Chapter 2

A Review of Parallel Implementations of Backpropagation Neural Networks

Jim Torresen
Department of Computer Systems and Telematics
Norwegian University of Science and Technology, Norway
Email: jimtoer@idt.unit.no

Shinji Tomita
Department of Information Science,
Kyoto University, Japan

2.1 Introduction

In the previous chapter we reviewed the different ANN paradigms, their learning characteristics etc. Since feedforward neural networks with back-propagation learning is the most widely used configuration, we deal with this paradigm in this part of the book in more detail. Since the backpropagation learning for a bigger network with a large training set takes a long time (in terms of days), it becomes imperative to look at parallel implementation schemes to reduce this large training time. Before a detailed study of this can be undertaken, it is worthwhile to survey the different parallel implementation schemes for BP networks which exist in the literature indicating their strengths and weaknesses. This is the main theme of this chapter. Since the implementations depend very much on the type of parallelism used, the type of processor networks and also the topology, a brief description and definition of these terms are given in the beginning of the chapter. After this a survey of the different parallel implementations are described along with special purpose hardware built for these schemes. Finally, the chapter concludes with a descriptions of the different applications where parallel implementations have been successfully used.

2.2 Parallelization of Feed-Forward Neural Networks

Before we start discussing parallel implementation scheme, it is useful to define the basic terminology along with the different types of parallelism possible for BP neural networks. Also, the commonly used measures for evaluating the performance of the parallel implementations are described.

Training Set. Consists of a number of training patterns, each given by an input vector and the corresponding output vector.

Network size. A network of N_i input units, N_h hidden units, and N_o output units is for short written N_i x N_h x N_o. Note that the word *network* is used for neural network in this study and not for processor topology network. In the case of the latter use of the word it will be explicitly stated.

(Training) Iteration. Denotes *one* presentation of the whole training set.

Weight updating strategies. Three different approaches:

- *Learning by pattern (lbp)*, update the weights after *each* training pattern has been presented.
- *Learning by block (lbb)*, update the weights after a subset of the training patterns has been presented.
- *Learning by epoch (lbe)*, update the weights after all patterns have been presented (i.e. one training iteration).

Weight update interval. The number of training patterns that is presented between weight updates is termed μ. For *lbp*, $\mu = 1$, while for *lbe* $\mu = P$, where P is the number of training patterns in the training set.

The BP algorithm reveals four different kinds of parallelism, as described in [1,2].

- *Training session parallelism*, Starts training sessions with different initial training parameters on different processing elements.

- *Training set parallelism*, Splits the training set across the processing elements. Each element has a local copy of the complete weight matrix and accumulates weight change values for the given training patterns. The weights are updated using *lbb/lbe*.

- *Pipelining*, Pipelines the training patterns between the layers, i.e. compute hidden and output layer on *different* processors. While the output layer processor calculates output and error values for the present training pattern, the hidden layer processor processes the *next* training pattern. The forward and backward phase may also be parallelized in a pipeline. Pipelining requires a delayed weight update or *lbb/lbe*.

- *Node parallelism*, The neurons within a layer is computed in parallel (named neuron parallelism). Further, the computation within each neuron may also run in parallel (Nordström [1] names this *weight* or *synapse* parallelism). In this method, the weights can be updated using *lbp*.

2.2.1 Distributed Computing for Each Degree of BP Parallelism

This section first describes training set partitioning. Then the other main parallel degree, network partitioning, is detailed by describing pipelining and node parallelism.

2.2.1.1 Training Set Parallelism

Training set parallelism is also called data parallelism, since the training set is partitioned, not the training program. An example is given in Figure 2.1 for training of the English alphabet.

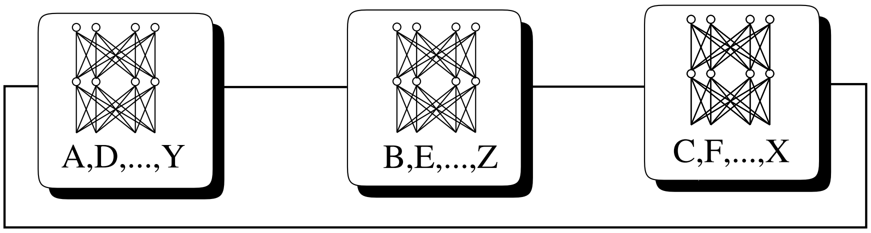

Figure 2.1: **Training set parallelism for learning the English alphabet.**

Each PE has a local copy of the complete weight matrices and accumulate weight change values for the given training patterns. The neural network weights must be consistent across all the PEs, thus weights are updated in a global operation (*learning by block/epoch*). The weight change values of each PE are summed and used to update the local weight matrices.

2.2.1.2 Pipelining

In pipelining, the different weight layers are computed in different PEs as illustrated in Figure 2.2. Figure 2.3 shows a pipelining example. First, the hidden layer processor computes output values of training pattern A. The output processor reads the values and computes output and error values of A. The hidden processor concurrently processes the next training pattern (B). Then, it reads the hidden error for A and both processors accumulate the weight change values for A. This method interleaves the forward phase with the backward phase. A further extension is to execute the two training phases in parallel [3].

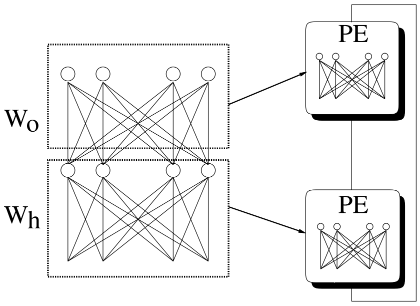

Figure 2.2: **Mapping of the weight matrices for pipelining.**

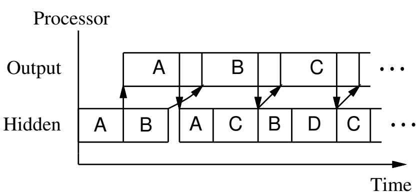

Figure 2.3: **Pipelining of the training patterns.**

2.2.1.3 Node Parallelism

Node parallelism contains two sub-degrees of network parallelism: neuron parallelism and synapse parallelism. The parallel mapping schemes, which use a mixture of these two, are named node parallel methods. Below, each of the two sub-degrees are explained.

Neuron Parallelism The most common way to parallelize feed-forward network is by using neuron parallelism, also called vertical slicing. Figure 2.4 indicates the principle for an example of three processing elements.

All incoming weights to one hidden and one output neuron are

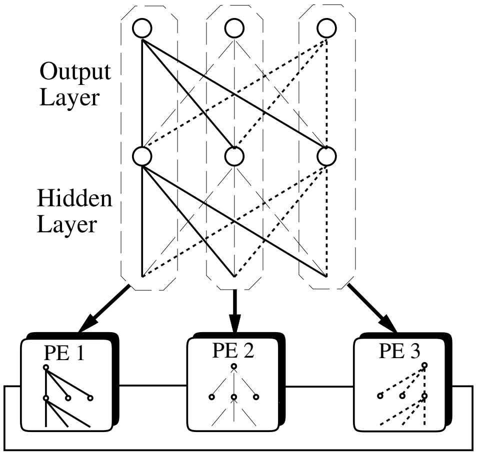

Figure 2.4: **Neuron parallelism, also called vertical slicing.**

mapped to each PE. That is, each PE stores all the incoming weights to the neuron assigned to that PE. The network slicing corresponds to storing one row of the weight matrix in each PE. The output of the neurons is computed by the matrix-vector product

$$\begin{bmatrix} y_{L,1} \\ \hline y_{L,2} \\ \hline y_{L,3} \end{bmatrix} = \begin{bmatrix} w_{L,11} & w_{L,12} & w_{L,13} \\ \hline w_{L,21} & w_{L,22} & w_{L,23} \\ \hline w_{L,31} & w_{L,32} & w_{L,33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{2.1}$$

where

$$L = \{\text{Layer} \mid \text{Layer} \in \{h, o\}\}$$

First, the value of one hidden neuron is computed in each PE. Then, each PE exchanges the values – e.g. over a ring bus, and continues by computing the value of the output neuron. The hidden layer error is a computed based on the output error. This can be formulated as a vector-matrix product

$$\left[\begin{array}{c|c|c} \delta'_{h,1} & \delta'_{h,2} & \delta'_{h,3} \end{array}\right] = \left[\begin{array}{c} \delta_{o,1} \\ \hline \delta_{o,2} \\ \hline \delta_{o,3} \end{array}\right] \left[\begin{array}{ccc} w_{o,11} & w_{o,12} & w_{o,13} \\ \hline w_{o,21} & w_{o,22} & w_{o,23} \\ \hline w_{o,31} & w_{o,32} & w_{o,33} \end{array}\right] \tag{2.2}$$

Since the weights are stored in row order, the sum-of-products is computed by adding partial products. However, more effective summation could be achieved if the weights were stored in column order. Thus, some implementations store the weights twice, both by row order – for forward pass, and column order – for backward pass. Either duplicated weight updates [4, 5] or communication of the weights [6] is by this method required. The former has been shown to be the most efficient [6].

If the number of neurons in a layer is larger than the number of PEs, each PE compute more than one neuron from each layer.

Synapse Parallelism Instead of mapping the rows of the weight matrices to each PE, the columns may be mapped. In synapse parallelism, each PE computes a partial sum of the neuron output as indicated in Figure 2.5.

The computation is more fine grained than for neuron parallelism. The sub-results from each PE have to be added and broadcast to all PEs before the next layer can be computed. The advantage is that the hidden layer error can be computed without communication

$$\left[\begin{array}{c|c|c} \delta_{h,1} & \delta_{h,2} & \delta_{h,3} \end{array}\right] = \left[\begin{array}{c} \delta_{o,1} \\ \delta_{o,2} \\ \delta_{o,3} \end{array}\right] \left[\begin{array}{c|c|c} w_{o,11} & w_{o,12} & w_{o,13} \\ w_{o,21} & w_{o,22} & w_{o,23} \\ w_{o,31} & w_{o,32} & w_{o,33} \end{array}\right] \tag{2.3}$$

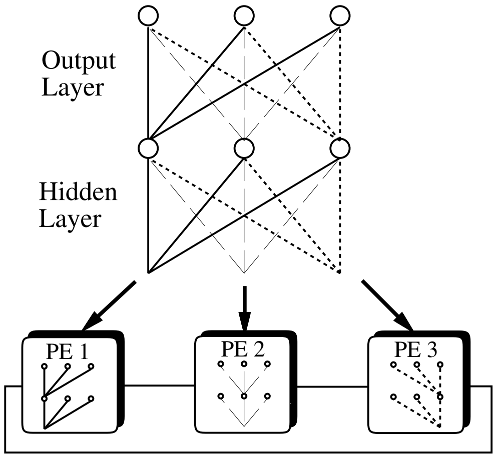

Figure 2.5: **Synapse parallelism.**

Thus, some implementations use neuron parallelism in the first layer and synapse parallelism in the second layer. The two degrees can be combined for both weight layers, as is described in Section 2.2.2.3 [7].

2.2.1.4 Dimension of Each Parallel Degree

Each parallel degree has an inherent limitation given by

Training set parallelism: The number of patterns in the training set

Pipelining: The number of weight layers

Neuron parallelism: The number of hidden units and output units

Synapse parallelism: The number of input units and hidden units

Thus, this indicate the maximum number of processing elements that can be assigned to each degree of parallelism. Before, we describe the different parallel implementations, it is worth while to understand the measures used to describe the learning performance of the parallel implementation of neural networks.

2.2.2 A Survey of Different Parallel Implementations

Many different mappings of BP onto parallel computers have been proposed and implemented – both for general purpose and special purpose computers. In this section, a description of the published work surveyed by the authors of this chapter is presented. As far as possible, both the architecture and the parallel implementation are described.

Mapping of neural network models onto parallel architectures may be categorized into two general groups: heuristic mapping and algorithmic mapping [8]. Most of the proposed mappings are of the heuristic kind. They are generated by trial and error based on knowledge about the algorithm and the target machine. The algorithmic technique relies on a systematic approach to the implementation. The work by Fujimoto et al. ([9, 10]), and Kumar et al. [11] belong to the algorithm mapping category. Moreover, some theoretical studies of parallel algorithms have been made, which are described in Section 2.2.2.3. The majority of other surveyed mappings assign PEs according to the number of elements in a single degree of parallelism e.g. the number of neurons. The same mapping is used for all kinds of neural applications. However, only a few networks and training sets will run optimally on a fixed mapping. What ought to be considered is what degree of parallelism should be included and how many PEs should be assigned for each of them to minimize the total training time.

2.2.2.1 General Purpose Computer Implementations

In this section, BP neural network simulations on various general purpose machines are described. A summary of the degrees of parallelism used and performance results are given in Section 2.2.2.9 and 2.2.2.10, respectively.

MasPar MP-1 MP-1 is a massively parallel SIMD computer from MasPar Computer Corporation. A variant of the mesh topology connects PEs to their 8 nearest neighbors – horizontally, vertically, and diagonally. In addition to nearest neighbor communication, there is a global router which allows any PE to send data to any other PE. However, the communication bandwidth is much larger for the nearest neighbor communication than for the global router. A fully configured system consists of a 128x128 array of PEs. Several node parallelism based implementations have been made for the machine. Chin et al. have made a node parallel implementation where the weights are ordered so that one (or more if available) column of PEs correspond to a single neuron [12]. For a 256 x 128 x 256 neural network, based on the results from smaller systems, 10 MCUPS is estimated for a 16K PE array [13].

