Parallel Implementation of Memory Neuron Network for Identification of Dynamical System

S. Suresh*, S. N. Omkar, V Mani
Department of Aerospace Engineering, Indian Institute of Science Bangalore 560012. India
*suresh99@aero.iisc.ernet.in

Abstract

This paper discusses parallel algorithm of memory neuron networks for identification of nonlinear dynamical systems. These are a class of recurrent networks obtained by adding trainable temporal elements to feed-forward networks that makes the output history-sensitive. By virtue of this capability, these networks can identify nonlinear dynamical systems without having to be explicitly fed past inputs and outputs. Artificial neural networks (ANN) consist of enormous number of massively interconnected nonlinear computational elements (neuron) and they are inherently parallel in nature. This paper presents an analysis of neuron parallelism or vertical slicing using Message Passing Interface (MPI) library routines. The results show that the parallel efficiency increase with increase in network size and also we can get linear speed up if the number of processors are increased.

1 Introduction

There has been considerable interest in the past few years in the application of ANN for identification and control of nonlinear dynamical systems[3,4]. In this paper we present a class of recurrent neural networks called Memory Neuron Networks (MNN)[6] as general model for identification of nonlinear dynamical systems. These networks are obtained by adding some trainable temporal elements to the feedforward networks. The main attraction of these networks is that they have trainable internal memory and hence can directly model dynamical systems.

Identification of dynamical nonlinear systems is recognition of temporal patterns of the systems. The output of the physical systems to be modeled is a function of past inputs and outputs as well. In general, the identification problem is complicated because of the model being used (e.g. ANN), should have some internal memory. One can include dynamics directly into the network structure so that we can learn nonlinear dynamical systems without assuming much knowledge of the systems. For this, what we need are networks with some “internal memory” and learning algorithms for such networks. Due to proven ability of feedforward networks to model nonlinear systems, it seems logical to explore recurrent networks that are closely related to multiplayer feedforward networks.

In this paper we describe a recurrent network model with internal memory called the memory neuron network[6,7]. Here each unit of neuron has, associated with it a memory neuron whose single scalar output summarizes the history of the past activations of that unit. These memory neurons, or more precisely the weight of connection into them represent trainable dynamical elements of the model. Since the connection between a unit and its memory neuron involve feedback loops, the overall network is now

 © UNIVERSITIES PRESS (INDIA) PRIVATE LIMITED

a recurrent one. This memory neuron and trainable temporal elements are sufficient for identification of dynamical systems. Backpropagation through time (BPTT) is used for training MNN[7]. However, the BPTT algorithm is computationally intensive. As a result, considerable interest has been focused on studying parallel implementations[10].

Two main paradigms are used for parallelising the BPTT algorithm. They are network-based parallelism and the training-set parallelism[8]. In network-based parallelism, the neural networks are partitioned and distributed among the processors. This approach is called network slicing[2]. In training-set parallelism, the training epochs are distributed to all the processors. Network-based parallelism uses online learning in which weights are updated after each pattern; Training-set parallelism uses batch learning, in which weights are updated after all the patterns are presented. Training-set parallelism converges slowly, then the network-based parallelism[10].

In this paper we are using network-based parallelism and Message Passing Interface (MPI) library routines for communications between the processors. The MPI library routine does not depend on the underlying hardware that supports it. So we can use this program in heterogeneous architecture. Heterogeneous processors networks consist of processors with different speed, memory and cost. They offer very powerful environment for parallel computing. The primary aim of this paper is to illustrate the utility of parallel algorithm to speed up the training of MNN.

The paper is organized as follows: section 2 gives brief description about MNN for identification of dynamical systems. Section 3 deals about parallelisation and MPI library. Section 4 is on evaluation of parallel algorithm, and the results are discussed in Section 5.

2 Memory Neuron Networks

In this section we describe the structure of the network that we use and the associated learning algorithm. The network and the learning algorithm we use are similar to the one described in[7]. It may be pointed out that it is possible to use other incremental learning algorithms, e.g., RTRL[9], ALOPEX[5] with this network.

