ON THE PARALLEL SIMULATION OF ARTIFICIAL NEURAL NETWORKS

Petridis V.[1], Adamidis P.[1], Margaritis K.G.[2]
[1] Department of Electrical Engineering, Aristotelian University of Thessaloniki, Greece
[2] Informatics Centre, University of Macedonia, Greece

ABSTRACT

Simulating an ANN on a parallel processing system one can use several techniques. This paper presents two methods on implementing parallel simulations of Artificial Neural Networks (ANNs) on Transputer Based Systems, using the C programming language under Helios O.S. and Component Distribution Language (CDL) or, alternatively, the 3L Parallel C language. The Processor Farm topology is used for the parallel implementation of Back-Propagation and Multi-Layered Feed-Forward ANNs. The Processor Farm topology is realized by means of different parallelizing tools offered by Helios (i.e. CDL scripts) and by 3L Parallel C (i.e. explicit process allocation and communication). Results of the parallel methods are compared to sequential implementations.

1. INTRODUCTION

From the moment that ANNs have been established as a new type of information processing, the problem of their implementation has been raised. The used methods can be classified into the following categories:

(a) Use of conventional machines and sequential programming languages

(b) Use of parallel processing systems and programming techniques in order to distribute the computational load in several processors

(c) Use of special purpose VLSI circuits as accelerators and co-processors

(d) Use of optical or hybrid electro-optical techniques

Implementing an ANN on a parallel processing system one can use either of the following techniques: (a) Parallelise the learning algorithm itself, (b) Distribute parts of the network over the available processors, (c) Distribute the learning information

1.1 ARTIFICIAL NEURAL NETWORKS

An ANN is a processing device, either an algorithm, or actual hardware, whose design was motivated by the design and functioning of human brains and components thereof.

It is a network of many very simple processors ("units" or "neurons") each possibly having a small amount of local memory. The neurons are connected by unidirectional communication channels ("connections") which carry numeric data ("weights"), and each neuron operates only on its local data and on the inputs it receive via the connections. Despite the diversity of network paradigms, nearly all function on the following way: A set of inputs is applied to a neuron. Each input is multiplied by an associated weight and the weighted inputs are added algebraically. The sum is usually further processed by an activation function to produce the neuron's output.

Thus, the output x_i of neuron i is calculated as : $x_i = f\left(\Sigma\, w_{ji}\, x_j\right)$

j

Most neural networks have some sort of "training" rule whereby the weights of connections are adjusted on the basis of presented patterns

1.2 TRANSPUTER BASED SYSTEMS

A Transputer-based machine may be defined as an implementation of the MIMD concept associated with a pure distributed memory model. The INMOS transputer provides a direct implementation of the message-passing mode of parallel computation. It is a single-chip micro-processor that provides a high-speed processor, fast interprocessor communications, and explicit support for multiple processes and multiple processor systems.

The 20 MHz T800 and T805 transputer types, that we used, have a 32-bit integer CPU, a Floating Point Unit, 4K of internal memory, 4 communication links at 20Mbps and 2Mb of local memory. Crossbar switches allows for software configuration of the transputer network topology.

Transputers can be used to realise all types of parallelism, but they are best suited in cases where medium to coarse grain parallel processing is used. Coarse grained parallelism is used here in order to distribute the training pattern set over the available transputers. We describe a parallel strategy for mapping ANN models as independent processes running on a Farm of processors which are controlled by a dedicated processor (controller). The strategy is based on the splitting-up of the patterns between the available transputers so that each transputer processes a different part of the training set, while the controller keeps track of the global network state.

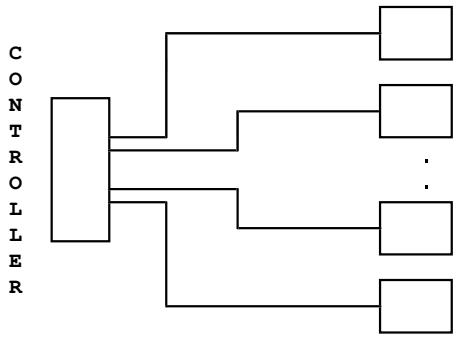

Fig. 1 Processor Farm Topology

The Farm topology was chosen, since it realizes exactly the simulation concept of simultaneously running multiple copies of the learning algorithm. The simulation sets up a network of Transputers so that each Transputer can run an independent ANN algorithm under the control of another process on the Controller.

