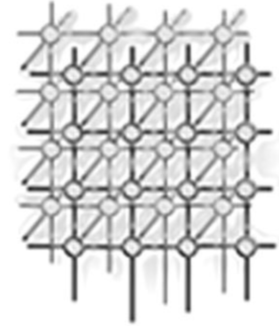
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2006; **18**:1559–1573
Published online 25 October 2005 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.987



An efficient concurrent implementation of a neural network algorithm

R. Andonie[1,*,†], A. T. Chronopoulos[2], D. Grosu[3] and H. Galmeanu[4]

[1] *Computer Science Department, Central Washington University, Ellensburg, WA 98926, U.S.A.*
[2] *Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249, U.S.A.*
[3] *Department of Computer Science, Wayne State University, Detroit, MI 48202, U.S.A.*
[4] *Department of Electronics and Computers, Transylvania University, 2200 Brasov, Romania*

SUMMARY

The focus of this study is how we can efficiently implement the neural network backpropagation algorithm on a network of computers (NOC) for concurrent execution. We assume a distributed system with heterogeneous computers and that the neural network is replicated on each computer. We propose an architecture model with efficient pattern allocation that takes into account the speed of processors and overlaps the communication with computation. The training pattern set is distributed among the heterogeneous processors with the mapping being fixed during the learning process. We provide a heuristic pattern allocation algorithm minimizing the execution time of backpropagation learning. The computations are overlapped with communications. Under the condition that each processor has to perform a task directly proportional to its speed, this allocation algorithm has polynomial-time complexity. We have implemented our model on a dedicated network of heterogeneous computers using Sejnowski's NetTalk benchmark for testing. Copyright © 2005 John Wiley & Sons, Ltd.

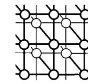
KEY WORDS: backpropagation; heterogeneous system; pattern allocation; parallel neural computing

*Correspondence to: Razvan Andonie, Computer Science Department, Central Washington University, Ellensburg, WA 98926, U.S.A.
†E-mail: andonie@cwu.edu

Contract/grant sponsor: NASA; contract/grant number: NAG 2-1383 (1999–2001)
Contract/grant sponsor: State of Texas Higher Education Coordinating Board through the Texas Advanced Research/Advanced Technology Program; contract/grant number: ATP 003658-0442-1999
Contract/grant sponsor: NSF; contract/grant numbers: ASC-9634775 and CCR-0312323

Copyright © 2005 John Wiley & Sons, Ltd.

Received 12 November 2004
Revised 13 April 2005
Accepted 30 April 2005



1. INTRODUCTION

Backpropagation is a widely used training algorithm for feedforward neural networks, in spite of the well-known general inefficiency of this algorithm. Moreover, since it was one of the first general-purpose neural network learning algorithms, it became a standard and several authors tried to improve its performance in several ways. This has motivated researchers to study parallel implementations as a means to reduce the training time. 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: multiple bus systems [1]; message-passing multicomputers [2]; hypercubes [3]; transputer-based architectures [4]; 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 weight changes are parallelized. The idea of *pattern partitioning* [6] is to distribute the training examples over the processors, i.e. it slices the training set and it 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 gives rise to a complex problem. This problem concerns the optimal mapping of the network and of the training patterns among the heterogeneous processors. This optimization problem 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 a few mapping schemes have been reported to implement neural network algorithms on parallel architectures with heterogeneous processors. Chu and Wah [2] presented an approximation algorithm for the mapping of large neural networks on multi-computers, given a user-specified error degree that can be tolerated in the final mapping. Foo *et al.* [4] 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.

There exist other approaches for optimal data partitioning in distributed systems. Notable are some works in divisible load theory [7,8] where a divisible load can be arbitrarily partitioned and distributed to more than one computer to achieve a faster execution time. For non-divisible or discrete loads there are (non-optimal) strategies that have been used, such as the *equal* and *rectilinear* allocation [9]. These strategies have been used for Grid problems but not for neural networks.

In this paper, we implement the backpropagation algorithm using the pattern-partitioning scheme. We use a manager–worker architecture defined on a network of heterogeneous computers. The computers are assumed to have the capability to perform computation and communication simultaneously, which is the case in most existing systems. The pattern-partitioning scheme leads to a pattern allocation problem for mapping the training patterns onto the computers. This allocation problem was first formulated in [10] and discussed in a different framework in [11]. Our pattern-mapping algorithm takes into account the CPU speed of the processors and overlaps computation and communication.

Copyright © 2005 John Wiley & Sons, Ltd.

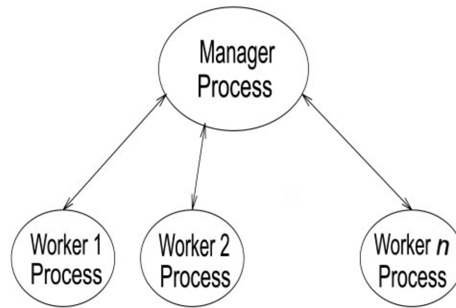


Figure 1. The manager–worker model.

