Mapping Of Backpropagation Learning Onto Distributed Memory Multiprocessors

S. Mahapatra* and R. N. Mahapatra
Department of Electronics & Electrical Communication Engineering
Indian Institute of Technology, Kharagpur 721 302, India.
email : rabi@eece.iitkgp.ernet.in

ABSTRACT

This paper presents a mapping scheme for parallel pipelined execution of the Backpropagation Learning Algorithm on distributed memory multiprocessors (DMMs). The proposed implementation exhibits training set parallelism that involves batch updating. Simple algorithms have been presented, which allow the data transfer involved in both forward and backward executions phases of the backpropagation algorithm to be carried out with a small communication overhead. The effectiveness of our mapping has been illustrated, by estimating the speedup of a proposed implementation on an array of T-805 transputers.

Keywords : multilayer neural nets, distributed memory multiprocessors, training set parallelism, forward pass, backward pass, speedup efficiency.

1 Introduction

Parallel implementation of multilayer neural nets can be broadly divided into two distinct categories: implementation on a set of simple processors having small local memory and implementation on distributed memory multiprocessors (DMMs), which consist of more powerful processors with sufficient memory. In the first case, all the neurons and weights are distributed across the processors of the parallel architecture, and the synaptic operations of a single layer are carried out in parallel [6]. The parallelism achieved in this way is the finest level of parallelism that can be identified in the parallel implementation of a multilayer neural net and is known as the synaptic level parallelism [5]. Two distinct types of parallelisms can be visualized in the parallel implementation of multilayer neural nets on DMMs [7]-[11]. First is the processing of all the neurons belonging to a single layer in parallel, by distributing them across the processors [7]-[9]. This exhibits what is known as neuron level parallelism. The second one, known as training set parallelism involves batch updating [5]. In this, the weights are not modified after the presentation of each training pair in the learning phase of the backpropagation algorithm. Rather, weight changes produced by multiple training pairs are accumulated and then used to modify the network weights once for the presentation of a batch of training pairs.

Basically, there are two methods of achieving training set parallelism. In the first method, different training pairs can be processed in multiple copies of a single neural net, each copy being assigned to one processor [10] or each copy distributed across multiple processors [11]. The second method, that is preferable for implementation on array processors, is to process multiple training pairs in a pipelined way in the rows of the processor array, each executing a different layer of the neural net. The forward and backward passes of more than one layer may be simultaneously executed for multiple training pairs in the rows of the processor array.

In the present paper we have investigated the pipelined execution of multiple training pairs in the rows of a two-dimensional processor array. After discussing the assignment of neurons and weights on the processors of the array architecture, we have identified the data transfer required for parallel execution of different layers of the neural net in the rows of the processor array. Multiple data items heading for a common destination are sent in the form of one packet. This results in a reduction in the total communication overhead encountered in executing both the forward and backward passes of any layer, for two different training patterns. Algorithms are presented separately for forward and backward passes, and also for the pipelined execution of both the execution phases for different training pairs. The performance of the proposed mapping technique has been analyzed for a T-805 based transputer array.
Section 2, following the introduction, discusses the architecture of the multilayer perceptron and outlines its principle of operation. Also, the backpropagation learning algorithm that is used for training this network has been discussed in this section. The mapping method is presented in Section 3. Algorithms are given for execution of the backpropagation algorithm on an array architecture. In Section 4, we have analyzed the performance of a proposed implementation on an array of T-805 transputers. Finally, Section 5 summarizes this paper and comments on the efficacy of the mapping method.

2 Multilayer Perceptron and Backpropagation Algorithm

Primary application of the multi-layer perceptron (MLP) network is in pattern classification or more simply, pattern matching [1]-[4]. In this, the network produces the correct output pattern for any pattern presented at its input. The learning algorithm followed to train the network to perform the above task is the backpropagation algorithm devised by Rumelhart et al. in the year 1986 [4]. In this algorithm, a training set, consisting of a number of input-target pattern pairs is repeatedly presented to the network and the network weights are adjusted till the correct output pattern is obtained for every input pattern in the training set. Afterwards, the network continues to classify the input patterns correctly, irrespective of whether they belong to the training set or not. Below, we discuss the structure of the multilayer perceptron and the backpropagation algorithm in brief.

