Algorithmic Mapping of Neural Networks with Multi-Activation Product Units onto SIMD Machines

Yiwei Chen
Western Geophysical
3600 Briarpark
Houston, Texas 77042

Farokh Bastani
Department of Computer Science
University of Houston
Houston, Texas 77204-3475

Abstract
This paper presents a modification to the algorithmic mapping algorithm for neural network models proposed by Lin, Prasanna and Przytula. The modified algorithm can accommodate a larger class of network models recently proposed. The new neural network model uses vectorial interconnections between neurons and multi-activation product units. The generalized delta rule for the Rumelhart-Hinton-Williams neural networks can still be used with appropriate enhancement. The implementation of the new model is targeted for fine grain mesh-connected SIMD machines. The basic routing procedures are similar to those in the algorithmic mapping algorithm but with more flexibility in specifying the size of the data to be shifted between processors.

1 Introduction
The progress in VLSI and multi-processor computers makes it feasible to efficiently implement neural networks. Although most of the current realizations of neural networks are simulated on conventional serial computers, the need for high-speed computers arises from the beginning due to the significant speed limitations on implementing larger networks for complicated problems.

Programmable massively parallel computers have become more and more popular for implementing large neural networks because of their flexibility as general purpose machines. The mapping of the neural network models onto the parallel architecture is purely a software technique. The algorithmic mapping relies on a systematic approach for more general neural network models [KUNG88], [LIN 91]. Lin, et al, proposed an algorithmic mapping algorithm for a large set of neural network models onto the parallel SIMD machines [LIN 91]. It is based on a graph-theoretic approach. The algorithm systematically maps the neuron activation states and connection weights to an \( N \times N \) mesh of parallel processors, where \( N \) is square root of the total number of neurons and interconnections. When the data are fully pipelined, the data routing for a single iteration step of the recall phase requires \( 24(N - 1) \) elemental shift operations, and that of one learning iteration takes \( 60(N - 1) \) shifts. The fewer the interconnections, the more efficient is the implementation.

The neural networks covered by the original method are the traditional networks. Here by traditional we mean that they all use the conventional standard summing units, i.e., the activation or output of a unit is \( z_j = f_i(\sum \omega_{ij}z_i) \), where \( j \) denotes the \( j \)th neuron, \( f_i \) is the activation function for that neuron, \( z_j \) is the output of the \( j \)th neuron and \( \omega_{ij} \) is the connection weight from the \( i \)th input unit to the \( j \)th unit. This type of unit is dominant in various neural network models. However, some other neuron models, such as the sigma-pi unit [RUME86] are also of potential importance in real applications.

A new class of neural networks has been proposed recently [CHEN92], where the link from one neuron to another has multiple branches and the neuron has multiple dendrites. It is a generalization of the conventional neural networks with scalar links and standard summing neurons. A more specific type of unit called the multi-activation product (MAP) unit, computes the local activation at each dendrite first and then takes the product of all the local activations as the unit activation level. The Lin-Prasanna-Przytula algorithmic mapping method can be modified to accommodate the extended class of neural networks, if we assume that the local memory for each PE is sufficiently large. Instead of passing a scalar number among PEs, we now need to pass up to an array of the numbers.

The rest of this paper is organized as follows. Section 2 introduces the new neuron model. Section 3 develops the algorithmic mapping algorithm for the neural networks with new neurons. Final section summarizes the paper and outline some research areas.

2 Multi-Dendrite Neurons
In conventional multilayer feedforward neural networks, the inter-neuron connection is always assumed to be a scalar. These scalar connections are oversimplified analogies of the real biological neural con-
nections. On the other hand, the higher order connections introduced in [PAO 89] are not as easy to be implement because of their special topological structures. A directed neuron connection is called a vectorial connection if it branches in the exiting end and each branch is individually weighted. This connection analogies to the neuron axon which has multiple branches connected to the receiving neuron in the other end. Let the maximal number of dendrite for each neuron be $K$. Then the connection weight $w$ is an $K$-dimensional vector

$$w = (w^{(1)}, w^{(2)}, \ldots, w^{(K)})^T.$$  

The new neuron is called the multi-activation product unit or MAP unit as an abbreviation.