Node parallelism and training set parallelism are combined in [14] for a 64x64 system. Each row of the array of PEs is assigned a copy of the network. Within each row neuron parallelism is used for computing the activation values of the hidden neurons. Then, synapse parallelism is used for computing the output activation values. A master processor adds the partial sums together and computes the output error. The error values for the hidden layer can then be computed by neuron parallelism. A network of 128 input neurons, 64 hidden neurons, and 16 output neurons achieved a maximum performance of approximately 12 MCUPS for 1536 training patterns. The weights were updated by *learning by epoch.*

A neural network simulator, Stuttgart Neural Network Simulator (SNNS), has been developed at the University of Stuttgart [15]. It is a portable software tool to generate, test and visualize ANNs. Many different learning algorithms have been implemented. A parallel SNNS kernel for multilayer feed-forward networks runs on MasPar MP-1216, which is the 16K PEs version of the MasPar computer. Two mappings are proposed based on a combination of training set and neuron parallelism. The first scheme assigns one PE to each vertical slice of the feed-forward network, which includes one neuron from each layer. The number of necessary PEs is equal to the number of neurons in the largest layer, except the input layer. Further, as many multiple copies as possible of the neural network are made to utilize all the PEs. The NETTALK network with 120 hidden units was trained in 41 MCUPS (down-loading time for training patterns from the host was not accounted for).

One of the biggest problems identified was the overhead for downloading the training patterns from the host to the parallel system. To avoid a single PE storing the remaining input values for an input layer greater than the other layers, the second implementation assigned the number of PEs equal to the largest number of neurons in *any* layer. A performance measure (including down-loading time for the training patterns) for NETTALK was 17.6 MCUPS. Fewer networks can be trained in parallel and more PEs become idle for networks with the largest number of neurons in the input layer.

A third prototype that combined training set parallelism and node parallelism gained slightly better peak performance than the second implementation. However, in this method, a variation in the number of neurons in each layer leads to dummy weight values in some of the PEs. The results from all three schemes demonstrate a performance dependence on the number of neurons in each layer. An unequal number of neurons in each of the layers reduces the performance.

IBM GF11 IBM GF11 is an experimental SIMD machine with 566 processors interconnected by a Benes Network[1]. BP has been implemented on this machine using training set parallelism [17]. The weight changes are summed by a binary tree processor configuration. To reduce the memory latency time, a *subset* of the weights are stored in SRAM. In this case, several patterns are processed on these weights before a new subset of weights is loaded into SRAM as opposed to finishing one training vector before a new one is input. NETTALK with 60 hidden units and a training set of 12022 patterns (epoch update) learned at 900 MCPS – no measure for CUPS given, on 356 processors. A ring summing algorithm achieved the highest performance for 128 processors of 222 MCPS, confirming the limitation of processors connected by a single ring.

[1] A multistage switching network that can perform all possible connections between inputs and outputs. Conflicts in the use of switches or communication links are solved by rearranging connections [16].

Hypercube Machines Simulation of BP trained neural networks on NCube/4$^+$ is reported by Kerckhoffs [5]. NCube/4$^+$ is a 4-dimensional hypercube machine with 16 processing elements. Larger systems of 128 and 1024 PEs are available. A PE can communicate with other PEs by use of asynchronous DMA. The implementation distributes the neurons in each layer between the PEs of the hypercube. A comparison was made between single weight storage and double weight storage for the output layer. For the double weight storage, the forward pass uses neuron parallelism – Equation 2.1, while the backward pass uses synapse parallelism – Equation 2.3. The same double weight storage scheme is employed for MUSIC – see Section 2.2.2.6. The drawback of this method is that the weights are stored and updated twice. Thus, the results show roughly equivalent communication time, but less computation time for the single weight storage scheme. The peak performance is 0.19 MCUPS. For synapse parallelism the broadcast time has been shown to increase linearly for a NCube size from 1 to 128 [18]. It is concluded in the paper that the effect of reducing the training time by this parallel implementation is minimal.

The hypercube computer Intel iPSC/860 is capable of being configured with up to 128 Intel i860 processors. A node parallel implementation of BP on a 32 PE version of the iPSC/860 is given in [19]. The network is partitioned vertically, i.e. all input neurons and at least one hidden unit is mapped to each PE. Each PE computes the output value of its assigned hidden neurons (neuron parallelism). Then, each PE computes a part of the output value based on the assigned hidden units (synapse parallelism). Thus, the only communication required in the forward pass is to sum the partial sums of the output units. For the NETTALK application with 80 hidden units, 11 MCUPS was reported. The weights were updated for every 2000 training vectors.

Connection Machines Singer gives a review of five different ANN implementations on the Connection Machine (CM-1 and CM-2[2]) [2]. The most sophisticated of those is developed by Zhang et al. [21] for the CM-2. This is a massively parallel SIMD computer with between 16K and 64K processors, each consisting of a one bit processing unit and 8KB or 32KB local memory. Moreover, every 32 processors share one floating point unit. The parallel BP algorithm combines neuron parallelism and training set parallelism. An implementation published by Rosenberg and Blelloch assigns one processor for each neuron in the network and two processors for each weight [3]. This is an inefficient processor assignment and they suggest an improved algorithm using one processor per weight and no separate processor for each neuron. Further, to avoid only processors with weights from one layer to be active at each time, pipelined computation – as described in Section 2.2.1.2, of the training patterns is suggested.

BP for CM-5 is presented in [22]. The CM-5 consists of 544 PEs, each a 33 MHz SPARC-2 chip with its own 32 Mbyte memory. The mapping is based on node parallelism, where the input and output neurons are evenly distributed over all processors. To use the *Control Network* for adding operation and avoid message passing communication, *all* the hidden neurons are mapped to each processor. Each processor computes a part of each hidden neuron sum in parallel – synapse parallelism. The partial results are summed and the sum is sent to *all* PEs. Then, each processor performs the sigmoid function.

[2]CM-2 is the same architecture as CM-1 with some added features like floating point accelerators and larger local memory. [20]

This is said to be the only redundant computation in the scheme. For the output layer, the PEs compute the outputs in parallel. No communication is needed since all hidden layer outputs are readily available in every processor. The implementation was tested on protein tertiary structure prediction with a network of 257 input neurons, 256 hidden neurons and 131,072 output neurons. For 512 processors a performance of 76 MCUPS was obtained. For a CM-5 with 32 PEs, the NETTALK network (80 hidden units) learned at 18.33 MCUPS using a batch implementation [23].

Kumar et al. [11] propose a hybrid scheme using node and training set parallelism for hypercube architectures. The node parallel scheme is called *checkerboarding.* This method makes all-to-all broadcast unnecessary, since only communication within rows or columns is needed. The processors are partitioned into clusters. A cluster forms a hypercube and contains one copy of the network. Each cluster contains different training patterns. The method makes it possible to vary the number of processors assigned to node parallelism against those assigned to training set parallelism. Theoretical expressions for the total training time were derived based on the cost of computation and communication. The computation cost was based on time per weight per pattern, while communication cost was based on startup time and per-word time for communication. The effect of changing weight update interval is not considered. The expressions are used to compare the model to simpler schemes. A non-optimized version of the hybrid method performed over 50 MCUPS for a 1024 x 256 x 64 neural network on a 256 processor CM-5.

Warp The Warp machine consists of a linear array of 10 PEs called cells. Each cell can communicate with its left and right neighbors in the array. In a much cited paper, Pomerleau et al. propose several BP mapping algorithms onto the Warp computer [24]. In the first implementation, columns of weights were stored in each cell – synapse parallelism, whereas in the second implementation, training set parallelism was used. As only a 32 Kword[3] memory is available in each cell – weights are in the latter implementation, pumped through the array from a central cluster memory. Nine cells compute the forward and backward pass, while the tenth is reserved for updating the weights. Each weight is used for all the training patterns in a cell, before the next weight is received. For the NETTALK network, the performance was measured to be 17 MCPS (CUPS measurement is not reported), which was the fastest performance reported at that time. The synapse parallel implementation was much slower.

Supercomputers Supercomputers are an alternative to parallel computers in providing high speed computation. Supercomputers consist of one or a few complex processing elements. They usually consist of multiple functional units, vector registers, and a highly pipelined processor. BP training has been implemented on the Fujitsu VP-2400/10 vectorial supercomputer [25]. The computer was programmed using an extended FORTRAN77 compiler. A high degree of vectorization is reported. The NETTALK network trained at 60 MCUPS.

Symult S2010 Symult S2010 is a computer with 16 processors. It is a mesh connected message-passing parallel computer with dedicated hardware for wormhole routing. An interesting BP mapping is

[3]Each word is 4 byte.

described in [26], which combines neuron parallelism and pipelining. The processors are connected by a ring bus as shown in Figure 2.6.

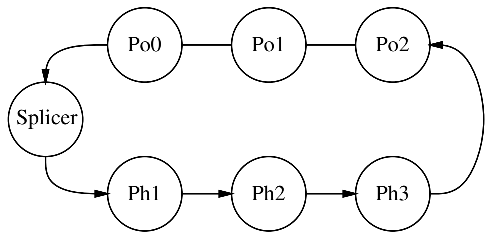

Figure 2.6: **Combined neuron parallelism and pipelining in Symult S2010 (from [26]).**

In this example the three lower processors are designated for the hidden layer, while the processors in the upper row are assigned to the output layer. The splicer processor composes messages of training vectors, one vector per message, which are sent to the first element of the pipeline – Ph1. Each message progresses through the pipeline and at each processor, activation values are computed and added to the received activation values which are sent on to the next processor on the ring. After a message has passed all Ph processors, each Po processor in the ring computes its output activation values based on the received hidden activation values. Moreover, the output processors compute the output error values which are also passed through the pipeline. The splicer includes the error values in the training vector messages to be used for computing weight change in each hidden processor. The weights are updated by *learning by epoch*. An expression for estimating the best number of processors for each layer is derived based on the number of neurons in each layer and the ratio between computing one hidden layer connection versus one output connection. The method has a reduced amount of communication. A disadvantage of this method is that the time for filling the pipeline is proportional to the number of processors. Moreover, to avoid using old weights after a weight update, the pipeline needs to be refilled after a weight update.

2.2.2.2 Interconnected Workstations

For mapping BP training onto workstations background workload has to be considered. A detailed study of both static and dynamic mapping algorithms is reported in [27]. In the dynamic case the neurons are re-mapped as the workload changes.

2.2.2.3 Research by Use of Models of Parallel Machines

In this section, schemes that have not been implemented on a real system are described.

Data-Driven Systems Data-driven computation is based on asynchronous parallel execution of computations, which may be represented by directed graphs. A processor instruction is executed based on the availability of its data operands. Q-x is a floating point data-driven processor. A message-passing multiprocessor is able to be configured of up to 1024 processors. The processors are interconnected by a 2D-torus communication network. Processing and storage in the processor is carried out as packets flowing through elastic pipelines. A description of the mapping of BP onto the architecture is reported in [28]. The scheme is based on combining neuron parallelism and training set parallelism, similar to [21]. That is, one row of PEs compute one copy of the network. The neurons are distributed among the processors in a row by vertical slicing of the network. Performance is evaluated by using a system simulator for training an image compression 256 x 128 x 256 network. For 64 processors 44 MCUPS was predicted.

Mesh Topology A fine grain BP algorithm using node parallelism for mesh-connected multiprocessors is proposed in [7]. It combines neuron and synapse parallelism, i.e. each cell computes a partial sum for a single neuron. The weight matrices are divided into rectangular blocks or sub-matrices that are distributed among the cells. The method is similar to the method — called checkerboard partitioning [29], described in the section on CM-5 implementations. The forward phase of the parallel BP algorithm can be illustrated by using a simple fully connected network of 4 input, 4 hidden, and 4 output neurons and a system of 4 cells. The weight matrices are partitioned into sub-matrices and multiplied by the input vectors $\mathbf{X}_L$ as follows:

$$\begin{aligned}
\left[\begin{array}{c} y_{L,1} \\ \hline y_{L,2} \\ \hline y_{L,3} \\ \hline y_{L,4} \end{array}\right] &= \left[\begin{array}{cc|cc} w_{L,11} & w_{L,12} & w_{L,13} & w_{L,14} \\ w_{L,21} & w_{L,22} & w_{L,23} & w_{L,24} \\ \hline w_{L,31} & w_{L,32} & w_{L,33} & w_{L,34} \\ w_{L,41} & w_{L,42} & w_{L,43} & w_{L,44} \end{array}\right] \left[\begin{array}{c} x_{L,1} \\ \hline x_{L,2} \\ \hline x_{L,3} \\ \hline x_{L,4} \end{array}\right] \\
&= \begin{bmatrix} \mathbf{W}_{L,11} & \mathbf{W}_{L,12} \\ \mathbf{W}_{L,21} & \mathbf{W}_{L,22} \end{bmatrix} \begin{bmatrix} \mathbf{X}_{L,1} \\ \mathbf{X}_{L,2} \end{bmatrix} \qquad (2.4)
\end{aligned}$$

where

$$L = \{\text{Layer} \mid \text{Layer} \in \{h, o\}\}$$

The mapping scheme can be effective if the network matches the architecture in a way that divides matrices and vectors into *equal* parts for each processing element. If not, some of the processors may become idle. Although communication is minimized in the method, the communication overhead will still be large for most real neural networks and parallel machines because the computation grains become too small.