The multilayer perceptron consists of an input layer, an output layer and one or more hidden layers of neural elements or neurons. There is full connectivity in between two adjacent layers and the connections are weighted. Fig. 1 shows a 3 layer perceptron with 4 neurons in each layer. Function of the input layer neurons is only to distribute the input values to all the neurons of the first hidden layer. The backpropagation algorithm that is used to train this network consists of two phases: Recall phase or forward pass and the learning phase or backward pass. The forward pass starts with each neuron computing a weighted sum of all the inputs and then determining its output by applying a non-linear activation function to this sum. The output is then supplied to all the neurons of the next higher layer, which then proceed to compute their own outputs. This process is continued till the network output, composed of state values of all the output layer neurons are found out. Thereafter starts the learning phase or the backward pass, which begins with each neuron in the output layer finding out its error value by comparing the output with the target value. The error values of the output layer are then sent backward through the network calculating the error value for each lower layer neuron and adjusting its input weights. The process terminates after all the input weights of the first hidden layer have been adjusted. This completes execution of the backpropagation algorithm for a single training pair. After multiple presentations of the entire training set the weights finally converge to definite values, which signals the end of training phase. Now, the network can be used to classify arbitrary sets of input patterns.

Our assumptions regarding the multilayer perceptron and the notations used in the ensuing discussion are given below.

- The MLP network considered in this paper is assumed to have L+1 layers numbered from 0 to L, where layer l=0 is the input layer, layer l=L is the output layer and rest, 2 ≤ l ≤ (L - 2), are hidden layers.
- nl(l) and nq(l), respectively represent the ith and the jth neuron of layer l.
- αl and δl represent the state and error (delta) values of nl(l). Similarly, αq(l) and δq(l) represent the state and error values of nq(l).
- wjl(l) is the weight of the link from jth neuron of layer l-1 to the ith neuron of layer l.
- Netl(l) denotes the weighted sum for neuron nl(l).
- tj denotes the jth target pattern.
- T is the total number of training pairs in the training set.
- p and q represent the pth and the qth training pair, respectively.
- The data for the pth training pair are represented by placing a superscript p over the respective symbols. For example, the state value of nq(l) for the pth training pair is given by αq(p). The notation without any superscript carries a more general meaning, eg. αq(l) denotes the state value of nq(l) for any training pair.

Using the above notations the equations governing execution of the backpropagation algorithm are as follows:

Equation for computing the state values of layer 1 neurons from the state values of layer 1-1 is:

\[
Net_i(l) = \sum_{j=1}^{n} w_{ij}(l), a_j(l-1)
\]

\[
a_i(l) = f(Net_i(l))
\]

where $f$ is a non-linear activation function. Usually, the non-linear function used is the sigmoid function $f(x) = 1/(1 + e^{-x})$ \cite{1}. This function is preferred as it has got a very simple derivative given by:

$$d(f(x)) = f(x). (1 - f(x))$$

Equations 2 and 3 are followed in the learning phase. Equation 2, used to compute the error values of a lower layer neuron from error values of its higher layer is:

$$\delta_l(i - 1) = (t_l - a_l(i)).(1 - a_l(i)) \quad \text{for } l = L$$

$$= \left( \sum_{i=1}^{n} w_{lj}(l).a_j(l) \right)\cdot a_j(l - 1). (1 - a_j(l - 1))$$

$$\quad \text{for } 2 \leq l \leq L - 1 \quad (2)$$

Changes in input weights of layer 1 neurons are found by the equation:

$$\Delta w_{ji}(l) = \eta. \delta_i(l). a_j(l - 1) \quad (3)$$

In the above equation $\eta$ is the learning rate that usually lies between 0.25 to 0.75 \cite{1}.

3 The Mapping Method

We have proposed a mapping method for parallel pipelined execution of the backpropagation algorithm on a two-dimensional array of processors. Below, we discuss the target architecture, initial data assignment and the data routing involved in this implementation.

3.1 Target Architecture

The target architecture consists of a host computer and an QxP array of processors. The processors in this implementation are assumed to have unlimited memory. The Q rows of the two-dimensional array are numbered from 1 to Q, each row having P processors, numbered from 1 to P. The $i$-th processor in row $l$ ($1 \leq i \leq P; 1 \leq l \leq Q$) is denoted as $P_i(l)$. Each processor inside the array is connected to four of its nearest neighbours. The processor array is assumed to have horizontal wrap-around connections. Further, it is assumed that all the vertical links are bidirectional. A 2x4 processor array has been depicted in Fig. 2.

