Simulation of Spiking Neural Networks - Architectures and Implementations

Martin Schfer [a] Tim Schœnauer [b] Carsten Wolff [a,1]
Georg Hartmann [a] Heinrich Klar [b] Ulrich Rckert [a]

[a] *Heinz Nixdorf Institute, University of Paderborn, 33102 Paderborn, Germany,*
Email: {hartmann,schaefer,wolff}@get.uni-paderborn.de,
rueckert@hni.uni-paderborn.de

[b] *Institute of Microelectronics and Solid State Electronics,*
Technical University of Berlin, Einsteinufer 17, 10587 Berlin, Germany,
Email: {klar,tim}@mikro.ee.tu-berlin.de

Abstract

The fast simulation of large networks of spiking neurons is a major task for the examination of biology-inspired vision systems. Networks of this type label features by synchronization of spikes and there is strong demand to simulate these effects in real world environments. As the calculations for one model neuron are complex, the digital simulation of large networks is not efficient using existing simulation systems. Consequently, it is necessary to develop special simulation techniques. This article introduces a wide range of concepts for the different parts of digital simulator systems for large vision networks and presents accelerators based on these foundations.

Key words: Spiking Neural Networks; PCNN Simulation; Accelerators; Vision Networks

1 Introduction

Within the research area of artificial neural networks (ANN) pulse-coded neural networks are of major interest - especially for pattern recognition purposes. Pulse-coded neural networks (PCNN) are also known as pulse-coupled, pulse-coding or spiking neural networks. In the context of this article these terms are used synonymously for spike-response- and intregate-and-fire-models [21].

[1] Supported by the Deutsche Forschungsgemeinschaft DFG, Me872/4-1

Pulse-coded neural networks are examined for two reasons. Firstly to understand and to reproduce processing in the brain. Secondly the results of this research are for use in technical systems. The pattern recognition capabilities are of interest especially in image processing tasks because PCNNs can produce effects which cannot be achieved by less biology-inspired model neurons. Pulse-coded neurons transfer their activity into pulse- or spike-trains and the exact timing of these spikes can be used to represent features in images. Neurons representing a coherent feature - e.g. a continuous line in their receptive field - synchronize their pulses. Different features are separated with a different phase of the spikes of the neuron groups representing these features. This mechanism is supposed to be advantageous for many perception tasks, e.g. object segmentation [9, 32], and several examinations have shown this behaviour in brains [2, 3, 8]. To understand these effects and to demonstrate their capabilities the real time processing of real world sceneries is applied. This requires the simulation of large networks (several million neurons) at the processing speed of biological systems. The simulation performance of standard workstations is not sufficient for such simulations [15, 26] causing a demand for a special simulator system. This article presents methods and architectures developed at the Institute of Microelectronics at Berlin and at the Heinz Nixdorf Institute at Paderborn. In the following chapters the characteristics of pulse-coded vision networks and the model neuron are presented. The description of a basic simulation algorithm is followed by a chapter about the different techniques for simulation acceleration. Subsequently the integration of learning algorithms into these architectures is dealt with. The different accelerator systems developed in the two working groups are presented. Advantages and disadvantages are discussed and a performance evaluation is given in conclusion.

2 Pulse-Coded Neural Vision Networks

The communication in PCNNs is based on spike exchange. In contrast to conventional model neurons, e.g. McCulloch & Pitts neurons, the generation of a spike requires high computational effort in connection with the time behaviour in the biological example. The computational effort for individual neuron calculations compared to whole network processing is much higher in PCNNs than in conventional ANNs.

Common simulation techniques for neural networks make use of vector representations for the neurons and matrix representations for the connection network [11]. These techniques are not suitable for PCNNs because the actual activity of one neuron is not representable by only one value. Hence, common simulation techniques based on the acceleration of matrix-vector- calculations are not sufficient for PCNNs. A new simulation paradigm is required with

respect to the special characteristics of neural vision networks.

Vision networks are based on retinal sampling of images. Neurons only respond to stimulation in a limited retinal area called the receptive field (RF) of the neuron. Consequently, each neuron type in the network is represented by one neuron for each receptive field [12, 13]. Thus, the neurons are arranged in layers and each neuron in a layer corresponds to an area in the presented image. The neurons connected with neurons in another layer all use similar connection schemes because they all do the same processing - only the receptive field is different. This leads to a systematic network architecture and a regular connection topology. The network is not fully connected but mainly the neighboured layers are connected which causes a sparse connectivity.

Neuron layers extract relevant information from the image and, furthermore, these layers only process special features. Different features can be found in different image areas but the neuron layers contain neurons for the entire image area. Hence, only few neurons in a specialized neuron layer receive input matching their special feature and due to this only few neurons are active while most neurons retain their rest values. Only the active neurons can emit a spike and due to the refractory period only very few spikes are produced in the whole network. In a discrete timeslot simulation this can be defined as a low spike rate. Additionally, the activity in vision networks is controlled by inhibition neurons that inhibit groups of neurons depending on the activity in this group or in another group.

Table 1
Typical features of conventional ANNs compared to PCNNs for vision purposes.

conventional ANNs	PCNNs for vision purposes
simple model neurons	complex model neurons
no considerations of timing effects	modelling of neural timing
continuous activation	activity conversion into spike trains
almost full connectivity	sparse connectivity
all neurons involved	few neurons involved
mainly supervised learning	unsupervised learning

Learning in ANNs usually deals with the modification of connections. In PCNNs, connections represent the axon, the synapse and the dendrite of biological neurons. The connection parameters are subject to temporal changes which are divided - with respect to the time range of these changes - into long term and short term potentiation. Slow connection changes within a long time range are also known as Long Term Memory (LTM) and changes within a short time range are known as Short Term Memory (STM). Neurons are divided into presynaptic and postsynaptic neurons - depending on their po-

sition related to the synapse. The connections are described by the following parameters:

- connection weight
- various delays (axonal, synaptic and dendritic delay)
- type of influence on the postsynaptic neuron (excitation, inhibition)

Several rules derived from the Hebbian learning rule are used concerning long term potentiation. These procedures are based on local data. Only the data from the presynaptic and the postsynaptic neuron is used for modification of the connection and no global network data has to be calculated. The learning procedure is triggered by a spike. Short term potentiation is considered as a further filter function for the synapses.

In conclusion, PCNNs for vision tasks can be characterized by the following features:

- systematic network architecture with neuron layers and receptive fields
- mainly regular and similar connection schemes
- sparse connectivity
- low network activity and low spike rates
- inhibition neurons for activity control
- several learning rules based on local data

Efficient simulation techniques and hardware architectures have to make use of these features.

3 Model Neuron

Pulse-coded neurons transfer their activity - the membrane potential of biological neurons - to a spike train. A threshold operation decides whether the neuron emits a spike or not. The spike is weighted, delayed and transferred via the network topology and used for the modification of the postsynaptic neuron activity. The model neuron is divided into several parts representing the dendrites, the soma, the axon and the synapse of a biological neuron.

The dendrites are commonly modelled with leaky integrators which represent a whole group of similar dendrites of one neuron. The leaky integrators process the spatio-temporal integration of the received spikes for one dendritic tree and they form the dendritic potential (DP) of this tree (see Fig. 1). Other filter functions for the dendritic trees are also possible. The membrane potential (MP) of the neuron is calculated from these dendritic potentials. The dendritic potentials can be excitatory leading to an addition, they can be inhibitory

and then they are subtracted or they can be modulatory and then they are multiplied. The use of a modulatory linking tree is characteristic for Eckhorn neurons [2, 3]. The spike generation is processed from a threshold comparison of the membrane potential (MP) and a dynamic threshold (DT). The dynamic threshold similar to French & Stein [6] is used to reproduce the refractory period of a biological neuron.

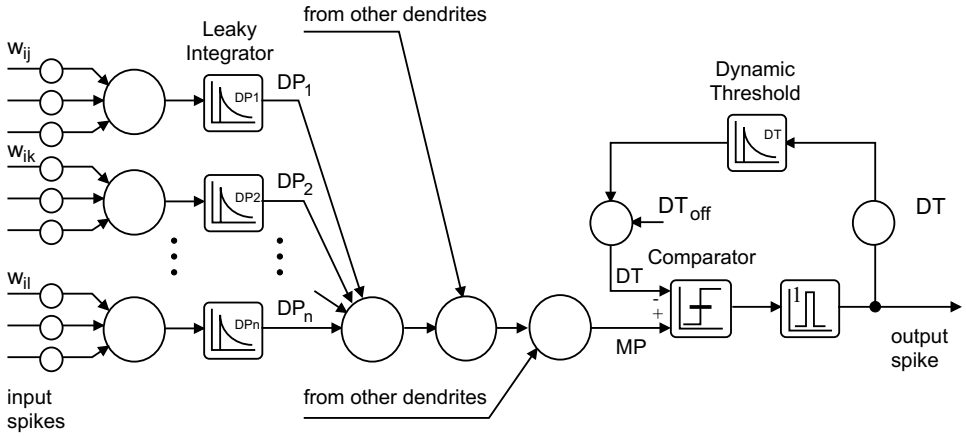

Fig. 1. The pulse-coded model neuron corresponding to Eckhorn [2, 3]

The spikes are weighted via the topology with the connection weight (w) and summed up to give the input of the postsynaptic neuron. The spikes can be delayed with an axonal delay referring to all spikes of one presynaptic neuron or they can be delayed with a dendritic or synaptic delay referring to only one synapse of the neuron. Hence, the axonal delay has to be calculated for the presynaptic spike and the synaptic or dendritic delay has to be calculated for the postsynaptic spike. Both delays and the connection weight may be modifiable.

The following simulation techniques and hardware architectures do not support all features listed above. In all cases a model neuron with at least four dendritic trees with excitatory, inhibitory and modulatory influence, a dynamic threshold and axonal delays is supported. For the leaky integrators at least exponential decay functions are available.

4 Basic Ideas

For digital simulation it is necessary to develop a discrete model close enough to the desired example. Hence, a suitable resolution for the parameters and variables has to be found and a *timeslot simulation* with an appropriate division of the simulation time has to be established. In the case of PCNNs the chosen time division has to guarantee the reproduction of the spike timing with the desired exactness. This exactness depends on the application and is

chosen to one millisecond represented with one timeslot for the vision purposes considered in this contribution. The presented algorithms are of course also suitable for other time divisions.

The main task of the simulation procedure is to calculate at least those network parameters that allow the generation of all spikes occuring in a continuous network processing. Spike emission is derived from the neurons membrane potential and from the dynamic threshold. Hence, the simulation procedure has to process valid values for the membrane potential derived from the dendritic potentials and for the dynamic threshold of all neurons which will emit a spike in the following timeslot. Furthermore, valid connection weights and delays are required. Therefore, the simulation procedure has to provide valid values of the data of all neurons and connections that are possibly involved in the emission of a spike. These parameters can be stored in a memory and the network state is represented by this memory. The simulation is processed using this data.

Common ANNs work with matrix and vector representations of this data. Using the simple neurons and connections the network state can be derived from matrix-vector calculations. PCNNs - in contrast - require the *calculation of individual neurons and synapses* because their complex model neurons are not suitable for matrix-vector representations. Because the large numbers of neurons and synapses cannot be represented individually by their own calculation units, at least partly sequential processing is needed.

