A Distributed Backpropagation Algorithm of Neural Networks on Distributed-Memory Multiprocessors*

Hyunsoo Yoon, Jong H. Nang, and S. R. Maeng

Center for Artificial Intelligence Research &
Department of Computer Science
Korea Advanced Institute of Science and Technology
P.O. Box 150 Cheongryang, Seoul 130-650, Korea
e-mail : hyoon@sorak.kaist.ac.kr

Abstract

In this paper, we present a distributed backpropagation algorithm of a fully connected multilayered neural network on a distributed-memory multiprocessor system. In our system, the neurons on each layer are partitioned into p disjoint sets and each set is mapped on a processor of a p-processor system. A fully distributed backpropagation algorithm, necessary communication pattern among the processors, and their time/space complexities are investigated. The p-processor speed-up of the backpropagation algorithm over a single processor is analyzed which can be used as a basis in determining the most cost-effective or optimal number of processors. The experimental results with a network of Transputers are also presented to confirm our model and analysis.

1. Introduction

Recently, extensive research efforts are being devoted to the theory, implementation, and application of neural networks. However, since they require huge amounts of computational resources, and current technology is not mature enough to implement a large neural network directly in hardware, digital computer simulations are the primary method of implementing, experimenting, and applying neural networks. Because the size of neural networks is typically conceived as being very large, the ability to simulate them is generally limited by the speed and storage capacity of digital computers.

One natural way to overcome the time and space limit of neural network simulations is to exploit the parallel processing technique, and as such, there have been several research efforts to implement a fast simulator on commercially available parallel computers such as the Connection Machine [4], the Warp [11], the MPP [8], and the BBN Butterfly [5].

In this paper, we investigate neural network simulations on a distributed-memory, message-passing multiprocessor (DMM) system such as the Intel Hypercube, Ncube, or the Inmos Transputer system which are more widely available than the aforementioned parallel computers. In particular, we investigate a fully connected multilayered network using the backpropagation learning algorithm, the most common and popular neural network [3], on the DMM. Simulating neural networks on a DMM have already been considered in some previous works. For example, [6] and [10] used the Transputer-based DMM, but their approaches are based on the systolic algorithms which are not well suited for the DMM nature. Also [7] and [2] considered using the DMM, but they assumed that neural networks can be partitioned into groups of neurons such that the connectivity within a group is much higher than the overall network connectivity, which is not well applicable to the fully connected networks.

In our model, a multilayered network is vertically partitioned into p subnetworks, and each subnetwork is mapped on a processor of the p-processor DMM. Based on this partitioning and mapping scheme, a fully distributed parallel backpropagation algorithm is derived, and the time/space complexities of it are also computed. The p-processor speed-up and space-reduction bound are also derived by comparing them with the single processor's case. The experimental results with 32 Transputers show that our algorithm and analysis are valid.

2. Multilayer Neural Networks

The neural network we are focusing is a fully connected multilayered neural network using the backpropagation learning algorithm which is the most popular one [3].

2.1 A Fully Connected Multilayer Network

A fully connected multilayered network consists of $L+1$ layers as shown in Figure 1. The first layer at $l=0$ is the input layer and contains n_0 neurons. Subsequent layers are labeled 1 $\leq l \leq L$; the l-th layer contains n_l neurons. Each neuron in a layer is connected to all neurons in the next layer. Associated with each neuron i on layer l is an activation value $a_i(l)$, and attached to each connection, connecting neuron i on layer l to neuron j on layer $l+1$, is a weight $w_{ji}(l,l+1)$.

