TOWARDS EFFICIENT HARDWARE FOR SPIKE-PROCESSING NEURAL NETWORKS

Axel Jahnke, Ulrich Roth and Heinrich Klar
TU-Berlin, Institut für Mikroelektronik, Jebensstr. 1, D-10623 Berlin
E-mail: jahnke@mikro.ee.tu-berlin.de

Abstract. We present the requirements for a neurocomputer for spike-processing neural networks. In a simulation study we investigated the performance of available hardware and showed, that there is still a need for a specific neurocomputer dedicated to the simulation of spike-processing networks. On the basis of our simulation study and an investigation of the features of spike-processing networks we analyses the requirements for the design of dedicated hardware. An efficient hardware architecture should contain an event-list module, a sender-oriented connection module and a number of fixed-point processing units.

1 Introduction

Experimental results [1] [2] together with theoretical studies [3] [4] suggest that the time structure of neuronal spike trains is relevant in neuronal signal processing. The synchronized firing of neuronal assemblies could serve as a versatile and general mechanism for feature binding, pattern segmentation and figure/ground separation. This mechanism could also be useful for machine vision, where robust scene segmentation is still a difficult and intricate problem in a real world environment. Various model neurons and network architectures have been presented which allowed the reproduction the essential phenomena of synchronized activity in simulation studies: [5]-[12]. These models belong to two different categories: the basic units are either coupled oscillators or spiking neurons. In the following we will solely concern ourselves with spike-processing neural networks.

Obviously, a serial processing general-purpose computer is not very well suited to the simulation of neural networks. On the other hand, the high acquisition and maintenance costs of high performance parallel computer justify their use only for very few applications. Our first question was whether the available neurocomputers, which reach a very high performance for back-propagation networks, are also appropriate for large networks of spiking neurons. Networks with thousands of neurons are necessary in order to tackle low vision problems such as scene segmentation or to model brain areas. We performed a simulation study on a parallel computer (CNAPS [13]) and investigated on this basis the requirements for an efficient design of a digital accelerator for spike-processing neurons.

In section 2 we describe the specific properties of spiking neurons and the model network we chose for our investigations. Section 3 presents the results of our simulation study. Finally, in section 4 we summarize the requirements for an efficient architecture of an accelerator for spike-processing neural networks.

2 Spike-Processing Neural Networks

2.1 Spiking Neurons

At a spiking neuron incoming spikes $x(t) \in \{0,1\}$ are weighted and induce a time-varying potential u(t) at a synapse which changes according to a response function on a time scale much longer than a single spike. The time course of the response function at a synapse models a postsynaptic potential. This function may be composed of a sharp rise and a following exponential decay or it be may be a solely decaying function. The second one can be easily implemented as a leaky integrator or a first-order recursive digital filter in a discrete version with a relaxation factor $r = \exp((-T) / \tau)$, where T denotes the basic time unit for one simulation step. A combination function accumulates the various potentials yielding the membrane potential $u_m(t)$. A threshold function compares then the membrane potential with the firing threshold to determine whether to emit a spike or not.

An outgoing spike is then transmitted to the connected neurons. Transmission and other delays can be combined to axonal [6] or synaptic delay times [14]. Incoporation of a time-varying firing threshold results in a refractory period for the model neuron. Setting the firing threshold to a infinite value for a fixed time leads to a absolute refractory period. If a outgoing spike is fed back to induce a time-varying threshold potential a relative refractory period is achieved [5]. This enables the model neuron to act as a local nonlinear oscillator.

2.2 Model network and Implementation Issues

As model network for the study of simulation times we chose a enlarged version of the neural network presented in [15]. Our network consists of a two-dimensional layer of up to 256 x128 neurons. Each neuron receives an input signal to its feeding input from its corresponding pixel in the input image. Furthermore each neuron is connected to its eighty nearest neighbors via linking inputs in a square of 9x9 neurons. This type of interconnection is identical for all neurons in the network, except for those close to a border of the layer. So, each neuron owns three time-varying potentials or leaky integrators: *feeding*, *linking* and *threshold*. Moreover, there is a global inhibitory neuron to which all neurons of the layer are connected. If the network is presented an input image with two or more objects it is able to bind together pixels which belong to one object and to separate one object from others.

