Sushmita Mitra

Fuzzy radial basis function network: a parallel design

Received: 8 September 2003 / Accepted: 13 May 2004 / Published online: 31 July 2004

Abstract The fuzzy radial basis function (FRBF) network comprises an integration of the principles of a radial basis function (RBF) network and the fuzzy *c*-means (FCM) algorithm. A programmable parallel architecture design is proposed for the FRBF, both for FCM clustering at the hidden layer and the weight training at the output layer of the network. The behavior of the system is described in terms of processor utilization. The performance of the parallel design is quantitatively evaluated.

Keywords Parallel neural architecture · Soft computing · Radial basis function network · Fuzzy clustering · Performance evaluation

1 Introduction

Soft computing is a consortium of methodologies that works synergistically and provides a flexible information processing capability for handling real life ambiguous situations [9]. Its aim is to exploit the tolerance for imprecision, uncertainty, approximate reasoning, and partial truth in order to achieve tractability, robustness, and low-cost solutions. Neuro-fuzzy computing [8] is the earliest and most widely reported hybridization in this framework. This integration provides intelligent systems, in terms of parallelism, fault tolerance, adaptivity, and uncertainty management, in order to handle real life recognition/decision-making problems.

The radial basis function (RBF) [6] is a three-layered network, typically used for supervised classification. The hidden layer performs crisp clustering using a Gaussian basis function at the nodes. The output layer performs a linear combination of the weighted activations from the hidden layer. The fuzzy radial basis function (FRBF) [5] is designed by integrating the principles of an RBF network and the fuzzy *c*-means (FCM) algorithm [1]. It incorporates fuzzy set-theoretic concepts at the input, output, and hidden layers. The model can handle both linguistic and numeric inputs, and provides a soft decision in case of overlapping pattern classes at the output. The use of the FCM algorithm in the hidden layer allows the network to provide a more accurate representation of real life situations, where a pattern can have finite non-zero membership to two or more classes. The architecture of the network is suitably modified at the hidden layer to realize the fuzzy clustering algorithm.

As the field of neural networks matures towards real-world applications, a need arises for hardware systems to efficiently model larger networks and/or larger data sets. Many neural algorithms are suitable for implementation on special-purpose hardware, typically because of their intrinsic fine-grained parallelism and computationally intensive nature. Parallel programming imitates the human thought process by allowing computations to be simultaneously made and distributed throughout the computer system [2]. Recently, substantial work has been carried out to design parallel hardware implementations for neural networks [4]. The network models have been redesigned so as to enable the use of parallel programming techniques for faster adaptability of the system. There has been research on using systolic arrays for implementing backpropagation in multilayer networks [7, 3], and fuzzy clustering neural networks [10], to enhance the performance of the system by increasing the amount of processing done per unit time.

In this article, we propose a parallel hardware design of the FRBF network. The architecture exploits the inherent parallelism in the FCM algorithm and the FRBF model for possible VLSI implementation. The

S. Mitra
Machine Intelligence Unit,
Indian Statistical Institute,
203 B. T. Road, Kolkata, 700 108, India
E-mail: sushmita@isical.ac.in

abundance of local computations enable easy parallel modeling of the FRBF network. The effectiveness of the design for both FCM clustering at the hidden layer and the weight training at the output layer of the FRBF network, is provided in terms of processor utilization. The performance of the system is evaluated using different quantitative measures.

Section 2 describes the FRBF network [5] in short. The parallel architecture design for the hidden layer, using FCM clustering, is elaborated upon in Sect. 3. The parallel design for the weight learning at the output layer of the FRBF network is provided in Sect. 4. The article is concluded in Sect. 5.

2 Fuzzy radial basis function network

The FRBF network [5] incorporates an amalgamation of FCM clustering at the hidden layer of the RBF network. Before proceeding to the details of its parallel design, we provide a brief overview on the RBF network, followed by that of the FRBF network.

