ELSEVIER

Parallel Computing 28 (2002) 1611–1633

PARALLEL COMPUTING

www.elsevier.com/locate/parco

An efficient parallel algorithm for LISSOM neural network

Li-Chiu Chang, Fi-John Chang *

Department of Bioenvironmental Systems Engineering, National Taiwan University, Taipei 10617, Taiwan, ROC

Received 24 October 2001; received in revised form 6 August 2002; accepted 28 August 2002

Abstract

We present a parallel algorithm for laterally interconnected synergetically self-organizing map (LISSOM) neural network, a self-organizing map with lateral excitatory and inhibitory connections, to enhance the computational efficiency. A general strategy of balancing workload for different sizes of LISSOM networks on parallel computers is described. The parallel algorithm of LISSOM is implemented on IBM SP2 and PC cluster. The results demonstrate the efficiency of this LISSOM parallel algorithm in processing networks with large sizes. Parallel implementation for different input dimensions in networks of the same size (i.e., 20×20) show that the speedup can sustain at a high level. We demonstrate the LISSOM can be applied to complex problems through the parallel algorithm devised in this study.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Laterally interconnected synergetically self-organizing map; Parallel neural networks; Parallel implementation, Balancing load

1. Introduction

Parallel processing has a tremendous impact in many areas of computer applications. The computing speed of any current single-processor computer has not satisfied the needs of the continually growing data processes in science, engineering and business. Parallel processing is one of the approaches that would make feasible those applications with a large number of variables or iterations. The concept of parallel processing is to partition the task of a sequential processing to multiple processors

* Corresponding author.
E-mail address: changfj@ccms.ntu.edu.tw (F.-J. Chang).

0167-8191/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0167-8191(02)00166-7

in order to reduce the execution time. However, partitioning the task to different processors might be not effective due to additional computations to synchronize, communication and load imbalance among processors. As a result, several issues need to be addressed in order to effectively carry out the extra computations in parallel processing. These issues include construction of efficient parallel algorithms, parallel architectures, parallel programming languages and performance analysis.

ANNs are inspired by neurobiology to perform brain-like computations. Due to their immense abilities of modeling complex nonlinear systems, ANNs have become a very promising alternative to traditional methods for solving complex problems in many fields, including robotics, image processing, speech recognition, space exploration, and hydroinformatics [3,4,6,8,9]. An ANN derives its power through the massively interconnected computational neurons (units). Each neuron receives input from other neurons, processes a weighted summation, applies an activation function to the sum, and sends the results to other neurons in the network. Although the operations and/or computations in each unit and the whole network are generally very simple, the total computational load could be very large, especially in the training phase. For applications with large training data, it is not uncommon to have a training time on the order of days to weeks [21]. This has been the main problem for efficient use of ANNs in real-world applications.

This problem can be overcome either by devising faster learning algorithms or by implementing the existing algorithms on parallel computers. Given the parallel nature of neural network, many researches have endeavored to parallelize different neural networks on various computer systems [2,7,13,16,22,23]. However, there are still several bottlenecks for mapping neural networks onto parallel systems. These challenging problems include strong mutuality among neurons, hardware complexity, high communication overhead and load imbalance. According to Amdahl's law, the improvement of overall system performance attributed to the speeding up of one part of the system is limited by the fraction of the job that is not speedup [21]. The main purpose of our study is to develop a general strategy of balancing load for different sizes of laterally interconnected synergetically self-organizing map (LISSOM) implemented on parallel computers. Section 2 provides a detailed description of the LISSOM. In Section 3 we present the algorithms of balancing load for different sizes of LISSOM networks on parallel processors. The implementations for parallel-LISSOM on IBM SP2 and PC cluster are described in Section 4. A brief conclusion is presented in Section 5.

1.1. Message passing interface

The message passing interface (MPI) is a standard library for massively parallel machines or workstation clusters to write message-passing programs in C, FORTRAN, and C++ languages. It provides an application programming and efficient communication interface that can be implemented in a heterogeneous environment. The main advantage of using MPI is to establish a portable and ease-to-use message-passing standard interface [14]. It has been widely used to design the parallel programs for solving significant engineering and scientific problems [5]. In this study, the

proposed parallel algorithm is based on the communication and message-passing model of MPI to implement parallel processing on IBM SP2 and PC cluster. The main features of IBM SP2 and PC cluster used in this study are shown in Appendix A.

1.2. Performance analysis—speedup and efficiency

A sequential program is usually evaluated in terms of its execution time, which can be expressed as a function of the size of its input. For a parallel algorithm, important factors for evaluating its performance include not only its execution time and input size but also the number of processors and the architecture of parallel computers. In addition, speedup and efficiency are important measures of the quality of a parallel algorithm.

Speedup measures the relative benefit of solving a computational problem in parallel and is defined as the ratio of the sequential execution time to the parallel execution time on p processors required for solving identical problems. Mathematically, it is defined by

$$S = \frac{T_s}{T_p} \tag{1}$$

where S is the speedup; T_s, the execution time of the best sequential program; and T_p, the execution time of parallel processing by using p processors.

The efficiency of parallel computation is defined as the ratio of speedup to the number of processors. It is a measure of the fraction of time for which a processor is usefully employed in a parallel computation. Thus,

$$E = \frac{S}{p} \tag{2}$$

where E is the efficiency; and p, the number of processors.

In an ideal parallel system, speedup is equal to p; the efficiency is equal to one. Practically, extra computations required for synchronization, communication and the load distribution imbalance among parallel units result in less than ideal speedup and efficiency.

2. Laterally interconnection synergetic self-organizing map

In this section we present the LISSOM neural network model for the self-organization of both lateral and afferent connections (as shown in Fig. 1). Its algorithms are fundamentally based on self-organizing map (SOM), which will be briefly reviewed below.

2.1. Self-organizing feature map

The SOM is one of artificial neural networks introduced by Kohonen [11] in the early 1980s. Since then, SOM has been extensively explored and applied to a variety

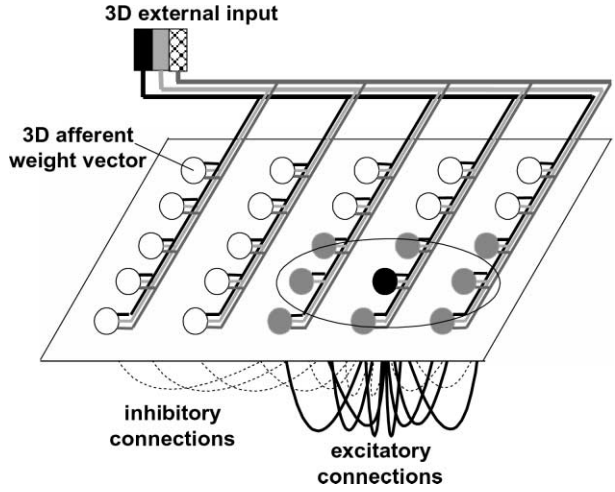

Fig. 1. The architecture of LISSOM.

of problems encountered in pattern recognition, signal processing, diagnostic and control systems [10,12]. In SOM, inputs close to each other are mapped to output neurons that are spatially close together. This is known as topology preservation and is an important aspect of feature mapping in the human brain. The architecture of Kohonen's SOM network usually consists of a two-dimensional array of neurons with each neuron fully connected to all input nodes while there is no lateral connection.

