CABIOS

Vol. 11 no. 3 1995
Pages 309–315

A parallel neural network simulator on the connection machine CM-5

M.Reczko, A.Hatzigeorgiou, N.Mache[1], A.Zell[1] and S.Suhai

Abstract

We here present a parallel implementation of artificial neural networks on the connection machine CM-5 and compare it with other parallel implementations on SIMD and MIMD architectures. This parallel implementation was developed with the goal of efficiently training large neural networks with huge training pattern sets for applications in molecular biology, in particlar the prediction of coding regions in DNA sequences. The implementation uses training pattern parallelism and makes use of the parallel I/O facilities of the CM-5 and its efficient reduction operations available within the control network to achieve a high scalability. The parallel simulator obtains a maximum speed of 149.25 MCUPS for training feedforward networks with backpropagation on a 512 processor CM-5 system without using the CM-5 vector facility. The implementation poses no restriction on the type of network topology and works with different batch training algorithms like BP, Quickprop and Rprop.

Introduction

The great challenges in computational biology ask for increasingly more powerful hardware resources. In the last years artificial neural networks have been applied successfully in many areas of molecular biology (Qian and Sejnowski, 1988; Bohr *et al.*, 1990; Rost and Sander, 1994; Reczko *et al.*, 1994). The main criterion for the applicability of neural networks is the availability of large data sets generated by complex systems. With the rise of many large-scale genome sequencing projects aiming to capture all the genetic information of different organisms, this criterion asks for large-scale neural network applications. In this paper we describe a parallel implementation of the learning algorithms for training neural networks that are especially suited to processing the large amounts of data encountered in applications such as protein structure prediction or prediction of protein coding regions in DNA, and we compare the training speeds on different hardware platforms for the coding region prediction task. Previous attempts to predict coding regions have employed different combinations of statistical and neural network methods (Lapedes *et al.*, 1990; Brunak *et al.*, 1991; Uberbacher and Mural, 1991; Snyder and Stormo, 1992) with reasonable accuracy. While our initial experiments using feedforward neural networks show the same predictive capabilities as these attempts, the very rapid adaptation of the networks should enable us to employ much more complex and modular network models leading eventually to highly accurate predictions. After a short introduction to artificial neural networks we outline the special architecture of the connection machine CM-5 used in this study and compare the performance in training speed with other parallel simulation environments for neural networks.

Molecular Biophysics Department, German Cancer Research Centre, Im Neuenheimer Feld 280, D-69120 Heidelberg, Germany and [1]Institute for Parallel and Distrib. High Performance Systems (IPVR), University of Stuttgart, Breitwiesenstr. 20-22, D-70565 Stuttgart, Germany

System and methods

Stuttgart Neural Network Simulator (SNNS)

SNNS (Stuttgart neural network simulator) is an efficient universal simulator of neural networks for Unix workstations which was developed at the IPVR, University of Stuttgart. It consists of a simulator kernel, a graphical user interface for the creation, visualization and modification of neural networks, and an optional network compiler.

The simulator kernel maintains the internal representation of the neural networks and performs all operations of the simulation during learning and recall. Its data structures were designed for efficient storage and manipulation of the networks, making the simulator suitable for large neural network training tasks.

The graphical user interface is based on X-Windows X11R5 and displays the topology and the state of the network graphically. It also permits the interactive construction and modification of networks with an integrated network editor.

The network compiler was used to generate the network description file of large, structured networks with a high level programming language. Part of its functionality has now been integrated into the user interface.

SNNS is portable and extensible. The user may link user-defined activation functions, output functions and learning procedures into the existing simulator. All components of SNNS are designed in a modular way

and use detailed open interfaces. Figure 1 shows the structure of SNNS. The following networks models and learning algorithms are currently included in SNNS: backpropagation (different variants), Quickprop, counterpropagation, backpercolation, Rprop, cascade correlation (CC), recurrent CC, dynamic LVQ, time delay networks, RBF networks, ART1, ART2, ARTMAP, BPTT, batch BPTT, QPTT, self-organizing maps, Jordan networks, Elman networks, associative memory.