3.2 Initial Data Assignment

The initial data assignment, that gives the distribution of the neural net data on the target parallel architecture is quite simple. The input layer neurons do not perform any sort of processing and thus, they need not be mapped onto processors of the array. The distribution of the input values is assumed to be done by the host computer. It is necessary to map only the (L-1) hidden layers and the output layer. Thus, in the following mapping the neural net architecture has been considered to be an LxN array.

Initially, we assume that Q=1 and P=n, Hence, both the processor network and the neural net are considered to be Lxn arrays. We implement an one-to-one mapping of the neurons onto the processor array. Each layer of the neural net is mapped onto the corresponding row of the array, with neuron $n_i(l), 1 \leq i \leq n, 1 \leq l \leq L$, assigned to processor $P_i(l)$. Fig. 3 shows the data assignment for mapping of a 3 layer perceptron, having 4 neurons in each layer, onto the 2x4 processor array of Fig. 2. Processor $P_i(l)$ also stores the weights $w_{ij}(l)$ for $1 \leq j \leq n$. That is all the input weights of a neuron are stored with it. The T target patterns are stored in the Q-th (L-th) row of the array.

3.3 Inter-processor Communication

The inter-processor communication (IPC) in both the phases of the backpropagation algorithm involve a set of nearest neighbour shifts. First, the data transfer involved in the two phases has been discussed separately and then we present an algorithm to execute the two phases concurrently for multiple training pairs. The addition and subtraction operations considered in these algorithms are assumed to be modulo(P).

Forward Pass

The forward pass begins by the host computer communicating the input values to the corresponding neurons of the first hidden layer by sending $n_i(0)$ to $P_i(1)$, $1 \leq i \leq n$. The host computer is assumed to represent the 0-th layer of the processor array. Afterwards, Algorithm 1 is executed to determine the static values of layer 1 neurons in the l-th row of the processor array, $1 \leq l \leq L$.

Algorithm 1

1. In each processor $P_i(l), 1 \leq i \leq n, 1 \leq l \leq L$, initialize a variable $Net_i(l) = w_{ii}(l).a_i(l - 1)$.
2. Transfer the lower layer static value $a_i(l - 1)$ from $P_i(l - 1)$ to $P_i(l)$.
3. Send $a_i(l - 1)$ from $P_i(l)$ to $P_{i+1}(l)$.
4. In each of the next (P-1) steps do the following in processor $P_i(l)$:
   - Receive a lower layer state value $a_j(l-1), 1 \leq j \leq n$, from $P_{i-1}(l)$.
   - Update $Net_i(l) = Net_i(l) + w_{ij}(l).a_j(l-1)$.
   - Send $a_j(l-1)$ to $P_{i+1}(l)$.

Now, each processor $P_i(l), 1 \leq i \leq n$, will have the weighted sum:

$Net_i(l) = \sum_{j=1}^{n} w_{ij}(l).a_j(l-1)$. $P_i(l)$ then applies the sigmoid function to this sum in order to find the state value $a_i(l)$ (Eqn. 1). Total communication time for executing this algorithm is CP.

**Backward Pass**

The backward pass involves the calculation of lower layer error values, as well as the determination of changes in input weights of this layer. Algorithm-2 is followed to compute the error values of the $l$th layer neurons and to adjust their input weights.

**Algorithm-2**

1. In the processor $P_i(l+1), 1 \leq i \leq n, 1 \leq l \leq L$, initialize a variable (say an accumulator) $A_i(l) = w_{il}(l+1).a_i(l+1)$.

In each of the following steps:

2. Send $A_i(l)$ from $P_i(l+1)$ to $P_{i+1}(l+1)$.
3. In the processor $P_i(l+1)$, receive the data $A_i(l), 1 \leq j \leq n$, from $P_{i+1}(l+1)$ and do the following:
   - Update $A_i(l) = A_i(l) + w_{il}(l+1).a_i(l+1)$.
   - Send $A_i(l)$ to $P_{i+1}(l+1)$.

4. Execute step 3 (P-1) times. Afterwards, processor $P_{P_i(l+1), 1 \leq j \leq n}$ has the accumulator,

$A_i(l) = \sum_{j=1}^{n} w_{il}(l+1).a_i(l+1)$.

5. Transfer $A_i(l)$ from $P_{i}(l+1)$ to $P_{i}(l)$. Also, send $a_i(l-1)$ from $P_{i-1}(l)$ to $P_{i}(l)$.

6. Then, in (P-1) right shifts send $a_i(l-1)$ to all the processors of the $l$th row.

7. In processor $P_i(l)$, compute the error value $\delta_i(l)$ and the weight changes $\Delta w_{ij}(l), 1 \leq k \leq n$, using Eqs. 2 and 3.

