In *Proc. of the 10th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'98)*, October 28–31, 1998, Las Vegas, Nevada, USA, IASTED Press, pp. 555–560.

# DISTRIBUTED BACKPROPAGATION NEURAL NETWORKS ON A PVM HETEROGENEOUS SYSTEM

R. ANDONIE *
A. T. CHRONOPOULOS
Division of Computer Science,
The University of Texas at San Antonio,
6900 North Loop 1604 West, San Antonio, TX 78249-0677, USA
e-mail: {andonie, atc}@cs.utsa.edu

D. GROSU
H. GALMEANU
Department of Electronics and Computers,
Transylvania University
Politehnicii 1-3, 2200, Brasov, Romania
e-mail: {grosu, galmeah3}@vega.unitbv.ro

## Abstract

This study analyses how we can optimally implement the backpropagation training algorithm on a parallel architectures with heterogeneous processors. We use pattern-partitioning parallelization on a master-slave model based on the PVM (Parallel Virtual Machine) network. The training pattern set is distributed among the heterogeneous processors using a static approach (i.e., the mapping is unchanged during the learning process). Our optimization criterion is based on minimizing the execution time of backpropagation learning. We provide a mapping algorithm which guarantees minimum computation time. With respect to time minimization, we use an optimal number of the available processors. The computations and communications are overlapped, considering the following strategy: each processor has to perform a task directly proportional to it's speed. We show that the optimal mapping problem of the patterns is a polynomial-time problem, solvable by dynamic programming.

We have implemented our mapping method on a network of six PC's using the PVM message-passing system, and we have tested the parallel implementation of backpropagation on Sejnowski's NetTalk benchmark. Using the optimal allocation of patterns, the obtained time reduction of backpropagation learning were between 14.5% and 53.7%, compared to the equal allocation method. Higher gains in performance are expected for a larger number of varying speed computers in the network.

**Keywords:** backpropagation, heterogeneous system, PVM, optimal pattern mapping.

## 1 Introduction

Backpropagation [16] is the most widely used training algorithms for feedforward neural networks, in spite of the wellknown general inefficiency of this algorithm. Moreover, since it was one of the first general-purpose neural networks' learning algorithm, it became a standard and several authors tried to improve in several ways its performance. This has motivated researchers to study parallel implementations as a means to reduce the training time. References to their work may be found in [4], [6], [8], [9], [10], [11], [12], [14].

There is no consensus on how to simulate artificial neural networks on parallel machines. During the last years, researchers have been trying to achieve maximal performance on their favorite (or available) parallel machine. Backpropagation networks were implemented on almost all known general-purpose parallel architectures, including: linear arrays [15], multiple bus systems [6], message-passing multicomputers [3], hypercubes [10], transputer based architectures [8], [9], and LAN's of workstations [5].

Backpropagation can be parallelized by network-partitioning, by pattern-partitioning, or by a combination of these two schemes. In network-partitioning, nodes and weights of the neural network are partitioned among different processors, and thus the computations of node activations, node errors, and

*On leave of absence from the Department of Electronics and Computers, Transylvania University, Brasov, Romania

weight changes are parallelized. The idea of pattern-partitioning [13] is to distribute the training examples over the processors, i.e. slices the training set and assigns one slice to each processor while keeping a complete copy of the whole network in each processor node.

The implementation of a neural network on a heterogeneous parallel architecture give rise to a hard problem. This problem concerns the optimal mapping of the network and of the training patterns among the heterogeneous processors. This optimization model is generally a NP-complete integer (or mixed) programming problem which can be solved either directly (for instance, by branch-and-bound), or simplified heuristically to a polynomial problem. The mapping algorithms can be static or dynamic. In the static case, we assume that the mapping is unchanged throughout the learning process. In the dynamic case, we assume that the background workload is time varying; hence, it may be necessary to perform a remapping as workload changes.

Only few optimal mapping schemes have been reported to implement neural network algorithms on parallel architectures with heterogeneous processors. Chu and Wah presented an approximation algorithm for the optimal mapping of large neural networks on multi-computers, given a user-specified error degree that can be tolerated in the final mapping [3]. Saratchandran *et al.* [8], [9] optimized pattern partitioning in backpropagation learning on a heterogeneous array of transputers. They solved the optimization problem in two ways: by branch-and-bound and by genetic algorithms.