Fig. 2 shows the basic architecture of the SOM network. For simplicity, we assume that the number of a two-dimensional array of neurons is m. The external n-dimensional inputs to SOM shown in Fig. 2 are denoted by

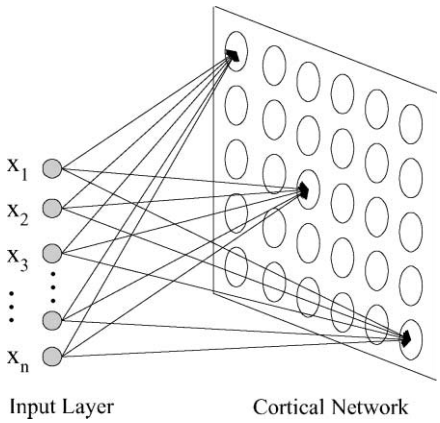

Fig. 2. The architecture of SOM.

$$x = [x_1, x_2, \ldots, x_n]^{\mathrm{T}} \tag{3}$$

and the synaptic weight vector connecting input units to the output neuron i is denoted by

$$w_i = [w_{i1}, w_{i2}, \ldots, w_{in}]^{\mathrm{T}} \quad i = 1, 2, \ldots, m \tag{4}$$

During an adaptation phase, SOM only adjusts a winning neuron and the neighborhood neuron around it. Thus, the winning neuron c is defined as the one whose weight vector is the closest to the input x, i.e.,

$$c(x) = \min_{\forall i} \|x - w_i\| \quad i = 1, 2, \ldots, m \tag{5}$$

The learning process takes place subsequent to this competition. In the learning phase, each wining neuron and its surrounding neurons gradually change the value of their reference vectors to match the input vector it has won. The learning rule is given by

$$w_i(k+1) = \begin{cases} w_i(k) + \mu(k)[x(k) - w_i(k)] & i \in N_c(k) \\ w_i(k) & i \notin N_c(k) \end{cases} \tag{6}$$

where N_c is the topological neighborhood around the winning neuron c; k, the number of iterations; and $0 < \mu(k) < 1$, the learning rate. Note that N_c is considered to decrease monotonically with time as shown in Fig. 3.

Despite their simplicity, SOM remains very demanding in terms of computing time, because the processes involve a very large number of input and neurons. The neuron's weight vectors may also have dimensions numbered in the thousands. Many studies have worked on implementing SOM on different parallel computers to gain better application performances [7].

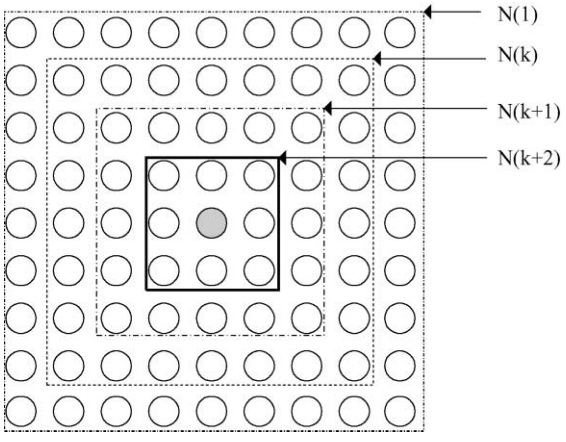

Fig. 3. Topological neighborhood around the winning neuron.

2.2. The algorithm of LISSOM

The LISSOM model was presented by Sirosh and Miikkulainen [18,19] to account for cortical self-organization in which both afferent and lateral connections can organize simultaneously from the same external input in a mutually supportive manner. Since then, there have been extensive studies of LISSOM, leading to several variations and applications [1,20]. LISSOM has a two-dimensional array of neurons with a set of afferent connections from the external input and a set of recurrent lateral connections between neurons (Fig. 1). Both these connections are self-organizing simultaneously in a cooperative fashion. Each neuron generates an initial activity as a weighted sum of input vector in its afferent connections. Then, the effects of lateral connections between neurons concentrate the response to the center of the map. After the activity settles into a stable pattern, all afferent and lateral connection weights of each neuron are modified, and the activation function is adapted through the Hebbian rule. The above process is repeated for every input vector drawn from the input space. After the weak connections are killed, large number of lateral connections die off. As the self-organization progresses, the neuronal responses become more nonlinear, selective, and sensitive. The algorithm of LISSOM is described in detail as follows.

2.2.1. Response generation

Each neuron has a set of afferent connections and lateral connections. The afferent connections are all excitatory, while the lateral connections contain reciprocal excitatory and inhibitory connections with other neurons. Usually lateral excitatory connections are short ranged, and lateral inhibitory connections reach long distances. The response of each neuron is defined as a sigmoid function of the scalar product of input vector and afferent weight vector.

$$\eta_{ij} = \sigma\left(\sum_h \mu_{ij,h}\xi_h\right) \tag{7}$$

where η_{ij} is the response of neuron (i,j); μ_{ij}, a afferent weight vector of neuron (i,j); h, the dimension of the input vector; and ξ, an external input vector. The sigmoid function is a piecewise linear function (Fig. 4):

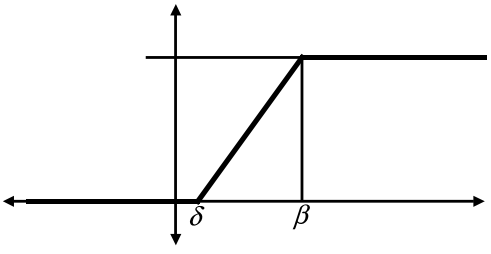

Fig. 4. The activation function of each neuron.

$$\sigma(x) = \begin{cases} 0 & x \leqslant \delta \\ (x-\delta)/(\beta-\delta) & \delta < x < \beta \\ 1 & x \geqslant \beta \end{cases} \tag{8}$$

Through lateral reciprocal connections, the initial response is modified:

$$\eta_{ij} = \sigma\left(\sum_h \mu_{ij,h}\xi_h + \gamma_e \sum_{k,l} E_{ij,kl}\eta_{kl}(t-1) - \gamma_i \sum_{k,l} I_{ij,kl}\eta_{kl}(t-1) \right) \tag{9}$$

where $E_{ij,kl}, I_{ij,kl}$ are the excitatory and inhibitory connection weight on the connection from neuron (k,l) to neuron (i,j) respectively, $\eta_{kl}(t-1)$ is the response of neuron (k,l) during the previous step, and γ_e, γ_i are scaling factors for determining the strength of the excitatory and inhibatory weights.

