Efficiency of Parallel Computations for Error Backpropagation Neural Network Training

Milena K. Lazarova[1]

Abstract - **The paper investigates the efficiency of parallel computations at the training stage of an error backpropagation neural network. Flat, multithreaded and hybrid parallel computational models are suggested. Performance comparison is made and scalability in respect to the multicomputer size and its impact on the performance of the parallel system is estimated.**

Keywords - **parallel computations, message passing, shared memory, neural network training, error backpropagation**

I. Introduction

Artificial neural network (NN) is an interconnected assembly of simple processing elements whose functionality is loosely based on the structure and functioning of the brain [1]. The processing ability of the NN is stored in the inter-unit connection strengths (weights) obtained by a process of adaptation to or learning from a set of training patterns. Among the basic properties of the NN that distinguishes them from the von Neuman computer are [2]: massive parallelism, distributed representation and computation, learning ability, self-organization, generalization ability, adaptivity, fault tolerance.

In order to apply a NN for solving particular problem the network has to be trained to effectively map an input set of values into an output space defined by the output element values. The training can be either supervised or self-organization process but in both cases it requires consecutive repetition of the adjustment of the values of inter-node weights. Therefore the NN training usually involves a lot of computations and is time-consuming. The problem of large training times can be overcome either by devising faster learning algorithms or by implementing the existing algorithms on parallel computing architectures [3, 4]. Shifting towards parallelization is a natural process of NN due to the inherent massive parallelism of their biological original: the brain. Since the computations of the model components are largely independent of each other most of the NN models have great potential for parallelism. The larger are the NN architecture and the training data set the more relevant is the exploitation of the inherent parallelism of NN.

The approach of NN parallelization depends on the specifics of the NN architecture and the organization of inter-node communications but most NN models are rather easily viable on parallel hardware of all kinds. Parallel architectures for simulating neural networks can be subdivided into general purpose parallel computers and neurocomputers [5, 6]. Neurocomputers are designed as boards and systems for high-speed ANN simulations [3, 4].

[1]Milena Lazarova is with the Faculty of Computer Systems and Control, Department of Computer Systems, 8 Kliment Ochridsky str., 1756 Sofia, Bulgaria, E-mail: milaz@tu-sofia.bg

Parallelization of NN was made on various parallel or distributed hardware architectures as workstation clusters [7-9], large grain supercomputers [10-12] and specialized neural network hardware systems [13, 14].

The aim of the paper is to investigate the efficiency of parallel computations at the training stage of an error backpropagation neural network. Flat parallel computational model is suggested for NN training. The model employs message passing for necessary data communication between the concurrent running processes and is targeted to a distributed memory parallel system. The utilization of multithreading is considered for parallel computation of the NN training on a shared memory platform. In order to exploit both the high-level parallelism through the message-passing and the low-level (loop) parallelism of the multithreading a hybrid (multi-level) parallel programming model is also utilized. Performance comparison is made for flat, multithreaded and hybrid parallel programming models. Since pattern recognition is among the most successive application examples of the NN utilization the case study for character recognition is used in the experimental estimation of the performance parameters. Speedup and efficiency as well as scalability in respect to the multicomputer size and its impact on the performance of the parallel system are estimated.

II. Neural Network Training by Error Back Propagation

A NN consists of an enormous number of massively interconnected nonlinear computational elements. Each element performs a weighted summation of input values received from other elements of the NN, applies an activation function to the weighted sum and outputs its results to other elements of the network.

A popular type of neural network is the multilayer perceptron, in which the elements are organized into layers. The layers are at least three and depending on the source of their input values and the availability of their output values are regarded as input, hidden and output layer. Each layer is usually fully interconnected to its adjacent layers.