In this paper, we use the pattern-partitioning parallelization. This scheme is particularly suited for feedforward neural networks with backpropagation learning where the size of the training set is large compared to the size of the network. Meanwhile, pattern-partitioning is a coarse grained method. Its efficiency collapses when small training sets are handled and when many processors are used.

We use a master-slave heterogeneous model based on a PVM network. The training pattern set is distributed among the processors using a static approach. We succeeded in modeling the optimal mapping problem as a polynomial problem, solvable by dynamic programming.

## 2 The PVM Master-Slave Model

PVM is a software package that allows programs to be run on a heterogeneous network of Unix computers [7]. It consists of two parts, a daemon process that resides on all the computers making up the virtual machine and a PVM library containing user callable C functions for message passing, spawning processes, coordinating tasks, and modifying the virtual machine. The speed of communication between machines across a network is slow in comparison with that of dedicated concurrent processing hardware. It depends on the network and the machine load. Consequently, running parallel programs under PVM is only beneficial when communications are minimized. The major advantage offered by PVM is that it allows the processing power and large memory capacity of workstations to be exploited.

One of the most common programming model used in developing parallel application under PVM is the master-slave model. In this model we have a control program called master and a number of slave programs. The master program is responsible for spawning slave programs, initialization and collection of results. The slaves programs perform the computation on data allocated by the master or by themselves. The master-slave model involves no communication among the slaves. Only the master can communicate with slaves by message-passing.

The PVM master-slave communication is in some way "asymmetrical". The master can use a multiple-broadcast (managed and optimized by the PVM system) when sending one message to the slaves. This multiple-broad-cast can be regarded as a parallel process (from the user's point of view). The message-passing from the slaves to the master is different, since the master can receive only one message at a time. Therefore, the user has to schedule in an optimal way the message-passing from the slaves to the master.

## 3 Distributed Backpropagation

Backpropagation trains a given feedforward neural network for a given set of learning patterns. The training of the neural network can be viewed as discovering values for its weights in order to match the effective outputs of the network with the desired outputs, for each input pattern.

In backpropagation learning, weights can be updated in two ways:

1. In the per-pattern regime the weights are updated after each training pattern is presented;

2. In the set-training regime the weight increments are computed for each training pattern. The increments are summed for all patterns and the weights are updated with the total increment after all patterns have been presented one time [16].

Pattern-partitioning schemes to parallelize backpropagation are applicable only to set-training updating [10]. Therefore, our pattern-partitioning scheme is based on a set-training regime.

We assume a master-slave model with $n$ slaves. The training set $S$ is partitioned into $n$ subsets, $S_i,\ i = 1, 2, ..., n$. These training subsets are distributed to

the available processors ($n$ slaves). Each slave process contains a complete copy of the whole neural network.

One epoch of the backpropagation algorithm has the following coarse description:

1. The weight changes and bias for the current epoch are initialized to zero.

2. Each slave process ($P_i$) carries out the forward and the backward phase for each pattern assigned to it.

3. Each slave process also accumulates the weight changes and error according to the local patterns.

4. Each slave process sends the weight changes and errors to the master. The master process computes the sum of all weight changes and of all errors.

5. The master broadcasts the new weights to all slaves. The weights are updated on each slave. The master checks if the convergence is reached.

The timing diagram for an epoch is shown in Figure 1. In this diagram we used the following notations:

- $P_1, P_2, \ldots, P_n$ are the slave processes.

- $M$ is the master process.

- $t_{init}(i)$ is time taken to initialize the weight changes and error.

- $P(i)$ is the number of patterns allocated to the slave process $P_i$.

- $T(i)$ is time taken to perform the forward and backward phase of the algorithm for a single pattern.

- $t_{comm}$ is time taken to send the weight changes and errors from the slaves to the master.

- $t_{bcast}$ is time taken to broadcast the updated weights.