2.2 The Backpropagation Learning Algorithm

The backpropagation learning algorithm [12], a kind of supervised learning one, consists of three passes; forward execution (Eq.(1)), backpropagation of the error (Eq.(2)), and weight update (Eq.(3)), where f is a nonlinear sigmoid function of the form of $f(x) = (1+e^{-x})^{-1}$, $\delta_i(l)$ is the error value of neuron i on layer l, $t_i(L)$ is the desired value of neuron i in the output layer, and η is a learning rate.

$$a_i(l)=f\left(\sum_{j=1}^{n_{l-1}} w_{ij}(l-1,l)\cdot a_j(l-1)\right),\ i=1,\ldots,n_l \text{ and } l=1,\ldots,L \quad (1)$$

* This work was supported in part by the Korean Science and Engineering Foundation, under Contract No. KOSEF 891-1105-003-1.

CH2908-2/90/0000/0358/$01.00 – 1990 IEEE

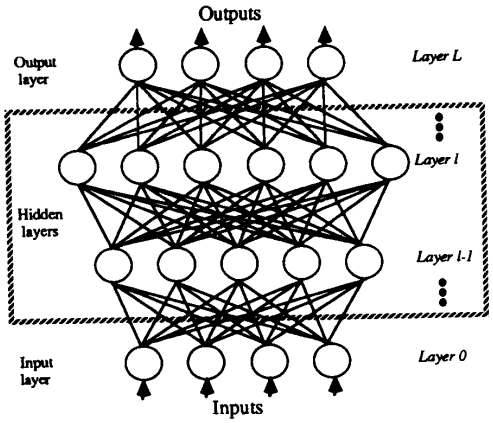


Figure 1: A Fully Connected Multilayered Network

$$\delta_i(l)=\begin{cases}[t_i(l)-a_i(l)]\cdot[a_i(l)\cdot(1-a_i(l))], & l=L \\ [\sum_{k=1}^{n_{l+1}}\delta_k(l+1)\cdot w_{ki}(l,l+1)]\cdot[a_i(l)\cdot(1-a_i(l))], & l=L-1,..,1\end{cases} \quad (2)$$

$$\Delta w_{ij}(l-1,l)=\eta\cdot\delta_i(l)\cdot a_j(l-1) \quad (3)$$

3. Multilayer Neural Network on a Distributed-Memory Multiprocessor

Though the full implementation, in which a dedicated processor is assigned to each neuron, provides a natural way to implement the neural network, it is difficult to change the neural network parameters such as neural network structure, neuron characteristics and learning rule. So at present, almost all neural network systems are being virtually implemented in which a processor simulates a subset (or whole) of network in a multiplexing fashion. The central issue in virtual implementation of neural networks on parallel computers is the way of mapping neural networks onto them so that the performance of simulation can be maximized.

3.1 Partitioning Strategy

A DMM in our model consists of p processors, has no shared memory, and communicates only by message passing via a point-to-point link. Each layer of n_l neurons is partitioned into p parts, and each part of n_l/p neurons is assigned to each processor as shown in Figure 2.

Each processor maintains in its local memory the activation values, the error values, and the input and output weight vectors of the assigned neurons. Since an input weight value of layer l is the output weight value of layer $l-1$, the same value is stored in two processors. Though this partitioning scheme results in the duplication of weight values, it allows to avoid the complex communication requirement during the execution of the distributed backpropagation algorithm. On the other hand,