The rest of the paper is organized as follows. In Section 2, we define our dedicated parallel architecture for backpropagation. Section 3 describes the dynamic programming solution for the pattern-mapping problem. In Section 4 we present the experimental results obtained on Sejnowski's NetTalk benchmark [12]. We compare our mapping algorithm with the equal and proportional pattern allocation algorithms. The proportional allocation takes into account the CPU speed of the computers but does not overlap communication with computation as we do. Section 5 concludes with some closing remarks and open problems.

2. A PARALLEL ARCHITECTURE FOR BACKPROPAGATION LEARNING

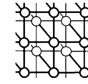
One of the most common programming models used in developing concurrent applications on a network of computers (NOC) is the manager–worker model. In this model we have a control program called the manager and a number of worker programs. The manager program is responsible for spawning worker programs, initialization and collection of results. The worker programs perform the computation on data allocated by the manager or by themselves. The manager–worker model involves no communication among the workers. Only the manager can communicate with workers by message-passing. The structure of the manager–worker model is shown in Figure 1.

We now describe the mapping of the backpropagation onto the computers. 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 the following 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 once [13].

Copyright © 2005 John Wiley & Sons, Ltd.



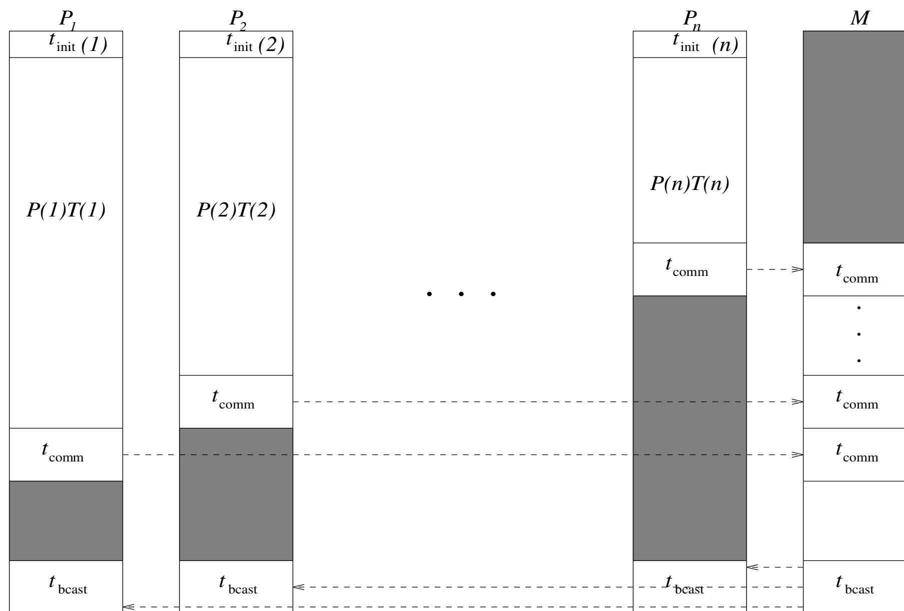


Figure 2. The timing diagram for an epoch.

Pattern-partitioning schemes to parallelize backpropagation are applicable only to set-training updating [3]. Therefore, our pattern-partitioning scheme is based on a set-training regime.

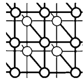
We assume a manager–worker model with n workers (processors). The training set S is partitioned into n subsets, S_i, $i = 1, 2, \ldots, n$. These training subsets are distributed to the n processors. Each worker process contains a complete copy of the whole neural network.

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 worker process (P_i) carries out the forward and the backward phase for each pattern assigned to it.
3. Each worker process also accumulates the weight changes and error according to the local patterns.
4. Each worker process sends the weight changes and errors to the manager. The manager process computes the sum of all weight changes and of all errors.
5. The manager broadcasts the new weights to all workers. The weights are updated on each worker. The manager checks if the convergence is reached.

Remark. We do not use a reduce operation for implementing the communication required in item 4 above, because in a LAN the reduction tree must be mapped to a single bus and this is very inefficient. More importantly, the manager is needed to assign patterns according to our algorithm.

Copyright © 2005 John Wiley & Sons, Ltd.



The timing diagram for an epoch is shown in Figure 2. In this diagram we used the following notation.

- $P_1, P_2, \ldots, P_n$ are the worker processes.
- M is the manager process.
- $t_{\text{init}}(i)$ is the time taken to initialize the weight changes and error.
- $P(i)$ is the number of patterns allocated to the worker process P_i.
- $T(i)$ is the time taken to perform the forward and backward phase of the algorithm for a single pattern on the worker processor P_i.
- t_{comm} is the time taken to send the weight changes and errors from the workers to the manager.
- t_{bcast} is the time taken to broadcast the updated weights.
- T_n is the parallel execution time on n processors.