SNNS is already used for applications in the fields of pattern recognition, classification, diagnosis, forecasting and process control at more than a thousand sites worldwide. Applications of SNNS include character recognition (printed characters, handwritten characters, cursive handwriting), recognition of machine parts and screws, noise reduction in speech understanding systems, stock price prediction, texture recognition, genome sequence analysis, protein structure prediction, EEG signal classification, power management, electron beam stabilization in accelerators and storage rings, robot manipulator control and many more (Figure 2). SNNS is currently available on workstations of Sun (SparcStation 2, 5, 10, 20), DEC (DECStations, Alpha Workstations), IBM (IBM RS 6000), Hewlett-Packard (HP 9000/7xx Series), Silicon Graphics (SGI Indigo, Indigo2, Indy, Crimson, etc.) and Unix PCs (386, 486 and Pentium under Linux).

The sequential version of SNNS is distributed by the University of Stuttgart, IPVR, free of charge for research purposes via anonymous FTP over the Internet from Fileserver ftp.uni-stuttgart.de in /pub/SNNS.

Parallel versions of SNNS exist for a number of SIMD and MIMD computers:

(i) a node parallel and training pattern parallel version on the SIMD computers MasPar MP-1 and MP-2;
(ii) a training pattern parallel version on the MIMD supercomputer Intel Paragon;
(iii) a node parallel version on the neurocomputer Adaptive Solutions CNAPS server II;
(iv) a training pattern parallel version on the MIMD supercomputer Connection Machine CM-5.

In the following the latest addition to the list of parallel simulator kernels, the Connection Machine CM-5 version of SNNS is described in detail.

The Connection Machine CM-5

The hardware platform used for implementation of the training set parallel learning algorithms is the massive parallel computer CM-5 produced by Thinking Machines. This is a scalable MIMD (multiple instructions, multiple data) multiprocessor machine with a powerful communication network (Figure 3). It contains between 16 and 2048 processing nodes equipped with a SPARC processor and 32 megabytes local memory. Special vector units provide a peak performance of 128 MFLOPS per node. The data communication network has a 'fat-tree' topology where initially four nodes mounted physically on one board communicate with a fast local bus. Four groups containing four processors each are then connected with a second bus. A maximum of 4 subsets containing a number

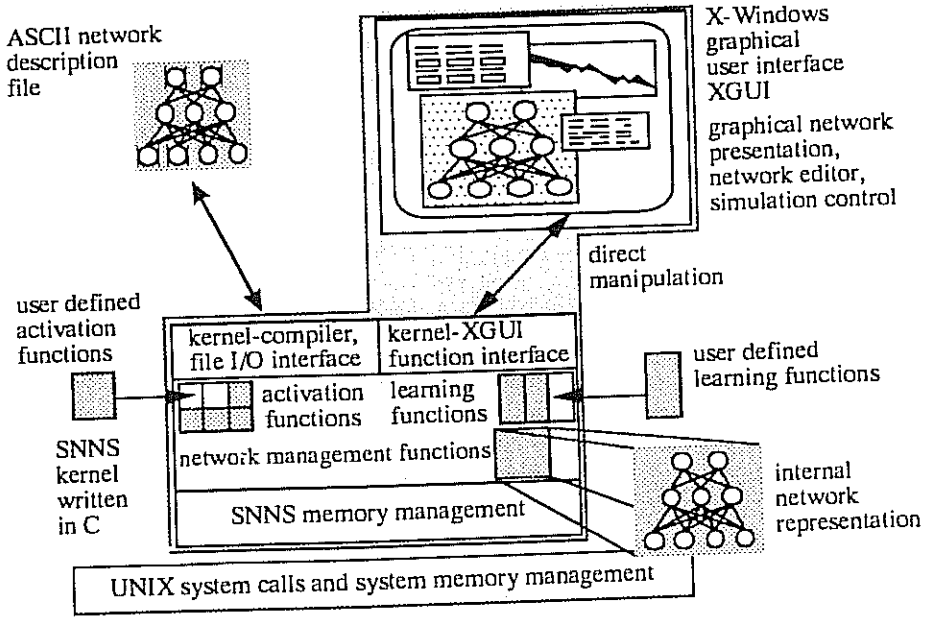

Fig. 1. Structure of SNNS (workstation version).