One type of multilayer perceptron is the backpropagation neural network [15]. It is trained on a set of pairs (input value, targeted output value) using a two pass supervised algorithm based on the error correction learning rule [16]. During the NN training the weights of the connections between the elements are adjusted according to some learning rules. The error backpropagation learning algorithm is given bellow in pseudocode:

```
INITIALIZE
    Iter = 0; Max_Iter = IMax; Error = 0; Max_Error = EMax;
    random small weights for all element connections;
```

```
REPEAT
    FOR all training pairs $m = 1, \ldots, M$
        /* initialize forward pass */
            FOR all elements $j$ in the input layer
                    $o_j = x_j^{(m)}$                                  (1)
            END FOR
        /* run forward pass */
            FOR all elements $j$ in the hidden layers ($l$=1,2,…L–1)
            and the output layer (L)
                    $a_i^{(l)}(t) = \sum_{j=1}^{n_{l-1}} w_{ji}^{(l)}(t) o_j^{(l-1)}(t) + \theta_i^{(l)}$          (2)
                    $o_i^{(l)}(t) = f(a_i^{(l)}(t)),\ 1 \le i \le n_l$          (3)
            END FOR
        /* initialize backward pass */
            FOR all units $j$ in the output layer $L$
                    $\delta_i^{(L)}(t) = (d_i^m - o_i^{(L)}(t)) o_i^{(L)}(t)(1 - o_i^{(L)}(t))$   (4)
            END FOR
        /* run backward pass */
            FOR all units $j$ in the hidden layers ($l$ = 1, 2, … L-1)
                $\delta_i^{(l)}(t) = o_i^{(l)}(t)(1 - o_i^{(l)}(t)) \sum_j \delta_j^{(l+1)}(t) w_{ij}^{(l+1)}(t)$   (5)
            END FOR
            /* update weight values */
            FOR all units $j$ in the hidden layers ($l$ = 1, 2, … L-1)
            and the output layer (L)
                $w_{ji}^{(l)}(t+1) = w_{ji}^{(l)}(t) + \Delta w_{ji}^{(l)}(t), l = L, L-1, \ldots, 1$   (6)
                $\Delta w_{ji}^{(l)}(t) = \eta \delta_i^{(l)} o_j^{(l-1)}(t) + \mu \Delta w_{ji}^{(l)}(t-1)$          (7)
            END FOR
    END FOR
    Iter = Iter +1;
    /* estimate the error */
    FOR all training pairs m = 1, …, M
                $\varepsilon^{(m)} = 1/n_L \sum_i (d_i^m - o_i^L)^2$          (8)
    END FOR
UNTIL (Error < EMax) or (Iter > Max_Iter)
```

where $w^{(l)}_{ji}(t+1)$ is the new value of the weight connecting the ith element of layer l with the jth element of the previous layer l-1, $w^{(l)}_{ji}(t)$ is the same weight value at the previous iteration of the algorithm, $o^{(l-1)}_{j}(t)$ is the output value of the jth element in layer l–1, η is a learning rate parameter, μ is a momentum term parameter and $\delta_i^{(l)}$ is the error value.

III. Parallelization Strategies for Neural Network Training

The strategies for parallelization of NN training (Fig.1) can be distinguished as training, exemplar, node or weight level parallelism [6]. Training session parallelism implies independent training of the network on each node of a multicomputer system using different training parameters and initial network state. In this way the parallelism is actually implemented as running several attempts of the training that is a useful strategy if the error surface has a lot of local minima.

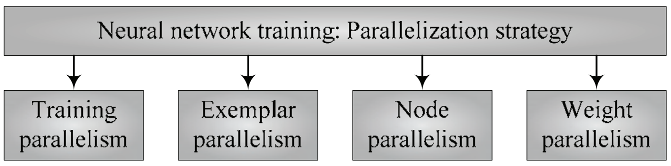

Fig.1. Strategies for parallelization of the NN training

Exemplar parallelism, also called training example parallelism, is suitable when very large training sets are utilized. In this case each process is assigned a subset of the training exemplars and it only trains the network for this training subset. At the end of each epoch the modifications of the network weights are gathered and applied to the NN. The node level parallelism takes the advantage of the natural parallelism implied by the distributed nature of NN and maps the elements to the processors of the multicomputer platform. There are different approaches for distributing the training phase computations between the nodes of a cluster exploiting the fact that the calculations at each NN node are independent [17]. In the next section a parallel computational model for NN training on a multicomputer platform using node level parallelism is suggested. Weight parallelism is the finest grained solution that requires parallel calculation of the input from each synapse. The weight parallelism provides no additional capabilities over the node parallelism strategy and introduces significantly more transactions of short messages. That is why it is considered not suitable parallelization strategy for a cluster computer [7]. On the other hand a combination of weight level and node level parallelisms is suitable for parallel computations on a shared memory system utilizing the multithreaded programming model discussed in chapter V.

