Proc. IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics (IMACS), Berlin, 1997

SIMULATION OF SPARSE RANDOM NETWORKS ON A CNAPS SIMD NEUROCOMPUTER

PETER PASCHKE

Email: peter.paschke@informatik.tu-ilmenau.de
Dept. of Neuroinformatics, Technical University of Ilmenau,
Gustav-Kirchhoff-Str. 2,
D-98684 Ilmenau, Germany

RALF MÖLLER

Email: moeller@ifi.unizh.ch
AI Lab, Dept. of Computer Science, University of Zürich,
Winterthurerstr. 190,
CH-8057 Zürich, Switzerland

Neurons of the cortical tissue in a mammalian brain are connected in an extremely sparse and random fashion. We use findings of neuroanatomy to model this special connection scheme and present methods for a parallel simulation of such networks on a digital CNAPS neurocomputer. A considerable speedup in comparison to sequential computation is achieved.

1 Introduction

Biologically-inspired sensorimotor systems embedded in complex environments are models for investigating the principles of biological information processing. A real time simulation of neural networks for such systems demands an extremely high computing performance. Although hardware implementations may be much faster, software simulations are needed for the design of neuromorphic models. General purpose parallel computers or special digital or analog hardware speed up the simulations. This paper describes the simulation of special network structures modelling some architectural aspects of the cerebral cortex on a CNAPS system, which is integrated in the mobile robot MILVA at the Department of Neuroinformatics in Ilmenau. Sparse connection structures are very relevant for neuromorphic engineering. Faggin and Mead [1] compared the human central nervous system with VLSI technology and considered the aspect, that the brain is frugal in its use of communication resources.

2 Network Models

The complete (or at least regular) interconnection between or inside groups of neurons is characteristic for the vast majority of artificial neural networks. In real nervous systems, on the contrary, synaptic connections between neurons are often *sparse* — there are few synaptic connections at a neuron compared to the total number of neurons — and show a *random* pattern in some cases. A typical example for this class of biological networks is the cerebral cortex of mammalia.

The connectivity pattern which is the result of a self-organization process in the animal's ontogenesis is probabilistic and can be described by statistics.[2,3] Anatomical studies [4] performed on the cortex of mice reveal an isotropic but random pattern of synaptic connections between pyramidal cells, the main neuron type of the cortex (about 85% of all neurons) which builds excitatory synapses. A pyramidal cell connects to its postsynaptic neighbours with one or only a few synapses. With approximately 8000 synapses counted at a single pyramidal cell compared to around 20 millions of pyramidal neurons, the mean connectivity is only 0.04%. Another group of cells (among the remaining 15% of all neurons) that are considered in many models are inhibitory interneurons which build inhibitory synapses on both main cell types.

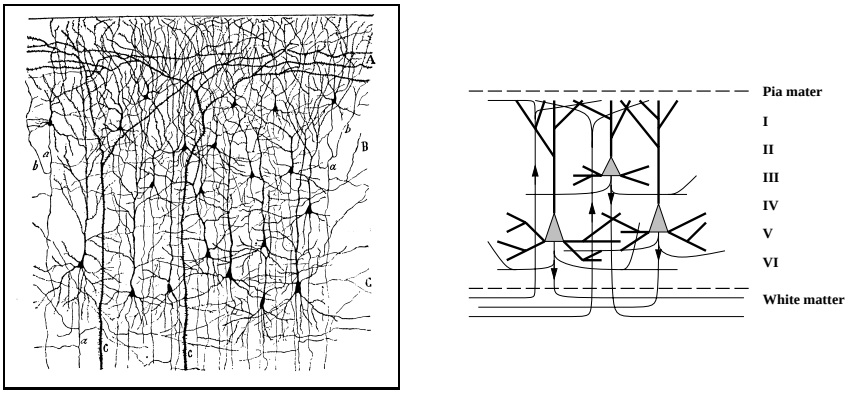

Figure 1: Left: Pyramidal cells of upper layers of the cortex.[5] Right: Symbolic sketch of the A- and B-system of pyramidal cells.