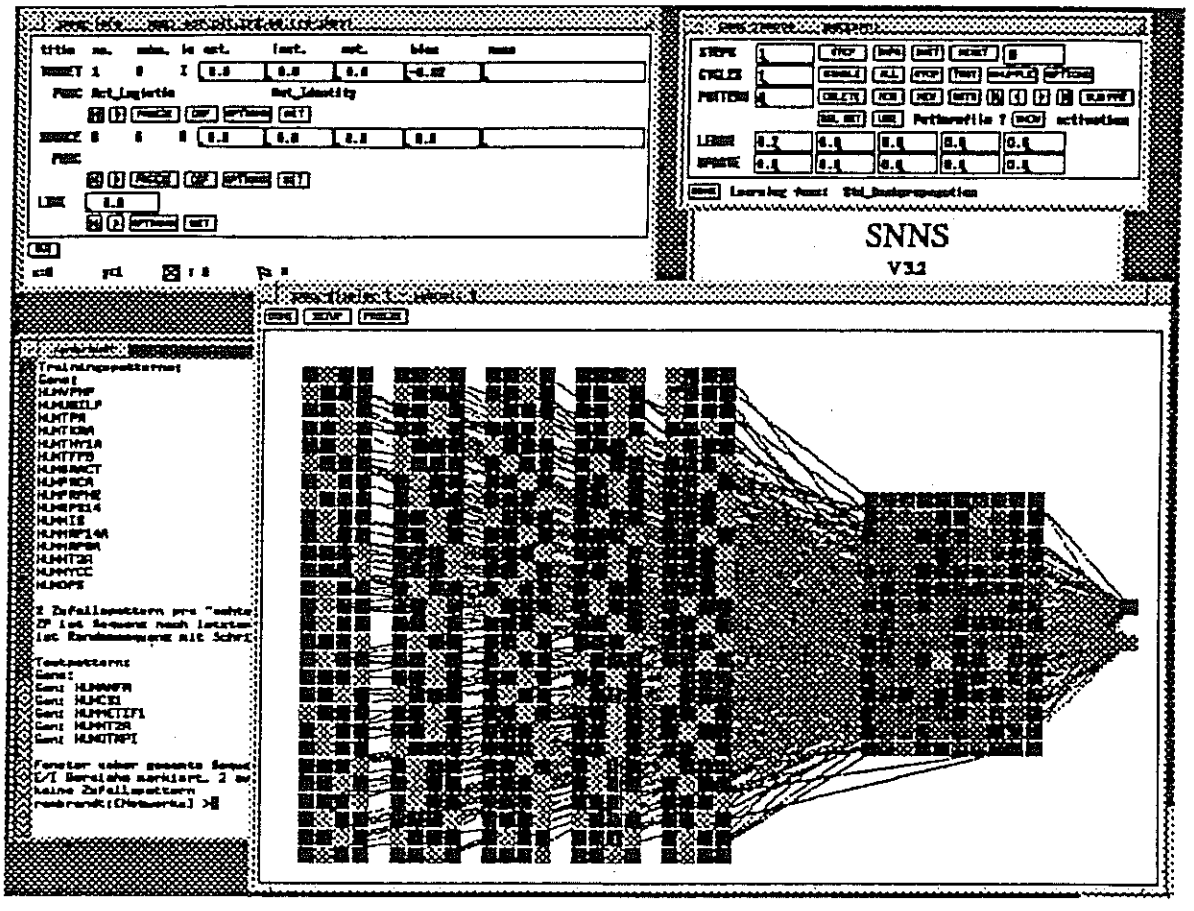

Fig. 2. Graphical user interface of the SNNS simulator, Version 3.2, with a feedforward-network for exon/intron predictions.

of processing nodes corresponding to a power of 4 are always connected to one bus. The bandwidth of the bus increases with each level so that the same data transfer speed is guaranteed between any pair of nodes in the data network. A second control network is used to give physical support for global operations like broadcasts, summations, determination of maxima or minima or boolean operations.