The input and output nodes of an RBF network correspond to the input features and output classes, while the hidden nodes represent the number of clusters that partition the input space. Let $\vec{\mathbf{X}} = (X_1, \ldots, X_i, \ldots, X_n) \in R^n$ and $\vec{\mathbf{y}} = (y_1, \ldots y_i, \ldots, y_l) \in R^l$ be the input and output, respectively, and c be the number of hidden nodes. The problem is to minimize the error:

$$E = \frac{1}{2}\sum_{p=1}^{N}\sum_{j=1}^{l}\left(y_j^{(p)} - {}^{*}y_j^{(p)}\right)^2 \tag{1}$$

where ${}^{*}y_j^{(p)}$ and $y_j^{(p)}$ are the desired and computed output at the jth node for the pth pattern, respectively, N is the size of the data set, and l is the number of output nodes. A fixed set of cluster centers, $\vec{\mathbf{v}}_j$, $j = 1, \ldots, m$, is first formed by a clustering algorithm. Associations of the cluster centers with the output are then learned by squared error minimization of E.

In an RBF network, the clustering of the input data is represented by crisp partitions and the clusters are modeled by the Gaussian distribution. It is more natural, from the fuzzy set-theoretic point of view, to determine the membership value of each data point to different clusters using the FCM algorithm [1]. Here, the membership value of a point to different clusters is determined based on the relative closeness of the point to the different cluster centers.

In an FRBF network, the input space is partitioned using overlapping linguistic sets, thereby, utilizing more local information that aids in better classification. Each input feature, X_j, is expressed in terms of membership values to each of the three linguistic property sets low, medium, and high. Therefore, an n-dimensional pattern, $\vec{\mathbf{X}}_i = [X_{i1}, X_{i2}, \ldots, X_{in}]$, is represented as a $3n$-dimensional vector:

$$\begin{aligned}[x_1, \ldots, x_{3n}] &= \left[\mu_{\text{low}}(X_{i1})\left(\vec{\mathbf{X}}_i\right), \mu_{\text{medium}(X_{i1})}\left(\vec{\mathbf{X}}_i\right),\right. \\ &\quad \left.\mu_{\text{high}(X_{i1})}\left(\vec{\mathbf{X}}_i\right), \ldots \mu_{\text{high}(X_{in})}\left(\vec{\mathbf{X}}_i\right)\right]\end{aligned} \tag{2}$$

Here, the linguistic properties low, medium, and high are modeled using $1-S$, π, and S functions [8], respectively. The output is provided in terms of class membership values to the l classes, such that $0 \leqslant \mu_k(\vec{\mathbf{x}}_i) \leqslant 1$ for $k = 1, \ldots, l$. This is proportional to the weighted distance of the training pattern from the kth class mean.

The input-hidden layer weights are initialized by cluster centers using fuzzy c-means (FCM), instead of the more conventional hard c-means. The intermediate (hidden) layer is suitably modified to incorporate FCM clustering [1] during learning, such that each output node receives the weighted membership value (as opposed to a Gaussian-function-based measure of proximity) of the enhanced input vector within each cluster. The resultant FRBF architecture is depicted in Fig. 1 [5].

In the FCM algorithm, the membership value of any pattern vector, $\vec{\mathbf{x}}_j$, to a class k is represented as:

$$u_{kj} = \frac{1}{\sum_{i=1}^{c}\left(\frac{d_{kj}}{d_{ij}}\right)^{\frac{2}{m-1}}} \tag{3}$$

where d_{kj} is the distance of the pattern vector, $\vec{\mathbf{x}}_j$, from the center, $\vec{\mathbf{v}}_k$ of the kth cluster. Here,

$$\vec{\mathbf{v}}_k = \frac{\sum_{j=1}^{N}\left(u_{kj}\right)^m \vec{\mathbf{x}}_j}{\sum_{j=1}^{N}\left(u_{kj}\right)^m} \tag{4}$$

with fuzzifier $1 < m < \infty$, such that $u_{kj} \in [0, 1]$, for c clusters $1 \leq k \leq c$, for N pattern points $1 \leq j \leq N$, with $\sum_{k=1}^{c} u_{kj} = 1$, for $1 \leq j \leq N$, and $\sum_{j=1}^{N} u_{kj} > 0$, for $1 \leq k \leq c$.

