

Speed up Performances on MIMD Machines¹

Harold Szu, Charles Yeh, George Rogers, Michael Jenkins, Ali Farsaie
Naval Surface Warfare Center, Silver Spring, MD 20903-5000
and
Chin-Hwa Lee, Naval Postgraduate School, Monterey, CA 93943

ABSTRACT

This paper describes the implementation of a second order backpropagation algorithm for pattern recognition on a PC, an INMOS transputer network and on Intel hypercubes. The trade-off between communication and computation has been investigated. The speed up concurrency is given with respect to the number of processors in terms of a sigmoidal-like curve for the first time.

1. Introduction

In order to harness the computational power of parallel computers, several investigators [1,2,3,4] have examined into this problem. The work described in this paper was undertaken to develop the technology base needed for using parallel processing and signal processing technologies for meeting current and future navy real-time applications requirements. It is based on the experience gained through the concurrent processing architecture testbed, an independent exploratory project in which a 16 nodes transputer network was designed, built, and used to implement a multi-targets tracking algorithm using Kalman filter. The parallel software development approach used in the backpropagation implementation is the same as that used in the multi-targets implementation. The approach is to parallelize a sequential program by identifying the parts of the sequential code that can be executed in parallel and distributing these parts evenly among available processors in the network for processing. This paper is organized as follows: section 2 describes the pattern recognition problem, section 3 lists the backpropagation algorithm, section 4 discusses the decomposition strategy used to partition the algorithm, section 5 discusses the parallel software development issues for transputer network, section 6 discusses the parallel software development issues for Intel hypercube, and section 7 discusses the results obtained for these networks with different machine architectures.

2. Patterns recognition problem

The problem consists of classifying noisy binary images of missiles into one of the four possible classes. Each missile image is made up of 928 pixels(58 x 16) with binary values and is used as input to the input layer of a fully connected three-layered network with architecture as shown in Figure 1 (Connections are drawn only to the left most nodes of hidden and output layers to simplify the drawing). The network uses the input values to update the hidden layer nodes whose val-

1. The work on transputer network was funded by the Focus Technology Program at NAVSWC, White Oak and the work on Hypecube by the Engineering of Complex System Block of the Office of Naval Technology.

ues are in turn used to update the output layer nodes to come up with a classification for the images.

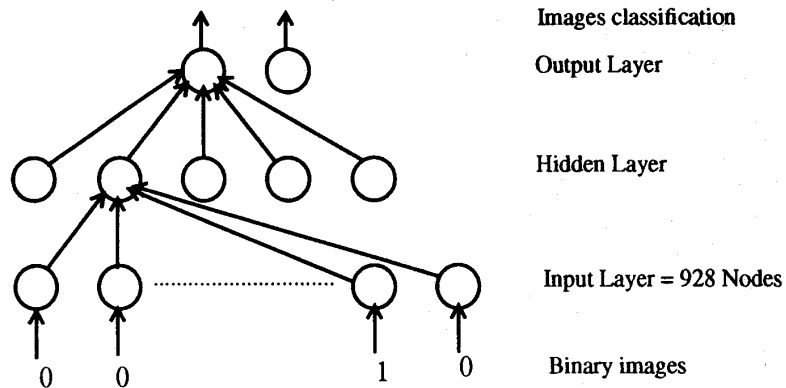


FIGURE 1. A Fully Connected 3-layered Network

3. Second Order Backpropagation Algorithm

The reason for the learning algorithm to be second order is given in [5]. The learning algorithm used to adjust the connections' weights of the network is divided into two phases, forward pass and backward pass and is described in pseudo code as follows:

Do until sum squared error for the training patterns set is within the convergence criteria

For each pattern do

/* Forward Pass */

Update the nodes in Hidden layer

Update the nodes in Output layer

/* Backward Pass */

Adjust weights between Output and Hidden layers

Adjust weights between Hidden and Input layers

4. Decomposition Strategy

It is a common sense approach without yet a systematic analysis. This becomes obvious by examining the following equations. The computational load for updating nodes involves mainly finding the dot products in the following equations

$$\sum_{i=0}^N w_{ji} x_i$$

$$\sum_{i=0}^N (w'_{ji} x_i + w_{ji} x'_i)$$

one node to another using intermediate nodes as connecting points if the source and destination nodes are not directly connected. This deadlock problem does not occur in sequential program because it is using shared memory with only one process accessing it. To prevent deadlock from happening in the network, we used buffers for each of the processes running on the transputer node as shown in Figure 3. Note that message intended for other node will pass from input process to output process bypassing computation process.