2.2.2. Weight adaptation

The lateral weights are modified through the Hebbian learning rule that constrains the sum of excitatory and inhibitory weight to be constants. Through successive iterations of Eq. (9), the response settles into a stable pattern. The lateral weights of neuron (i,j) are modified according to:

$$\Gamma_{ij,kl}(t+1) = \frac{\Gamma_{ij,kl}(t) + \alpha_L \eta_{ij}\eta_{kl}}{\sum_{kl}(\Gamma_{ij,kl}(t) + \alpha_L \eta_{ij}\eta_{kl})} \tag{10}$$

Similarly, the afferent weights are modified according to:

$$\mu_{ij,h}(t+1) = \frac{\mu_{ij,h}(t) + \alpha\eta_{ij}\xi_h}{\sqrt{\sum_h \left[\mu_{ij,h}(t) + \alpha\eta_{ij}\xi_h\right]^2}} \tag{11}$$

where the sum of squares of the afferent weights is kept constant, $\Gamma_{ij,kl}$ substitutes for $E_{ij,kl}$ or $I_{ij,kl}$ of Eq. (9), α_L is the learning rate for lateral connections, and α is the learning rate for afferent weights.

2.2.3. Modification of activation functions

The sigmoid function (Eq. (8) and Fig. 4) affects the self-organizing process in two ways. First, as the lower threshold δ increases, neurons become selective to smaller and smaller areas of the input space. Second, if β is close to δ, the neuron is more nonlinear. Small differences in its input can thus produce large changes in its ouput. When β decreases, the map amplifies smaller differences in the activity pattern. Therefore, neurons can be made more selective and sensitive by increasing δ and decreasing β, resulting in smaller and more refined activity bubbles.

In LISSOM, δ is increased and β decreased in proportion to the response of the neuron at each input vector, up to predetermined maximum and minimum limits:

$$\delta_{ij}(t+1) = \min(\delta_{ij}(t) + \alpha_\delta \eta_{ij}, \delta_{\max}) \tag{12}$$

$$\beta_{ij}(t+1) = \max(\beta_{ij}(t) - \alpha_\beta \eta_{ij}, \beta_{\min}) \tag{13}$$

Consequently, the neurons grow nonlinear faster at those parts of the map that have more activity; neurons that work harder mature faster.

2.2.4. Connection death

During critical periods of cortical development, large number of connections die off [15]. This process depends on the strength of activity to determine which connections survive. For the same reason, lateral connections in the LISSOM model can survive only if its weight vectors are significant.

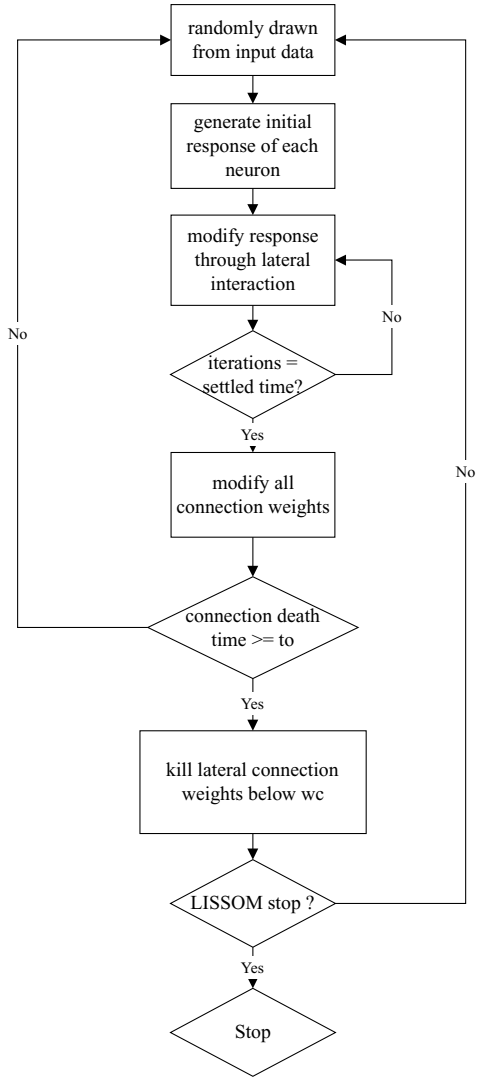

Fig. 5. Flow diagram of the sequential LISSOM on training module.

In LISSOM, a parameter t_0 determines the onset of connection death. During connection death, lateral connections with strengths below a specific threshold w_c are killed. From t_0 on, more weak connections are eliminated at intervals t_c during the self-organizing process. A flow diagram of training the sequential LISSOM is shown in Fig. 5.

3. Parallel algorithm design for LISSOM

A major issue determining the efficiency of a parallel algorithm resides in the way in which the computational load is divided between the multiple processors. Load imbalance among working processors adversely affects its performance. It is therefore an important task to partition lattice of neurons and to distribute them to processors. The weight adjustment of each neuron in LISSOM (Eqs. (7)–(13)) possesses independent computing characteristics (i.e., no connection to other neurons), except that in each iteration they need the responses (η_{kl}) of the other neuron. Its interprocessor communication would be much less than those of other neural networks (such as backpropagation networks, and SOM). Therefore, LISSOM is well suited for implementation in a massively parallel computing environment.

3.1. Computing work load of the map

For the parallel implementation of LISSOM on different number of processors of multiple instruction streams multiple data streams architecture, we develop a general strategy that uniformly partitions the computational loads of the whole network into different processors. Each neuron in the LISSOM network has afferent connections from the external input and a set of lateral connections from the other neurons in the map. Depending on its location, the number of lateral connections could vary from $r \times r$ (at the center) to $\frac{1}{2}r \times \frac{1}{2}r$ (at the corners) (r is the lateral excitation radius or lateral inhibition radius). This is illustrated in of Fig. 6(a)–(d). In Fig. 6(a), the neuron has full lateral connections at the center. In Fig. 6(b), the neuron merely has half lateral connections at the border. In Fig. 6(c), the neuron has quarter lateral connections at the corner. The gradient color represented in Fig. 6(d) is the distribution of the computational loads in the whole map; dark color expresses heavy load, while light color expresses small load. This indicates that the computation loads of neurons in the map are nonuniform. In order to balance the computational loads and generalize a parallel implementation of LISSOM among any number of working processors, the number of operations of individual neurons and the whole map need to be counted. First, the number of individual neurons' lateral excitatory connection or inhibitory connection can be calculated from the upper-left and lower-right corners of the lateral connectional rectangles. The two corners of the rectangles are formulated as follows:

$$L_k = \begin{cases} 0 & k \leqslant r \\ k - r & k > r \end{cases} \tag{14}$$

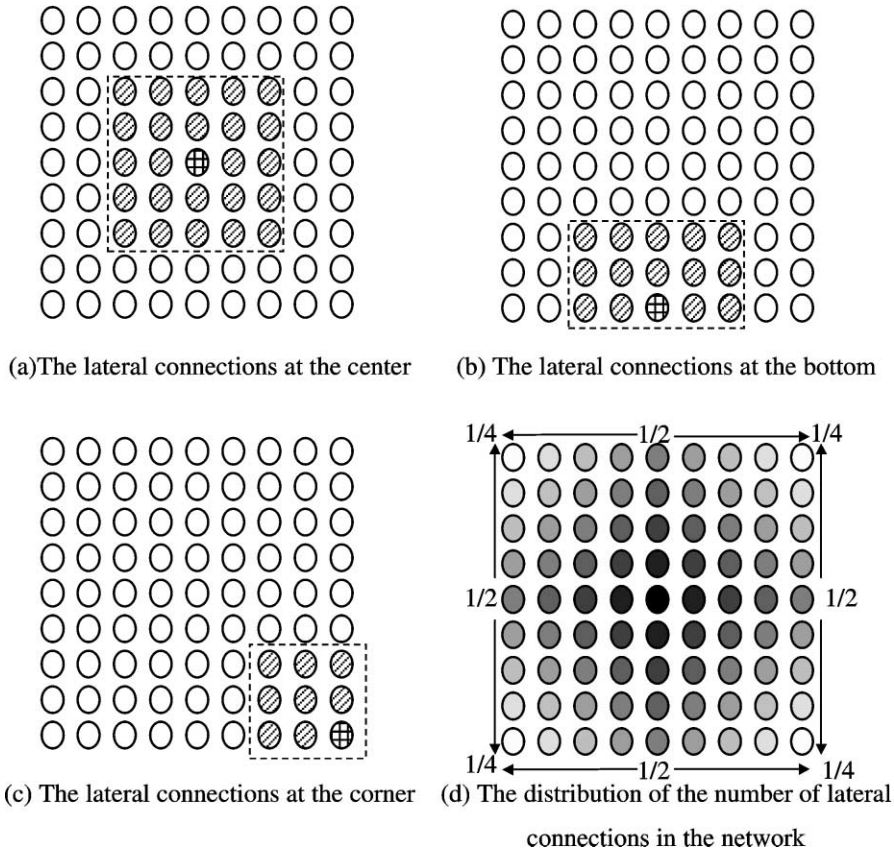

Fig. 6. Topological lateral connections in the network.

$$R_k = \begin{cases} k + r & k \leqslant N - 1 - r \\ N - 1 & k > N - 1 - r \end{cases} \tag{15}$$

where k is a specific neuron in the kth row or column of the map; L_k, the L_kth row or L_kth column of upper-left corner; R_k, the R_kth row or R_kth column of lower-right corner; r, the lateral radius such as the excitation radius or the inhibition radius; and N, the size of the row or column of the map. Generally speaking, the excitation radius d is less than $N/2$, and the inhibition radius d' is larger than $N/2$. Therefore, the lengths of excitatory and inhibitory rectangles are formulated as:

$$W_k^d = \begin{cases} k + d + 1 & k \leqslant d \\ 2d + 1 & d < k \leqslant N - d - 1 \\ N + d - k & k > N - d - 1 \end{cases} \tag{16}$$

$$W_k^{d'} = \begin{cases} k + d' + 1 & k < N - d' - 1 \\ N & N - d' - 1 \leqslant k \leqslant d' \\ N + d' - k & k > d' \end{cases} \tag{17}$$

where W_k^d is the length of excitatory rectangles of a neuron in the kth column or row of the map with excitation radius d; and $W_k^{d'}$, the width or the height of the inhibitory rectangles. For a specific neuron (i,j), the number of lateral excitatory connections is written as

$$C_{i,j}^d = W_i^d W_j^d \tag{18}$$

and the number of lateral inhibitory connections is written as

$$C_{i,j}^{d'} = W_i^{d'} W_j^{d'} \tag{19}$$

Thus, the summation of the lateral excitatory connections and the summation of the lateral inhibitory connections in the whole map are given by Eqs. (20) and (21), respectively.

$$C_{\text{map}}^d = \left[\sum_{i=0}^{d}(i+d+1) + \sum_{i=d+1}^{N-d-1}(2d+1) + \sum_{i=N-d}^{N-1}(N+d-i)\right] \times \left[\sum_{j=0}^{d}(j+d+1) + \sum_{j=d+1}^{N-d-1}(2d+1) + \sum_{j=N-d}^{N-1}(N+d-j)\right] \tag{20}$$

$$C_{\text{map}}^{d'} = \left[\sum_{i=0}^{N-d'-2}(i+d'+1) + \sum_{i=N-d'-1}^{d'} N + \sum_{i=d'+1}^{N-1}(N+d'-i)\right] \times \left[\sum_{j=0}^{N-d'-2}(j+d'+1) + \sum_{j=N-d'-1}^{d'} N + \sum_{j=d'+1}^{N-1}(N+d'-j)\right] \tag{21}$$

where C_{map}^d is the total lateral excitatory connection of the map, and $C_{\text{map}}^{d'}$, the total lateral inhibitory connection of the map. Although the two equations above are quite different, their results have similar formulae.

$$C_{\text{map}}^d = [d(2N-d-1)+N]^2 \tag{22}$$

$$C_{\text{map}}^{d'} = \left[d'(2N-d'-1)+N\right]^2 \tag{23}$$

For computing the initial response of each neuron given in Eq. (7), the number of operations (only the number of multiplication is counted) is h. For refining the initial response through the lateral interaction (according to Eq. (9)), the number of operations is the iterations of activity settled t_{settle}. For adjusting the lateral excitatory and inhibitory weights given in Eq. (10), the number of operations is 2; and for adjusting the afferent weights given in Eq. (11), the number of operations is $3h$. According to Eqs. (12) and (13), the number of operations is 2. Hence, the number of operations of a specific neuron (i,j) $\text{Load}_{i,j}$ and the total number of operations Load_{map} are given by Eqs. (24) and (25), respectively.

$$\mathrm{Load}_{i,j} = (C^{d}_{i,j} + C^{d'}_{i,j})(t_{\mathrm{settle}} + 2) + 4h + 2 \tag{24}$$

$$\mathrm{Load}_{\mathrm{map}} = (C^{d} + C^{d'})(t_{\mathrm{settle}} + 2) + 4N^2h + 2N^2 \tag{25}$$

The average number of operations is

$$\mathrm{Load}_{\mathrm{avg}} = \frac{\mathrm{Load}_{\mathrm{map}}}{\mathrm{nproc}} \tag{26}$$

where nproc is the number of working processors. We add the load of neurons in sequence until the sum is approximately equal to $\mathrm{Load}_{\mathrm{avg}}$; and these neurons called group1 are distributed into processor1. The procedure is continuous for the following processors. However, to prevent the computational load from being concentrated on the last processor, the total load and average load are revised by Eqs. (26) and (27).

$$\mathrm{Load}_{\mathrm{map}} = \mathrm{Load}_{\mathrm{map}} - \mathrm{Load}_{\mathrm{group}i} \tag{27}$$

$$\mathrm{Load}_{\mathrm{avg}} = \frac{\mathrm{Load}_{\mathrm{map}}}{\mathrm{nproc} - i} \tag{28}$$