In order to obtain the minimum epoch time we have to overlap communication time ($t_{comm}$) with computation time and to find a proper pattern distribution among the processors.

# 4 Optimization of pattern mapping

To schedule is to allocate a set of tasks or jobs to resources such a way that optimizes the use of these resources. If these tasks are not interdependent, the problem is known as *task allocation*. The general

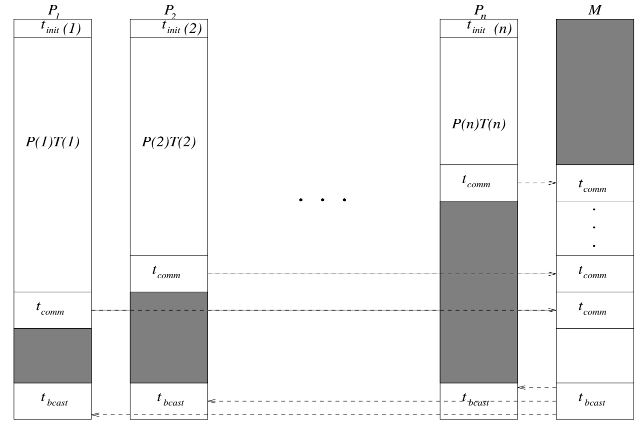


Figure 1: The timing diagram for an epoch

problem of determining an optimal schedule on a heterogeneous parallel architecture is known to be NP-complete [18]. Nevertheless, simplified versions of task allocation could be solved by dynamic programming [1], a method which leads usually to a polynomial solution.

Our pattern-mapping optimization problem can be formulated in the following way. We have to distribute $p$ patterns among $n$ heterogeneous processors, minimizing the parallel computation time. In other words, we have to minimize the parallel computation time for one epoch, on $n$ heterogeneous processors, considering also the weight passing from each slave to the master.

We shall cascade the computation times on the $n$ processors in the following way. The computation time for one epoch on a faster processor has to overlap (as much as possible) with computation plus message passing times on a slower processor. This would make the faster processor to be the last one sending it's weights to the master after one epoch. This means also that we have to map more patterns to a faster processor than to a slower one. Hence, we prefer to use the fastest processors over the slowest ones. Moreover, we shall actually use an optimal number of processors, which is a subset of the available $n$ processors. This subset consists of the fastest processors.

We shall use the following notations:

- $P$ = a vector of $n$ elements, where $P(i) \geq 0$ is the number of patterns assigned to processor $i$, $1 \leq i \leq n$, $P(1) + \ldots + P(n) = p$.

- $T$ = a vector of $n$ elements, where $T(i)$ is the backpropagation processing time for one pattern (and one cycle) assigned to processor $i$, $1 \leq i \leq n$.

We shall suppose from now on, without restricting the generality, that $T(1) \leq T(2) \leq \ldots \leq T(n)$.

Our objective is to find an optimal vector $P^*$ ($P(i) \geq 0$, $1 \leq i \leq n$, $P(1) + \ldots + P(n) = p$), minimizing the parallel computation time. In this case,

it is obviously better to send more work to the faster processors (those with a low $T(i)$). The first processor (the fastest) has to be the last one sending the weights to the master. The parallel processing time is proportional to $T(1)P(1)$ (and this means to $P(1)$, since $T(1)$ is constant). Therefore, we have to minimize $P(1)$.

The task allocation problem can be simplified by considering the following assumption.

### Assumption 1

The vector $P^*$ ($P^*(i) \geq 0, 1 \leq i \leq n, P^*(1) + \ldots + P^*(n) = p$) is optimal if

$$T(i)P^*(i) \geq T(i+1)P^*(i+1)+t_{comm}, \qquad 1 \leq i \leq n-1$$

This assumption reduces the size of the search space, giving us the possibility to find a polynomial solution to the optimization problem. The meaning of this restriction is that we always try to have no idle time periods for the fastest processors, by keeping them as much as possible busy. The result is an unequal pattern distribution with overlapping of communication time and computation time.