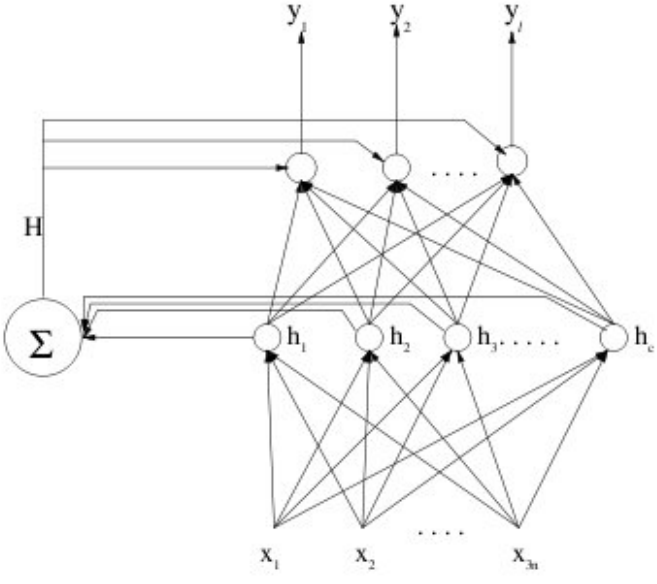

Fig. 1 Fuzzy radial basis function (FRBF) network

The objective is to perform fuzzy partitioning of the data in the hidden layer of the FRBF network. In order to perform the local computation of Eq. 3, a modified architecture is used. Equation 3 is rewritten as:

$$u_{kj} = \frac{h_k^{(j)}}{\sum_{i=1}^{c} h_i^{(j)}} \tag{5}$$

where

$$h_i^{(j)} = \left(\frac{1}{d_{ij}}\right)^{\frac{2}{m-1}} \tag{6}$$

The activation of each node in the output layer is given as:

$$y_i^{(p)} = \sum_{j=1}^{c} W_{ij} u_{jp} \tag{7}$$

where $y_i^{(p)}$ is the response of the ith output node when $\vec{\mathbf{x}}_p$ is present at the input of the network. From Eqs. 5 and 6, $y_i^{(p)}$ can be written as:

$$y_i^{(p)} = \frac{1}{H_a^{(p)}} \sum_{j=1}^{c} W_{ij} h_j^{(p)} \tag{8}$$

where

$$H_a^{(p)} = \sum_{j=1}^{c} h_j^{(p)} \tag{9}$$

Equations 6 and 8 reveal that the h_is can be computed locally in the hidden nodes and that the activation of the output nodes can be computed from the hidden node activations with an additional normalization by the total output in the hidden layer (H_a). An auxiliary hidden node is used in the FRBF (as shown in Fig. 1) to compute the total activation in the hidden layer and feed it to the output layer. The weights of the links from all hidden nodes to the auxiliary hidden node are set to unity. Note that the membership value, u_{kj}, of Eq. 3 is implicitly included in the network architecture in terms of the hidden node activations.

During training, the rule for updating the weights is given as:

$$\Delta W_{ij}^{(p)} = \frac{\eta}{H_a^{(p)}} \left(*y_i^{(p)} - y_i^{(p)}\right) h_j^{(p)} \tag{10}$$

where η is the learning rate and $*y_i^{(p)}$ is the target output in terms of class membership of the training pattern. Here, $\Delta W_{ij}^{(p)}$ is the change in W_{ij} during training when $\vec{\mathbf{x}}_p$ is presented as the input in the $3n$-dimensional form of Eq. 2.

3 Parallel architecture for clustering

In this section, we propose a parallel implementation of the clustering layer of the FRBF architecture. For this purpose, we model the FCM using Eqs. 4 and 5. This is used at the input-hidden layer of the network. The parallel design, processor utilization, and performance evaluation are presented in sequence.

The processors are partitioned into four categories: A, B, C, and D, such that there are c processors each of type A, C, and one processor each of the other two types. These processors are capable of simultaneously working on a $3n$-dimensional vector. We do not, however, go into the details of either register/gate-level designs. The buses are wide enough to simultaneously transmit the $3n$-dimensional data.

3.1 Design

Figure 2 provides a schematic diagram of the architecture for implementing the FCM algorithm. Processors of type A, additionally, store the centers v_i of the c clusters. These are initialized by the first c samples from the input data. Thereafter, the $3n$-dimensional data, $\vec{\mathbf{x}}_j$ (of Eq. 2), is simultaneously fed to all of the A-type processors. Each processor calculates $h_i^{(j)}$ for the input pattern w.r.t. its v_i, by Eq. 6, and passes a copy of it to processor B. In addition, the As are also connected to processors of type C by one-to-one bidirectional connections. These buses pass the input data, $\vec{\mathbf{x}}_j$, and the computed $h_i^{(j)}$ from A to C in the forward pass.