2.1 Network structure

The architecture of the memory neuron network is shown in Fig. 1. The structure is the same as a feedforward ANN except for the memory neurons (shown by small filled circle) attached to each unit in the network (shown by large open circle). To distinguish these two types of units, we use term network neuron and memory neuron. As can be seen from the figure, at each level of the network except the output, each of the network neuron has exactly one memory neuron connected to it.

The memory neuron takes its input from the corresponding network neuron and it also has a self-feedback as shown in inset of Fig. 1. This leads to accumulation of past data of the network neuron in the memory neuron. All the network neuron and memory neuron send their outputs to the network neurons of the next level. In the output layer, each network neuron can have cascade of memory neuron. Figure 1 shows a network with two input nodes, one output node and a single hidden layer.

2.2 Identification

Here we discuss the example we had taken for simulation. We are using backpropagation through time algorithm as explained in Sastry et al.[7]. We explain MNN for the identification of dynamical systems. Denote $u_i(k)$ and $y_p(k)$ as input and output of a plant.

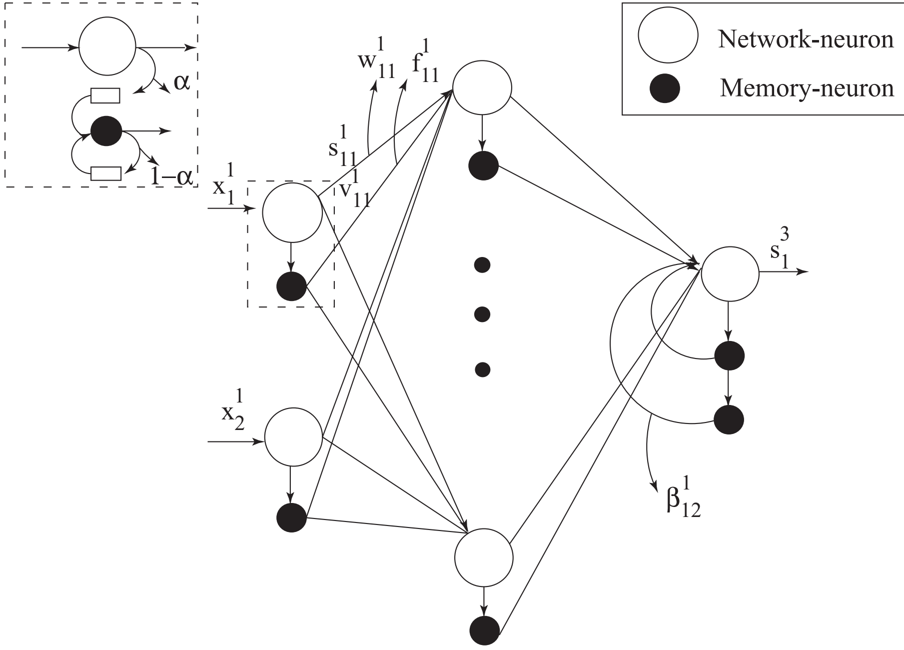

Fig. 1 Architecture of memory neuron network

Let $u_i(k)$ be the input fed into the network at any time k and let $y_p(k)$ be the output of the network at any time k. $y_p(k)$ is the teaching signal to the network. We are using series-parallel model for identification[6]. Series-parallel model needs actual input $u_i(k)$ and past output of the plant $y_p(k-1)$ as input to the network shown in the Fig. 2. Now we have,

$$\hat{y}_p = F\big(u(k), u(k-1), \ldots, y_p(k), \ldots, \hat{y}_p(k-1), \ldots\big). \tag{1}$$

To identify an M-input and a P-output plant, we will use a network with $M + P$ inputs and P outputs. This will be a case irrespective of the order of the system. We shall use the actual output of the plant at each instant as teaching signals. Here we are using 25-inputs and a 25-output network, with hidden nodes varying from 250 to 200 nodes.

