Bull. Nov. Comp. Center, Comp. Science, 27 (2008), 133–142
© 2008 NCC Publisher

Mapping a neural network onto a distributed image processing system

Mikhail S. Tarkov

Abstract. Algorithms for mapping a weight matrix of a neural layer onto distributed computer systems with a torus structure are proposed for parallel image processing. It is shown that the choice of the mapping technique depends upon the ratio between the number of neurons and the number of weight coefficients in a neuron (the number of pixels): if this ratio is sufficiently small, the column distribution is more efficient, otherwise the row distribution is more suitable. In particular for the Hopfield network, the row distribution is always more efficient than the column distribution.

1. Introduction

The neurocomputer technology of image processing is one of the promising directions in the solution of the problem of information systems performance increase [1, 2]. A possibility to solve different image processing problems by a common algorithmic basis is an advantage of such a technology. In addition, it is necessary to mention that most of image processing problems have a "natural" parallelism of computations because of features of the digital image representation as a two- or a three-dimensional array. A massive computation parallelism is the main property of neurocomputers. Currently, for this reason, image processing (image enhancement and segmentation, object registration, etc.) is a conventional sphere of the neurocomputer application.

An element of a neural network (a neuron) implements the transformation $y = f((w, x))$, where (w, x) is the inner product of an input signal vector x by a vector w of the neuron weight coefficients, f is a nonlinear activation function. Every weight coefficient corresponds to one input (a synapse) of a neuron. A set of non-connected neurons processing the same input vector forms a neural layer. The layer functioning is described by the formula

$$Y = f(Wx), \tag{1}$$

where W is a matrix with neuron layer weight vectors as rows, Y is a vector of output signals. The major part of neural network computations consists in multiplication operations of the weight matrix of its first layer by the vector of the image pixel intensities. The result is a large-scale number of computations and the necessity to use a highly parallel computer system.

A distributed computer system (CS) (a computer system with distributed memory) is a set of elementary computers (EC) connected by a network controlled by computers. The structure of the distributed CS is described by a graph with nodes corresponding to ECs and edges corresponding to computer interconnections. In a modern supercomputer with distributed memory, the three-dimensional tori are most often used as computer interconnection graphs [3, 4].

A k-dimensional torus $E_k(p_1, p_2, \dots, p_k)$ is an undirected graph, in which the nodes can be labeled as k-tuples $(i_1, i_2, \dots, i_k)$, $0 \le i_j \le p_j - 1$. Every node $(i_1, i_2, \dots, i_k)$ of the graph has two neighbors in each dimension of the torus. Its left neighbor in the jth dimension is $(i_1, \dots, (i_j - 1) \bmod p_j, \dots, p_k)$, and its right neighbor in this dimension is $(i_1, \dots, (i_j + 1) \bmod p_j, \dots, p_k)$ [8].

A preliminary stage of image processing is filtration [5], which is often described by the convolution $H * F$ of the intensity function $f(i,j)$ with a set of the filter weight coefficients

$$H = (h(k,l,i,j)),\ k,l = -M,\dots,M,\ i = 1,\dots,N_1,\ j = 1,\dots,N_2,$$

$$g(i,j) = \sum_{k=-M}^{M} \sum_{l=-M}^{M} h(k,l,i,j) f(k+i, l+j). \tag{2}$$

In transformation (2), a square window of the size $(2M+1)\times(2M+1)$, $M \ll \min(N1, N2)$ is widely used. So, computations in a pixel are associated with processing of a small neighborhood of this pixel, i.e., filtration algorithms (2) are local. In a limit, for a maximal parallelization of computations, every image pixel is in one-to-one correspondence with a processor. Such a style of parallelism is known as geometric.

It follows from the locality of transformation (2) that:

- the neighborhood of EC in a system must correspond to that of an image pixel;
- mapping a data segment processed by algorithm (2) onto ECs must preserve the neighborhood of this segment;
- the mesh shown in Figure 1 can be efficiently used as a graph of a parallel system of processes of filtration of images.

This mesh has a good mapping onto a torus structure of the computer system. From now on, we assume components of the image x to be equally distributed over the system computers so that the neighboring pixels are always mapped onto the same EC or onto the adjacent ECs of the mesh.