Parallel input and output is supported with a scalable disk array where each processor may access a disk independently. Each processing node runs a UNIX operating system and is accessed via a SUN workstation

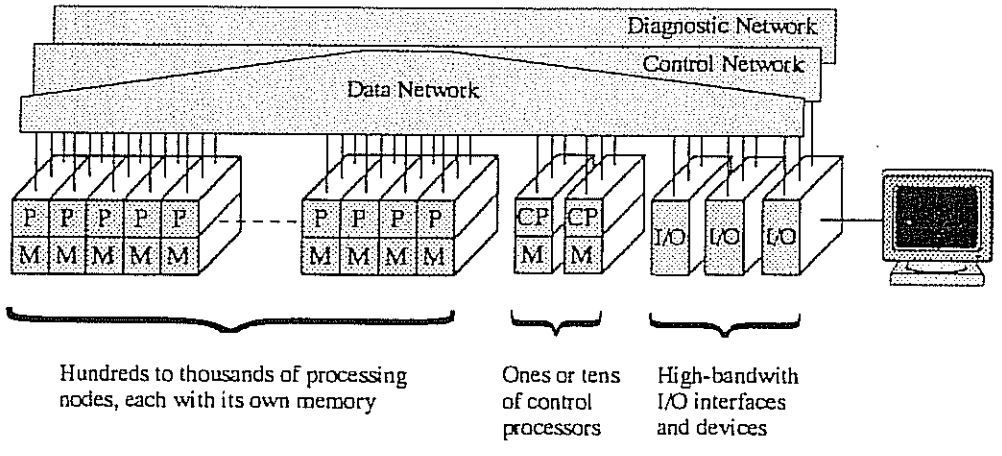

Fig. 3. The structure of the Connection Machine CM-5 with the communication network and its processing nodes.

that manages at least 16 processing nodes in a partition. The version of the CM5 operating system is CMOST 7.3 Final 1 Patch 3 (based on SunOS 4.1.3_U1), the neural network simulator was compiled using gcc version 2.5.8 and the message passing library CMMD 3.2 Final.

Algorithm

Identification of human genes with a neural network

The identification of genes can be derived from the composition of nucleotides on a sequence segment (Singer and Berg, 1991). Our experiments with neural networks indicate that segments with more than 80 nucleotides are required to separate protein coding from the non-coding regions. The training set for the neural network contains segments extracted from 60 human genes with well-known genetic structure taken from the Genbank database. The generalization performance is tested on additional 25 human genes.

Each nucleotide is represented using an activity pattern on four input units (A:0001, C:0010, G:0100, T:1000). Several feed forward networks are trained with different supervised learning algorithms. The number of hidden units is varied between 20 and 150. The output layer (Figure 4) contains one unit representing wether the middle nucleotide of the input segment belongs to an exon or an intron.

The best results are obtained using the Rprop learning algorithm (Riedmiller and Braun, 1993) with 30 hidden units. The input segment contains 81 nucleotides. The best network is tested in four other genes (Genbank entries humalatp, humalgly2, humactga, humalpha) that are not contained in the training or the test set. The successive input segments to the neural network are shifted by one nucleotide over the whole sequence generating a complete exon/intron assignment for a gene; 89.58% of all nucleotides of these genes are classified correctly and 3.83% of all nucleotides are classified into the exon class but belong to the intron class.