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.

3. PATTERN-MAPPING OPTIMIZATION

The allocation of tasks (or jobs) in distributed systems may be considered a special case of task scheduling, without any precedence relationships among the execution tasks. The purpose of a task allocation technique is to find some task assignment in which the total cost due to interprocessor communication and task execution is minimized. The task allocation problem is known to be NP-complete [14]. Optimal algorithms are obtained in very restricted cases. For instance, simplified versions of task allocation could be solved by dynamic programming [15], a method which usually leads to a solution in polynomial time. The intractability of the problem has led to the introduction of many heuristics.

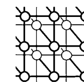
Our pattern-mapping optimization is a task allocation problem. We have to distribute p patterns among n heterogeneous processors, minimizing T_n. In other words, we have to minimize T_n for one epoch, on n heterogeneous processors, considering also the weights transmission from each worker to the manager. The computations are overlapped with communications, considering the following strategy: each processor has to perform a task directly proportional to its speed.

We 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 its weights to the manager after one epoch. This also means 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 actually use a subset of the available n processors. This subset consists of the fastest processors.

We use the following notation:

- **P** is a vector of n elements, where $P(i) \geq 0$ is the number of patterns assigned to processor i, $1 \leq i \leq n$, $P(1) + \cdots + P(n) = p$;
- **T** is a vector of n elements, where $T(i)$ is the backpropagation processing time for one pattern (and one cycle) assigned to processor i, $1 \leq i \leq n$.

Copyright © 2005 John Wiley & Sons, Ltd.



Since we can label the processors in any order, let us assume from now on, without loss of generality, that $T(1) \leq T(2) \leq \cdots \leq T(n)$. The objective is to find a value for vector **P** minimizing the parallel execution time T_n. In this case, it is obviously better to send more work to the faster processors (those with a low $T(i)$). We can simplify this optimization problem by considering the following assumption.

Assumption 1. *We have*

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

We call the vector $\mathbf{P}^*$ ($P^*(i) \geq 0, 1 \leq i \leq n,\ P^*(1)+\cdots+P^*(n) = p$) *optimal* if it maximizes the number of patterns allocated to the slowest available processor (as can be seen from Figure 2), under the constrains of Assumption 1.

Assumption 1 reduces the size of the search space, giving us the possibility to find a polynomial solution to the optimization problem. The optimal solution $\mathbf{P}^*$ is not necessarily an optimal solution for the general optimization problem (i.e. the pattern-mapping optimization without Assumption 1) and this can be easily shown by an example (see Example 2).

A similar result can be achieved if, under the same constraints, instead of maximizing the number of patterns processed by the slowest processor, we minimize the number of patterns processed by the fastest processor. In this case, the optimization is easier to illustrate. The first processor (the fastest) has to be the last one sending the weights to the manager. Intuitively, T_n 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 reason for Assumption 1 is that we always try to have no idle time periods for the fastest processors, by keeping them busy as much as possible.

We chose to maximize the number of patterns processed by the slowest processor because this can be easier formulated by a dynamic programming approach.

The next proposition will be used later. For any vector **P** ($P(i) \geq 0, 1 \leq i \leq n$, $P(1) + \cdots + P(n) = p$) for which

$$T(i)P(i) \geq T(i+1)P(i+1) + t_{\text{comm}}, \quad 1 \leq i \leq n-1 \tag{2}$$

we have the following property (the proof is obvious, since $T(1) \leq T(2) \leq \cdots \leq T(n)$).

Property 1. We have

$$P(i) \geq P(i+1), \quad 1 \leq i \leq n-1 \tag{3}$$

3.1. Dynamic programming (DP) solution

We maximize the number of patterns allocated to the slowest processor, building 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 on processors $1, 2, \ldots, i$.

We initialize the array as follows:

$$g(1, j) = j, \quad j = 1, \ldots, p \tag{4}$$

$$g(i, 1) = 0, \quad i = 2, \ldots, n \tag{5}$$

Copyright © 2005 John Wiley & Sons, Ltd.

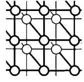


Table I. The gain array g for $n = 3$ and $p = 15$.

i/j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4
3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	2

We have to compute the rest of the elements of g. The optimality principle holds, since Assumption 1 is a recursive relation, and we have

$$g(i, j) = \begin{cases} 0 & \text{if } g(i-1, j-1)T(i-1) \\ & \quad \le T(i) + t_{\text{comm}} \\ \max\{k \mid g(i-1, j-k)T(i-1) > kT(i) + t_{\text{comm}}\} & \text{otherwise} \end{cases} \tag{6}$$

We can complete g line by line or column by column. We can reduce the time to compute the gain array using the following observation.