A sequential simulation has to guarantee consistent data concerning the time. One possiblity is to calculate the spike emission of all neurons in a first step before these spikes are used in a second step to modify the dendritic potentials of all the postsynaptic neurons. Hence, the timeslot is divided into these two steps. Furthermore, synapses have to be modified before they are used for spike transmission in the following timeslot. In the case of other - e.g. parallel - simulation procedures the use of consistent data concerning time also has to be considered. To provide this consistency, in all cases data has to be stored until one calculation step has been finished for all neurons. Presynaptic spikes are well suited for this storage, because on the one hand they represent the relevant information and on the other hand they form a small amount of information due to the low spike rates in vision PCNNs.

Based on these two simulation phases a trivial algorithm for PCNN simulation can be designed (see Fig. 2). In the first phase the dendritic potentials of the neurons are read from the neuron memory and the membrane potential is calculated. The dendritic potentials are decayed and written to the neuron memory. If the neuron's membrane potential is supraliminal, a spike is produced and collected in a *spike list*. The dynamic threshold is incremented and decayed or - if the neuron is not supraliminal - only decayed. In the second

step of the timeslot the spikes are read from the spike list, weighted (with weight w_{ij}) and distributed via the network topology and used for the modification of the dendritic potentials of the postsynaptic neurons. In the case of learning, the synapses can be modified with respect to the spikes from the spike list after this step. The network is modified to an actual state and the next timeslot can be started. The single steps in one timeslot can be combined or calculated in parallel if the consistency of the data is guaranteed.

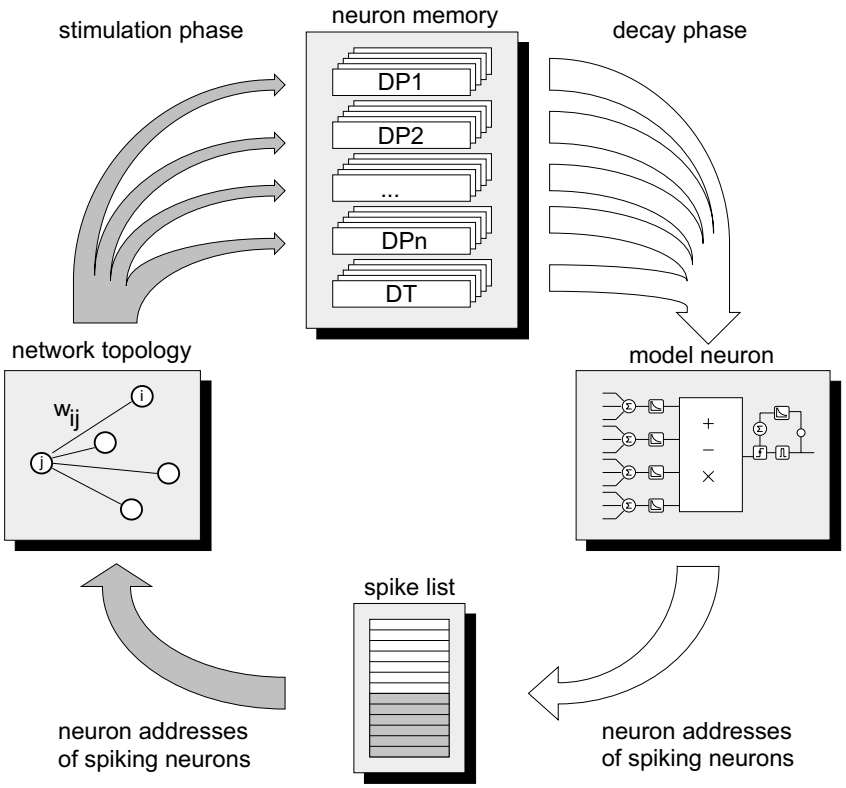

Fig. 2. Basic algorithm for digital PCNN simulation

For this type of network processing, the spike has to carry information about the emitting neuron. The address of the neuron's data in the neuron memory is used as such a label for a spike leading to a simulation based on address events. These addresses are called *neuron address* in the following. In the *decay phase* the neuron addresses of spiking neurons are collected in the spike list buffer and in the *stimulation phase* they are used to address the postsynaptic neurons via the network topology.

The calculation of neurons and synapses can be processed with sequential, parallel or pipelined units. In the case of pipelining, for each calculation step of a neuron or a synapse one unit is provided and these units are arranged in a pipeline which may be fed with one neuron or synapse in each clock cycle. Parallel units can process neuron calculations simultaneously - e.g. decaying of all dendritic potentials in one step or processing of synapses and neurons with parallel units. A further parallel approach is the distribution of neurons

or synapses into several equal processing units [22, 23, 33]. The communication between the units is based on spikes or spike list exchange. As a neuron is connected to other neurons by one axon but many synapses, many postsynaptic spikes are calculated from one presynaptic spike. This leads to a smaller communication amount for systems based on presynaptic spike exchange.

In addition to simulation acceleration by parallelism or pipelining, simulation time can be reduced by exploration of the special characteristics of vision PCNNs (see Chapter 2) if the *number of processed neurons and synapses* can be reduced to those involved in spike generation. Based on the low activity in vision networks and with utilisation of event-driven simulation techniques the calculation amount can be limited to the steps required for a correct spike-train generation. These basic considerations can be transferred to hardware architectures. Some main problem classes are shared by all these architectures and the basic approaches for solving these problems are mentioned below.

- *Calculation steps:* The number of calculation steps can be greatly reduced by utilisation of the special characteristics of vision PCNNs. Combined with techniques like parallelism and pipelining architectures this leads to a collection of methods for an optimized simulation time.
- *Storage capacity:* The problem of limited number and size is especially urgent for the dedicated fast memories of special processing units. Therefore, large but slow shared memories have to be used. For small dedicated memories techniques for efficient usage are presented.
- *Memory bandwidth:* Shared memories commonly suffer from limited bandwidth and access. To solve these problems, methods for the reduction of accesses and transferred data amount are described.
- *Communication:* Especially parallel processing units have to use limited communication resources. Hence, fast communication techniques in the context of PCNNs and procedures for the reduction of communication are required.
- *Load balancing:* Due to the simulation of network parts on several processing units the problem of load balancing occurs. This problem has to be solved in the context of the unknown activity distribution in PCNNs.

5 Concepts for Accelerated Simulation of Large PCNNs

For an accelerated simulation of PCNNs the mentioned problem classes have to be treated. There are two points of view that can be used: the neuron processing time and the synapse processing time. Spikes are the central events of the simulation and therefore the reduction of calculations is based on the following approaches:

- Only the spike emitting and spike receiving neurons are processed.
- Only the neuron parts involved in spike generation are processed.
- Only the spike transmitting synapses are processed.

These approaches can be divided into methods concerning the spike emitting neurons and methods concerning the spike receiving neurons.

5.1 Reducing the Neuron Calculations

Concerning the spike emitting neurons, the neurons that may generate a spike in one timeslot are collected or marked in the preceding timeslot. This can be achieved with a *decay list* [4, 5, 10], where all neurons with dendritic potentials higher than the rest value are collected (*valid potentials*). Only these neurons can become supraliminal and their neuron addresses are stored in the decay list during the decay phase. In the stimulation phase neuron addresses of spike receiving neurons are also stored, as their dendritic potentials are modified. Multiple entries are avoided by using a tag memory. In the following timeslot, only the neurons from the decay list are processed.

Using a very fine time division many neurons in the decay list only perform decay steps during their refractory period or while decaying to their rest values. These neurons are not involved in spike processing and their calculation has to be avoided. They can be dismissed until their refractory period is over and then they have to be resubmitted and to be modified in one decay step. The resubmission has to be processed if the neuron receives input or is able to generate a new spike. Otherwise the neuron can be removed from the *resubmission list* after a given time period [33, 34]. The resubmission time can be calculated from the time when - after an increment caused by a spike emission - the decaying dynamic threshold reaches again the decaying membrane potential of the neuron. This prediction is quite time consuming due to the many components forming the membrane potential. This leads to an approximate calculation by predicting the time when the threshold reaches the old membrane potential of the neuron at the time of dismissal followed by a new resubmission time calculation using the membrane potential at this time. The expense for further resubmission steps is relativated by the fact that a resubmission is required for every spike received by the neuron during the resubmission time.

Calculation effort for a single neuron can be reduced by only processing relevant dendritic potentials (DP) of the neurons during the decay phase. Extended to a *DP-tagging* for the valid neurons this technique can replace the decay list. Instead of processing the whole valid neuron only the valid dendritic potentials of this neuron are processed with respect to a threshold. As in the

decay list and resubmission list algorithm, spike receiving neurons have to be considered supplementary. Even these valid potentials can be irrelevant for a spike emission, e.g. because of a linking potential at a zero value. With a *Pre-Analysis* of the potentials while feeding them into the processing pipeline these potentials can be excluded from calculation. DP-tagging reduces the required memory bandwidth for the neuron state memory because only valid data is read. The simulation time can be drastically reduced if the DP-tagging is combined with Pre-Analysis especially for neurons with many dendritic potentials and a calculation pipeline with only few processing units for the dendritic potentials [30].

The simulation of PCNNs is an IO-bounded problem [26] because many parameters are required for the few processing steps of a single neuron. Especially integrated circuits for PCNN simulation are not only limited from their calculation capability but also from their IO bandwidth. This fact becomes even more crucial with respect to parallel processing. *DP-compression* can be used to exploit a processors bandwidth more efficiently [29]. To achieve this compression, in contrast to the neuron state memory with an entry for each neuron at a dedicated neuron address a neuron memory with entries only for the valid potentials is used. These potentials are stored in a defined order and can be fed continuously into parallel processing pipelines. The calculation of the neuron address of a concrete potential is performed from the distance of a tag bit to the tag bit of a preceding neuron. Hence, the potentials can be read together with the tag-bits from one central memory in a continuous data stream which leads to less required storage capacity and a better load balancing of the processing pipeline.

Furthermore, a reduction of processing effort can be achieved using a special feature of vision PCNNs: the *inhibition neurons*. The inhibition effect of these neurons can be combined with the calculation of the inhibited target neurons by using an increment or decrement directly for the membrane potential or the dynamic thresholds of these target neurons [30]. The inhibition neurons are not required any more which leads to a substantial reduction of calculation amount and especially of communication.

5.2 Reducing the Synapse Calculations

After introducing algorithms for a wide reduction of neuron calculations, in the following methods concerning the synapse and topology calculations are presented. As in the neuron calculations the main task is to reduce the number of synapses that have to be computed to the spike transmitting ones. For synapse considerations there are two points of view, a *sender-oriented* from the spike emitting presynaptic neuron or a *receiver-oriented* view from the

spike receiving postsynaptic neuron. In both cases the synapse computation is triggered by a spiking neuron which leads to calculation of only the spike transmitting synapses. The spiking neurons can be taken from the spike list where they have been stored during the neuron calculations.