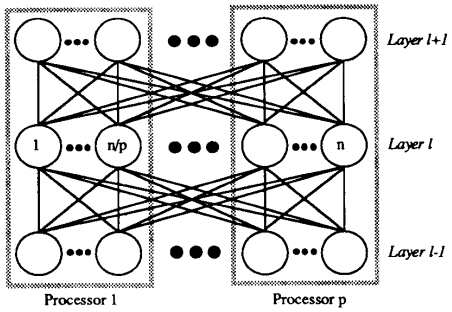
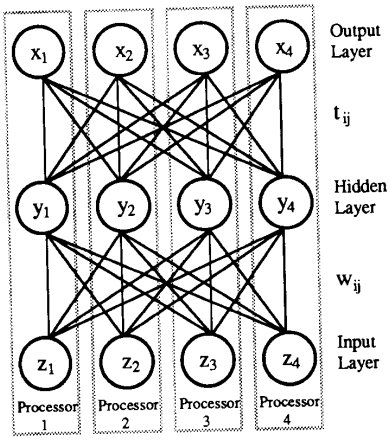
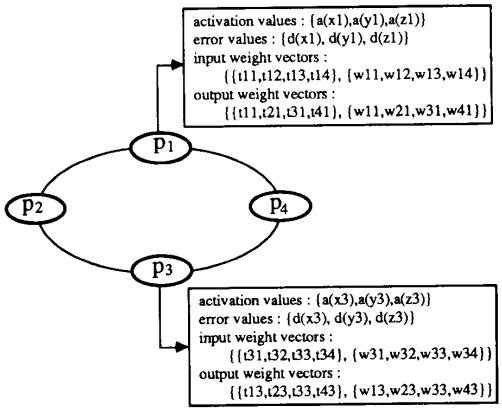


Figure 2: A Neural Network Partitioning Example

all the values of activations and errors are completely partitioned into p disjoint sets. Figure 3-(a) shows our partitioning scheme onto a 4-processor DMM when a fully connected network consists of 3 layers and each layer has 4 neurons, while Figure 3-(b) presents the distributions of activation values, error values, and weights when a DMM is connected using the ring topology.



(a) Network Partitioning



(b) Data Distribution

Figure 3: A Partitioning and Data Distribution Example

3.2 A Fully Distributed Backpropagation Algorithm

Each processor in a DMM basically executes the three passes expressed in Eq.(1)-(3), but some communications are necessary because data in layer l such as activation values ($a_i(l)$) and error values ($\delta_i(l)$) are fully distributed over p processors.

In the following, it is shown the necessary communication patterns and the distributed version of backpropagation algorithm for each pass, where it is assumed that $N_t(l)$ is a set of indices of neurons in layer l assigned to a processor p_t. Note that $|N_t(l)| = \frac{n_l}{p}$.

3.2.1 First Pass : Forward Execution

To compute the $a_i(l)$ using Eq.(1), the p_t should know all $a_j(l-1)$ stored in different processors. This requirement could be satisfied by the way in which every processor p_t broadcasts the $a_j(l-1)$ for each neuron $j \in N_t(l-1)$, and receives the $a_{j'}(l-1)$ for each neuron $j' \notin N_t(l-1)$. It is a process, called *all-to-all broadcasting* [9], whereby a set of messages, with a distinct message initially residing at each processor, is disseminated so that eventually a copy of each message comes to reside at each processor.

Once all-to-all broadcasting is completed, every processor is informed of all the necessary activation values of all other neurons in layer $l-1$, and can execute Eq.(1) for the assigned neurons independently. The forward pass of the distributed backpropagation algorithm for p_t is presented in Algorithm 1, in which *all-to-all-broadcast* is a procedure to broadcast and receive $a_j(l-1)$. (This procedure will be described in more detail in Section 3.3.)

Algorithm 1: A Distributed Forward Execution Algorithm

for l = 1 **to** L **do**
  /* Broadcasts and Receives $a_j(l-1)$ */
  for each neuron $j \in N_t(l-1)$ **do**
    all-to-all-broadcast($a_j(l-1)$) ;
  end_for
  /* Computes $a_i(l)$ */
  for each neuron $i \in N_t(l)$ **do**
    $a_i(l) = f\left(\sum_{j=1}^{n_{l-1}} w_{ij}(l-1,l) \cdot a_j(l-1)\right)$;
  end_for
end_for

3.2.2 Second and Third Pass: Backpropagation of the Error and Weight Update

The backpropagation of the error pass is similar to the 1st pass except for broadcasting $\delta_k(l+1)$ rather than $a_j(l-1)$. To compute the error value $\delta_i(l)$ where $i \in N_t(l)$, all $\delta_k(l+1)$ stored in different processors are required. This requirement could be satisfied by the way in which every processor p_t broadcasts the $\delta_k(l+1)$ for each neuron $k \in N_t(l+1)$, and receives $\delta_{k'}(l+1)$ for each neuron $k' \notin N_t$.