For any vector $P^*$ respecting Assumption 1, we have the following two properties (the proofs are obvious, since $T(1) \leq T(2) \leq \ldots \leq T(n)$):

### Property 1

$$P^*(i) \geq P^*(i+1), \qquad 1 \leq i \leq n-1$$

### Property 2

$$P^*(i) = 0 \Rightarrow P^*(i+1) = 0, \qquad 1 \leq i \leq n-1$$

Instead of minimizing the number of patterns allocated to the fastest processor, we can reverse the problem, maximizing the number of patterns allocated to the slowest processor. Based on Assumption 1, we can find now an optimal solution $P^*$ by dynamic programming.

### Dynamic programming solution:

We define a gain array $g$ of $n \times p$ elements, where $g(i,j)$, for $1 \leq i \leq n$ and $1 \leq j \leq p$, is the maximum number of patterns allocated to processor $i$ (the slowest) if we distribute $j$ patterns to processors $1, 2, \ldots, i$.

We initialize the array as follows:

$$g(1,j) = j, \qquad j = 1, \ldots, p$$

$$g(i,1) = 0, \qquad i = 2, \ldots, n$$

We have to compute the rest of the elements of $g$. The optimality principle holds and we have:

$g(i,j) = 0 \qquad$ if $g(i-1,j-1)T(i-1) \leq T(i)+t_{comm}$

$g(i,j) = \max\{k | g(i-1, j-k)T(i-1) > kT(i) + t_{comm}\}, \qquad$ otherwise

### Observations on the implementation:

We can complete $g$ line by line, or column by column. We can reduce computation time (up to a multiplicative constant) observing that:

If for some $j = 1, \ldots, p$ :

$$g(i,j) = 0 \text{ then } g(q,j) = 0, \quad \text{for } q = i+1, \ldots, n$$

This is in accordance with Property 2.

Another observation is that:

$$g(i,j) = 0 \Rightarrow g(i+1, j+1) = 0$$

After completing $g$, the solution to our optimization problem can be obtained backwards:

$P^*(n) = g(n,p)$

$P^*(n-1) = g(n-1, p - P^*(n))$

$\vdots$

In general:

$P^*(i) = g(i, (p - P^*(n) - P^*(n-1) - \ldots - P^*(i+1))), \qquad 1 \leq i \leq n-1$

### Efficiency of the algorithm:

An element $g(i,j)$ can be computed in $O(p)$ time. The whole array can be completed in $O(np^2)$ time. Subsequently, the optimal solution is found in $\Theta(n)$ time.

## 5 Experimental results

To demonstrate the performance of our optimal mapping method we conducted our experiments on a network of PC (Pentium 166Mhz) using PVM message passing system. The parallel implementation of backpropagation was tested on NetTalk benchmark. The NetTalk is a text to phoneme transcription network proposed by Sejnowski and Rosenberg [17]. This network is a feedforward neural network with three layers. Input to the network represent a sequence of seven consecutive characters from sample English text. The network learns to map these to a representation of a single phoneme corresponding to the fourth character in the sequence. We used a binary encoding for characters and phonemes. The neural network has 36 input neurons, 80 hidden neurons and 9 output neurons. In our experiments we used the NetTalk training set, with 7570 patterns.

The set of experiments was done on a simulated heterogeneous computers. In order to simulate heterogeneity, we introduced different processing delays on each computer. The backpropagation processing time for one pattern considering the NetTalk network on a Pentium PC 166Mhz is $2.3ms$. We have considered $T[1] = 2.3ms$ and $T[i+1] = T[i] + 2.5ms, i = 1, \ldots, 5$. The number of computers are scaled from 3 to 6. For

Table 1: Pattern allocation for heterogeneous computers

| pat./ ep. | No. of processors | | | |
|---|---|---|---|---|
| | 3 | 4 | 5 | 6 |
| 400 | 247, 100, 53 | 233, 93, 48, 26 | - | - |
| 500 | 304, 126, 70 | 282, 116, 64, 38 | - | - |
| 600 | 359, 153, 88 | 331, 140, 80, 49 | 317, 133, 75, 46, 29 | 310, 130, 73, 44, 27, 16 |
| 700 | 414, 180, 106 | 381, 163, 95, 61 | 362, 145, 89, 57, 38 | 352, 150, 86, 54, 35, 23 |

each such configuration we have used two types of pattern allocation. The first one was the usual approach in which the patterns are equally distributed among computers. This type of allocation will produce a poor execution time due to the presence of waiting time associated to communications. The second one was the optimal allocation based on the overlapping of communications and computations, and on the following politics: the allocation of patterns is proportional to computer's speed. The solution of the optimization problem was obtained using the algorithm described in Section 4. In this algorithm we used $t_{comm} = 88ms$ which was found experimentally.

