Parallel Environments for Implementing Neural Networks

Manavendra Misra
Dept of Mathematical and Computer Sciences
Colorado School of Mines
Golden, CO 80401
Ph: (303)-273-3873 Fax: (303)-273-3875
email: mmisra@mines.edu
WWW: http://www.mines.edu/fs_home/mmisra/

Abstract

As artificial neural networks (ANNs) gain popularity in a variety of application domains, it is critical that these models run fast and generate results in real time. Although a number of implementations of neural networks are available on sequential machines, most of these implementations require an inordinate amount of time to train or run ANNs, especially when the ANN models are large. One approach for speeding up the implementation of ANNs is to implement them on parallel machines. This paper surveys the area of parallel environments for the implementations of ANNs, and prescribes desired characteristics to look for in such implementations.

1 Introduction

Although traditional von Neumann computing has been successful in many applications, it has not proved effective in solving a variety of important complex problems. At the same time, it has been observed that human beings solve these problems routinely in real time. Typical problems that fall into this class consist of perception in the visual, auditory and olfactory domains, motor control, and the synthesis and understanding of natural languages. Inspiration from biological systems has resulted in artificial neural models that show great promise in developing better solutions to these problems.

Artificial Neural Networks (ANNs) attempt to mimic the computational power of the mammalian brain by massively interconnecting very simple computational "neurons". Typically, the human brain consists of approximately 10^{11} neurons, each with an average of 10^3–10^4 connections. It is believed that the immense computing power of the brain is the result of the parallel and distributed computing performed by these neurons [84]. The result of following the design philosophy of massively interconnecting simple units has provided models that have proved to be successful in a number of applications, including text to speech conversion [86], protein structure analysis [79], autonomous navigation [75], game playing [92], handwriting recognition [9], image and signal processing [27, 88], etc. Another area that has benefited from ANN models is that of building intelligent vision models. Since roughly 60% of the human brain is involved in interpreting visual input, it is not surprising that biologically inspired systems have proved to be useful in the field of Computer Vision. Developing intelligent artificial vision systems has proved to be a very challenging task, and looking towards the human visual system for inspiration has yielded exciting results [21, 54, 74, 99, 100]. These artificial models rely heavily on highly interconnected computational units functioning in parallel.

The inspiration behind ANN models are biological models that are massively parallel, with many simple biological cells cooperating to solve problems. Therefore, implementations of the resulting massively parallel

[0] Updates, corrections, and comments should be sent to Manavendra Misra at mmisra@mines.edu.

Neural Computing Surveys vol 1, 48-60, 1997, http://www.icsi.berkeley.edu/~jagota/NCS

models on sequential machines are highly inefficient, as one processor has to simulate the models unit by unit. This results in inordinately long implementation times. It is therefore natural to try and develop implementations of these models on parallel computers. However, parallel implementation of these neural models is not trivial, and requires sophisticated algorithmic design techniques to efficiently map the model onto a chosen parallel computer. The mapping and the algorithms then need to be converted to user-friendly software environments that help neural network users run their models on available parallel machines. In this paper, we will discuss efforts that have attempted to achieve such parallel implementations, including support technologies necessary for these implementations. Our final goal is to see if a parallel environment exists that allows easy mapping of various neural network models onto parallel machines. What are some of the desired features to look for in such an environment? The neural network literature presents a number of models inspired by biology [5, 35]. Each model has strengths and weaknesses, and one model is more appropriate than another for a given application. A parallel simulation environment should provide the user with a choice of models. In addition, the user should be able to choose the parallel machine that is available to the user, and the simulation system should make it easy for the user to implement the chosen model on the chosen machine. A graphical user interface (GUI) to the environment will enhance its user-friendliness, and is thus desired. We will describe research in the area of parallel implementation of ANNs and comment on how these efforts can lead up to an ideal environment for parallel ANN implementation. Finally, we will prescribe directions that researchers interested in this area might follow.

The next section of the paper (Section 2) describes the state of the art in parallel implementations of neural networks. Section 3 lists the desired characteristics that a parallel ANN implementation environment should possess. Finally, Section 4 presents the conclusions.

2 Survey of Research

Before we discuss research in the area of parallel implementations of ANNs, let us investigate relevant work in sequential simulators of ANNs. This will allow us to identify the features that should be carried over to parallel implementation environments. A significant amount of work has been done in developing simulation environments for ANNs on sequential machines. An earlier survey of sequential ANN simulators was provided in [68]. Historically, one of the more popular simulators was provided with the PDP book [84]. This simulator was ported to a number of sequential platforms, and provided the user with an opportunity to learn about ANNs, and use them to solve problems. A newer version of this simulator (called PDP++) has been released by the developers [72]. This version provides the user with a graphical user interface (GUI) based on the InterViews toolkit through which the user can choose a particular ANN model to simulate. In addition, the GUI can be used to structure the network, choose parameters, set parameters of training, and to inspect the results of the simulation. The software is designed and implemented using object-oriented principles and is written in C++. PDP++ also provides a scripting language called CSS (C SuperScript) which can be used to debug a network, access current values of variables in the network, and add features to the basic capabilities provided in the software. This makes the simulator very flexible. However, adding features using CSS should only be used for prototyping, as code written in the scripting language runs very slowly. This is not a severe limitation, however, since CSS uses C++ syntax, so a prototype code written in CSS can be compiled into the simulator for additional speed without too much effort. The software comes with an extensive on-line manual, and has been ported to a number of sequential platforms. Executables for certain platforms, as well as source code is available freely [72]. Although parallel versions of the software are not being developed, the design of the simulator presents a number of desirable features that could be used in the development of a parallel environment.