IV. Flat Parallel Computational Model of NN Training

Flat (MPI-based) parallel computational model for error backpropagation neural network training on a multicomputer platform based on node level parallelism is presented in Fig.2. The model is based on a combination of two parallel programming paradigms: SPMD and synchronous iterations. The SPMD paradigm implies the data decomposition. For the NN training case it is implemented by concurrent computations of several processors at node level that is each network layer is divided between the available processors.

The synchronous iterations paradigm is a special case of the phase-parallel paradigm that can be regarded as alternating between two phases: computation and communication phase. At the computation phase each processor calculates the input values of the nodes assigned to it. During the communication phase each processor sends the calculated part of the output vector to all other processes using global communication function. The iteration is divided into forward pass that corresponds to the calculation at the forward stage of the learning algorithm followed by backpropagation pass that corresponds to the calculation and distribution of the errors backwards from the output to the input layer.

The NN is divided between the processes so that each process is responsible for calculation of a part of the output vector at each layer during the forward stage and part of the error vector at each layer during the backward stage of the training. The number of elements in each layer is equally spread among the available processors in order to guarantee good load balance. Thus before the training starts each processor calculates the range of node indices at each of the NN layers that are assigned to it according to the number of active processes (function MPI_Size) and its rank (function MPI_Rank). The input training parameters and the training set comprising pairs of an input value and the targeted output value are available to each process at the beginning of the training. Each process performs the following tasks:

- forward pass, computation phase: calculates the output values according to (3);
- forward pass, communication phase: sends the calculated part of the output vector to all other processes (function MPI_Allgather); in this way all processes receive the data necessary for the calculation of the outputs of the next layer;
- repeats the previous two phases for all layers of the NN until the output values of the output layer are calculated;
- backward pass, computation phase: calculates the errors and the weights at the output layer according to (4) and (6);
- backward pass, communication phase: sends the calculated part of the error vector to all other processes (function MPI_Allgather); in this way all processes receive the data necessary for calculation of the error values at the previous layer according to the error backpropagation algorithm;
- repeats the previous two phases for all hidden layers backwards to the input layer until all weights are updated according to (5) and (6);
- calculates the output error for each training pattern and terminates itself if the error limit or the predefined number of iterations are reached.

The above described sequence of computation-communication phases is repeated for all pairs in the training set. The NN training continues until either of the termination conditions occur. Each process resolves the end of the training and terminates itself.

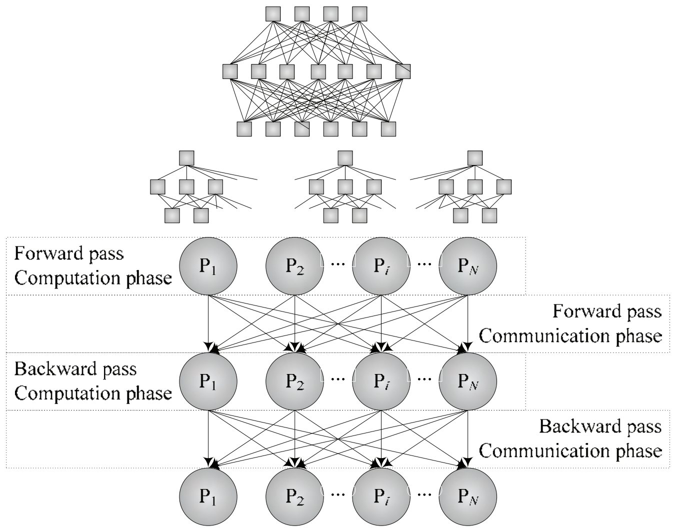

Fig.2. Parallel computational model for error backpropagation training using flat (MPI) programming model

V. Parallel Computation of NN Training by Multithreading

Multithreaded (OpenMP-based) parallel computational model for error backpropagation neural network training on a multiprocessor platform is implemented both at the forward and backward stage of the NN training by utilization of node and weight level parallelisms. Since the calculations performed for the elements in each layer are completely independent, unaware of the processing of other elements in the same layer, a fork-join parallelism is used based on loop level parallelization provided by the OpenMP API. Parallel execution of *for* loops by dispatching of threads at the beginning of a loop and assigning non-overlapping selections of the loop to each thread is implemented by the OpenMP pragma *omp parallel for*:

```
#pragma omp parallel for private (j, Net, Output)
#pragma omp parallel for private (j, Target, Actual, Delta)
```

where j is the number of the element in the layer, Net and Output are respectively the net activation and the output value calculated according to (2) and (3), Target and Actual are respectively the target output and the current output and Delta is the value calculated for the node according to (7).