Now, let us explain how to update the weights stored in two different processors. A naive method to maintain the consistency of weight values is by communication, *i.e.*, one processor computes the new weight value and sends it to the other processor as presented in [13]. This kind of information exchange can be accomplished by *all-to-all personalized communication* [9], because computing and sending the new weights are necessary for all connections between the neurons mapped onto different processors. The all-to-all personalized communication is a process by which each processor sends a unique message to every other processors. However the time complexity of this method depends on the processor interconnection topologies[13]. Furthermore, because it is proportional to the number of connections in the neural networks, it is a very time-consuming process when the size of a neural network is very large.

Another method to maintain the consistency of weight values, used in our model, is a *recomputation method* with an additional all-to-all broadcasting of activation or error values. Assume that neuron k in layer $l+1$ and neuron i in layer l are mapped onto p_x and p_t, respectively. The difference of weight, $\Delta w_{ki}(l,l+1)$, can be computed in p_t if it knows the $\delta_k(l+1)$ stored in p_x, since the $w_{ki}(l,l+1)$ is kept locally in p_t as an output weight. Similarly, the difference of weight, $\Delta' w_{ki}(l,l+1)$, can also be computed in p_x independently if it knows the $a_i(l)$ stored in p_t, since the $w_{ki}(l,l+1)$ is kept locally in p_x as an input weight. Furthermore, $\Delta w_{ki}(l,l+1)$ and $\Delta' w_{ki}(l,l+1)$ are identical because p_t and p_x use the same values for computation, and thus the weight value consistency is guaranteed. When p_t sends $a_i(l)$ and p_x sends $\delta_k(l+1)$, they should be broadcasted because each neuron in layer l is connected to all neurons in layer $l+1$.

A distributed error backpropagation and weight updating scheme for p_t is presented in Algorithm 2.

Algorithm 2: A Distributed Error Backpropagation and Weight Update

/* Computes Error Values for Output Layer */
for each neuron $i \in N_t(L)$ **do**
  $\delta_i(L) = [t_i(L) - a_i(L)] \cdot [a_i(L) \cdot (1 - a_i(L))]$
end_for

for $l = L-1$ **to** 1 **do**
  /* Broadcasts and Receives $\delta_k(l+1)$ */
  for each neuron $k \in N_t(l+1)$ **do**
    all-to-all-broadcast($\delta_k(l+1)$) ;
  end_for

  /* Computes $\delta_i(l)$ */
  for each neuron $i \in N_t(l)$ **do**
    $\delta_i(l) = \left[\sum_{k=1}^{n_{l+1}} \delta_k(l+1) \cdot w_{ki}(l,l+1)\right] \cdot [a_i(l) \cdot (1 - a_i(l))]$
  end_for

  /* Updates Output Weight Vector $w_{ki}(l,l+1)$ */
  for each neuron $i \in N_t(l)$ **do**
    for k = 1 **to** n_{l+1} **do**
      $\Delta w_{ki}(l,l+1) = \eta \cdot \delta_k(l+1) \cdot a_i(l)$;
      $w_{ki}(l,l+1) = w_{ki}(l,l+1) + \Delta w_{ki}(l,l+1)$;
    end_for
  end_for

  /* Updates Input Weight Vector $w_{ij}(l-1,l)$ */
  /* Broadcasts and Receives $a_j(l-1)$ */
  for each neuron $j \in N_t(l-1)$ **do**
    all-to-all-broadcast($a_j(l-1)$) ;
  end_for

```
    /* Updates w_ij(l-1,l) */
    for each neuron i ∈ N_t(l) do
        for j = 1 to n_{l-1} do
            Δw_ij(l-1,l) = η·δ_i(l)·a_j(l-1) ;
            w_ij(l-1,l) = w_ij(l-1,l) + Δw_ij(l-1,l) ;
        end_for
    end_for
end_for
```