The way of organizing intercomputer communications for the parallel implementation of operation (1) is determined by distribution of entries of the matrix W over computers. Currently, many techniques for mapping neural networks onto parallel computer systems are being developed

[6, 7], however these techniques do not take into account features of the above geometric parallelism of algorithms (2) for preliminary image processing. The solution to this problem is an objective of this paper.

Let us consider two ways of a neural layer mapping onto a distributed CS structure:

1. Distribution of the matrix W rows over computers (parallelism of neurons);
2. Distribution of the matrix W columns over computers (parallelism of synapses).

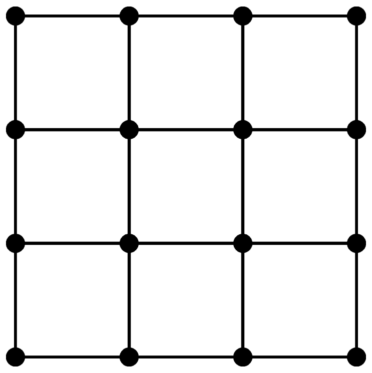
Figure 1. Graph of parallel program for preliminary (low-level) image processing ("mesh")

2. Mapping a neural layer by distribution of weight matrix rows over computers

Let us consider organization of intercomputer communications for distribution of the weight matrix W rows over computers. Since every matrix row corresponds to one neuron of the network, the distribution of rows describes the mapping of neurons onto computers. To do computations for all neurons by expression (1), it is necessary to gather components of the image x in every computer, i.e., to implement the translational-cyclic exchange ("all to all") of components of the vector x. As a result, multiplication of the weight matrix W by this vector can be executed in parallel in all computers (the number of simultaneous multiplications is equal to the number of computers).

The translational-cyclic exchange in a k-dimensional torus is reduced to a number of translational-cyclic exchanges in the torus rings, i.e. in the structures described by cyclic subgroups. In every ring, exchanges are executed as shown in Figure 2. Every computer M_j, $j = 0, \dots, p_i - 1$, sends its array of pixels to the computer $M_{(j-1) \bmod p_i}$, $j = 0, \dots, p_i - 1$, and receives an array from the computer $M_{(j+1) \bmod p_i}$, $j = 0, \dots, p_i - 1$. We assume all intercomputer connections in the same ring to work simultaneously. The described actions are iterated until all computers in the ring receive all pixels distributed over them. The exchanges are implemented in parallel for all rings of the dimension i and sequentially in order of dimensions $i = 1, \dots, k$. For example, for a two-dimensional torus (Figure 3), exchanges can be executed in parallel for horizontal rings and then in parallel for all vertical rings.

