ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

Artificial Neural Networks on Massively Parallel Computer Hardware

Udo Seiffert

University of Magdeburg, Germany
Institute of Electronics, Signal Processing and Communications
Magdeburg, Germany
seiffert@iesk.et.uni-magdeburg.de

It seems to be an everlasting discussion. Spending a lot of additional time and extra money to implement a particular algorithm on parallel hardware is often considered as the ultimate solution to all existing time problems for the ones - and the most silly waste of time for the others. In fact, there are many pros and cons, which should be always individually weighted. Besides many specific constraints, in general artificial neural networks are worth to be taken into consideration. This tutorial paper gives a survey and guides those people who are willing to go the way of a parallel implementation utilizing the most recent and accessible parallel computer hardware and software. The paper is rounded off with an extensive reference section.

1 Introduction

The story is almost as old as computers themselves and it sometimes strikes as rather philosophical. As soon as one microprocessor is able to solve a particular problem, people try to make this faster. From a hardware point of view this has led to two major directions - the acceleration of the execution speed of the microprocessor and the parallel application of more than one processor to the problem solution. This has been achieved either by new processor layouts, advanced production technologies, rising clock speeds etc. or the possibility of a parallel execution of instructions. From the user's point of view it is much more straightforward to profit by the first-mentioned developments simply by investing in a faster processor generation. In most cases the original algorithms often even the source code could be easily adapted to the new hardware.
However, to take advantage of any of the different architectures of parallel computers, two conditions have to be met. The considered problem must be in principle able to be processed in parallel and the programme code has to reflect the underlying parallel hardware [1], [2]. In fact, particular problems and thus the derived mathematically formulated algorithms are more or less suitable for parallel processing and furthermore not all parallel computers are equally suitable for a particular problem [3].
In that light artificial neural networks are, depending on their specific characteristic, rather easily viable on parallel hardware of all sorts. This comes from the inherent parallelism of their biological original [4]-[9]. The many implementations support this very impressively [10]-[16]. These days there are three main reasons to implement neural networks on specialised hardware. While some parallel implementations are intended to adapt a technical model as close as possible to its

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

biological original, the second objective becomes evident when dealing with large scale networks and increasing training times. In this case parallel computer hardware can significantly accelerate the training of existing networks or make their realisation viable at all. And finally sometimes a particular hardware, which has not necessarily to be parallel, is essential to meet some requirements of a practical application, such as robustness, size, weight, power consumption, etc.
At this stage the important role of software to speed up a problem's solution should not be underestimated. Every effort has been made to get numerically optimised computer programmes for serial as well as parallel computer systems. Standard single processor software has been ported to parallel versions.
All this shows that the at the beginning formulated question, *parallel implementation or not*, is still open but in continuously changing surroundings of advanced and not just faster available hardware, improved software and last but not least new results coming out of the basic research of neural networks.
This paper focuses on the implementation of artificial neural networks on common and highly accessible parallel computer hardware, mainly PC or workstation clusters and multiple processor machines.

2 Hardware Platforms

2.1 Availability

To come right away to the point, an optimal hardware platform for a particular problem will not be available in most cases or does not exist at all. There are too many different and often also conflicting constraints to describe an optimal system, to say nothing of its availability. Since this is a paper on neural networks we are not going to mention all the different parallel computer architectures systematically sorted into classes by a number of distinguishing marks [17]. We should rather start from the point of availability.
Looking into computer labs of universities and research institutions clearly shows, a potential of parallel and distributed computing is already available. Almost everywhere a number of standalone sequential computers (*PCs*) interconnected by a network (*Ethernet*) can be found. This is the first and simplest step of parallel computing hardware. More and more these computer networks and their components have been originally designed to form a so-called computer cluster (*Beowulf* [18], [19]). And finally sometimes even a multiple processor machine (*Sun*, *HP*, *SGI*), which is usually much more expensive, can be utilized.

2.2 Suitability

Now the question is whether these systems are really suitable to advantageously simulate artificial neural networks. For that purpose let us have a look (Figure 1) at an example of the topology of both supervised trained networks - *Multiple-Layer Perceptrons* (*MLP*) - and unsupervised ones - *Self-Organizing Maps* (*SOM*) to demonstrate basic properties.