3.3 Time Complexity and Speed-up Ratio

The time required for a single processor to execute the backpropagation algorithm given in Eqs.(1)-(3) for a layer of n neurons is given by $T_1 = t_1 + t_2 + t_3$, where t_i is the time to execute the i-th pass. They could be expressed approximately as follows;

$$\begin{aligned} t_1 &= n\cdot(n\cdot M_a + F) \\ t_2 &= n\cdot(n\cdot M_a) \\ t_3 &= n\cdot(n\cdot M_a) \\ T_1 &= t_1 + t_2 + t_3 \\ &= n\cdot(3\cdot n\cdot M_a + F) \end{aligned} \tag{4}$$

where M_a is a multiply-and-add time of two floating-point numbers, and F is the time to evaluate the sigmoid function. It is assumed in evaluating the time complexity that $n_l = n$ for l=0,...,L for the sake of simplicity.

The time complexity of our distributed backpropagation algorithm for a layer of n neurons running on a p-processor DMM, T_p, are shown in Eq.(5). In Eq.(5), t_1' is the time to execute Algorithm 1, and t_2' is to execute Algorithm 2, and AAB(p) is the time to complete all-to-all broadcasting on the p-processor DMM. The factor $\frac{n}{p}$ is due to the fact that every processor is assigned $\frac{n}{p}$ neurons per layer.

$$\begin{aligned} t_1' &= \frac{n}{p}\cdot AAB(p) + \frac{n}{p}\cdot(n\cdot M_a + F) \\ t_2' &= \frac{n}{p}\cdot(2\cdot AAB(p) + 3\cdot n\cdot M_a) \\ T_p &= t_1' + t_2' \\ &= \frac{n}{p}\cdot(3\cdot AAB(P) + 4\cdot n\cdot M_a + F) \end{aligned} \tag{5}$$

Now, let us consider the all-to-all broadcasting algorithm and its time complexity on a p-processor DMM. To evaluate this, one must first assume the communication capability of each processor. In this paper we adopt the *one-port* communication [9] in which a processor can send and receive a unit of message on one of its ports, during each period of unit time. The port on which a processor sends and receives can be different. The unit time of communication for a processor sends and/or receives a unit of message is defined as C which includes the overhead of setting up the necessary communication mechanism and the actual message transfer time. The message unit is a word representing the floating-point number of the activation or error value.

In one-port communication, the lower bound for all-to-all broadcasting on a p-processor DMM is $(p-1)\cdot C$ since each processor needs to receive from every other processor, i.e., $(p-1)$ processors. This lower bound can be achieved in the ring topology. Note that as far as one-port communication is assumed, other topologies such as the complete connection, hypercube, and mesh which have more connections than the ring, do not perform all-to-all broadcasting better than the ring. If a processor can send and receive on its d-ports, $d>1$, concurrently during each time unit, more richly connected topologies would result in less all-to-all broadcasting time. However, the d-port communication is not realistic in most commercial message-passing multiprocessors, which is the reason why we adopted the assumption of one-port communication.

An AAB(p) algorithm for p_i when processors are connected by a ring topology is presented in Algorithm 3. In this algorithm, we assume that p processors in the ring-connected DMM are numbered from 0 to $p-1$ successively such that processor p_i is directly connected to $p_{(i+1) \bmod p}$ and $p_{(i-1) \bmod p}$.