When simulating spike-processing networks on digital computers, some specific features have to be taken into account. 1.) The time t proceeds in discrete basic time units T. Usually, one basic time unit is chosen as T = 1 ms in analogy to the duration of one action potential. 2.) Let us call the simulation of one basic time step a *time slice*. The *minimal realtime requirement* is then that one time slice has to be computed in less then one millisecond. 3.) Each time slice can be divided into two simulation steps. *Step 1*: The spike receiving neurons have to increment their corresponding potential. *Step 2*: Each neuron has to accumulate its potentials to the membrane potential and to relax the potentials. 4.) Let us call the average number of active neurons, which emit a spike in one time slice, divided by the total number of neurons N the *network activity a*. A low network activity is characteristic for spike-processing networks, because only the neurons representing one object are active at a time. 5.) It has been shown that a very efficient representation of such sparsely-coded signals is the *event-list protocol*, which contains for each time slice the addresses of spiking neurons [16][17]. 6.) One method of representing the connectivity of sparsely connected networks is the use of lists, one for each neuron n_i. The items in the lists are datasets consisting of weights, delays and addresses. The addresses a_j in the list denotes the neuron n_j, to which n_i, sends a spike or from which n_i, receives a spike. The former represents *sender-oriented*, the later *receiver-oriented connectivity* [18]. For low network activity the sender oriented connectivity is significantly faster. 7.) Finally, we can take advantage of the deterministic connection structure of the network. Calculating the connections *on-line* drastically decreases the storage effort.

3 Experimental results

3.1 Resolution analysis

In order to increase the simulation speed and in order to achieve a hardware realization, we wanted to use fixed-point numbers. For our study of the requisite arithmetic precision [19] we used a simplified one-dimensional version of the model network. A value exceeding the maximum representable number, the limitation value, was set to this limitation value. Quantities smaller than the quantization step q (the minimum representable number) were truncated. The results of this study showed that finite resolution has no significant influence on the performance of the network as long as the wordlength does not falls below a certain limit. Going beyond the limit results in a break-down of the network performance. There was no serious problem for any potentials except the threshold potential, when the limitation value was slightly smaller than their peak value. The only potential which must never be limited is the threshold potential, because a limited threshold potential results in a lasting firing of the neuron. The peak value of the various potentials can be computed in advance [19].

Computation with truncation has two effects: weights with smaller magnitude are added (weight quantization) and the values of the potentials are lowered after each multiplication with the relaxation factor r (arithmetic error). Thus, the potentials are decaying faster and arrive at zero, when their value falls below q. The network performance was only slightly impaired as long as the quantization step does not become greater than a certain limit value. We have derived the following upper bound for the requisite quantization step: $q < w_{min} \frac{r^{k_{inf}} - r^{k_{inf}+1}}{2 - r - r^{k_{inf}+1}}$

where w_{min} denotes the minimal weight, which should have an influence at least for the time k_{inf} [19]. Our simulation results with different parameter settings have validated this upper bound for the quantization step and the lower bound for the limitation value. For the model network described in section 2.2 we have calculated the minimal requisite word length for the various potentials. Our result was that 7 to 14 bits are sufficient.

3.2 Performance analysis

First of all, we examined, whether a single PE (processing element) hardware could be sufficient. For implementa-

tion on a conventional workstation we used event-lists and calculated sender-oriented connections for each active neuron. The results (SP-10 in fig. 2) indicated that the processing of the step 2 accounts for the major workload. Furthermore, even very fast single PEs would not achieve the required performance, especially in the case of larger networks. Therefore we have to introduce some kind of parallelism.

Hence we implemented our network on a SIMD parallel computer with bus architecture, the CNAPS/256 with 256 16b PEs and 4KB local memory each [13]. A very efficient mapping of neural networks on parallel computer is such that each PE represents several neurons with their corresponding synapses. This mapping can be called n(euron)-parallel in difference to s(ynapse)-parallel, where the synapses of one neuron are mapped to different PEs. The n-parallel mapping is most efficient for step 2. However, its seems to be disadvantageous for step 1. Only a few neurons and their corresponding PEs are active any given time step [20]. This could be overcome by mapping several neurons to one PE. But this results in more difficulties when representing the sender-oriented connectivity. We have to know for each connection not only the neuron address but also the PE to which the neuron is mapped.

It should be noted, that we have to calculate the connections due to the limited local memory. When connections do not follow any simple deterministic rules, the CNAPS/256 would be limited to networks with less than 500 neurons. However, the results of the investigations could be applied to similar hardware structures with more memory.