Number of nodes. In general it is only possible to handle independent parts in parallel processes. That means only neurons belonging to the same layer can be run in parallel. For example, any neuron of the second hidden layer (MLP) needs the outputs of the first hidden layer but not from other neurons within its own layer. Consequently, neural network topologies with many neurons

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

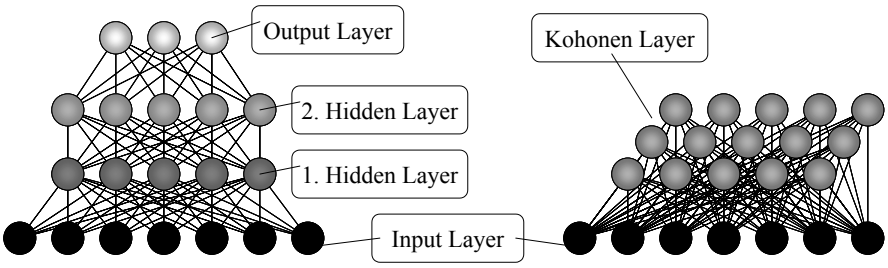

Figure 1: Network topology of a *Multiple-Layer Perceptron* (left) and a *Self-Organizing Map* (right). Each layer is independent and all of its neurons can be processed in parallel.

in one layer (i.e. SOM) seem to be more suitable for parallel processing than those having less neurons in a layered structure, provided that the applied hardware supports the required parallelism [16]. In general the ideal case was to have an equal number of parallel neurons and parallel processors. Commonly there are more neurons than processors, what means that several parallel computations are sequentially performed in a loop.

Communication load. Another very important issue is the communication load and its time distribution of a particular neural network architecture. There are training algorithms (i.e. *Backpropagation*) with extensive communication after each iteration. This sometimes ruins the speed-up reached by parallel computation of neurons. However, this is not necessarily a feature of the neural topology but rather of the training algorithm. For example, a Multiple-Layer Perceptron trained with standard Backpropagation takes less advantage of parallel computing than trained with alternative methods (i.e. *Weight Perturbation* [20], *Genetic Algorithms* [21], [22], *Directed Random Search* [23], [24]). Seen from that angle the network physically connecting the parallel processors is more or less important. It can be characterized by two major parameters - the *data throughput* and the *latency* to establish a connection between two nodes. Often the impact of the second parameter is underestimated, especially when small data packages have to be transmitted very frequently. This restricts the suitability of commonly available *(Fast-)Ethernet* connected computer clusters for a number of neural networks.

Numerical complexity. A third topic should be kept in mind. The available numerical complexity of the applied processors should correspond to the required mathematical complexity of the neural network. Some algorithms require extensive nonlinear computations (Backpropagation) while others are rather simple (SOM). On the other hand, some special hardware, i.e. some *Reduced Instruction Set Computers* (*RISC*) [25] or *Transputers* [26], without or with just reduced *Floating Point Unit* (*FPU*) might not be able to perform all basic mathematical operations, such as multiplication or division [16]. Problems can sometimes be avoided by using similar but numerically simpler substitutions.

Particular Properties. Besides these more general aspects there are also some problem-dependent matters, such as network size, data set, etc. These properties may complicate or facilitate the parallel implementation.

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

2.3 Survey

The issues of the previous subsection are not complete, but may be used to finally judge commonly available general purpose parallel computers, which have *not* been designed to exclusively simulate neural networks. A detailed survey can be found in Table 1.

Table 1: Survey of the suitability of general purpose parallel computers with respect to several demands of artificial neural networks (meaning of the signs: ++ … excellent, + … good, - … poor).

Demand	Heterogeneous Computer Cluster	Beowulf Cluster	Multi-Processor Computer
number of parallel nodes	+	+	-
connecting network • data throughput • latency	- . . . + - . . . +	- . . . + - . . . +	++ ++
numerical complexity	+	++	++
availability in standard computer labs	++	+	-
low capital expenditure	++	+	-
software • availability • operation system independence • platform independence	std. compiler, PVM ++ ++	compiler, PVM, MPI + +	compiler, PVM, MPI - . . . ++ - . . . ++

Summarizing the details given in the table can be seen that Beowulf clusters have no marked weaknesses. They are simply the best price / performance systems available today. These often self-made computer systems range from a few nodes up to several hundred nodes with remarkable performance. Beowulfs are only outperformed by inferiorly parallel and more expensive multiple processor computers relating to the speed of the data transmission between the processors. The less expensive and above all highly available computer clusters are not very suitable for very communication intensive neural networks.