Another excellent public domain sequential simulator has been developed at the University of Stuttgart. This simulator is called the Stuttgart Neural Network Simulator or SNNS [109, 110, 111]. Some ideas in this simulator were inspired by another sequential simulator, the Rochester Connectionist Simulator, RCS [26]. SNNS comes with an extensive and well written user's manual that makes it easy for a user to use and modify the software. SNNS also provides the user with a GUI interface whereby choices can be made about the ANN model, the structure of the network, the network parameters, etc. Unlike the new version of the

PDP simulator, SNNS has a single kernel and different models are handled as switches in the same program. A parallel version of the SNNS program has been developed for the MasPar [113]. SNNS provides an option to train multiple copies of the same ANN, or different ANNs simultaneously on a number of networked workstations. A version of SNNS that uses training pattern parallelism has also been developed for the Intel Paragon (an MIMD distributed memory machine).

A description of a number of sequential (and some parallel) ANN simulation environments is presented in [90] and [108]. These include the UCLA-SFINX [53] environment, Nexus [85], SWIM [13], NSL (Neuron Simulation Language) [104], SNNS [112], RCS [25], and Asprin/Migraines [46]. The book by J. Skrzypek therefore forms a good starting point for anyone interested in learning about software environments for simulating neural networks on sequential machines. Another interesting effort in this regard is described in [45] which talks about a simulation environment to simulate heterogeneous/hybrid ANNs. The environment is called DESCARTES (Development Environment for Simulating Connectionist ARchiTEctureS), and is written in LISP. The paper describes sequential, as well as an SIMD implementation on the Connection Machine-2. It also describes how DESCARTES might be implemented on MIMD machines.

A number of other commercial and free ANN simulators are also available for sequential machines. Some of these include PlaNet, UCLA-SFINX, Xerion, NeuroGraph, BrainMaker, Asprin-Migraines (also developed for the Cray family of vector supercomputers), and Pygmalion. Details about these simulators can be found in the Frequently Asked Questions of the newsgroup comp.ai.neural-nets [70]. Of these, BrainMaker is marketed by California Scientific Software [6] and is one of the more popular commercial programs. Its basic capabilities consist of training Back-propagation networks, but additional add-ons can be purchased. It comes with a utility called NetMaker which can import training data from a variety of formats, and thus simplify the process of creating the network. BrainMaker products are available for DOS, Windows, and Macintosh environments as well as the CNAPS parallel hardware from Adaptive Solutions Inc. [52].

A number of authors have worked in the area of developing concise languages for describing ANN architectures. These languages can then be used to create simulation environments [10] for both sequential as well as parallel machines. One effort in this regard was the Neuron Simulation Language [104] that can be used to describe single neurons, as well as networks of neurons. Another concise language (MDL) for ANN description is described in [91]. MDL allows the specification of a large network in just a couple of pages. Its parallel structure also allows highly optimized implementation on parallel machines. SLONN [103] can efficiently represent single neurons, small networks, as well as large networks.

Before an environment is developed to efficiently implement ANN models on parallel computers, there has to be a theoretical analysis of the mapping of ANN models onto various parallel machine models. A number of researchers have contributed to the area of parallel implementations of ANNs by designing and analyzing algorithms to map specific ANN models on to specific parallel architectures. A discussion of the related literature follows below.

Ghosh et al. [23] discuss the requirements to efficiently implement a generic neural network model on a multicomputer. The discussion includes mapping strategies, and an analysis of simulations of the mappings. In a similar vein, Chu and Wah [7, 101] describe optimal mapping of the learning process in multi-layer feed-forward networks on message-passing multicomputers. Predicted and actual results in applying this strategy to implement learning schemes like back-propagation on a network of Sun workstations and the Intel iPSC/2 Hypercube multicomputer are presented.

Often, the computational requirements of a neural network can be expressed as matrix-vector computations. In such cases, special algorithms can be developed to map these computations onto SIMD (Single Instruction, Multiple Data streams) machines. Techniques from the field of parallel matrix computations can also be brought to bear on this problem. Algorithmic work that can be placed in this category includes that of Przytula and Prasanna [76, 77], Lin, Prasanna and Przytula [47], Scherson [12], Shams [87], Tomboulian [94, 95]. A collection of parallel algorithms to implement ANNs is available in the book edited by Przytula and Prasanna [78].

A number of the algorithms described above deal with fully connected ANNs (networks where each unit is connected to every other unit). However, as we try to solve larger and larger problems that require ANNs with a large number of neural units, it is very likely that the connections between the units will be sparse (much like the human brain). In addition to being biologically plausible, sparse networks have also been shown to be useful in applications [3], so a number of researchers have looked at parallel implementations of sparse networks. Again, in a number of situations, parallel sparse matrix-vector techniques [55, 60] can be used to solve the problem [54, 56, 57, 58, 59, 61]. Gupta et al. have approached the problem of implementing sparse ANNs on the DAP [32], and Müller et al. have done the same on the RAP [64].