If, for some j, we have $g(i, j) = 0$, then

$$g(i+1, j) = 0 \tag{7}$$

(this is in accordance with Property 1)

$$g(i+1, j+1) = 0 \tag{8}$$

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,

$$P^*(i) = g(i, (p - P^*(n) - P^*(n-1) - \cdots - P^*(i+1))), \quad 1 \le i \le n-1 \tag{9}$$

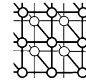
Example 1. In Table I we present the gain array g for a system of $n = 3$ processors. The total number of patterns is $p = 15$. The execution times for one pattern on each processor are the following: $T(1) = 1$, $T(2) = 2$, $T(3) = 3$. We assume the communication time: $t_{\text{comm}} = 0.5$.

The solution found by our allocation method is $P^*(1) = 9$, $P^*(2) = 4$, $P^*(3) = 2$, that gives a total execution time of 9.5.

If we take $p = 7$ in the previous example, the DP allocation will give $P^*(3) = 0$. This means that we do not have to use all three available processors. The minimum execution time is achieved when using only the first two fastest processors.

There are situations in which our algorithm will not produce the optimal solution. We give such an example below.

Copyright © 2005 John Wiley & Sons, Ltd.



Example 2. We consider a system of $n = 2$ processors. The total number of patterns is $p = 7$. The execution times for one pattern on each processor are the following: $T(1) = 5, T(2) = 7$. We assume the communication time $t_{\text{comm}} = 1$. The solution found by our allocation method (under Assumption 1) is $P^*(1) = 5$, $P^*(2) = 2$, which gives a total execution time of 26. We can find another solution that gives lower execution time. This solution is $P(1) = 4$, $P(2) = 3$. It violates Assumption 1 (i.e. processor 1, the fastest, terminates before processor 2) but the overall computation time is 22 which is less than 26.

Computational complexity

The complexity of the DP algorithm is $O(np \log p)$. This follows from: (1) an element $g(i, j)$ can be computed in $O(\log p)$ time using binary search; (2) the whole array can be completed in $O(np)$ time; (3) the backward phase of the dynamic programming algorithm is in $\Theta(n)$ time.

3.2. Reducing the complexity of DP algorithm

The complexity of the DP algorithm can be reduced based on the following observation. If there is a sufficient number of patterns we can allocate them in two phases.

- *Phase 1*: We allocate most patterns in constant time so that all the processors will finish execution at the same time.
- *Phase 2*: We apply DP on the remaining patterns.

We make the following assumptions: (i) all n processors will be used in the most efficient way; (ii) t_{comm} is not negligible compared with the execution time of one pattern on any processor and so we want to overlap communication and computation.

The following example illustrates this technique.

Example 3. We consider a system of $n = 3$ processors. The total number of patterns is $p = 77$. The execution times for one pattern on each processor are the following: $T(1) = 2$, $T(2) = 3$, $T(3) = 5$. We assume the communication time $t_{\text{comm}} = 1$. We compute the least common multiplier (*lcm*) of $T(i)$, $i = 1, 2, 3$. In this case $lcm = 30$ and it represents the execution time for one time block. We allocate two time blocks on each processor as in Figure 3. In this figure the dark segments represents t_{comm}.

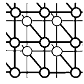
A formal description of this procedure is as follows. Let T_{block} be the *lcm* of $T(i)$, $i = 1, \ldots, n$. We have two cases: (I) when $t_{\text{comm}} \leq T_{\text{block}}$ and (II) when $t_{\text{comm}} > T_{\text{block}}$.

Case I.

1. Since the communication is to be overlapped with computation, we subtract the number of patterns $p_c = \sum_{i=1}^{n} \lfloor (n - i) t_{\text{comm}} / T(i) \rfloor$ whose execution will overlap with communication. Then, we determine the number k of possible stages of time blocks that can be formed:

$$k = \left\lfloor \frac{p - p_c}{T_{\text{block}} \sum_{i=1}^{n} (1/T(i))} \right\rfloor$$

Copyright © 2005 John Wiley & Sons, Ltd.



