Neural Networks 72 (2015) 152–167

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2015 Special Issue

Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks

Florian Walter *, Florian Röhrbein, Alois Knoll

Institut für Informatik VI, Technische Universität München, Boltzmannstraße 3, 85748 Garching bei München, Germany

ARTICLE INFO

Article history:
Available online 18 August 2015

Keywords:
Neurorobotics
Brain-inspired robotics
Spiking neural networks
STDP
Neuromorphic
Learning

ABSTRACT

The application of biologically inspired methods in design and control has a long tradition in robotics. Unlike previous approaches in this direction, the emerging field of neurorobotics not only mimics biological mechanisms at a relatively high level of abstraction but employs highly realistic simulations of actual biological nervous systems. Even today, carrying out these simulations efficiently at appropriate timescales is challenging. Neuromorphic chip designs specially tailored to this task therefore offer an interesting perspective for neurorobotics. Unlike Von Neumann CPUs, these chips cannot be simply programmed with a standard programming language. Like real brains, their functionality is determined by the structure of neural connectivity and synaptic efficacies. Enabling higher cognitive functions for neurorobotics consequently requires the application of neurobiological learning algorithms to adjust synaptic weights in a biologically plausible way. In this paper, we therefore investigate how to program neuromorphic chips by means of learning. First, we provide an overview over selected neuromorphic chip designs and analyze them in terms of neural computation, communication systems and software infrastructure. On the theoretical side, we review neurobiological learning techniques. Based on this overview, we then examine on-die implementations of these learning algorithms on the considered neuromorphic chips. A final discussion puts the findings of this work into context and highlights how neuromorphic hardware can potentially advance the field of autonomous robot systems. The paper thus gives an in-depth overview of neuromorphic implementations of basic mechanisms of synaptic plasticity which are required to realize advanced cognitive capabilities with spiking neural networks.

1. Introduction

Neurorobotics seeks to establish an experimental link between robotics and neuroscience by running biologically plausible simulations of neurobiological structures on robots. Differently from other approaches which draw only basic inspiration from neural mechanisms, a close correspondence between simulation and biological reality is therefore essential in order to make findings from both disciplines seamlessly exchangeable (Seth, Sporns, & Krichmar, 2005). Especially in the light of strong experimental evidence on the important role of the precise timing of the *action potentials* emitted by biological neurons (Bohte, 2004), this is a strong motivation for the use of *spiking neural networks* in neurorobotic applications. While classical neuron models from artificial intelligence are coarse approximations based on simplistic threshold logic and activation functions, spiking neurons employ dynamic models to reproduce the temporal dynamics exhibited by biological neurons as closely as possible. This allows for a diverse variety of efficient temporal neural codes (Gerstner & Kistler, 2002) and enables neurobiological learning based on spike-timing dependent plasticity (Bi & Poo, 2001; Markram, Lübke, Frotscher, & Sakmann, 1997; Sjöström, Turrigiano, & Nelson, 2001). The superior computational power of spiking neurons compared to artificial neuron models has also been theoretically proven by Maass (1997).

1.1. Spiking neural networks for neurorobotic applications

While spiking neural networks have been studied extensively in theory, there are considerably less results on real-world applications. This is mainly due to the fact that the execution of these networks is computationally extremely expensive because every single neuron is a small dynamical system which needs to be considered individually. The simulation of spiking neural networks with practically relevant sizes in the order from thousands up to millions of neurons therefore requires powerful hardware

* Corresponding author.
E-mail addresses: florian.walter@tum.de (F. Walter), florian.roehrbein@in.tum.de (F. Röhrbein), knoll@in.tum.de (A. Knoll).

http://dx.doi.org/10.1016/j.neunet.2015.07.004
0893-6080/

to achieve reasonable performance. In case of neurorobotics, the demands are even higher. To allow for interactions with the environment, the neural network must be executed in biological real-time.[1] At the same time, energy and space are very limited in most robots. But also for other applications, the requirement for powerful workstations and high performance computing to achieve acceptable simulation speeds is currently a major obstacle for a broader use of spiking neural networks. Moreover, it seems paradox that simulating a tiny fraction of the human brain is only possible at a multiple of its power consumption. This can be at least partly explained by the completely different paradigms underlying standard Von Neumann CPUs on the one hand and neural networks on the other. While the former implement a sequential model of computation which is based on a centralized local storage, information processing in the latter is massively parallel and distributed.

1.2. Early neuromorphic chip designs

The bad performance results achieved with standard microprocessors early motivated the development of neuromorphic hardware designs which are specifically dedicated to the efficient simulation of neural networks. First concepts for the VLSI implementation of chips tailored to the execution of artificial neural networks date back to the end of the 1980s (Graf, Jackel, & Hubbard, 1988; Sivilotti, Emerling, & Mead, 1986). However, the idea of mimicking neural structures with hardware is even older. As pointed out by Floreano, Ijspeert and Schaal (2014), a first approach towards neurorobotics has already been made in the 1950s by Walter (1950) who used electric tubes as neuron-like elements for controlling a simple mobile robot. At the end of that decade, the *Perceptron*, an artificial neuron with a learning rule for classification tasks, was introduced by Rosenblatt (1958) and implemented in hardware as the *Mark 1 perceptron* (Bishop, 2006, p. 196). Synaptic inputs were provided by an array of 400 photocells. To allow for autonomous learning, the efficacies of the individual synapses were controlled by motor-driven potentiometers. While these early physical devices are analog by design, analog computation also dominates neuromorphic integrated circuits. For the case of artificial neural networks, it is pointed out by Murray, Del Corso and Tarassenko (1991) that this is mainly due to the fact that the arithmetic operations required to compute a neuron's state can be implemented much more efficiently using analog units. The inferior accuracy compared to digital logic is compensated by the fault-tolerant information processing of neural systems. This, however, is not true for problems related to the transmission of analog signals between the neural computation units. With the spike-based communication of biological neurons in mind, Murray and Smith (1987) therefore proposed hybrid neuromorphic systems which transmit signals digitally as series of pulses while the neural computations are still carried out by analog circuitry. This *pulse-stream* architecture not only solves the problems caused by analog signal transmission but also enables the creation of larger networks by multiplexing several streams of pulses over a common wire (Murray et al., 1991). An extensive review of early hardware designs for the execution of artificial neural networks is available in Heemskerk (1995).

1.3. Neuromorphic hardware for neurorobotics

First work on neuromorphic integrated circuits for the execution of spiking neural networks was published only a few years after the pioneering research of Sivilotti et al. (1986) on dedicated hardware for artificial neural networks. Today, there is a huge variety of analog and digital neuromorphic chips available, each of them designed with different priorities in mind (Furber, Galluppi, Temple, & Plana, 2014; Merolla et al., 2011; Renaud, Tomas, Bornat, Daouzli, & Saighi, 2007). Unlike in earlier years, research and development are no longer only driven by prospective applications in machine learning and computer vision. The high degree of biological plausibility of the simulated spiking neural networks opens up a completely new field of applications in neuroscience which is relying more and more on large-scale simulations of brain structures to validate models against experimental findings and to make predictions based on theoretical hypotheses. In turn, new results from neuroscientific research can contribute to improve existing hardware designs. Inspired by this synergistic loop, the EU-funded *Human Brain Project* has been established with the goal of integrating all available knowledge about the human brain into a large computer simulation (The Human Brain Project, 2015). To achieve this goal, the project specifically promotes the development of powerful neuromorphic hardware designs.

Neuromorphic chips for spiking neural networks are a key ingredient for neurorobotics since they enable the execution of realistic brain structures at low power consumption and in biological real-time. But analogously to the microcontrollers encountered in classical robots, these chips need to be programmed. It is clear that this cannot be done using a standard programming language. Like a real brain, the neural network running on a neuromorphic chip needs to *learn* the desired behavior. Appropriate biologically plausible methods for unsupervised learning, reinforcement learning and supervised learning have been studied extensively in computational neuroscience. But it is often unclear which of these algorithms can be implemented on which type of neuromorphic architecture. This work tries to fill this gap by reviewing both selected hardware designs from neuromorphic engineering and neurobiological learning techniques. In Section 2, we present six current neuromorphic processors based on both analog and digital computation. This overview is complemented by a brief discussion of neuromorphic sensors as a natural means of interfacing neurorobotic agents with the environment. Section 3 is dedicated to algorithms for neurobiological learning and thus provides the theoretical foundations for applying neuromorphic hardware to real-world tasks. The actual implementation of learning on the neuromorphic processors presented is discussed in Section 4. Section 5 puts the results of this work into the context of robotics and argues how the proposed neuromorphic hardware designs can contribute to the field of autonomous robotics. A summary of the results is provided in Section 6.

2. Neuromorphic devices—an overview

Every microprocessor architecture must address two central questions (Murray et al., 1991): How are *computations* performed and by which means of *communication* is involved data exchanged? Standard Von Neumann CPUs use registers and RAM to store and exchange data. This data also includes the instructions for the arithmetic and logical units which perform the actual computations. In spiking neural networks, information is exchanged via spikes which are generated in a neuron's soma, then propagated along its axon and finally reach other neurons via synaptic connections. Every spike arriving at a neuron elicits a change in its membrane potential. A commonly used model for describing the temporal dynamics of this potential is the *leaky integrate-and-fire neuron* (Gerstner & Kistler, 2002):

$$\tau_m \frac{\mathrm{d}u}{\mathrm{d}t} = -u(t) + R \cdot I(t) \tag{1}$$

$$t^{(f)} : u(t^{(f)}) = \vartheta. \tag{2}$$

[1] Following the convention from Schemmel et al. (2010), the term *biological real-time* will refer to the temporal dynamics experimentally observed in biological neurons and biological neural networks throughout this work.