3 Software

3.1 Compiler and Programming

Besides the hardware as basic condition for any parallel implementation, the software has to be considered as well. Parallel programming must take the underlying hardware into account. But first of all the problem has to be divided into independent parts which can later be processed in parallel. Since this requires rather deep understanding of the algorithm, automatic routines to parallelize the problem based on an analysis of data structures and programme loops usually lead only to weak results. Some compilers of common computer languages offer this option. In most cases a manual parallelization still offers more satisfying results. Fortunately neural networks provide originally a certain level of parallelism as already mentioned in section 1. Thus only a mapping of neurons of each layer to the available nodes must be found.

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

Commonly used mathematical or technical computer languages (*C*, *C++*, *Fortran*) are also available on parallel computers, either with specialised compilers or with particular extensions to code instructions controlling the parallel environment. Using a parallelizing compiler makes working not very different from a sequential computer. There are just a number of additional instructions and compiler options. However, compilers that automatically parallelize sequential algorithms are limited in their applicability and often platform or even operating system dependent [27].
Obviously the key to parallel programming is the exchange or distribution of information between the nodes. The ideal method for communicating a parallel programme to a parallel computer should be effective and portable which is often a conflict [28]. A good compromise is the *Message Passing Interface* (*MPI*) [29], [30] that has been originally designed to be used with homogeneous computer clusters (Beowulf), but is available on multi-processor computers as well. It complements standard computer languages with information distribution instructions. Since it is based on C or Fortran and its implementation is pretty effective and available on almost all platforms and operating systems, it has evolved into the probably most frequently used parallel programming language.
In case of a heterogeneous computer cluster a similar system - *Parallel Virtual Machine* (*PVM*) [31], [32] - is widespread and de facto standard. It has been developed to provide an uniform programming environment for computer clusters consisting of different nodes running probably different operating systems, which are considered as one *virtual* parallel computer. Since real parallel computers and homogeneous clusters are a subgroup of heterogeneous clusters, PVM is also available on these systems. Two further parallel programming environments - *Pthreads* [33] and *OpenMP* [34], [35] - are just mentioned for the sake of completeness.

3.2 Administration and Operating System

The operating system on multi-processor computers is usually Unix based and often set by the manufacturer (i.e. *Sun* - *Solaris*). It is in most cases the best choice to take full advantage of all available hardware features.
Nevertheless, universal and hardware independent operating systems can be used as well. *Linux* [36], [37] is the standard for homogeneous computer clusters. Depending on the particular Linux distribution [38], [39], [40] it shares more or less basic concepts, system calls, instruction sets and application programming interfaces. It widely conforms to the *IEEE Portable Operating System Interface* (*POSIX*) standard [41]. After releasing its initial kernel in 1991 it has been developed to an open and platform independent operating system with easy-to-install commercial and non-commercial distributions available and maintained at dozens of internet sites [37].
There are also some systems running *Microsoft Windows NT* (*2000*). Sometimes it might be easier to integrate peripheral components due to a higher availability of device drivers.

4 Simulation Results

4.1 Network and Data Set

As already been mentioned, it is evident that each particular neural network implementation has its own characteristics and even the same network with different parameters or training data set

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

may lead to an altered behaviour on the same parallel hardware. From that point of view gathering universal and above all duplicable simulation results seems to be impossible. However, some simulation results which qualitatively describe and help to illustrate the main topics discussed in the previous two chapters might be of some general interest.
Let us come back to our two example network types, Multiple-Layer Perceptron and Self-Organizing Map (Figure 1). As we will remember, SOMs are particularly predestined to be run on massively parallel hardware, because all neurons are located within the same layer. The SOM considered in this chapter has *64* input neurons and a *5* x *5* (therefore *25* neurons) mapping layer. Of course this network is rather a smaller one and nobody would seriously demand parallel processing, but it is very suitable to illustrate all basic effects which can also be observed running much larger nets.
The training data set has *2790* examples. Its background is a gray-scale image divided into *8* x *8* blocks. The task of the network is to find characteristic block patterns. It is run for *30* epochs (*83700* iterations).

4.2 Utilized Hardware

According to the computer types shown in Table 1 the following parallel hardware was used to obtain simulation results.