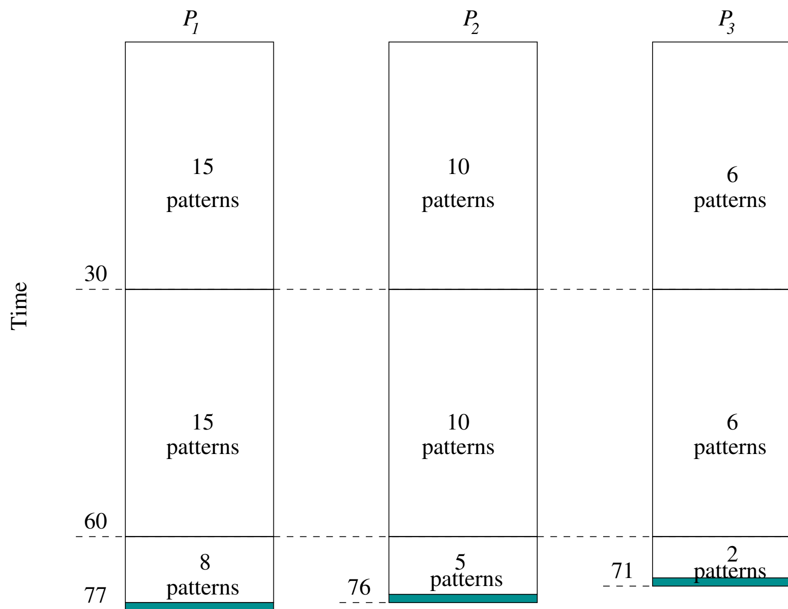


Figure 3. Pattern allocation for Example 3.

2. We allocate $k(T_{\text{block}}/T(i))$ patterns to processor i. Then apply DP algorithm on p' patterns, where $p' = p - kT_{\text{block}} \sum_{i=1}^{n} (1/T(i))$.

Case II. We find the minimum k_c such that $T'_{\text{block}} = k_c T_{\text{block}} > t_{\text{comm}}$. Then apply the procedure (I) with T'_{block} instead of T_{block}.

Using the procedures described above we can reduce the complexity of DP algorithm to $O(n^2 \log n)$. This can be shown as follows:

$$p' \leq n \frac{T_{\text{block}}}{T(1)} \Rightarrow p' = O(n) \tag{10}$$

From the remark above, the complexity of DP is in $O(np' \log p')$. This implies that the complexity is in $O(n^2 \log n)$. This is an important reduction in complexity because in most practical situations $p \gg n$.

Remark. One can generalize this approach to consider the case when it is more efficient to use fewer than n processors. Then all $p_c = \sum_{i=1}^{k} \lfloor (k-i)t_{\text{comm}}/T(i) \rfloor$ for $k = 1, \ldots, n$ must be examined.

4. EXPERIMENTAL RESULTS

To demonstrate the performance of our DP allocation algorithm we conducted our experiments on a NOC using the PVM [16] message-passing system. The network is dedicated to this application so

Copyright © 2005 John Wiley & Sons, Ltd.

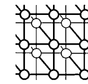


Table II. Processing time for one pattern (μs).

Workstation	P1	P2–P3	P4–P8
Time	383	467	532

that no other users are allowed to use it. The parallel implementation of backpropagation was tested on NetTalk benchmark. The NetTalk is a text to phoneme transcription network proposed by Sejnowski and Rosenberg [12]. This network is a feedforward neural network with three layers. Input to the network represents 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 nine output neurons. In our experiments we used the NetTalk training set, with 7560 patterns.

The set of experiments was done on a heterogeneous system composed of one Pentium PC 500 MHz (P1) and two Pentium PC 440 MHz (P2, P3) and five Pentium PC 400 MHz (P4–P8) connected by an Ethernet (10 Mb) network. The backpropagation processing times for one pattern considering the NetTalk network on each workstation are presented in Table II.

We have considered clusters of n computers with $n = 2, 3, 4, 5, 6, 7$ and 8. For each such configuration we have used three pattern allocation methods.

(1) *Equal allocation method*. The patterns are equally distributed among computers. This type of allocation will produce a poor execution time due to the fact that slower machines will adversely dominate the execution time. The communication steps are performed after the computation and they do not overlap.

(2) *Proportional allocation method*. The patterns are allocated in proportion to the CPU speeds. This is suitable for heterogeneous systems. The allocation can be computed in $O(n)$ time. The communication is not overlapped with computation. The patterns are allocated using the method presented in Section 3.2 (Case I, item 1) and the remaining patterns (p') are allocated according to the following equations:

$$P(i) = \left\lceil \frac{p'/T(i)}{\sum_{j=1}^{n} 1/T(j)} \right\rceil, \quad i = 1, \ldots, n-1 \tag{11}$$

$$P(n) = p' - \sum_{j=1}^{n-1} P(j) \tag{12}$$

(3) *The DP allocation method*. The DP allocation method is based on overlapping communications and computations. The solution of the optimization problem was obtained using the algorithm described in Section 4. In this algorithm we used $t_{\mathrm{comm}} = 4$ ms, which was found experimentally.

The performance of the three allocation schemes is characterized by the execution time (T_n) for one epoch. The allocations of patterns to computers for the three methods are shown in Table III.

Copyright © 2005 John Wiley & Sons, Ltd.

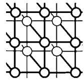


Table III. Pattern allocation for heterogeneous computers.