Weight level parallelism is implemented as parallel *for* loops calculating formulas (2), (5) and (6).

VI. Parallelism Profiling and Performance Analysis

The suggested flat and multithreaded models are implemented in C++ and compiled using Microsoft Visual Studio 2005. Message passing is accomplished by MPI implementation MPICH2 v.1.0.3. Hybrid model is also developed utilizing both coarse grained (message-passing) and fine grained (multithreading) parallelism.

The case study for experimental evaluation of the performance parameters of the parallel NN training is character recognition. Experiments were carried out for a training set comprising examples of the digits from 0 to 9 represented in an aperture of 13 by 13 pixels as binary thresholded images. Thus the input network layer consists of 169 input elements and 10 output elements.

The experimental computer platform consists of 10 workstations (Intel Pentium 4, 3.2GHz, 1G RAM, Hyper-Threading) connected via Fast Ethernet Switch (100 Mbps). Multithreaded model is also executed on a dual core machine (Intel Core2 Duo, 1,83GHz, 1GB RAM).

Performance results are gathered for different sizes of the hidden NN layer (25, 50, 150, 1000, 2000) and different number of processors of the multicomputer (1 to 10).

The results for the speedup evaluated experimentally using the flat, multithreaded and hybrid programming models are given in Fig.3, Fig.4 and Fig.5 respectively. The successive computation and communication phases of the MPI model are seen in the Gantt's chart of the communication transaction given on Fig.6.

A speedup is gained with the flat (MPI) model only when the number of the NN elements in the hidden layer is big enough so that the communication overhead is comparable with the computation time during the NN training. For less than 150 hidden elements no speedup is achieved. The more hidden elements are used the bigger is the speedup: it is 5.1 for 1000 and 6.56 for 2000 hidden elements. The scalability in respect to the number of the processors is good up to a certain size of the multicomputer where the speedup saturates and can even decrease: for 150 hidden elements the serial training outperforms the parallel calculations by more than 6 processors, for 1000 and 2000 hidden elements the speedup does not increase for 8 or more elements.

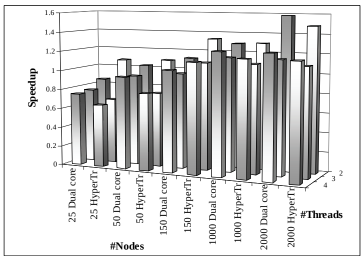

Fig.3. Speedup of the flat model

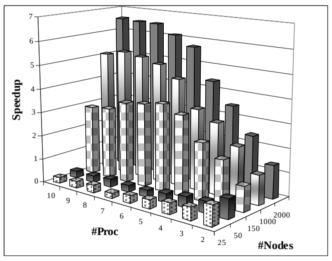

Fig.4. Speedup of the multithreaded model

Similar results are observed for the experiments with the OpenMP based multithreaded model: there is no speedup of the parallel model for less than 150 hidden elements. The obtained speedup for 2000 hidden elements on Dual Core machine is 1.58 and 1.47 on Hyperthreading (HT) machine. The smaller values for HT computer can be explained with the bigger number of cache misses. The utilization of more threads than the number of the processors does not increase the performance parameters.