Heterogeneous Computer Cluster. It is a rather typical virtual computer cluster built from available single processor machines of diverse architecture running PVM. The network is a mixture of Ethernet and Fast-Ethernet connections. For our investigations all nodes were exclusively available. That means, apart from necessary operating system jobs, no further tasks were running. All nodes have been ordered according to their performance and are utilized beginning with the most powerful ones.

Homogeneous Beowulf Cluster. It is a cluster of *32 Dual-Pentium III* computers with *1.26* GHz clock speed running MPI and Linux. The connecting network is Myrinet [42] with fibre optic cables. Further details can be obtained from the cluster home page [43]. As long as less than *33* processors are required, there are two running modes possible:

- Mode 1: only one processor per node is used. The running task has exclusive access to the Myrinet network;
- Mode 2: both processors of each active node are used and share the Myrinet network.

Multi-Processor Machine. *Symmetric Multi-Processing* (*SMP*) architecture with *64 HP-PA-8700* processors with *750* MHz clock speed running MPI and HP-UX. All nodes are internally connected by a specialised and very fast bus [44].

4.3 SOM Training

The first thing to care for is the mapping of neurons to available processors [16]. In order to demonstrate this we look at Table 2. Obviously the simplest case is to use *25* processors because we have *25* neurons. All other cases simulate the usually occurring situation that we have less processors than neurons. In this case several neurons must be handled by the same processor in a sequential loop. For example, if there is just *1* processor (sequential computer), it must simulate all *25* neurons one after the other. Using *2* processors one of them simulates *13* and the other *12* neurons. While one processor is dealing with neuron *13* the other is not used (idle). So we need *13* sequential loops until the next training example can be processed.

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

Table 2: Distribution of neurons by means of a *5* x *5* SOM and a parallel computer using between *1* and *25* processors. Marked numbers (bold face) indicate the total of necessary sequential loops.

Used Processors	1	2	3	4	5	6	7	8	9	10	11	12	13
neuron distribution among available processors	1x**25**	1x**13** 1x12	1x**9** 2x8	1x**7** 3x6	5x**5**	1x**5** 5x4	4x**4** 3x3	1x**4** 7x3	7x**3** 2x2	5x**3** 5x2	3x**3** 8x2	1x**3** 11x2	12x**2** 1x1
Used Processors	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	
neuron distribution among available processors	11x**2** 3x1	10x**2** 5x1	9x**2** 7x1	8x**2** 9x1	7x**2** 11x1	6x**2** 13x1	5x**2** 15x1	4x**2** 17x1	3x**2** 19x1	2x**2** 21x1	1x**2** 23x1	25x**1**	

The next steps up to *5* processors reduce the total of sequential loops. Spending *6* processors is no use. It just changes the neuron's distribution, because *1* processor must still handle *5* neurons. At first sight one may think the more processors are spent the faster is the network's training. However, this is just one aspect of an optimal mapping which assumes there were no delay exchanging data between the neurons resp. processors. Starting from using *1* processor, where all data is kept locally, the effort and consequently the time to transfer data and provisional results between the processors is increasing step by step. This negatively compensates the speed-up obtained by the parallel processing. Depending on

- the execution time of those parts of the algorithm being processed in parallel in relation to the sequential part;
- the speed of the connecting network, in the first place characterized by bandwidth and latency;
- the amount of data to be transmitted