A group of researchers have chosen systolic architectures for implementing ANNs. There is an underlying similarity between the simple, special purpose computational units of a neural network, and the dedicated processing elements of a systolic array that apply a predefined computation on data elements as the data are pumped through the array. In [43, 44], S. Y. Kung and J. N. Hwang describe a scheme for designing special purpose systolic ring architectures to simulate neural nets. By recognizing that neural algorithms can be re-written as iterative matrix operations, the authors were able to directly apply techniques for mapping iterative matrix algorithms onto systolic architectures. H. T. Kung et al. [42] have reported results of implementing the Backpropagation learning algorithm on the CMU Warp. The Warp exploited coarse grained parallelism in the problem by mapping either partitions or copies of the network onto its ten systolic processors. In [51], the authors provide a formal analysis of the systolic processing required to implement a Hopfield like network for solving a particular problem. The techniques used are similar in spirit to Kung and Hwang [43, 44]. Systolic implementation of associative memory NNs (such as the Hopfield network, the Bidirectional Associative Memory, Temporal Associative Memory) are presented in [50]. Eswaran et al describe a hardware implementation of Hopfield and Hamming Nets using systolic ideas [17]. Other systolic implementations are presented by Ramacher [80] and Jones et al [39].

A number of researchers have developed algorithms for implementing ANNs on specific machines. Zhang et al. describe an implementation of multi-layer feed-forward ANNs running Backpropagation on the Connection Machine-2. The authors describe how to implement a *multiply-accumulate-rotate* iteration for a fully connected network, using the 2-D mesh connections of the CM-2. Vipin Kumar et al. present a technique for mapping the Backpropagation learning algorithm on hypercubes and related architectures [41]. They present theoretical and experimental results to show that their technique performs quite well on the nCUBE and the CM-5. Other implementations on SIMD machines include Ranka et al [82] and Wilson [106]. Deprit [11] presents results of implementing the Recurrent Backpropagation algorithm on the CM-2. The two algorithms used in this work are in turn based on [83] and [94, 95]. Grajski et al. discuss ANN implementations on the MasPar [30, 31]. Singer [89] presents a detailed comparison of five different techniques of mapping the Backpropagation algorithm onto the CM-2. In [15], the authors show that a network of 2,480 neurons with 921,600 links simulated on 16 transputers runs 14 times faster than on one transputer. The results in [16] use a four transputer network, and simulate a network with 74,996 connections to achieve a performance of 250,000 CUPS (connection updates per second). The authors of [93] implement a model called EDANN (Entropy Driven ANN) on a transputer array and show an intersting application of ANNs in orientation extraction from images. Barbosa and Lima [4] present an Occam implementation of Hopfield Nets on distributed memory architectures. Fortuna et al [18] describe a simulator for cellular neural networks called PSIMCNN that is implemented on transputer based machines. An analysis of ANN implementations on transputers is presented in [67]. Wang and Wu [102, 107] show how to simulate five different NN models (ART1, ART2, feed-forward networks, recurrent networks, and Hopfield nets) on a shared memory vector multiprocessor (the Alliant FX/80). El-Amawy and Kulasinghe [14] present a method to implement feed-forward networks on an architecture called Multiple Bus System (MBS). An MBS has p processors, and b buses, with $p \geq b$. The algorithm treats a feed-forward ANN as a feed-forward computational graph, and maps this graph onto an MBS. The authors show how an optimal mapping of the graph can be carried out.

Although the above work is extremely important, most of it focuses on mapping a single ANN to a single machine architecture and is therefore limited in scope. An ideal parallel implementation environment would provide users with optimized implementations of a variety of ANN models on a variety of parallel machines.

One area of effort not covered in this paper is that of simulation environments that attempt to simulate the operation of a single neuron in great detail (see for example [105]). These simulators model the individual neuron, and are therefore of great interest to neurobiologists.

Special Purpose versus General Purpose hardware: A researcher can choose between developing special purpose parallel hardware to implement ANNs and developing software on existing general purpose parallel digital machines. In this paper, we have concentrated on describing the work done in the latter area. However, a brief description of special purpose parallel hardware for ANN implementation is presented for completeness.