Algorithm 3: An AAB(p) Algorithm on the Ring Topology

```
/* Sends an element (A[0]) and
   Receives p-1 elements (A[1]-A[p-1]) */
for i = 0 to p-2 do
    parbegin
        send A[i] to p_{i+1} ;
        receive A[i+1] from p_{i-1} ;
    parend
end_for
```

Using Eqs.(4) and (5), the p-processor speed-up ratio, $S(p) = \frac{T_1}{T_p}$, can be obtained as follows :

$$\begin{aligned} S(p) &= \frac{T_1}{T_p} \quad (by\ C = \Delta\cdot M_a,\ F = \theta\cdot M_a) \\ &= \frac{p\cdot(3\cdot n\cdot M_a + \theta\cdot M_a)}{3\cdot\Delta\cdot M_a\cdot(p-1) + 4\cdot n\cdot M_a + \theta\cdot M_a} \\ &= \frac{p\cdot(3\cdot n + \theta)}{3\cdot\Delta\cdot(p-1) + 4\cdot n + \theta} \end{aligned} \tag{6}$$

Once the learning is complete for a specific application, the application can be hard-wired, and the network may execute only the forward execution pass. The p-processor speed-up for forward execution, $S'(p)$, can be obtained as follows;

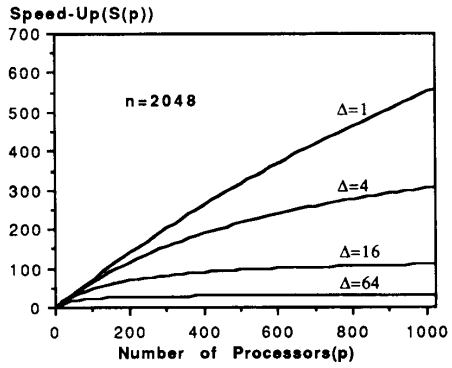
$$S'(p) = \frac{t_1}{t_1'} = \frac{p\cdot(n+\theta)}{\Delta\cdot(p-1) + n + \theta} \tag{7}$$

If the size of a neural network is extremely larger than p, the $S(p)$ is $\frac{3}{4}\cdot p$ and the $S'(p)$ is p in a p-processor DMM ;

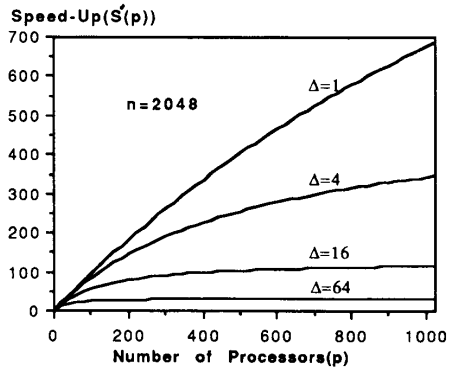
$$\begin{aligned} \lim_{n\to\infty} S(p) &= \frac{3}{4}\cdot p \\ \lim_{n\to\infty} S'(p) &= p \end{aligned}$$

The reason that $S(p)$ is not p but $\frac{3}{4}\cdot p$ when p processors are used, comes from the fact that the same computations for weight updates are performed twice in weight update pass.

In Eqs.(6) and (7), the most important parameter is the communication/computation ratio Δ whose value lies between 0.5 ~ 256 [1]. The theoretical learning speed-up aspects for various Δ values are shown in Figure 4-(a) graphically, where n = 2048 neurons/layer and θ=40. The theoretical forward execution speed-up is also shown in Figure 4-(b) under the same assumptions.



(a) The p-processor Speed-up of the Backpropagation Algorithm



(b) The p-processor Speed-up of the Forward Pass of the Backpropagation Algorithm

Figure 4: The p-processor speed-up when n=2048

For the distributed simulation of a multilayered neural network, it is seen from Figure 4 that there is a cost-effective number of processors depending on the Δ values, such that though more processors are added to the simulation, the speed-up ratio is not increased as much.