Parallel neural network simulation

Several approaches for training neural networks using parallel hardware are reported in the literature (Blelloch and Rosenberg, 1987; Zhang *et al.*, 1989; Singer, 1990; Zell *et al.*, 1993). The methods may be characterized by the complexity of the elements that are processed in parallel. The smallest element is a single synaptic connection that has to simulated using a floating point multiplication. For a general purpose neural network simulator the problem of mapping any network topology on to the communication network of a parallel processing architecture has to be solved in each case if the elements to be processed in parallel are the links or units of a neural network. The largest element that may be processed in parallel is the whole network if there are several patterns that may be propagated independently. In the case of training feedforward neural networks using backpropagation (Rumelhart *et al.*, 1986) the N_{pat} patterns of the training set may be independently propagated forward and backward through the network, if the batch update rule is used for changing the weights. This is the normal update method that uses the average of the gradient of the error

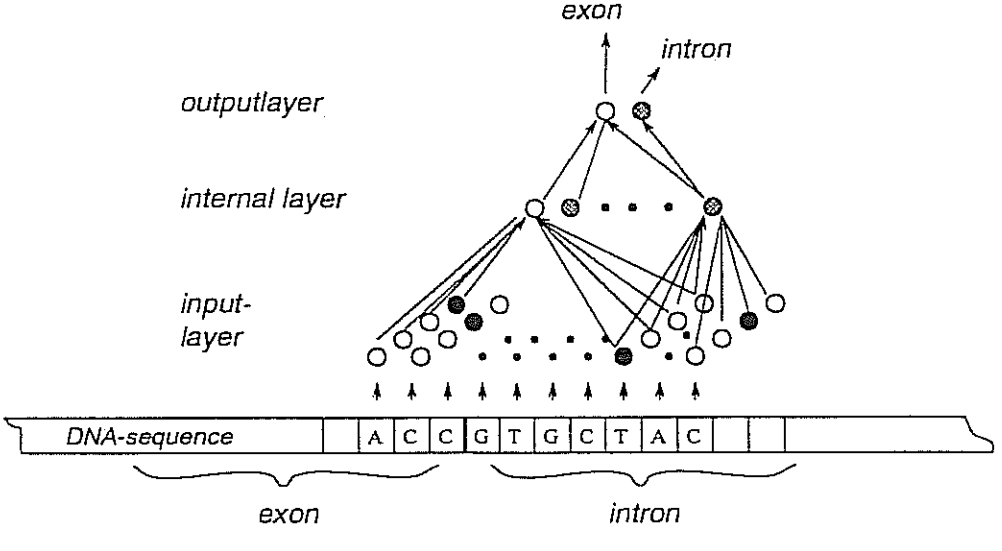

Fig. 4. Recognition of exons with a feedforward neural network. Each nucleotide is represented by 4 neurons. The activation of the output neuron indicates the probability that the central nucleotide in the input sequence belongs to an exon.

function E calculated for each pattern p of the training set for determination of each weight change Δw:

$$\Delta w = -\eta \frac{1}{N_{pat}} \sum_{p=1}^{N_{pat}} \frac{\partial E}{\partial w} \qquad (1)$$

This calculation of the gradient of the can be done in parallel on N_{proc} if the patterns are divided into partitions of N_{pat}/N_{proc} patterns. The sum is then split up according to

$$\Delta w = -\eta 1 N_{pat} \sum_{p=1}^{N_{pat}/N_{proc}} \frac{\partial E}{\partial w} + \sum_{p=(N_{pat}/N_{proc})+1}^{2*(N_{pat}/N_{proc})} \frac{\partial E}{\partial w} + \ldots + \underbrace{\sum_{p=N_{pat}-(N_{pat}/N_{proc})+1}^{N_{pat}} \frac{\partial E}{\partial w}}_{N_{proc}}$$

where each sum is calculated in parallel on a different processor. After each processor has calculated this sum, a global sum has to be determined for each weight over all processors and has to be redistributed to each processor so that the same weight changes will be applied to the weights on each processor. This can be done by using the control network of the CM-5 that supports such global operations with specialized hardware. If the number of patterns is not divisible by the number of processing nodes, the modulo rest of the training set patterns is distributed to the first (N_{pat} mod N_{proc}) processors, causing the rest of the processors to wait at the end of each period for one extra pattern to be processed. The worst case occurs if the first