Instead of explicitly expressing the complex interplay of different types of ionic currents, the above equations define a phenomenological approximation. As usual in computational neuroscience, the dynamics of the neuron are described by an electrical circuit with the voltage $u(t)$ being the membrane potential. In case of the considered leaky integrate-and-fire neuron, the circuit consists of a resistor R and a capacitor C operating in a parallel configuration. When setting $\tau_m = RC$, $u(t)$ can be expressed with Eq. (1). Spikes are emitted at times $t^{(f)}$ when the membrane potential reaches a predefined threshold ϑ from below.

Unlike the membrane potential, spikes are discrete events. The actual shape of the current pulse is irrelevant. All information is encoded in the time of emission and the originating neuron. Many neuromorphic chips therefore implement an *address event representation* (AER) protocol for the transmission of spikes between neurons (Boahen, 2000). Like the pulse-stream architectures mentioned in the last section, AER uses digital circuitry for interneuron communication. But with the main information being encoded already in the time of spike, there is no need for additional modulation which would be required to encode analog signals. All information is carried by a single pulse. Multiplexing and thus more efficient usage of connection lanes between different sets of neurons is achieved by including an identifier of the source neuron in the AER packets.

In the next two subsections, we present six neuromorphic microprocessors realizing different approaches for computation and communication. Four of them implement the AER paradigm for propagating spikes throughout the network. Our focus is on designs which are under active development and which are in principle suited for large-scale spiking neural network simulations. An exhaustive review of all neuromorphic chips available would go far beyond the scope of this paper. To get an idea of the huge diversity of the field, we refer the reader to work of Indiveri et al. (2011) for an overview over analog implementations of different neuron models. The last subsection is dedicated to *neuromorphic sensors*. In a brief introduction to this strongly related field of research, we show how these types of sensors can complement neuromorphic microprocessors as input devices in neurorobotic applications.

2.1. Analog neuromorphic chips

The modeling of neural dynamics based on functionally equivalent electrical circuits gives way to direct physical implementations of neuron models. The resulting analog neuromorphic chips *emulate* biological neurons by means of electric currents and voltages. Compared to the digital designs from the next subsection, these *silicon neurons* (Cruz-Albrecht, Derosier, & Srinivasa, 2013) enable continuous state updates without discretization errors. By leveraging the inherent dynamics of their analog building blocks for computation, they allow for a higher integration density than digital chips (Renaud et al., 2007). On the downside, analog computation is susceptible to noise. Indiveri et al. (2011) therefore point out that exact quantitative simulation results are only possible with digital circuitry.

Although basic design principles like the AER paradigm are shared among many neuromorphic microprocessors, different priorities and project goals have yielded unique architectures with special functional and topological properties. The first of the four chips presented in the following is designed for maximum flexibility regarding the types of supported network topologies. In contrast, the chips discussed in the second and third part are strongly optimized for energy-efficient operation. The ROLLS neuromorphic processor presented in the last subsection implements three different types of synapses with individual properties and temporal dynamics.

2.1.1. The BrainScaleS Project

The neuromorphic hardware architecture presented in the following was initially conceived in the FACETS Project (Meier, 2013) and the BrainScaleS Project (Meier, 2015). Today, development is continued within the Human Brain Project. The central element of this architecture is the *High Input Count Analog Neural Network* (HICANN) chip which contains a total of 131 072 synapses and – depending on the configuration – up to 512 neurons (Fieres, Schemmel, & Meier, 2008; Schemmel, Fieres, & Meier, 2008). The simulation of larger networks is enabled by wafer-scale integration, i.e. the parallel operation of many chip dies on a common wafer. This technique allows for the integration of 384 HICANN chips on a single wafer of 20 cm diameter (Schemmel et al., 2010).

Computation. Every HICANN chip contains an *Analog Network Core* (ANC) which is comprised of 512 *dendrite membrane circuits* and two arrays of synapses (Schemmel et al., 2010). As illustrated in Fig. 1, the dendrite membrane circuits are organized in two symmetric columns. They act as basic building blocks for the application-specific assembly of neurons. Each of them is connected to a row of the corresponding synapse array which leads to a minimum number of 224 synapses per neuron. Neurons with larger synapse count can be configured with the *neuron builder* which contains a switch matrix for combining up to 64 dendrite membrane circuits to a single unit. With every of the dendrite membrane circuits being driven by 224 synaptic inputs, the total number of synapses per neuron supported by the architecture amounts to 14 336. The spikes arriving via these synapses drive the analog circuitry which emulates the neural dynamics of the conductance-based *exponential integrate-and-fire neuron* (Brette & Gerstner, 2005). Synaptic weights are stored locally at every synapse in digital memory cells with 4-bit precision. For spike transmission, the weight is converted to an analog value. The conversion scale is controlled by a programmable conductance parameter. The resulting analog weights determine the size of the postsynaptic potentials which are forwarded to the corresponding dendrite membrane circuits in response to incoming spikes. Each of the circuits has two configurable inputs which emulate ion channels with different reversal potentials. Setting appropriate values for these potentials allows, for example, to create excitatory and inhibitory synaptic connections. A switch in each synapse circuit selects the input of the corresponding dendrite membrane circuit to which incoming spikes are forwarded. Altogether, these mechanisms determine how synaptic impulses arriving at a connected group of dendrite membrane circuits contribute to the overall membrane potential. The emission of spikes is controlled by dedicated circuits located in the block *Digital Event Generation*.

All model parameters governing the dynamics of synaptic spike transmission, ion channels and the membrane potential are kept in programmable analog storage cells (Schemmel et al., 2008). Compared to biological neurons, these parameters are orders of magnitude smaller. The consequences of this are twofold: First, the electrical circuitry implementing the neuron model can be built smaller. Second, shrinking down the overall scale of physical units inherently comes along with a speedup of the resulting dynamics. As a result, the neurons emulated by the ANC run from 10^3 up to 10^5 times faster than biological real-time, enabling extensive experimental studies which are possible with neither standard hardware nor with biological nervous systems.

Communication. The highly accelerated neuron emulation requires a powerful communication system which is able to forward spikes between ANCs which are located in different regions of a single wafer or which may even be distributed across several interconnected wafers. The following paragraph is based on the detailed technical descriptions given by Fieres et al. (2008) and Schemmel et al. (2008).

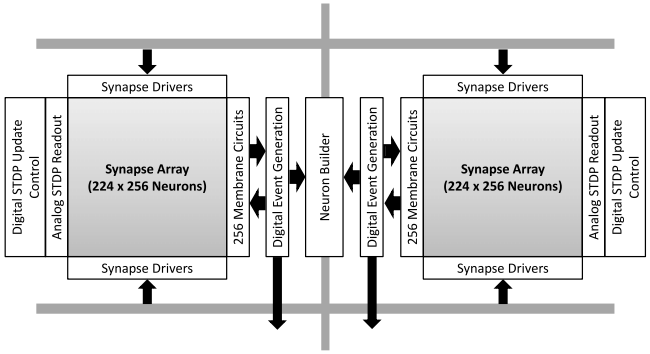

Fig. 1. Architecture of a single Analog Neural Network Core (ANC) (Schemmel et al., 2010). Each ANC has a symmetric structure with two 224 × 256 arrays of synapses. Every row of these arrays is connected to a membrane circuit, yielding a maximum number of 512 neurons per ANC. The *Neuron Builder* can be used to combine up to 64 membrane circuits to form neurons with more synaptic inputs. *Synapse Drivers* receive presynaptic spikes and inject them into the synapse arrays. Separate blocks implement weight updates via STDP (see Section 4.1). The gray bars indicate communication lanes for spike transmission.

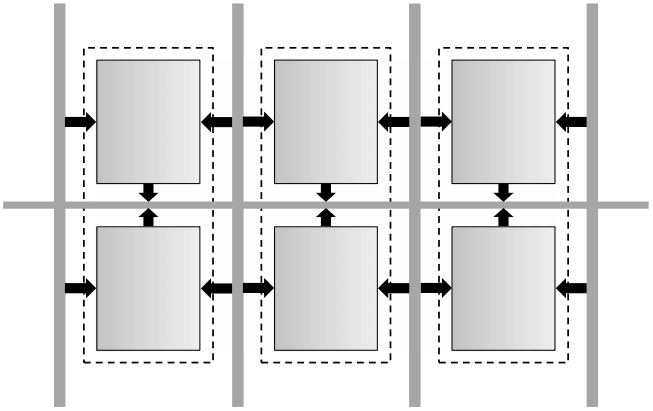

Fig. 2. Illustration of the Layer-1 communication infrastructure. Emitted spikes are first injected into the horizontal bus which has a width of 64 lanes. The vertical buses with 256 lanes then deliver the spikes to the synapse drivers. Note that the ANCs are rotated by 90° with respect to Fig. 1.

Spike transmission within a wafer is carried out by a network of horizontal and vertical data lanes interconnecting the ANCs in a grid-like layout. A schematic illustration of this *Layer-1* communication system is depicted in Fig. 2. Every horizontal pathway contains 64 bus lanes, each of which transfers the spikes of up to 64 neurons. The neurons in the ANCs are therefore partitioned into groups of 64 units. All neurons belonging to a group are connected to a *priority encoder* which ensures that at every time instant only one spike is propagated from the neuron builder to the bus lane. In case several neurons of the same group emit concurrent spikes, the one originating from the neuron with the highest priority is forwarded first. All other spikes are delayed. Every spike is represented as a data packet storing a 6-bit identifier of the source neuron. Following the AER paradigm, the timing of the spike is directly encoded in the timing of the data packet.[2] Repeaters at the boundaries of each HICANN chip enable signal transmission across the whole wafer without data loss. Crossbar switches at the intersections of horizontal and vertical pathways route spikes towards the ANCs containing their target neurons. The vertical data buses thus gather spikes for all connected ANCs which explains their increased width of 256 lanes. As depicted in Figs. 1 and 2, the ANCs are connected to these lanes via the *Synapse Drivers* at both sides of the synapse arrays. Each synapse driver provides the input of a selectable bus lane to two rows of synapses. Further constraints on the connectivity are imposed by the restriction that a certain bus lane can only be connected to at most one synapse driver within one ANC. However, this is partly mitigated by a mechanism allowing neighboring synapse drivers to share their input. As soon as a spike packet arrives at a synapse driver, the first two bits of its address are decoded, yielding a group of synaptic circuits. Every of these circuits contains a further decoder which checks the remaining four address bits and forwards the spike to the dendrite membrane circuit in case of a match.