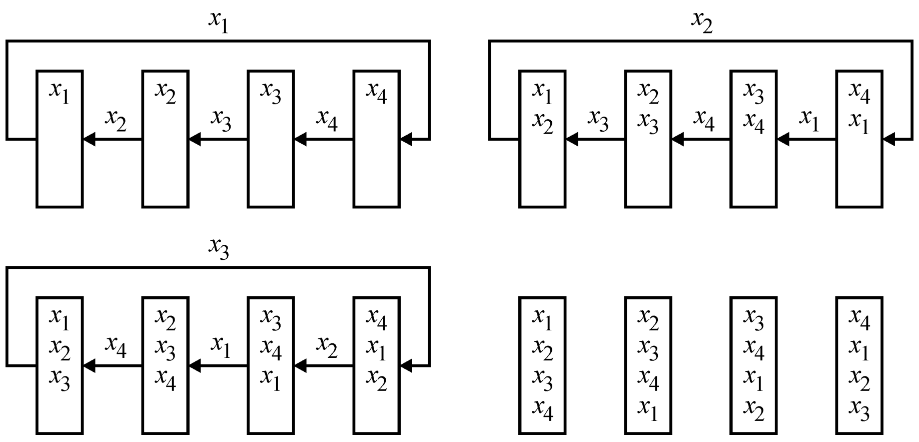

Figure 2. Exchanges in a ring

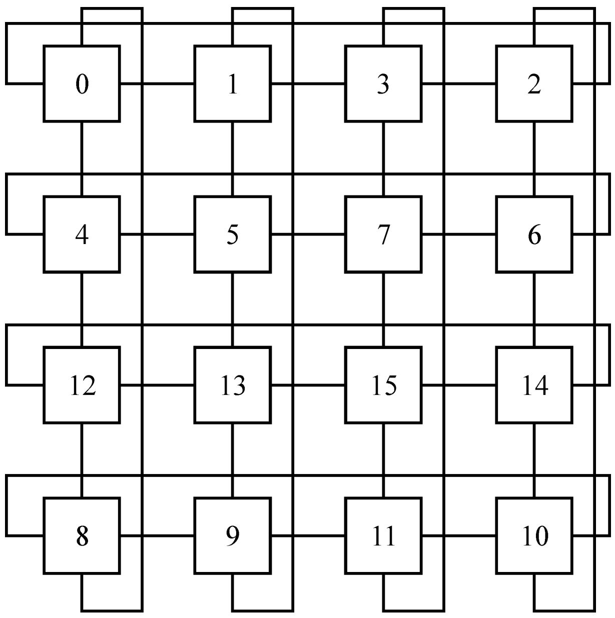

Figure 3. Example of a two-dimensional torus

Let n be the number of pixels in an image, m be the number of neurons in a layer, p be the number of computers in the system, t_{o} be the time of one arithmetical operation, t_{w} be the time for one data transmission, p_i be the order of the ith generator of the torus. We suppose n to be a multiple of p.

Statement 1. *The time of the translational-cyclic data exchange does not depend upon the torus dimension and is equal to*

$$T_{\mathrm{e}} = n\Big(1 - \frac{1}{p}\Big)t_{\mathrm{w}}. \tag{3}$$

Proof. It follows from the proposed algorithm of a translational-cyclic exchange in a k-dimensional torus that after completion of the l-th step of the exchange, $l \in \{1, \ldots, k\}$, every computer has $\frac{n}{p}\prod_{i=1}^{l} p_i$ data elements and, respectively, after k steps it has n elements because $\prod_{i=1}^{k} p_i = p$.

Then the time of l steps implementation is equal to

$$T_{\mathrm{e}}^{(l)} = \frac{n}{p}\sum_{i=1}^{l}(p_i - 1)\prod_{j=1}^{i-1} p_j t_w. \tag{4}$$

Transforming formula (4) for $k = l$, we have

$$\begin{aligned} T_{\mathrm{e}}^{(k)} &= \frac{n}{p}\sum_{i=1}^{k}(p_i - 1)\prod_{j=1}^{i-1} p_j t_w = \frac{n}{p}\Bigg[\sum_{i=1}^{k}\prod_{j=1}^{i} p_j - \sum_{i=1}^{k}\prod_{j=1}^{i-1} p_j\Bigg] t_w \\ &= \frac{n}{p}\Bigg[\prod_{j=1}^{k} p_j - \prod_{j=1}^{0} p_j\Bigg] t_w = \frac{n}{p}(p-1)t_w. \qquad \square \end{aligned}$$

Because the number of the image pixels $n \gg 1$, we neglect the time for computation of the activation function f. Hence the time of the sequential execution of multiplication W on x in the layer is

$$T_{\mathrm{seq}} \approx 2mnt_{\mathrm{o}}\,. \tag{5}$$

Let the number of neurons m be a multiple of p. We consider a case of a uniform neuron distribution over computers, i.e., every computer has $\frac{m}{p}$ neurons. For a torus with p computers, the time of parallel execution of Wx is

$$T_{\mathrm{r}} = \frac{2mn}{p}t_{\mathrm{o}} + n\Big(1 - \frac{1}{p}\Big)t_{\mathrm{w}}. \tag{6}$$

For m to be a multiple of p and a uniform distribution of neurons over computers, the speedup is

$$S_{\rm r} = \frac{T_{\rm seq}}{T_{\rm r}} \approx \frac{2mnt_{\rm o}}{\frac{2mnt_{\rm o}}{p} + n(1 - \frac{1}{p})t_{\rm w}} = p\frac{1}{1 + \frac{(p-1)t_{\rm w}}{2mt_{\rm o}}}.$$

So, we arrive at

Statement 2. *For the distribution of rows of the matrix W over computers (parallelism of neurons), the speedup does not depend upon the number of pixels and is equal to*

$$S_{\rm r} \approx p\frac{1}{1 + \frac{(p-1)t_{\rm w}}{2mt_{\rm o}}}. \tag{7}$$

3. Mapping a neural layer by distribution of weight matrix columns over computers

For the distribution of the matrix W columns over computers, it is possible to implement multiplications of the matrix coefficients by suitable components of the image vector x without intercomputer data exchanges. Then it is necessary to sum up the products obtained. Partial sums are evaluated in p computers in parallel. After that, for evaluation of complete sums, it is necessary to realize exchanges across a binary tree of intercomputer connections mapped onto a graph of the computer system.