We use 66 000, 77 000 time-steps for training the network with longer training sequence for more complex plants. We train the network for 2000 iteration with zero inputs; then for two-third of the remaining epochs the input is iid sequence uniform over $[-2, 2]$, and for the remaining training time, the input is single sinusoid signal given by sin $(\Pi k/45)$. After training we compare the output of the network with that of the plant on a test signal for 1000 iteration. Our test signal consist of mixture of sinusoid and constant inputs,

$$\begin{aligned} U(k) &= \sin\left(\frac{\Pi k}{25}\right), && k < 250 \\ &= 1.0 && 250 \le k \le 500 \\ &= -1.0 && 500 \le k \le 750 \end{aligned} \tag{2}$$

$$= 0.3 \sin\left[\frac{\Pi k}{25}\right] + 0.1 \sin\left[\frac{\Pi k}{32}\right] + 0.6 \sin\left[\frac{\Pi k}{10}\right] \qquad 750 \leq k \leq 1000.$$

Consider the following example:

$$y_p(k+1) = f\big(y(k), y(k-1), y(k-2), u_i(k), u_i(k-1)\big) \tag{3}$$

where,

$$f\left(x_1, x_2, x_3, x_4, x_5\right) = \frac{x_1 x_2 x_3 x_5 \left(x_3 - 1\right) + x_4}{\left(1 + x_3^2 + x_2^2\right)}.$$

The output of the plant depends on the three previous outputs and two past inputs. Though the function F has five arguments, we need $u(k+1)$ and $y_p(k)$ to get $y_p(k+1)$. The test inputs are given in equation (3). We need the past three outputs and the present and past input as an argument in the function.

3 Message Passing Interface Library

Message passing is a programming paradigm used widely on parallel computer, especially scalable parallel computer with distributed memory, and on network workstations. MPI defines both the syntax

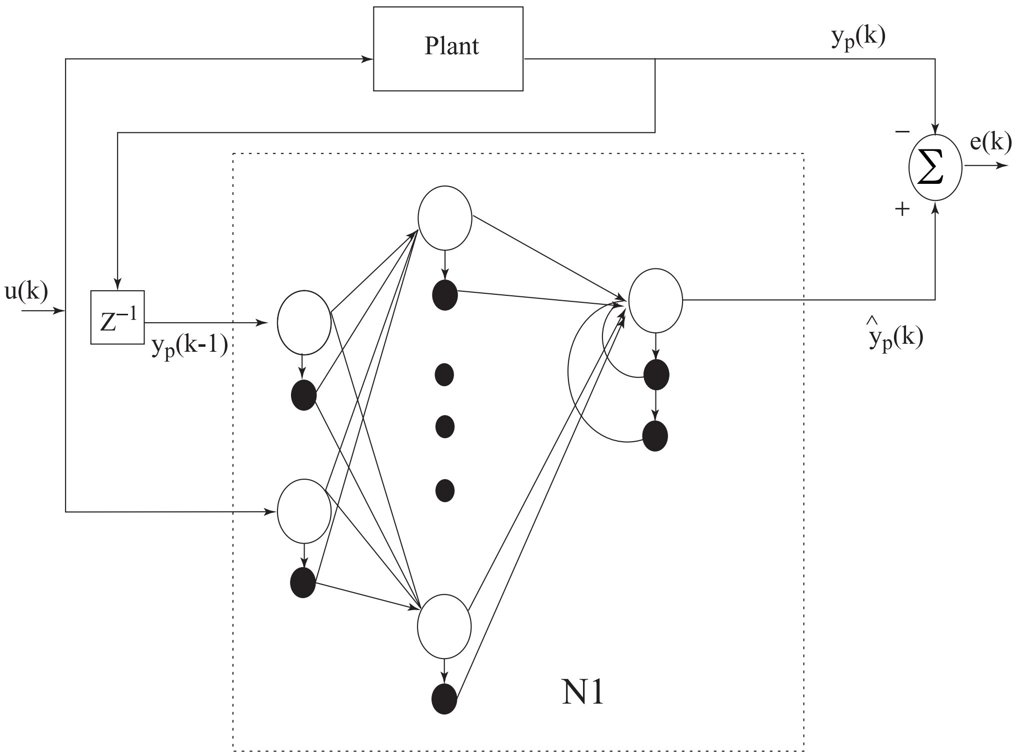

Fig. 2 Series parallel identification model with memory neuron network. Current input into the plant and the most recent output of the plant are fed into the network. The error $e(k)$ is used for learning the network parameters.

and the semantics of a standard core of library routines, which is not only useful for a wide range of users but is also efficiently implementable on a wide range of computers.