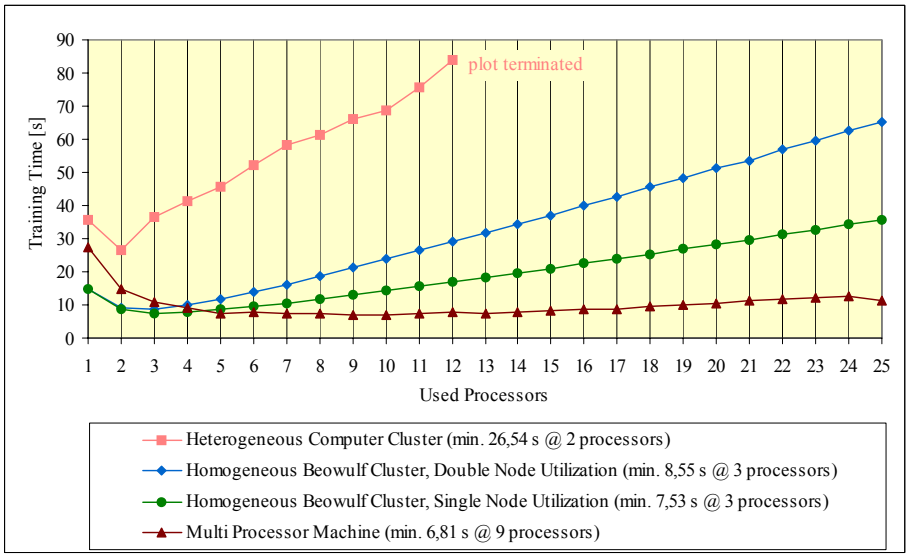

Figure 2: Training time of the example SOM depending on the number of used processors. The minimum value varies between *2* and *9* processors.

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

there is one point where a minimum training time is obtained. As in Figure 2 clearly to be seen, this is by far not at the maximum number of used processors but between *2* (heterogeneous cluster) and *9* (multi-processor machine).
The heterogeneous cluster which has the slowest connecting network shows most significantly the impact of the data transmission, which can not be compensated by the additional computers. In the multi-processor machine always increased parallelism and network load nearly compensate each other. The performance at *5* or more parallel processors remains almost steady.
Although the single processors of the Beowulf cluster are the fastest (*14,83 s* : *27,22 s*), it is finally slightly outperformed by the SMP machine (*7,53 s* : *6,81 s*). However, the cluster needs only *3* processors - where the SMP is still slower (1*0,88 s*) - while the SMP needs three times more processors to reach maximum performance for the sample SOM.
Comparing the two possible modes of the Beowulf cluster shows also the impact of the speed of the connecting network. Both start of course at the position '*1* utilized processor' with equal training time because here is no difference between the modes. If we use both processors of the dual-board computers, the shared connecting network slows down the entire cluster. This impressively demonstrates that a number of processors located on dual-board computers (mode 2) is not equivalent to the same number of independently arranged processors (mode 1).

4.4 MLP Training

As we have already noticed, Multiple-Layer Perceptrons do not offer that high internal parallelism. Furthermore they perform a quite extensive communication between the neurons after each iteration. Of course they can be and are in fact successfully implemented as parallel programmes. The problem which arises when MLPs containing two or more hidden layers are trained by *Back-propagation* is not only the pure training time but also their tiresome feature to get stuck in local minima. Often the only way out is to initialize the weights again and restart the entire training from an altered starting point. Numerous modifications of the original training algorithm have been suggested to avoid this but often this leads to some side effects changing desired features of the neural net.
From the parallel computing point of view there is an elegant way to handle this by running several instances with differently initialised weight sets on parallel processors. All instances themselves are run sequentially. In fact this is not a parallel programme in the true sense but takes enormous advantage from all considered hardware architectures because there is no data exchange between the separately trained nets.
In general this can be done with every type of neural network requiring this or a similar non-interactive procedure. On the other hand, as soon as we are looking for optimal neural network parameters by restarting the training several times with different parameter sets, these instances are not independent any more, since we will usually modify parameters interactively based on the results of previous tests, and thus a real parallel programme would be more advantageous.

4.5 Generalization

The previous section demonstrated main properties of neural network implementations by means of some typical examples and several common parallel hardware. Now the crucial point is to generalize. The message is that the number of parallel processors and the connecting network form an integrated whole. There is always an *extreme* at which there is no further advantage by spend-

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

ing more processors. The position of this point depends on many factors and circumstances, such as

- topology, size and training schedule of the network, more precisely the ratio of possible parallel and necessary sequential execution;
- ratio of computation load and data transmission activity in conjunction with
  - speed of the processors;
  - speed of the connecting network;
  - arrangement of the processors with respect to shared resources;
- compiler, operating system;
- individual properties of the particular problem and the utilized computer system
- …

In those cases where direct parallel programmes are not appropriate parallel computers still offer significant advantages by running the neural net simultaneously with different initial weight sets (as shown in section 4.4) or maybe different parameters. There are also some approaches splitting a large training data set into a number of smaller ones which are independently trained and then the results are assembled.

5 Conclusions

The motivation to deal with parallel programming and parallel computer hardware is evident. It is a pretty challenging prospect to build or at least use a system that solves problems that would have taken days or weeks in hours. The motivation is even more evident when keeping in mind that artificial neural networks offer a high portion of internal parallelism which lifts it out of many other extremely time consuming algorithms. But the best news is yet to come - powerful parallel computer hardware is often more available than expected and not as expensive as feared.