As data is passed to B, the processor calculates the sum, $H_a^{(j)}$ by Eq. 9. Because of the need for synchronization, this process takes extra cycles to let the type A processors feed the data to the B-type processors. As soon as all of the A-type processors have transferred their output to the B processors, the next input pattern, $\vec{\mathbf{x}}_{j+1}$, enters A. Simultaneously, the output, $H_a^{(j)}$ from B is passed to the type C processors.

Processors of type C are used to calculate the membership value, u_{ij}, of the input pattern, $\vec{\mathbf{x}}_j$, to cluster i by Eq. 5. They also compute the product of the computed membership and the input data, using the fuzzifier m. This product, $u_{ij}^m x_j$, along with the membership value, u_{ij}^m, are transmitted to processors A during the backward pass. As a single bidirectional bus is used to connect the As with the Cs, it is, therefore, used in alternation during both passes. Processors A compute the sum of the numerator and denominator of Eq. 4 over each pattern presentation. At the end of an iteration, the cluster centers, v_i, are updated in A.

Membership values, u_{ij}, in respective clusters are also passed to D. Processor D is used to compute the sum of all u_{ij}s. It is used to detect the iteration when the change in total membership values, summed over all patterns, becomes smaller than a threshold, ϵ. This provides a stopping signal to processors A from further updating the cluster centers by Eq. 4, thereby, signifying convergence.

3.2 Processor utilization

Table 1 provides the processor utilization diagram for the proposed parallel FCM design. For ease of

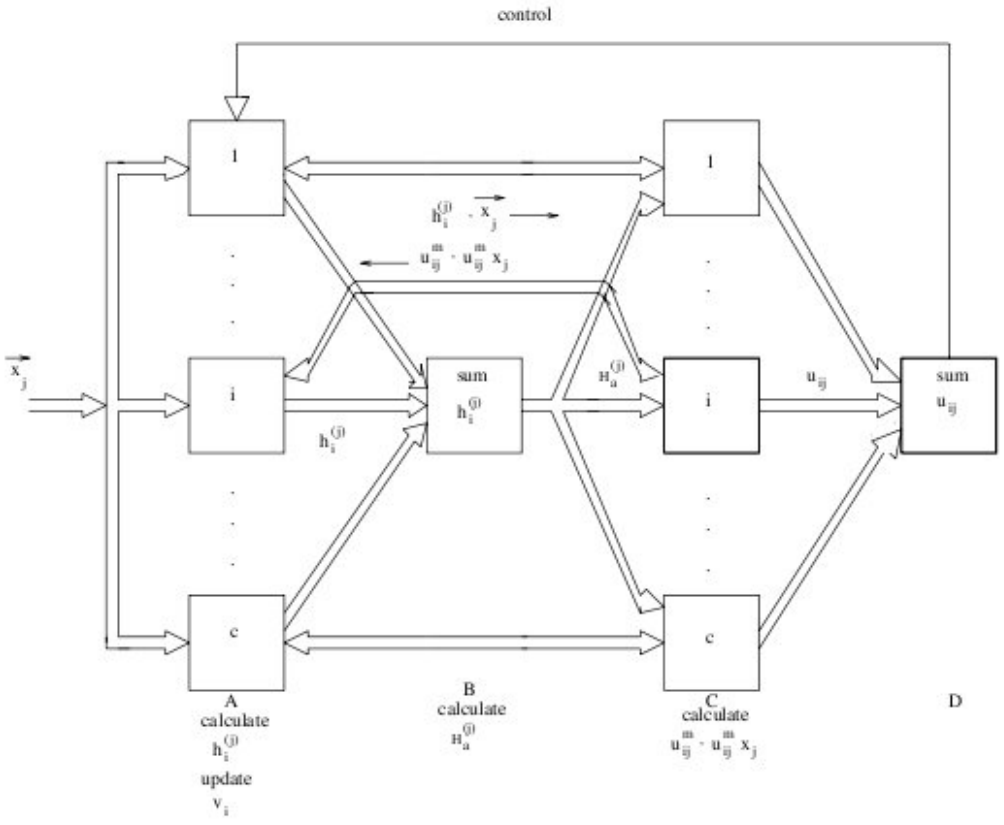

Fig. 2 Parallel architecture for FCM algorithm

explanation, let us assume that there are three hidden nodes such that $c=3$. Thus, we have three processors, each of types A (1, 2, 3) and C (5, 6, 7), while B and D correspond to processors 4 and 8, respectively. The arrows indicate the usage mode of the processors. An upward arrow indicates input of data to the processor and a downward arrow is used for clock cycles when the processor performs some computation. It is seen from the figure that, while data is fed to processors 1, 2, 3, and 5, 6, 7 in one clock cycle, it takes $c-1$ clock cycles to feed the data to processors 4 and 8. This is because of the synchronization problem, which occurs due to contention in feeding the data to processors B and D from processors As and Cs, respectively.