Therefore, in our first implementation *CP1* receiver-oriented connections have been calculated. The simplified algorithm for the step 1 is shown in fig. 1a. The results are shown in fig. 2. As one would expect the performance of this algorithm is quite low. Especially for high network activity, the computation on a 256 PE computer takes more time than on a single PE.

Fig. 1: Simplified algorithm for step 1 (S denotes serial code, P parallel code)

S 1 for each PE_j: *S 2 for each active neuron n_a* *S 3 send address* P 1 for each neurons on PE P 2 if connected to n_a P3 update neuron	*S 1 for each PE_j:* *S 2 if active neurons: send p_j* P 1 calculate d = p_j - p *S 3 for each active neurons:* *S 4 send sector number* P 2 update neuron with corresponding s_n
1a) receiver-oriented implementation (CP1)	1b) receiver/sender-oriented algorithm (CP2)

Our second implementation *CP2* uses another approach. The network has been divided in sectors with the size of 256. Each neuron of one sector is mapped to another PE_p, where the number p of a PE denotes the neuron position in the sector. Neurons on a PE are distinguished by their sector number s_n. The algorithm for step 1 is showed in fig. 1b. Note, that it is a combination of receiver-oriented (S1-P1) and sender-oriented connectivity (S3-P2). The results (fig. 2) show the improved performance of CP2. The execution time is nearly independent from network activity and a speed-up of about 15 compared to the serial implementation (with faster PE) has been achieved. However, the mini-

Fig. 2: Computation times (in ms) for time slice vs. network activity a (in percent), N = 16384:
SP-10 : single PE implementation (Sparc-10)
CP-1 : receiver-oriented impl. (CNAPS/256)
CP-2 : sender-oriented impl. (CNAPS?256)

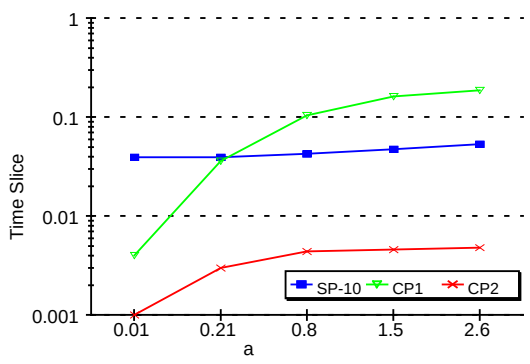
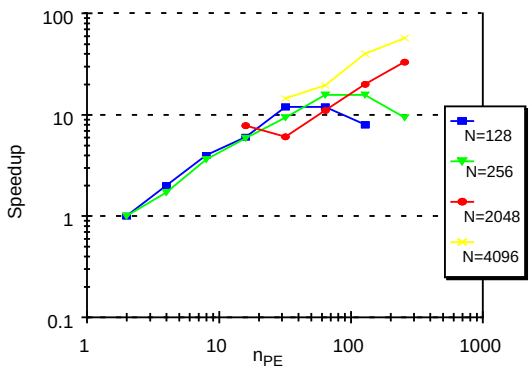


Fig. 3: Parallelization Speedup on a CNAPS vs. n_{PE} (Number of PEs) for different network sizes N
N = 128, 256, 2048, 4096



mal real-time requirements formulated above were not met. Computation times less than the required 1 millisecond per per time slice could eventually be achieved by the use of micro programming.

In order to evaluate the relative speedup through parallelization (independent of the performance of a single PE), we measured computation time of our CP2 while varying the network size N and the number of PEs n_{PE} . Figure 3 shows the behavior of the speedup for N = 256...32K and n_{PE} = 2..256. The results indicate, that the speedup is approximately linear to n_{PE} up to a limit which is reached for less than 4 neurons per PE. The linear speedup is due to the sparse signal coding and the completely parallelizable code for step 2.

4 Conclusion

For an efficient hardware design it is necessary to know the effects of wordlength limitation. The results of our examination indicate that there is no need for floating-point precision and that the minimal requisite wordlength can be estimated in advance.

On the other hand, we have to use some kind of parallel processing in order to meat the realtime requirements. Our simulations indicates, that a parallel computer with bus architecture is well suited to such networks, but only when sender-oriented connectivity is used. Either large memory and/or an efficient on-line calculation for a large class of connections has to be available. Due to low network activity, the connectivity could be represented non-locally to PEs. This would further simplify the representation of sender-oriented connectivity even for reverse dataflow, e.g. for some kind of learning algorithm. Therefore, an efficient hardware architecture should contain an event-list module, some sender-oriented connection modules and a large number of fixed-point processing units.