Number of processors	Allocation method	Computers							
		P1	P2	P3	P4	P5	P6	P7	P8
8	Equal	945	945	945	945	945	945	945	945
	Prop	1214	993	993	872	872	872	872	872
	DP	1856	1360	1199	911	770	629	488	347
7	Equal	1080	1080	1080	1080	1080	1080	1080	—
	Prop	1370	1123	1123	986	986	986	986	—
	DP	1918	1412	1251	957	815	674	533	—
6	Equal	1260	1260	1260	1260	1260	1260	—	—
	Prop	1576	1291	1291	1134	1134	1134	—	—
	DP	2029	1503	1342	1037	895	754	—	—
5	Equal	1512	1512	1512	1512	1512	—	—	—
	Prop	1854	1519	1519	1334	1334	—	—	—
	DP	2214	1655	1494	1169	1028	—	—	—
4	Equal	1890	1890	1890	1890	—	—	—	—
	Prop	2249	1845	1845	1621	—	—	—	—
	DP	2520	1906	1745	1389	—	—	—	—
3	Equal	2520	2520	—	2520	—	—	—	—
	Prop	2976	2440	—	2144	—	—	—	—
	DP	3152	2423	—	1985	—	—	—	—
2	Equal	3780	3780	—	—	—	—	—	—
	Prop	4154	3406	—	—	—	—	—	—
	DP	4242	3318	—	—	—	—	—	—

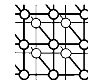
Table IV. Epoch time (ms).

Allocation method	Number of computers						
	2	3	4	5	6	7	8
Equal	2030	1662	1443	1420	1591	1820	1813
Proportional	1882	1607	1536	1552	1742	1955	1910
DP	1860	1542	1365	1354	1431	1603	1474

Table V. DP gain in execution time for one epoch.

	Number of computers						
	2	3	4	5	6	7	8
Gain (%)	1.16	4.04	6.16	3.87	10.05	11.92	18.69

Copyright © 2005 John Wiley & Sons, Ltd.



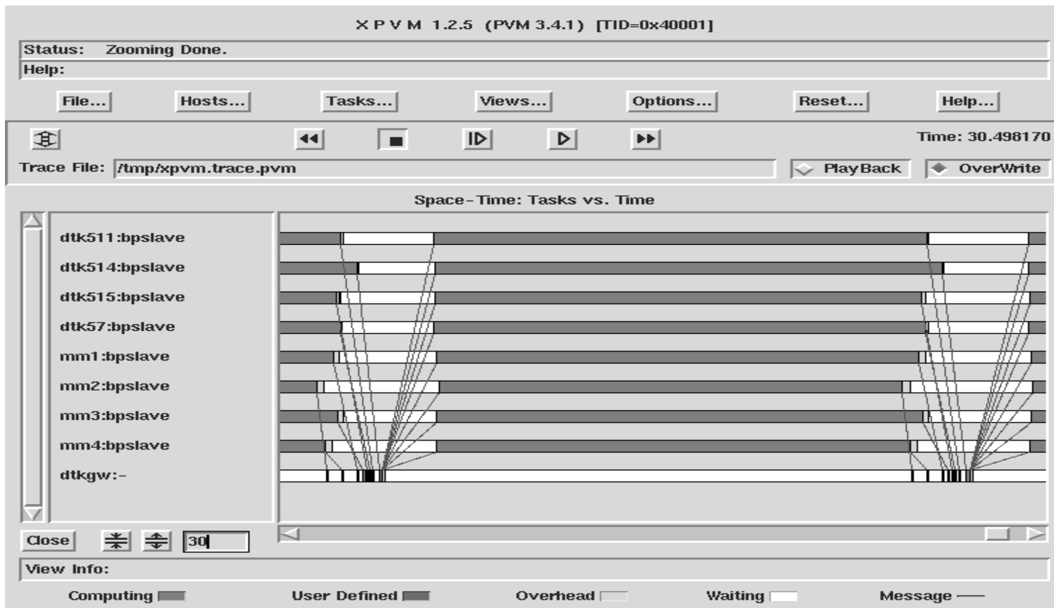


Figure 4. An execution (epoch) using the DP allocation on eight workstations.