At time cycle $t=1$, let pattern $\vec{\mathbf{x}}_j$ be input to processors A. At $t=2$, $h_i^{(j)}$ is computed at A, and a copy of this is forwarded to B at $t=3$, 4. At $t=5$, $H_a^{(j)}$ is computed at B, $\vec{\mathbf{x}}_j$ is passed from A to C, and $\vec{\mathbf{x}}_{j+1}$ is input to A. At $t=6$, $h_i^{(j)}$ is forwarded from A to C, while $h_i^{(j+1)}$ is computed at A. At $t=7$, $H_a^{(j)}$ is transmitted from B to C. At $t=8$, 9, $h_i^{(j+1)}$ is forwarded from A to B, while $u_{ij}{}^m$ and $u_{ij}{}^m x_j$ are computed, respectively, at C. At $t=10$, 11, u_{ij} is passed from C to D. At $t=10$, simultaneously, $H_a^{(j+1)}$ is computed at B, $\vec{\mathbf{x}}_{j+1}$ is passed from A to C, and $\vec{\mathbf{x}}_{j+2}$ is input to A. At $t=11$, $h_i^{(j+1)}$ is forwarded from A to C, while $h_i^{(j+2)}$ is computed at A. At $t=12$, $H_a^{(j+1)}$ is transmitted from B to C, and u_{ij} is summed at D. Simultaneously, at $t=12$, 13, $u_{ij}{}^m$ and $u_{ij}{}^m x_j$ are fed back from C to A.

At $t=14$, 15, a copy of $h_i^{(j+2)}$ is forwarded from A to B, while $u_{i(j+1)}{}^m$ and $u_{i(j+1)}{}^m x_{j+1}$ are computed at C, respectively. At $t=16$, 17, $u_{i(j+1)}$ is passed from C to D,

Table 1 Processor utilization for parallel FCM

Time cycle	Processor							
	A			*B*	*C*			*D*
	1	2	3	4	5	6	7	8
1	↑	↑	↑					
2	↓	↓	↓					
3				↑				
4				↑				
5	↑	↑	↑	↓	↑	↑	↑	
6	↓	↓	↓		↑	↑	↑	
7					↑	↑	↑	
8				↑	↓	↓	↓	
9				↑	↓	↓	↓	
10	↑	↑	↑	↓	↑	↑	↑	↑
11	↓	↓	↓		↑	↑	↑	↑
12	↑	↑	↑		↑	↑	↑	↓
13	↑	↑	↑					
14				↑	↓	↓	↓	
15				↑	↓	↓	↓	
16	↓	↓	↓	↓				↑
17	↓	↓	↓					↑
18	↑	↑	↑		↑	↑	↑	↓
19	↓	↓	↓		↑	↑	↑	
20	↑	↑	↑		↑	↑	↑	
21	↑	↑	↑					
				...				

while $u_{ij}{}^m$ and $u_{ij}{}^m x_j$ are summed at A, respectively. Simultaneously, at $t=16$, $H_a{}^{(j+2)}$ is computed at B. At $t=18$, $\vec{\mathbf{x}}_{j+2}$ is passed from A to C, $\vec{\mathbf{x}}_{j+3}$ is input to A and $u_{i(j+1)}$ is summed at D. At $t=19$, $h_i^{(j+2)}$ is forwarded from A to C, while $h_i^{(j+3)}$ is computed at A. At $t=20$, $H_a{}^{(j+2)}$ is transmitted from B to C. Simultaneously, at $t=20, 21$, $u_{i(j+1)}{}^m$ and $u_{i(j+1)}{}^m x_{j+1}$ are fed back from C to A. The process in cycles 14 to 21 repeat themselves for the entire pattern set presentation.

After pattern $\vec{\mathbf{x}}_N$ is input at A, and the corresponding u_{iN} is computed at C and passed to D for the checksum, this pattern set presentation (from $j=1,\ldots,$ N) is termed complete. At the end of this (say at $t=T$), processor D checks whether the change in total membership values, summed over this set, is greater than ϵ. At this stage, an additional cycle is required by A for updating the cluster centers by Eq. 4.