The *Layer-1* communication described above enables flexible signal transmission between the chips of a single wafer. Communication between different wafers involves considerably longer latencies and cannot be realized with Layer-1 packets. A separate *Layer-2* communication protocol therefore stores the timing of spikes digitally in the payload of the packets. The required interfaces are implemented in 12 FPGAs[3] per wafer (Jeltsch, 2014). Every FPGA is equipped with two 1 Gbit uplinks for direct communication with other FPGAs and Ethernet connections. While it is possible to use the latter option for Layer-2 communication, another important purpose of the Ethernet interface is to provide an input and output channel for external devices. However, since the bandwidth available for real-time communication with external hosts is limited, each of the FPGAs additionally contains an internal storage for up to $2.5 \cdot 10^8$ spike events which can be injected at runtime with low jitter (Jeltsch, 2014).

Programming. The previous two paragraphs give clear proof of the high complexity of the BrainScaleS system. Running an actual neural network on the platform requires a mapping of neurons to ANCs and a corresponding configuration of the neuron builders, synapse drivers, crossbar switches, etc. Making the platform accessible to researchers from other fields requires an automated mapping between high-level descriptions of neural networks and low-level settings of hardware parameters. Based on this motivation, the software framework *PyNN* was developed by Brüderle et al. (2009). PyNN defines a convenient interface for specifying neural networks in the programming language Python. A simulator-specific backend converts these specifications to executable representations for the selected simulator. By exchanging the backend, a neural network can be easily run on different types of simulators and hardware.

The mapping of a neural network specified with PyNN to a configuration set for the BrainScaleS system is performed in two steps (Fieres et al., 2008). First, the neurons of the model must be assigned to the ANCs. Based on this assignment the routing can be computed. Since there is no arbitration mechanism governing access to the bus lanes, it is in general crucial to ensure that each lane is connected to the output of at most one ANC. However, disabling the repeater circuits between ANCs also allows for splitting lanes and thus connecting one lane to several ANCs. At the beginning, the routing algorithm connects ANCs with adjacent neurons by setting the crossbar switches accordingly. In a second step, the configurations for the synapse drivers and the address decoders of the synapses are computed. Due to connectivity constraints imposed by the hardware architecture, not all synaptic connections might be realized.

[2] Fieres et al. (2008) argue that this communication mechanism does not follow the AER scheme. However, they refer to a specific arbitration-based implementation of AER from Mortara and Vittoz (1994) while the definition assumed in this paper addresses solely aspects related to the encoding of information in AER packets.

[3] Field-programmable gate arrays.

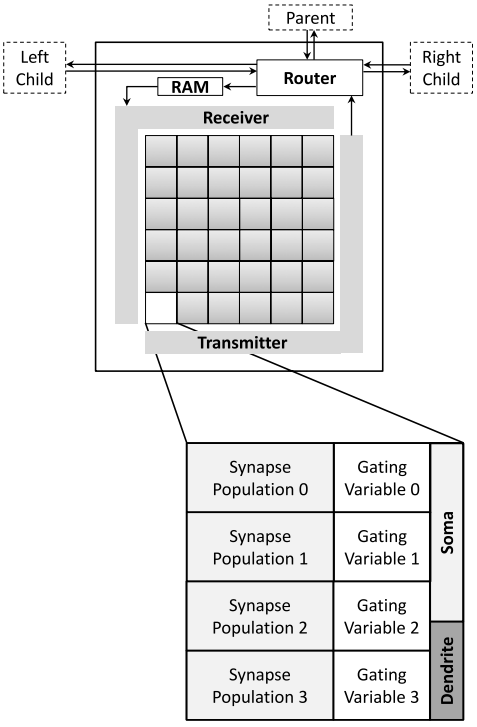

Fig. 3. The architecture of a single Neurocore (Benjamin et al., 2014). All neural circuits are arranged in a 256 × 256 grid which is surrounded by transmitters and receivers for sending and retrieving spikes. An integrated router is connected to the parent core and the child cores. As shown in the inset at the bottom, a single neuron is comprised of 4 synaptic populations, 4 gating variables and circuitry implementing the dynamics of the dendrite and the soma.

2.1.2. Neurogrid

Most of the space on the HICANN chip discussed in the last section is occupied by the synapse arrays. But still, when one wants to maximize the number of neurons emulated per chip, the resulting 224 synaptic inputs per unit are relatively small. Moreover, the at most 512 neurons accommodated on an ANC are not sufficient for complex real world applications. The *Neurogrid* neuromorphic hardware platform developed by the *Brains in Silicon* research group at Stanford University implements a much higher number of neurons and synapses at the trade-off of decreased flexibility (Boahen, 2015). Neurogrid consists of 16 interconnected *Neurocores*. Each of them contains 65 536 analog two-compartment neurons, yielding a total of 1 048 576 neurons with up to six billion synapses (Boahen, 2015). Fig. 3 provides an overview of the architecture of a single Neurocore. The complete system description in the following paragraphs is based on the documentation published by Benjamin et al. (2014) and Merolla, Arthur, Alvarez, Bussat and Boahen (2014).

Computation. The 16 Neurocores of the Neurogrid system are optimized with respect to energy efficient operation while at the same time accommodating a large number of neurons with many synaptic inputs. As shown in the upper part of the architecture diagram from Fig. 3, the silicon neuron circuits are arranged in a quadratic grid with 256 rows and columns. Realizing a two-compartmental dynamic model, every neuron has separate circuits for its *soma* and the *dendrite*. Incoming spikes from presynaptic afferents enter through four *synapse population circuits* and elicit time-varying synaptic conductances. A *shared dendritic tree* propagates the changes also to neighboring neurons to model the dynamics of overlapping dendrites found in biological neural networks. Currents from the synapse populations enter both the soma and the dendrite. Moreover, the soma also receives input from the dendrite. Spike generation is controlled by an emulated sodium current which triggers a reset pulse. The refractory period after spikes is realized by a potassium conductance. Finally, two *channel populations* implement conductances driving the dendrite.

The detailed behavior of all the circuits mentioned above can be controlled by parameters hold in on-chip RAMs of every Neurocore. The silicon neurons implemented by these circuits operate in biological real-time using energy-efficient subthreshold operation of the involved electronic elements. One of the most notable features of the neuron model emulated by the Neurocores is its *shared dendrite* architecture. Unlike in the synapse array of the HICANN chip which contains separate circuits for every single synapse, all synapses of a Neurocore neuron share only four synaptic population circuits. The individual weights of the virtual synapses are stored in a RAM. As already explained earlier, a shared dendritic tree additionally emulates crosstalk between the dendrites of neighboring neurons. While this approach requires multiplexing of the synapse circuits over all incoming spikes, it allows for a considerably larger number of synapses per neuron compared to an architecture with dedicated synapse circuits.

Communication. Spike transmission in the Neurogrid system is carried out by a digital communication infrastructure which is based on the AER paradigm. As shown in the upper part of Fig. 3, every Neurocore contains both a transmitter and a receiver. Outgoing and incoming spikes are gathered and delivered row-wise, i.e. all spikes of neurons placed in the same row of the array are transmitted in a single packet and delivered in parallel. Pipelining of the involved processing steps enables an even higher performance by already initializing the spike transmission for the next row while the packet for the previously processed one is being sent. Additional speed is gained by parallel use of the four synapse population circuits available for every neuron. A router contained in every Neurocore is responsible for forwarding packets to their destination. To avoid deadlocks in the communication system caused by competing packets, the routers implement a provably deadlock-free wormhole routing protocol which is based on a binary tree topology. As illustrated in Fig. 4, all 16 Neurocores are connected according to this topology. The protocol encompasses two steps, in the first of which the data packet generated by a Neurocore travels upward the tree until it reaches the *lowest common ancestor* of the source node and the target node. In the second step, the packet is forwarded down into the subtree containing the destination Neurocores. Multicast routing is accomplished by copying the data packet to all child nodes as soon as it arrives at the lowest common ancestor of the destination nodes. Every Neurocore finally decides locally if a received packet is processed or discarded. The routing information itself is completely encoded in the packet, which allows for fast processing without memory lookups.

The mechanisms required for the multicast routing described at the end of the last paragraph are completely implemented in every Neurocore. They realize *secondary axon-branching* for connecting corresponding locations in different Neurocores. The *Daughter Board* depicted in Fig. 4 is capable of *primary axon-branching* which is required for freely configurable synaptic connections.

Programming. The Neurogrid system includes a complete software stack for the definition, simulation and visualization of spiking neural networks on the hardware platform. A graphical user interface provides access to hardware settings and is used to control and visualize simulations of neural networks which are constructed using the Python API *NGPython*. Further components implement the transmission and mapping of data between the hardware and the software. The controllers *FX2* and *CPLD* depicted in Fig. 3 handle host communication via a USB.