MPI is a library, not a language. It specifies the names, calling sequence and result of the functions to be called from C or FORTRAN program. Portable parallel programming with MPI is designed to accelerate the development of parallel application programs. The message-passing model fits well on separate processors connected by a fast or slow communication networks. MPI does not depend on the underlying hardware that supports it. The major goal of MPI is high degree of portability across different machines

Another type of compatibility offered by MPI is the ability to run transparently on heterogeneous environment, that is, a collection of processors with distinct architectures. The MPI implementation will convert the data automatically and do the necessary data conversion and utilize the correct communication protocol[2]. The MPI is designed to encourage overlap of communication and computation so as to take advantage of intelligent communication agents and to hide communication latencies.

The basic communication mechanism of MPI is the transmission of data between a pair of processes, one side sending, the other receiving. This is called point-to-point communication. For training artificial neural networks, we have to transmit the data to all the processors. If we are using point-to-point communication, it will increase the communication overhead. We have to use collective communication. Basically, collective communications transmit data among all the processors in a group specified by an intracommunicator object[2]. This will considerably reduce the communication overhead. There are two types of collective communications. They are data movement operation and collective computation operation. Data movement operation is used to rearrange data among the processors. Collective computation operation uses operations like sum, logical AND, max, min, user-defined function, etc., to process the data and then transmit to all the processors[2].

4 Algorithm Parallelisation

Network-slicing concept can be used to parallelise the backpropagation algorithm in IBM-SP machines. It consists of a master processor where all the weights are initialized. Each processor will have equal number of neurons in the hidden layer. All the processors are connected to all input and output nodes. Each will evaluate the output in the forward step and broadcast to all other processors. Similarly, during backward phase the error in the hidden layer are broadcast to all the other processors so as to update the weight connection between the input and hidden layer. This will increase the communication overhead. To avoid this we are using collective communication[2].

Figure 3 shows a memory neuron network with six hidden nodes. For the purpose of illustration we are using two processors. Both the processors have input nodes A and output node B. The block C is kept in one processor and D kept in another. We need two communications between the processors for each epoch—one for forward pass to exchange the output and the other for backward pass to exchange the error in hidden nodes to update the weights.

5 Results and Discussion

In this paper we have suggested a parallel algorithm of memory neuron networks for identification for nonlinear dynamical systems. MPI communication features on an IBM SP machines were evaluated in

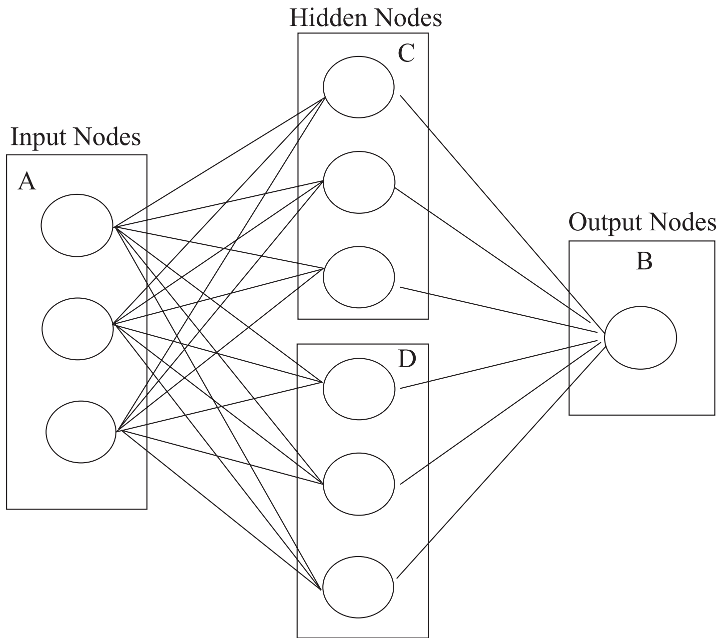

Fig. 3 Parallel split up for MNN with one input and one output with two processors.