In conventional ANNs the network connections are usually stored in a weight matrix [11] where each matrix element represents a possible connection between one neuron marked by the matrix column index and one marked by the matrix row index. This matrix representation can also be used for vision PCNNs but it is inefficient due to the sparse connectivity. A much more efficient way of connection storage is the use of *connection lists* [4, 5, 10] (Fig. 3) similar to the *adjacency lists* commonly used in computer science. There is one list for each neuron containing only the connections to its postsynaptic neurons. The single connection is composed of the neuron address (NA) of the postsynaptic neuron, the connection weight w_{ij}, and some further information. Because a connection list can contain different numbers of connections for different neurons they are stored consecutively and addressed indirectly via a *blockstart-memory (BSM)* with a *blockstart-address (BSA)*.

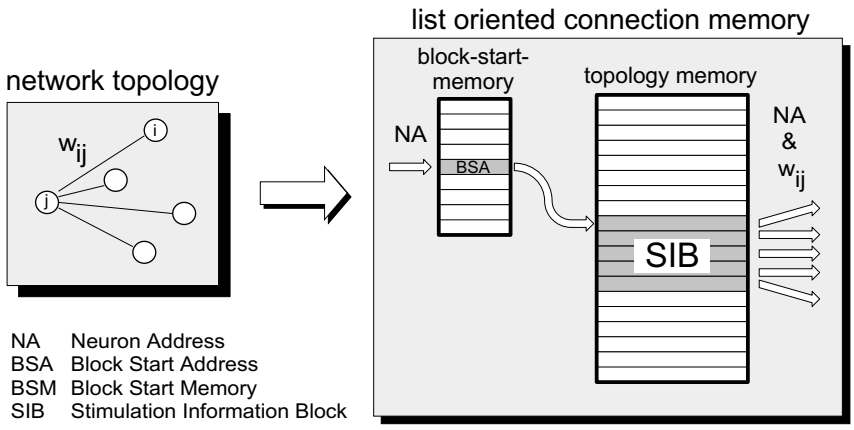

Fig. 3. Network topology representation with connection lists

The connection lists are named as *stimulation information blocks (SIB)*. During the simulation only the SIBs of spiking neurons are addressed and consequently only the spike transmitting synapses from these SIBs are computed. This leads to a low number of calculated connections corresponding to the low spike rates. Although - compared to matrix representation - the storage requirement using SIBs is reduced, the number of connections in vision networks leads to large topology memories. Especially if many highly integrated, parallel processing units are used, the dedicated memories of these units cannot offer the required storage capacity. Hence, different methods for compact topology storage are needed.

This compact topology storage can benefit from the high regularity in vision PCNNs. Whole layers of neurons are connected via *regular connection schemes*

with their target layers [25, 33]. Because all neurons of one source layer share the same connection scheme the method can be named *weight sharing*. The connection scheme can be described with a mathematical function on the coordinates of the target neuron layer and corresponds to the detector masks known from computer vision. With these connection schemes the topology can be stored in a very compact manner and the individual connections are calculated on-line only if they are required. One possibility to store the schemes is to store the parameters of the describing mathematical function [25]. For a sender-oriented topology representation this can lead to the calculation of a high number of possible but not existing connections. Also, only a limited and fixed number of function classes can be implemented. Only if sending and receiving neuron are both known, the connection weight can be calculated faster than with other methods because no search operations are required. A second possibility of regular connection storage is the use of *connection masks* [33] similar to the SIBs. These connection lists do not contain absolute target neuron addresses but relative positions of target neurons and the connection weights. A regular SIB is addressed via a BSM and used for all neurons of a source layer. The absolute target neuron addresses can be processed from the mask and the source neuron address with few calculations. A main disadvantage of regular connection schemes is the missing learning ability. Learning modifies individual connections but the connection schemes contain one virtual connection for many real connections.

The learning ability and the use of highly non-regular connection schemes can only be offered with a SIB representation of all connections. Hence, a large topology memory is required which suffers from communication bandwidth especially if accessed by many parallel units. To overcome this bottleneck caching strategies are useful:

With *SIB-Caching* [33], frequently used SIBs are stored in a cache memory dedicated to the processing unit which processes the presynaptic neuron of the SIB. If the neuron emits a spike and the SIB is stored in the cache, the stimulations can be processed without access of the main topology memory. As spiking neurons will not emit a spike during their refractory period the SIB has to be stored significantly longer than in a standard caching algorithm. A further point to be considered is the consistency of the cache during learning. Modified SIBs have to be exchanged with the old ones stored in the cache. Due to the low activity and high locality in vision tasks - objects in real scenes need many timeslots to move - the hit rate of even a small cache memory is very high.

A further caching approach, *weight caching* [29], reduces the communication between topology memory and processing unit by incrementing the weights for each postsynaptic dendritic potential in a special memory during the timeslot. Hence, for the next neuron calculation only the increments of the dendritic

potentials have to be transferred to the processing unit and the required bandwidth is reduced. A further advantage is the parallel execution of synapse calculations and neuron processing. While a neuron unit is processing the neurons and emitting the spikes, the connection unit can calculate the increments for the dendritic potentials. In the next timeslot the neuron unit reads the dendritic potentials for membrane potential calculation and receives the corresponding increments at the same time. The dendritic potential can be modified and decayed in one step instead of two steps for a separate decay and stimulation phase.

After considering the sender-oriented strategies some special features of a *receiver-oriented* topology storage are mentioned in the following. The storage of connections can be implemented with SIBs or connection masks, but these connections are not stored in the direction from the presynaptic to the postsynaptic neuron but vice versa. The main advantage of this topology representation is the interlocked execution of decaying and stimulating the dendritic potentials because for a neuron processed for decaying all synapses that might stimulate the neuron are known from its SIB. As with weight caching the double access of dendritic potential memory is unnecessary. A further advantage is achieved for several learning procedures which require the state of the presynaptic neuron to modify the connections of a spiking postsynaptic neuron. Because many learning algorithms make use of both sender- and receiver-oriented topology storage a connection memory with an additional pointer memory can be helpful. This shall be dealt with in the following chapter.

Network Feature / Presented Concept	Systematic Network Architecture; Regular Connection Schemes	Sparse Connectivity	Low Network Activity; Low Spike Rates	Inhibition Neurons for Activity Control	Locality of Network Activity
Spike List	+	0	++	-	0
Decay List	+	+	++	--	+
DP Tagging	+	+	++	--	+
Resubmission List	0	0	+++	-	-
Compressed DPs	0	0	++	0	0
Pre-Analysis	0	0	++	0	0
Global Inhibition	++	0	0	+++	0
Sender Oriented Topology	0	+++	0	--	0
Receiver Oriented Topology	0	+++	0	--	0
Regular Connection Calculation	+++	+	0	-	0
Weight Caching	0	+	++	+	+++
SIB Caching	0	+	+	-	++

+++: high benefit for concept; 0: no impact on concept; --- negative impact on concept

Fig. 4. Impact of the diverse network features on the efficiency of the acceleration concepts

6 Integration of Learning Algorithms

Learning in neural networks normally means changing connection weights between the neurons in order to optimize the behavior of the network. Dealing with pulse-coded neural networks, we will concentrate on biologically motivated learning methods. In biology there are several known mechanisms modifying synaptic efficacy leading from short-time effects in the range of milliseconds to lifelong changes of the nervous structure. Building a neuro-computer for pulse-coded neural networks, we have to define a model for synaptic modification that includes the properties we want to simulate. Aiming at some flexibility in simulation approaches, such a model should support at least some biological effects. On the the other hand modeling of learning is restricted to the hardware realization. Data structures or speed of simulation have to be considered. The accelerated calculation of some million neuron parameters has to be compared to the modification of connections which may count some hundredfold. Clearly, problems concerning hardware-architecture must not be the main aspect for the definition of a learning-model, but the approach has to be suitable for a hardware realization.

Focus of interest in simulating biologically motivated neural networks is hebbian learning. Hebbian learning can be seen as a class of learning rules in which correlation of pre- and postsynaptic activity determines synaptic modification. In a general form hebbian learning may be described by four terms:

(1) A connection is strengthened if pre- and postsynaptic neuron are simultaneously *active*.
(2) A connection is weakened if only the presynaptic neuron is *active*.
(3) A connection is weakened if only the postsynaptic neuron is *active*.
(4) A connection remains unchanged if none of the neurons is *active*.

Synaptic modification is determined only by local parameters at the synapse, global parameters as the mean network activity or assembly activities do not influence synaptic strength. There is no *right* or *wrong* network behavior and thus there is no *teacher* who defines synaptic modification. Therefore, hebbian learning is an unsupervied learning method.

Variants of hebbian learning have been developed by various groups. A simple rule for rate-coded neural networks is defined by: $\Delta w_{ij}(t) = (x_i(t) - \bar{x}_i)(x_j(t) - \bar{x}_j)$. Synaptic modification is determined by the correlation of the neural output activities x_i and x_j, which can be interpreted as mean firing rates. Since simulation of pulse-coded neural networks is motivated by the investigation of biological information processing by spikes, this kind of learning rule seems not to be very suitable in this case. Instead we have to look for more biologically motivated learning rules.

For a hardware-realization of pulse-coded neural networks neurophysical models of long-term-potentiation (LTP) and long-term-depression (LTD) are a good choice for learning methods. A brief functional description of LTP and LTD is that the efficacy of an active synapse is increased if the postsynaptic membrane potential is depolarised and it is decreased if not. The first case is named LTP, the second LTD. Both cases can be compared to the terms 1) and 2) in the description of hebbian learning above. Term 3) is not part of this behavior because synaptic modification is only observed at an active synapse.

In order to reach a flexible simulation environment, it is a good idea to add term 3) to the learning-model, even if it does not fit to the known experimental results of LTP. One reason for this is that hebbian learning with rate-coded neurons was shown to be able to simulate biology-inspired self-organization effects [1, 18, 19, 20]. Another fact is, that in biology synapses that do not affect postsynaptic activity vanish on long time scale.

Two important results can be observed on taking a more detailed look at the experiments concerning LTP:

(1) There is a kind of threshold in postsynaptic depolarisation which activates either LTP or LTD.
(2) Influence of backpropagated postsynaptic spikes form a learning-behavior depending on the time-difference between pre- and postsynaptic spikes.

Both results lead to two learning rules that are very interesting for biologically motivated simulation of pulse-coded neural networks.

6.1 Threshold-Based Learning

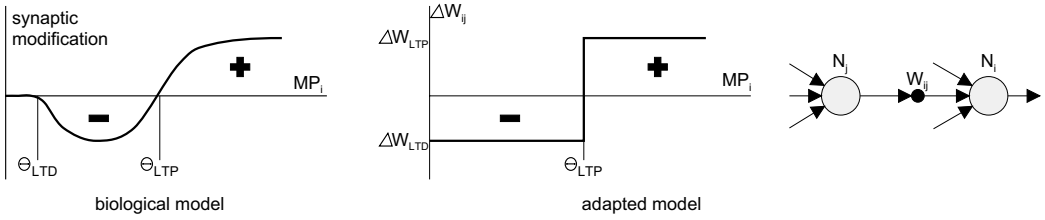

Fig. 5. Threshold based learning rule (synaptic modification Δw_{ij} versus postsynaptic membrane potential MP_i)

The modification of synaptic efficacy depends on depolarisation of the postsynaptic membrane potential. Between the thresholds Θ_{LTD} and Θ_{LTP} the synaptic efficacy is decreased. If the postsynaptic membrane potential MP of an active synapse lies within this interval, LTD takes place [31]. If MP exceeds Θ_{LTP} the synapse is strengthened and LTP occurs. Compared to the model of hebbian learning, the transmission of a presynaptic spike defines the

presynaptic *activity*. Postsynaptic *activity* is given by the value of MP.