3.3 Performance evaluation

The number of clock cycles required is $T_c' = 13 + 8\times(N-2) + 2 = 8N-1$. The whole process is repeated (say, K_c times) until a stopping signal from D terminates the updating of cluster centers, and, hence, the fuzzy clustering. The total number of cycles consumed is, therefore, $T_c = K_c \times T_c' + 1$.

The number of clock cycles required by a sequential processing element (PE) is $T_{cs}' = 3n(9N+2)c$. Hence, for K_c repetitions, $T_{cs} = K_c \times T_{cs}' + 1$. Therefore, we have speedup:

$$S_c = \frac{T_{cs}}{T_c} \simeq \frac{3n(9N+2)c}{8N-1} \tag{11}$$

and parallelization efficiency:

$$\eta_{P_c} = \frac{S_c}{\#\text{PE}} = \frac{3n(9N+2)c}{2(8N-1)(c+1)} \tag{12}$$

Latency, λ, is defined as the number of cycles between input vector presentation and the appearance of output [4]. Here, $\lambda_c = 6$.

4 Parallel architecture for output layer

In this section, we concentrate on the parallel implementation of the hidden-output layer of the FRBF architecture. As in Sect. 3, there are c processors, each of types A, C, and one each of types B, D. The interconnection buses simultaneously transmit $3n$-dimensional data. The parallel design and processor utilization are described, followed by a quantitative evaluation of the performance.

4.1 Design

The schematic design for implementing the hidden-output layer architecture of the FRBF network is provided in Fig. 3. The interconnections between the processors of types A, B, C, and D are now different, implemented using reconfigurable programmable networks. These are modeled by switch lattice connections between the processors that can be changed to get the required design. As before, the $3n$-dimensional data, $\vec{\mathbf{x}}_j$, is fed in parallel to the c processors of type A, and $h_i^{(j)}$ is computed w.r.t. the cluster centers, v_i, already stored therein.

Then, the $h_i^{(j)}$s are transmitted from A to C, and, finally, a copy of these from C to B, where they are summed to generate $H_a{}^{(j)}$ by Eq. 9. Each processor C, additionally, stores the corresponding weight vector, $\vec{\mathbf{W}}_i$, from hidden node i to the l output nodes. Initially, these are set as small random numbers, which are then gradually updated during learning. Processor C computes the product $h_i^{(j)} W_{ik}$ in the numerator of Eq. 8 and passes it on to B, where these are summed to compute $y_k{}^{(j)}$. This process is repeated for $k=1,\ldots,l$ output nodes.

$H_a{}^{(j)}$ is passed from B to D. Processor D, additionally, stores the desired output, $*\vec{\mathbf{y}}^{(j)}$, which is compared with the $\vec{\mathbf{y}}^{(j)}$ transmitted from B. The error component, $\vec{\mathbf{E}}^{(j)} = \frac{\eta\left(*\vec{\mathbf{y}}^{(j)} - \vec{\mathbf{y}}^{(j)}\right)}{H_a^{(j)}}$, of Eq. 10 is computed at D.

The error component, $\vec{\mathbf{E}}^{(j)}$, is fed back from D to C. Here, the weight update, $\vec{\mathbf{W}}_k^{(j)}$, is computed on-line, using Eq. 10, by multiplying it with $h_i^{(j)}$ residing there. The change in total error, at the end of the training, is also computed at D. This determines the termination of weight updating.

4.2 Processor utilization

Table 2 provides the processor utilization diagram for the proposed output layer design of the FRBF. The meaning of the arrows and the number of hidden nodes are the same as in Table 1. Here, processors 1, 2, 3 and 4, 5, 6 refer to types A and C, respectively, while processors 7 and 8 correspond to B and D, respectively. It is assumed, for the ease of explaining, that there are only two output nodes ($l=2$). As before, some clock cycles are wasted for synchronization while transferring the data from parallel processors C to the single processor B.