A nice overview of special purpose hardware for ANNs is presented in the book by Glesner and Pöchmüller [24]. Another overview of hardware implementations of ANNs is presented in [48]. An overview of how ANNs can be implemented in Ultra Large Scale Integration (ULSI) circuits is presented in [28]. Geller [22] and Hammerstrom [34] have designed hardware to implement neural models. Similarly, Graf et al. [29] have developed a CMOS implementation of a neural network model. James et al. [38] have also proposed special purpose hardware for neural network simulation. Morgan et al. [62, 63] have used the Ring Array Processor (RAP) (designed for speech recognition tasks) for Connectionist applications. Nordström et al. [71] discuss the implementation of ANNs on existing machines, as well as the design of new architectures specifically tuned to ANN implementation. Researchers at the University College London have built a 16-bit RISC processor with local memory and a communication unit on one chip for ANN implementation [73, 96]. Turega [97] describes a special purpose architecture consisting of one conventional processor, and a number of very simple processor and memory nodes. Ariel [20] is an architecture built by Texas Instruments that uses fast DSPs, and very large semiconductor memories in order to simulate large ANNs. Siemens built the SYNAPSE neurocomputer and its architecture is described in [81]. SYNAPSE uses 8 of Siemens' MA-16 matrix-matrix multiplier chips while the SYNAPSE 2 is a PC accelerator board with one MA-16 chip. Adaptive Solutions Inc. [1] markets special purpose ANN hardware under the name CNAPS [52]. CNAPS is based on the proprietary CNAPS-1064 Digital Parallel Processor chip that has 64 sub-processors operating in SIMD mode. Each sub-processor can emulate one or more neurons and multiple chips can be grouped together. BrainMaker [6] software has also been ported to run on the CNAPS PC cards. Nestor Inc. [69] marketed the NI1000 chip. This chip implements a network with Radial Basis Function neurons. It can store up to 1024 prototypes, with 256 dimensions, 5-bits per dimension. Another silicon implementation of ANNs came from Intel in the form of the 80170NX Electrically Trainable Analog Neural Network (ETANN) chip [37]. The ETANN implements an analog neural network with 64 inputs, 16 internal biases, and 64 neurons with sigmoidal transfer functions. It does not provide for on-chip learning. Emulation is done in software and the weights have to be downloaded to the chip.

One approach to building special purpose hardware for ANNs is to use Digital Signal Processing (DSP) chips as the building blocks for such hardware. A parallel system using DSPs was developed at the Fujitsu Labs [40]. The Processing Element (PE) in this machine is the floating point DSP TMS320C30 from Texas Instruments. PEs are arranged in a linear array and an algorithm similar to S.Y.Kung's [43] is used for the implementation. Another DSP based machine is the MUSIC system (MUlti-Signal processor with Intelligent Communication) built at ETH [66, 65]. MUSIC too can serve as a special purpose NN machine.

Special purpose implementations can be fast, efficient and cost effective. However, these implementations offer very little flexibility. Implementations on a general purpose parallel machine have the advantage that many ANN models of different sizes and running various learning algorithms can be simulated on the same architecture. Thus, these implementations provide a good balance of speed and flexibility. We have therefore chosen to concentrate on general purpose implementations in this paper. Online current information about special purpose hardware implementations of ANNs can be found at [49].

3 Desired Characteristics

As is apparent from the above discussion, there has been a substantial amount of research done in the area of parallel implementations of ANNs. However, there doesn't yet seem to be an integrated environment that allows a user to choose a particular ANN model from a menu of the most popular ANN models, and implement it on a variety of parallel machines. Such an environment would ease the hard transition that most researchers face when moving from sequential implementations to parallel implementations. What characteristics should such an environment possess? The following are some prescriptions for the work needed to create an environment that allows a user to generate optimized parallel code to implement a particular ANN model on a particular parallel machine:

- **Theoretical analysis of the inherent parallelism**: Before we can create an environment for parallel ANN implementation, a thorough analysis of the parallelism inherent in the ANN models needs to be carried out. Such an analysis will look at each model, and analyze the inherent parallelism at the finest grain. Next, this analysis should investigate which parallel computation models can best exploit the parallelism in each model. The parallel computational and communication complexity of each of the ANN architectures should be computed on the common models of parallel computation such as PRAM [19], BSP [98], C^3 [33], logP [8], L-PRAM [2], H-PRAM [36], etc. This will provide researchers with an idea of the amount of parallelism that can be exploited when each model is implemented on the appropriate parallel machine. This theoretical analysis will also ensure that the parallel implementations generated by the environment will be efficient[1]. As shown in the previous section, a number of steps have already been taken in this direction.

- **Portability**: The turnover in the world of parallel computers is such that the life of a particular parallel machine is generally only a few years. A user who had developed code for one machine a couple of years ago, often has to rewrite it for another machine if the original machine is no longer available. In the right kind of simulation environment, the users should be able to easily update their implementations for a new machine through the environment. One way of achieving this portability is to develop the environment in a modular fashion, where the description of a particular ANN is maintained in a machine independent format, and translation modules are available for each parallel machine. This way, the extinction of a machine will mean the removal of the appropriate translation module, and a new machine will mean that the environment's designers create a new translation module for that machine. All of this development effort can be transparent to the user.

- **Ease of use from the user's perspective**: The environment should provide a graphical user interface (GUI) that allows a user to input information into the system. The user should be able to choose a particular ANN architecture, set appropriate parameters, and choose the target parallel machine, all from the GUI. The environment should then generate optimized code to implement the chosen ANN architecture on the chosen machine. Another important function that the graphical interface should serve is to help visualize the learning and the structure of the network.

- **Access to ANN model description at various levels**: the environment should provide users with access to the description of the ANN at various levels. For instance, the novice user can use the environment purely at the GUI level, choose the ANN model to be implemented, set parameters, and choose the parallel machine at that level. The user with an intermediate level of expertise should be able to access the machine-independent description of the neural model, and modify it directly at that level. Finally, the expert user should be able to access the parallel code directly, and make changes as desired.

An ideal environment for implementing ANNs on parallel machines should therefore be a portable, easy to use environment that allows users to implement a variety of ANN models on a variety of parallel architectures in an efficient manner.

