ON THE REALIZATION OF BACK-PROPAGATION ON A TRANSPUTER BASED SYSTEM

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

SESSION: Parallel Processing

ABSTRACT

This paper presents some experimental results on the realization of a parallel simulation of an Artificial Neural Network (ANN) on a Transputer based system. The ANN training method chosen is the Back-Propagation algorithm and the parallelization technique utilized is the Processor Farm based on the partition of the training patterns between the available transputers.

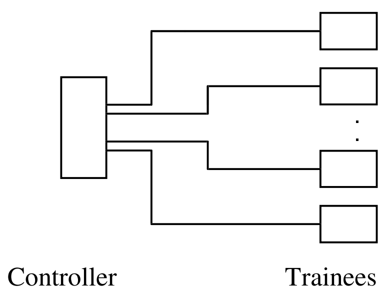
1. INTRODUCTION

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

Transputers can be used to realise all types of parallelism, but since they are powerful enough, 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.

This paper describes 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 (trainee) is trained on a different part of the training set, while the controller keeps track of the global network state. The simulation was implemented, using each Transputer as an independent processor running the Back-Propagation learning algorithm.

The Processor Farm topology is shown in the following figure:



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 which runs either on a separate Transputer (the **Controller**) or on a Transputer that already runs an ANN algorithm.

The Telmat transputer system that we used to implement the simulation consists of 24 20Mhz T800 transputers. Each transputer has 2MB of memory, and four 20Mbps communication links. The network is hosted by a Motorola 68040 Unix system. The software was developed in C under Helios, and Component Distribution Language (CDL).

2. SIMULATION STRATEGY ON THE TRANSPUTER BASED SYSTEM

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.

The main issues that have been experimentally addressed are: (a) the speed-up of this the parallel simulation against the conventional simulation on one processor and (b) the existence of an optimal value of the Update Rate

The typical computation and communication timing of the T800 Transputer was used to evaluate the performance of both the conventional and parallel implementations.

3. EXPERIMENTAL PERFORMANCE MEASUREMENTS

We present some of the results on the training of three different functions:

i) The first function is $f(x)=e^x$ with values of x ranging from -7.5 to 7.5, incremented by 0.01. The pattern file contains 1500 training patterns. The learning rate (n) and the momentum (a) were kept constant (n=0.1, a=0.8). The ANN used, has one input, one hidden layer with 5 neurons, and 1 output.

Figure 1 shows the performance of the parallel ANN simulation for different number of Trainees and different values of Update Rate. The implementation on one Transputer took 6.56 seconds and one iteration.

ii) The second function 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 (n=0.4, a=0.4). 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.

iii) Functions $z=(x^2+y^2)^{1/2}$ and è=arctan(y/x) were used, to transform the Cartesian co-ordinates x, y into Polar co-ordinates z, è. The pattern file contains 1600 patterns. The training parameters were kept constant (n=0.4, a=0.4). Figure 3 shows the performance of a parallel simulation with 2 inputs (x, y), 2 hidden layers with 10 neurons each, and 2 outputs (z, è). The implementation on one Transputer requires 324 iterations and 14194 seconds.

4. CONCLUSIONS

As shown in Figures, when using a number of processors to distribute the computational workload, the speed-up is considerably smaller than the theoretical speed-up calculated. This is because the number of iterations required, increases along with the number of Trainees involved. Therefore it is concluded that in general the training information cannot be effectively partitioned.

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.

Another way of partitioning the training pattern file was also tested: every Trainee was trained with patterns scattered all over the pattern file. In this case the number of iterations keeps incrementing with the number of Trainees. The value of the Update Rate does not influence the number of iterations needed to train the network. It remains constant and independent of the values of Update Rate. Thus in this case, it is better to use the maximum value of Update Rate, since it reduces the communication overhead and accelerates the convergence of the network.

5. REFERENCES

1. McCelland 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. Smith L S, "Neural net simulation using Transputer nets", SERC/DTI Transputer Loan Report, 1988.
4. Tollenaere T, Orban G A, " Simulating modular neural networks on message-passing multiprocessors", Parallel Computing 17, 1991.
5. Wang C, Wu C, "Parallel simulation of Neural Networks", Simulation, 56, 1991.