2. BACK-PROPAGATION ON A TRANSPUTER-BASED SYSTEM

Back-propagation is the most widely used learning method for ANNs. It is for the training of layered (i.e., neurons are grouped in layers) feed forward (i.e., the arcs joining neurons are unidirectional, and there are no cycles) nets. The algorithm tries to find a set of weights according to training input/output patterns, so that given each input pattern produced by the network is sufficiently close to the desired output pattern.

It consists of the following phases: forward activation flow, back propagation of the error, and weight update. In the forward pass, each neuron calculates the weighted sum of its inputs, and computes a new activation value for output. The next phase involves the comparison between the actual output pattern and the desired one, and the back propagation of the error. Finally the weight update is performed.

2.1 IMPLEMENTATION

The Telmat transputer system that we used to implement the simulation consists of 24 20Mhz T800 transputers. The network is hosted by a Motorola 68040 UNIX system. The software was developed in C under Helios, and Component Distribution Language (CDL).

The simulation was implemented, using each Transputer as an independent processor running the Back-Propagation learning algorithm. The simulation strategy is as follows:

Initially the Controller sends to each Transputer (the **Trainee**) information on: network architecture, initial weights and a part of the training pattern file. Each Trainee processes every pattern presented to it (forward flow, back propagation and weight update), and then it returns the updated weights and the output error that it has computed to the Controller.

The Controller computes the average error and the average of each weight received from each Trainee. If the average error has not reached an acceptable level, then the Controller sends the new weights and another part of the training pattern file to each Trainee. This procedure continues, until the error reaches a specified level. The number of patterns sent to each Trainee is kept constant and it is termed **Update Rate** (UR), since the average of each weight is found only after each Trainee has finished processing these patterns.

Trainees update their connection weights each time they finish processing a training pattern, since it was found that this accelerates the convergence of the ANN. Assuming TR is the number of Trainees, and NP is the number of training patterns, then the Update Rate (UR) must be a divisor of NP/TR. If we take as UR = NP/TR then the Controller sends to each Trainee UR training patterns. When each Trainee finishes processing these patterns it sends its weights and the computed error to the Controller where average weights and error are computed, thus completing an iteration. If a divisor of NP/TR is taken as Update Rate, e.g. UR=(NP/TR)/K then the Controller must send K times, UR different training patterns to each Trainee in order to complete an iteration.

2.2 EXPERIMENTAL PERFORMANCE MEASUREMENTS

When using a number of processors to distribute the computational workload, the speed-up is considerably smaller than the calculated theoretical speed-up. This is because the number of iterations required, increases along with the number of Trainees involved.

In the same figures it is shown that using different values of Update Rate leads to different training rates. The worst performances are obtained for very small and very large values of Update Rate. This is because small values of Update Rate increase the communication overhead, while large values of Update Rate delay excessively the weight modification process.

An example of the results is shown in figure 2. The function used is f(x)=sin(x) with values of x in the range [0-360]. The pattern file contains 450 patterns. It took 1200.37 seconds and 138 iterations of the whole pattern file to train the conventional simulation on one Transputer. The training parameters were kept constant. The network implemented has 1 input, 2 hidden layers with 10 and 5 neurons respectively, and 1 output. The performance is shown in figure 2.

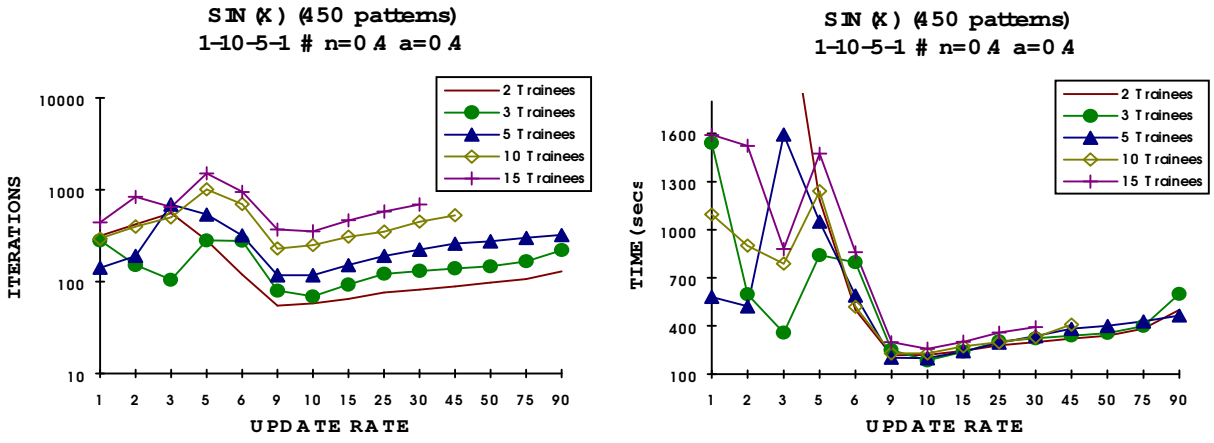

Fig. 2 Performance of the Parallel ANN simulation trained on the function sin(x)