4 Conclusions

The paper surveys the research being performed in the development of environments for the parallel implementation of neural networks. After a discussion of the related literature in this area, we have identified the

[1] Note that for a model of parallel computing to accurately predict the performance on a real machine, the parameters of the model have to capture all the salient features of the real machine. This is not always possible if one wants to work at a high level of abstraction.

desired characteristics that such an environment should possess. These prescriptions should provide guiding directions for researchers working in this area.

5 Acknowledgements

The author would like to thank the anonymous reviewers for their thorough reviews and valuable suggestions.

References

[1] Adaptive solutions inc web page. http://www.asi.com/.

[2] A. Aggarwal, A. K. Chandra, and M. Snir. Communication Complexity of PRAMs. *Theoretical Computer Science*, 71:3–28, 1990.

[3] D. M. Anthony. Reducing connectivity in compression networks. *Neural Network Review*, 1990.

[4] V. C. Barbosa and P. Lima. On the distributed parallel simulation of Hopfield's neural networks. *Software, practice and experience*, 20(10):967–983, Oct 1990.

[5] C. M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon Press, Oxford, 1995.

[6] Brainmaker home page. http://www.calsci.com/home.htm.

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

[8] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *$4^t h$ ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, pages 1–12, 1993.

[9] Y. L. Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[10] C. D'Autrechy, J. Reggia, G. Sutton, and S. Goodall. A general purpose simulation environment for developing connectionist models. *Simulation*, 51(1), July 1988.

[11] E. Deprit. Implementing recurrent back-propagation on the Connection Machine. *Neural Networks*, 2:295–314, 1989.

[12] K. I. Diamantara, D. L. Heine, and I. D. Scherson. Implementation of neural network algorithms on the P^3 parallel associative processor. In *International Conference on Parallel Processing*, volume I, pages 247–250, 1990.

[13] O. Ekeberg, P. Hammarlund, B. Levin, and A. Lansner. Swim: A simulation environment for realistic neural network modeling. In J. Skrzypek, editor, *Neural Network Simulation Environments*, chapter 3. Kluwer Academic Publishers, 1993.

[14] A. El-Amawy and P. Kulasinghe. Algorithmic mapping of feedforward neural networks onto multiple bus systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(2):130–136, February 1997.

[15] C. Ernoult. Performance of backpropagation on a parallel tranputer-based machine. In *Neuro Nimes 88*, pages 311–324, 1988.

[16] H. Ernst, B. Mokry, and Z. Schreter. A transputer based general simulator for connectionist models. In R. Eckmiller, G. Hartmann, and G. Hauske, editors, *Parallel Processing in Neural Systems and Computers*, pages 283–286. North-Holland, Amsterdam, 1990.

[17] C. Eswaran and K. V. Asari. Systolic array implementation of artificial neural networks. *Microprocessors and Microsystems*, 18(8):481–488, 1994.

[18] L. Fortuna, G. Manganaro, and G. Nunnari. Parallel simulation of cellular neural networks. *Computers & Electrical Engineering*, 22(1):61–84, 1996.

[19] S. Fortune and J. Wyllie. Parallelism in Random Access Machines. In *10-th ACM Symposium on Theory of Computing*, pages 114–118, 1978.

[20] G. Frazier. Ariel: A scalable multiprocessor for the simulation of neural networks. *Computer Architecture News*, 18(1):107–114, Mar 1990.

[21] K. Fukushima and S. Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 15(6):455–469, 1982.

[22] R. D. Geller and D. W. Hammerstrom. A VLSI architecture for a neurocomputer using high-order predicates. In *Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pages 153–161, 1987.

[23] J. Ghosh and K. Hwang. Mapping Neural Networks onto Message-Passing Multicomputers. *Journal of Parallel and Distributed Computing*, 6:291–330, 1989.

[24] M. Glesner and W. Pöchmüller. *Neurocomputers–An overview of Neural Networks in VLSI.* Chapman & Hall Neural Computing Series. Chapman and Hall, 1994.

[25] N. Goddard. Rochester Connectionist Simulation Environment. In J. Skrzypek, editor, *Neural Network Simulation Environments*, chapter 10. Kluwer Academic Publishers, 1993.

[26] N. H. Goddard, K. J. Lynne, T. Mintz, and L. Bukys. *The Rochester Connectionist Simulator: User Manual.* University of Rochester, Tech Report 233, 1989.

[27] R. Gorman and T. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[28] K. F. Goser. Implementation of artificial neural networks into hardware: Concepts and limitations. *Mathematics and computers in simulation*, 41(1/2):161–171, Jun 1996.

[29] H. P. Graf and P. deVegar. A CMOS implementation of a neural network model. In P. Losleben, editor, *Advanced Research on VLSI*, pages 351–367. MIT Press, Cambridge, MA, 1987.

[30] K. A. Grajski. Neurocomputing using the MasPar MP-1. Technical Report 90-010, Ford Aerospace, Advanced Dev. Dept., Mail Stop X-22, San Jose, CA 95161-9041, 1990.

[31] K. A. Grajski, G. Chinn, C. Chen, C. Kusymail, and S. Tomboulian. Neural network simulation on the MasPar MP-1 massively parallel processor. In *Proceedings of the INNC, Paris, France*, 1990.