A theoretical study of implementing pipelining of training patterns on a mesh topology is given in [30]. The method is based on assigning a dedicated mesh to each layer, see Figure 2.7. The hidden

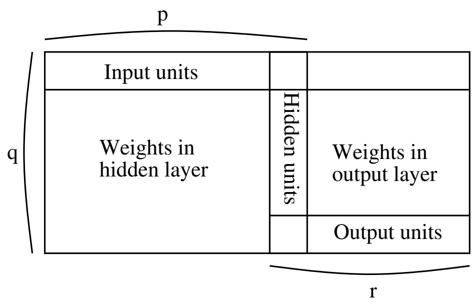

Figure 2.7: **Piped-mesh implementation (from [30]).**

layer is computed by $p \times q$ PEs and $q \times r$ PEs compute the output layer as indicated in the figure. The data for the hidden units are transmitted from the left to the right mesh. Theoretical expressions for transmission count and computation count are derived and used to estimate the optimal number of neurons per PE. For a 1024 x 128 x 128 network, 6 neurons per PE (u) is found to be optimal and therefore

$$u = \left\lceil \frac{N_i}{p} \right\rceil = \left\lceil \frac{N_h}{q} \right\rceil = \left\lceil \frac{N_o}{r} \right\rceil \tag{2.5}$$

where N_i, N_h and N_o are the number of neurons in each layer as defined in Section 1.3.1.

A framework for implementing neural networks on massively parallel machines is proposed by Azema-Barac [31]. The model formalizes a speedup for the different degrees of neural network parallelism on the given parallel machine. Derived expressions for one training iteration are made based on the grain of parallelism[4] and the communication overhead. It is not specified how to estimate or measure the communication overhead. Moreover, the accuracy of the method is not substantiated. Further, the effect of weight update interval on the convergence is not considered. The distribution scheme is designed for node parallelism, training set parallelism, and a combination of these two.

The target machine for implementing the scheme has been DAP 600 (Distributed Array Processor). This is a SIMD machine with 64x64 PEs, each interconnected to its four nearest neighbors. One of the the few results given is based on a digit recognition application [32]. A speedup of 50 over a Sparc station is reported.

2D-Lattice Topology A proposed reconfigurable machine, called Dynamically Reconfigurable Extended Array Multiprocessor (DREAM) is presented in [33] for implementation of neural networks. It is classified as a SIMD machine and the PEs are arranged on a 2D lattice, i.e. each PE is connected to eight of its closest neighbors through programmable switches. Multiple rings of various lengths can be mapped onto the machine. An *embedded* ring structure is proposed for the case of uneven neuron layers in a multilayer network. For a network of N_l and N_{l-1} neurons in layer l and $l-1$, respectively, the ring size is set to $\max\{N_l, N_{l-1}\}$. If $N_{l-1} > N_l$ the shorter ring of length N_l is embedded into the ring of length N_{l-1} as shown in Figure 2.8. The shorter ring of size N_2 is embedded into the ring of size N_1. The

[4] The number of neurons assigned to each processor.

weight matrices are partitioned vertically – synapse parallelism, and the output of a neuron is computed by accumulating partial sums from each PE. First, N_{l-1} cycles are used for computing the layer $l-1$ outputs. Secondly, a shortened version of the ring, equal to the N_l, is used for the next layer. Thus, the ring size is dynamically changed as the computation moves from layer to layer. If the number of neurons is larger than the number of PEs, time-multiplexing is used for creating virtual PEs.

The results presented in the paper are from analytical estimation and not real implementations. Chapter 9 thoroughly describes the DREAM machine architecture.

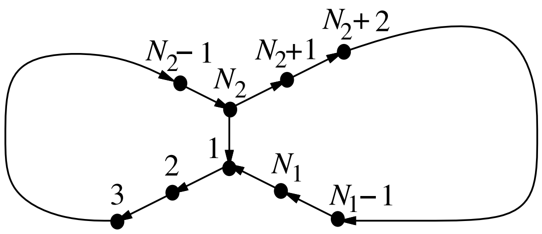

Figure 2.8: **A ring of size N_2 embedded into a longer ring of size N_1 (from [33]).**

Hypercube Topology A theoretical study of the mapping of a tree structure onto a hypercube topology is proposed in [34]. The trees are used for communication of elements. One tree is used for broadcasting a value from a root to all the leaves and another tree is used to sum values from the leaves into a root. $\mathcal{O}(\log N)$ time is required, where N is the size of the largest layer. The scheme is based on node parallelism, similar to the one described in section on Mesh Topology above [7]. Pipelining of the patterns is also suggested included.

Systolic Array Design A programmable systolic array is proposed in [35] for a wide variety of artificial neural networks. VLSI is the target architecture for the mapping. A data *dependency graph* (DG) is used to express the recurrence and parallelism associated with the neural network algorithm. The graph is mapped onto a proposed ring systolic array – one ring for each weight layer. The weights for one output neuron, i.e. one row of the weight matrix, are mapped to one systolic element, see Figure 2.9. Thus, both the learning and recall phase share the same storage and processing hardware. Input elements are rotated across each ring array.

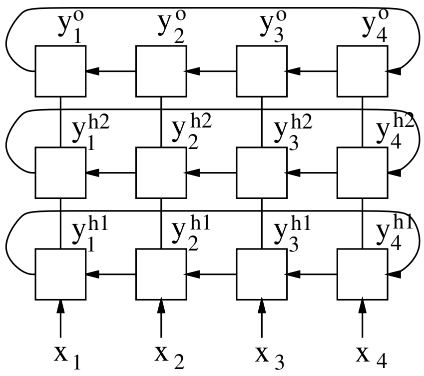

Figure 2.9: **Cascaded systolic ANN (from [35]).**

The proposed systolic design is efficient when the sizes of the different layers are approximately equal. To make the design applicable for multi layer networks of different sizes, the number of systolic elements assigned to each layer may be varied. Each weight layer is mapped to a number of systolic elements equal to the number of rows or columns, depending of which is smallest in number. A two layer neural network with configuration (3-9-4) is used in the paper, and the design is shown in Figure 2.10. The hidden weight matrix is vertically partitioned, and one column of weights is stored in each systolic element of the lower PE row. The output weight matrix is partitioned horizontally and stored in the upper PE row. This approach minimizes the number of PEs needed.

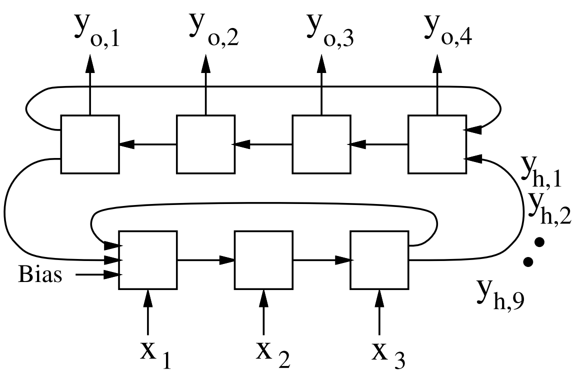

Figure 2.10: **Systolic design for feed-forward networks with layers of different sizes (from [35]).**

First, the forward pass starts by residing the inputs $\{x_i, i = 1, 2, 3\}$ at the corresponding PEs in the first layer. The bias values are pipelined one-by-one into the array from the leftmost PE. Each bias input initiates an accumulation operation where the product $w_{h,ji}x_i$ is added at the i-th PE. The complete hidden neuron element $y_{h,j}$ is output at the rightmost PE after 3 accumulation operations. The 9 hidden neuron elements are sequentially pumped into the upper array of PEs and used for computing the output layer elements $y_{o,k}$. After the nine elements $y_{h,j}$ have passed the four upper PEs, the output $y_{o,k}$ is ready at the k-th PE. In a similar way, the backward phase is computed.

A more general mathematical formulation, covering mapping of a wide range of ANNs to ring systolic arrays, is presented by the same authors in [36, 37]. However, a single ring is used and one layer is computed at a time. The design assumes that all the neuron layers are of uniform size. Further, a systolic mapping where each PE stores a row of the hidden weight matrix and a column of the output weight matrix is presented in [38, 39]. The PEs are connected by a ring bus. Each input element is sequentially input to the upper PEs. The method is extended by using *learning by epoch*, which allows pipelining of the training patterns in rectangular arrays of PEs.

SIMD Mesh-connected Machine Implementations A fine grain node parallel mapping is proposed in [8, 40]. Although claims are made that the method is applicable to various SIMD machines, the details are given for the Systolic/Cellular Array Processor (SCAP). However, no implementation results are reported. The scheme is based on a graph-theoretic approach, and there is no requirement for fully interconnected layers. If the the total number of neurons and weights in the network is less than or equal to the number of available processors, then each neuron and each weight is mapped to a separate processor. The input elements are then sent to the processors storing the weights to be multiplied. A transformation scheme is given for transforming the weight matrix between row and column major order. In the opposite case, where the number of available processors is less than the total number of neurons and weights, the data is stored in a global memory. The size of the local memory is very limited. Thus, the processor array is operating on sub-arrays from the global memory. The essential data movement operations are realized by embedding a *Benes* network.

2.2.2.4 FPGA Implementations

Field Programmable Gate Arrays (FPGAs) have recently become an alternative to full custom VLSI design. FPGA is reasonably priced and makes the development cycle shorter, in-comparison to VLSI. However, the flexibility leads to less performance and functionality per unit area of silicon. To fully utilize the flexibility of the FPGA technology it should be used with run-time reconfiguration. This is exploited in the RRANN (Run-time Reconfiguration ANN) system [41]. That is, the BP algorithm is divided into sequentially executed stages: At run-time only one stage is configured on the FPGA at a time. Since only one stage is used at any given time, more hardware resources are available for each stage. In the system, 6 neurons are mapped to each FPGA (XC3090). Results show that to achieve a higher performance, than that achieved by a non-run-time reconfigurable system, more than 22 FPGAs need be used. This is due to the re-configuration overhead.

A system for feed-forward recall-phase is presented in [42]. Each hidden and output neuron is implemented in 2 FPGAs (XC3042) connected to an EPROM. The EPROM contains a lookup table for the sigmoid activation function. A network with 5 input units, 4 hidden units and 2 output units executes at 4MCPS. They conclude that the size and speed of the system can be greatly improved by using higher density FPGAs. An experimental project called REMAP consists of building an entire computer using only FPGA and memory [43]. It is based on bit-serial processing elements with SIMD control. The PEs are designed for neural computation. A full-scale prototype (256PEs) can run in 10 MHz.

Since one of the main bottlenecks in implementing ANNs in FP-

GAs is the multiplication operation, the circuitry devoted to multiplication must be significantly reduced, according to Bade and Hutchings [44]. This can be accomplished in two ways. First, by reducing the number of multipliers and/or secondly, by reducing the complexity of the multiplier. Bade and Hutchings propose to use bit-serial stochastic computing to reduce the circuitry necessary for multiplication. That is, to represent signal values as long bit-streams according to the statistical distribution of "1"s in the bit-stream. Then, multiplication can be implemented as a bit-wise Boolean ("and") operation, which is implemented as a lookup-table. Since FPGAs are normally based on lookup-tables, this scheme is very efficient. The system can process 181K patterns per second (recall phase). This is regardless of the network size. The drawback of the method is that learning is difficult due to lack of precision.

2.2.2.5 Transputer Implementations

A transputers chip contains a microprocessor, memory, and ports for communication. The most commonly used transputer is the INMOS T800.

An analysis of a transputer system shows that for a fully or randomly connected neural network, the topology of the processor network only has a small, constant effect on the iteration time [45]. An expression for the optimal number of processors is derived. Communication overhead is shown to be a major limiting factor in the efficiency of implementation on transputers.

Yoon and Nang [46] have shown that hypercube networks have less communication time complexity than mesh networks for backpropagation training. Thus, as the number of processing elements involved increases, the performance of hypercube mapping outpace the mesh mapping. The model was verified by implementations on a transputer system.

In [47], a neural specification language is defined. The purpose is to parallelize neural networks automatically based on a high level language. The parallelizing scheme only considers neuron parallelism. The approach scales acceptably for a system of 13 transputers, when the network consists of a large number of neurons.

Transputers Interconnected by Mesh or 2D-Torus Network

Fujimoto et al. propose a load balanced mapping of the BP and Hopfield neural networks onto both a Toroidal Lattice Architecture (TLA) [10] and a Planar Lattice Architecture (PLA) [9]. A TLA is a one-way directed 2D-torus network, while a PLA is a bidirectional mesh network. Both schemes are based on mapping a single weight to a virtual processor. Thus, the method uses node parallelism, i.e. both neuron and synapse parallelism. The matrix of virtual processors is partitioned into sub-matrices, each of which is mapped to a real transputer processor. Before the partitioning, the rows and columns in the virtual matrix are permuted so that each sub-matrix contain an equal amount of computation (multiply-accumulate and sigmoid computation).

A feed-forward network of 256 x 64 x 256 neurons is used in the work. The performance for a 16 transputer system interconnected by a PLA is estimated to be slightly higher than that achieved on a TLA interconnected system. An implementation on TLA learned at 0.6 MCUPS. The conversion rate from MFLOPS to MCUPS is roughly estimated to 1/40 in the worst case. The proposed schemes are proved to exhibit almost linear performance with respect to the number of processors when expanded for large scale neural networks. The processor topologies are planned implemented in wafer scale integration (WSI) technology. See Chapter 8 in this book for further details.

The research reported in [48] considers each layer of the network as a stage of a pipeline. Theoretical expressions for the performance of an implementation of pipelining and node parallelism on a 2D-torus transputer system is given. The weight matrices are divided into QxQ blocks, where Q represent the number of PEs in a row or column. The matrix-vector product is obtained by broadcasting the input elements along columns and accumulating sub-results in the rows. This is similar to the *checkerboard* partitioning presented earlier. However, in this work, little is said about the mapping of pipelining of the computation in the layers onto the computer.