Since this model describes synaptic modification for an active synapse, the emission of a presynaptic spike can operate as a trigger for a learning event. Synaptic strength has to be updated only if a presynaptic spike occurs. This fact satisfies the requirement for an efficient simulation, since a neural spike is rather seldom. Thus, only a small part of all synapses have to be processed during one simulation step.

A further simplification with regard to a hardware realization can be reached if Θ_{LTD} is set to zero. The resulting learning process can be described by the following algorithm:

if presynaptic neuron j emits a spike
  if $MP_i > \Theta_{LTP}$
    increase w_{ij}
  else
    decrease w_{ij}

Here w_{ij} is the connection weight from neuron j to neuron i, MP_i is the postsynaptic membrane potential. However, the algorithm above only describes terms 1) and 2). Term 3), which defines synaptic modification only if the postsynaptic neuron is active, is still missing. The following algorithm completes the desired learning behavior:

if $MP_i > \Theta_{LTP}$
  if presynaptic neuron j does **not** emit a spike
    decrease w_{ij}

This learning process is only activated if the membrane potential of the postsynaptic neuron exceeds Θ_{LTP}. This happens far more frequently than the emission of a spike, but rarely occurs due to sparsely coding of pulse-coded neural networks.

6.2 Correlation-Based Learning

A further model of synaptic modification deals with time differences between pre- and postsynaptic spikes. Presynaptic spikes and postsynaptic spikes backpropagated to the dendritic tree add up to the postsynaptic membrane potential and determine synaptic modification. Since both signals have an effect on a short time scale, correlation of both is decisive for connection changes. Thus, the time difference between pre- and postsynaptic spikes is the main parameter for learning. Change of connection weight can be described by the so called *window of learning* [17]. Setting the occurrence of the presynaptic spike

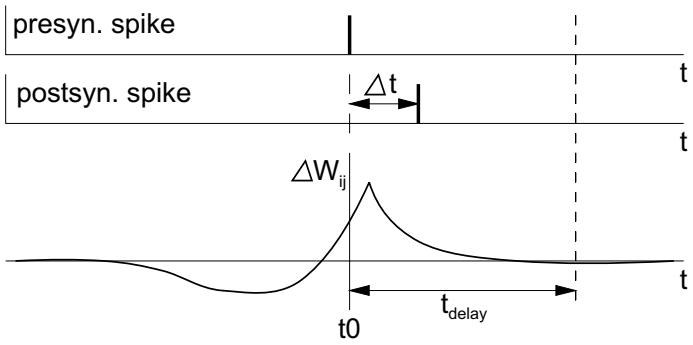

Fig. 6. Correlation based learning rule with learning window

to $t = t_0$, the time of postsynaptic firing defines the weight modification Δw_{ij} (Fig. 6). The window of learning is composed by superposition of exponential-functions. Its values tend to zero for large absolute time differences. In a digital simulation it can be approximated by a function with a finite width that is mainly determined by the refractory period of the postsynaptic neuron. Thus, most neurons fire not more than once within the window. To enable registration of postsynaptic spikes following the presynaptic one, the learning event has to be delayed by t_{delay}. Postsynaptic spikes occurring after this delay time do not influence synaptic modification. With the considerations above, correlation based learning can be written as an algorithm triggered by the firing of the presynaptic neuron:

if presynaptic neuron j sent out a spike t_{delay} before
  look for last time of postsynaptic firing
    change w_{ij} according to this time

As in the first algorithm for threshold based learning term 3) is missing. Again there is the necessity for completion:

if postsynaptic neuron i fires
  if presynaptic neuron j does **not** emit a spike
    decrease w_{ij}

Biologically motivated learning with a threshold learning rule or one based on spike correlation can be described by two event-driven algorithms. *Event-driven* in this case means that only the synapses that actually underlie modification have to be processed. The two events triggering learning processes are presynaptic spikes and the level of postsynaptic membrane potential.

6.3 Sender- and Receiver-Oriented Learning

For hardware realization there is a significant difference between learning processes triggered by presynaptic events and those triggered by postsynaptic

events. In the first case, starting from a firing neuron all synapses transmitting spikes from this neuron have to be processed. Quantity of modification depends on the state of the postsynaptic neuron, either the membrane potential or time of last firing. This process is very similar to the sender-oriented stimulation. Addressed by the presynaptic neuron the contents of the blockstart memory provide a pointer to the first connection in connection memory. During the stimulation phase, connection weights are read out and added to the addressed target neurons. Similarly, in the case of learning connection weights are read, modified according to the states of the addressed target neurons and finally written back. As in the *sender-oriented stimulation*, we call this *sender-oriented learning* (see Fig. 7).

In the second case, a postsynaptic event triggers learning and modification depends on the state of the presynaptic neuron, which refers to the emission of a spike in both learning rules described above. This process is called *receiver-oriented learning*. To meet the requirements of a flexible simulation basis, both types of learning should be implemented. Thus, access to connections must be provided from two directions: from the neuron address of the presynaptic and the neuron address of the postsynaptic neuron. This is essential for hardware realization. Sender-oriented learning fits very well into the sender-oriented architecture described above. Data-flow is similar. Connections are processed sequentially, starting from the first one that is addressed by the blockstart memory.

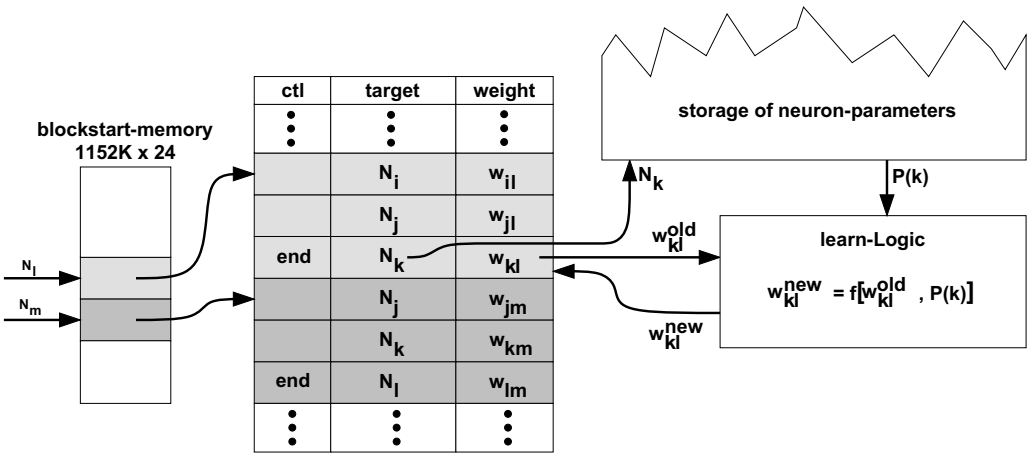

Fig. 7. Sender-oriented learning (neuron address N, weight w)

In general, not all connections in a network are modified by learning. Many may remain unchanged for the whole simulation. Thus, there are plastic and static connections in a network. According to this, a SIB in the connection memory that contains all connections starting from one neuron is divided into two blocks (Fig. 8). The first block stores all plastic connections, the second the static ones. The first static connection is marked by an LE-bit (Learn-End), the last by the BE-bit (Block-End) mentioned above.

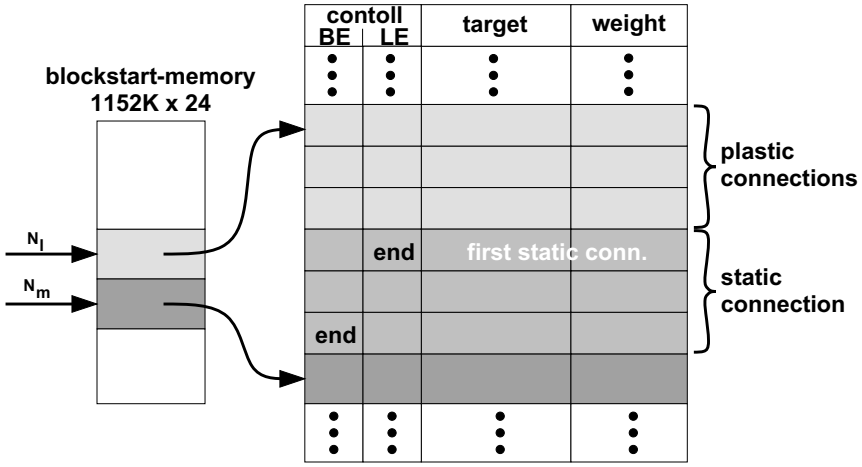

Fig. 8. SIB partition for plastic and static connections

Stimulation starts at the first position in SIB and processes sequentially all connections including the last one marked by the BE-bit. Similarly, learning starts at the first position, however it ends at the first static connection marked by the LE-bit. Consequently, only plastic connections are calculated and no simulation time is wasted by processing static connections. In the sender-oriented storage of connection data, there is no information about connections leading to a given neuron. Therefore an extension is required. This is possible by adding a second pointer memory that provides the address to the first connection leading to a given neuron in the connection memory (Fig. 9). This pointer memory acts just like the blockstart memory. Further additions are required for the connection memory. Each connection word must contain a pointer to the next connection leading to the same postsynaptic neuron. The address of the presynaptic neuron provides access to the state of the sender. A control-bit ctl_r marks the end of a receiver-oriented list of connections.

Triggered by the occurance of a postsynaptic neuron, the pointer memory addresses the first connection leading to this neuron. This connection can be modified, depending on the state of the corresponding presynaptic neuron. Additionally, the connection word provides a pointer to the second connection, and so on. All plastic connections leading to the same neuron form a list, which is processed sequentially until the last connection is detected by the ctl_r-bit. According to a connection count of 16M and 1152K sending neurons, as used in the SPIKE128k system presented below, the pointer to the next connections is a 24 bit-word and the number of the presynaptic neuron is a 21bit-word. With the pointer memory for the first connection, the total additional requirement for the receiver oriented extension is somewhat more than the one for the receiver-oriented structure.