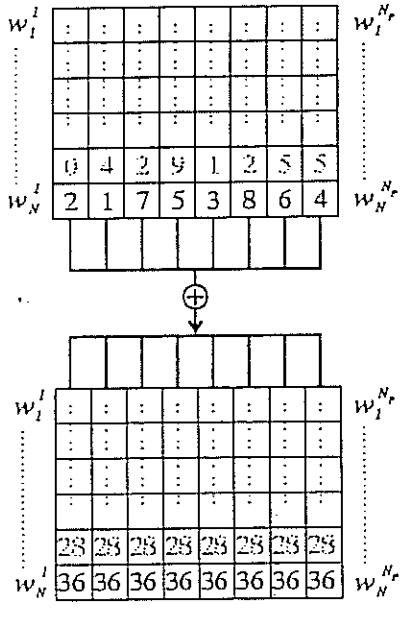

Fig. 5. The reduction function using the summation operator.

processor has one pattern more than all other processors. If the number of patterns is large and the network size is small, the time the other processors are waiting is neglectable. In other cases number of training patterns should be chosen to be divisible by the number of processors.

Implementation

The actual implementation is done using the CMMD message passing library. The time-critical global summation with subsequent broadcast of the result is performed with the function `CMMD_reduce_v` which takes a vector containing $N_{weights}$ real values on each processing node, calculates the sum over all processing nodes for each element in these vectors and replaces each element in the vectors on all nodes by the corresponding sum. The effect of this operation is illustrated in Figure 5. The averaged gradient is then used by each processor to calculate the same weight change for all corresponding weights in parallel. The update procedures implemented include the Quickprop (Fahlman, 1988) and the Rprop (Riedmiller and Braun, 1993) algorithms, which have been shown in many neural network applications to produce networks with good generalization performance using only very few presentations of the whole training set.

In order to achieve fast initialization of the neural network simulator, all input files are read into the memory of the processing nodes in parallel using a special I/O mode supported in the CMMD library. In this global synchronous broadcast (`CMMD_sync_bc`) mode, only one processing node actually reads from an input file and broadcasts each piece of information immediately to all other nodes. In this way the network description file and the training pattern file can be read without any overhead communication usually occurring as the number of processors grows. Before the training patterns are loaded, each processing node calculates the first and last pattern of its fraction of the training set and only allocates the amount of memory that is required to hold this fraction of the training patterns. This reduces the amount of memory used by each processor and provides a significant increase in speed compared to the serial simulation on a single processor in those cases where the training patterns do not fit into the physical memory and where the operating system has to transfer patterns between main memory and a disk used as external memory during access to the training patterns. Since the working set size on each processor is also much smaller, the weight and pattern information can be stored in faster cache memory if the total size of this information is in the range of the 64 kilobyte cache size of the CM-5 processors.

Table I. Peak performance of parallel SNNS simulator kernels for networks of optimal topology

System	MCUPS	Comments
SUN Sparc-10	0.75	Backprop
HP 9000/730	1.09	Backprop
MasPar MP-1216	129.00	Backprop
MasPar MP-2216	360.00	Backprop
Intel Paragon $N_{proc} = 72$	48.83	Backprop
$CM\_5N_{proc} = 64$	21.46	Rprop
$CM\_5N_{proc} = 256$	91.21	Rprop
Adaptive Sol. CNAPS $N_{proc} = 512$	ca. 1400.00	Backprop 16 bit

Results

For a comparison of the CM-5 neural network implementation we first give the peak performance of various parallel SNNS simulator kernels for networks of optimal topology in Table 1. From the peak performance numbers it looks as if other parallel simulator kernels, especially those of the MasPar SIMD machines are considerably faster than the CM-5 implementation. The last line, the training performance of the neurocomputer CNAPS is not really comparable with the other implementations, because it uses special hardware optimized for neural network training and only 16 bit integer weights and 8 bit inputs, whereas the other simulators use floating point arithmetic. For large real-world neural network applications with a large number of training patterns like the one described above the performance for the networks needed for the task, not the optimal networks, is important. Then the situation is quite different, as seen in actual performance measurements for the sequence analysis task given in Table 2.