The above algorithm needs a communication time of CP.

**Pipelined Execution**

The basic idea of the mapping is to execute different layers of the neural net concurrently in different rows of the processor array for multiple training pairs. Again, it is possible to execute both the forward and backward passes for two different training pairs in a single row of the processor array. Although it is not possible to parallelize the computations involved, as seen from algorithms 1 and 2, the data transfers in both the execution phases are similar and so can be combined as discussed in algorithm 3 given below. This algorithm computes error values of layer $l-1$ neurons from the error values of layer $l$, for the $p$-th training pair and also the state values of layer $l$ neurons from the state values of layer $l-1$, for training pair $q=p+2(l-1)$. Further, it helps in computing changes in the input weights of layer $l-1$ for the $p$-th training pair. Fig. 4 shows four snapshots of pipelined execution of the first four training pairs.

**Algorithm-3**

1. Transfer the data items $a_i^q(l-1)$ and $a_i^q(l-1), 1 \leq i \leq n$, in the form of a single packet from processor $P_i(l-1)$ to processor $P_i(l)$.
2. In the processor $P_i(l-1)$, initialize a variable $\Delta w_{il}(l-1) = 0, 1 \leq j \leq n$ and in processor $P_i(l)$, initialize two variables (accumulators):

$A_i^q(l-1) = w_{il}(l).A_i^q(l)$ and,

$A_i^q(l) = w_{il}(l).a_i^q(l-1)$.

3. In $P_i(l)$, combine $a_i^q(l-1).a_i^q(l-1)$ and $A_i^q(l-1)$ into a single packet and transfer the packet to $P_{i+1}(l)$.
4. In each of the next (P-1) steps do the following in processor $P_i(l)$:
   - Receive a packet comprising of the data items $a_i^q(l-1), a_i^q(l-1) and A_i^q(l-1), 1 \leq j \leq n$, from $P_{i-1}(l)$.
   - Update own accumulator $A_i^q(l) = A_i^q(l) + w_{il}(l).a_i^q(l-1)$.
   - Update the accumulator $A_i^q(l-1) = A_i^q(l-1) + w_{il}(l).A_i^q(l)$.

5. Update the accumulated weight change:

$\Delta w_{il}(l) = \Delta w_{il}(l) + \Delta A_i^q(l)$.

6. Recombine the three data items into one packet and send the packet to $P_{i+1}(l)$.

Now, processor $P_i(l), 1 \leq l \leq P$, has got the sums:

$A_i^p(l-1) = \sum_{j=1}^{n} w_{il}(l).A_i^p(l)$ and

$A_i^p(l) = \sum_{j=1}^{n} w_{il}(l).A_i^p(l-1)$.

Send $A_i^p(l-1)$ to $P_{i}(l-1)$ and in $P_i(l)$ compute $a_i^p(l)$ using $A_i^p(l)$ (Eqn. 1).

6. In processor $P_i(l-1)$ compute $A_i^p(l-1)$ using $A_i^p(l-1)$ (Eqn. 2).

Algorithm-3 takes a communication time of $CP+1$.

In the above discussions, it has been assumed that the number of processors in a row of the target architecture is same as the number of neurons in a layer of the neural net, which is rarely the case. Also, the neural net is assumed to have a regular architecture or the same number of neurons in each layer. The proposed mapping technique is equally valid when number of neurons in a layer of the neural net is greater than the number of processors in a row and the neural net architecture is not.
regular. It is enough that the number of processors be a multiple of the number of layers in the neural net, so that the layers are mapped onto different processors. When the number of processors per row of the array is less than the number of neurons in a layer of the neural net, multiple neurons of one layer would be processed in a single processor.

4 Analysis

We derive an expression for the time required to process a complete training set. Our assumptions in the following derivation are as follows.
- \( t_a \) and \( t_m \) are respectively the times for floating point addition and floating point multiplication.
- \( C \) is the communication time for transferring data between two adjacent processors.
- \( t_s \) is the time required to evaluate the sigmoid function.

Using the above notations, determination of state values of any layer takes a computation time \( t_f = n(t_a + t_m) + t_s \) and a communication time of \( CP \).