The generalised delta rule for the new network model is a further generalisation of the backpropagation learning for traditional feedforward neural networks. It is different from that in [RUME86] due to the obvious differences in the network architecture. Since the Rumelhart-Hinton-Williams neural network is just a special case of the new model, the classical learning rule should be a special case also for the newly developed algorithm. The similar approach as in [RUME86] is used to generate the new learning rule.

The terminology set includes indexed vector elements such as $net^{(k)}_j$ for the local net input of the $k$th dimensional of unit $j$ of a hidden or the output layer, and $w^{(k)}_{ji}$ for the $k$th element of the vectorial weight $w_{ji}$ from the $i$th unit to the $j$th unit. The change to the $k$th element of the vectorial weight upon the presentation of a training pattern $p$ is

$$\Delta w^{(k)}_{ji} = \eta \delta^{(k)}_{pj} o_{pi}.$$  

Then the modified generalised delta rule is

1. Assume $j$ is an output unit, then

$$\delta^{(k)}_{pj} = (t_{pj} - o_{pj}) f_j'(net^{(k)}_j) \prod_{i \neq k} f_i(nets')_i.$$  

2. If $j$ is not an output unit, the chain rule is applied differently, and the result is

$$\delta^{(k)}_{pj} = f_j'(net^{(k)}_j) \prod_{i \neq k} f_i(nets')_i \sum_m \sum_i \epsilon^{(m)}_i w_{jm}.$$  

where $t_{pj}$ is the target output for the $j$th unit of the output layer for pattern $p$, $o_{pj}$ is the actual output of the $j$th unit at a hidden or output layer produced by the presentation pattern $p$.

3 Algorithmic Mapping Algorithm

3.1 Target Machines

The Lin-Prasanna-Przytula mapping algorithm fits a large set of commercial and experimental parallel machines, such as the Connection Machine, MPP, Hughes 3-D Computer, and Systolic/Cellular Processor, etc. Any SIMD machine that meets the following assumptions can be a target machine:

1. The processors are at least mesh-connected.
2. The processors can perform all the computations required in the neural network learning and recall phases.
3. Each processor has a small local memory that is sufficient to store a vectorial weight, some activation values, error terms and the activation base function.
Two routing mechanisms are supported. One is to transfer data with tags. Address tags are determined a priori by the routing algorithm and stored in the processors. The another is to transfer data through elemental shift operation. Each processor would store mask bits computed a priori by the routing algorithm, which determine if the processor should be enabled or disabled.

Suppose the mesh-connected processor array is of size \( N \times N \), for \( N = (\sqrt{n} + c) \), where \( n \) is the total number of neurons and \( c \) is the number of interconnections. The feedforward network is assumed to have \( L \) layers. Each neuron in the net is a K-dendrite MAP unit. The connection is therefore K-D vectorial. The algorithm will map the network onto the SIMD machine. This mapping consists of two stages: loading of neuron activation states and activation functions, and the connection weights into processors, and the routing of neuron interconnections on the mesh network. The path between two processors representing two neurons is traversed in the sequence of elemental four-directional data shifts between neighboring processors.

![Figure 2: An example of the new neural network model and its weight matrix \( W \)](image)

3.2 Initial Data Assignment

Let \( W \) be a weight matrix, where the element at \((i, j)\) is a K-D vector which represents the weight for the vectorial connection from the jth neuron to the ith.

If there is no connection between two neurons, say from the ith to the jth, the respective element in \( W \) is an all-zero vector, which is termed the zero element of \( W \). The initial values of the neuron activation descriptions, \((x_1^0, f_1), (x_2^0, f_2), \ldots, (x_n^0, f_n)\), for initial activations \( x_1^0, \ldots, x_n^0 \) are assigned to the first \( n \) PEs in row major order. These PEs are assigned neuron PEs. The remaining PEs are called weight PEs because they contain non-zero \( W \) elements in consecutive columns, assigned also in row major order. In this way, a particular column in \( W \) can form a closely connected region, in order to facilitate the data transferring. Figure 2 shows a three layer feedforward neural network with two-dendrite neurons and the respective weight matrix \( W \).