3.4 Space Complexity

The total space requirement to execute the backpropagation algorithm for a fully connected neural network which consists of L layers of n neurons, on a single processor, M_1, could be computed as follows;

$$M_1 = n \cdot L + n \cdot L + n^2 \cdot (L-1)$$

The total space requirement of the distributed backpropagation algorithm for the same neural network on a p-processor DMM, M_p, could be computed as follows :

$$\begin{aligned} M_p &\approx n \cdot L + n \cdot L + 2 \cdot n^2 \cdot (L-1) + p \cdot n \\ &= n \cdot (2 \cdot L + p) + 2 \cdot n^2 \cdot (L-1) \end{aligned}$$

Since the M_p could be equally partitioned into p spaces in a p-processor DMM, the space-reduction ratio, $M(p)$, could be computed as follows ;

$$M(p) = \frac{M_1}{\frac{1}{p} \cdot M_p} = \frac{p \cdot (2 \cdot n \cdot L + n^2 \cdot (L-1))}{n \cdot (2 \cdot L + p) + 2 \cdot n^2 \cdot (L-1)}$$

If the size of a neural network becomes large, the space-reduction ratio approaches to $\frac{p}{2}$;

$$\lim_{n \to \infty} M(p) = \frac{p}{2}$$

4. Experimental Speed-up on a Network of Transputers

Our distributed backpropagation algorithm has been implemented on a network of Transputers, especially T414. The Transputer provides relatively small Δ value because the capabilities to communicate with other Transputers are embedded in its architecture, which motivated us to adopt the network of Transputers as our testbed system. The Δ value of Transputer is about 0.05 [14].

An experimental speed-up of our distributed backpropagation algorithm on a ring of 32 T414-17MHz are obtained. All benchmark applications consist of 3 layers, in which the neurons in adjacent layers are fully connected. The sizes of benchmark application are as follows :

Number (80 x 9 x 4) : 8-by-10 Numbers
Hangul (1600 x 50 x 256) : 40-by-40 Korean Characters
Num18x12 (216 x 216 x 216) : 18-by-12 Numbers

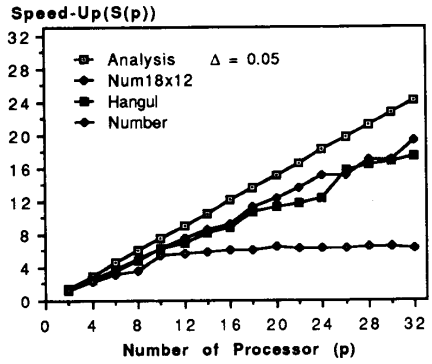
Figure 5-(a) shows the speed-up of the distributed backpropagation algorithm while Figure 5-(b) shows the speed-up of the forward pass of it.

The speed-up of *Hangul* and *Num18x12* applications are generally same as the analyzed speed-up, while *Number* is not. This aspect comes from the fact that *Number* application does not have enough neurons(or connections) to utilize the full power of multiprocessor system.

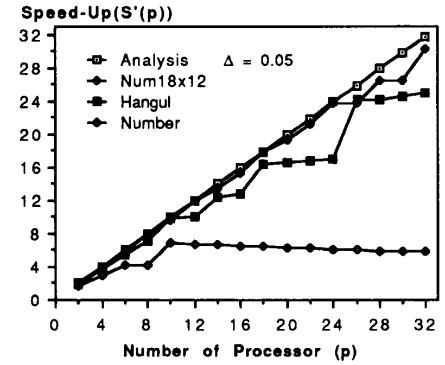
Another interesting feature in Figure 5 is that the speed-up of the forward pass of *Hangul* application is not linear, but almost a step function. The reason is that the number of neurons in a layer, n_l, is not always divisible by p in the p-processor DMM. Therefore, though a processor in the p-processor DMM has finished the computation for $\lfloor \frac{n_l}{p} \rfloor$ neurons, it may have to wait other processors which take charge of $\lceil \frac{n_l}{p} \rceil$ neurons. Because of this synchronization overhead, the speed-up ratio is increased in a stepwised fashion. For example, the speed-up ratios of *Hangul* application with 18 and 24 Transputers are almost same. (See Figure 5-(b))