This tutorial paper presents a survey of implementations of artificial neural networks on several massively parallel hardware. In order to focus on rather practical aspects and not to get lost in the great variety of parallel computer systems, the availability of common parallel hardware was chosen as starting point.

Heterogeneous computer clusters are almost everywhere at hand and mark the entrance to parallel computing. Although many neural networks applications, including the examples of this paper, show that these clusters provide only limited speed-up, it is worth to take them into account.

On the other end of largely available parallel computers stand multi-processor machines of all the big brands of workstation manufacturers. They can often be found in computer labs of universities and are not yet too expensive to be bought in a bigger industry or university project.

The most interesting architecture seems to be a Beowulf cluster, preferably equipped with a high speed connection network such as Myrinet. It offers an excellent performance at a very competitive price. This cost advantage often can be as high as an order of magnitude over multi-processor machines of comparable capabilities. And after all they can also be used as development platforms for neural applications that are eventually migrated to faster computer architectures.

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

Acknowledgement

The author would like to thank Bernd Michaelis and Tobias Czauderna for their valuable support.

References

[1] S. Roosta, *Parallel Processing and Parallel Algorithms* (New York: Springer, 1999).
[2] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing* (San Francisco: Benjamin Cummings / Addison Wesley, 2002).
[3] R. Greenlaw, H.J. Hoover, and W.L. Ruzzo, *Limits to Parallel Computation* (Oxford University Press, 1995).
[4] M.A. Arbib, Artificial intelligence and brain theory: Unities and diversities, *Ann. Biomed. Eng.*, **3**, 238-274, 1975.
[5] G.E. Hinton, and J.A. Anderson (Eds.), *Parallel Models of Associative Memory* (Mahwah, NJ: Lawrence Erlbaum, 1981).
[6] D.E. Rumelhart, and J.L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Cambridge, Ma.: The MIT Press / Bradford Book, 1986).
[7] D.H. Ballard, Cortical connections and parallel processing: Structure and function, *Behavioral and Brain Sciences*, **9**, 67-120, 1986.
[8] M.A. Arbib, *Brains, Machines, and Mathematics* (New York: Springer-Verlag, 1987).
[9] W. Maass, and C.M. Bishop (Eds.), *Pulsed Neural Networks* (Cambridge, Ma.: The MIT Press / A Bradford Book, 1999).
[10] K. Obermayer, H. Ritter, and K. Schulten, Large-scale simulation of a self-organizing neural network: Formation of a SOM totopic map, In: R. Eckmiller et al. (Eds), *Parallel Processing in Neural Systems and Computers* (Amsterdam: North-Holland, 1990), 71-74.
[11] P. Kotilainen, J. Saarinen, and K. Kaski, Mapping of SOM neural network algorithms to a general purpose parallel neurocomputer. In: S. Gielen, and B. Kappen (Eds.), *Proc. of International Conference on Artificial Nneural Networks (ICANN '93)* (London: Springer-Verlag, 1993), 1082-1087.
[12] Q.M. Malluhi, M.A. Bayoumi, and T.R.N. Rao, An efficient mapping of multilayer perceptron with Backpropagation ANNs on hypercubes, In: *Proc. of Symposium on Parallel and Distributed Systems (SPDP '93)* (Los Alamitos: IEEE Computer Society Press, 1994), 368-375.
[13] V. Demian, F. Desprez, and H. Paugam-Moisy, and M. Pourzandi, Parallel implementation of RBF neural networks, In: *Proc. of Europar'96 Parallel Processing, Lecture Notes in Computer Science*, Vol. 1124 (Heidelberg: Springer-Verlag, 1996), 243-250.
[14] R.N. Mahapatra, and S. Mahapatra, Mapping of neural network models onto two-dimensional processor arrays, *Parallel Computing*, **22(10)**, 1345-1357, 1996.
[15] T. Hämäläinen, Parallel implementations of Self-Organizing Maps. In: U. Seiffert, and L.C. Jain (Eds.), *Self-Organizing Neural Networks. Recent Advances and Applications* (Heidelberg: Springer-Verlag, 2001), 245-278.
[16] U. Seiffert, and B. Michaelis, Multi-dimensional Self-Organizing Maps on massively parallel hardware, In: N. Allinson et al. (Eds): *Advances in Self-Organising Maps* (London: Springer-Verlag, 2001), 160-166.