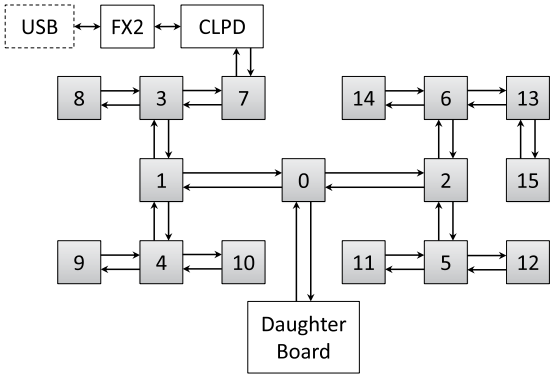

Fig. 4. Connection topology of the Neurogrid system (Benjamin et al., 2014). The numbered boxes correspond to the 16 Neurocores. The chips *CLPD* and *FX2* implement communication with the host via USB. The *Daughter Board* enables *primary axon-branching* for arbitrary synaptic connectivity between any pair of neurons.

2.1.3. The HRL SyNAPSE Project

One of the principal challenges in neuromorphic hardware design is to find an efficient way of realizing potentially arbitrary connectivity between the neurons while at the same time simulating as much synapses per neuron as possible. In case of the HICANN chip discussed in Section 2.1.1, this issue is resolved by physically implementing a huge number of discrete synapses which occupy a considerable portion of the space on the chip. Neurogrid shares synaptic circuitry and uses a special routing protocol to enable high spike rates while maintaining dense connectivity. In this section, we introduce yet another neuromorphic implementation of synapses by discussing a neuromorphic chip design developed within the DARPA-funded *HRL SyNAPSE Project* (DARPA, 2015; Srinivasa & Cruz-Albrecht, 2012). It is based on *synaptic time multiplexing* (STM) and uses memristors to store synaptic weights. Fig. 5 provides a high-level schematic overview over its architecture. The chip contains a grid of 24×24 nodes. Every of these nodes emulates a neuron with 128 synapses, resulting in a maximum network size of up to 576 neurons and up to 73 728 synaptic connections. The overall power consumption of the system amounts to 130 mW. In the following three paragraphs, we will explain how the chip realizes neural computation and how it is programmed.

Computation. It has been already mentioned above that the architecture of the HRL chip is based on the STM paradigm. As indicated by the name, this means that a single synaptic circuit on the chip computes multiple logical synapses of the simulated neural network. The justification that this is feasible is based on the same arguments as in case of the AER communication scheme: The electrical circuitry is able to operate at much higher speeds than biological neurons. Therefore, a single synaptic circuit can easily emulate multiple logical synapses by quickly switching between the corresponding parameter sets. The jitter introduced by this multiplexing is minimal compared to the temporal timescale of the dynamics of the individual neurons and the neural network (Cruz-Albrecht et al., 2013).

The inset at the bottom of Fig. 5 depicts the main components of a single node on the chip. The actual computation of neural dynamics is implemented in the *Processing Core*. Unlike the architectures discussed before, the HRL chip emulates a simplified version of the leaky integrate-and-fire neuron model which is highly optimized for low energy consumption (Cruz-Albrecht, Yung, & Srinivasa, 2012). The model neither supports inhibitory synaptic input nor does it have a refractory period following after spike emission. Moreover, the emitted spikes are uniform rectangular voltage pulses which are forwarded to the synaptic circuits of other nodes. On arrival of a presynaptic spike, these

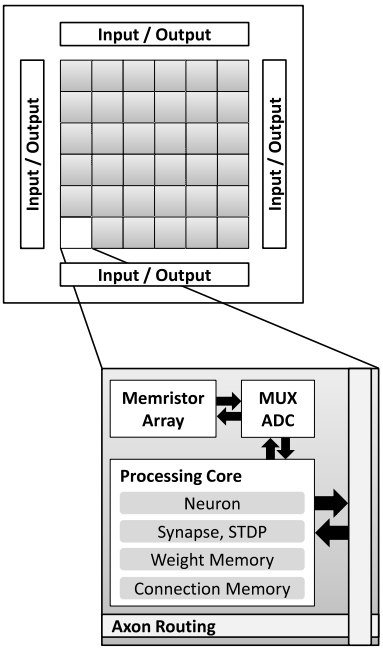

Fig. 5. Simplified system architecture schema of the neuromorphic chip developed by the HRL SyNAPSE Project (Cruz-Albrecht et al., 2013). The upper part of the figure depicts the complete chip. It is comprised of 576 nodes which are arranged in a two-dimensional grid of 24×24 nodes. The details of a single node are shown in the inset at the bottom. Every node contains storage for synaptic weights, a neural circuit for the actual processing and routing logic for forwarding and receiving spikes from other nodes.

circuits generate a postsynaptic current in the corresponding neuron circuit. The actual strength of this current is determined by the weight of the synapse.

Every node contains only a single synaptic circuit. STM enables this circuit to emulate up to 128 logical synapses. The corresponding synaptic weights are stored in the 8×16 *memristor array* shown in Fig. 5. In every *cycle period* of the synaptic multiplexing, each logical synapse is assigned a *slot* of 100 μs. Within a single slot, multiplexers (MUX) are used to address the corresponding cell of the memristor array. The synaptic efficacy stored in this cell is then converted to a 3-bit digital value (ADC) and forwarded to the synapse. At the end of the time slot, possible weight changes are converted to an analog value (ADC) and stored in the corresponding memristor cell. In addition to the memristors, the nodes of the HRL architecture also contain auxiliary CMOS memory for storing synaptic weights which can be used to operate the chip without the memristors. Additional memory further allows for the adjustment of various neural parameters and time constants (Cruz-Albrecht et al., 2013).

Communication. Data exchange with other external devices is enabled by the input/output channels depicted at the top of Fig. 5. The horizontal and vertical *axonal routing channels* shown at the bottom transmit spikes to other nodes of the same chip. Since the mapping of synapses changes in every STM time slot, the routing configuration of the nodes must be adapted correspondingly. For this reason, every node contains a connection memory which stores the configuration of a set of switches that control the routing along the axonal data lanes (Cruz-Albrecht et al., 2013). At the beginning of a new time slot, the states of these switches are adapted according to the corresponding entry of the connection memory. It is important to note that there is no AER protocol required since the routing mechanism establishes point-to-point connectivity between the presynaptic neuron and its postsynaptic receivers (Minkovich, Srinivasa, Cruz-Albrecht, & Youngkwan Cho, 2012). In particular, this implies that – if an appropriate routing

configuration can be determined – there is a guarantee that all spikes can be transmitted without data loss due to congested interconnects.

Programming. Executing an arbitrary network topology on the HRL neuromorphic architecture requires to determine a mapping which assigns neurons to processing nodes and synapses to STM time slots with an appropriate routing. The first task is the *placement problem*, the second one the *routing problem* (Minkovich et al., 2012). Each task is solved in a separate step. In the following, we will only give a quick outline of the general principles underlying the mapping algorithm. For a complete description of the *neuromorphic compiler*, the reader is referred to the work of Minkovich et al. (2012).

The *placement* of the neurons is solved heuristically in four consecutive steps. First, an initial mapping of neurons to nodes is computed using an iterative *quadratic wirelength minimization* algorithm which minimizes the total length of axonal routing lanes occupied for connecting the neurons. At the same time, this also minimizes the number of switches required to set up the routes. In the second step of the placement algorithm, the density of clusters of neurons is reduced by applying *diffusion-based smoothing*. The actual assignment of neurons to nodes is carried out in the *legalization phase* using bipartite matching from graph theory. Finally, the resulting placement is further improved by a *simulated annealing* algorithm which tries to heuristically reduce the wirelength by moving single neurons to their optimal positions.

The routing is set up based on the placement computed by the algorithm described above. First, an estimate of the minimum number of time slots necessary to implement the given neural network is computed and synapses are distributed over these slots in a round robin fashion. If an A^* routing algorithm finds collision free routes between all synapses in all time slots the resulting switch configuration can be computed. Otherwise, the assignment process is restarted with an increased number of time slots. If no more free time slots are available the network cannot be executed on the chip without omitting synaptic connections. In a final step, the resulting switch configurations for each STM time slot are compressed to reduce the amount memory required to store the configuration data on the nodes.

2.1.4. The ROLLS neuromorphic processor

One of the most recent chip designs presented in this review was developed by Qiao et al. (2015). The *Reconfigurable On-line Learning Spiking* (ROLLS) neuromorphic processor contains 256 analog silicon neurons and a total of 131 072 synapses. Compared to previous chip designs which already implement many of the features realized in the ROLLS neuromorphic processor (Indiveri & Fusi, 2007), the new design integrates a considerably higher number of neurons and offers a huge amount of different configuration options. Like its predecessors, the ROLLS chip is able to operate in biological real-time. The power consumption in typical scenarios is 4 mW (Qiao et al., 2015). Special focus has been put on the implementation of different synaptic plasticity mechanisms, which will be discussed in Section 4.3. In the following, we will only consider how the ROLLS neuromorphic processor emulates neural dynamics and how it is programmed. A schematic overview of the chip architecture is available in Fig. 6.

Computation. The 256 analog neurons of the ROLLS chip are depicted as triangles at the bottom of the figure. Like the silicon neurons of the HICANN chip, they emulate the dynamics of the exponential integrate-and-fire neuron model. Every neural circuit is connected to three different types of synaptic circuits. The first 256×256 synapse array at the top of Fig. 6 contains synapses which store configurable but fixed synaptic weights with a precision of 2 bit. Each of the synapses can be declared to be either excitatory or inhibitory. To enable operation in biological real-time in spite of small circuit sizes, the electrical circuitry is realized based on the *Differential Pair Integrator* (DPI) (Bartolozzi & Indiveri, 2007). In every column of the grid, two of these integrators emulate the temporal dynamics of the synaptic AMPA and GABA receptors for excitatory and inhibitory synapses, respectively. Every synaptic circuit contains a pulse generator which is attached to the AER communication bus and generates pulses in response to AER events. The synapses in the second 256×256 array only operate in excitatory mode. Consequently, there is only one DPI circuit per column which emulates the synaptic NMDA receptors. The efficacies are adapted during runtime based on a learning rule which will be discussed in Section 3.1.3. Finally, a third array contains 256×2 *virtual synapses* which have configurable weights and support both excitatory and inhibitory connectivity. Unlike the other synaptic circuits which correspond to individual connections between neurons, the virtual synapses operate in a shared mode which allows for connecting the postsynaptic neuron to a large population of presynaptic neurons through a single synapse. While this approach is considerably more efficient in terms of space compared to individual circuits for every connection, it implies that all synaptic connections emulated by a virtual synapse share the same parameters.