In the receiver-oriented term of the threshold based learning rule, a learning event occurs if the postsynaptic membrane potential MP_i exceeds the learning threshold Θ_{LTP}. If a neuron is *active*, this happens for a series of following

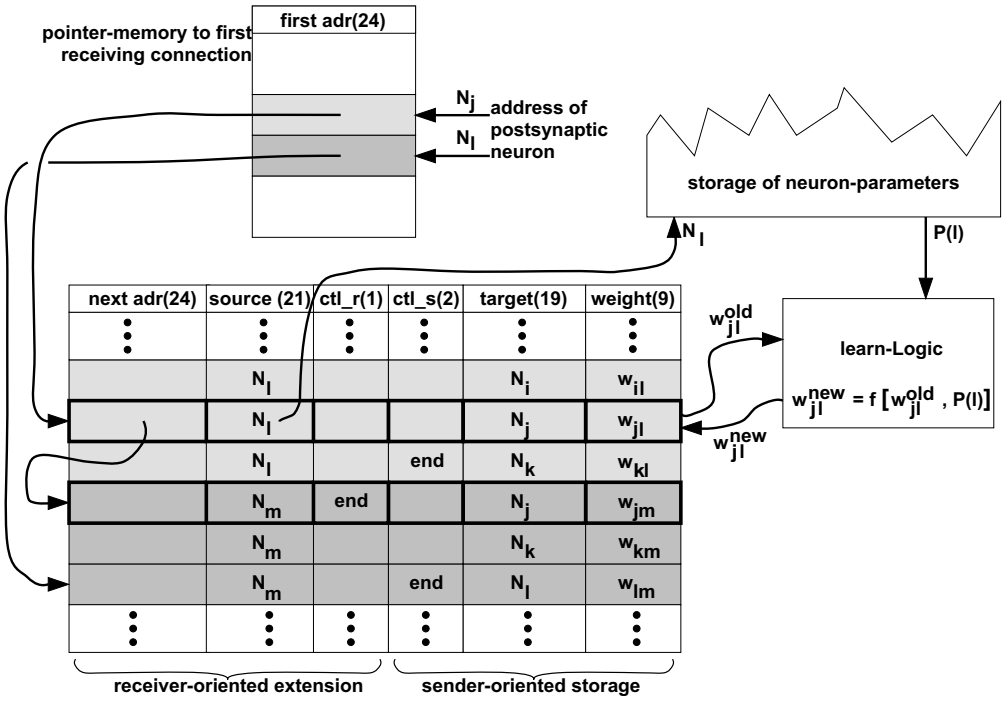

Fig. 9. Receiver-oriented pointer memory (neuron address N, weight w)

simulation steps (Fig. 10). In contrast, an action potential occurs only for one single simulation step. Thus, most learning events are receiver-oriented. Depending on the simulated net, the relation of sender-oriented events to receiver-oriented ones is about 1 to 10. One way to rectify this unbalanced behavior is the use of a sufficiently small decrement in term 3). As learning should be a rather slow process, connections must not change abruptly, and therefore a high resolution for connection weights is required. In the SPIKE128k system this is not the case as connections are stored in 9bit-words.

Restriction of receiver-oriented processes to every n-th step instead of every simulation step, with n approximately 10..100, leads to the same result. In simulations with some thousand simulation steps average network behavior is the same, even if Δw has the same order for sender- and receiver-oriented processes. The advantage of this approach is the possibility to choose a rather large value for Δw and consequently a lower resolution for connection weights. This leads to a significant simplification in storage and processing.

If a loss in simulation speed and a slight modification of learning behavior is accepted, the hardware requirement for the receiver-oriented extension is not necessary. In this case receiver-oriented changes of connection weights can be triggered by presynaptic neurons. In every n-th simulation step a learning process is started referring to ALL presynaptic neurons. In threshold-based learning a connection is weakened if the membrane potential of the postsynaptic neuron is above Θ_{LTP}. The same happens for correlation-based learning if

the postsynaptic neuron sent out a spike in the last m simulation steps, where m is the width of the discrete window of learning.

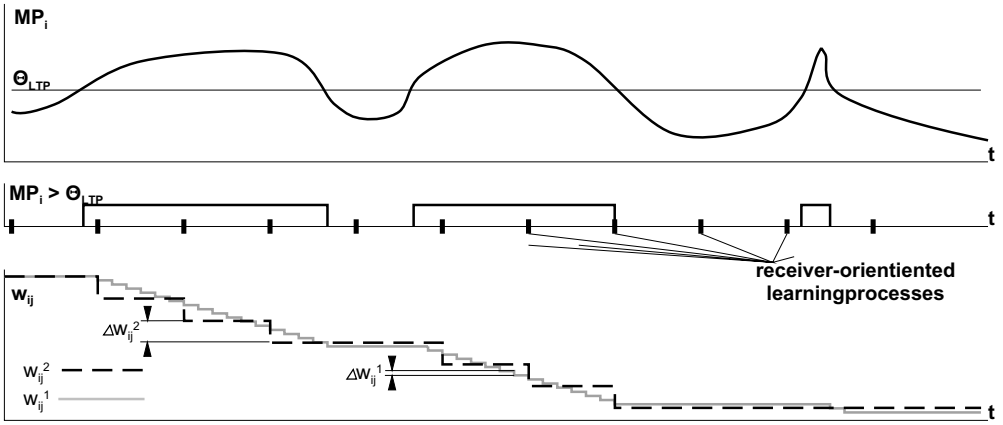

Fig. 10. Fine (w_{ij}^1) and coarse (w_{ij}^2) modification of synaptic weight

There are two disadvantages in connection with a significant simplification of hardware structure. Receiver-oriented learning is not triggered by the state of the postsynaptic neuron but by the virtual spiking of every presynaptic neuron. Since in this approach all plastic connections are processed, simulation is relatively slow. The state of a presynaptic neuron does not influence weight modification. If the presynaptic neuron sends out a spike, a connection should be strengthened according to term 1) instead of the receiver-oriented term 3) which decreases connection weights. The coincidence of presynaptic spike and learning is infrequent. Thus, on average, the error in learning behavior leads to a slight decrease of the increment according to term 1). This can be adjusted by choosing suitable values for Δw.

7 Simulation Platforms for Complex PCNNs

In the following several platforms developed at the Technical University of Berlin and the University of Paderborn for the simulation of complex PCNNs are presented. For platforms that have already been implemented, measured results are given, otherwise performance estimations are based on system simulations. As benchmarks two networks designed by the group of Prof. Eckhorn at the University of Marburg [32] were employed. The networks perform image segmentation and exhibit similar characteristics: the number of active neurons is about 15-20% and on average about 0.5% of all neurons emit a spike.

The *SPIKE128k* was developed and implemented by the group of Prof. Hartmann at Paderborn [4, 5, 10, 28]. This platform is based on a single processor unit as dedicated hardware that allows the computation of a neuron in a pipeline. Thereby a throughput of one neuron per cycle is achieved. Network topology is stored in a sender-oriented form in SIB-lists. The model neuron of SPIKE128k may consist of up to four dendrite potentials, axonal delays and an exponential and three other types of functions for the decay of dendrite potentials. In order to speed up simulation, the concepts of a decay list and a spike event list are implemented. Different algorithms of Hebbian or modified Hebbian learning are supported. The SPIKE128k consists of SRAM and DRAM, programmable logic such as PLDs and FPGAs as well as commercial arithmetic and logic components. As a modular system, it is based on a VME-backplane. Transputers are employed for communication between different modules and to the host. The system is capable of simulating up to 130,000 neurons with 16 million synapses.

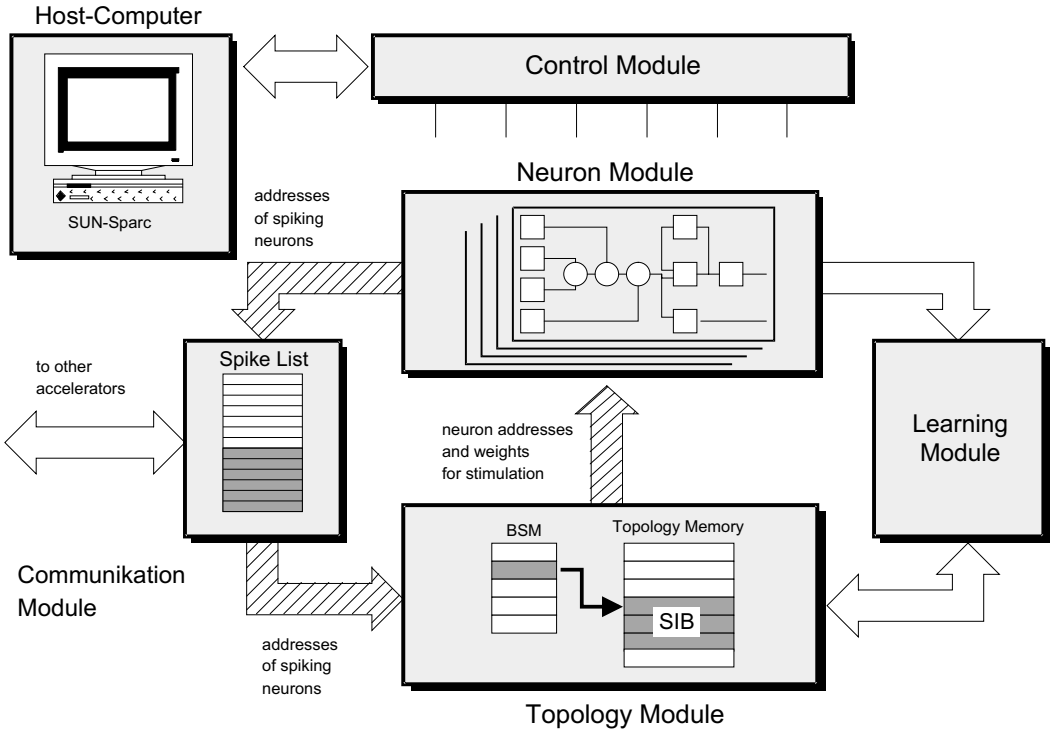

Fig. 11. Modular structure of the SPIKE128k-platform (simulation flow with hatched arrows similar to Fig. 2, topology module similar to Fig. 3)

The processing of neurons and the decay list takes place in a neuron module, containing six submodules for computing dendrite potentials, dynamic thresholds and axonal delays. Spikes are stored in a spike event list within a communication module. The communication module also stores incoming

spikes from other systems or the host and sends outgoing spikes to other systems. Communication is controlled by a transputer. In a topology module, the addresses of spiking neurons address a blockstart memory containing a pointer to the corresponding SIBs in a topology memory. SIBs are propagated to the neuron module where they stimulate dendrite potentials. Additionally, a learning module receives the addresses of the spiking neurons together with a ready-for-learning (RFL)-flag. This flag is the result of a comparison of the membrane potential with the learning threshold. Based upon this information the learning module modifies synaptic weights in the memory of the topology module. The simulation of one time-slice is divided in two phases: in a decay phase neurons from the decay list are computed and spikes are stored in the spike event list. In a subsequent stimulation phase the spike event list is read by the topology module and SIBs from the topology memory stimulate dendrite potentials. Learning is performed in parallel to the decay phase of the next time-slice. By employing a spike event list, out of 4 million synapses only 25,000 need to be computed. The decay list reduces the number of neurons to compute from 130,000 to 17,500 within the example network.

Threshold-based learning and correlation-based learning represent two variations of hebbian learning. Both of them may be described as event-controlled algorithms. The main difference is in the choice of postsynaptic parameters responsible for the modification of synaptic weights. For threshold-based learning the value of the membrane potential is decisive, for the correlation-based learning the occurrence of the last spike is crucial. Formally, learning may be described as $w_{ij}^{new} = f(w_{ij}^{old}, P(i), MODE)$, where the new, modified weight is a function of the old weight, the state of the postsynaptic neuron and the learning mode. Learning modes are sender-oriented and receiver-oriented learning procedures. A look-up table adequately fulfills such a function and is easily realized by a simple memory component. Thus, the actual learning algorithm does not require explicit calculation which would otherwise require complex processing components.