Results from a 16 transputer system show that for a three neuron layer network, the time usage for the pipelined and the non-pipelined implementation was almost identical. Preliminary experiments show that the pipelining algorithm – using delayed weight update, seems to converge in the same way as the non-pipelined algorithm – updated after each training pattern. It is concluded that pipelining increases the computation load versus communication per processor and enables the computation to be more evenly distributed throughout the network. Thus, pipelining is preferable since the overhead of its operation is negligibly small.

Transputers Connected by a Ring A thorough study of training set parallelism on a ring of 32 T800 transputers is undertaken by Paugam-Moisy [49]. Each transputer is assigned an equal number of training patterns, and different block sizes are tested for *learning by block*. For a classification application – described in Section 1.3.1.3, the total training time is computed for different weight update intervals – see Section 7.3.2. A block size in the range [192, 384] is found to be optimal. A model is also made for estimating the time for one training iteration and estimation of number of iterations needed for convergence is undertaken – see Section 7.3.2.3.

A transputer ring topology is presented in [50]. The idea is to combine training set parallelism and pipelining. A three layer network is mapped onto 15 PEs as shown in Figure 2.11. Each shaded box represents one network cluster. One PE is assigned for each layer within a cluster. No performance results are given.

The MEIKO Computing Surface is a reconfigurable parallel computer based on the T800 INMOS transputer. A mapping scheme of BP onto this system is given in [51]. The mapping scheme combines node and training set parallelism. The method implies that a subset of the rows of the hidden weight matrix and subset of the columns of the output weight matrix are stored in each processor. In this way, no communication between processors is needed until the partial results from each PE is to be summed to compute the activation of the output layer. This is done by a *master* processor. The number of processors that can be efficiently used, by using only node parallelism and *learning by pattern*, is 20. For a larger number of processors, the performance is only marginally increased. A performance of 8 MCUPS is reported for a 512 x 256 x 64 network, when training set parallelism is included and the PEs are connected by a 3x40 mesh.

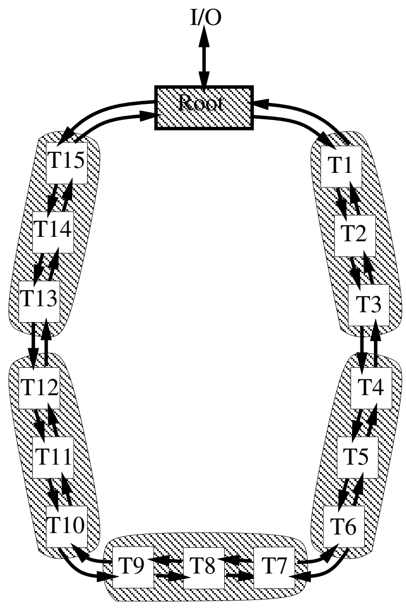

Figure 2.11: **A transputer ring topology combining training set parallelism and pipelining.**

A similar mapping is used on the MP-1 [14].

Transputers connected in a pipelined ring topology are used for training set parallelism by Foo in [52, 53]. An administrative transputer distributes the training set evenly amongst the other processors in the pipe. For weight update, each pipe processor receives weight changes and errors from its upstream neighbor, adds its own weight changes and errors to these and passes the new values to the downstream neighbor. Thus, the last downstream processor will hold the overall weight change and the total error. After computing the new weight values it propagates them back through the pipe. Simulations show that for small networks the serial implementation out-performs the parallel implementations. An analytic expression for the training time per iteration is derived. However, the scheme does not consider the effect of the weight update frequency on the convergence. The weight change summing seems to be leaving processors idle, even though an improvement is proposed. In that case, the downstream transputers are assigned a larger number of training patterns [54]. Using this latter scheme, 1.54 MCUPS is reported for NETTALK using 53 T805-20 transputers.

Transputer Tree Several implementations on 9 transputers are reported in [55]. Both fully connected network and the extension of shared weights–receptive fields (SW–RF) network are used. The latter network is beneficial for shift-invariant feature extraction and improves the generalization properties.

The best performance for the SW–RF network (almost linear) is seen when the transputers are connected in a tree and use training set parallelism. However, the problem of slow convergence due to *learning by epoch* is mentioned. Another implemented scheme combines node parallelism and pipelining. The parallel processing speedup approaches a value of six for the largest input image size. For a fully connected network the speedup is higher, which is explained by an increased utilization of the allocated processors. However, the SW–RF neural network showed better generalization properties.

Transputer Hypercubes Training set parallelism for hypercubes made up of transputers is studied in [56]. An $\mathcal{O}(\log n)$ step algorithm for summing the weight change matrices is proposed. To be able to use all the communication channels in the cube concurrently, the weight matrix is split into $\log n$ parts, where n is the number of processors. Each part is summed along one axis at a time. The redundant summation in every two PEs is eliminated by doing one half of the summation in each PE and exchanging the results. For a small training set, the described optimizing methods improves the speedup. However, for a training set of 512 samples or more, the difference in speedup is marginal.

The study is restricted to hypercubes of dimension 1, 2, and 3. Moreover, the weight update interval is not considered. The application used was speech recognition.

A training set parallel scheme is reported in [57] for training a Time Delay Neural Network (TDNN) for speech recognition. The convergence in the number of iterations varied for different weight update frequency. However, with no favor for frequent weight updates. 16 transputers showed a speedup factor of 8.8 over a VAX 3600.

Summary of Transputer Techniques Much published work has been undertaken in this area. Communication seems to be a bottleneck in large systems. Some of the implementations depend on using a master processor, which limits the scalability. In many of the projects, training set parallelism is employed, see Table 2.3. However, with a few exceptions [49, 57], experiments on the effect of the weight update frequency on the total convergence time has not been reported.

2.2.2.6 Digital Signal Processor (DSP) Based Systems

MUSIC (MUlti Signal processor system with Intelligent Communication) is a DSP based parallel system with distributed memory. A system of 45 PEs with a peak performance of 2.7 GFLOPS [6] is implemented. Each PE consists of a DSP (Motorola 96002), up to 8Mbyte of Video RAM (VRAM), up to 1 Mbyte of SRAM and an FPGA communication controller. Each board consists of 3 PEs controlled by a transputer, which also interfaces the PEs to the host computer. Inter-PE communication is by a pipelined ring. A dynamic load balancing scheme is implemented, where the operating system distributes data according to each PE's computation in the previous step. An implementation of BP based on standard neuron parallelism is used – where one network layer is computed after the other. However, to be able to compute the hidden delta error without using partial sums, each PE stores a local copy of the weights needed to compute the error. For a network of two weight layers, this only applies to the output weight layer. This double weight storage was also used in the NCube/4+ mapping described earlier. The machine it outlined in more detail in Chapter 10.

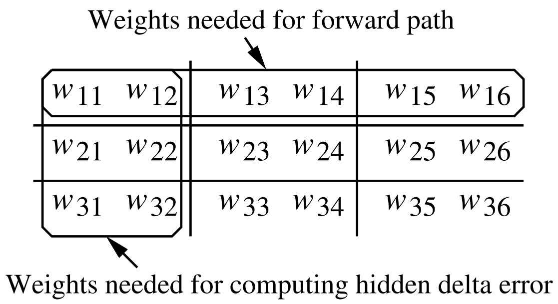

Figure 2.12: **The output weight matrix, where the weights required by processing element 1 are circled.**

Figure 2.12 shows the sub-matrices of the output weights stored in PE 1 for a network of 3 output neurons. The horizontal slice is used to compute the forward pass, while the vertical slice is used to compute the delta error for the hidden layer. To avoid communicating the weights, each processor updates both weight subsets individually. Thus, each weight will be updated twice. Muller el al. claim that this implementation is faster than computing partial sums of the hidden layer delta error. A performance of 203 MCUPS is reported. In [58], Solheim has compared several neuron based mapping schemes and found that storing the weights twice is not beneficial, except for small networks.

RAP (Ring Array Processor) is a neurocomputer designed for continuous speech recognition at the International Computer Science Institute, Berkeley, California [59]. The machine is a DSP based system with a low-latency ring interconnecting the PEs. Each PE consists of a TMS 320C30 DSP, 4–16 Mbyte DRAM and 256 Kbyte of local memory. FPGAs were used for the interprocessor ring and the memory controllers. Four interconnected PEs are contained on each board. To speed up the computation, a library of matrix-oriented assembly language routines were written. A working system of 40 PEs was used in the experiments. For a network with 128 neurons in each layer, the learning speed for vertically sliced feed-forward network was 57.3 MCUPS. Recall speed was 211.1 MCPS.

Based on the experiences of designing RAP, the CNS-1 (Connectionist Network Supercomputer-1) was designed [60]. It is proposed to supply orders of magnitude more computing capability than RAP. Custom designed VLSI digital processing nodes for neural network calculations are connected in a mesh topology and operate independently in a MIMD style. This is different from RAP, which uses standard, commercially available components. The initial system will be built with 128 processing nodes and a total of 4 GB distributed storage. The processor consists of a scalar CPU, a vector coprocessor running at 125 MHz, and hardware for communication, which are all comprised in a single chip. A PE consists of the processor chip and DRAM memory, minimizing the board area required per processor. The goal is that the minimum configuration should perform recall at 100 GCPS and learning should in the worst case be 1/5 of the recall speed.

Sandy is a ring-based neurocomputer developed by Fujitsu [61,62]. The system is based on TMS320C30, and the PEs are connected by a ring bus. The mapping of BP training is by neuron parallelism, i.e. vertical slicing. A prototype has been built, running NETTALK learning at 42 MCUPS.

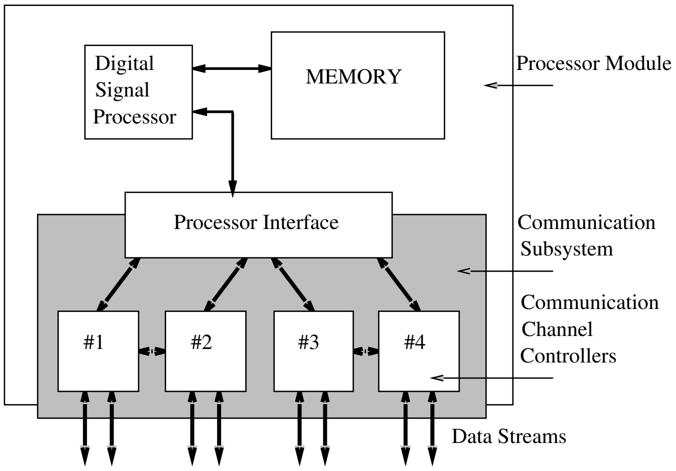

Figure 2.13: **Block diagram of the RENNS Module**

RENNS (REconfigurable Neural Network Server) is a DSP based parallel computer designed at the Norwegian University of Science and Technology (NTNU). It is a flexible general purpose neurocomputer designed to experiment with adapting the hardware to the problem at hand. One of the differences to general purpose parallel computers is the flexibility of the communication network. An in-depth description of the system is given in [58,63,64]. The computer is operative with 15 processing elements – called Processing Modules (PMs). Each consists of a Processor Module and a Communication Subsystem. A block diagram of the RENNS module is shown in Figure 2.13. The RENNS Communication Subsystem is implemented in field programmable gate arrays (FPGAs), thus it can be reconfigured in-system to meet different requirements from ANN algorithms.

Several training algorithms for different neural networks models have been implemented on the RENNS system. Several different parallel implementations of Self-Organizing Feature Maps (SOM) have been constructed by Myklebust [63,65–69]. Solheim has implemented several backpropagation learning programs based on neuron parallelism [58, 70, 71]. Utne has implemented variants of the Hopfield network [64].

2.2.2.7 Commercially Available Digital Neurocomputers

During recent years several neurocomputers have approached the market. They are all designed in digital technology[5] and most of them are general purpose machines. To obtain high performance they consist of custom designed VLSI circuits and some of them with reduced floating point precision. The neurocomputers presented in this section all provide both learning and recall.

[5] Information about commercial machines using other technologies has not been seen by the authors.

One of the first commercially available VLSI neural implementations which offered on-chip learning was the CNAPS-1064 (Connected Network of Adaptive Processors) chip from Adaptive Solutions, Inc [72,73]. Low cost is emphasized for inclusion in commercial applications. Each PE was made as small as possible to maximize the number of PEs on a single chip. The chip contains a parallel array of 64 PEs – called processing nodes (PNs), configured for SIMD operation, as depicted in Figure 2.14. The CNAPS Server II offers from 64 to 512 PNs.

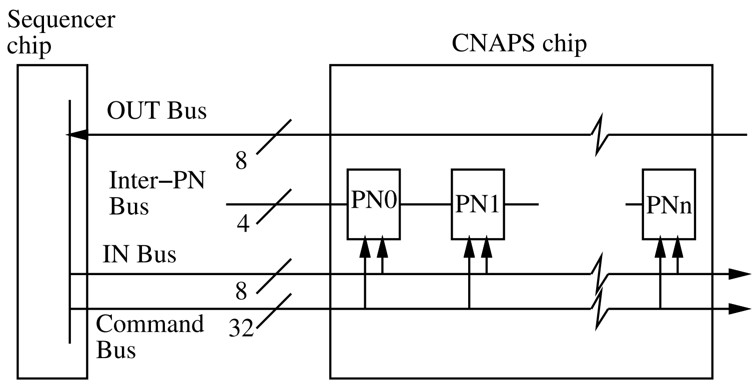

Figure 2.14: **The CNAPS architecture consisting of processing nodes (PNs) connected together by three global buses. A sequencer chip controls the flow of data and commands on the buses.**

Each processor is a simple digital signal processor with 4Kbyte of local memory. Each chip can compute 960 million multiply-accumulate operations per second. The weights can be 1, 8, or 16 bits. Multiple chips can be connected together and are controlled by a single sequencer chip. The mapping of the feed-forward network is by neuron parallelism, i.e. one or more neuron(s) is computed in each PN. Neurons from different layers may either be mapped to the same PN or separate PNs. The NETTALK network learned more than 40 times faster on a CNAPS chip than on a Sun SparcStation.