It can be seen that for the network topology and training pattern sizes demanded by this application the CM-5 compares very favourably with the other parallel systems. It scales nearly linearly with increasing number of processors and is relatively insensitive to the network topology and the number of links of the network. This scaling behaviour at a large number of processors shows the effectiveness of the CM5 communication hardware and software; this behaviour is not normally found in other parallel architectures. It is, however, sensitive to the number of training patterns available, yielding better numbers for larger pattern set sizes. This characteristic makes it very suitable as an experimental platform for large neural network training runs with different topologies.

The data parallel implementations on the MIMD systems CM-5 and Intel Paragon have the additional advantage that irregular network topologies, sparse networks, recurrent networks or networks with receptive fields all can be simulated with the same parallelized simulator kernel. On the other hand, only training algorithms with a 'batch' characteristic of the batch size being a multiple of the number of processors available yield good performance with this method of parallelization.

Discussion

We have presented a parallel implementation of artificial neural networks on the Connection Machine CM-5 developed for large data applications as the task of predicting coding regions in DNA sequences. This application is one of several similar applications in molecular biology which requires the use of extremely powerful hardware and sophisticated parallel neural network implementations to obtain the desired results in prediction accuracy. This application, and problems in the field of molecular biology in general, have the characteristic that large numbers of training patterns can be extracted from genome databases and used for training the networks. This is the necessary prerequisite for using such training pattern parallel neural network implementations like the one we developed for the CM-5 described here. The parallel simulator obtained a training speed of 149 MCUPS for feedforward networks with a faster converging variant of backpropagation on a 512 processor CM-5 system without using the CM-5 vector facility. The

Table II. Performance of various sequential and parallel SNNS kernels for the sequence analysis task, with emphasis on the CM-5 performance. Measurements on the Intel Paragon or the neurocomputer CNAPS were not available for the same task

System	MCUPS				
	4096 patterns		8192 patterns		Comments
	3210 links	28890 links	3210 links	28890 links	
SUN Sparc-10	0.36	0.42	0.32	0.44	RProp
MasPar MP-1216	2.30	16.67	2.42	17.77	Backprop
$CM\_5N_{proc} = 16$	5.99	5.45	6.07	5.45	Rprop
$CM\_5N_{proc} = 32$	11.62	11.12	12.45	11.21	Rprop
$CM\_5N_{proc} = 64$	18.39	21.11	24.18	21.46	Rprop
$CM\_5N_{proc} = 128$	43.75	42.02	46.88	42.13	Rprop
$CM\_5N_{proc} = 256$	82.30	72.73	91.21	80.73	Rprop
$CM\_5N_{proc} = 512$	142.62	116.21	149.25	148.75	Rprop

implementation poses no restriction on the type of network topology and works with different batch training algorithms like BP, Quickprop and Rprop.

In future research the implementation will be extended to efficiently use the vector capability of the CM-5. To do this, in a preceding development a vector kernel of the sequential SNNS simulator has recently been completed, which is a prerequisite for the parallel implementation using the vector facilities of the CM-5. We expect to obtain a further speed improvement by a factor of between 20 and 30 against the current parallel CM-5 implementation with this final parallel and vectorized SNNS simulator kernel on the CM-5.

Acknowledgements

We want to thank the NCSA and the University of Illinois, Urbana-Champaign, Dr H. Bohr at the Center for Biological Sequence Analysis, Technical University of Denmark, and the GMD, Bonn, for support in the form of CM-5 computing time used partly for the generation of the reported benchmark results. This research was supported in part by the BIOINFORMATIK program by the Bundesministerium für Forschung und Technologie, project NEUROGEN, Förderkennzeichen 01 IB 303 A.