[32] S. N. Gupta, M. Zubair, and C. E. Grosch. Simulation of neural networks on massively parallel computer (DAP-510) using sparse matrix techniques. Technical report, Dept of Computer Science, Old Dominion University, VA 23529-0162, May 1990.

[33] S. E. Hambrusch and A. A. Khokhar. C^3: A parallel model for coarse-grained machines. *Journal of Parallel and Distributed Computing*, 32(2):139–154, Feb 1996.

[34] D. Hammerstrom. A VLSI architecture for high-performance, low-cost, on-chip learning. In *International Joint Conference on Neural Networks*, volume II, pages 537–544, 1990.

[35] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley, 1991.

[36] T. Heywood and S. Ranka. A Practical Hierarchical Model of Parallel Computation: I. The Model. *Journal of Parallel and Distributed Computing*, 16:212–232, 1992.

[37] M. Holler. Intel 80170NX Electrically Trainable Analog Neural Network (ETANN). In *Proc. Int. Joint Conf. on Neural Networks, Washington D.C.*, volume II, page 191, 1989.

[38] M. James and D. Hoang. Design of low-cost, real-time simulation systems for large neural networks. *Journal of Parallel and Distributed Computing*, 14:221–235, 1992.

[39] S. Jones, K. Sammut, and J. Hunter. Learning in linear systolic neural network engines: Analysis and implementation. *IEEE Transactions on Neural Networks*, 5(4):5844, 1994.

[40] H. Kato, H. Yoshizawa, H. Iciki, and K. Asakawa. A parallel neurocomputer architecture towards billion connection updates per second. In *International Joint Conference on Neural Networks IJCNN 90*, volume II, pages 47–50, January 1990.

[41] V. Kumar, S. Shekhar, and M. B. Amin. A scalable parallel formulation of the backpropagation algorithm for hypercubes and related architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4(10):1073–1090, 1994.

[42] H. T. Kung, D. A. Pomerleau, G. L. Gusciora, and D. S. Touretzky. How we got 17 million connections per second. In *International Conference on Neural Networks*, volume 2, pages 143–150, 1988.

[43] S. Y. Kung. Parallel architectures for artificial neural nets. In *International Conference on Systolic Arrays*, pages 163–174, 1988.

[44] S. Y. Kung and J. N. Hwang. A Unified Systolic Architecture for Artificial Neural Nets. *Journal of Parallel and Distributed Computing*, 6:358–387, 1989.

[45] T. Lange. Simulation of heterogeneous neural networks on serial and parallel machines. *Parallel computing*, 14(3):287–303, Aug 1990.

[46] R. Leighton and A. Wieland. The Asprin/Migraines software package. In J. Skrzypek, editor, *Neural Network Simulation Environments*, chapter 11. Kluwer Academic Publishers, 1993.

[47] W.-M. Lin, V. K. Prasanna, and K. W. Przytula. Algorithmic mapping of neural network models onto parallel SIMD machines. *IEEE Transactions on Computers*, 40(12):1390–1401, December 1991.

[48] C. Lindsey and T. Lindblad. Review of hardware neural networks: a user's perspective. In *Proceedings of ELBA94*, 1994. Also available from http://www1.cern.ch/NeuralNets/nnwInHepHard.html.

[49] C. S. Lindsey, B. Denby, and T. Lindblad. Neural networks in hardware web page. http://www1.cern.ch/NeuralNets/nnwInHepHard.html.

[50] K. Margaritis. On the systolic implementation of associative memory artificial neural networks. *Parallel Computing*, 21(5):825–840, May 1995.

[51] K. Margaritis and D. Evans. Systolic implementation of neural networks for searching sets of properties. *Parallel computing*, 18(3):325–334, Mar 1992.

[52] H. McCartor. Back propagation implementation on the Adaptive Solutions CNAPS neurocomputer chip. In R. L. et al., editor, *Advances in Neural Information Processing Systems 3*, pages 1028–1031. Morgan Kaufmann Publishers, 1991. More details about CNAPS can be found at the Adaptive Solutions web site at http://www.asi.com.

[53] E. Mesrobian, J. Skrzypek, A. Lee, and B. Ringer. A simulation environment for computational neuroscience. In J. Skrzypek, editor, *Neural Network Simulation Environments*, chapter 1. Kluwer Academic Publishers, 1993.

[54] M. Misra. *Implementation of Neural Networks on Parallel Architectures.* PhD thesis, Dept. of EE-Systems, University of Southern California, 1992.

[55] M. Misra and V. K. P. Kumar. Efficient VLSI Implementation of Iterative Solutions to Sparse Linear Systems. In J. McCanny, J. McWhirter, and E. S. Jr., editors, *Systolic Array Processors*, pages 52–61. Prentice Hall, 1989. Proceedings of the 3rd Int. Conf. on Systolic Arrays.

[56] M. Misra and V. K. P. Kumar. Massive Memory Organizations for Implementing Neural Networks. In *International Conference on Pattern Recognition*, volume II, pages 259–264, June 1990.

[57] M. Misra and V. K. P. Kumar. Neural network simulation on a Reduced Mesh of Trees organization. In *SPIE/SPSE Symposium on Electronic Imaging*, Feb. 1990.