As previously pointed out, learning takes place in parallel to the decay phase when no access to the SIBs in the topology memory is required by the membrane module. However during the decay phase neuron parameters may not be accessed by the learning module. They are therefore transferred in two steps to the learning module via a FIFO-memory 1 depending on the learning algorithm. Sender-oriented learning considers spiking presynaptic neurons. Their addresses are transferred to FIFO-memory 1 during the stimulation phase, where they initiate learning actions. For receiver oriented learning a mirrored receiver-oriented topology list presents an efficient way of accessing all postsynaptic neurons connected to a presynaptic neuron. However in the SPIKE128k all presynaptic neurons are still considered as starting points. Their addresses are generated by a counter instead of being read from FIFO-memory 1.

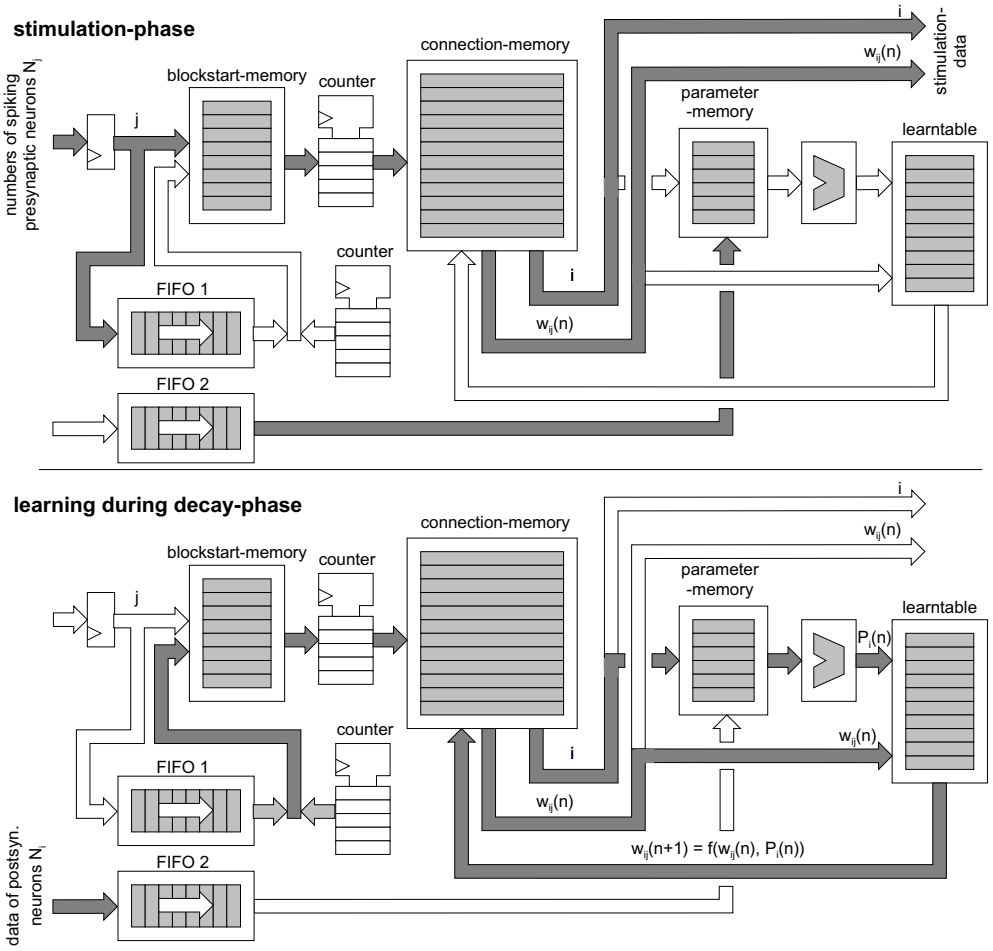

Fig. 12. Block diagram of the combined learning and topology module of the SPIKE128k during the two main simulation phases [28] (black arrows = active datapath)

7.2 ParSPIKE

The *ParSPIKE*-System [33, 34] was also developed by the group of Prof. Hartmann at Paderborn. The system is a further development of the SPIKE128k to a parallel computer consisting of digital signal processors (DSP). DSPs take over the main processing unit of the SPIKE128k, the neuron module.

A prototype implementation of the system using the SHARC ADSP21160 from Analog Devices is foreseen. For parallelization, neurons are distributed to several of these processors. Due to the reduced computational power and memory capacity of the SHARC in comparism with the SPIKE128k, the DSP can simulate 16,384 neurons at the most. Since the algorithm of the DSP is implemented in software, ParSPIKE offers high flexibility concerning neuron models and their computation. Performance evaluation of the ParSPIKE is based on the neuron model of the SPIKE128k using a decay list. The use of a

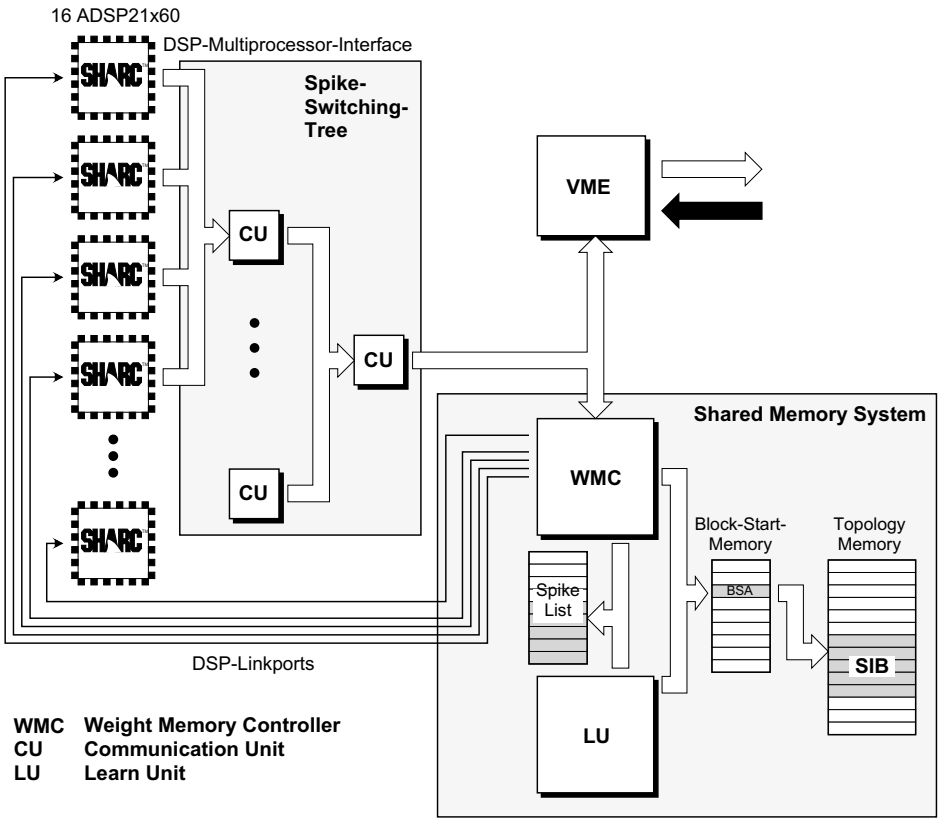

Fig. 13. Structure of the ParSPIKE-system for non-regular connections (nrc).

resubmission list showed only minimal performance improvement: out of 128k neurons of the benchmark network, 17,500 need to be computed when using a decay list and 15,000 when using a resubmission list. For this example the network received a static input with a frequent stimulation of similar neurons, so that only short dismissal times occurred.

There are two alternatives for storing the network topology. To support learning and irregular connection schemes a global sender-oriented SIB-memory is employed. For each accessing DSP there is a small local SIB-cache. Another alternative supports only regular connections. These are stored locally in the on-chip-memory of the DSPs as SIB-connection masks. The SHARC-DSP as a processing node offers several special characteristics. It contains a 512kByte on-chip-memory with link ports and a multi-processor-interface. In the ParSPIKE-system, the multi-processor-interface is used as output for the addresses of presynaptic spiking neurons. They are transferred through a tree-structure of hardware-switches (CU) to other SHARCs or to the global topology memory, which is controlled by a controller (WMC) (see Fig. 13). Out of the read SIBs, WMC sends each DSP the corresponding stimulation data via a dedicated linking port. Apart from the WMC, the global memory subsystem consists of a learning unit (LU). The learning unit supports the learning algorithms of the SPIKE128k, in particular the threshold-based

learning using the RFL-signals, in conjunction with a sender-oriented topology memory and an inverse receiver-oriented pointer memory. The memory subsystem consists of SDRAMs, for the CUs, the WMC and the LU the use of FPGAs is planned. The prototype is conceived as a VME-Bus-System. Boards for irregular connections contain 16 DSP and the memory subsystem, while boards for regular connections consist only of DSPs and CUs. An irregular-connection-board (nrc) may simulate up to 256k neurons with up to 32M synapses, a regular-connection-board (rc) 512k neurons. The prototype architecture with 2 irregular-connection and one regular-connection board, as well as a VME-workstation can simulate up to 1M neurons.

7.3 NESPINN

At the same time as the SPIKE128k was developed in Paderborn, Professor Klar's group in Berlin conceived the *NESPINN-System* (Neuro-Computer for Spiking Neural Networks) [14]. Compared to the FPGA-based approach of the SPIKE128k, NESPINN's main processing unit consists of two ASICs (Application Specific Integrated Circuits): the *NESPINN-Chip*, which is dedicated to the processing of the model neuron, as well as a *Connection-Chip* [25], which computes the regular connections of the network (Fig. 14). The use of ASICs allows an increased clocking frequency (SPIKE128K: 10MHz; NESPINN: 50MHz). As in the SPIKE128k, a NESPINN-Board is connected to a host computer and other NESPINN-Boards via a VME-bus. Main units on the NESPINN-board are a spike event list, a regular connection unit (Connection Chip), an irregular connection unit (DRAM-Unit) and a neuron-processor, the NESPINN-Chip. Addresses of neurons emitting a spike are written to the spike event list. For each spiking neuron the connection units supply the receiver-neurons and respective connection weights. The NESPINN-Chip adds these connection weights to the current dendrite potentials and combines them into a membrane potential. Dendrite potentials represent the actual excitation state of the neuron and are stored in the neuron state memory. If the membrane potential exceeds a threshold, the neuron emits a spike and its address is again written to the spike event list. With the capability of the Connection Chip to compute not only all receivers of a sender neuron, but also all senders to a receiver neuron, the regular connection unit allows the application of hebbian and modified hebbian learning rules.

In contrast to the SPIKE128k, NESPINN is not limited to a fixed neuron model, but allows the configuration of a neuron model with up to 16 dendrite potentials with different functionality (e.g. inhibitory, excitatory, multiplicative). Instead of a decay list, NESPINN marks active dendrite potentials and thereby prevents not only unnecessary computation of inactive neurons, but furthermore neglects inactive dendrite potentials of active neurons.