In addition to the characteristics mentioned above, the network models used here consider the main fibre systems of the cerebral cortex, called A- and B-system according to Braitenberg,[4] in a simplified way (figure 1). The A-system is established by long-range axons of pyramidal cells forming synapses

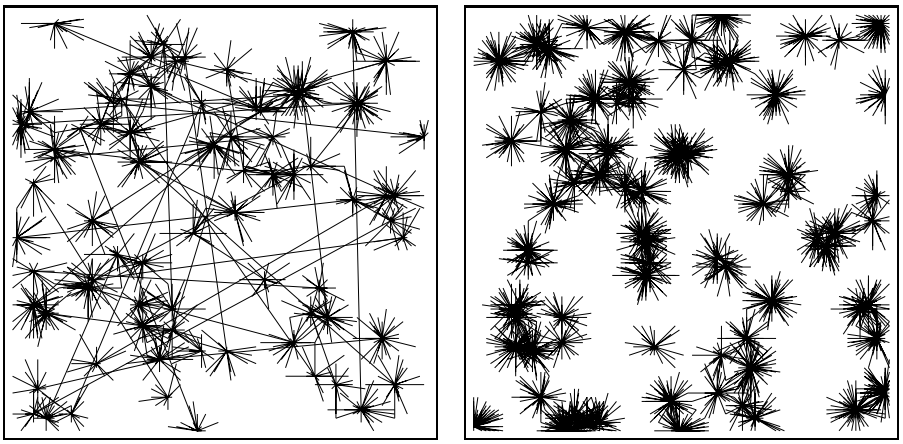

Figure 2: Postsynaptic connections of an example network with a 100×100 grid of neurons. Left: Connections from about 0.5% of the pyramidal neurons. Right: Connections from about 5% of the inhibitory neurons.

at apical dendrites of far pyramidal cells, while the B-system consists of short-range axon collaterals and basal dendrites of neighbouring pyramidal cells. At the end of axons connecting distant regions of the cortex (A-system) there is a local arborization, used to distribute the signals to several thousand synapses.

The special architectural properties of the cerebral cortex lead to assumptions about its information processing. One is that the function of the cortex may be that of an associative memory because there are much more neurons and intracortical connections than input connections.[6]

Although the input connections are necessary for sensorimotor systems, until now our models include only the intracortical connections. Figure 2 shows postsynaptic connections of some randomly chosen neurons out of a 100×100 grid for an example network used to verify the CNAPS implementation. The following generation method of the postsynaptic connections implies that the presynaptic input of a neuron can come from all parts of the neuron grid: the number of postsynaptic neurons q_i of neuron i is a random number from the interval $[q_{\min}, q_{\max}]$. Each pyramidal neuron i is assumed to send out only one long-range fibre, connecting a number $q_i/2$ of neurons selected at random from a circular patch at the reentrance point of the fibre (A-system). The remaining $q_i/2$ neurons are selected in a similar manner from a circular area around that neuron (B-system). Each inhibitory neuron i connects to q_i neurons out of its local circular neighbourhood which has a radius that is smaller than the collateral length of pyramidal cells.[7] Between two neurons there exists at most

one connection. There are 85% pyramidal and 15% inhibitory cells in the model networks.

A simple rate-coded neuron model with a nonlinear function applied to the input scalar product and without learning is currently used, but can easily be extended to more complex dynamic models with learning because the algorithm of the simulation is similar.

3 CNAPS Architecture

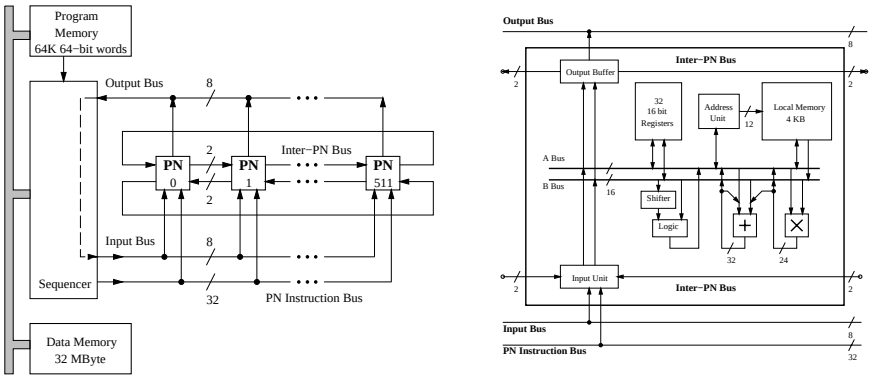

Figure 3: Left: CNAPS architecture. Right: Architecture of a processor node.