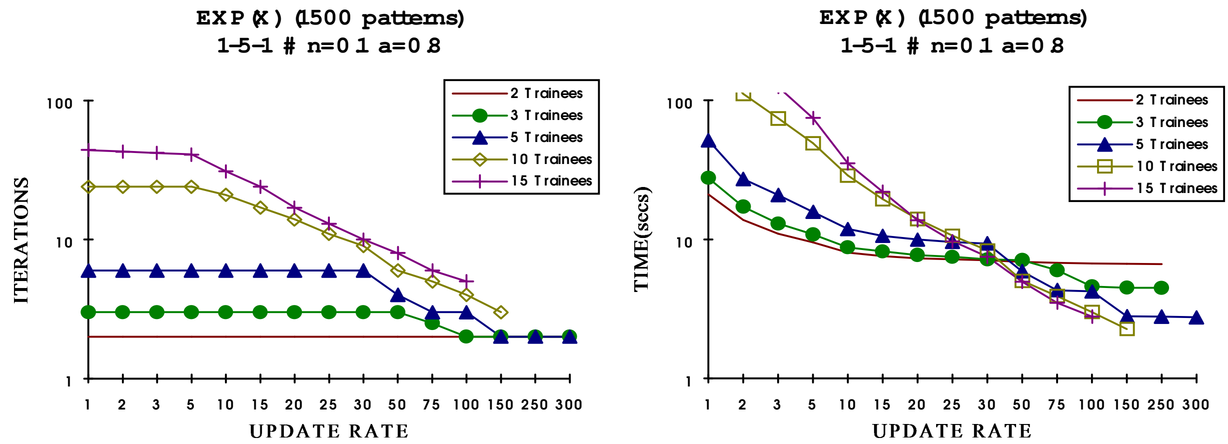


Fig. 1 Performance of the Parallel ANN simulation trained on the function e^X

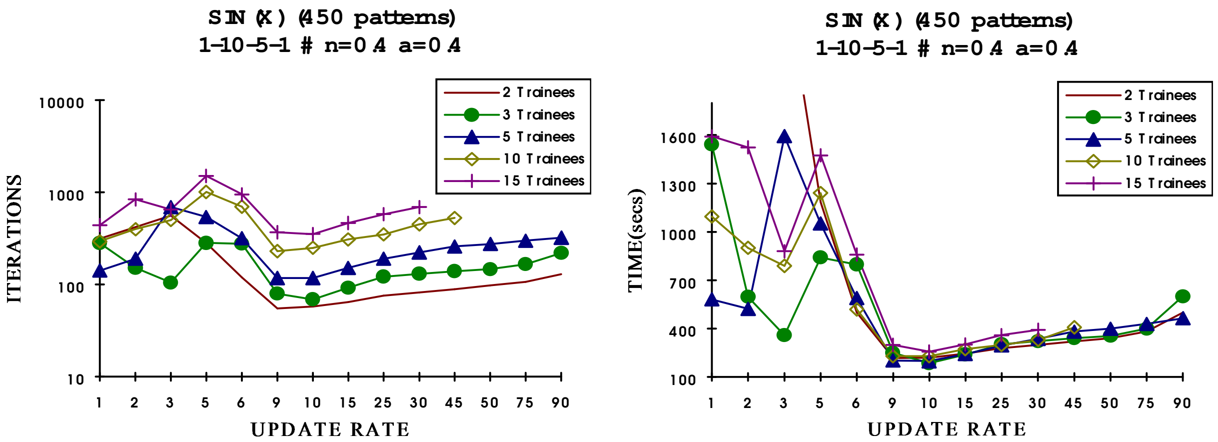


Fig. 2 Performance of the Parallel ANN simulation trained on the function sin(x)

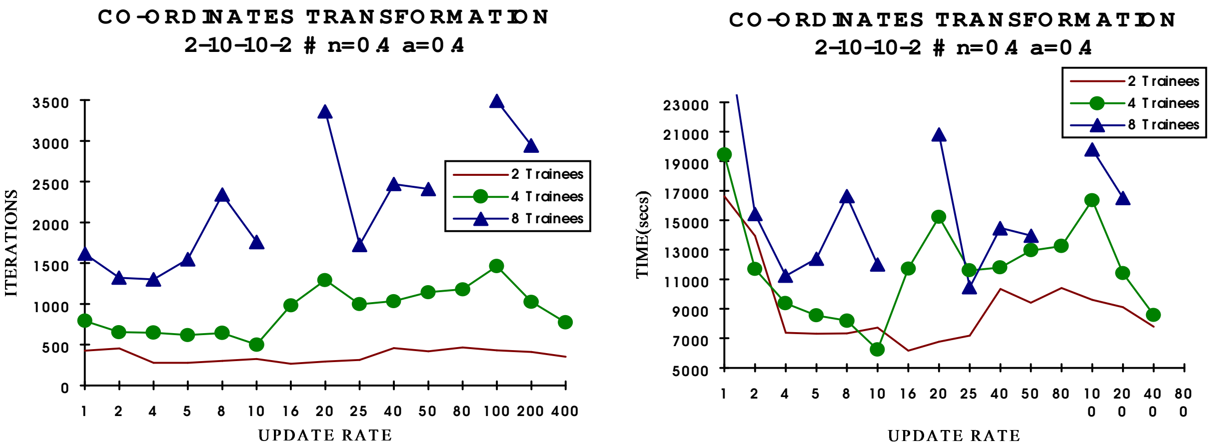


Fig. 3 Performance of the Parallel ANN simulation trained on the functions $z=(x^2+y^2)^{1/2}$ and è=arctan(y/x).