The number of complete sums is equal to the number m of neurons. This number can be arbitrary and, in particular, a multiple to the number of computers. To balance a load of all computers, it is necessary to provide a maximal parallelism of the summation operations. This requirement is satisfied with the summation scheme known as "butterfly" (Figure 4).

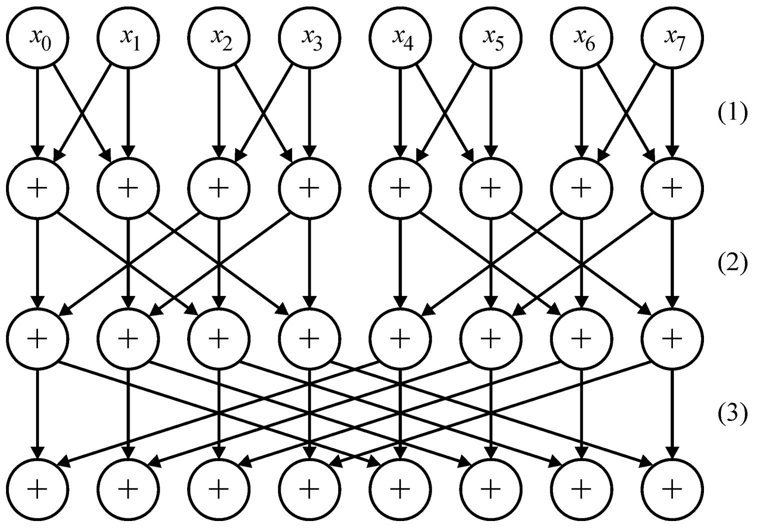

Figure 4

The butterfly scheme provides a simultaneous evaluation of several sums. If the number of terms is sufficient, the number of simultaneously executed operations is equal to the number of computers.

Because not all the butterfly processes work simultaneously, it is necessary to join processes that cannot be implemented in parallel. In Figure 4, the joined processes lie along vertical lines. As a result of joining, we have a hypercube (Figure 5). Here the numbers in brackets show the steps of interactions between hypercube nodes.

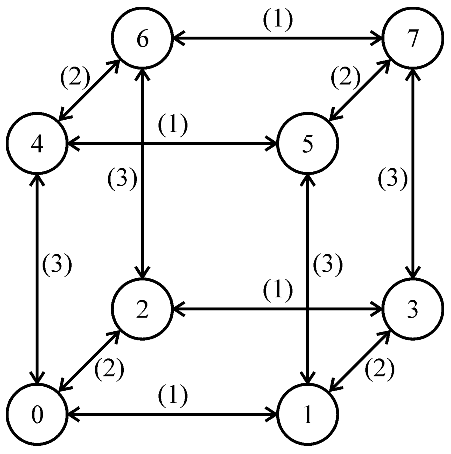

Figure 5

Bidirectional arrows in Figure 5 show duplex channels, transmitting data between hypercube nodes. Such a channel provides two opposite simultaneous message transmissions. So, at every butterfly stage (stages (1)–(3) in Figure 4), it is possible to make simultaneous summations of entries for two different arrays.

The hypercube edges are mapped onto a torus with dilations [8, 9]. It is shown in [8] that a d-dimensional hypercube can be mapped onto a torus $E_k(2^{d_1}, \ldots, 2^{d_k})$, $d_i > 1$, $i = 1, \ldots, k$, with $\sum_{i=1}^{k} d_i = d$ and with an average dilation

$$D = \frac{1}{d} \sum_{i=1}^{k} (3 \cdot 2^{d_i - 2} - 1). \tag{8}$$

The dependence of the average dilation D on the number of computers $p = 2^d$ for mapping a d-dimensional hypercube onto a torus is shown in Table 1. The dilation was calculated for the case of $d_i = d/k$, $i = 1, \ldots, k$.