5. Summary and Concluding Remarks

The backpropagation learning can take huge amount of time for a practical application. One natural way to overcome the time and space limit is to use parallel computers. We studied the distributed backpropagation on the p-processor DMM. Especially, we derived the fully distributed backpropagation algorithm, identified the necessary communication mode (all-



(a) The p-Transputer Speed-up of the Distributed Backpropagation Algorithm



(b) The p-Transputer Speed-up of the Forward Pass of the Distributed Backpropagation Algorithm

Figure 5: The Experimental Speed-up with Ring of 32 Transputers

to-all broadcasting), derived the time complexity of communication and computation, thereby obtained the theoretical upper-bound of speed-up, and finally presented the experimental speed-up of our algorithm on a ring of 32 Transputers which confirmed our model and analysis. Also we found a very interesting result that the processor interconnection topologies such as ring, mesh, hypercube and complete connection do not influence to the speed-up ratio because they all could provide the $O(p-1)$ complexity which is the lower bound for AAB(p). Thus, no more powerful topologies than the ring need to be used in our distributed backpropagation algorithm.

Our model and equations for the speed-up/space-reduction ratios can be very useful for studying the effect of the number of neurons in a neural network, the communication / computation ratio, and the number of processors, on the backpropagation algorithm speed-up.

References

[1] M. Annaratone, C. Pommerell, and R. Ruhl, "Interprocessor Communication Speed and Performance in Distributed-Memory Parallel Processors," *Proc. of the 16th Annual Int'l Symp. on Computer Architecture*, pp.315-324, 1989.

[2] J. Cook and J. Gilbert, "Parallel Neural Network Simulation using Sparse Matrix Techniques," *Microprocessing and Microprogramming 24*, pp.621-626, 1988.

[3] *DARPA Neural Network Study*, AFCEA International Press, 1988.

[4] E. Deprit, "Implementing Recurrent Back-Propagation on the Connection Machine," *Neural Networks*, Vol.2, pp.295-314, 1989.

[5] J. A. Feldman et al., "Computing with Structured Connectionist Networks," *Comm. of ACM*, Vol.31, No.2, pp.170-187, 1988.

[6] B. M. Forrest et al., "Implementing Neural Network Models on Parallel Computer," *The Computer Journal*, Vol.30, No.5, pp.413-419, 1987.

[7] J. Ghosh and K. Hwang, "Mapping Neural Networks onto Message-Passing Multicomputers," *Journal of Parallel and Distributed Computing*, April 1989.

[8] J. Hicklin and H. Demuth, "Modeling Neural Networks on the MPP," *Proc. of the 2nd Symp. on the Frontiers of Massively Parallel Computation*, pp.39-42, 1988.

[9] S. L. Johnsson and C. T. Ho, "Optimum Broadcasting and Personalized Communication in Hypercubes," *IEEE Trans. on Computers*, Vol.38, No.9, pp.1249-1268, 1989.

[10] J. Millan and P. Bofill, "Learning by Backpropagation : A Systolic Algorithm and Its Transputer Implementation," *Neural Networks*, Vol.1, No.3, pp.119-137, 1989.

[11] D. A. Pomerleau et al., "Neural Network Simulation at Warp Speed: How We Got 17 Million Connections Per Second", *Proc. of IEEE 2nd Int'l Conf. on Neural Networks*, Vol. II, pp. 143-150, 1988.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol.1, pp.318-362, MIT Press, 1987.

[13] H. Yoon and J. H. Nang, "Multilayer Neural Networks on Distributed-Memory Multiprocessors," to appear in the *Proc. of Int'l Neural Network Conference*, Paris France, July 1990.

[14] H. Yoon and J. H. Nang, "Parallel Simulation of Multilayered Neural Networks on Distributed-Memory Multiprocessors," to appear in the *EUROMICRO Journal: Microprocessing & Microprogramming*.