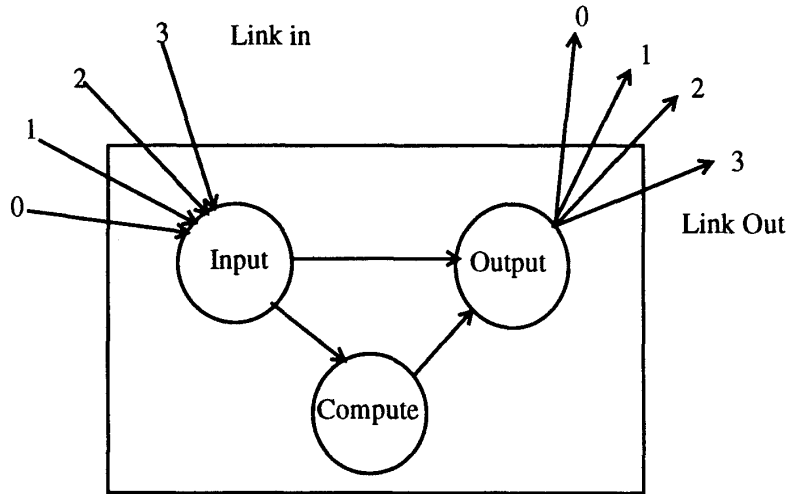


FIGURE 3. Processes Within a Transputer Node

6. Software Development Issues for Intel Hypercube

Once we have debugged the program for running on a transputer network, it was easy to port it over to Intel Hypercube. There was no need to design a message router since Intel Hypercube has a direct connect module built into the system to route messages efficiently. There was also no need for the input and output processes within each node to have buffers to prevent deadlock since the direct connect module makes the message passing act if the source and destination nodes are directly connected. The measure of multiprocessor performance used is the speed up, S , as defined in [7] :

$$S = \frac{t_s}{t_p}$$

where t_s is the time it takes to run on a network with one node, and t_p is time it takes to run on the multiprocessor. The speed up for transputer network is shown in Figure 4.

7. Results and Conclusions

The upper curve is for the training phase of the algorithm which backpropagates the errors from output layer nodes to input layer nodes to adjust the weights between output layer and hidden layer and the weights between hidden layer and output layer. The lower one is for testing

where w_{ji} is the weight for connection from node i in lower layer to node j in upper layer, x_i is the output value of the node in the lower layer, N is the number of the nodes in the lower layer and prime is the time derivative. Since the computational load for the hidden layer nodes is heavier than that of output layer nodes as can be seen by comparing the number of connections incident on the hidden layer node(928) with the five connections incident on the output layer node(4 plus 1 for bias), we decided to parallize the codes for updating the hidden layer nodes. For the same reason we picked the code for adjusting weights between Hidden and Input layers.

5. Software Development Issues for Transputer Network

There are three issues that we came across in getting the backpropagation program to work. One is the design of the message router, which depends on the topology of the transputer network. The toroid topology we used for the transputer network is shown in Figure 2 and takes at most 4 hops for a message to go from any one node to any other node in the network.

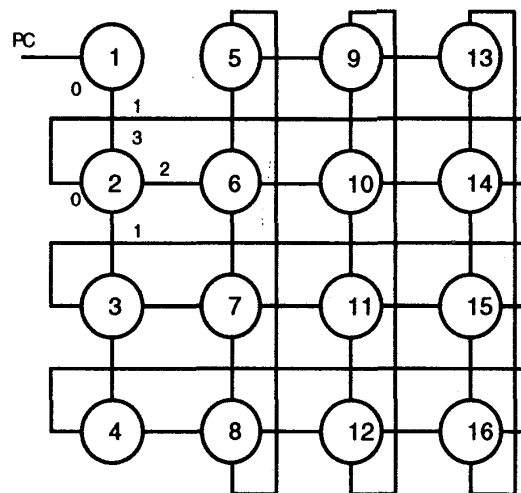


FIGURE 2. Toroidal Transputer Network Topology

Table lookup routing algorithm [6] was chosen over the routing algorithm that calculates which link of the transputer to send the message out because the table lookup routing algorithm is easy to change if a link or node failed to function correctly. Another issue is the synchronization of processes in the network to preserve the logic of the sequential algorithm. We used node 1 as a synchronization process to collect results from other nodes in the network and to run the sequential part of the parallel program. It blocks(suspends execution of sequential program) until it receives all required data back from the network. The last issue of process deadlock came about due to the memory structure of the network. Being a distributed memory multiprocessor, the transputer network has a local memory for each transputer and has to use message passing to transfer data from

phase of the algorithm involves just feeding forward the output from input layer nodes to output layer nodes to come up a classification for the missile images. These two curves show that the network is scalable only up to a network size of 9 and performance degrades after that due to communication bottleneck in the network.