Table 1 Time (seconds) taken for training the MNN with different number of processors

Number of hidden nodes	No. of processors			
	One	Two	Three	Four
250	2234	1151	811	650
240	2098	1110	771	618
230	1964	1066	743	583
220	1895	1038	717	576
210	1723	976	689	541
210	1614	930	661	513

order to implement this algorithm in an efficient way. For the testing purpose we have taken 25 inputs and 25-output plant with varying hidden nodes similar to the equation (3) with interdependency among the past inputs and outputs. They are trained and tested as explained in the section 2. Table 1 shows the time taken to train the network in different processors. In general total time taken to train the neural network with N number of processors is $1/N$ times that of the time taken by sequential (one) processor. The same can be observed from the Table 1.

Next a study is done on parallel efficiency and speedup. They are defined as follows,

$$\text{Parallelisation efficiency} = \frac{S}{MP}; \tag{4}$$

$$\text{Speedup} = \frac{S}{P}; \tag{5}$$

Where M is the number of processors, P the time taken by the program to run in parallel machine, and S the time taken by the program to run in sequential machine.

Figure 4 shows the variation of parallel efficiency with the increase in number of hidden nodes for 2, 3 and 4 processors. It shows that parallel efficiency increases with increase in network parameter. Figure 5 show the variation of speedup with increase in number of processors for 250, 240 and 230 hidden nodes. As can be seen from figure, we can achieve linear speedup with increase in number of processors.

We can achieve linear speedup if we increase the number of processors. But at the same time it will reduce the parallelisation efficiency. If we increase the number of processors then communication latency will become higher than the computation done by each processor. This will lead to reduction in speedup. In order to avoid reduction in speedup we should use overlapped communication and computation. But in network parallelism it is not possible to have overlapped communication and computation because of the communication required between the processors at the same time. This shows that parallelisation will be effective only when the computation requirement is more, i.e., when the network parameters are high.

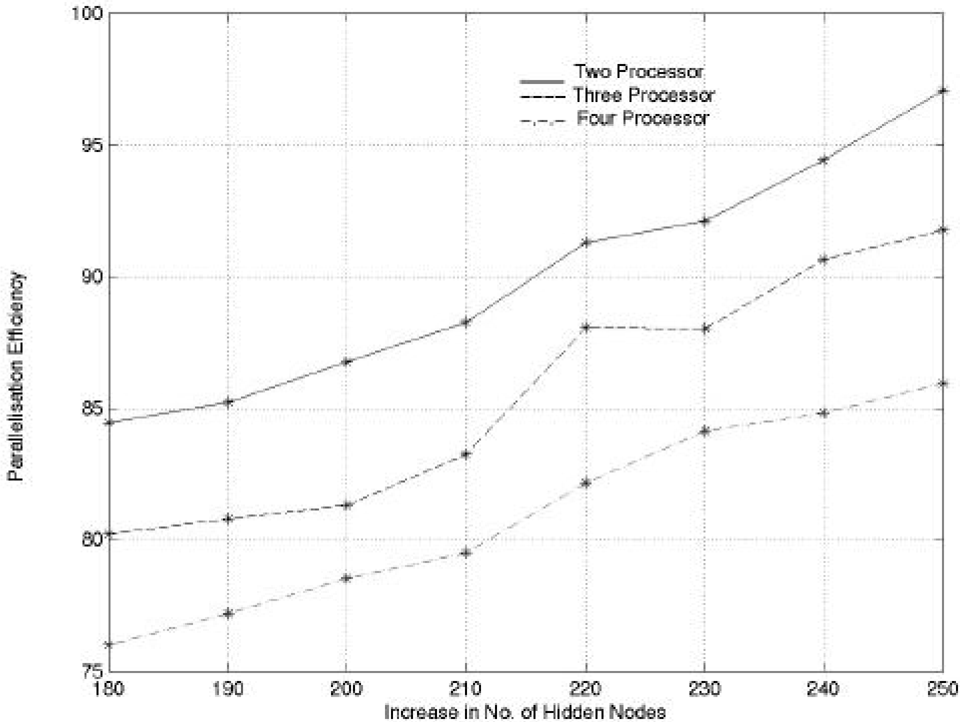

Fig. 4 Efficiency versus hidden nodes.