The digital CNAPS neurocomputer (Adaptive Solutions, Inc.) has been designed as a general purpose parallel computer for artificial neural networks and image processing tasks and was introduced by Hammerstrom in 1990.[8] CNAPS (Connected Network of Adaptive Processors) consists of a sequencer and a linear array of maximal 512 processor nodes and is a SIMD computer (single instruction stream - multiple data stream). A single instruction bus and three data busses connect the sequencer and the processor nodes: a broadcast input bus (8 bit), a shared output bus (8 bit), and a circular inter-PN bus (2 bit) connecting neighbouring processors (figure 3 left). The sequencer broadcasts an instruction to the processor nodes (single instruction stream), which synchronously execute it, but process the data from their local memories (multiple data stream). If a special flag is reset and a conditioned execution command is used, processing nodes can be excluded from execution. Each processor node (PN) owns an integer arithmetic-logic-unit (ALU) and 4 KByte RAM to store the neuron's states and weights and provide on-chip learning.

Hammerstrom [9] explained the design decisions of CNAPS and suggested some improvements of the architecture, e.g. 16 bit input-, output- and inter-PN-bus, 16×16 bit multiply in one clock cycle, more on-chip memory and higher frequencies, but these improvements were not realized so far.

CNAPS is flexible enough to simulate different network types: artificial neural networks like Multi-Layer-Perceptrons with Backpropagation training,[10] Self-Organizing-Maps,[11,12] Adaptive Resonance Theory,[13] Growing Cell Structures [14] and also biologically inspired pulse-coupled networks.[15] Other applications are described by Hammerstrom.[9] All these network types have a full or regular connection structure. In the case of complete connections between groups of neurons, a single processor emits the activity of one of its neurons to the output bus. This value is transmitted to the input bus and read from all processors containing postsynaptic neurons. There, another activity-weight product is computed and added to the input scalar products of the postsynaptic neurons in parallel on all processors.

Broadcast communication is especially suited for the implementation of fully or regularly interconnected neural networks, but works much less efficient when applied to sparse random networks because of idle processors.[9] Nevertheless, by use of special simulation algorithms and data structures inclusively preparatory optimization procedures there can be still a considerable speedup compared to sequential computation for this field of application, which will be shown in the next sections.

4 Simulation of Sparse Networks on CNAPS

To simulate sparse networks on CNAPS some problems had to be solved: The network structure had to be mapped onto the CNAPS architecture, the variables for states, activities and weights and computation had to use integer arithmetic, and algorithms which efficiently use the SIMD parallel computing had to be developed.

4.1 Efficient Description of the Network Structure

Sparse recurrent networks without regular structure need a special description of their connection pattern: the memory demand for a complete *weight matrix* (stored distributed in the local memory of the processing nodes) is far beyond the memory bounds of the CNAPS processors. Because of the sequential data transmission on the output bus, swapping of local memory would be not efficient.

CNAPS provides a hardware solution for compressed storage of sparse

weight matrices. This special 'virtual zero mode' uses one bit per memory item to decide if a zero item is returned without memory access or if the (non-zero) memory content is read. Even though the memory demand of zero items is reduced to one bit, the total weight memory demand is still proportional to N^2 (N being the number of neurons), so the virtual zero mode becomes more and more inefficient with decreasing mean connectivity of the network.

In these cases, the network structure can be described much more efficiently in a graph-like manner by *successor lists* where, for a given neuron, all postsynaptic neurons are enumerated. That way, the memory demand is reduced to a value proportional to the total number of synapses S with $S \ll N^2$ for extremely sparse networks.

For the CNAPS architecture, the successor lists are not stored at the presynaptic neuron, but are distributed to the processors containing the corresponding postsynaptic neurons. The local parts of the successor lists are consulted by the processors when an activity value is read, in order to determine which local neurons use this value in their computation.

Before generating the CNAPS data structures from a network description file the neurons are mapped onto processors using a greedy optimization algorithm, which balances the demand of computing time and memory between processors.

4.2 Efficient Computation

A problem during SIMD simulation is the efficient usage of the hardware's fine-grained parallelity. The CNAPS simulation works in three phases:

1. Neuron-sequential communication: Activity values of the neurons have to be sequentially distributed over the output bus. This is a disadvantage because the bus architecture prohibits working in parallel. Because of the local memory limit communication and computation (phase 1 and 2) are interlaced and the communication of an activity is delayed until a processor needs this value to avoid idle cycles in the second phase.

2. Synapse-parallel computation: The states (input scalar products) of the neurons are computed in parallel for the synapses. With the help of a ring buffer the activity values of *different* presynaptic neurons can be used in the *same* cycle of computation by different processors.

3. Neuron-parallel computation of the output function: The new activity values result from the neurons' states in a neuron parallel computation. For the nonlinear function a 256 byte lookup table is used. For large networks the processor local memory for the lookup table has to be shared

with the ring buffer and before phase 3 destroyed values have to be restored.