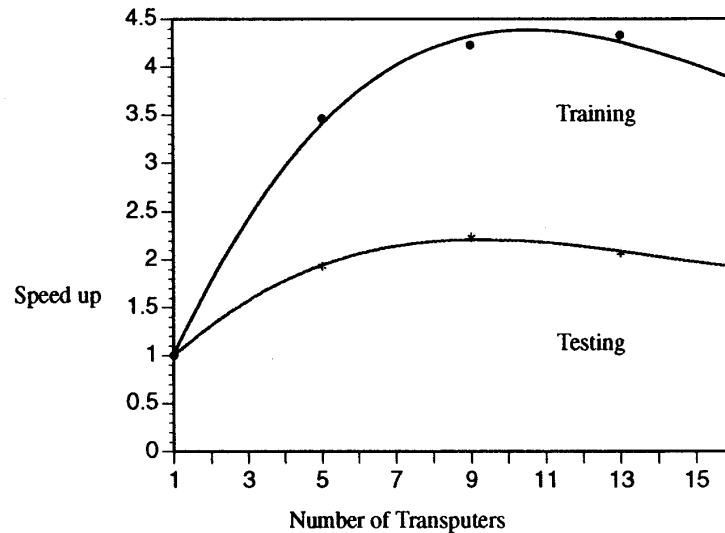


FIGURE 4. Speed up for Toroidal Transputer Network

The speed up for Intel's Hypercube iPSC/2 is shown in Figure 5.

The speed up curve for Intel's hypercube 860 is similar to that of iPSC/2. We have only run the parallel program up to a hypercube of degree three(9 nodes= 8 nodes + host, system resources manager); it would be interesting for future work to run the program on a hypercube of degree four consisting of Intel's hypercube 860 and iPSC/2 to see if the network is scalable for a massively interconnected algorithm.

Systems Performance Comparison for Backpropagation algorithm

Training Time for HP Vetra 286 PC = 22 minutes
 Training Time for 16 nodes transputer network = 5 minutes
 Training Time for iPSC/2= 0.5 minutes

We have described the process we used to implement a communication intensive backpropagation algorithm on two multiprocessors with different architecture, toroid and hypercube. We have presented the results that show the networks can be used to improve a system performance only up to a point due to communication bottleneck. Despite this shortcoming, it is still better to use parallel processing over the conventional sequential processing.

It has been known in neural net global interconnect simulations on SIMD-type machines (e.g. Connection Machine[8]) that the communication overhead is the major bottleneck due to the n-n neighbor interconnection of the simple nodes. Thus we studied MIMD-type machines. In this case, the decomposition strategy described in section 4 became the major effort to remove the bottleneck that degrades the speed up.

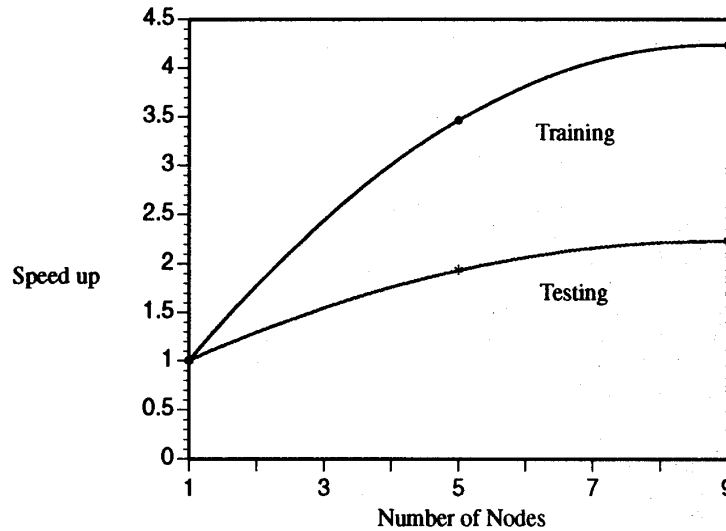


FIGURE 5. Speed up for Intel's Hypercube iPSC/2

Acknowledgments

Elizabeth Farrar translated and modified the pascal version of the backpropagation program listed in the appendix of the reference 1 into sequential program in C. We thank James Restorff for allowing us to use his transputer network. We also thank Steve Howell and Katherine Murphy for letting us use the Intel hypercubes.

References

- [1] H. Yoon, et. al., "A distributed Backpropagation Algorithm of Neural Networks on Distributed-Memory Multiprocessors," *Proc. of the 3rd Symp. on the Frontiers of Massively Parallel Computation*, 1990
- [2] J. Cook and J. Gilbert, "Parallel Neural Network Simulation using Sparse Matrix Techniques," *Microprocessing and Microprogramming* 24, pp. 621-626, 1988.
- [3] 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
- [4] B. M. Forrest et. al., "implementing Neural Network Models on Parallel Computer," *The Computer Journal*, Vol. 30, No. 5, pp. 413-419, 1987.
- [5] D. B. Parker, "Second Order Back Propagation: Implementing an Optimal O(n) Approximation to Newton's Method as an Artificial Neural Network,"
- [6] G. McIntire, "Design of A Neural Network Simulator on a Transputer Array," *Space Operations-Automation and Robotics Workshop* 87
- [7] I. D. Scherson and P. F. Corbett, "Communications Overhead and the Expected Speedup of Multidimensional Mesh-Connected Parallel Processors," *Journal of Parallel and Distributed Computing*
- [8] J. Wiley, Private Communication, NRL Code 5348.