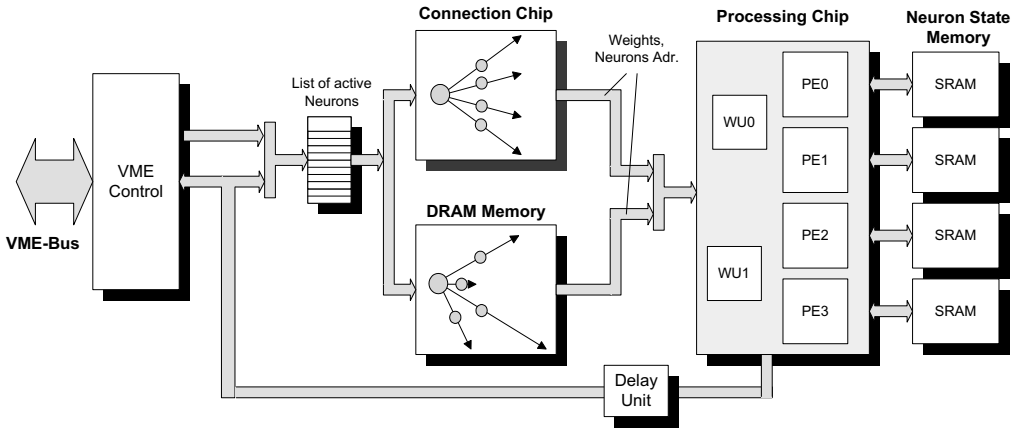

Fig. 14. Structure of the NESPINN-system (Processing Chip = NESPINN Chip)

7.4 MASPINN

After the NESPINN-System, the MASPINN-System was designed in Berlin [29]. The NESPINN-System aimed at the real-time simulation of about 10^5 spiking neurons. However, since network sizes attractive for image processing task are in the order of 10^6 neurons, the goal of the MASPINN (Memory optimized Accelerator for Spiking Neural Networks) was to gain another order of magnitude in speed. The basic structure of MASPINN is quite similar to the NESPINN architecture. MASPINN consists of a spike event list, a Connection Unit and a Neuron Unit, with a core processor: the *NeuroPipe-Chip* [30] (see Fig. 15). The NeuroPipe-Chip also allows a configurable neuron model with up to 16 dendrite potentials. The gain in performance compared to NESPINN is achieved by a higher system frequency of 100MHz and new architectural features. Such features are applied on board-level and chip-level. On board-level, weight caches have been introduced. They allow a further parallelization of the processing steps necessary for the simulation of a spiking neural network. Furthermore, a compressed Dendrite Potential (DP)-Memory, relaxes the bandwidth requirements of the neuron-processor-chip. On chip-level, the NeuroPipe-Chip applies pre-analysis of dendrite potentials to be processed. Pre-analysis tests the relevance of a dendrite potential in the context of the other dendrite potentials of the corresponding neuron. This reduces the computational load during the computation of the membrane potential in the pipeline of the NeuroPipe-Chip. Also on chip-level, an inhibition unit in the NeuroPipe-Chip may emulate the inhibition of the entire network or large parts of it. Since all parameters related to inhibition and their computation are hosted on-chip, the bandwidth and computational requirements of the entire system are also reduced. The performance of the NeuroChip has been evaluated on register-transfer-level by a VHDL-simulation (HDL: Hardware Description Language) [27]. A prototype of the chip has been fabricated in May 2000.

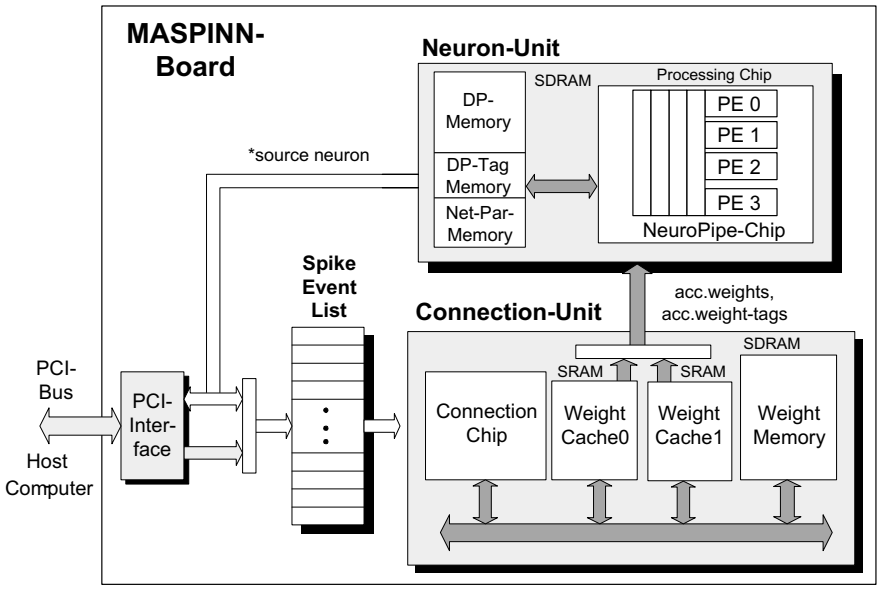

Fig. 15. Structure of the MASPINN-system (2 ASICs: NeuroPipe Chip and Connection Chip)

7.5 Parallel PVM-Software-Simulator

Based on the examinations for the ParSPIKE-System in the group of Prof. Hartmann in Paderborn a parallel software simulator based on PVM (Parallel Virtual Machine) for a Sun-workstation-cluster has been developed (Fig. 16). The simulation is organized as a farmer-worker-system. The simulator realizes the parallel algorithm of the ParSPIKE for the nrc-topology representation in combination with the SIB-cache and for the rc-topology representation by using locally stored SIB-masks (see Chap. 7.2). As a third option, the distribution of the nrc-SIB-topology to the workers is possible, because the workstations offer a much higher capacity of local memory than the DSPs of ParSPIKE. The performance evaluation of the ParSPIKE-approach is based on data from the simulation of the PVM-software-simulator in conjunction with results from a DSP-evaluation-system. In particular, the possibility of parallelizing the simulation has been examined in order to find an optimal mapping of neurons with minimal communication and optimal load balancing between the workers.

As a division of the network with a minimum of connections cut is desirable, graph-partitioning libraries [24] are applied. Depending on the partitioning communication and loads vary. In particular communication is an issue for the nrc-topology with SIB-cache. In case of a cache-miss, the SIB must be transferred from the farmer via the network to the worker. Connections of neurons between two different workers may not be stored in the cache. The cache-hit-rate is therefore a good indicator of the number of cut connections.

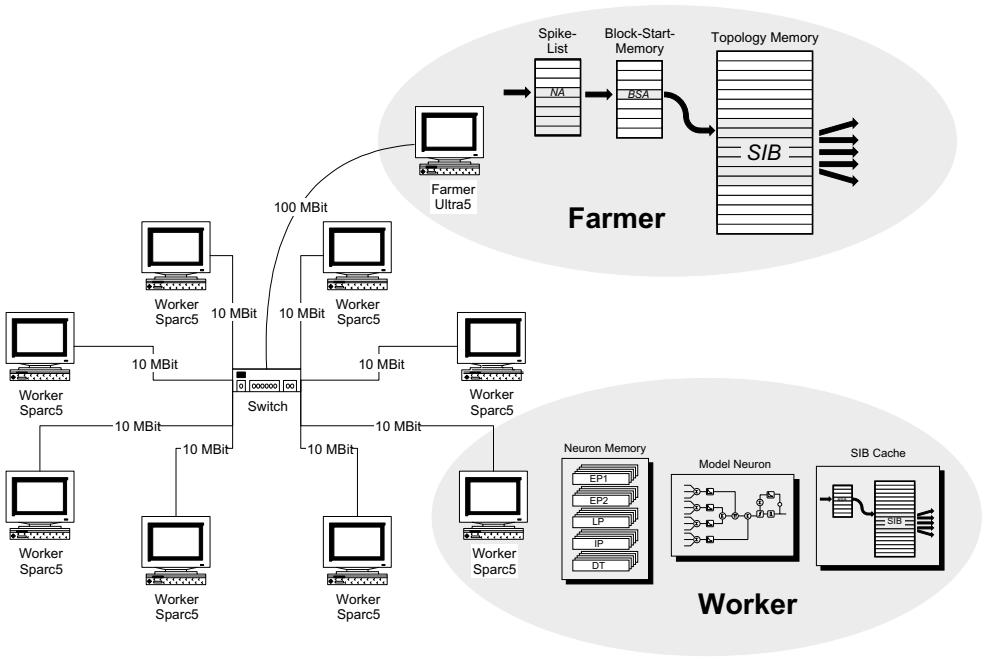

Fig. 16. PVM software simulator for workstation cluster

In the presented system of 8 workers cache-hit-rates of 60% were achieved. Concerning the computational load of workers distributions with variations of +/- 5% were found. Further investigations have shown that distributions with good load balancing require a high number of cut connections, while distributions with only few connection cuts result in bad load balancing.

Supported Concepts	SPIKE128k	ParSPIKE	NESPINN	MASPINN	PVM
Decay List	■	■			■
DP Tagging			■	■	
Resubmission List		■			■
Compressed DPs				■	
Pre-Analysis				■	
Global Inhibition				■	
Sender Oriented Topology	■	■	■	■	■
Receiver Oriented Topology		■			
Regular Connection Calculation		■	■		■
Weight Caching				■	
SIB Caching		■			■
Threshold Learning	■	□	□	□	
Time Correlated Learning	■				

Fig. 17. Supported concepts of implementations

8 Summary and Conclusion

The simulation of complex PCNNs operating on a similar time-scale to biological neural networks requires extreme computational power which conventional computers do not supply [15, 26]. Therefore concepts are necessary to provide the required power in terms of computation and communication e.g. as a dedicated hardware to compute such complex PCNNs in real-time. The presented concepts take advantage of the characteristics of PCNNs in image processing, in order to minimize simulation time and facilitate implementation in hardware. The concepts were developed by the groups of Prof. Hartmann in Paderborn and Prof. Klar in Berlin and were used in various combinations for simulator systems. Hence, the performance of the concepts may only be evaluated in the context of these simulator systems. Their performance was partially measured in existing implementations and partially estimated by simulations and extrapolations (marked *). The results shall facilitate a classification of the system. Note that PE (processor element) refers to a processing pipeline which in the case of the SPIKE128k corresponds to an entire board, in the case of the ParSPIKE to a DSP and in the case of NESPINN, MASPINN and CNAPS to a single pipeline on the processor-ASIC. NESPINN- and MASPINN boards may be used for further parallelization beyond parallel PE on a single chip. The evaluation is based on benchmark networks [32] with 128k neurons, 4M synapses, 15% activity and 0.5% spikes per time step. A resolution of 1ms per time-step is desirable. For larger networks performance data was determined by extrapolation. In doing so, for SPIKE128k and ParSPIKE a load distribution similar to the one of the benchmark network was assumed. Performance data for the CNAPS-Neurocomputer were taken from [15] for comparison [2].

Number of Neurons	Sun Ultra-Sparc II 200MHz	DEC Alpha 266MHz	Intel Pentium II 266MHz	CNAPS 50MHz 256PE	SPIKE128k 10MHz 1-8PE	ParSPIKE 100MHz 64PE *	NESPINN 50MHz 4PE *	MASPINN 100MHz 4PE*
16k	11ms	18ms	11ms	1.5ms	1.5ms	<1ms	0.38ms	0.06ms
128k	85ms	140ms	85ms	~1s	9ms	<1ms	3ms	0.39ms
512k	-	-	472ms*	-	11ms(4PE)*	2.5ms	11ms	1.56ms
1M	-	-	-	-	12ms(8PE)*	5ms	23.4ms	3.1ms

Fig. 18. Performance evaluation of different simulator-implementations

The presented implementations and concepts pursue different goals and therefore are not comparable solely on the basis of simulation speed. Apart from a maximum simulation performance, composed of simulation time and number of neurons, these goals are:

[2] The required simulation time to simulate one time-step is partially estimated by extrapolations and simulations (*). PE (processor element) refers to an entire board for the SPIKE128k, to a DSP for ParSPIKE and for the rest of the systems to a single pipeline on-chip.