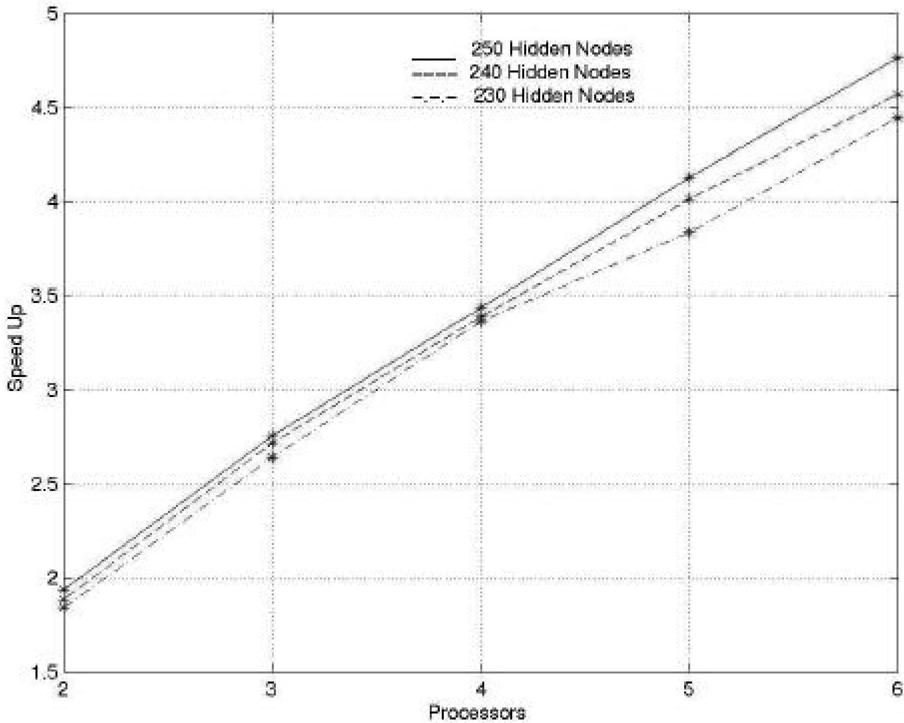

Fig. 5 Speed up versus number of processors.

Conclusion

Identification of nonlinear dynamical system using memory neuron network is parallelised effectively. Collective communication is used to implement network parallelism. This will considerably reduce the communication overhead. We can achieve linear speedup if we increase the number of processor, and also increase the efficiency with increase in network size.

References

[1] Danese, G., de cotto, I. and Leporati, F., A parallel processor for neural networks, *IEEE Transaction on Neural Networks*, vol. 1(2), 1999.

[2] Mars Snir, Steve otto, Steven Huss-Ledermen, David Walker and Jack Dongarra, *Message Passing Interface- The Complete Reference*, MIT Press London England.

[3] Narendra, K.S. and Parthasarathy, K., Identification and control of dynamical systems using neural network, *IEEE transactions on Neural Networks*, vol.1(1), pp. 4–27, 1990.

[4] Narendara, K.S. and Parthasarathy, K., Gradient method for optimization of dynamical systems containing neural networks, *IEEE Transaction on Neural Networks*, vol. 2(2), pp. 252–262, 1991.

[5] Pandya, Abhi S., Shen, Ercan and Hsu, Sam, Buffer allocation optimization in ATM switching networks using ALOPEX algorithm, *Neurocomputing*, vol. 24, pp. 1–11, 1999.

[6] Poddar, P. and Unnikrishnan, K.P., Memory neuron networks:A prolegomenon, Tech Rep. GMR-7493, General Motors Research Laboratories, 1991.

[7] Sastry, P.S., Santharam, P.S. and Unnikrishnan, K.P., Memory neuron networks for identification and control of dynamical systems, *IEEE Transaction on Neural Networks*, vol.3(2), pp. 305–319, 1994.

[8] Sundararajan, N and Saratchandran, P., *Parallel Architecture For Artificial Neural Networks-Paradigms And Implementation*, IEEE Computer Society Press.

[9] Williams, R.J. and Zipser, D., A learning algorithm for continually running fully recurrent networks, *Neural computation*, vol.1, pp. 270–280, 1989.

[10] Witbrock, M. and.Zagha, M., Backpropagation learning on IBM GF11, in *Parallel Digital Implementation of Neural Networks*, Przytula, and Prasanna, V.K., (eds.), ch.3, pp. 77–104, Englewood Cliffs, NJ:PTR Prentice Hall, 1993.