[58] M. Misra and V. K. P. Kumar. Implementation of Sparse Neural Networks on Fixed Size Arrays. In M. A. Bayoumi, editor, *Parallel Algorithms and Architectures for DSP Applications.* Kluwer Academic Publishers, 1991.

[59] M. Misra and V. K. P. Kumar. Implementation of neural networks on parallel architectures. In B. Souček, editor, *Fast Learning and Invariant Object Recognition.* John Wiley and Sons, Inc., 1992. In print.

[60] M. Misra, D. Nassimi, and V. K. Prasanna. Efficient VLSI implementation of iterative solutions to sparse linear systems. *Parallel Computing*, 19:525–544, 1993.

[61] M. Misra and V. K. Prasanna. Implementation of neural networks on massive memory organizations. *IEEE Transactions on Circuits and Systems*, 39(7):476–480, July 1992.

[62] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, and J. Beer. The RAP: a Ring Array Processor for Layered Network Calculations. In *Proc. of Intl. Conf. on Application Specific Array Processors, Princeton, N.J.*, pages 296–308, 1990.

[63] N. Morgan, J. Beck, P. Kohn, J. Bilmes, E. Allman, and J. Beer. The Ring Array Processor: A multiprocessing peripheral for connectionist applications. *Journal of Parallel and Distributed Computing*, 14:248–259, 1992.

[64] S. M. Müller and B. Gomes. Efficient mapping of randomly sparse neural networks on parallel vector supercomputers. In *Proceedings of the Sixth IEEE Symposium on Parallel and Distributed Processing, Dallas*, October 1994.

[65] U. Muller, A. Gunzinger, and W. Guggenbuhl. Fast neural net simulation with a DSP processor array. *IEEE Transactions on Neural Networks*, 6(1):203–213, Jan 1995.

[66] U. A. Muller, B. Baumle, PeterKohler, and AntonGunzinger. Achieving supercomputer performance for neural net simulation with an array of digital signal processors. *IEEE micro*, 12(5):55–65, Oct 1992.

[67] J. M. J. Murre. Transputers and neural networks: An analysis of implementation constraints and performance. *IEEE Transactions on Neural Networks*, 4(2):284–292, March 1993.

[68] J. M. J. Murre. Neurosimulators. In M. A. Arbib, editor, *Handbook of Brain Research and Neural Networks.* MIT Press, 1995.

[69] Nestor web site. http://www.nestor.com.

[70] Neural Networks: Frequently Asked Questions web page. http://www-leibniz.imag.fr/RESEAUX/osorio/faqs/FAQ.html#questions.

[71] T. Nordström and B. Svensson. Using and designing massively parallel computers for artificial neural networks. *Journal of Parallel and Distributed Computing*, 14:260–285, 1992.

[72] R. C. O'Reilly, C. K. Dawson, and J. L. McClelland. PDP++ users manual. Technical report, Carnegie Mellon University, April 1997. Available from http://www.cnbc.cmu.edu/PDP++/PDP++.html.

[73] M. Pacheco and P. Treleaven. Neural-RISC: A Processor and Parallel Architecture for Neural Networks. In *International Joint Conference on Neural Networks IJCNN 92*, volume II, pages 177–182, November 1992.

[74] T. Poggio and S. Edelman. A neural network that learns to recognize three-dimensional objects. *Nature*, 343(6255):263–266, Jan 18 1990.

[75] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 305–313. Morgan Kaufman, San Mateo, 1988.

[76] K. W. Przytula and V. K. P. Kumar. Algorithmic mapping of neural networks models on parallel SIMD machines. In *International Conference on Application Specific Array Processing*, 1990.

[77] K. W. Przytula, W.-M. Lin, and V. K. P. Kumar. Partitioned implementation of neural networks on mesh connected array processors. In *Workshop on VLSI Signal Processing*, 1990.

[78] K. W. Przytula and V. K. Prasanna, editors. *Digital Parallel Implementations of Neural Networks*. Prentice Hall, Spring 1992.

[79] N. Qian and T. J. Sejnowski. Predicting the secondary structure of globular proteins using neural network models. *Journal of Molecular Biology*, 202:865–884, 1988.

[80] U. Ramacher and J. Beichter. Systolic Architectures for Fast Emulation of Artificial Neural Networks. In J. McCanny, J. McWhirter, and E. S. Jr., editors, *Systolic Array Processors*, pages 277–286. Prentice Hall, 1989. Proceedings of the 3rd Int. Conf. on Systolic Arrays.

[81] U. Ramacher, W. Raab, J. Anlauf, J. Beichter, U. Hachmann, N. Brüls, M. Wesseling, E. Sicheneder, R. Männer, J. Gräss, and A. Wurz. Multiprocessor and memory architecture of the neurocomputer SYNAPSE I. In *Digest ICANN '93*, pages 1034–1039, 1993.

[82] S. Ranka, N. Asokan, R. Shankar, C. K. Mohan, and K. Mehrotra. A neural network simulator on the Connection Machine. In *Fifth IEEE International Symposium on Intelligent Control*, September 1990.

[83] C. R. Rosenberg and G. Blelloch. An implementation of network learning on the Connection Machine. In D. Waltz and J. Feldman, editors, *Connectionist Models and their Implications*. Ablex, Norwood, NJ, 1988.