A similar chip is designed by Hitachi and included into their MY-NEUPOWER neurocomputer [74]. One chip consists of 8 digital micro-programmable PEs, each of which includes local memory for on-chip learning. The machine has been shown to learn at a maximum speed of 1.26 GCUPS. 32 neural chips are placed on each neural board. Two neural boards are connected together by a bus which interconnects all the processing elements. This makes a total of 512 processing elements. The weights are either 8 or 16 bits. The digital processing elements operate in SIMD mode, as depicted in Figure 2.15. Performance measurements show that the computer can learn 100 to 1,000 times faster than a workstation.

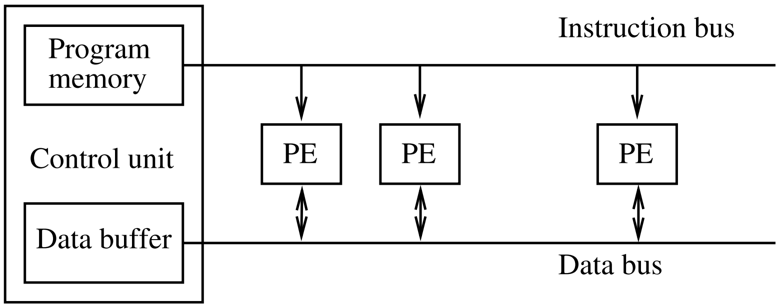

Figure 2.15: **The MY-NEUPOWER computer architecture.**

SYNAPSE-1 (SYnthesis of Neural Algorithms on a Parallel Systolic Engine) is a general purpose neurocomputer developed by Siemens AG [75, 76]. The design of the computer is shown in Figure 2.16.

It consists of four separate boards. The intensive parts of the neural computation are undertaken in the MA16 array, while non-

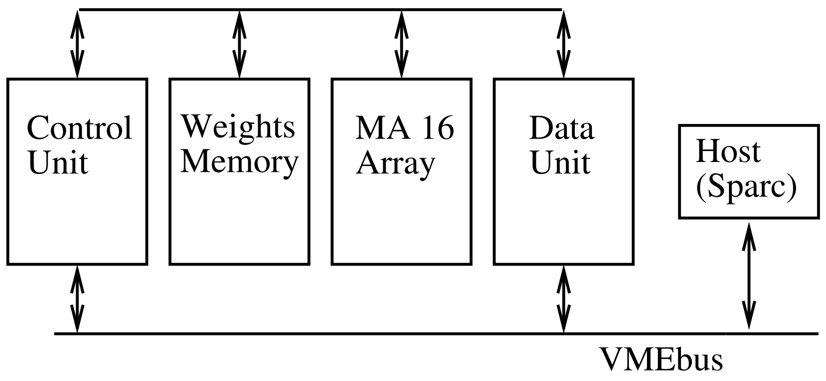

Figure 2.16: **The SYNAPSE-1 neurocomputer.**

intensive computation is performed by the Data Unit. The design differ from many others by storing the weights on a separate board. The MA16 array is a 2-dimensional systolic array of ASIC neural signal processors (MA16s), arranged in two rows of four columns. Each MA16 itself consists of a linear systolic array of 4 processing modules. A local memory is connected to each MA16 for storing intermediate results. The weight resolution default is 16 bits. A programming language, called nAPL (neural Algorithms Programming Language), is defined for the computer. The SYNAPSE-1 has proved to be 8000 times faster than a SUN SparcStation 2 for computation of one layer of neurons. The peak performances of the CNAPS server and the SYNAPSE-1 are stated to be similar. However, SYNAPSE-1 needs less additional hardware for running large applications.

The SNAP (SIMD Numerical Array Processor) system is designed by HNC for speeding up neural network applications [77, 78]. It is based on a general purpose chip, HNC100, designed with a SIMD systolic linear array of 4 processors, each of which implements 32-bit floating-point arithmetic operations. Figure 2.17 shows the layout of

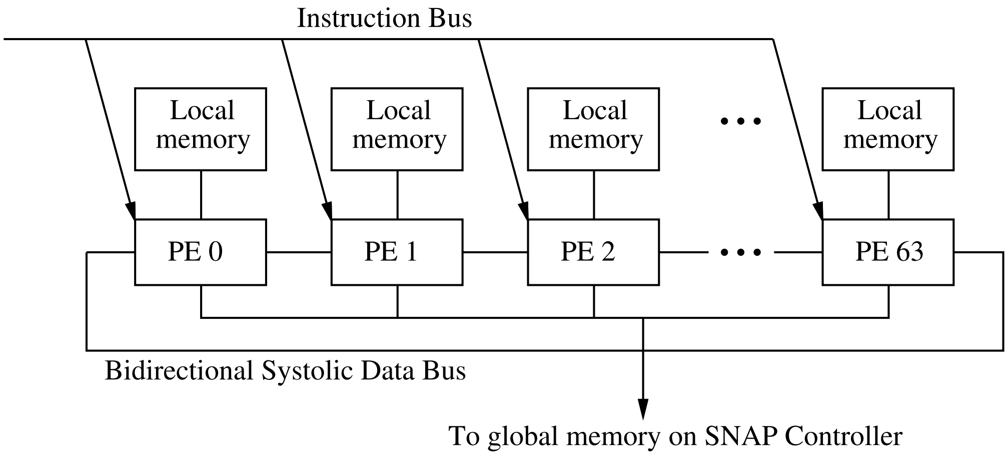

Figure 2.17: **The SNAP-64 architecture.**

the SNAP system. Each PE contains a local memory of 512 Kbyte. All the PEs are controlled by a single instruction from the external controller. Each PE is connected to a triple-ported global memory for communication to the host processor and/or peripherals. The global data bus allows data in the global memory to be broadcasted to all PEs at the same time. Alternatively, data in a single PE can be broadcasted. The SNAP system comprises of 1 to 4 boards, each with 4 HNC chips, making a total of 16 to 64 SNAP processing elements in the linear ring array. The largest system of 64 PEs provides 2.56 GFLOPS. A feed-forward network with 512 neurons in each layer learned at 302 MCUPS on the system of 64 PEs, while 64 neurons in each layer learned at 76.6 MCUPS [79].

General Remarks Except for SYNAPSE-1, all the systems operate in SIMD mode. The PEs are made as simple as possible to include as many as possible on a single chip. The BP parallelization scheme is based on neuron parallelism. It is interesting to note that synapse parallelism has not been of interest so far.

An important issue when designing a neurocomputer is the I/O interface. Research applications on neurocomputers have in some cases performed a thousand times slower than peak performance due to I/O-limits [60].

2.2.2.8 Other Technologies

The survey of BP implementations have been for digital target architectures. However, work is also continuing into the use of other technologies like analog and optoelectronics. These are based respectively on analog voltage levels and light, for signal representation. Since these technologies are not yet commercialized, a survey is not included in this study. Description of the many proposals for VLSI implementations (both digital and analog) has also been omitted here. Many of these projects are only "paper-projects" that have not yet been implemented in silicon.

A survey of Nordström and Svensson on parallel implementation of neural network includes descriptions of some special-purpose neural hardware systems [1].

A survey of hardware implementations of neural networks in Japan is done by Hirai [80]. Several characteristics of neural chips are listed, which appear in Table 2.1.

The digital approach is currently the best solution to implement neural networks with high precision, even though the circuit size for neural networks becomes larger than for the other approaches. The scalability of digital chips, allows many chips to be easily connected

Table 2.1: **Advantages and disadvantages of different chip technologies (from [80]).**

	Advantages	Disadvantages
Analog	Compact High speed	Susceptible to process-parameter variation Susceptible to noise Lacks scalability The precision of synaptic weights is limited Difficult to make modifiable synapses
Opto-electronics	Large fan-in and fan-out Modifiable synapses possible	Opto-electronic and electro-optic transformation Susceptible to noise and parameter variation
Digital	High precision High scalability Modifiable synapses	Large circuit size

together. To attain scalability, many analog chips employ a 1 bit digital output. Hirai concludes that for special purpose use, which does not require high precision, analog internal circuits with digital output is the best solution. For general purpose use and where high precision is required, the digital approach is the best. Another opinion [81] is that often it is desirable to process analog sensor/actuator data without conversion to digital form. Moreover, a number of tasks, for reasons of speed or integration, require hardware (preferably analog) implementation [82].

One of the benefits of custom design is that the designer can limit the precision to that required by the neural algorithm. However it is limited how much the precision can be reduced. Franzi reports that for fixed point representation, the neural learning only converged in a few cases [83].

Garth [82] gives the following reasons for why so few commercial systems implementing neural networks in hardware have emerged:

- Hardware solutions are frequently poorly equipped to accept modification to algorithms as they arise, i.e. the trade-off between efficiency and flexibility.
- Difficulties in providing a complete solution. External pre- and post processing may obliviate many of the advantages of the neural chip.
- Many chips achieve much of their speedup by severely restricting the precision in the number of bits. These restrictions may affect the performance of the network.

There seems to be an inverse proportional relation between performance speed and reconfigurability. By making special-purpose chips it is impossible to gain both high speed and reconfigurability. Neural chips often lack flexibility and scalability, compared to general purpose computer simulation of neural networks.

The design choice between analog and digital technology may depend on whether the research goal is biological modeling or making a programmable research tool [60]. Moreover, a theoretical high speed is useless if the machine is not flexible to do what the user requires.

2.2.2.9 Summary of the Parallel Mapping Schemes

The degrees of parallelism used parallel schemes on general purpose architectures are listed in Table 2.2. Neuron parallelism is most often used, while pipelining is rarely used. For transputer technologies, Table 2.3 indicates the parallel mapping schemes used. Training set parallelism is often used. This is probably to minimize the communication bottleneck of transputer systems. For neurocomputers, only

Table 2.2: **Parallel general purpose computer mappings of BP training. A mapping is implemented on the computer unless "M" is indicated in the leftmost column. An "M" indicates that a theoretical model of the system is used.**

Computer	# PEs	Tr. set p.	Pipel.	Neuron p.	Synapse p.	Mapping
CM-2 [21]	64K	x		x		H
CM-2 [3]	16K			x	x	H
CM-5 [22]	512			x	x	H
CM-5 [23]	32	x		x	x	H
CM-5 [11]	256	x		x	x	A
DAP 600 [32]	4096	x		x	x	-
DREAM [33] (M)					x	A
GF-11 [17]	356					H
Hypercube (M) [34]			x	x	x	A
Intel iPSC/860 [19]	32			x	x	H
Mesh (M) [30]			x	x	x	A
MP-1 [14]	4096	x		x	x	H
MP-1 [12]				x	x	H
MP-1 [15]	16K	x		x		H
NCube/4$^+$ [5]	16			x		H
Q-x system (M) [28]	64	x		x		H
SCAP (M) [8,40]				x	x	A
Systolic array (M) [35]			x	x	x	A
Warp [24]	10	x				H

Table 2.3: **Transputer implementations of BP. In the leftmost column, "M" indicates that a theoretical model of the system is used.**

Computer	# PEs	Tr. set p.	Pipel.	Neuron p.	Synapse p.	Mapping
TLA/PLA [10]	16			x	x	A
2D-torus [48]	16		x	x	x	H
Ring (M) [50]	15	x	x			A
Ring [51]	80	x		x	x	A
Ring (M) [52,53]	–	x				H
Ring [49]	32	x				H
Tree [55]	9	x				H
Hypercube [56]	8	x				H
Hypercube [57]	16	x				H

Table 2.4: **BP neurocomputer implementations.**

Computer	# PEs	Tr. set p.	Pipel.	Neuron p.	Synapse p.	Mapping
CNAPS [72,73]	512			x		-
MUSIC [6]	45			x		H
MY-NEUPOWER [74]	512			x		-
RAP [59]	40			x		H
Sandy [62]	256			x		H
SNAP [62]	256			x		H

implementations employing neuron parallelism are yet published, see Table 2.4. However, few implementations for these computer architectures have been reported.

The majority of parallel *implementations* in the undertaken survey are of the heuristic kind, see Table 2.2. The assignment of processors to each degree of parallelism does not change according to the network size and training set size. This represents a problem since only a few neural network applications will train near optimally for fixed implementations.

The proposed algorithmic mappings are, with a few exceptions ([10,11,51]), only theoretical studies. Moreover, some sophisticated methods are suggested but only implemented on small systems, e.g. 16 transputers [10].

2.2.2.10 Summary of the Performance

The training performance of the parallel back propagation implementations presented in Section 2.2.2 are listed in Table 2.5. Performance is measured in CUPS.

The NETTALK network has been used in many of the reports.

However, the commercial neurocomputer companies seem to avoid using this benchmark. Since different networks are used to measure the learning speed, direct comparison between them is not possible. Also, the comments in Section 1.3.1.4 regarding the CUPS measure should be borne in mind when comparing the results. The performance is highly dependent on the degree of optimization of the program code [86]. The double loops can be organized so that the data variables can be kept in fast memory. To optimize speed, the code must be written in assembly language.

2.2.3 Neural Network Applications

To efficiently utilize parallel processing for speeding up neural network training, we should be aware of which types of networks and training sets are used in todays neural applications. This section provides a survey of neural network applications. Mainly large applications – where parallel processing is of interest, will be described. Feed-forward neural network (FF-NN) with a single hidden layer is assumed, unless otherwise stated.