One optimization of SIMD programming is to avoid tests and the conditioned execution. In some cases this can be done, if unused processors work on dummy data structures, which have no influence on the simulation result, instead of switching off the processors. Another possibility is to make the tests in preparation of the parallel simulation and build data structures, which control the sequencer in an appropriate manner. On CNAPS a trade-off between memory and time is necessary — the simulation algorithm has to consider the local memory bound and use buffering to process large networks, which is an overhead in comparison to an algorithm with no memory limits. Another problem is the trade-off between programming effort and simulation speed. We used CNAPS Assembler Language to get maximal speed, but the programming effort was high and the portability to improved architectures is bad. For further discussion of simulation methods and data structures see Paschke and Möller.[16]

4.3 Application of the Simulation

Our network models are discrete dynamic systems and the properties of a network, for instance the development of the mean activity and mean activity changes, depend on the initialization parameters.[2,7,17] Parameters are the spatial distribution of synapses on the axons during network generation (see section 2), ratios of excitation and inhibition and intervals for the random generation of the initial activities and weights.

First experiments with different initializations for the mean activity of the network and ratios between excitatory and inhibitory weights have shown the expected qualitative effects on the development of the mean activity: If the inhibitory weights are too strong the mean activity alternates in every step between high and low values. Otherwise if the inhibition is to weak all neurons go to saturation. With other ratios the mean activity converges although the activities of single neurons change in every simulation step. It remains to investigate, whether it is possible to generalize these results for more complex neuron models.

5 Speedup Results

The simulation algorithm has been implemented using the CNAPS microcode assembler language CPL, because the executables run more efficient than the code generated by the CNAPS-C compiler. Preparation of the data structures

(activities, weights, tables, buffers) for the local processor memories and optimization of neuron-processor assignment is done at a host workstation before the CNAPS simulation.

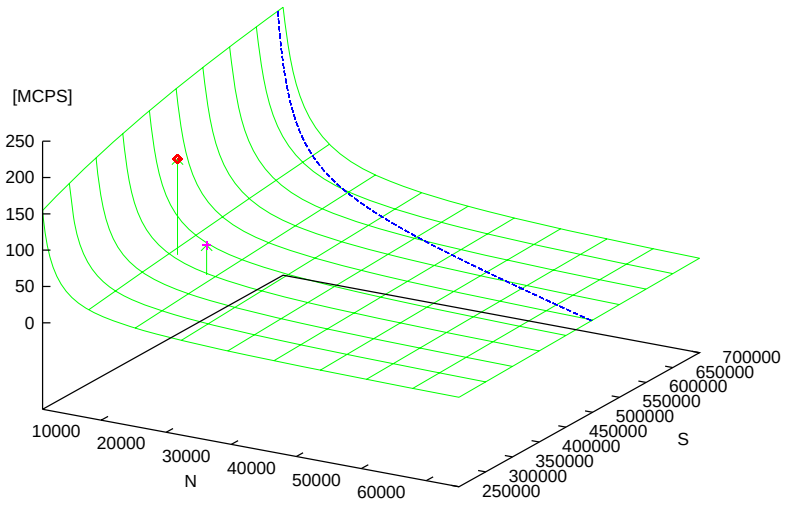

Figure 4: The upper performance bound of the CNAPS simulation depends on the number N of neurons and S of synapses. The broken line marks the memory limit of CNAPS with 512 processors. The symbols $\diamond$ and $+$ show the results really achieved for 2 examples.

Using CNAPS (512 PNs, 20 MHz) the simulation can be speeded up considerably, although the result is far from the theoretical peak performance of 10240 MCPS (million connections per second), because of the loss of parallelity with sparse networks. The performance bound in figure 4 is derived from the lower bound $\check{T}$ of CNAPS' computing time T with P processors, N neurons and S synapses:

$$T \geq \check{T} = T_1 \cdot \left\lceil \frac{N}{P} \right\rceil + T_2 \cdot \left\lceil \frac{S}{P} \right\rceil + T_3 \cdot N + T_4$$

T_1 is the time for a neuron parallel computation step, T_2 for a synapse parallel, T_3 for a neuron sequential and T_4 for other sequential computation steps.

The performance achieved with the CNAPS implementation is demonstrated for a network with $N = 2500$ neurons, $S = 484251$ synapses, $q_{\min} = 97$ and $q_{\max} = 292$. With this network, the simulation runs at 106.9 MCPS on a CNAPS ($\diamond$ in figure 4). For approximate comparison with a sequential computation, a C program using integer arithmetic like CNAPS has been executed on a 200 MHz Sun UltraSPARC workstation for the same network, resulting in 3.5 MCPS. The corresponding speedup is approximately 30.