[84] D. E. Rumelhart, J. L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, Massachusetts, 1986.

[85] P. Sajda, K. Sakai, S.-C. Yen, and L. Finkel. Nexus: A neural simulator for integrating top-down and bottom-up modeling. In J. Skrzypek, editor, *Neural Network Simulation Environments*, chapter 2. Kluwer Academic Publishers, 1993.

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

[87] S. Shams and K. W. Przytula. Mapping of neural networks onto programmable parallel machines. In *International Symposium on Circuits and Systems*, May 1990.

[88] P. Simpson. *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations.* Elmsford Press: Pergamon Press, 1990.

[89] A. Singer. Implementations of artificial neural networks on the Connection Machine. *Parallel Computing*, 14:305–315, 1990. Also appears as Technical Report RL90-2 from Thinking Machines Corporation, MA, USA.

[90] J. Skrzypek, editor. *Neural Network Simulation Environments.* Kluwer Academic Publishers, 1993.

[91] J. Teeters. MDL: A system for fast simulation of large layered neural networks. *Simulation*, 56(6):369–379, 1991.

[92] G. Tesauro and T. J. Sejnowski. A neural network that learns to play backgammon. In D. Z. Anderson, editor, *Neural Information and Processing Systems (Denver 1987)*, pages 442–456. Americal Institute of Physics, New York, 1988.

[93] T. Tollenaere, M. M. V. Hulle, and G. A. Orban. Parellel implementation and capabilities of entropy-driven artificial neural networks. *Journal of Parallel and Distributed Computing*, 14(3):286–305, Mar 1992.

[94] S. Tomboulian. A system for routing arbitrary directed graphs on SIMD architectures. Technical Report ICASE Report No. 87-14, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, March 1987.

[95] S. Tomboulian. Introduction to a system for implementing Neural Net connections on SIMD architectures. Technical Report ICASE No. 88-3, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, January 1988.

[96] P. Treleaven, M. Pacheco, and M. Vellasco. VLSI Architectures for Neural Networks. *IEEE Micro*, pages 8–27, December 1989.

[97] M. Turega. A computer architecture to support neural net simulation. *The Computer Journal*, 35(4):350–360, Aug 1992.

[98] L. G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[99] C. von der Malsburg. Goal and architecture of nueral computers. In Eckmiller, editor, *Neurocomputers.* Springer-Verlag, 1988.

[100] C. von der Malsburg. Pattern recognition by labeled graph matching. *Neural Networks*, 1:141–148, 1988.

[101] B. W. Wah and L.-C. Chu. Efficient mapping of neural networks on multicomputers. In *International Conference on Parallel Processing*, volume I, pages 234–238, 1990.

[102] C.-J. Wang and C.-H. Wu. Parallel simulation of neural networks. *Simulation*, pages 223–232, April 1991.

[103] D. Wang and C. Hsu. SLONN: A simulation language for modeling of neural networks. *Simulation*, 55(2):69–83, Aug 1990.

[104] A. Weitzenfeld and M. Arbib. NSL: Neural Simulation Language. In J. Skrzypek, editor, *Neural Network Simulation Environments*, chapter 4. Kluwer Academic Publishers, 1993.

[105] M. Wilson, U. Bhalla, J. Uhley, and J. Bower. GENESIS: A System for Simulating Neural Networks. In D.S.Touretzky, editor, *Advances in Neural Information Processing Systems I.* Morgan Kaufmann Publishers, 1989.

[106] S. S. Wilson. Neural computing on a one dimensional SIMD array. In *International Joint Conference on Artificial Intelligence*, pages 206–211, 1989.

[107] C.-H. Wu, C.-J. Wang, and S. Sivasundaram. Neural network simulation on shared-memory vector multiprocessors. *Proceedings of Supercomputing 89, Reno NV.*, pages 197–203, 1989.

[108] A. Zell. *Simulation Neuronaler Netze.* Addison Wesley, Germany, 1994. Only available in German.

[109] A. Zell et al. SNNS Manual. SNNS can be retrieved from ftp.informatik.uni-stuttgart.de from /pub/SNNS.

[110] A. Zell, T. Korb, N. Mache, and T. Sommer. Recent developments of the SNNS Neural Network Simulator. In *Proceedings of the Applications of Neural Networks Conference, SPIE*, volume 1294, 1991.

[111] A. Zell, T. Korb, T. Sommer, and R. Bayer. A neural network simulation environment. In *Proceedings of the Applications of Neural Networks Conference, SPIE*, volume 1294, pages 534–544, 1990.

[112] A. Zell, N. Mache, R. Hubner, G. Mamier, M. Vogt, M. Schmalzl, and K.-U. Herrmann. SNNS (Stuttgart Neural Network Simulator). In J. Skrzypek, editor, *Neural Network Simulation Environments*, chapter 9. Kluwer Academic Publishers, 1993.

[113] A. Zell, N. Mache, M. Vogt, and M. Huettel. Problems of massive parallelism in neural network simulation. In *Proceedings of the IEEE Int. Conf. on Neural Networks, San Francisco, CA*, volume III, pages 1890–1895. IEEE Press, March 1993.