Table 1. Average dilation of hypercube edges mapped onto a torus

Mapping onto 2D torus			Mapping onto 2D torus			Mapping onto 3D torus		
d	p	D	d	p	D	d	p	D
4	16	1	12	4096	7.833	6	64	1
6	64	1.667	14	16384	13.571	9	512	1.667
8	256	2.75	16	65536	23.875	12	4096	2.75
10	1024	4.6						

Let us evaluate the computation speedup for the case of distribution of the matrix W columns over computers. The time for multiplication of weights with appropriate pixel values with p computers is

$$T_{\mathrm{mult}} = \frac{mn}{p} t_{\mathrm{o}}.$$

The time for summation of products with p computers is

$$T_{\mathrm{add}} = m\Big(\frac{n}{p} - 1\Big) t_{\mathrm{o}}.$$

So, the time of the partial sums evaluation with p computers is

$$T_{\mathrm{ps}} = T_{\mathrm{mult}} + T_{\mathrm{add}} = m\Big(\frac{2n}{p} - 1\Big) t_{\mathrm{o}}. \tag{9}$$

After that, the complete sums are evaluated in $\log_2 p$ steps for all m neurons of the hypercube mapped onto the torus. At every step, no more than two summation operations can be executed. Because the number of terms is divided by 2 from step to step, the time of the complete sums evaluation for m neurons is

$$T_{\mathrm{cs}} = (Dt_{\mathrm{w}} + t_{\mathrm{o}}) \sum_{i=1}^{\log_2 p} \max\Big(1, \frac{m}{2^i}\Big). \tag{10}$$

So, the complete time of implementation of the parallel neural layer is

$$T_{\mathrm{c}} = T_{\mathrm{ps}} + T_{\mathrm{cs}} = m\Big(\frac{2n}{p} - 1\Big) t_{\mathrm{o}} + (Dt_{\mathrm{w}} + t_{\mathrm{o}}) \sum_{i=1}^{\log_2 p} \max\Big(1, \frac{m}{2^i}\Big). \tag{11}$$

For m to be multiple of p, we have from (10) that

$$T_{\mathrm{cs}} = m(Dt_{\mathrm{w}} + t_{\mathrm{o}}) \sum_{i=1}^{\log_2 p} \frac{1}{2^i} = m\frac{p-1}{p}(Dt_{\mathrm{w}} + t_{\mathrm{o}}). \tag{12}$$

Therefore, we derive from expressions (11), (12):

$$\begin{aligned} T_{\mathrm{c}} &= m\Big(\frac{2n}{p} - 1\Big) t_{\mathrm{o}} + m\frac{p-1}{p}(Dt_{\mathrm{w}} + t_{\mathrm{o}}) \\ &= \frac{m}{p}[(2n-1)t_{\mathrm{o}} + (p-1)Dt_{\mathrm{w}}]. \end{aligned} \tag{13}$$

Finally, the speedup in this case is

$$S_{\mathrm{c}} = \frac{T_{\mathrm{seq}}}{T_{\mathrm{c}}} \approx \frac{2mnt_{\mathrm{o}}}{\frac{m}{p}[(2n-1)t_{\mathrm{o}} + (p-1)Dt_{\mathrm{w}}]} = p\frac{1}{\frac{2n-1}{2n} + \frac{(p-1)Dt_{\mathrm{w}}}{2nt_{\mathrm{o}}}}, \tag{14}$$

and we arrive at

Statement 3. *For the column distribution of the matrix W over computers* (*the synapse parallelism*), *the speedup does not depend upon the number of neurons in the layer and is equal to*

$$S_\mathrm{c} \approx p \frac{1}{1 + \frac{(p-1)Dt_\mathrm{w}}{2nt_\mathrm{o}}}. \tag{15}$$

Comparing (7) and (15), we conclude

Statement 4. *If $m > n/D$, then $S_\mathrm{r} > S_\mathrm{c}$, otherwise $S_r \le S_c$. In other words, if the number m of neurons exceeds the number n of synapses* (*pixels*) *divided by the average dilation D of mapping the hypercube edges onto a torus, then the row distribution of the matrix W* (*parallelism of neurons*) *is more efficient than its column distribution* (*parallelism of synapses*), *and vice versa.*

To evaluate the speedup, we use parameters of the supercomputer Cray T3E [2]: the processor performance of 1200 Mflops and the channel communication capacity of 480 Mb/s. We assume any data item to have 4 bytes. Then $t_\mathrm{o} = \frac{1}{1.2 \cdot 10^9} \approx 0.83 \cdot 10^{-9}$ s and $t_\mathrm{w} = \frac{4}{480 \cdot 10^6} \approx 8.3 \cdot 10^{-9}$ s. In Table 2, the speedup coefficient S_r is compared with S_c for $p = 1024$ for different values of m. It is seen that the neuron parallelism is more efficient for a large number of neurons in a layer ($m \ge 16384$) and the synapse parallelism is more suitable for smaller values ($m \le 8192$).