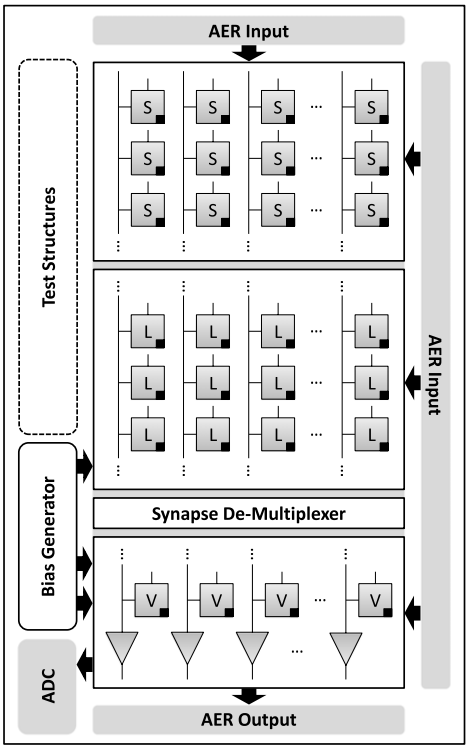

Fig. 6. Schematic overview over the ROLLS neuromorphic processor (Qiao et al., 2015). The chip contains two 256×256 grids of synapses for short-term plasticity (STP) and long-term plasticity (LTP) (see Section 4.3). A *synapse de-multiplexer* can assign several rows of synapses to a single silicon neuron. The additional *virtual synapses* above the neuron circuits can simulate background activity of the neural network. A *bias generator* stores global network parameters. The *analog digital converter* (ADC) can be used to read analog state information from neural and synaptic circuits. The *test structures* are irrelevant for the neural simulation.

As already mentioned above, the 256 silicon neurons emulate the exponential integrate-and-fire model. By default, every neural circuit receives input from the 514 synapses of its column. The *Synapse De-Multiplexer* allows for the configuration of neurons with more synaptic inputs by combining arbitrarily many synapse columns. Like in the case of the HICANN chip, the neural circuits

whose synaptic input column has been connected to other neurons are no longer available for simulation. Merging synapses therefore allows for increased connectivity at the price of a smaller network size.

Communication. All neural spikes generated and received by the ROLLS neuromorphic processor are transmitted using an AER protocol with 8-bit neuron addresses. As illustrated in Fig. 6, the corresponding input and output blocks are placed next to the synapse arrays. The *AER Output* block implements an arbitration mechanism which ensures that not more than one neuron accesses the AER bus at a given time. Spike retrieval via the *AER Input* blocks and network connectivity are controlled by digital configuration logic present in every synapse. There are three different modes of operation defined by the architecture. In *direct activation mode*, the synapse is activated by AER packets with matching row and column addresses. If the *broadcast activation mode* is enabled, it also processes special AER broadcast events which are addressed row-wise. Finally, enabling the *recurrent activation mode* of a synapse placed at row j in column i will connect it to the neuron from column j.

The AER bus handles both the internal communication between the neurons on the chip and the external communication with the host computer and other peripheral devices. In addition, the *Analog Digital Converter* (ADC) circuit shown in the bottom left of Fig. 6 converts the analog currents of selected neurons and synapses to digital signals and thus allows to monitor the internal state of the chip at runtime.

Programming. Besides the analog circuitry which emulates the neural dynamics, every neuron and synapse on the chip also contains digital logic for controlling model parameters and network connectivity. The configuration options of the latter have already been explained in the last paragraph. Global parameters which apply to all neurons and synapses, respectively, are set in the *Bias Generator* which, for example, stores time and leak constants. Additional parameters in the neural circuits further allow to choose between different constants provided by the bias generator. Based on this mechanism, the chip is able to simulate up to four differently parameterized populations of neurons.

The AER protocol and the configuration interface share the same bus which implements an address space of 21 bits. The addresses available on the bus are partitioned to accommodate three types of messages. *Addressing* events correspond to AER input and are thus forwarded to the synapse arrays. *Local Configuration* events set parameters of a specific synapse or neuron circuit. Finally, global system properties like the configuration of the *Synapse De-Multiplexer* or the *Bias Generator* can be controlled by sending *Global Configuration* signals.

Unlike for the other chips presented in this paper, there is no dedicated software tool for configuring and monitoring the ROLLS neuromorphic processor. However, the authors point out that they have implemented the formal hardware specifications which allow the configuration of the chip with the *PyNCS* framework (Stefanini, Neftci, Sheik, & Indiveri, 2014). Akin to the already discussed PyNN, PyNCS is an abstraction layer interfacing high-level descriptions of neural networks with the low level configuration interfaces of simulation tools. In addition, PyNCS defines an extensible modular architecture with a generic hardware interface which can be easily augmented with support for new AER-capable neuromorphic devices. PyNCS simulations are not limited to a single chip but also support hybrid setups with different types of neuromorphic architectures. This is possible because all spikes are transmitted via the AER protocol, which enables data transmission between different chips by rather simple address space translations. With all architecture-specific functionality being hidden behind abstract high-level programming interfaces, Stefanini et al. (2014) refer to PyNCS as a microkernel for neuromorphic systems.

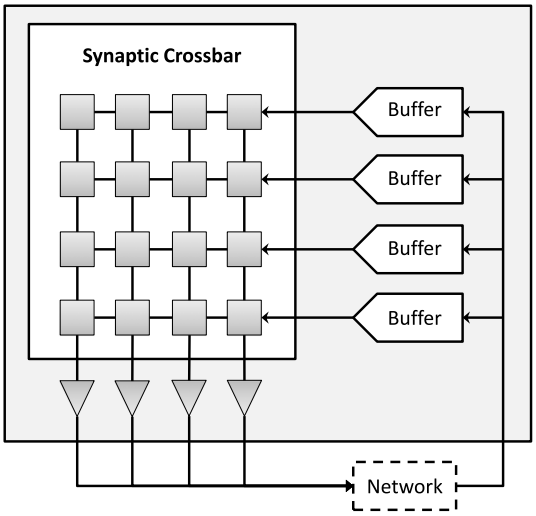

Fig. 7. A single neurosynaptic core of the IBM TrueNorth chip (Preissl et al., 2012). The *Synaptic Crossbar* contains a grid of 256 × 256 synapses. Incoming spike events are stored in buffers to simulate axonal delays. The actual neurons are depicted as triangles below the *Synaptic Crossbar*.

2.2. Digital neuromorphic chips

Analog neuromorphic hardware emulates biological neurons using physical systems driven by equivalent dynamics. It has already been pointed out that analog silicon neurons can be implemented very efficiently both in terms of the number of electronic elements, space and energy consumption. However, analog computation underlies noise and external influences like temperature and is thus not suited for experiments relying on exact and reproducible results. In the following, we present two digital approaches to neuromorphic hardware design. While the TrueNorth chip developed by IBM is based on a completely new architecture, the SpiNNaker system relies on standard Von Neumann microprocessors to keep track of the neural dynamics.

2.2.1. IBM TrueNorth

The IBM TrueNorth chip is developed as a part of the DARPA-funded SyNAPSE research program (DARPA, 2015). It is assembled from 4096 identical and independently operating neurosynaptic cores. Each of these cores implements 256 digital silicon neurons. Every of these neurons receives spikes from 256 synapses which are arranged on a crossbar as depicted in Fig. 7 (Merolla, Arthur, Alvarez-Icaza et al., 2014). Differently from the other architectures discussed so far, the focus is not on simulating the brain for research purposes but on creating an efficient tool for concrete applications (Preissl et al., 2012). Nevertheless, the chip operates in biological real-time.

Computation. The model underlying the digital neurons in the neurosynaptic cores is derived from the standard leaky integrate-and-fire neuron (Cassidy et al., 2013). Special additions include the option to configure stochastic parameter variations and different kinds of leaks increasing or decreasing the membrane voltage. All of the introduced parameters can be set individually for every neuron in a neurosynaptic core. As shown in the architecture diagram in Fig. 7, incoming spikes are forwarded via a *Synaptic Crossbar*. Every neuron receives input from 256 axons and thus has 256 synapses. However, the architecture allows only for four distinct synaptic weights per neuron. These weights are not assigned to synapses but to four different axon types. The contribution of a spike originating from axon i to the membrane potential of neuron j is computed as follows (from the

supplementary material of Merolla, Arthur, Alvarez-Icaza et al., 2014):

$$A_t(i) \cdot w_{i,j} \cdot S_j^{G_i}. \tag{3}$$

In the above equation, the bit $A_t(i)$ is set to one if a spike arrives on axon i at time t. The variable $w_{i,j}$ controls whether the synapse is active and $S_j^{G_i}$ stores the synaptic weight which applies to the type G_i assigned to axon i. The contribution of all incoming spikes to the neuron states is computed by a dedicated controller on the neurosynaptic core which stores all state information in a central memory block. State updates are triggered by a global clock running at a frequency of 1 kHz. Real-time execution is accomplished by setting the step size of the updates to 1 ms. The synchronization of state updates across all neurons and neurosynaptic cores ensures one-to-one correspondence of the performed computation with software simulations (Merolla, Arthur, Alvarez-Icaza et al., 2014).