5 Acknowledgment

We would like to thank R. Eckhorn and G. Hartmann for stimulating discussions, M. Stoecker for providing us the source code of his network and M. Pfister for supporting our work with the CNAPS. Special thanks to L. Bala, who did a lot of the programming work. This work has been supported in part by the Deutsche Forschungsgemeinschaft (DFG) under Grant No. Kl 918/1-1.

6 References

[1] R. Eckhorn, R. Bauer, W. Jordan, M. Brosch, W. Kruse, M. Munk, H. J. Reitboeck, "Coherent oscillations: A mechanism of feature linking in the visual cortex ?" *Biol. Cybern.*, 60: 121-130, 1988.
[2] C. M. Gray, W. Singer, "Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex," *Proc. Natl. Acad. Sci. USA*, 86: 1698-1702, 1989.
[3] C. von der Malsburg , W. Schneider, "A neural cocktail-party processor," *Biol. Cybern.*, 54: 29-40, 1986
[4] F. Crick and C. Koch, "Towards a neurobiological theory of consciousness," *Seminars in The Neuroscience*, 2: 263-275, 1990.
[5] R. Eckhorn, H. J. Reitboeck, M. Arndt, P. Dicke, "Feature linking via stimulus-evoked oscillations: Experimental results from cat visual cortex and functional implication from a network model," *Proc. ICNN*, I: 723-730, 1989.
[6] W. Gerstner, R. Ritz, J. L. van Hemmen, "A biologically motivated and analytically soluble model of collective oscillations in the cortex," *Biol. Cybern.*, 68: 363-374, 1993.
[7] G. Hartmann, "Hierarchical neural representations by synchronized activity: a concept for visual pattern recognition," In: *Neural Network Dynamics*, edited by J. G. Taylor et al., Springer Verlag, Berlin, pp. 356-370, 1992.
[8] D. Horn, M. Usher, "Segmentation and Binding in an oscillatory neural network," *Proc. IJCNN*, II: 243-248, 1991.
[9] D. Kammen, C. Koch and P. J. Holmes, "Collective oscillations in the visual cortex," *Advances in Neural Information Processing Systems*, 2: 76-83, 1990.
[10]P. Koenig, W. Schillen, "Stimulus-dependent assembly formation of oscillatory responses: I. Synchronization," *Neural Computation*, 3(2): 155-166, 1991.
[11]H. Sompolinsky, D. Golomb, D. Kleinfeld, "Global processing of visual stimuli in a neural network of coupled oscillators," *Proc. Natl. Acad. Sci. USA*, 87: 7200-7204, 1990.
[12]O. Sporns, J. A. Gally, G. N. Reeke, G. M. Edelman, "Reentrant signalling among neuronal groups leads to coherency in their oscillatory activity," *Proc. Natl. Acad. Sci. USA*, 86: 7265-7269, 1989.
[13]D. Hammerstrom, "A VLSI Architecture for High-Performance, Low-Cost, On-Chip Learning," *Proc. IJCNN*, 537-543, 1990.
[14]J. R. Beerhold, M. Jansen, R. Eckmiller, "Pulse-Processing Neural Net Hardware with Selectable Topologynand Adaptive Weights and Delays," *Proc. IJCNN*, II: 569-574, 1990.
[15]H. J. Reitboeck, M. Stoecker, C. Hahn, "Object separation in dynamic neural networks," *Proc. ICNN*, II: 638-641, 1993.
[16]J. Lazarro, J. Wawrzynek, "Silicon Auditory Processors as Computer Peripherals", *Advances in Neural Information Processing Systems*, 5: 820-827, 1992.
[17]L. Watts, "Event-Driven Simulation of Networks of Spiking Neurons", *Advances in Neural Information Processing Systems*, 6: 927-934, 1993.
[18]G. Hartmann,1. Workshop zum Förderungsschwerpunkt „Elektronisches Auge“, Summary: 10-19, 1993.
[19]U. Roth, A. Jahnke, H. Klar, "Hardware Requirements for spike-processing neural networks," submitted to IWANN, 1995.
[20]E.Niebur, D. Brettle, "Efficient Simulation of Biological Neural Networks on Massively Parallel Supercomputers with Hypercube Architecture", *Advances in Neural Information Processing Systems*, 6: 904-910, 1993.