Table 2. The speedup coefficient S_r versus $S_\mathrm{c} = 753$

m	1024	2048	4096	8192	16384	32768	65536
S_r	171	293	455	630	780	886	950

4. Mapping the Hopfield network

The Hopfield neural network is a one-layer network with feedback (Figure 6). Its functioning is described by the recurrent expression

$$x^{k+1} = f(Wx^k). \tag{16}$$

The suitable weight matrix is a square matrix, i.e., the number of neurons is equal to the number of neuron synapses (or to the number of image pixels).

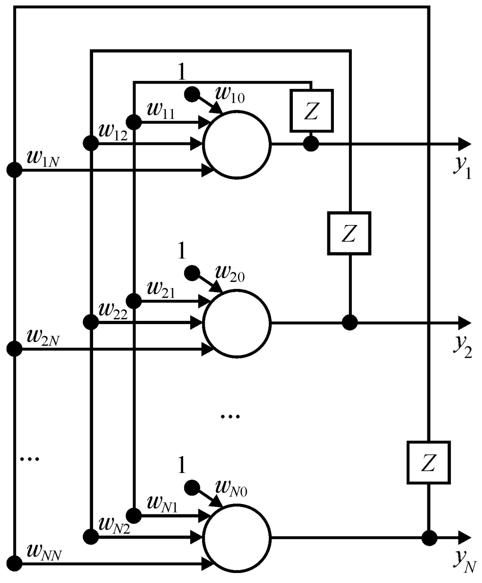

Figure 6

With regard to this restriction, we have from (7) that

$$S_{\mathrm{r}} = p\frac{1}{1+\frac{(p-1)t_{\mathrm{w}}}{2nt_{\mathrm{o}}}}. \tag{17}$$

Comparing (15) and (17) and taking into account the fact that $D \geq 1$, we conclude that $S_{\mathrm{r}} \geq S_{\mathrm{c}}$ regardless parameters of an image and of the computer system, i.e., the neuron parallelism is always more efficient than the synapse parallelism for mapping the Hopfield neural networks onto a torus.

References

[1] Egmont-Petersen M., de Ridder D., Handels H. Image processing with neural networks — a review // Pattern Recognition. — 2002. — Vol. 35. — P. 2279–2301.

[2] Ghennam S., Benmahammed K. Image restoration using neural networks // Proc. of IWANN 2001. — Springer, 2001. — P. 227–234. — (LNCS; 2085).

[3] Cray T3E. — http://www.cray.com/products/systems/crayt3e/1200e.html.

[4] Yu H., Chung I-H., Moreira J. Topology mapping for Blue Gene/L supercomputer // Proc. of the ACM/IEEE SC2006 Conf. on High Performance Networking and Computing, November 11–17, 2006, Tampa, FL, USA. — ACM Press, 2006. — P. 52–64.

[5] Gonzalez R.C., Woods R.E. Digital Image Processing. — Prentice Hall, 2002.

[6] Sundararajan N., Saratchandran P. Parallel architectures for artificial neural networks. Paradigms and implementations. — IEEE Computer Society, 1998.

[7] Ayoubi R.A., Bayoumi M.A. Efficient mapping algorithm of multilayer neural network on torus architecture // IEEE Trans. on Parallel and Distributed Systems. — 2003. — Vol. 14, No. 9. — P. 932–943.

[8] Gonzalez A., Valero-Garcia M., Diaz de Cerio L. Executing algorithms with hypercube topology on torus multicomputers // IEEE Trans. on Parallel and Distributed Systems. — 1995. — Vol. 6, No. 8. — P. 803–814.

[9] Tarkov M.S., Mun Y., Choi J., Choi H.-I. Mapping adaptive fuzzy Kohonen clustering network onto distributed image processing system // Parallel Computing. — 2002. — Vol. 28, No. 9. — P. 1239–1256.