Communication. All neurosynaptic cores of a TrueNorth chip are interconnected by a two-dimensional mesh of horizontal and vertical wires (Merolla, Arthur, Alvarez-Icaza et al., 2014). Signal transmission within this network is controlled by routers which are placed at the intersections of the horizontal and vertical pathways. Spikes generated by the neurons on a neurosynaptic core are converted to data packets storing an axonal delay, relative offsets to the destination core within the mesh and an axon address. The routers forward the packets based on the offsets using a deadlock-free dimension-order routing protocol (Merolla, Arthur, Alvarez-Icaza et al., 2014). As soon as the spike reaches its destination axon, the scheduler inserts it into a buffer. After the axonal delay time has passed, the spike is delivered to its target neurons.

Programming. The IBM TrueNorth chip is specifically targeted at commercial applications. The toolchain provided by the manufacturer is therefore considerably different from other neuromorphic platforms which are designed mainly for research purposes. Most notably, the chip is programmed using a novel *Corelet Language* (Amir et al., 2013). A *TrueNorth program*, i.e. a specific configuration of a network of neurosynaptic cores, is abstracted by so-called *corelets* which can be hierarchically composed of other corelets. Every corelet forms a closed subsystem, accessible only via input and output ports. The Corelet Language defines a grammar and abstractions for all relevant components of the hardware. Completed corelets are stored in a *Corelet Library* for later reuse. Finally, the application development is supported by a *Corelet Laboratory*, which also enables the execution of TrueNorth programs on the *Compass* simulator (Preissl et al., 2012).

2.2.2. SpiNNaker

An inherent drawback of all types of silicon neurons is limited flexibility. Their hard-wired physical implementation makes major changes impossible after chip fabrication. Adaptions or improvements require a new chip design and expensive prototyping. The *SpiNNaker* architecture solves these issues by replacing the silicon neurons with standard microprocessors. The project was originally conceived at the University of Manchester (APT Group, 2015) and is today further developed as part of the Human Brain Project. A SpiNNaker chip contains 18 ARM microprocessors, each of which is capable of simulating around 1000 spiking neurons with about 1000 synapses each in biological real-time (Painkras et al., 2012; Stromatias, Galluppi, Patterson, & Furber, 2013). The architecture allows for an integration of up to 65 536 chips (Furber et al., 2013). The following paragraphs are based on the system documentation published by Furber et al. (2014, 2013) and Xin Jin et al. (2010).

Computation. Fig. 8 gives a schematic overview of the architecture of a single SpiNNaker chip. Computations are carried out on

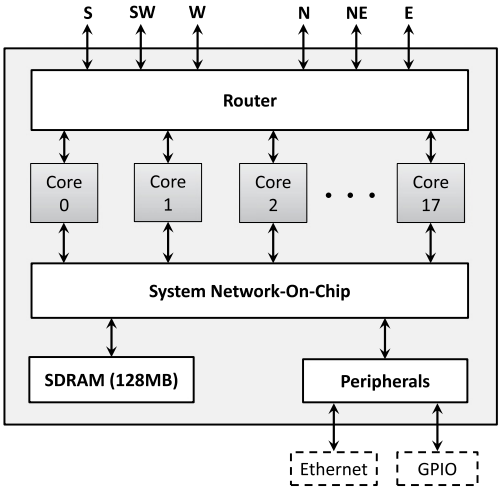

Fig. 8. Architecture of a single SpiNNaker chip (Furber et al., 2014). Every chip contains 18 cores. A *System Network-On-Chip* provides access to shared resources like a SDRAM or peripheral devices. External communication with other SpiNNaker chips is implemented by a dedicated router which connects each chip to six neighbors (see also Fig. 9).

energy-efficient embedded ARM968 multi-core microprocessors which run at a clock rate of 200 MHz. Small but very fast local memory units store instructions and data. Every ARM core is further equipped with a controller handling timers and interrupts, a DMA controller and a controller for accessing the SpiNNaker communication network. Other devices on the chip like the 128 MB shared memory are accessible via a *System Network-on-Chip*. Out of the 18 cores on a chip, only 16 are available for the execution of application code. One dedicated core is reserved for monitoring and management tasks while another core is kept as an unused spare to achieve increased fault tolerance. The use of general purpose ARM cores makes it possible to implement arbitrary neuron models. Within the limits of the system, the programmer can define whether the dynamics will be evaluated in biological real-time or at a different timescale. Special care must be taken since the system only supports fixed point arithmetic.

Communication. All chips in a SpiNNaker system are interconnected by a toroidal mesh. An illustration of the resulting topology is provided in Fig. 9. Although the toroidal connectivity pattern is fixed by the SpiNNaker architecture, the protocol implemented by the routers on the individual chips allows for the simulation of neural networks of arbitrary connectivity. This *topological virtualization* is possible because data transfers over the mesh network are extremely fast compared to the timescale of biological real-time neural dynamics. As depicted in Fig. 8, the on-chip routers have six output ports connecting a node to neighboring chips in horizontal, vertical and diagonal direction. The connections in the latter direction add additional redundancy and allow for emergency routes in case of defective links.

All spikes emitted by the neurons of a chip are forwarded to its local router. If the target neuron resides on the same chip, the spike is transmitted directly. Otherwise, it is propagated to one of the six neighboring chips. Every spike is represented by an AER packet which carries only the address of the source neuron and additional management information. All routers forward packets based on the source address which is looked up in a local routing table. The table entries also control multicasting by sending an incoming packet to more than one output port. A default route for source neuron addresses not listed in the routing table simply redirects packets on a straight line to the next output. The routing mechanism is completely implemented in hardware. It is fully asynchronous and thus non-deterministic. In particular, packets

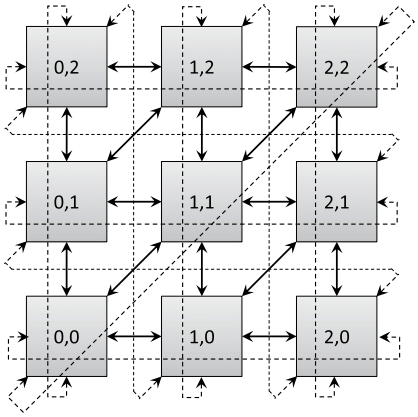

Fig. 9. Toroidal connection topology of the SpiNNaker architecture (Furber & Brown, 2009). Every numbered node corresponds to a single SpiNNaker chip. The arrows indicate the connectivity between neighboring chips. For a better overview, edges wrapping around the borders of the grid are printed in thin dashed lines.

which cannot be forwarded within a certain time-frame are discarded to avoid deadlocks (Navaridas, Miguel-Alonso, Plana, Lujan, & Furber, 2009).

Programming. Neural networks running on SpiNNaker are defined with PyNN, the same utility which is also used in the BrainScaleS project for programming the HICANN chip. The actual translation of a PyNN model to executable SpiNNaker code is implemented by an application called *PACMAN* which maps neuron populations to cores and computes the entries of the routing tables. Although the system architecture is clearly optimized for neural computation, it is possible to write arbitrary applications for SpiNNaker by programming the cores directly.

All SpiNNaker programs are built upon a basic infrastructure of system software and libraries. The dedicated monitor core present in every chip runs the *SpiNNaker Control and Monitor Program* which acts as a basic system software and implements the *SpiNNaker Datagram Protocol* (SDP). SDP packets are used for inter-chip communication and also enable data exchange over the system's Ethernet port. The actual application code is linked to the *SpiNNaker Application Runtime Kernel* (SARK). SARK provides basic hardware abstraction and a communication link to the monitor core. Finally, an optional API called *Spin1* implements a completely event driven software execution model with a priority-aware scheduler.

2.3. Neuromorphic sensors

In analogy to biological neural networks, every of the six neuromorphic hardware platforms presented in the previous sections transfers all application data solely in the form of spikes. In consequence, attaching a sensor to one of these chips – a very common use case in neurorobotic applications – requires a conversion of the sensor output to spikes. While this is definitely feasible, a more elegant solution is to use a *neuromorphic sensor* which directly mimicks the functionality of biological sensory organs. The differences between conventional and neuromorphic sensors are best highlighted at the example of silicon implementations of retinas. Unlike a standard image sensor which synchronously captures a complete image in every time step, these retinas only emit spikes to indicate relevant changes in the scenery. According to a classification made by Delbruck, Linares-Barranco, Culurciello and Posch (2010), there are four types of neuromorphic retinas. *Spatial contrast* and *spatial difference* sensors avoid spatial information redundancy by only transmitting intensity ratios and differences, respectively. Analogously, *temporal contrast* and *temporal difference* sensors only react to input changes occurring over time. For a detailed discussion of neuromorphic sensor devices, we refer the reader to the work of Liu and Delbruck (2010). A direct comparison of different types neuromorphic retina sensors is available in Delbruck et al. (2010).

3. Learning in spiking neural networks

The dynamics implemented by a neural network are defined by its connectivity and its synaptic efficacies. Learning implies the adaption of the latter ones. The optimal connectivity pattern for a given task can be inferred by evolutionary processes and genetic programming, which are both beyond the scope of this paper. However, it should be mentioned that the ROLLS neuromorphic processor can change its connectivity at runtime (Qiao et al., 2015), which is an important prerequisite for implementing an evolutionary algorithm in an autonomous robot. In this section, we provide a brief and non-exhaustive overview of learning techniques for spiking neural networks. The focus is on simple local learning rules which only rely on information available locally at every synapse and therefore allow for biologically plausible neuromorphic implementations. The second subsection briefly covers global synaptic update rules. In the last section, we discuss two methods enabling neural information processing without neuromorphic synaptic plasticity.

From a simplified abstract point of view, the goal of all learning methods presented in the following subsections is to modify the neural output produced in response to a given input spike train. Every neuron receives spikes from its presynaptic afferents. The synaptic weight w_k of a synapse k determines the contribution of an incoming spike to the neuron's membrane potential. By implicitly or explicitly adapting these weights w_k, a learning algorithm can control the properties of the produced output spike train. The actual meaning of a spike train, i.e. the information it carries, is defined by a neural code. Over time, researchers have proposed many different types of such codes. A review is available in an introductory paper by Ponulak and Kasinski (2011).