Thus, through Eqs. (22)–(28), the computational load of any sized network can be uniformly distributed among different number of processors. For example, given a 10×10 network with corresponding parameters in Table 1 ($N = 10$, $h = 2$, $d = 2$, $d' = 7$, $t_{\mathrm{settle}} = 15$) and used five processors, the computational loads are calculated as follows. According to Eqs. (22) and (23), we get the total lateral excitatory $C^{d}_{\mathrm{map}} = [2 \times (2 \times 10 - 2 - 1) + 10]^2 = 1936$ and inhibitory connection $C^{d}_{\mathrm{map}} = [2 \times (2 \times 10 - 2 - 1) + 10]^2 = 1936$ of the map. The total number of operations $\mathrm{Load}_{\mathrm{map}} = (1936 + 8836) \times (15 + 2) + 4 \times 10^2 \times 2 + 2 \times 10^2 = 184{,}124$ and the av-

Table 1
The simulation parameters for different size of the LISSOM model

Parameters	Size				
	10×10	15×15	20×20	25×25	30×30
d	2	3	4	5	6
d'	7	11	15	19	23
t_{settle}	15	15	15	15	15
t_0	8000	6000	5000	6000	8000
Interval t_c	1500	1500	1000	1000	1000
Threshold	0.001	0.001	0.001	0.0003	0.0001
$\gamma_{\mathrm{e}}, \gamma_{\mathrm{i}}$	0.9	0.9	0.9	0.9	0.9
α	0.005	0.005	0.005	0.005	0.005
$\alpha_{\mathrm{E}}, \alpha_{\mathrm{I}}$	0.001	0.001	0.001	0.001	0.001
$\alpha_{\delta}, \alpha_{\beta}$	0.00011	0.00011	0.00011	0.00011	0.00011
Initial $\delta, \delta_{\max}$	0.6, 0.9	0.6, 0.9	0.6, 0.9	0.6, 0.9	0.6, 0.9
Initial $\beta, \beta_{\min}$	1.3, 1.13	1.3, 1.13	1.3, 1.13	1.3, 1.13	1.3, 1.13

erage number of operations $\text{Load}_{\text{avg}} = \frac{184,124}{5} \approx 36,824$ are calculated from Eqs. (25) and (26), respectively. Then, the number of operations of each neuron can be calculated from Eq. (24), and the load of neurons for the first processor is sequentially added from the first neuron to the neuron that the sum of the load is nearly Load_{avg}.

$$\begin{aligned}\text{Load}_{\text{group1}} &= \text{Load}_{0,0} + \text{Load}_{0,1} + \cdots + \text{Load}_{2,1} \\ &= 1166 + 1370 + \cdots + 1880 = 36,107\end{aligned} \tag{29}$$

In this example, the first 22 neurons are grouped and distributed into processor 1. Then, the average load is modified through Eqs. (27) and (28). $\text{Load}_{\text{avg}} = (\text{Load}_{\text{map}} - \text{Load}_{\text{group1}})/(5-1) = \frac{184,124-36,107}{4} \approx 37,004$. The procedure is continuous for the following processors. The result of distributing these neurons into five processors is shown as Fig. 7.

3.2. Parallel implementation

In the implementation of LISSOM, the process of initializing three different kinds of weights, i.e., afferent weights, lateral excitatory connection weights and inhibitory connection weights cannot be parallelized. The initial values of each kind of weights must be done on a single processor to maintain the characteristic of randomly uniform distribution. However, it is not necessary to initialize all those three kinds of weights on a single processor. We can synchronously assign any three of the working processors to individually initialize one kind of weight. In this way, it will take less time than a single processor that initializes all kinds of weights.

Designing parallel programs for implementing the LISSOM model on IBM SP2 and PC cluster is based on the communication and message-passing model of MPI. Unfortunately, MPI-1.1, the one we used on our computers, does not support for parallel I/O, so the external inputs and predetermined parameters are read from master processor (rank = 0) and broadcasted from master processor to every

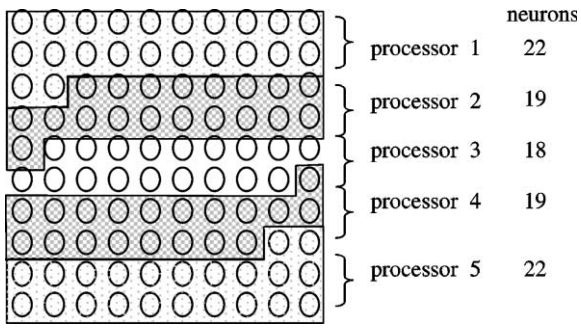

Fig. 7. The result of uniformly distributing the computational loads of a 10×10 network into five processors.

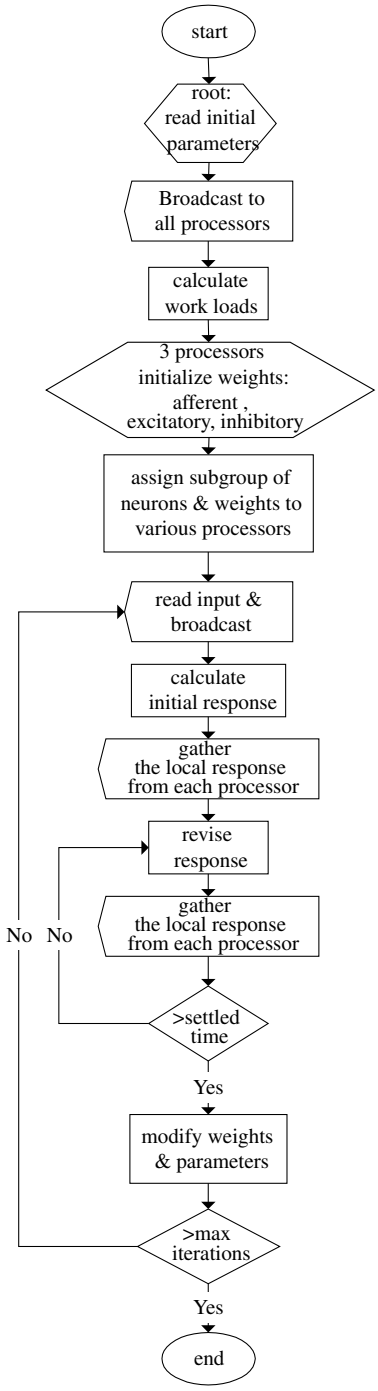

Fig. 8. The flowchart of LISSOM implementation on parallel environment.

processor. This is one of the overhead sources. Recently, some parallel systems provide an update version of MPI, MPI-2.0. The MPI-2.0 has several new functions to support parallel I/O. The communication overhead might, then, be decreased or even eliminated.

In addition, all the lateral weights and responses (Eqs. (9) and (10)) in each neuron (i, j) are relative to the responses of other neurons (k, l) that might not be all in the same processor. To minimize the complexity of interprocessor communications, every processor has a copy of the responses of the whole map. In each time step (t in Eq. (9)), the responses of each processor are updated when all processors have completely updated their own responses; the newest responses of the whole map are then "gathered" into every processor. The flowchart of the parallel implementation of LISSOM is shown in Fig. 8.