The performance of the two allocation schemes was reported as execution time for an epoch and as number of CUPS (Connection Updates per Second). The number of CUPS can be calculated as in the following formula:

$$CUPS = \frac{(no.\ of\ weights) \times (no.\ of\ patterns)}{time\ for\ an\ epoch}$$

For the neural network used in our experiments the number of weights is 3600.

For example, consider a network of 4 computers and the number of patterns for an epoch 400. Using our optimal allocation method, the optimal distribution of patterns is found to be 233, 93, 48, 26. The resulting epoch time is 1321 ms. This is equivalent to 1.090 MCUPS. Using equal distribution i.e. 100 patterns per computer, the epoch time is 1550 ms (927 KCUPS). The gain in performance was about 14.7%. This example led us to the conclusion that the equal distribution used in the case of heterogeneous network is not the best way to allocate the patterns.

The results obtained solving the optimization problem considered in Section 4 are shown in Table 1.

In Table 2 we show the epoch time considering the two allocation methods and different number of patterns for an epoch. The execution time for an epoch using the optimal allocation is considerably smaller than in the case of using the equal allocation method. The improvement in performance is between 14.7% and 53.7%. The maximum values are obtained for 600 and 700 patterns per epoch.

Table 2: Epoch time (ms)

| patterns/ epoch | allocation method | No. of processors | | | |
|---|---|---|---|---|---|
| | | 3 | 4 | 5 | 6 |
| 400 | equal | 1963 | 1550 | - | - |
| | optimal | 1247 | 1321 | - | - |
| 500 | equal | 1765 | 1793 | - | - |
| | optimal | 1275 | 1237 | - | - |
| 600 | equal | 2576 | 2715 | 2800 | 2846 |
| | optimal | 1600 | 1530 | 1630 | 1800 |
| 700 | equal | 2854 | 2872 | 3180 | 3113 |
| | optimal | 1827 | 1554 | 1890 | 1895 |

The performance in KCUPS for each allocation is summarized in Table 3.

Table 3: Performance in KCUPS

| patterns/ epoch | allocation method | No. of processors | | | |
|---|---|---|---|---|---|
| | | 3 | 4 | 5 | 6 |
| 400 | equal | 733 | 927 | - | - |
| | optimal | 1154 | 1090 | - | - |
| 500 | equal | 1019 | 1003 | - | - |
| | optimal | 1411 | 1455 | - | - |
| 600 | equal | 838 | 795 | 771 | 758 |
| | optimal | 1350 | 1411 | 1325 | 1200 |
| 700 | equal | 882 | 877 | 792 | 809 |
| | optimal | 1379 | 1621 | 1333 | 1329 |

# 6 Final remarks

Our optimization criterion is based on minimizing the execution time of backpropagation learning. We provide a mapping algorithm which guarantees minimum computation time. With respect to time minimization, we use an optimal number of the available processors. The computations and communications are overlapped, considering the following strategy: each processor has to perform a task directly proportional to it's speed. The fastest processor is always the latest one in computing one epoch. Using the optimal allocation of patterns we have obtained improvements in performance between 14.7% and 53.7%, compared to the equal allocation method. Higher gains in per-

formance are expected for a larger number of varying speed computers in the network.

Variants of the basic backpropagation algorithm with better convergence properties can directly be implemented using the mapping method reported here.

Our optimal allocation method can be useful also in the case of homogeneous systems. In this case only the overlapping of computations and communications is exploited.