3. THE FEED FORWARD METHOD OF TRAINING ON A TRANSPUTER SYSTEM

In the sequel the Feed Forward (FF) method which has been developed by V.Petridis and K.Paraschidis [3] is presented. It is iterative and it is terminated when the total error becomes less than a predefined value. One of its basic features is that the weights are updated one at a time, within each iteration (iteration is defined as the cycle within which all the weights are updated once). The weights are numbered in arbitrary order from 1 to N. Their values define what we call the network weight point in the weight space. They are initially set to random values and are updated one by one from w_1 to w_N by an updating procedure presented below.

The updating procedure of a particular weight w_i consists of three phases. Phase 1: The direction of search is determined by perturbation. The network weight point moves along the w_i axis by an amount f or -f (f>0 is a fraction of the stepsize s) If the total error reduces when the network weight point moves by f, the direction of search is positive, otherwise (change by -f) it is negative. Phase 2: The network weight point moves in steps of size s along the direction of search, until, either the maximum allowable number of forward steps, d, is reached (in which case the updating procedure of this weight is completed and its output is the network weight point reached after the last step taken), or a total error larger than the one produced at the previous step is observed (in which case the network weight point reached after the last step taken is the starting point for the next phase). Phase 3: The network weight point reached during the previous phase, moves in a direction opposite to the direction of search by a number of steps. The size of the first step is s/2 and it is decreased progressively by a factor of 2 for every following step. The backward steps continue until, either the maximum allowable number of backward steps, c, is reached (in which case the output of the updating procedure is the network weight point reached after the last step taken), or an increase in the total error is observed (in which case the output of the updating procedure is the network weight point reached before the last step taken).

3.1 IMPLEMENTATION

The Quintek Fast9 Transputer Board was used to implement the parallel simulation of the FF method. The board consists of 9 25MHz T805 transputers. The host system is a 80386 PC. The simulation was developed using the 3L Parallel C. The transputers are physically connected in a pipe, but one can use the available configurer in order to program the link exchange switch and implement another topology. The configurer is also used to distribute the processes on the available transputers. The farm topology is used in this implementation as well. The simulation is as follows:

Initially, the controller sends to each transputer (worker), network information and a part of the training patterns. Assuming NP is the total number of patterns and Tr the number of workers used, then each transputer processes NP/Tr patterns. For each iteration the following is repeated for all the weights from 1 to N: The controller sends to each worker the amount f to change the first weight. Each worker sends back the computed error. The controller compares the new total error with the previous one and reacts according to this, sending another f to change the weight. When the updating procedure for this weight is completed, it is repeated for the next weight.

3.2 EXPERIMENTAL RESULTS

We implemented several configurations using different number of workers. We noticed that the algorithm shows a stochastic behaviour. That is for the same ANN structure, initial weights and training patterns the algorithm should need the same number of iterations in order to converge, regardless of the

number of transputers that are used. However we found that this is not true and that the algorithm needs a different number of iterations in order to converge.

We ran simulations using the conventional sequential method running on one transputer, and parallel simulations running on 2, 4 and 8 workers. When using a greater number of workers, the number of iterations may either decrease or increase. That is the updating procedure is not the same. This is because of round off error problems when comparing the new error received by the controller with the previous one.

We found that the speed-up when we double the number of workers varies from 1.88 to 1.98 depending on the size of the ANN and the number of training patterns, as long as the update procedure remains the same.

4. REFERENCES

1. McClelland J L, et al (eds), "Parallel distributed processing", MIT Press, 1986.
2. Muhlenbein H, Wolf K, "Neural network simulation on parallel computers", in Proc. Parallel Computing 89, North-Holland 1990.
3. Petridis V., Paraschidis K., Private Communication
4. Smith L S, "Neural net simulation using Transputer nets", SERC/DTI Transputer Loan Report, 1988.
5. Szu H, et al, "Speed up performances on MIMD machines", Proc. IJCNN'92, p.742-747, Baltimore, USA, 1992
6. Tollenaere T, Orban G A, "Simulating modular neural networks on message-passing multiprocessors", Parallel Computing 17, 1991.
7. Wang C, Wu C, "Parallel simulation of Neural Networks", Simulation, 56, 1991.