Two of the main contributors to the neural network research, Widrow and Rumelhart, recently published the paper called “Neural networks: Applications in Industry, Business and Science” [89]. They list a wide variety of commercial applications for neural networks. The increase in commercial products in the last years is partly explained by the availability of an increasingly wider array of dedicated hardware. Many of the products have to be cheap to be of commercial interest.

2.2.3.1 Speech

Speech Recognition Several research groups are working on the difficult task of continuous speech recognition. Promising results using neural networks are described in [90]. However, the network and training set need to be large. For recognizing 300 sentences – speaker independently, the system achieved 4-5% error, which is competitive with statistically based systems. For a larger recognition problem, the error for the neural network system became twice that of the best mainstream system. However, the mainstream system is larger than the neural network based system. As the speech networks get larger they tend more towards networks that are not fully connected [60]. This is in the form of fully-connected subnets.

Speech Synthesis NETTALK is a two layer feed-forward network that transforms text to phonemes[6], using 203 input units, 60–120 hidden units and 26 output units [91]. The input to the network is series of 7 consecutive letters from one of the training words, see Figure 2.18.

The central letter is the one for which the phonetic output is to be produced. The three letters on each side of central character help to determine the pronunciation. The input word is scrolled through the window of 7 letters so that each letter, one after another, is placed in the center of the window. The network can be used for continuous text or a dictionary of words. For the latter approach, the words are moved through the window individually. Thus, empty letters are added in front of and after the input word as seen in the Figure 2.18.

[6]Elementary speech sounds.

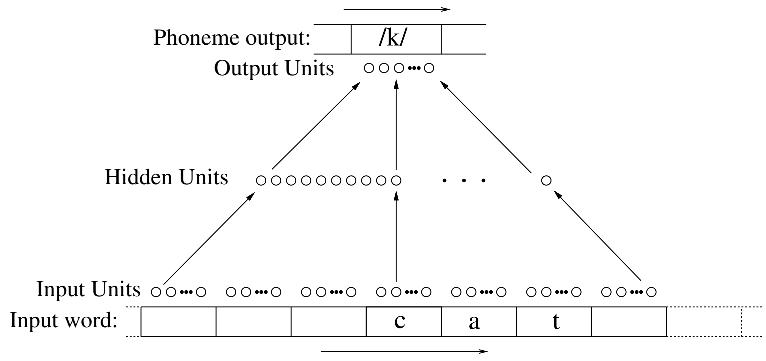

Figure 2.18: **Schematic drawing of the NETTALK network.**

For a 1000 word training set, 98 % correct transformation was reported. The training set is freely available and is often used as a benchmark.

2.2.3.2 Image Processing

Satellite Images Remote sensing is of interest for agricultural, environmental, and weather control applications. For some of these it is necessary to identify human-made structures like roads or residential areas. Wolfer has shown that neural networks can be used to extract roads in raw Landsat TM images [92].

Face Recognition High order neural networks have been used for human face recognition [93]. The network consists of a preprocessing network followed by a feed-forward network with one or two hidden layers. Four image planes, each of 16 x 16 elements, are input to the network. The originally proposed network had to be reduced to make computer simulations possible on a i486 personal computer.

The recognition rate varied according to the transforming – scaling and rotation, of the input image. In the best case, over 96 % of the transformed training patterns were classified correctly.

Medical Imaging Much research is published within the field of medical imaging [94]. Molecular biology is one area in which neural networks often outpace traditional tried methods. However, in most other areas, neural networks have not yet been shown to outperform statistical techniques. A feed-forward neural network used for cancer cell classification is described in [95]. A classification accuracy of as high as 96.8 % was obtained. The method is based on using extracted features from the raw image as input to the network.

Analyzes of smears for cervical cancer is investigated by McKenna [96]. 80 features were derived from the Fourier spectrum of each 256x256 pixel image and used as input to the network. The classification rate was 92.9 %, higher than the 91.3 % achieved by the Bayesian classifier[7]. One hidden layer resulted in better performance than two hidden layers. The classification rate by using the neural network was for all experiments higher than by using the Bayesian classifier. A second experiment indicated that images of higher resolution improved the classification rate.

Blood vessel detection in angiograms has been shown to obtain a superior 92% classification performance by using BP trained networks, compared to 68% and 83% for Bayesian maximum likelihood classifier (MLE) and iterative ternary classification (ITC), respectively [97]. Including more hidden layers was shown not to result in a performance improvement.

[7]Statistical pattern recognition.

Object Inspection Object recognition is one of the key problems in factory automation. Kang [98] proposes a system for inspection of objects moving down a conveyor belt based on a feed-forward neural network. 94% correct recognition was obtained for the two objects used in the experiments.

Onda et al. [99] have designed a neural system for identification of metal welding defects. The average identification rate is 75 %, reported to be approximately equivalent to that obtained by expert engineers. Training of the network required about three hours on their neuroboard based system.

Shiotani et al. [100] show that a FF-NN can be used to speed up the recognition of overlapping plant cell objects. The neural network directs the movement of a window to place an object of interest within the window. Afterwards, a template matching system is used to check the located object.

Road and Obstacle Recognition A vision system for real time recognition of traffic situations is described by Tsinas [101]. Separate FF-NNs are used for recognizing the driving lane and obstacles on the lane, respectively. The time for processing an image of a road scene varied between 0.1 s and 0.65 s on a 486 PC running at 50 MHz. Recognition of 2 minutes of real time video recording reported the one obstacle present and produced one false alarm.

Image Compression Multi-layer neural networks can be used to transform image data into compressed data by reduction of the spatial redundancy [102]. The number of neurons in the hidden layer is chosen to be smaller than in the input and output layer. Thus, the output of the hidden layer is a compressed image, and the output layer de-compresses the image. In order to increase the generalization properties, the image is dived into blocks. By dividing the original image into 8x8 pixel blocks and using an adaptive compression ratio according to the complexity of each block, 25:1 compression ratio was obtained by Cho et al. [103]. However, they report higher ratios by use of Self-organizing Maps.

Optical Character Recognition Recognition of characters in printed documents seems to be almost error free, whereas handwritten characters still make recognition prone to errors. In the work of Diep the network recognizes multi-size and multi-font printed characters successfully [104]. Garis achieved handwritten writer-independent digit recognition of 92% [105]. The network consists of a preprocessing network followed by a single hidden layer network. However, combing two neural networks, one which inputs the digit image and one that inputs the contour of the digits, resulted in 97.8% correct recognition [106].

Recognizing Paper Bills and Coins Automatic recognition of paper bills and coins is an important issue for vending machines and within banking. Error free recognition of Japanese or U.S. paper currency is proposed by Takeda and Omatu [107]. To reduce the size of the network a technique called *random masks* is used. Some parts of the scanned input are covered using random masks. The sum of the pixels not covered is used as input to one input unit. Other masks are used for the other input units. The masks used in the training of the network are also applied when the network is used for recognition. This method implies that the number of weights in the network can be reduced to 1/10 of the original number, without increasing the recognition error.

A coin classification system can be based on several different kinds of sensors – acoustic and weight based, used as inputs to a FF-NN. 100% accuracy was obtained by this kind of system for recognition of British coins [108].

2.2.3.3 Miscellaneous Applications

Weather Prediction Weather prediction normally involves computations using approximations of complex mathematical models. The prediction models require detailed information of the areas to be studied. Carpintero et al. [109] used Adaptive Time-Delay neural networks for local, short-term weather forecasting. The input is METOSAT images and local information about temperature, pressure, speed and wind direction, and visibility. Promising predictions of temperature and wind speed were obtained, although no comparison with traditional methods was undertaken.

Rainfall prediction using a neural network with 2 hidden layers is presented in [110]. Infrared-light and visible-light images from geostationary meteorological satellite (GMS) are used as input data. An average classification accuracy of approximately 90 % is reported.

Sonar Return Classification Gorman et al. [111,112] used feed-forward neural network to the classification of sonar returns from two underwater targets, a metal cylinder and a similar shaped rock. The network classification performance was shown to be better than that of trained human listeners. The network uses one hidden layer and 60 continuous-valued input units. The best performance was achieved in the case of 24 hidden units. However, this was only slightly better than that achieved with only 12 hidden units. Two output units were used, where (1,0) represented a return signal from a metal cylinder, and (0,1) represented a return signal from a rock. A set of 208 returns (111 cylinder returns and 97 rock returns) were used in the experiments.

2.2.3.4 Commercial Products

Magnetic Character Reader Verifone has developed the Onyx check reader, which does Magnetic-Inc Character Recognition (MICR) of the digits on the bottom of checks. The company claims 99.6 % accuracy even with checks that are crumpled or overwritten [113]. The character reader uses a custom chip designed by Synaptics. The chip combines a retina-like optical sensor with a neural network.

Financial Forecasting By training a neural network to mimic a market, its prediction can be used to guide investments. As described by Hammerstrom [113], many companies are secretly using neural networks for various financial tasks. However, little information is available on their methods and results, since neural networks have demonstrated an outstanding performance. A system mentioned in the article was tried for two months with a $ 10 million investment. The performance was 2.8 % above bench-marked performance.

Process Control Neural networks are ideal for process control because they can build predictive models of the process directly from multidimensional data collected from sensors [113]. The networks need history – which is often abundant, and not theory – which is often absent. Thus, neural networks are well-suited for predicting, controlling, and optimizing industrial processes.

2.2.3.5 Summary of Applications

Table 2.6 lists a summary of back propagation applications. As far as possible, variations in the network structure are indicated. Some freely available training sets are given in Table 2.7.

2.2.3.6 Do Neural Network Applications Need Parallel Hardware?

From the survey in the previous section, at least two common characteristics of many applications are noticeable. They indicate that the ANNs are quite different from the complex organization of the human brain. First, the networks are small enough to be trained and applied on a computer with a single processor. Only in a few instances has parallel processing been used. The survey concentrated on large networks, thus many other applications using small networks have been omitted. Second, the output neuron layer often consists of a small number of neurons. The latter indicates that allocation of many processors for computing the output layer is unnecessary.

Several researchers have reported the need for selection of good feature vectors to make the network smaller and executable on their serial computers. They demonstrate that the same level of performance can be obtained as when using larger networks. Reports about improved performance by using larger neural network are few. Mainly all are based on a single hidden weight layer. Most of the published applications are from academic work. Since commercial companies rarely publish in scientific conferences, there probably exist many commercial applications which are not described here.

It is obvious that some of the neural network applications require parallel processing, i.e. for speech recognition this is required both for training of the network and when the network is applied for recognition of real time speech. However, quite many applications can be successfully implemented without parallel hardware. This may also be necessary for a product to be marketable, e.g. control applications in consumer products. However, it is still possible for parallel processing to be used to reduce the *training time* of an application.

2.2.4 Conclusions on Neural Applications and Parallel Hardware

In Section 2.2.3.6 it was reported that few researchers studying neural network applications were using parallel processing. In this section, we recorded that researchers making parallel programs and hardware for neural computing have mainly been running one or a few benchmark programs, e.g. NETTALK. In addition, for some instances only the maximum obtainable performance is reported. This is usually based on a large non-realistic sized network.

To make parallel computers more accessible to application developers there ought to be a standardized neural network language. Several languages have been proposed, e.g. PYGMALION [118], without leading to frequent employment.

It would seem that parallel processing will be employed in the future for applications that need fast real-time performance like speech recognition and image processing applications. A market for special purpose hardware for sensor and control applications will also exist. These assumptions are supported by the fact that the number of *hardware* related papers submitted to neural network conferences is said to be dropping. However, the remaining papers are principally about sensor and control applications.

Few experiments have been undertaken to see if a larger and more complex neural network together with a larger number of training patterns can gain a better application performance – e.g. recognition rate, than the neural models used today. Parallel processing seems to be the tool to make such experiments possible. If it becomes a success, it may bring artificial neural networks into the number one method for many application, where neural networks today are striving to show better performance than traditional methods.