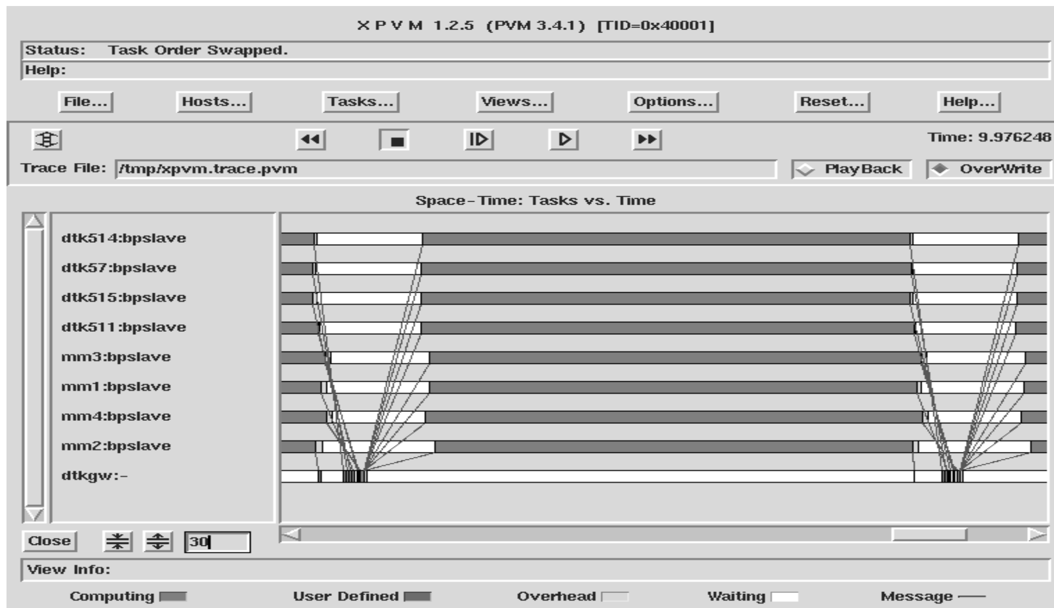
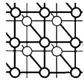


Figure 5. An execution (epoch) using the proportional allocation on eight workstations.

Copyright © 2005 John Wiley & Sons, Ltd.



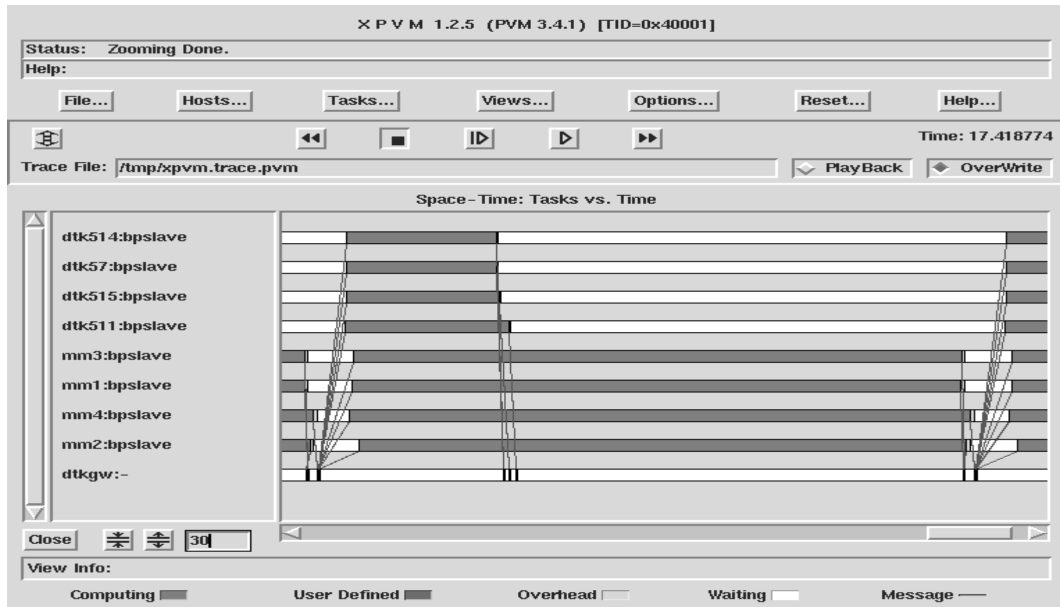


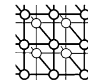
Figure 6. An execution (epoch) using the equal allocation on eight workstations.

In Table IV we show the epoch time considering the three allocation techniques. The epoch times in this table are obtained by running the NetTalk benchmark on the system described at the beginning of this section. T_n for one epoch using the DP allocation algorithm is considerably smaller than in the case of using equal and proportional allocations. The gain in performance is shown in Table V. DP is faster than the proportional scheme because DP overlaps communications and computations. This gain would increase as t_{comm} and the number of workers increases.

The maximum gains are obtained for up to eight computers. For example, consider a network of eight computers. Using DP allocation, the best distribution of patterns is found to be 1856, 1360, 1199, 911, 770, 629, 488, 347. The resulting epoch time is 1474 ms. Using equal distribution, i.e. 945 patterns per computer, the epoch time is 1813 ms. The gain in performance was about 19%.

In Figures 4–6, we show a run (on eight computers) using the allocation methods (1)–(3) above. Each process is represented by a horizontal bar having a different color depending on the state of the process. The dark gray segments represent the times when the tasks execute useful computation. The white segments represent the times spent waiting for messages from other tasks. The light gray (small) segments correspond to communication or task control routines. The thin lines linking task execution bars represent message exchanges. It can be observed that in the case of DP allocation the useful computation segments of each worker task appears to be well balanced. This is not the case for the equal allocation method where the slow worker computers have higher execution times compared

Copyright © 2005 John Wiley & Sons, Ltd.



with the faster ones. For the proportional allocation method the execution times are balanced, but the communication times increase and the overall performance degrades.