Time for computing the error values of each output layer neuron is:
\[ t_o = 2(t_a + t_m) \]

For each of the hidden layers, calculation of error values needs a computation time \( t_b \) given by:
\[ t_b = n(t_a + t_m) + (t_a + 2t_m), \] and a communication time of \( CP \).

Time to update the accumulated weight change is:
\[ t_w = n(t_a + 2t_m) \]

Theorem 1

The total time taken by an \( L \times P \) array to execute the backpropagation algorithm for the entire training set once is given by:
\[ T_m = (T + 1)(t_f + t_b + t_w) + 2(t_o + t_w) + (T + L + 1)PC + (T - L - 1)C \]

Proof

This expression follows directly from the principle of linear pipelining, as it takes \( L \) steps to get the first output pattern after presenting the corresponding input pattern, and later, one output pattern is obtained in every processing step. Each of the above processing steps takes a time of \( (t_f + PC) \). Hence the multiprocessor time is:
\[ T_{mp} = (T + L - 1)(t_f + PC) \]

The corresponding uniprocessor time is:
\[ T_{mp} = T_{u\text{unip}} = T_{u\text{unip}} \]

Finally, it may be observed that execution of backpropagation for the output layer has to be performed separately for the first, as well as the last training pair. For this, the additional time needed is \( 2(t_o + t_w) \).

Summing up, the total time taken by our implementation is:
\[ T_m = L(t_f + CP) + (T - L + 1)T_f + L(t_o + CP) + 2(t_o + t_w). \]

Substituting the value of \( T_f \) in the above equation and rearranging we have,
\[ T_{mp} = (T + 1)(t_f + t_b + t_w) + 2(t_o + t_w) + (T + L + 1)PC + (T - L + 1)C \]

Uniprocessor time to execute the backpropagation algorithm for the entire training set is given by:
\[ T_{unip} = nT(t_f + t_b + (L - 1)t_w + Lt_w) \]
And, speedup of the proposed implementation is:
\[ S = T_{unip}/T_{mp} \]
5 Summary and Conclusion

In this paper, we have proposed a mapping scheme for the parallel pipelined execution of the backpropagation learning algorithm on distributed memory multiprocessors. The layers of the neural net are assigned to the corresponding rows of the array and are executed simultaneously for different training pairs. Multiple data items heading for a common destination are assumed to be sent in the form of a single packet. This enables the communication in the forward and backward execution phases of two different training pairs to be combined, resulting in a reduced communication overhead. The weights are not modified after the processing of every individual training pair. Rather, changes due to multiple training pairs are accumulated and are used to modify the network weights once for the presentation of the entire training set.

Performance of the mapping scheme has been investigated by estimating the speedup of the implementation on an array of T-805 transputers. Each transputer has 4 communication links each of which can communicate with a speed of 20 Mbps. It uses a clock of 20 MHz, giving a cycle time of 50 ns. T-805 takes a time of 350 ns for adding two 32-bit numbers and the multiplication time is 550 ns [12]. As the communication speed is 20 Mbps, time for the transfer of a 32-bit number over the communication link comes out to be 1.6 ms. The sigmoid non-linear function used in this implementation is assumed to be computed using a Range Reduction Technique. The range reduction technique takes a total of 4 additions, 3 multiplications, 1 division, 1 logical and 1 shift operation to compute the exponential function. The sigmoid function needs an additional division and 1 subtraction operation [11]. For T-805 the division time is 850 ns and subtraction time is same as addition time of 350 ns. Assuming the logical and shift operation each to take a single clock cycle of 50 ns, time for computing the sigmoid function is estimated as 5.2 ms. With these timing parameters the speedup of the implementation has been estimated. Figs. 5 and 6 show the variations in S and S for different number of neurons per layer and Figs. 7 and 8 show the variations for different number of processors.

In each figure we have considered neural nets having 2, 4, and 8 layers (in these counts we do not include the input layer). It is clear that the mapping scheme works better for neural nets with more number of layers. The linear variation of speedup with increase in the number of processors proves the validity of our mapping technique.

References

FIG. 1 A 3 LAYER PERCEPTRON

FIG. 2 - A 2x4 Processor Array

FIG. 3 - Initial Data Assignment
Fig. 4 - Four snapshots of execution steps of the backpropagation algorithm for the first four training pairs.
Fig. 5 Training speedup versus n for 64 processors.

Fig. 6 Forward pass speedup versus n for 64 processors.
Fig. 7 Training speedup vs. processor no. (n=4096)

Fig. 8 Forward pass speedup vs. processor no. (n=4096)