Table 2.5: **Performance of neural network training. μ is the weight update interval, and FP is the floating point precision. The horizontal lines separate general purpose computers, transputer systems, and neurocomputers, respectively.**

Computer	No of PEs	Network size	μ	FP	MCUPS
Sun-3 [19]	1	NETTALK	1	32	0.034
NCube/4$^+$ [5]	16	Optimal network	1	32	0.19
Sun SparcStation 10 [84]	1	–	1	32	1.1
Alpha Station [84]	1	–	1	32	3.2
CM-2 [85]	16K	NETTALK (60)	1	–	2.8
Cray 2 [22]	4	257 x 256 x 131,072	1	32	10[1]
iPSC/860 [19]	32	NETTALK (80)	2000	32	11
MP-1 [14]	4096	128 x 64 x 16	1536	32	12
IBM RISC/6000 550 [86]	1	1000 x 1000 x 1	1000	32	17.6
Cray X-MP [22]	4	257 x 256 x 131,072	1	32	18[1]
CM-5 [23]	32	NETTALK (80)	lbb	32	18.33
CM-2 [21]	65536	NETTALK (80)	4096	32	40
Cray Y-MP [22]	2	257 x 256 x 131,072	1	32	40[1]
MP-1216 [15]	16384	NETTALK (120)	lbb	32	41
Q-x system [28]	64	256 x 128 x 256	8192	32	44[1]
CM-5 [11]	256	1024 x 256 x 64	lbe	32	50
Fujitsu VP-2400/10 [25]	1	NETTALK (60)	1	32	60
CM-5 [22]	512	257 x 256 x 131,072	1	32	76
CM-2 [87]	64k	128 x 128 x 128	65536	32	350
TLA [10]	16	256 x 64 x 256	1	32	0.6
Transputers [54]	53	NETTALK (60)	1500	32	1.54
MEIKO [51]	120	512 x 256 x 64	30	32	8
Sandy [62]	32	NETTALK (60)	1	32	42
Cellular arch. [88]	4175	NETTALK (60)	1	16	51.5[1]
SNAP [79]	64	64 x 64 x 64	-	32	76.6
RAP [59]	40	640 x 640 x 640	1	32	102
MUSIC [6]	45	–	1	32	203
SNAP [79]	64	512 x 512 x 512	-	32	302
Sandy [62]	256	Optimal network	1	32	567
MY-NEUPOWER [74]	512	Optimal network	-	16	1,260
CNAPS [72, 73]	512	1900 x 500 x 12	1	16	2,379[1]

[1] The performance has been estimated or simulated (not measured)

Table 2.6: **Neural network applications using feed-forward neural networks. A reference to the data set is given if it is freely available. (P) indicates that a preprocessing network is used in front of the FF-NN inputs.**

Application	Data set	Network size (I x H x 0)	No of tr. pat.
Blood vessel det. in angiograms [97]		121 x 17 x 2	75
Cancer cell classification [95]		3600 x 20 x 1	467
Coin recognition [108]		259 x 5 x 6	–
Handwritten digit recognition (P) [105]	[114]	32 x 15 x 10	2000
Image compression [103]		64 x 8,6,4 (x 64)	–
Location of plant cells in images [100]		100 x 15 x 10	52;67;69
NETTALK text-to-phonemes [91]	[115]	203 x 60-120 x 26	5438
Object detector [101]		961 x 50 x 1	50
Object inspection [98]		4096 x 64 x 2	–
Optical character recognition [104]		3000 x 20 x 94	94
Optical character recognition [104]		2500 x 100 x 94	1,128
Papanicoloau smear cell classification [96]		80 x 6 x 1	702
Paper currency recognition [107]		128 x 64 x 12	–
Road classifications in satellite images [92]		56 x 20 x1	552
Speech recognition [90]		234 x 1000 x 61	1,300,000
Speech recognition [90]		351 x 4000 x 61	6,000,000
Welding defect identification [99]		15 x 10 x 9	1024

Table 2.7: **Freely available data training sets.**

Data set	No of elements per pattern.	No of patterns
26 capital letters, 20 different fonts [116]	16 units	20,000
Isolated handwritten digits (49 writers) [114]	32 x 32 pixels	3,471 characters
NETTALK corpus [115]	29 per character	20,008 English words
Sonar return data [117]	60 units	208

Bibliography

[1] Tomas Nordstrom and Bertil Svensson, "Using and designing massively parallel computers for artificial neural networks," *Journal of Parallel and Distributed Computing*, vol. 14, pp. 260–285, March 1992.

[2] Alexander Singer, "Implementation of artificial neural networks on the Connection Machine," *Parallel Computing*, vol. 14, pp. 305–315, Summer 1990.

[3] C.R. Rosenberg and G. Blelloch, "An implementation of network learning on the Connection Machine.," *Connectionist Models and their Implications.* (D. Walz and J. Feldman, eds.), pp. 329–340, Ablex, Norwood, NJ, 1988.

[4] Hyunsoo Yoon and Jong H. Nang and S.R Maeng, "Parallel simulation of multilayered neural networks on distributed-memory multiprocessors," *Microprocessing and Microprogramming*, vol. 29, pp. 185–195, 1990.

[5] E.J.H. Kerckhoffs and F.W. Wedman and E.E.E. Frietman, "Speeding up backpropagation training on a hypercube computer," *Neurocomputing*, vol. 4, pp. 43–63, 1992.

[6] U.A. Muller and B. Baumle and P. Kohler and A. Gunzinger and W. Guggenbuhl, "Achieving supercomputer performance for neural net simulation with an array of digital signal processors," *IEEE Micro*, pp. 55–65, October 1992.

[7] Takashi Yukawa and Tsutomu Ishikawa, "Optimal parallel back-propagation schemes for mesh-connected and bus-connected multiprocessors," *Proc. of IEEE Int. Conference on Neural Networks*, pp. 1748–1753, 1993.

[8] Wei-Ming Lin and Viktor K. Prasanna and K. Wojtek Przytula, "Algorithmic mapping of neural network models onto parallel SIMD machines," *IEEE Transactions on Computers*, vol. 40, pp. 1390–1401, December 1991. ISSN: 0018-9340.

[9] Yoshiji Fujimoto, "An enhanced parallel planar lattice architecture for large scale neural network simulation," *Proc. of Int. Joint Conference on Neural Networks*, vol. 2, (San Diego), pp. 581–586, 1990.

[10] Yoshiji Fujimoto and Naoyuki Fukuda and Toshio Akabane, "Massively parallel architecture for large scale neural network simulation," *IEEE Trans. on Neural Networks*, vol. 3, pp. 876–887, November 1992.

[11] Vipin Kumar and others, "A scalable parallel formulation of the back propagation algorithm for hypercubes and related architectures," *IEEE Trans. on Parallel and Distributed Systems*, vol. 5, pp. 1073–1090, October 1994.

[12] G. Chinn and others, "Systolic array implementations of neural nets on the MasPar MP-1 massively parallel processor," *Proc.*

of Int. Joint Conference on Neural Networks, vol. II, pp. 169–173, 1990.

[13] Grajski, Kamil A., *Parallel Digital Implementations of Neural Networks*, pp. 51–76. PTR Prentice Hall, 1993. In Neurocomputing using the MasPar MP-1, K. Wojtek Przytula and Viktor K. Prasanna editors.

[14] Antonio d' Acierno and Roberto Vaccaro, "A parallel implementation of the back-propagation or errors learning algorithm on a SIMD parallel computer," *Proc. of Int. Conference on Artificial Neural Networks*, pp. 1074–1077, 1993.

[15] Andreas Zell and others, "Problems of massive parallelism in neural network simulation," *Proc. of IEEE Int. Conference on Neural Networks*, pp. 1890–1895, 1993.

[16] Kai Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability.* McGraw-Hill, Inc., 1993.

[17] Michael Witbrock and Marco Zagha, "An implementation of backpropagation learning on GF11, a large SIMD parallel computer," *Parallel computing*, vol. 14, pp. 329–346, 1990.

[18] Shumping Tang and Marino J. Niccolai and Michael W. Bringmann, "A model for the execution of a neural network upon a multiprocessor," *Proc. of the Fifth Workshop on Neural Networks: Academic/Industrial/NASA/Defence*, pp. 259–262, 1993.

[19] Darin Jackson and Dan Hammerstrom, "Distributing back propagation networks over the Intel iPSC/860 hypercube," *Proc. of Int. Joint Conference on Neural Networks*, vol. I, pp. 569–574, 1991.

[20] Tucker, Lewis W. and Robertson, George G., "Architecture and Applications of the Connection Machine," *Computer*, vol. 21, pp. 26–38, Aug. 1988.

[21] Xiru Zhang, "The backpropagation algorithm on grid and hypercube architectures," *Parallel Computing*, vol. 14, pp. 317–327, Summer 1990.

[22] Xiao Liu and George L. Wilcox, "Benchmarking of the CM-5 and the Cray machines with a very large backpropagation neural network," *Proc. of IEEE Int. Conference on Neural Networks*, vol. 1, pp. 22–27, 1994.

[23] J.M. Adamo and D. Anguita, "Object oriented design of a BP neural network simulator and implementation on the Connection Machine (CM-5)," tech. rep., International Computer Science Institute, September 1994. TR-94-46.

[24] D. A. Pommerleau and others, "Neural network simulation at Warp speed: How we got 17 million connections per second," *Proc. of IEEE Int. Conference on Neural Networks*, 1988.

[25] E. Sànchez and S. Barro and C.V. Regueiro, "Artificial neural networks implementation on vectorial supercomputers," *Proc. of IEEE Int. Conference on Neural Networks*, (Orlando, FL), pp. 3938–3943, June 28 - July 2, 1994.

[26] Wayne Allen and Avijit Saha, "Parallel neural-network simulation using back-propagation for the ES-kit environment," *Proc. of 1989 Conf. Hypercubes, Concurrent Computers and Application*, pp. 1097–1102, 1989.

[27] Lon-Chan Chu and Benjamin W. Wah, "Optimal mapping of neural-network learning on message-passing multicomput-

ers," *Journal of Parallel and Distributed Computing*, vol. 14, pp. 319–339, 1992.

[28] Ali M. Alhaj and Hiroaki Terada, "Exploiting parallelism in neural networks on a dynamic data-driven system," *IEICE Trans. on fundamentals*, vol. E76-A, pp. 1804–1811, October 1993.

[29] Vipin Kumar and Ananth Grama and Anshul Gupta and George Karypis, *Introduction to parallel computing*, ch. 5. Benjamin-Cummings, 1993.

[30] Suzuki,Y. and Atlas,L., "A comparison of processor topologies for a fast trainable neural network for speech recognition," *Proc. IEEE Intl. Conf. on Acoustics, Speech and Signal Processing, Glasgow*, May 1989.

[31] Magali E. Azema-Barac, "A conceptual framework for implementing neural networks onto massively parallel machines," *6th International Symposium on Parallel Processing (IPPS)*, (IEEE Press), pp. 527–530, 1992.

[32] Magali E. Azema-Barac, "Neural network implementations and speed-up on massively parallel machines," *Microprocessing and microprogramming*, vol. 35, pp. 747–754, 1992.

[33] Soheil Shams and Jean-Luc Gaudiot, "Implementing regularly structured neural networks on the DREAM machine," *IEEE Trans. on Neural Networks*, vol. 6, pp. 407–421, March 1995.

[34] Q.M. Malluhi and M.A. Bayoumi and T.R.N. Rao, "A parallel algorithm for neural computing," *Proc. of World Congress on Neural Networks*, vol. 2, pp. 563–569, 1994.

[35] S. Y. Kung and J. N. Hwang, "Parallel architectures for artificial neural nets," *Proc. of IEEE Int. Conference on Neural Networks*, vol. 2, (San Diego), pp. 165–172, July 24-27 1988.

[36] S. Y. Kung and J. N. Hwang, "A unified systolic architecture for artificial neural networks," *Journal of Parallel and Distributed Computing*, vol. 6, pp. 358–387, 1989.

[37] S. Y. Kung and J. N. Hwang, "A unifying algorithm/architectures for artificial neural networks," *Proc. of International Conference on Acoustics, Speech and Signal Processing*, vol. 4, pp. 2505–2508, 1989.

[38] Sun-Yuan Kung and Wei-Hsin Chou, "Parallel Digital Implementations of Neural Networks," *Mapping of neural networks onto VLSI array processors* (K. Wojtek Przytula and Viktor K. Prasanna, eds.), ch. 1, pp. 3–49, PTR Prentice Hall, 1993.

[39] S.Y. Kung, *Digital Neural Networks*. PTR Prentice Hall, 1993.

[40] K. Wojtek Przytula and Viktor K. Prasanna and Wei-Ming Lin, "Parallel implementations of neural networks," *Journal of VLSI Signal Processing*, vol. 4, pp. 111–123, May 1992.

[41] James G. Eldredge and Brad L. Hutchings, "RRANN: The run-time reconfiguration artificial neural network," *Proc. of the IEEE Custom Intergrated Circuits Conf.*, pp. 77–80, 1994.

[42] Nazeih M. Botros and M. Abdul-Aziz, "Hardware implementation of an artificial neural network using field programmable gate arrays," *IEEE Trans. on Industial Electronics*, vol. 41, pp. 665–667, December 1994.

[43] Arne Linde and Tomas Nordstrom and Mikael Taveniku, "Using FPGAs to implement a reconfigurable highly parallel computer," *Selected papers from: Second International workshop on Field-Programmable Logic and Applications (FPL'92)*, pp. 199–210, Springer-Verlag, 1992.

[44] Stephen L. Bade and Brad L. Hutchings, "FPGA-based stochastic neural networks – Implementation," *Proc. of IEEE Workshop on FPGAs for custom computing machines*, pp. 189–198, 1994.

[45] Jacob M. J. Murre, "Transputers and neural networks: An analysis of implementation constraints and performance," *IEEE Trans. on Neural Networks*, vol. 4, pp. 284–292, March 1993.

[46] Hyunsoo Yoon and Jong H. Nang, "Multilayer neural networks on distributed-memory multiprocessors," *INNC*, pp. 669–672, 1990.

[47] I. Cloete and J. Ludik, "Parallelization of artificial neural networks on transputers," *Proc. of the International Conference on Parallel Computing '91*, pp. 367–373, Elsevier Science Publishers B.V., 1992.

[48] Alain Petrowski and Gérard Dreyfus and Claude Girault, "Performance analysis of a pipelined backpropagation algorithm," *IEEE Trans. on Neural Networks*, vol. 4, pp. 970–981, November 1993.

[49] Helene Paugam–Moisy, "Parallel neural computing based on neural network duplicating," *Parallel algorithms for digital im-*

age processing, computer vision and neural networks (Ioannis Pitas, ed.), ch. 10, pp. 305–340, John Wiley & Sons, 1993.

[50] S. Zickenheimer and M. Wendt and B. Klauer and K. Waldschmidt, "Pipelining and parallel training of neural networks on distributed-memory multiprocessors," *Proc. of IEEE Int. Conference on Neural Networks*, vol. 4, pp. 2052–2057, 1994.

[51] Antonio d' Acierno and Roberto Vaccaro, "The backpropagation learning algorithm on parallel computers: A mapping scheme," *Proc. of Sixth Italian Workshop Neural Nets WIRN VIETRE - 93*, (Salerno), pp. 249–254, 1994.