5. CONCLUSIONS AND FUTURE WORK

Our main contribution is the pattern-mapping optimization algorithm, which reduces the generally NP-complete problem to a polynomial-time dynamic programming solution. The optimization criterion is based on minimizing the execution time for one epoch (or T_n) of backpropagation learning. Using this allocation method we have obtained significant gains in performance compared with the proportional and equal allocation methods. Other backpropagation type algorithms with better convergence properties can directly be implemented using the mapping methods reported here.

Our allocation scheme can also be used on homogeneous systems. In this case, only the overlapping of computation and communication is exploited. In our paper, we analyze the performance in a dedicated environment, where the CPUs are not shared with other applications. If other applications would be active, our algorithm should be adjusted dynamically.

This pattern-mapping optimization could also be used for other parallel computation tasks. However, it is connected to the presented parallel architecture. The backpropagation algorithm was a very good match to this architecture.

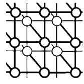
ACKNOWLEDGEMENTS

Some of the reviewers comments which helped enhance the quality of presentation of this article are graciously acknowledged. This research was supported, in part, by research grants from: (1) NASA NAG 2-1383 (1999–2001); (2) State of Texas Higher Education Coordinating Board through the Texas Advanced Research/Advanced Technology Program ATP 003658-0442-1999; (3) NSF Grant ASC-9634775; (4) NSF Grant CCR-0312323.

REFERENCES

1. El-Amawy A, Kulasinghe P. Algorithmic mapping of neural networks onto multiple bus systems. *IEEE Transactions on Parallel and Distributed Systems* 1997; **8**(1):130–136.
2. Chu LC, Wah BW. Optimal mapping of neural-network learning on message-passing multicomputers. *Journal of Parallel and Distributed Computing* 1992; **14**:319–339.
3. Kumar V, Shashi S, Amin MB. A scalable parallel formulation of the backpropagation algorithm for hypercubes and related architectures. *IEEE Transactions on Parallel and Distributed Systems* 1994; **5**:1073–1090.
4. Foo SK, Saratchandran P, Sundararajan N. Parallel implementation of backpropagation neural networks on a heterogeneous array of transputers. *IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics* 1997; **27**(2):118–126.
5. Crespo M, Piccoli F, Printista M, Gallard R. Parallel shaping of backpropagation neural networks in a workstations-based distributed system. *Proceedings of EIS'98 International ICSC Symposium on Engineering of Intelligent Systems*. ICSC Academic Press: Millet, AB, 1998; 709–715.
6. Paugam-Moisy H. Parallel neural computing based on network duplicating. *Parallel Algorithms for Digital Image Processing, Computer Vision, and Neural Networks*, Pitas I (ed.). Wiley: New York, 1993; 305–340.
7. Bharadwaj V, Ghose D, Mani V, Robertazzi TG. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press: Los Alamitos, CA, 1996.
8. Sohn J, Robertazzi TG, Luryi S. Optimizing computing costs using divisible load analysis. *IEEE Transactions on Parallel and Distributed Systems* 1998; **9**(3):225–234.
9. Nicol DM. Rectilinear partitioning of irregular data parallel computations. *Journal of Parallel and Distributed Computing* 1994; **23**:119–134.

Copyright © 2005 John Wiley & Sons, Ltd.



10. Andonie R, Chronopoulos AT, Grosu D, Galmeanu H. Distributed backpropagation neural networks on a PVM heterogeneous system. *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Systems (PDCS'98)*, October 1998, Pan Y, Akl SG, Li K (eds.). IASTED/ACTA Press: Anaheim, CA, 1998; 555–560.
11. Beaumont O, Legrand A, Robert Y. The master–slave paradigm with heterogeneous processors. *IEEE Transactions on Parallel and Distributed Systems* 2003; **14**:897–908.
12. Sejnowski TJ, Rosenberg CR. Parallel networks that learn to pronounce English text. *Complex Systems* 1987; **1**:145–168.
13. Zurada JM. *Artificial Neural Systems*. PWS Publishing Company: Boston, MA, 1992.
14. Ali H, El-Rewini H. On the intractability of task allocation in distributed systems. *Parallel Processing Letters* 1994; **4**:149–157.
15. Boffey B. *Distributed Computing-Associated Combinatorics Problems*. Blackwell Scientific: Oxford, 1992.
16. Geist A *et al*. *PVM: Parallel Virtual Machine. A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press: Cambridge, MA, 1994.

Copyright © 2005 John Wiley & Sons, Ltd.