the rotation of the image pixels to a sequence of translations of the image rows and columns. Since values of translations for all the pixels of the same row or column are equal to each other, the translation operation can be readily vectorized. Experiments show that the rotation time is weakly dependent upon the rotation angle. Future modifications of the algorithm suggest its implementation on a multiprocessor system. The simplicity of this implementation is also conditioned by the above decomposition of the rotation matrix.

References

[1] Bourennane E., Milan C., Paindavoine M., Bouchoux S. Real time image rotation using dynamic reconfiguration // Real-Time Imaging. – 2002. – Vol. 8. – P. 277–289.

[2] Research Center MODULE. “Processor NeuroMatrix NM6403. Introduction to architecture”. – http://www.module.ru/files/archover.pdf.