- an efficient implementation that can be designed quickly and with little expense in terms of size, financial and human resources,
- a high degree of flexibility concerning the neuron model, network topology and learning,
- good handling with an easy-to-use simulation environment, efficient network specification and extensive debug-capabilities.

Neurocomputers based on dedicated neuroprocessor-ASICs like NESPINN and MASPINN certainly offer maximum simulation speed. Commercial processors like DSPs require several ten or hundred clock cycles to compute a neuron or a synapse and therefore cannot compete with the capability of an ASIC in computing main simulation steps within one clock cycle in a deep pipeline. The realization of the NESPINN and MASPINN-ASICs on a VME- and PCI-board provides the capability for a good integration into software simulator environments. On the other hand there are limitations concerning debug capabilities and flexibility to change i.e. the neuron model is only given within the foreseen programmability of the neuron model. Since NESPINN and MASPINN are based on processing with a very coarse granularity of parallelization, load balancing is not as crucial as for a software simulation on a parallel computer or the ParSPIKE-system. On the other hand the implementation of systems like MASPINN and NESPINN takes the highest efforts and costs since, apart from conventional components, ASICs need to be fabricated. Due to the dedicated hardware of these systems advances and modifications in algorithms are difficult and costly to implement. MASPINN as the successor of NESPINN presents the superior system.

Concerning flexibility, handling and the ease of implementation a software solution is superior to dedicated hardware. However the simulation performance for real scenarios is still insufficient. Even parallel implementation, e.g. on the basis of PVM, does not achieve a sufficient performance. Also, the fine granularity of parallelization on high-performance commercial parallel computers leads to the common problems of parallel processing. Communication between processing nodes with a high number of small data packages, as they frequently occur during the simulation of PCNNs, becomes the main bottleneck and prevents a satisfying simulation performance.

The ParSPIKE-concept combines commercial hardware with dedicated communication hardware to overcome this bottleneck in parallel processing. However a simulation performance comparable to the MASPINN-system is only achieved using a very high number of processors and good partitioning of the network. Due to the use of solely commercial processors and programmable logic, the implementation effort of ParSPIKE is smaller. Since all components are programmable, high flexibility is also guaranteed. Handling of ParSPIKE-boards is comparable to NESPINN- and MASPINN-boards. However the mapping of networks to the ParSPIKE-architecture with fine granularity of paral-

lelization is difficult.

Between the extremes of the ParSPIKE- and the MASPINN-system there is the SPIKE128k. Its implementation is mainly based on programmable logic, but as in the MASPINN a dedicated neuroprocessor-pipeline is realized. The system furthermore constitutes the basis for the presented learning algorithms for PCNNs. Even though latest technological achievements are not included in this system it shows a far superior performance than software implementations. Flexibility of the system is limited by the dedicated pipeline. Additionally, handling is reduced by the use of transputer communication links that are seldom used nowadays and the size of the system (one VME-chassis for 128k neurons).

The application of the presented systems and architectures pursues two aspects. On the one hand PCNNs should be developed and examined and on the other hand PCNNs should be applied to real-world tasks. For the development of PCNNs the required simulation performance is less crucial while there is a strong demand for flexibility and handling. Suitable simulators for such a task are software- simulators and for an increased simulation performance systems like SPIKE128K or ParSPIKE. For applications of real-world tasks on the other hand, systems like NESPINN and MASPINN - and to some extent ParSPIKE - present a superior platform for the simulation of PCNNs.

References

[1] E. L. Bienenstock, L. N. Cooper, P. W. Munro: *Theory of the Development of Neuron Selectivity: Orientation Specificity and Binocular Interaction in Visual Cortex.* Journal of Neuroscience 2, pp. 32-48 (1982)

[2] R. Eckhorn, H. J. Reitbœck, M. Arndt, P. Dicke: *Feature Linking via Stimulus - Evoked Oscillations: Experimental Results for Cat Visual Cortex and Functional Implications from a Network Model.* Proc. IJCNN89, Vol. I, pp. 723-730 (1989)

[3] R. Eckhorn, H. J. Reitbœck, M. Arndt, D. Dicke: *Feature Linking via Synchronization among Distributed Assemblies: Simulations of Results from Cat Visual Cortex.* Neural Computations 2, pp. 293-307 (1990)

[4] G. Frank, G. Hartmann: *An Artificial Neural Network Accelerator for Pulse-Coded Model Neurons.* ICNN95, Perth, Australia, In: Proc. ICNN95, Vol. 4, pp. 2014-2018 (1995)

[5] G. Frank: *Ein digitales Hardwaresystem zur echtzeitfhigen Simulation biologienaher neuronaler Netze.* HNI-Verlagsschriftenreihe, Georg Hartmann (Hrsg.), Bd. 26, Dissertation, Paderborn (1997)

[6] A. S. French, R. B. Stein: *A Flexible Analog Using Integrated Circuits.* IEEE Transactions on Bio-Medical Engineering, Vol. BME-17, No. 3, pp. 248-253 (1970)

[7] W. Gerstner, R. Kempter, J. L. van Hemmen, H. Wagner: *A neuronal learning rule for sub-millisecond temporal coding.* Nature, vol. 383, pp. 76-78 (1996)

[8] C. M. Gray, W. Singer: *Stimulus-specific neuronal oscillations in orientation columns of cat visual cortex.* Proc. Natl. Acad. Sci. USA, 86, pp, 1698-1702 (1989)

[9] G. Hartmann, S. Dre: *Self Organization of a Network Linking Features by Synchronization.* Parallel Processing in Neural Systems and Computers, G. Hauske (ed.), pp. 361-364 (1990)

[10] G. Hartmann, G. Frank, M. Schfer, C. Wolff: *SPIKE128K - An Accelerator for Dynamic Simulation of Large Pulse-Coded Networks.* MicroNeuro 97, Dresden, pp. 130-139 (1997)

[11] J.N.H. Heemskerk : *Neurocomputer for Brain-Style Processing. Design, Implementation and Application.* Ph.D. thesis at Leiden University, Rijksuniverditeit Leiden, Netherlands (1995)

[12] D. H. Hubel, T.N. Wiesel: *Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex.* The Journal of Physiol., Vol. 160, pp. 106-154 (1962)

[13] D. H. Hubel: *Exploration of the Primary Visual Cortex, 1955-78.* Nature, Vol. 299, pp. 515-524 (1982)

[14] A. Jahnke, U. Roth, H. Klar: *A SIMD/Dataflow Architecture for a Neurocomputer for Spike-Processing Neural Networks (NESPINN).* MicroNeuro 96, Lausanne, Switzerland, pp. 232-237 (1996)

[15] A. Jahnke, T. Schoenauer, U. Roth, K. Mohraz, H. Klar: *Simulation of Spiking Neural Networks on Different Hardware Platforms.* ICANN97. Springer Verlag, Berlin, pp.1187-1192 (1997)

[16] R. Kempter: *Hebbsches Lernen zeitlicher Codierung: Theorie der Schallortung im Hœrsystem der Schleiereule.* Darmstadt DDD , Naturwissenschatliche Reihe, Bd 17, Dissertation TU Mnchen (1997)

[17] R. Kempter, W. Gerstner, J.L. van Hemmen: *Hebbian Learning and Spiking Neurons.* Submitted to Physikal Review E vol. 59, pp.4498-4515 (1999)

[18] R. Linsker: *From basic network principles to neural architecture: Emergence of spatial-opponent cells.* Proceedings of Natl. Acad. Sci. USA, vol. 83, pp. 7508-7512 (1986)

[19] R. Linsker: *From basic network principles to neural architecture: Emergence of orientation-selective cells.* Proceedings of Natl. Acad. Sci. USA, vol. 83, pp. 8390-8394 (1986)

[20] R. Linsker: *From basic network principles to neural architecture: Emergence of orientation columns.* Proceedings of Natl. Acad. Sci. USA, vol. 83, pp. 8779-8783 (1986)

[21] W. Maass, C. M. Bishop: *Pulsed Neural Networks.* W. Maass and C. M. Bishop (Eds.), MIT Press (1998)

[22] K. Mohraz, U. Schott, M. Pauly: *Parallel Simulation of Pulse-Coded Neural Networks.* Proceedings of the IMACS World Congress '97, Berlin,

Vol. 6, pp. 523-528 (1997)

[23] E. Nierbur, D. Brettle: *Efficient Simulation of Biological Neural Networks on Massively Parallel Supercomputers with Hypercube Architecture.* Advances in Neural Information Processing Systems 6, pp 904-910 (1994)

[24] R. Preis, R. Diekmann: *PARTY - a software library for graph partitioning.* B.H.V. Topping, editor, Advances in Computational Mechanics with Parallel and Distributed Processing, pp 63-71 (1997)

[25] U. Roth, F. Eckhardt, A. Jahnke, H. Klar: *Efficient On-Line Computation of Connectivity: Architecture of the Connection Unit of NESPINN.* MicroNeuro 97, Dresden, pp. 31-38 (1997)

[26] U. Roth, A. Jahnke, H. Klar: *Hardware Requirements for Spike-Processing Neural Networks.* IWANN 95, Malaga, Spain, pp. 720-727(1995)

[27] T. Schœnauer, S. Attasoy, N. Mehrtash, H. Klar: *Simulation of a Digital Neuro-Chip for Spiking Neural Networks.* International Joint Conference on Neural Networks, IJCNN'00, Como, Italy (2000)

[28] M. Schfer, G. Hartmann: *A Flexible Hardware Architecture for Online Hebbian Learning in the Sender-Oriented Neurocomputer Spike 128K.* MicroNeuro 99, Granada, pp. 316-323(1999)

[29] T. Schœnauer, N. Mehrtash, A. Jahnke, H. Klar: *MASPINN: Novel Concepts for a Neuro-Accelerator for Spiking Neural Networks.* VIDYNN'98, Stockholm, (1998)

[30] T. Schœnauer, N. Mehrtash and H. Klar: *Architecture of a Neuroprocessor Chip for Pulse-Coded Neural Networks.* ICCIN'98 - International Conference on Computational Intelligence and Neuroscience, RTP, N. Carolina, USA, pp. 17-20 (1998)

[31] W. Singer: *Development and Plasticity of Cortical Processing Architectures.* Science, vol. 270, pp. 758-764 (1995)

[32] L. Weitzel, K. Kopecz, C. Spengler, R. Eckhorn, H. J. Reitbœck: *Contour Segmentation with Recurrent Neural Networks of Pulse-Coding Neurons.* CAIP'97, Kiel (1997)

[33] C. Wolff, G. Hartmann, U. Rckert: *ParSPIKE - A Parallel DSP-Accelerator for Dynamic Simulation of Large Spiking Neural Networks.* MicroNeuro 99, Granada, pp. 324-331 (1999)

[34] C. Wolff, G. Hartmann, H. Rahne: *Parallele Simulation groer pulscodierter neuronaler Netze auf DSPs.* DSP Deutschland 99, Mnchen, pp. 267-273 (1999)