4. A basic implementation for LISSOM

The main goal of designing a parallel algorithm for LISSOM is to reduce the execution time of application. In this section, a concrete implementation of the parallel-LISSOM model with different-sized networks and different input dimensions is illustrated. The execution time, speedup and efficiency are also presented.

4.1. A sequential simulation of LISSOM

In Sirosh's dissertation [17], an example of sequential simulating the distribution of two-dimensional inputs with a 20×20 network was described. Based on the set of parameters he suggested for a 20×20 network, we have explored a large number of sets of corresponding parameters for different-sized networks through simulations. These predetermined parameters presented in Table 1 are appropriate for simulating the given sizes of the LISSOM maps.

Figs. 9 and 10 show the self-organization processes of the afferent weights that represent the input distribution in two different sizes of the networks, i.e., 10×10 and 25×25. The weight vector of each neuron in the networks is connected to four immediate neighbors by a line. The resulting grid represents the topological organization of the map. The afferent weight vectors at four different epochs. The four graphs in Figs. 9 and 10 are described as follows. In Fig. 9(a), the afferent weight vectors are initially random uniformly distributed on the input space. In Fig. 9(b), the network gradually unfolds after 3000 epochs. In Fig. 9(c), the network starts killing lateral connections, and the number of epochs depends on the size of network (see Table 1). Finally after 50,000 epochs, Fig. 9(d) shows the network eventually converges and unfolds.

The following subsections illustrate the applicability of the proposed parallel algorithm of LISSOM in two ways. The first one shows the results of speedup and efficiency in networks of different sizes by using different number of processors. The

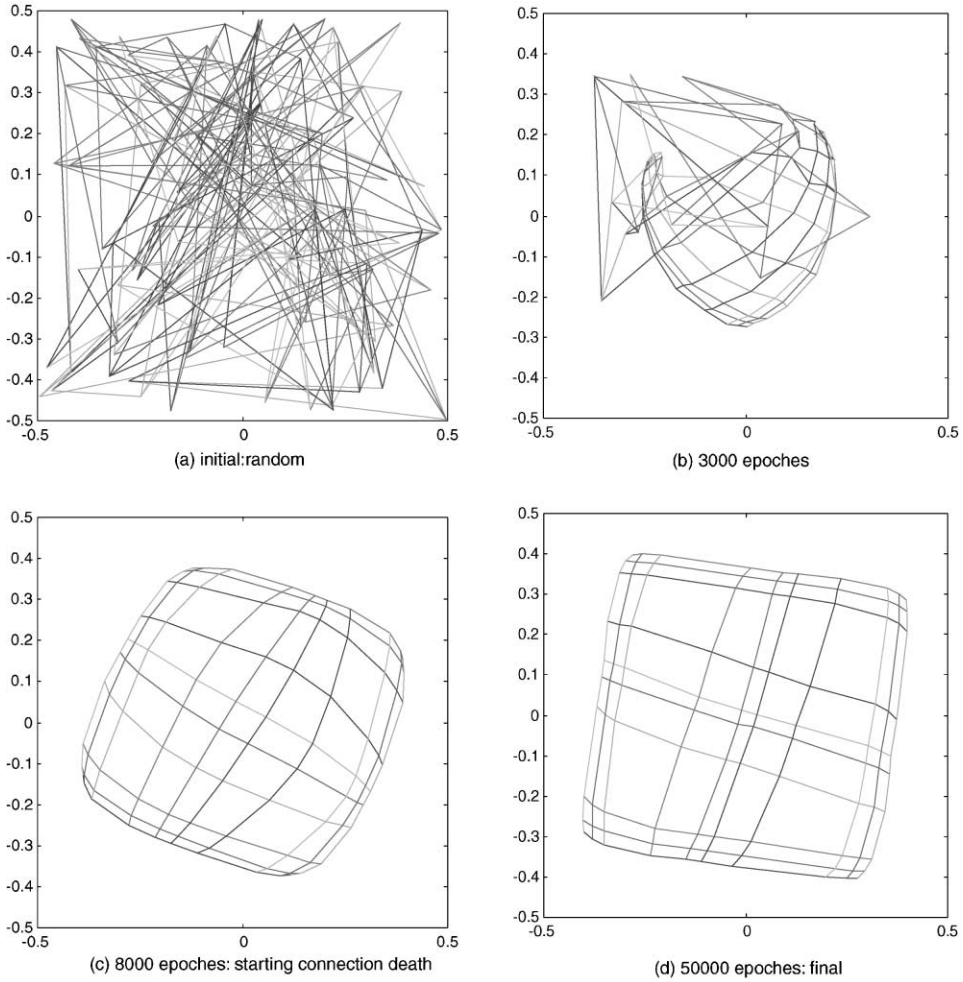

Fig. 9. The topological map formation in the simulation of 10×10 network. The network is trained with a two-dimensional input vector x randomly drawn from uniform distribution in [−0.5, 0.5].

second one shows the results of speedup and efficiency in different input dimensions by using different number of processors.

4.2. Parallel simulations of LISSOM for different network sizes

Since the size of the network depends on the complexity of problems and the requirement of accuracy, simulating different-sized networks on parallel environments is indispensable to engineering applications. In this subsection, two parallel computer systems, IBM SP2 and PC cluster, are used for simulating different network sizes.