If the sequential part of the simulation grows the speedup decreases. Therefore the CNAPS simulation for a network with $N = 10000$ neurons, $S = 447436$ synapses, $q_{\min} = 22$ and $q_{\max} = 68$ achieved a speedup of only 8.7 (+ in figure 4). In this case about 90% of simulation time is needed for the sequential part. If the parallel part of the simulation grows the speedup increases. This would be the case for instance if learning rules were introduced, which can be computed in parallel for the synapses.

6 Conclusions

It's worth using findings from neuroanatomy to model the connection structure of neuromorphic systems. Sparse neural network models reduce the number of connections to linearly scale up with increasing neuron numbers. This has some advantage for the memory and time requirements of software simulations and also for space in neuromorphic hardware implementations.

Special methods for the SIMD simulation of sparse networks on the CNAPS neurocomputer were developed. Although CNAPS was optimized for the simulation of regular networks, the architecture is flexible enough to simulate sparse neural networks with a reasonable speedup in comparison to sequential computation. The parallel simulation can be used for further investigations of optimal parameters for sparse random network models.

References

1. Federico Faggin and Carver Mead. VLSI Implementation of Neural Networks. In Steven F. Zornetzer, editor, *An introduction to neural and electronic networks*, pages 297–314. San Diego: Academic Press, 1995.
2. M. Abeles. *Corticonics: neural circuits of the cerebral cortex.* Cambridge University Press, 1991.
3. Almut Schüz. Neuroanatomy in a computational perspective. In M. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*, pages 622–626. The MIT Press Cambridge, Massachusetts, London, England, 1995.

4. Valentin Braitenberg. Cortical architectonics: general and areal. In M. A. B. Brazier and H. Petsche, editors, *Architectonics of the cerebral cortex*, pages 443–465. Raven, 1978.
5. S. R. Cajal. *Histologie du systeme nerveux de l'homme et des vertebretes.* Maloin Paris, 1911.
6. V. Braitenberg and A. Schüz. Cortex - hohe Ordnung oder größtmögliches Durcheinander? *Spektrum der Wissenschaft*, May 1989.
7. Günther Palm. *Neural Assemblies.* Springer, 1982.
8. Dan Hammerstrom. A VLSI Architecture for High-Performance, Low-Cost, On-chip Learning. In *Proc. of International Joint Conference on Neural Networks*, volume 2, pages 537–544, 1990.
9. Dan Hammerstrom. A Digital VLSI Architecture for Real-World Applications. In Steven F. Zornetzer, editor, *An introduction to neural and electronic networks*, pages 335–358. San Diego: Academic Press, 1995.
10. Hal McCartor. Back Propagation Implementation on the Adaptive Solutions CNAPS Neurocomputer Chip. In *Advances in Neural Information Processing Systems*, pages 1028–1031, March 1991.
11. Dan Hammerstrom and Nguyen Nguyen. An Implementation of Kohonen's Self-Organizing Map on the Adaptive Solutions Neurocomputer. In *Artificial Neural Networks*, pages 715–720, 1991.
12. Ville Pulkki and Taneli Harju. An Implementation of the Self-Organizing Map on the CNAPS Neurocomputer. In *Proceedings of ICNN'96*, pages 1345–1349, 1996.
13. C.S. Lindsey and Th. Lindblad. Unsupervised learning with ART on the CNAPS. In *Proc. of 5th Int. Workshop on Software Engineering, AI, and Expert Systems, for High Energy and Nuclear Physics (AIHENP96), Lausanne, Switzerland, Sept. 2-6*, September 1996.
14. Johannes Steffens and Marcel Kunze. Implementation of the Supervised Growing Cell Structure on the CNAPS Neurocomputer. In *Proc. of ICANN'95 Paris*, 1995.
15. Jason M. Kinser and Thomas Lindblad. Implementation of pulse-coupled neural networks in a CNAPS environment. In *Special Issue of the IEEE TNN on PCNN*, (to be published).
16. Peter Paschke and Ralf Möller. Simulation of Sparse Neural Networks on a CNAPS SIMD Neurocomputer. In *Proc. of 15th IMACS World Congress 1997 on Scientific Computation, Modelling and Applied Mathematics, Berlin, Germany*, August 1997.
17. I. E. Dammasch and G. P. Wagner. On the properties of randomly connected McCulloch–Pitts networks: differences between input-constant and input-variant networks. *Cybernetics and Systems*, 15:91–117, 1984.