[52] Shou King Foo and others, "A theoretical study of training set parallelism for backpropagation networks on a transputer array," *Proc. of World Congress on Neural Networks*, vol. 2, pp. 519–524, 1994.

[53] Shou King Foo and P. Saratchandran and N. Sundararajan, "Comparison of parallel and serial implementation of feedforward neural networks," *Journal of Microcomputer Applications*, vol. 17, pp. 83–94, January 1995.

[54] Shou King Foo and P. Saratchandran and N. Sundararajan, "Analysis of training set parallelism for backpropagation neural networks," *Int. Journal of Neural Systems*, vol. 6, pp. 61–78, March 1995.

[55] K. Diakonikolaou and S. Kollias and D. Kontoravdis and A. Stafylopatis, "Implementation of neural network learning strategies on a transputer-based parallel architecture," *Parallel and Distributed Computing in Engineering Systems. Proc.*

of the IMACS/IFAC International Symposium, pp. 365–370, North-Holland, June 1991. ISBN: 0 444 89276 1.

[56] Willis K. King and Goffredo Pieroni and Daniele Micci Barreca and Blaž Zupan, "A fast implementation of the back-propagation algorithm," *Proc. of Sixth Italian Workshop Neural Nets WIRN VIETRE – 93*, (Salerno), pp. 218–226, 1994.

[57] D.M. Weber, "Parallel implementation of time delay neural networks for phoneme recognition," *Proc. of IEEE Int. Conference on Neural Networks*, pp. 1583–1587, 1993.

[58] J. G. Solheim, *The RENNS approach to neural computing.* PhD thesis, Norwegian Institute of Technology, In preparation.

[59] Nelson Morgan and others, "The ring array processor: A multiprocessing peripheral for connectionist applications," *Journal of Parallel and Distributed Computing*, pp. 248–259, April 1992.

[60] K. Asanovic and J. Beck and J. Feldman and N. Morgan and J.Wawrzynek, "Designing a connectionist network supercomputer," *International Journal of Neural Systems*, vol. 4, pp. 317–326, December 1993. ISSN: 0129-0657.

[61] Hideki Kato and others, "A parallel neurocomputer architecture with ring registers," *Proc. of an international conference organized by the IPSJ to commemorate the 30th anniversary*, pp. 233–240, 1990.

[62] Hideki Yoshizawa and Kazuo Asakawa, "Highly parallel architecture for back-propagation using a ring register data path," *Fujitsu Sci. Tech. J*, pp. 227–233, September 1993.

[63] Myklebust, Gaute, *Implementations of an unsupervised neural network model on an experimental multiprocessor system.* PhD thesis, Norwegian Institute of Technology, 1996. ISBN 82-7119-893-9.

[64] Lisbet Utne, *Design of a reconfigurable neurocomputer Performance analysis by implementation of recurrent associative memories.* PhD thesis, Norwegian Institute of Technology, 1995. ISBN 82-7119-793-2.

[65] Myklebust, Gaute and Solheim, Jon and Greipsland, Jarle, "Self organizing maps on a reconfigurable computer," *Proceedings of the Sixth International Conference on Signal Processing Applications and Technology*, Oct. 1995.

[66] Myklebust, Gaute and Solheim, Jon Gunnar, "Parallel Self-organizing Maps for actual applications," *Proceedings of the IEEE International Conference on Neural Networks*, Dec. 1995.

[67] Myklebust, Gaute and Solheim, Jon Gunnar and Steen, Erik, "Speeding up small sized Self Organizing Maps for use in visualization of multispectral medical images," *Eighth IEEE Symposium on Computer-Based Medical Systems*, (Lubbock, Texas), pp. 103–110, IEEE Computer Society Press, June 1995.

[68] Myklebust, Gaute and Solheim, Jon Gunnar and Steen, Erik, "Wavefront implementation of Self Organizing Maps on RENNS," *International Conference on Digital Signal Processing*, (Limassol, Cyprus), pp. 268–273, 1995.

[69] Myklebust, Gaute and Solheim, Jon Gunnar and Tørresen, Jim, "Parallel implementations of self organizing maps on

RENNS," *Proceedings of The Norwegian Neural Network Seminar*, SINTEF Instrumentation, Dec. 1994. Report no. STF31 S94026, ISBN 82-595-8893-5.

[70] Solheim, Jon Gunnar and Myklebust, Gaute, "RENNS - an experimental computer system with a reconfigurable interconnection network," *Proceedings of the First International Workshop on Parallel Processing (IWPP '94)*, (Bangalore, India), pp. 205–210, Tata McGraw Hill, Dec. 1994.

[71] Solheim, Jon Gunnar and Myklebust, Gaute, "RENNS - a Reconfigurable Computer System for Artificial Neural Networks," *Proceedings of first IEEE International Conference on Algorithms and Architectures for Parallel Processing, Brisbane, Australia*, pp. 197–206, Apr. 1995.

[72] Dan Hammerstrom, "A VLSI architecture for high-performance, low cost, on-chip learning," *Proc. of Int. Joint Conference on Neural Networks*, vol. 2, pp. 537–542, 1990.

[73] Dan Hammerstrom, "CNAPS - 1064 Digital parallel processor," 1992. Product information AR800A, Adaptive solutions, Inc.

[74] Yuji Sato and others, "Development of a high-performance, general purpose neuro-computer composed of 512 digital neurons," *Proc. of Int. Joint Conference on Neural Networks*, pp. 1967–1970, 1993.

[75] U. Ramacher and others, "Multiprocessor and memory architecture of the neurocomputer SYNAPSE-1," *Proc. of World Congress on Neural Networks*, vol. 4, pp. 775–778, 1993.

[76] Ramacher, U. and Raab, W. and Anlauf, J. and Hachmann, U. and Weßeling, M., "SYNAPSE-1 – a general-purpose neurocomputer," tech. rep., SIEMENS AG, Feb. 1993.

[77] Robert W. Means, "High speed parallel hardware performance issues for neural network applications," *Proc. of IEEE Int. Conference on Neural Networks*, 1994.

[78] Robert W. Means and Kent Pu Qing, "Parallel image, signal and neural network processing with HNC's vision processor (ViP) and HNC's SIMD numerical array processor (SNAP)," *Int. parallel processing symposium (IPPS)*, 1993.

[79] SNAP – SIMD Numerical Array Processor, "High-performance parallel computing," 1994. Product information, Technology Development Division, HNC.

[80] Yuzo Hirai, "Hardware implementations of neural networks in Japan," *Neurocomputing*, vol. 5, pp. 3–16, 1993.

[81] Alan F Murray, "Analogue neural VLSI: Issues, trends and pulses," *Artificial Neural Networks* (I. Aleksander and J. Taylor, eds.), vol. 2, pp. 35–43, Elsevier Science Publishers B.V., 1992.

[82] SCJ Garth, "To simulate or not to simulate...," *Artificial Neural Networks* (I. Aleksander and J. Taylor, eds.), vol. 2, pp. 1397–1403, Elsevier Science Publishers B.V., 1992.

[83] Edoardo Franzi, "Neural accelerator for parallelization of backpropagation algorithm," *Microprocessing and Microprogramming*, vol. 38, pp. 689–696, 1993.

[84] Urs A. Muller, "A high performance neural net simulation environment," *Proc. of IEEE Int. Conference on Neural Networks*, (Orlando), pp. 1–4, 1994.

[85] G. Blelloch and C. R. Rosenberg, "Network learning on the Connection Machine," *Proc. of IJCAI87*, pp. 323–326, 1987.

[86] D. Anguita and G. Parodi and R. Zunino, "An efficient implementation of BP on RISC-based workstations," *Neurocomputing*, vol. 6, pp. 57–65, 1994.

[87] A. Singer, "Exploiting the inherent parallelism of artificial neural networks to achieve 1300 million interconnects per second," *Proc. of International Neural Networks Conference*, (Paris, France), pp. 656–660, July 9-13 1990.

[88] Bernard Faure and Guy Mazare, "A cellular architecture dedicated to neural net emulation," *Microprocessing and Microprogramming*, vol. 30, pp. 249–256, August 1990.

[89] Bernard Widrow and others, "Neural networks: Applications in industry, business and science," *Communication of ACM*, vol. 37, pp. 93–105, March 1994.

[90] Nelson Morgan, "Using a million connections for continuous speech recognition," *Proc. of Int. Conference On Neural Network Processing*, pp. 1439–1444, October 1994.

[91] Terrence J. Sejnowski and Charles R. Rosenberg, "Parallel networks that learn to pronounce English text," *Complex Systems*, vol. 1, pp. 145–168, 1987.

[92] James Wolfer and others, “Robust multispectral road classification in Landsat thematic mapper imagery,” *Proc. of World Congress on Neural Networks*, vol. 1, pp. 260–268, 1994.

[93] Rafal Foltyniewicz and Slawomir Skoneczny, “An improved high order neural network for invariant recognition of human faces in gray scale,” *Proc. of World Congress on Neural Networks*, vol. 1, pp. 587–592, 1994.

[94] John N. Weinstein, “Neural networks in the biomedical sciences: A survey of 386 publications since the beginning of 1991,” *Proc. of World Congress on Neural Networks*, vol. 1, pp. 121–126, 1994.

[95] Y. Hu and others, “Neural network based cancer cell classification,” *Proc. of World Congress on Neural Networks*, vol. 1, pp. 416–421, 1994.

[96] S. J. McKenna and others, “High-resolution classification of Papanicoloau smear cells using back-propagation neural networks,” *Proc. of Int. Conference on Artificial Neural Networks*, pp. 907–910, 1993.

[97] Reza Nekovei and Ying Sun, “Back-propagation network and its configuration for blood vessel detection in angiograms,” *IEEE Trans. on Neural Networks*, vol. 6, pp. 64–72, January 1995.

[98] Young-Im Kang, “Creation of an inspection system combining backpropagation networks and image processing system,” *Proc. of Int. Conference On Neural Network Processing*, pp. 174–179, 1994.

[99] Hiroyuki Onda, "Neural network application to welding defect identification," *FUJITSU Sci. Tech. Journal*, vol. 29, pp. 271–277, September 1993.

[100] Shigetoshi Shiotani and others, "Fast recognition of overlapping targets using neural networks," *Proc. of Int. Conference On Neural Network Processing*, (Seoul), pp. 614–619, 1994.

[101] Lampros Tsinas and Volker Graefe, "Coupled neural networks for real-time road and obstacle recognition by intelligent road vehicles," *Proc. of Int. Joint Conference on Neural Networks*, (Nagoya, Japan), 1993.

[102] N. Sonehara and others, "Image data compression using a neural network model," *Proc. of Int. Joint Conference on Neural Networks*, vol. 2, (Washington, D.C.), pp. 35–40, 1989.

[103] Kwang Bo Cho and others, "Image compression using multilayer perceptron with block classification and SOFM coding," *Proc. of World Congress on Neural Networks*, vol. 3, pp. 26–31, 1994.

[104] Thanh A. Diep and Hadar I. Avi-Itzhak, "A neural network approach to high accuracy optical character recognition," *Proc. of World Congress on Neural Networks*, vol. 3, pp. 76–86, 1994.

[105] M. D. Garris, "Methods for enhancing neural network handwritten character recognition," *Proc. of Int. Joint Conference on Neural Networks*, vol. 1, pp. 695–700, July 1991.

[106] A. Thepaut and Y. Autret, "Handwritten digit recognition using combined neural networks," *Proc. of Int. Joint Conference on Neural Networks*, pp. 1365–1368, 1993.

[107] Fumiaki Takeda and Sigeru Omatu, "High speed paper currency recognition by neural networks," *IEEE Trans. on Neural Networks*, vol. 6, pp. 73–77, January 1995.

[108] Martin D. Emmerson and Robert I. Damper, "Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application," *IEEE Trans. on Neural Networks*, vol. 4, pp. 788–793, September 1993.

[109] A. Carpintero and others, "Weather forecasting with adaptive time-delay neural networks: A case study," *Proc. of Int. Conference On Neural Network Processing*, pp. 842–846, 1994.

[110] Tao Chen and Mikio Takagi, "Rainfall prediction of geostationary meteorological satellite images using artificial neural network," *Int. Geoscience and Remote Sensing Symposium*, vol. 3, pp. 1247–1249, 1993.

[111] R. Paul Gorman and Terrence J. Sejnowski, "Analysis if hidden units in a layered network trained to classify sonar targets," *Neural networks*, vol. 1, pp. 75–89, 1988.

[112] R. Paul Gorman and Terrence J. Sejnowski, "Learned classification of sonar targets using a massively parallel network," *IEEE Trans. on acoustics, speech and signal processing*, vol. 36, no. 7, pp. 1135–1140, 1988.

[113] Dan Hammerstrom, "Neural networks at work," *IEEE Spectrum*, vol. 30, no. 6, pp. 26–32, 1993.

[114] National Institute of Standards and Technology, "*fl3* subset of the NIST Special Database 1, obtainable from sequoyah.ncsl.nist.gov in pub/databases/data/."

[115] Terrence J. Sejnowski, "NETtalk corpus, obtainable from ftp.idiap.ch in pub/benchmarks/neural/nettalk.tar.z."

[116] David J. Slate, "Letter image recognition data, obtainable from ics.uci.edu in pub/machine-learning-database/letter-recognition," 1991.

[117] Terrence J. Sejnowski, "Sonar, mines vs. rocks, obtainable from ftp.idiap.ch in pub/benchmarks/neural/sonar.data.z."

[118] Trealeaven, P. and Rocha, R.V., "Towards a general-purpose neurocomputing system," *Silicon Architectures for Neural Nets. Proceedings for the IFIP WG 10.5 Workshop*, pp. 167–177, North Holland, November 1991. ISBN: 0-444-89113-7.