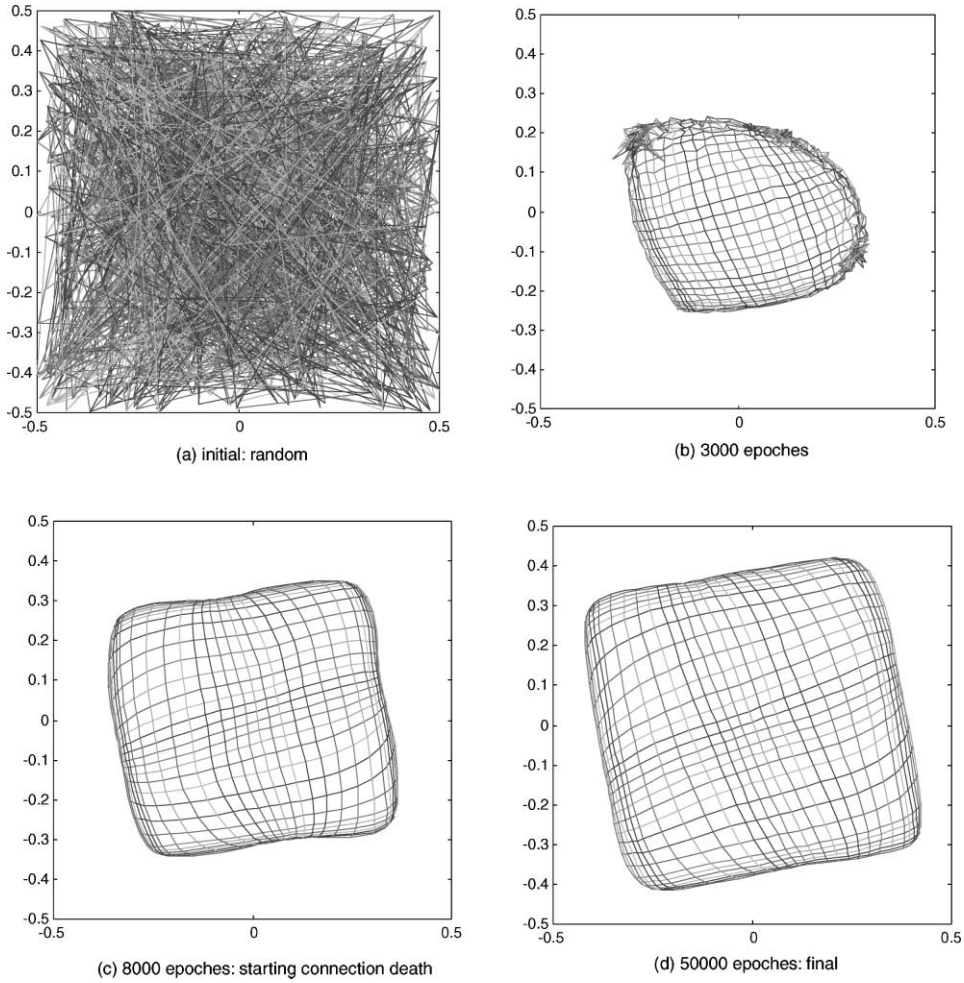

Fig. 10. The topological map formation in the simulation of 30×30 network.

4.2.1. IBM SP2

Tables 2–4 show the execution times, speedup and efficiency of 10×10, 15×15, 20×20, and 25×25 networks by using different number of processors. The results show that as the network size increases, the rate of speedup increases and the efficiency tends to high degree. Fig. 11 shows speedup versus number of processors curves for implementing 10×10, 15×15, 20×20, and 25×25 networks on IBM SP2.

4.2.2. PC cluster

We implement more networks by using more processors on PC cluster to demonstrate the performance of the proposed parallel algorithm. Tables 5–7 show the

Table 2
Execution times in seconds on IBM SP2

No. of processors	Network size			
	10×10	15×15	20×20	25×25
1	4128.50	20263.15	63721.93	155271.43
2	2929.47	10993.15	32839.05	79618.76
4	3019.04	7942.38	19230.06	42116.34
6	3079.29	6258.25	14723.59	30591.30
8	3481.39	6080.69	12237.14	24590.68

Table 3
Speedup (S) on IBM SP2

No. of processors	Network size			
	10×10	15×15	20×20	25×25
1	1.00	1.00	1.00	1.00
2	1.41	1.84	1.94	1.95
4	1.37	2.55	3.31	3.69
6	1.34	3.24	4.33	5.08
8	1.19	3.33	5.21	6.31

Table 4
Efficiency (E) on IBM SP2

No. of processors	Network size			
	10×10	15×15	20×20	25×25
1	1.00	1.00	1.00	1.00
2	0.70	0.92	0.97	0.98
4	0.34	0.64	0.83	0.93
6	0.22	0.54	0.72	0.87
8	0.15	0.42	0.65	0.86

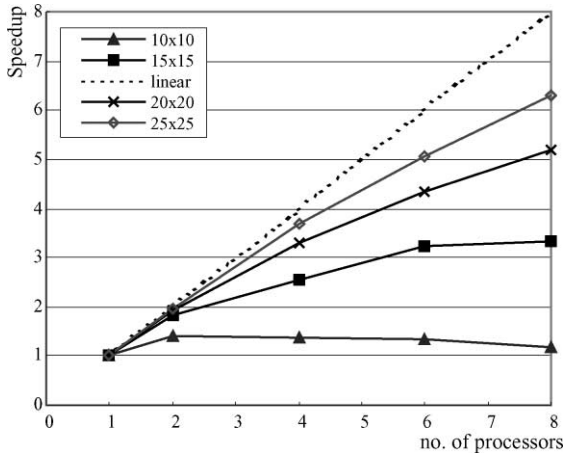

Fig. 11. Speedup versus the number of processors with different networks on IBM SP2.

Table 5
Execution time in seconds on PC cluster

No. of processors	Network size	
	10×10	20×20
1	791.45	21565.61
2	441.29	11218.34
4	305.6	5907.49
6	300.87	4192.59
8	303.79	3303.07
16	392.19	1633.75

Table 6
Speedup (S) on PC cluster

No. of processors	Network size	
	10×10	20×20
1	1.00	1.00
2	1.79	1.92
4	2.59	3.65
6	2.63	5.14
8	2.61	6.53
16	2.02	13.2

Table 7
Efficiency (E) on PC cluster

No. of processors	Network size	
	10×10	20×20
1	1.00	1.00
2	0.90	0.96
4	0.65	0.91
6	0.44	0.86
8	0.33	0.82
16	0.13	0.83

execution times, speedup and efficiency of 10×10 and 20×20 networks by using different number of processors. The results illustrate that the rate of speedup tends increased and the efficiency sustain high degree when the network size is large. Fig. 12 shows speedup versus number of processors curves for implementing different sized networks on PC cluster.

4.3. Parallel implementations of LISSOM for different inputs dimensions

In practice, the LISSOM model may be used for various problems with different number of dimensions. Therefore, implementing different input dimensions on

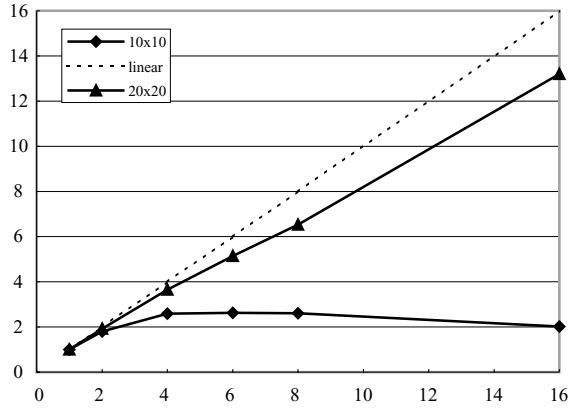

Fig. 12. Speedup versus the number of processors with different networks on PC cluster.

parallel environments is essential to diverse applications. In this subsection, PC cluster is used for implementing 16, 64, 128 and 1024 input dimensions in a 20×20 network. Tables 8 and 9 show the speedup and efficiency, respectively. Fig. 13 shows

Table 8
Speedup (S) on PC cluster

No. of processors	Dimension			
	16	64	128	1024
1	1.00	1.00	1.00	1.00
2	1.98	1.978	1.96	1.96
4	3.71	3.70	3.67	3.74
6	5.23	5.25	5.18	5.30
8	6.66	6.67	6.65	6.89
16	13.74	12.39	13.03	12.21

Table 9
Efficiency (E) on PC cluster

No. of processors	Dimension			
	16	64	128	1024
1	1.00	1.00	1.00	1.00
2	0.99	0.99	0.98	0.98
4	0.93	0.93	0.92	0.94
6	0.87	0.88	0.86	0.88
8	0.83	0.83	0.83	0.86
16	0.86	0.77	0.81	0.76

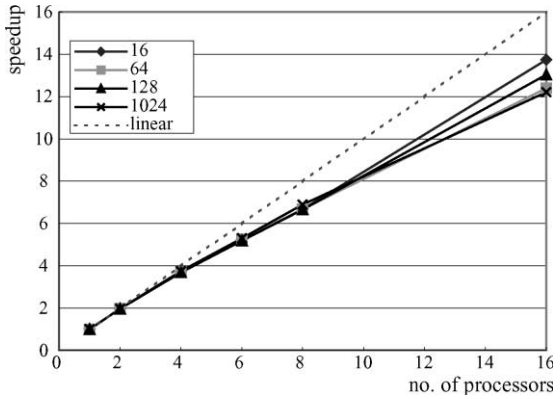

Fig. 13. Speedup versus the number of processors with different input dimensions on PC cluster.

speedup versus number of processors curves for implementing different number of input dimensions on PC cluster.