A possible approach would be to solve the pattern partitioning problem using another neural network. It would not be for the first time neural computation is used for task allocation in a heterogeneous network [2]. Although, using such a connectionist method, we do not expect an improvement, compared to our dynamic programming optimization. A more promising future research would be to use our optimal pattern-partitioning scheme in a general task allocation device for distributed computing.

We plan to extend our experiments in two directions. First, to use dynamic pattern mapping. Second, to consider other PVM-based parallel architectures.

# References


[1] B. Boffey. *Distributed Computing-Associated Combinatorics Problems.* Blackwell Scientific Publications, Oxford, 1992.

[2] M. Cena, M. L. Crespo, and R. Gallard. Transparent remote execution in LAHNOS by means of neural networks device. *ACM Operating Systems Reviews*, 29:17–28, 1995.

[3] L. C. Chu and B. W. Wah. Optimal mapping of neural-network learning on message-pasing multicomputers. *J. of Parallel and Distributed Computing*, 14:319–339, 1992.

[4] M. Cosnard, J. C. Mognot, and H. Paugam-Moisy. Implementation of multilayer neural networks on parallel architectures. In *Proc. 2nd IEEE Int. Spec. Seminar on the Design and Application of Parallel Digital Processors*, pages 43–47, April 1991.

[5] M. Crespo, F. Piccoli, M. Printista, and R. Gallard. Parallel shaping of backpropagation neural networks in a workstations-based distributed system. In *Proc. EIS'98 Int. ICSC Symp. on Engineering of Intelligent Systems*, pages 709–715. ICSC Academic Press, February 1998.

[6] A. El-Amawy and P. Kulasinghe. Algorithmic mapping of neural networks onto multiple bus systems. *IEEE Trans. Parallel and Distributed Syst.*, 8(1):130–136, January 1997.

[7] Al Geist et all. *PVM:Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing.* MIT Press, Cambridge, Massachusetts, 1994.

[8] S. K. Foo, P. Saratchandran, and N. Sundararajan. *Parallel Implementation of Backpropagation Neural Networks on Transputers.* World Scientific, Singapore, 1996.

[9] S. K. Foo, P. Saratchandran, and N. Sundararajan. Parallel implementation of backpropagation neural networks on a heterogeneous array of transputers. *IEEE Trans. on Syst., Man and Cybern. Part B: Cybernetics*, 27(2):118–126, February 1997.

[10] V. Kumar, S. Shashi, and M. B. Amin. A scalable parallel formulation of the backpropagation algorithm for hypercubes and related architectures. *IEEE Trans. Parallel and Distributed Syst.*, 5:1073–1090, 1994.

[11] Q. M. Malluhi, M. A. Bayoumi, and T. R. N. Rao. Efficient mapping of ANNs on hypercube massively parallel machines. *IEEE Trans. Comput.*, 44:769–779, 1995.

[12] B. Nikola. Simulating artificial neural networks on parallel architectures. *IEEE Computer Magazine*, 29(3):56–63, March 1996.

[13] H. Paugam-Moisy. Parallel neural computing based on network duplicating. In I. Pitas, editor, *Parallel Algorithms for Digital Image Processing, Computer Vision, and Neural Networks*, pages 305–340. John Wiley & Sons, 1993.

[14] A. Petrowski, G. Dreyfus, and C. Girauld. Performance analysis of pipelined backpropagation parallel algorithm. *IEEE Trans. on Neural Networks*, 4:970–981, 1993.

[15] D. A. Pomerleau, G. S. Gsciora, D. S. Touretzky, and H. T. Kung. Neural network simulation at warp speed: How to get 17 million connections per second. In *Proc. Int. Conf. on Neural Networks*, volume 2, pages 143–150, July 1988.

[16] D. E. Rummelhart and J. L. McClelland. *Parallel Distributed Processing, Exploration in Microstructures of Cognition*, volume 1. MIT Press, Cambridge, Massachusetts, 1986.

[17] T. J. Sejnowski and C. R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.

[18] J. Xu. Multiprocessor scheduling of processes with release times, deadlines, precedence, and exclusion relations. *IEEE Trans. Software Eng.*, 19:139–154, 1993.