[17] D.E. Culler, J.P. Singh, and A. Gupta, *Parallel Computer Architecture* (San Francisco: Morgan Kaufmann, 1998).
[18] T.L. Sterling, J. Salmon, D.J. Becker, and D.F. Savarese, *How to Build a Beowulf - A Guide to the Implementation and Application of PC Clusters* (Cambridge, Ma: The MIT Press, 1999).
[19] *The Beowulf Project On-line*, http://www.beowulf.org.
[20] J. Alspector, and D. Lippe, A study of parallel weight perturbative Gradient Descent, In: *Proc. of Advances in Neural Information Processing Systems (NIPS '96)* (Cambridge, Ma.: The MIT Press, 1996), 803-810.
[21] A.J.F. van Rooij, L.C. Jain, and R.P. Johnson, *Neural Network Training Using Genetic Algorithms* (Singapore: World Scientific, 1996).
[22] U. Seiffert, Multiple-Layer Perceptron training using genetic algorithms, In: *Proc. of the 9th European Symposium on Artificial Neural Networks (ESANN '01)* (Evere: D-Facto, 2001), 159-164.
[23] N. Baba, A new approach for finding the global minimum of error functions of neural networks, *Neural Networks*, **2**, 367-373, 1989.
[24] U. Seiffert, and B. Michaelis, Directed random search for multiple layer perceptron training. In: D.J. Miller et al. (Eds): *Neural Networks for Signal Processing XI*, (Piscataway: IEEE Press, 2001), 193-202.
[25] P.H. Stakem, *Practitioner's Guide to RISC Microprocessor Architecture* (New York: John Wiley & Sons, 1996).
[26] J. Wexler (Ed.), *Developing Transputer Applications* (Amsterdam: IOS Press, 1989).
[27] S. Pande, D.P. Agrawal, and P. Santosh (Eds.), *Compiler Optimizations for Scalable Parallel Systems: Languages, Compilation Techniques, and Run Time Systems* (Heidelberg: Springer-Verlag, 2001).
[28] T. Hey, and J. Ferrante (Eds.), *Portability and Performance for Parallel Processing* (New York: John Wiley & Sons, 1994).
[29] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI - The Complete Reference* (Cambridge, Ma: The MIT Press, 1996).
[30] *The MPI Forum On-line*, http://www.mpi-forum.org.
[31] V. Alexandrov (Ed.), *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science 1497* (Berlin: Springer-Verlag, 1998).
[32] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V.S. Sunderam, *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing* (Cambridge, Ma: The MIT Press, 1995).
[33] B. Nichols, D. Buttlar, and J.P. Farrell, *Pthreads Programming - A POSIX Standard for Better Multiprocessing* (Beijing: O'Reilly, 1998).
[34] R. Chandra, D. Kohr, R. Menon, L. Dagum, D. Maydan, and J. McDonald, *Parallel Programming in OpenMP* (San Francisco: Morgan Kaufmann, 2000).
[35] R. Eigenmann, and M.J. Voss (Eds.), *OpenMP Shared Memory Parallel Programming, Lecture Notes in Computer Science 2104* (Heidelberg: Springer-Verlag, 2001).
[36] E. Siever, J.P. Hekman, S. Spainhour, and S. Figgins, *Linux in a Nutshell*, (Cambridge: O'Reilly UK, 2000).
[37] *Linux On-line*, http://www.linux.org.
[38] *Linux S.u.S.E. Distribution*, http://www.suse.com.
[39] *Linux Red Hat Distribution*, http://www.redhat.com.

ESANN'2002 proceedings - European Symposium on Artificial Neural Networks
Bruges (Belgium), 24-26 April 2002, d-side publi., ISBN 2-930307-02-1, pp. 319-330

[40] *Linux Slackware Distribution*, http://www.slackware.com.
[41] *IEEE Standard for Information Technology - Posix-Based Supercomputing Application Environment Profile* (Piscataway: IEEE Press, 1995).
[42] *Myricom Inc. On-line*, http://www.myricom.com.
[43] *University of Magdeburg, Technical Computer Science Department: Minerva - Beowulf Cluster*, http://iesk.et.uni-magdeburg.de/~minerva.
[44] *Hewlet-Packard Superdome*, http://www.hp.com/products1/servers/scalableservers/superdome/index.html.