Depending on the specification of the desired output signal in the learning task defined above, one can distinguish between unsupervised learning, reinforcement learning and supervised learning. Unsupervised learning algorithms process the training data completely autonomously and do not require any external control or teaching signal. In reinforcement learning, a reward signal generated in response to previous output guides the learning process towards a desired solution. Finally, supervised learning allows for a complete specification of the desired system output by a teacher.

3.1. Local learning rules

In the following, we will provide an overview of common local learning rules which update synaptic efficacies solely based on information available locally at a neuron. For this reason, they are biologically plausible and especially suited for neuromorphic implementations.

3.1.1. Short term plasticity

One of the most basic mechanisms of synaptic plasticity observed in biological neurons is *short term plasticity* (STP). It comes in two different forms known as *habituation* and *sensitization* (Kandel & Siegelbaum, 2013). The former describes a decay of synaptic efficacy caused by repeated occurrences of a stimulus and is commonly referred to as *short-term depression*. Sensitization causes an opposite effect and is therefore also called *short-term potentiation*.

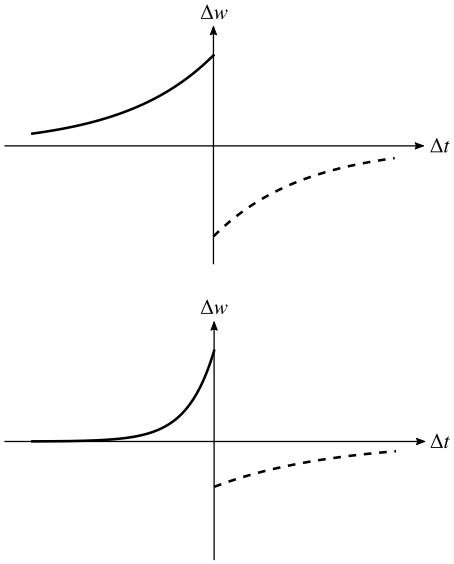

Fig. 10. Two examples for STDP windows. The solid line indicates potentiation, the dashed line depression. In the first example at the top, the parameters for both windows are identical. At the bottom, the LTP portion is slightly scaled down by a multiplicative constant and $\tau_+ < \tau_-$.

3.1.2. Spike-timing dependent plasticity

The probably most prominent and intensively studied form of local neural learning is *spike-timing dependent plasticity* (STDP) (Bi & Poo, 2001; Markram et al., 1997; Sjöström et al., 2001). STDP models changes of synaptic efficacies based on a causality relation between spikes occurring within a defined time window. If a presynaptic spike is followed by a postsynaptic one, the relationship is regarded as causal and the synapse is strengthened. Anti-causal spike sequences with a postsynaptic spike preceding a presynaptic one lead to synaptic depression. These two processes are commonly referred to as *Long-Term Potentiation* (LTP) and *Long-Term Depression* (LTD). Based on the original definition of STDP, researches have proposed additional variants which, for example, use different definitions of causality or modified learning windows. The general STDP update rule is usually formulated as follows (Morrison, Diesmann, & Gerstner, 2008):

$$\Delta w^+ = \quad F_+(w) \cdot \exp(-|\Delta t|/\tau_+) \quad \text{if } \Delta t < 0 \tag{4}$$

$$\Delta w^- = -F_-(w) \cdot \exp(-|\Delta t|/\tau_-) \quad \text{if } \Delta t \geq 0. \tag{5}$$

In the formulas above, Δt is the time difference between the presynaptic and the postsynaptic spike. The functions $F_+(\cdot)$ and $F_-(\cdot)$ model how the weight update depends on the current synaptic weight. τ_+ and τ_- are time constants which describe the width of the *STDP window*. Different examples for such windows are depicted in Fig. 10. After evaluating the STDP window, the new weight is computed as follows:

$$w \leftarrow w + \Delta w^{+/-}. \tag{6}$$

Depending on $F_+(\cdot)$ and $F_-(\cdot)$, one can distinguish between different basic types of STDP. Setting $F_+(w) = \lambda$ and $F_-(w) = \lambda\alpha$ yields *Additive STDP*. *Multiplicative STDP* weight updates are accomplished analogously by defining $F_+(w) = \lambda(1 - w)$ and $F_-(w) = \lambda\alpha w$. In both cases $w \in [0, 1)$ must hold. λ is the learning rate and α an asymmetry parameter. An extensive review of STDP-based synaptic plasticity rules is available in Morrison et al. (2008).

Reward-modulated STDP (R-STDP) extends the basic STDP mechanism with support for reinforcement learning (Legenstein, Pecevski, & Maass, 2008). In contrast to the STDP window which is only a few milliseconds wide, reward delivery can be delayed for seconds. To solve the *credit assignment problem*, pairs of spikes are stored in an eligibility trace. As soon as the reward is triggered, the synaptic update is determined as a product of the reward signal and the eligibility trace.

STDP also allows for supervised learning. *Supervised Hebbian Learning* forces neurons to produce the desired output by injecting synaptic currents during the training phase (Ponulak & Kasinski, 2011). The *Remote Supervision Method* improves on this basic mechanism by combining both a causal and an anti-causal STDP process to update synaptic efficacies (Ponulak & Kasiński, 2009).

3.1.3. Spike-driven synaptic plasticity with bistable synapses

The plasticity rule discussed in the following was originally proposed by Brader, Senn, and Fusi (2007). It is specifically designed to allow for an easy implementation with electrical circuits. Differently from the other mechanisms discussed so far, the rule is based on bistable synaptic dynamics in order to guarantee high robustness of the stored memories in the presence of random background activity in the neural network.

Unlike in the STDP protocol, changes of the synaptic weights are only triggered on the arrival of presynaptic spikes. Every synapse is *bistable*, which means that it has a depressing state J_- and a potentiating state J_+. Transitions between these two states depend stochastically on the spike history. Every spike produced by the postsynaptic neuron increases a leaky *calcium variable* $C(t)$ with slow decay. The internal state of the synapse is stored in another variable $X(t)$ which is updated on the arrival of a presynaptic spike at time t_{pre} as defined in the following (Brader et al., 2007):

$$X \to X + a \quad \text{if } V(t_{pre}) > \Theta_V \quad \text{and} \quad \Theta_u^l < C(t_{pre}) < \Theta_u^h \tag{7}$$

$$X \to X - b \quad \text{if } V(t_{pre}) \leq \Theta_V \quad \text{and} \quad \Theta_d^l < C(t_{pre}) < \Theta_d^h. \tag{8}$$

$V(\cdot)$ denotes the membrane voltage of the neuron, a and b are jump sizes, Θ_V is a voltage threshold and the remaining variables are thresholds for the calcium variable. In case no spikes arrive at the synapse, $X(t)$ converges towards one of the two stable synaptic states according to the following dynamics:

$$\dot{X} = \alpha \quad \text{if } X > \Theta_X \quad \text{and} \quad \dot{X} = -\beta \quad \text{if } X \leq \Theta_X. \tag{9}$$

The synaptic plasticity model described above is able to reproduce the STDP protocol (Brader et al., 2007). It consequently supports the supervised Hebbian learning principle. Compared to STDP, the *stop-learning conditions* defined by the thresholds and the bistability mechanism extend storage capacity and memory lifetime of spiking neural networks (Qiao et al., 2015).

3.2. Global learning rules

Global learning rules exist for both reinforcement learning and supervised learning. In this context, the term *global* refers to the fact that an implementation of the learning rule explicitly relies on external mechanisms to control the update of a neuron's synaptic efficacies. Note that this not necessarily precludes the biological plausibility of these methods.

Policy Gradient methods form a prominent class of algorithms within the field of reinforcement learning (Senn & Pfister, 2014). Synaptic efficacies are updated based on the gradient of a function predicting the future reward. *Temporal Difference learning* extends this concept by estimating not only the reward received in the next step but a weighted sum of all future expected reward signals. An extensive overview of existing algorithms is provided by Senn and Pfister (2014).

The *SpikeProp* algorithm developed by Bohte, Kok, and La Poutré (2002) is a global supervised learning rule for feedforward networks of spiking neurons with multiple layers. Being derived

from the classical backpropagation algorithm for artificial neural networks, it updates synaptic efficacies using a biologically implausible gradient update rule. Biological plausibility also does not apply to the *PBSNLR* algorithm which converts spiking neurons to perceptrons and thereby transforms the problem of supervised spike sequence learning to a classification task (Xu, Zeng, & Zhong, 2013). The *Finite Precision* learning algorithm proposed by Memmesheimer, Rubin, Ölveczky and Sompolinsky (2014) is able to reproduce spike patterns within given tolerance windows. The authors argue for the biological plausibility of their approach but also state the necessity of an external control which means that the learning rule cannot be implemented locally.

3.3. Static networks

Neural networks can also perform useful computations without supporting synaptic plasticity. The *Reservoir Computing* paradigm (Lukoševičius & Jaeger, 2009) leverages the dynamics of recurrent neural networks to project input data into a complex high dimensional space. The synaptic efficacies of the reservoir neurons are fixed. The actual output is computed by adjusting the connection weights of readout units which are connected to the reservoir neurons with an appropriate learning rule. Neuromorphic hardware can be used to speed up the complex simulation of the reservoir dynamics while the adaption of the readout units' weights is performed on a host computer.

Another way of avoiding the implementation of synaptic plasticity is to use the methodology developed by Stewart and Eliasmith (2014). The theoretical core of their proposed approach is formed by the *Neural Engineering Framework* (NEF) which defines a mapping from vectors, functions and differential equations to corresponding neuron populations, synaptic weights and network connections. Thus, the NEF can be used to directly synthesize a given algorithm which is based on these compute primitives to a corresponding spiking neural network. Learning is thereby replaced by a direct analytical derivation of synaptic weights. Advanced cognitive functions beyond purely numerical computation can be realized using the *Semantic Pointer Architecture* (SPA) which allows to store and modify syntactically structured data using the NEF compute primitives. The construction and simulation of neural networks based on NEF and SPA is supported by the dedicated software tool *Nengo*.