5. Conclusions

There is a continual demand for faster computations with current computer systems. Parallel programming uses multiple computers to solve a problem at a greater computational speed than using a single computer. It provides a means to tackle larger problems with existing computer systems. In this study, we develop a parallel program of LISSOM that distribute computations evenly across processors and obtain the highest possible execution speed.

The results of parallel implementation for different sizes of the LISSOM network show that this system is efficient for networks with larger sizes and less effective for smaller networks. For a small network (i.e., 10×10), the speedup tends to become saturated, and speedup curve flattens or even decays. The training processes of implementing small networks with several parallel processors could take more execution time than a single processor due mainly to the need for interprocessor communications in parallel environments. In contrast, parallel implementation of larger network in LISSOM yields high speedup and efficiency. For instance, on 20×20 networks the speedup is close to 6 when eight processors (on IBM SP2 or PC cluster) are used. This study demonstrates that the LISSOM model can be applied to complex problems through the parallel algorithm.

The results of parallel implementation for different input dimensions in the network of the same size (i.e., 20×20) show that the speedup and efficiency sustain at a high level. We demonstrate that the problem size (or the number of input dimension) does not affect the time of interconnection communications among parallel processors during the training phase except in I/O passing. The results

demonstrate that the parallel LISSOM model can be successfully used to larger sized problems.

Acknowledgements

This research is partially supported by the Nation Science Council (NSC), ROC under grant NSC90-2313-B002-261. We acknowledge National Center for High Performance Computing (NCHC) for providing IBM SP2, as well as Professor Chung Yeh-Ching in the Department of Information Engineering at the Feng Chia University for his support on providing PC cluster. Suggestions from Professor Chen Yung-Yaw in the Department of Electrical Engineering of the National Taiwan University are greatly appreciated.

Appendix A

A.1. IBM SP2

The IBM SP2 is a parallel machine equipped with Thin_P2SC160MHz processors with, 512 MB of RAM and 9 GB of Physical Disks at National Center for High Performance Computing, Taiwan, ROC.

A.2. PC cluster

The PC cluster has 32 processors and is operated by the Feng Chia University, Taiwan. This cluster is built with 16 dual 800 MHz Pentium III nodes, each node equipped with 512 MB of RAM, 20 GB of Physical Disk and 1.2 Gbit/s Gigabit Ethernet.

References

[1] J. Bednar, R. Miikulainen, Tilt aftereffects in a self-organizing model of the primary visual cortex, Neural Computation 12 (7) (2000) 1721–1740.
[2] M.W. Benson, J. Hu, Asynchronous self-organizing maps, IEEE Transactions on Neural Networks 11 (6) (2000) 1315–1322.
[3] L.C. Chang, F.J. Chang, Intelligent control for modelling of real-time reservoir operation, Hydrological Processes 15 (9) (2001) 1621–1634.
[4] F.J. Chang, Y.C. Chen, A counterpropagation fuzzy-neural network modelling approach to real-time streamflow prediction, Journal of Hydrology 245 (2001) 153–164.
[5] J. Dongarra, E. Luque, T. Margalef, Recent advances in parallel virtual machine and message passing interface, 6th European PVM/MPI User's Group Meeting. Barcelona, Spain, Proceedings, 1999.
[6] M. Hajmeer, I.A. Basheer, Artificial neural networks: fundamentals, computing, design, and application, Journal of Microbiological Methods 43 (2000) 3–31.
[7] T. Hämäläinen, H. Klapuri, J. Saarinen, K. Kaski, Mapping of SOM and LVQ algorithms on a tree shape parallel computer system, Parallel Computing 23 (1997) 271–289.
[8] S. Haykin, Neural Networks, Prentice Hall, New Jersey, 1999.

[9] M. Ibnkahla, Applications of neural network to digital communications—a survey, Signal Processing 80 (2000) 1185–1215.
[10] L.C. Jain, V. Vemuri, Industrial Application of Neural Networks, CRC Press, 1999.
[11] T. Kohonen, Self-Organization and Associative Memory, Springer Verlag, Berlin, 1989.
[12] R. Kothari, S. Islam, Spatial characterization of remotely sensed soil moisture using self-organizing feature maps, IEEE Transactions on Geoscience and Remote Sensing 37 (2) (1999) 1162–1165.
[13] S. Mahapatra, R.N. Mahapatra, B.N. Chatterji, Mapping of neural network models onto massively parallel hierarchical computer systems, Journal of Systems Architecture 45 (1999) 919–929.
[14] National Science Foundation Science and Technology Center (NSF), MPI: A Message-Passing Interface Standard, 1995.
[15] D. Purves, J.W. Lichtman, Principles of Neural Development, Sinauer, Sunderland, MA, 1985.
[16] A. Rauber, P. Tomsich, D. Merkl, parSOM: A parallel implementation of the self-organizing map exploiting cache effects: making the SOM fit for interactive high-performance data analysis, Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN 2000) 6 (2000) 177–182.
[17] J. Sirosh, A self-organizing neural network model of the primary visual cortex, Ph.D. thesis, Department of Computer Science, University of Texas at Austin, 1995.
[18] J. Sirosh, R. Miikkulainen, How lateral interaction develops in a self-organizing feature map, in: Proceedings of the IEEE International Conference on Neural Networks, San Francisco, 1993, pp. 1360–1365.
[19] J. Sirosh, R. Miikkulainen, Cooperative self-organization of afferent and lateral connections in cortical maps, Biological Cybernetics 71 (1994) 66–78.
[20] J. Sirosh, R. Miikkulainen, Topographic receptive fields and patterned lateral interaction in a self-organizing model of the primary visual cortex, Neural Computation 9 (1997) 577–594.
[21] N. Sundararajan, P. Saratchandran, Parallel Architectures for Artificial Neural Networks, IEEE Computer Society Press, 1998.
[22] B.H.V. Topping, J. Sziveri, A. Bahreinejad, J.P.B. Leite, B. Cheng, Parallel processing, neural networks and genetic algorithms, Advances in Engineering Software 29 (10) (1998) 763–786.
[23] D. Zhang, S.K. Pal, Parallel system design for time-delay neural networks, IEEE Transactions on Sytems, Man, and Cybernetics-Part C: Applications and Reviews 30 (2) (2000) 265–275.