At time cycle $t=1$, let pattern $\vec{\mathbf{x}}_j$ be input to processors A. At $t=2$, $h_i^{(j)}$ is computed at A. This is forwarded to C at $t=3$. At $t=4$, the product, $h_i^{(j)} W_{i1}$, is computed at C. Then, $h_i^{(j)}$ and the product are transmitted from C to B at $t=5$, 6 and $t=8$, 9, respectively, while $H_a{}^{(j)}$ is computed in B at $t=7$ and passed to D at $t=8$. Simultaneously, at $t=9$, $\vec{\mathbf{x}}_{j+1}$ is input at A.

At $t=10$, $h_i^{(j+1)}$ and $h_i^{(j)} W_{i2}$ are computed at A and C, respectively. At $t=11$, $h_i^{(j+1)}$ is forwarded from A to C and $y_1{}^{(j)}$ is computed at B. Then, $h_i^{(j)} W_{i2}$ is propagated from C to B at $t=12$, 13, while $y_1{}^{(j)}$ is passed from B to D at $t=12$. Simultaneously, at $t=13$, $E_1{}^{(j)}$ is calculated at D. A copy of this is fed back from D to C at $t=14$, while the total error is summed at D and $y_2{}^{(j)}$ is

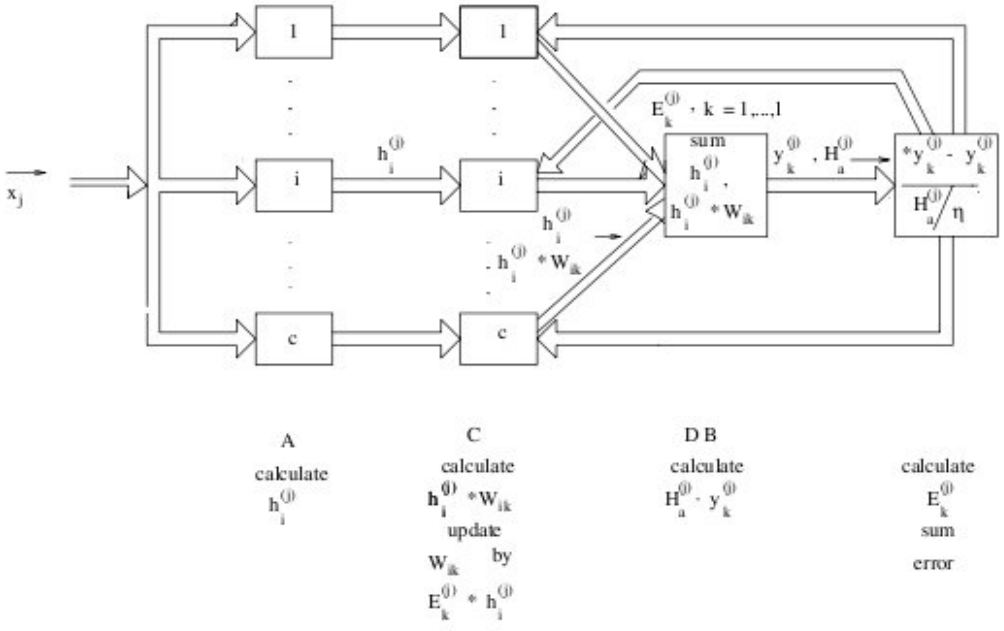

Fig. 3 Parallel architecture for output layer

Table 2 Processor utilization for output layer of the FRBF

Time cycle	Processor							
	A			*C*			*B*	*D*
	1	2	3	4	5	6	7	8
1	↑	↑	↑					
2	↓	↓	↓					
3				↑	↑	↑		
4				↓	↓	↓		
5							↑	
6							↑	
7							↓	
8							↑	↑
9	↑	↑	↑				↑	
10	↓	↓	↓	↓	↓	↓		
11				↑	↑	↑	↓	
12							↑	↑
13							↑	↓
14				↑	↑	↑	↓	↓
15				↓	↓	↓		↑
16							↑	↓
17				↑	↑	↑	↑	↓
18				↓	↓	↓	↓	
19							↑	↑
20							↑	
21	↑	↑	↑	↓	↓	↓		
				...				

computed at B. At $t=15$, weight W_{i1} is updated at C and $y_2^{(j)}$ is transferred from B to D. At $t=16$, 17, a copy of $h_i^{(j+1)}$ is forwarded from C to B. Simultaneously, $E_2^{(j)}$ is computed in D, a copy of it is fed back to C, and the total error is summed at D. At $t=18$, $h_i^{(j+1)}W_{i1}$ and $H_a^{(j+1)}$ are computed at C and B, respectively. Then, $h_i^{(j+1)}W_{i1}$ is passed from C to B at $t=19$, 20, while, simultaneously, $H_a^{(j+1)}$ is forwarded to D at $t=19$. At time cycle $t=21$, $\vec{\mathbf{x}}_{j+2}$ is input at A and weight W_{i2} is updated at C.