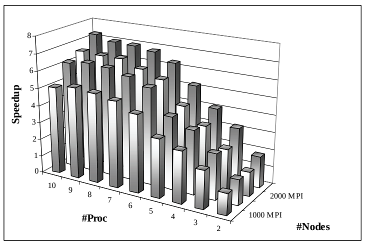

Fig.5. Speedup comparison of the flat and the hybrid models

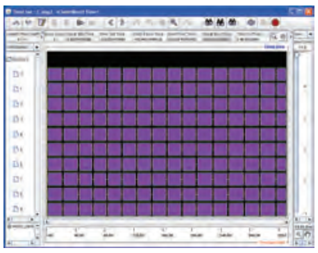

Fig.6. Gantt's chart of the communication transaction using MPI model

The speedup comparison of the flat and hybrid models of the NN training shows better utilization of the parallel system resources when a multi-level parallelism is utilized employing computations by several multithreaded parallel processes: speedup is increased with about 18% for less than 7 processes and with about 10 % for larger sizes of the multicomputer.

VII. Conclusion

The paper investigates the efficiency of parallel computations at the training stage of an error backpropagation neural network. Based on the strategy for parallelization of NN training known in the literature as node and weight level parallelisms, flat, multithreaded and hybrid parallel computational models are suggested and implemented. The experimental evaluation of the performance parameters shows that the parallel computational model outperforms the serial training only when the NN comprises lots of elements. Hybrid model leads to higher speedup than the flat model due to better utilization of the parallel computer system resources.

References

[1] C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.
[2] A. Jain, J. Mao, K. Mohiuddin, Artificial Neural Networks, IEEE Computer, Vol.29, pp.31÷44, 1996.
[3] C. Lindsey, T. Lindblad, Review of Hardware Neural Networks: A User's Perspective, Int. Journal of Neural Systems, Vol.6, pp.215÷224,1995.
[4] F. Diasa, A. Antunesa, A. Motab, Artificial Neural Networks: A Review of Commercial Hardware, Journal Engineering Applications of Artificial Intelligence, Vol.17, №8, pp.945÷952, 2004.
[5] N. Sundararajan, P. Saratchandran, Parallel Architectures for Artificial Neural Networks: Paradigms and Implementations, Wiley-IEEE Computer Society, 1998.
[6] T. Nordstrom, B. Svensson, Using and Designing Massively Parallel Computers for Artificial Neural Networks, Journal of Parallel and Distributed Computing, Vol.14, №3, pp.260÷285, 1992.
[7] M. Pethick, M. Liddle, P. Werstein, Z. Huang, Parallelization of a Backpropagation Neural Network on a Cluster Computer, Proc. of the Fifteenth IASTED International Conference on Parallel and Distributed Computing and Systems, ACTA Press, pp.574÷582, 2003.
[8] S. Suresh, S.N. Omkar, V. Mani, Parallel Implementation of Back-Propagation Algorithm in Networks of Workstations, Vol.16, No.1, pp.24÷34, 2005.
[9] R. Andonie, A. Chronopoulos, D. Grosu, H. Galmeanu, An Efficient Concurrent Implementation of a Neural Network Algorithm, Concurrency and Computation: Practice & Experience, Vol.18, №12, pp.1559÷1573, 2006.
[10] X. Zhang and M. McKenna, The Back-Propagation Algorithm on Grid and Hypercube Architecture," Technical Report RL90-9, Thinking Machines Corp., 1990.
[11] A. Gutierrez, F. Cavero, R. de Llano, J. Gregorio, Parallelization of a Neural Net Training Program in a Grid Environment, Proc. of 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp.258÷265, 2004.
[12] U. Seiffert, Artificial Neural Networks on Massively Parallel Computer Hardware, ESANN'2002 proceedings - European Symposium on Artificial Neural Networks, Bruges (Belgium), 24-26 April 2002, pp. 319-330.
[13] N. Šerbedžija, Simulating Artificial Neural Networks on Parallel Architectures, *IEEE Computer*, Vol.29, №3, pp.56÷63, 1996.
[14] S. Mahapatra, Mapping of Neural Network Models onto Systolic Arrays, Journal Parallel and Distributed Computing, Vol. 60, №6, pp.667÷689, 2000.
[15] Y. Chauvin, D. Rumelhart, Back Propagation: Theory, Architectures, and Applications, Lawrence Erlbaum Associates, 1995.
[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, Parallel Distributed Processing, volume 1. MIT Press, Cambridge, MA, 1986.
[17] R. Rogers, D. Skillicorn, Using the BSP Cost Model for Optimal Parallel Neural Network Training, Lecture Notes in Computer Science: Parallel and Distributed Processing, Springer, Vol.1388, pp.297÷305, 1998.