3.3 Basic Routing Operations

The original Lin-Prasanna-Przytula algorithm makes use of basic routing operations such as SEND, REDISTRIBUTE, RESHUFFLE, COLLECT and SEND-BACK. These operations have to be redefined here for the new neural network model. The parameter \( s \) specifies the size of the data item during each shift in terms of the numbers.

**SEND\((s)\):** For any given network structure and its connection weights, the mapping of the non-zero elements of \( W \) onto the weight PEs in row major order is computed at compile time. A leader weight PE for a particular \( W \) column is defined as the PE which contains the least indexed non-zero element of that column. The SEND operation moves each neuron activation \( x_i \) to the leader weight PE of the respective \( W \) column. A three-phase permutation [LIN 91] can be used to realize the data shifting in \( 6(N - 1) \) steps.

**REDISTRIBUTE\((s)\):** This operation is used to copy the data in the leader weight PE to all the PEs containing the corresponding \( W \) column. This is done simultaneously for all the least indexed PEs. 3\((N - 1)\) routing steps of data shift of size \( s \) are enough to finish this operation.

**RESHUFFLE\((s)\):** This operation redistributes the data in the PEs from the row major in the PE array for each column of \( W \) into that for each row of \( W \). The new destinations for data can be calculated at compile time based on \( W \). Again, with a three-phase permutation, it can be realized in \( 6(N - 1) \) steps of data shifts.

**COLLECT\((s)\):** This operation is used to collect the sum of \( s \)-dimensional vectors contained in each PE for the ith row of \( W \) into the leader weight PE of that row, in parallel for all the rows. This can be accomplished in no more than \( 2(N - 1) \) steps of \( s \)-number data shifting since it is similar to the reverse operation of RESHUFFLE.

**SEND-BACK\((s)\):** This operation routes the summed vectors back to the respective neuron PEs. As in the case of the SEND operation, the total number of routing steps is \( 6(N - 1) \).

3.4 Algorithm for the Recall Phase

The revised algorithm for the recall phase is listed below, which is similar to the original algorithm. The main difference lies in 1) the new requirement for the shifting of a K-dimensional vector, and 2) only one leader weight PE is necessary for each \( W \) column or row.

1. Store the initial activations \( \{x_i^k\}_{i=1}^{n} \), where \( k = 0 \), and the respective activation base functions in the \( n \) neuron PEs in row major order.

2. Derive the mapping of non-zero vectorial weights in each column of \( W \) onto the remaining PEs in row major order and store them.
3. Identify the leader weight PEs for each column. The PE status for the example in Figure 2 after performing Step 1 to 3 is shown in Figure 3.

4. Derive the mapping of non-zero elements in each row of \( W \) onto the PE array in row major order in the remaining PEs from step 1.

5. Identify the leader weight PEs for each row. Figure 4 gives the newly derived weight distribution among weight PEs.

6. Compute the routing tags for all the basic routing operations defined in the above.

7. Each neuron PE, say the \( j \)th, simultaneously broadcasts its activation value \( z_f^j \) to the weight PEs containing the \( j \)th column of \( W \) using SEND(1) and REDISTRIBUTE(1) operations, as the example shown in Figure 5.

8. Compute the scaled vector \( z_i w_{ji} \) for each PE containing \( w_{ji} \). Call RESHUFFLE(1) to reorganise the scaled \( W \) elements from column to row ordering as derived in step 4.

9. Call COLLECT(\( K \)) and SEND-BACK(\( K \)) to form the summed vectors of the scaled ones, \( \sum z_i w_{ji}^{(0)} \) for \( i = 1, \ldots, K \), in the rows, route them back to the PEs storing the components of \( z^k \), and update \( z^k \) to \( z^{k+1} \). Increment \( k \) by 1. Update the activations afterward. Figure 6 shows the destinations where the data should be sent to in the previous example.