The process in cycles 10 to 21 repeats for the entire pattern set, $j=1,\ldots,N$. The total error, $E_k^{(j)}$, for all patterns, j, in a single presentation and over all output nodes, k, is now available at D. At the end of this, say at $t=T$, processor D checks whether the change in this total error is less than a threshold δ. This determines the stopping criterion for the FRBF training.

4.3 Performance evaluation

The number of clock cycles required is $T_o' = 9+12\times(N-1)+1 = 2(6N-1)$. Let the training phase be repeated for K_o iterations. The total number of cycles consumed is, therefore, $T_o = K_o\ T_o'$.

Considering a purely sequential mode of operation, the number of clock cycles required by a single PE is $T_{os}' = 3n(13N+1)c$. Hence, for K_o repetitions, $T_{os} = K_o \times T_{os}'$. Therefore, we have speedup:

$$S_o = \frac{T_{os}}{T_o} = \frac{3n(13N+1)c}{2(6N-1)} \tag{13}$$

parallelization efficiency:

$$\eta_{P_o} = \frac{S_o}{\#\text{PE}} = \frac{3n(13N+1)c}{4(6N-1)(c+1)} \tag{14}$$

and latency $\lambda_o = 11$. It is, however, difficult to express these measures in terms of the number of output nodes involved, mainly due to the high degree of overlap between the forward and backward passes.

5 Conclusions and discussion

In this article, we have described a parallel hardware implementation of the FRBF network that involves an

integration of the FCM with the conventional RBF network. The effectiveness of the design, for both FCM clustering at the hidden layer and the weight updating at the output layer of the FRBF network, has been provided in terms of processor utilization. Quantitative measures like speedup, parallelization efficiency, and latency have been used to evaluate the performance of the system.

The synchronization overhead, while transmitting data from A or C to B or D, is counterbalanced by the possibility of simultaneous processing of two data patterns. It is seen that the weight update in the hidden-output layer is done in alternation with the forward processing of the next data pattern. This is permissible because the weight update of (say) the second output node is done for the jth pattern, while the weights of (say) the first output node are used for output computation corresponding to the $(j+1)$th pattern.

The design has a high throughput because of the utilization of almost every free cycle of the parallel processors. It is suggestive of tightly packed processing elements with a high amount of processing per unit time. However, the use of barriers would be able to stop all the parallel processors if one of them happens to consume some extra time in any computation. This would help the system avoid any error in the output in such conditions.

It is possible to enhance the performance of the system by introducing l registers (corresponding to l output nodes) for the systolic processing of weight and error vectors at C and D. This would result in reduced processing time, at the expense of an associated increased hardware cost. An investigation in this direction is currently under way.

References

1. Bezdek JC (1981) Pattern recognition with fuzzy objective function algorithms. Plenum Press, New York
2. Hwang K, Briggs F (1986) Computer architecture and parallel processing. McGraw-Hill, New York
3. Jones SR, Sammut KM, Hunter J (1994) Learning in linear systolic neural network engines: analysis and implementation. IEEE Trans Neural Netw 5:584–593
4. Lopez PI (1996) Programmable VLSI systolic processors for neural network and matrix computations. PhD thesis, Ecole Polytechnique Federale de Lausanne, Switzerland
5. Mitra S, Basak J (2001) FRBF: a fuzzy radial basis function network. Neural Comput Appl 10:244–252
6. Moody J, Darken CJ (1989) Fast learning in networks of locally-tuned processing units. Neural Comput 1:281–294
7. Naylor D, Jones S, Myers D (1995) Backpropagation in linear arrays—a performance analysis and optimization. IEEE Trans Neural Netw 6:583–595
8. Pal SK, Mitra S (1999) Neuro-fuzzy pattern recognition: methods in soft computing. Wiley, New York
9. Zadeh LA (1994) Fuzzy logic neural networks and soft computing. Commun ACM 37:77–84
10. Zhang D, Pal SK (2000) A fuzzy clustering neural networks (FCNs) system design methodology. IEEE Trans Neural Netw 11:1174–1177