References

Blelloch,G. and Rosenberg,C.R. (1987) An implementation of network learning on the Connection Machine. In *IJCAI-87, Proc. of the Tenth Int. Joint Conf. on Artificial Intelligence (Milano, Italy)*. Morgan Kaufmann, San Mateo, CA.

Bohr,H., Bohr,J., Brunak,S., Cotterill,R.M.J., Fredholm,H., Lautrupt,B. and Petersen,S.B. (1990) A novel approach to prediction of the 3-dimensional structures of protein backbones by neural networks. *FEBS Letters*, **261**, 43–46.

Brunak,S., Engelbrecht,J. and Knudsen,S. (1991) Prediction of human mRNA donor by and acceptor sites from the DNA sequence. *J. Mol. Biol.*, **220**, 2–17.

Fahlman, Scott E. (1988) Faster-learning variations on back-propagation: an empirical study. In Hinton,G.E., Sejnowski,T.J. and Touretzky,D.S. (eds), *1988 Connectionist Models Summer School*. Morgan Kaufman, San Mateo, CA, pp.38–51.

Farber,R., Lapedes,A. and Sirotkin,K. (1992) Determination of eukariotic protein coding regions using neural networks and information theory. *J. Mol. Biol.*, **226**, 471–479.

Lapedes,A., Barnes,C., Burks,C., Farberand,R. and Sirotkin,K. (1990) Application of neural networks and other machine learning algorithms toDNA sequence analysis. In *Computers and DNA*, SFI Studies in the Sciences of Complexity, vol. VII. Addison-Wesley.

Qian,N. and Sejnowski,T.J.(1988) Predicting the secondary structure of globular proteins using neural network models. *J. Mol. Biol.*, **202**, 865–884.

Reczko,M. and Bohr,H. (1994) The DEF data base of sequence based protein fold class predictions. *Nucl. Ac. Res.*, **22**, 3616–3619.

Reczko,M., Bohr,H., Subramaniam,S., Pamidighantam,S. and Hatzigeorgiou,A. (1994) Fold class prediction by neural networks. In Bohr,H. and Brunak, S. (eds), *Protein Structure by Distance Analysis*. IOS Press, Amsterdam, pp.277–286.

Riedmiller,M. and Braun,H. (1993) A direct adaptive method for faster backpropagation learning: the RPROP algorithm. In Ruspini,H. (ed.), *Proceedings of the IEEE International Conference on Neural Networks (ICNN 93)*, San Francisco, pp.586–591.

Rost,B. and Sander,C. (1994) *Proteins: Structure, Function, and Genetics*, **19**, 55–72.

Rumelhart,D.E., Hinton,G.E. and Williams,R.J. (1986) Learning internal representations by error propagation. In Rumelhart,D.E. and McClelland,J.L. (eds), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: (*Foundations. MIT Press, Cambridge, MA*.

Singer,M. and Berg,P. (1991) Genes and Genomes. University Science Books, Mill Valley, CA.

Singer,A. (1990) Implementations of artificial neural networks on the Connection Machine. Parallel Computing, **14**, 305–315.

Snyder,E. and Stormo,G. (1992) Identification of coding regions in genomic DNA sequences: an application of dynamic programming and neural networks. *Nucl. Ac. Res.*, **21**, 607–613.

Uberbacher,E.C. and Mural,R. (1991) Locating protein coding regions in human DNA sequences by a multiple sensor-neural network approach. Proc. Natl. Acad. Sci., **88**, 11261–11265.

Zell,A., Mache,N., Vogt,M. and Hüttel,M. (1993) Problems of massive parallelism in neural network simulation. In Proc. IEEE Conf. on Neural Networks, San Francisco, CA, Vol. 3, pp. 1890–1895.

Zhang,X., Mckenna,M., Mesirov,J.P. and Waltz,D.L. (1989) An efficient implementation of the back-propagation algorithm on the Connection Machine CM-2. In Touretzky,D.S. (ed.), Advances in Neural Information Processing Systems 2. Morgan Kaufmann, pp.801–809.

Received on November 22, 1994; revised on March 6, 1995; accepted on March 7, 1995