For the feedforward multi-layer neural networks, step 7 through 9 should be repeated for the number of layers in order to have the output for a given input pattern. However, the recall phase can be pipelined for a series of input patterns. That is the strength of this parallel algorithm.

### 3.5 Algorithm for the Learning Phase

The mapping and routing algorithm for the backpropagation learning is similar in principle to that of the recall phase but different from the Lin-Prasanna-Przytula algorithm. We will stay strictly on the layer-by-layer backward error propagation algorithm. It implies that the complexity of the algorithm depends
on the number of layers of the feedforward networks. Although our generalized back-propagation algorithm for the new neural network model is a little bit more sophisticated than the conventional one, we just need to perform extra $O(K)$ steps of multiplications on the neuron PEs containing the neuron activation values, and shift data of size $K$ for certain basic routing operations.

In addition to the activation states and connection weights, we also need to store the output layer error components, learning rate and the derivatives for activation functions at this phase.

1. Initialize the mapping as in steps 1 through 6 of the recall phase algorithm.

2. Store the learning rate and the derivative parameters of the activation base function in the same neuron PE containing the activation value $\sigma_j^t$ and its neuron activation base function.

3. For a given training pattern pair, store elements of the input pattern into neuron PEs representing neurons of the input layer. Store elements of the desired output pattern to neuron PEs representing the output layer.

4. Perform the recall steps 7 through 9 of the recall algorithm. But the activations broadcast in the recall step 7 will stay with the weights for later use in weight updates, as the $x_i$ to compute $\Delta w_j^{(k)}$. Also, for those PEs representing the output units, the difference between the $q_{pj}$ and $q_{pj}$ must be calculated after updating $\varepsilon$.

5. If the mapping of $W$ is in column major, call RESHUFFLE($K$) to get the ordering of $W$ in row major.

6. Broadcast $\eta_j^{(k)}$, as $\eta_j^{(k)} = \eta(t_p - \phi_p) f_j^{(net)}$, for the output unit $j$, or just $\delta_j^{(k)}$, as $\delta_j^{(k)} = f_j^{(net)} \prod_{l<k} f_l^{(net)} \sum_m \delta^{(m)} w_{jm}^{(m)}$, for the hidden unit $j$ though they are not available initially, using SEND(1) and REDISTRIBUTE(1) in rows of $W$. Then update the weight $w_{ji}^{(k)}, k = 1, \cdots, K$ at the PE containing it with change $\eta_j^{(k)} x_i$, for all the non-zero $W$ elements.

7. Call RESHUFFLE($K$) to reorganize $W$ elements from row to column ordering.

8. Use COLLECT($K$) and SEND-BACK($K$) to send $\sum_j \delta_j^{(k)} w_{ij}^{(k)}$ back to the PE representing unit $j$, for all $j$ in hidden where $\delta_j^{(k)}$, for $k = 1, \cdots, K$, are calculated upon receiving the sent-back data.

The last four steps should be repeated for about $L$ steps to carry out the complete iteration of the back-propagation learning for a given training pattern pair. However, this learning procedure can also be pipelined for a sequence of training pairs as some modified learning algorithm suggests [KUNG88], [VOGL88].

4 Conclusion

We have presented the modification of the algorithmic mapping algorithm for neural network models proposed by Lin, Prasanna and Przytula. The modified algorithm can accommodate the new network model with multi-dendrite neurons. The implementation of the new neural network model is targeted for fine grain mesh-connected SIMD machines. The basic routing procedures are similar to those in the algorithmic mapping algorithm but with more flexibility in specifying the size of the data to be shifted between processors. During the pipelined recall phase, the data routing for a single iteration takes $22(N - 1)$ elemental shift operations while during the pipelined learning phase, one backward iteration needs $28(N - 1)$ elemental shift operations.

The decrease in the number of shifting operations over the Lin's method during the learning phase is obtained by using 2 more extra local memory for each PE containing the weights, where they are used to store the broadcasted activations since they are needed twice.

References