4. Learning with neuromorphic hardware

Section 2 gave an introduction to six selected neuromorphic hardware platforms. In the previous section, we provided an overview over different algorithms available for learning in networks of spiking neurons. In the next subsections, we will link these findings by analyzing which of the discussed learning techniques can be implemented on the HICANN chip from the BrainScaleS project, the HRL SyNAPSE chip, the ROLLS neuromorphic processor, the TrueNorth Chip developed by IBM and the SpiNNaker platform. Note that we only review learning mechanisms which are implemented on-chip to run online during the execution of the network. Offline weight updates can performed on any neuromorphic platform with configurable synaptic weights by iteratively executing the network, computing offline weight updates on a host computer and transferring them back to the chip. For example, such an approach has been realized by Schmuker, Pfeil, and Nawrot (2014) on a previous version of the HICANN chip.

Neurogrid is not included in the following discussion since the platform does currently not implement any form of synaptic plasticity. As pointed out by Benjamin et al. (2014), this is due to the shared dendrite architecture which spreads synaptic input across the dendritic tree to neighboring neurons and thus does not allow for an adaptation of individual synaptic weights. However, the authors emphasize that Neurogrid in principle supports a shared synapse architecture with individual synaptic weights which are stored in a RAM on the daughter board and thus could be adapted via a learning rule. It is therefore not unlikely that a future release of Neurogrid will implement synaptic plasticity. Currently, without any on-chip learning mechanisms available, Neurogrid can be used as an accelerator for reservoir computing or must be programmed with the Neural Engineering Framework (Choudhary et al., 2012).

4.1. Synaptic plasticity circuits on the HICANN chip

The ANCs of the BrainScaleS wafer-scale hardware platform support both STP and STDP. The former is implemented by partitioning the efficacy of a synapse into an active and an inactive portion (Schemmel, Bruderle, Meier, & Ostendorf, 2007). For every spike transmitted by the synapse, the inactive portion is increased. At the same time, a continuously running recovery process shifts efficacy back to the active portion. This mechanism allows for the implementation of both habituation and sensitization by scaling the maximum synaptic conductance proportional to the active portion or the inactive portion, respectively. The electronic circuits implementing this functionality are documented in Schemmel et al. (2007) and work for both excitatory and inhibitory synaptic connections.

All synapses of the HICANN chip also implement long-term synaptic plasticity based on the STDP protocol. To reduce the required electronic circuitry, the plasticity logic is split into an analog part locally available at every synapse and a *Digital STDP Update Control* logic which is time-multiplexed among all synapses of an array (see also Fig. 1) (Pfeil et al., 2012). Each synapse has access to its presynaptic spikes. Since all spikes emitted by a postsynaptic neuron are propagated back to the synapse array (Schemmel et al., 2008), it can keep full track of all spike timings relevant for STDP. Two separate capacitors measure causal and anti-causal STDP modifications (Schemmel, Grubl, Meier, & Mueller, 2006). At the arrival of a presynaptic spike, the capacitor representing causal spike relationships is charged. Due to a leak current, this charge decays exponentially, which corresponds to the exponential functions in Eqs. (4) and (5). On emission of a postsynaptic spike, the remaining charge in the capacitor is accumulated. A separate circuit handles anti-causal STDP events analogously. When the STDP update controller triggers a weight update for a certain synapse, the accumulated causal and anti-causal portions are compared. Based on a configurable threshold criterion, the synaptic weight is potentiated or depressed. Instead of computing the weight-dependent scaling factors $F_+(\cdot)$ and $F_-(\cdot)$ explicitly, the actual weight updates are determined using a programmable lookup table (Schemmel et al., 2007). The new weight is then converted to a 4-bit value and stored in the synapse. It is important to note that the use of a shared controller for weight updates is only possibly because the dynamics of STDP-induced synaptic plasticity are slow compared to the dynamics of the neurons (Pfeil et al., 2012).

By programming the lookup table of the STDP update controller, it is possible to reproduce many different types of update rules like the additive and multiplicative ones introduced earlier. Similarly, by flipping signs, it is also possible to realize anti-causal STDP. However, due to the implementation with analog circuitry, it is not possible to change the exponential shape of the STDP windows.

4.2. The STDP protocol of the HRL SyNAPSE chip

All synapses of the HRL SyNAPSE chip are excitatory and support basic additive STDP (Cruz-Albrecht et al., 2012). The user can configure the width of the STDP window by setting the time constants τ_+ and τ_-. $F_+(\cdot)$ and $F_-(\cdot)$ are realized as constant functions

with configurable values A_+ and A_-. Similarly to the HICANN chip, the explicit calculation and storage of timing differences between pairs of spikes is avoided by continuous accumulation of depression D and potentiation P (Cruz-Albrecht et al., 2012):

$$\tau_- \cdot \dot{D} = -D \qquad \tau_+ \cdot \dot{P} = -P. \tag{10}$$

On the arrival of a spike, the state variables are updated according to the equations below:

$$D \leftarrow D + A_- \qquad P \leftarrow P + A_+. \tag{11}$$

At the end of every STM timeslot the actual synaptic update is computed as

$$\Delta w = P - D \tag{12}$$

and then applied to the current value in the memristor array.

The update logic described above is implemented using three integrator circuits. Two of them are leaky and accumulate the incoming pre- and postsynaptic spikes according to Eqs. (10) and (11). They are connected to a third integrator which corresponds to Eq. (12). Compared to the STDP logic of the HICANN chip which allows for the configuration of arbitrary weight-dependent update rules via lookup tables, the mechanism implemented in the HRL architecture is rather restricted since $F_+(\cdot)$ and $F_-(\cdot)$ are constant by design.

4.3. STP and STDP mechanisms implemented in the ROLLS neuromorphic processor

As depicted in Fig. 6, the ROLLS chip contains two arrays with 256×256 synapses each. Those in the upper array marked with S implement a short-term depression plasticity mechanism which can be activated if the corresponding synapse is operating in excitatory mode (Qiao et al., 2015). In the initial state, the weight of the synapse is completely determined by the stored 2-bit weight. Internally, this weight emulated by a current. On spike arrival a capacitor in the STP circuit is discharged by a configurable amount and the current decreases. Spikes which arrive before the capacitor voltage has recovered are consequently transmitted with a lower effective synaptic weight and cause the capacitor to discharge even further. This behavior exactly reproduces the desired STP dynamics.

The second array contains synapses marked with L which implement the spike-driven bistable synapse plasticity rule from Section 3.1.3. All relevant model parameters like the jump size and the bistable drift threshold can be globally programmed by the user in the *Bias Generator*. The value of the calcium variable is computed in the corresponding neuron circuit and provided to every synapse. Signals from the neural circuits also control jump blocks in the synapses which increase and decrease the synaptic state variable by the given jump sizes. Another logic block implements the bistable synaptic dynamics depending on the current value of the state variable and the drift threshold. Although the learning rule supports STDP, the underlying mechanisms considerably differ from the STDP-based plasticity circuits of the other chips considered so far. A direct comparison is therefore not possible. However, the bistable synapse model has proven as a highly capable learning mechanism in different experimental setups (Brader et al., 2007; Qiao et al., 2015).

4.4. Extensions for the TrueNorth Chip

The current design of the IBM TrueNorth Chip does not realize any synaptic plasticity mechanisms. The complete functionality must be programmed using the Corelet Language. However, Seo et al. (2011) proposed a possible extension to the chip design which implements a probabilistic digital version of STDP where every synapse is set to either 0 or 1. The main difference to the TrueNorth architecture is an extended digital neuron circuit which contains additional circuitry realizing the synaptic plasticity. Additionally, *transposable* SRAM cells allow for equally fast row-wise and column-wise access of the synaptic array during weight updates.

In the extended neuron circuit, the output of the digital leaky integrate-and-fire model is forwarded to an integrated STDP circuit which is shared among all synapses connected to the neuron. Two 8-bit counters C^+ and C^- measure the time elapsed since the last presynaptic and postsynaptic spike, respectively. On spike emission, C^+ and C^- are set to programmable values C^+_{set} and C^-_{set}. The counters then decay with configurable step sizes of C^+_{decay} and C^-_{decay} in every time step. When a presynaptic update is triggered, the considered neuron checks the C^+ counter values of all its presynaptic afferents. To enable probabilistic updates, each value is compared to random number. Depending on the outcome of this comparison, the corresponding weight is set to a configurable 1-bit value (i.e. 0 or 1) or left unchanged. Additional parameters allow for special actions in case the counter is zero. Postsynaptic weight updates are performed analogously by considering the counters C^- of the target neurons.

The learning mechanism described above can be configured to reproduce both an STDP and an anti-STDP protocol. However, compared to the other neuromorphic STDP-implementations considered so far, there is no close correspondence to experimental results or physiological processes.

4.5. Implementing STDP on SpiNNaker

The general purpose ARM microprocessors in the SpiNNaker chips enable the implementation of arbitrary local and global learning rules. Of course, the system also supports alternative approaches based on fixed synaptic weights. For example, the Neural Engineering Framework and Nengo (Galluppi, Davies, Furber, Stewart, & Eliasmith, 2012) are available on SpiNNaker. The actual challenge lies in an efficient use of the available system resources to achieve maximum performance. This becomes clear when considering the STDP implementation developed by Jin, Rast, Galluppi, Khan and Furber (2009).

On SpiNNaker, all synapses of a neuron group are stored in a dedicated memory region called the *synapse block*. Every row of this memory corresponds to a single synapse with a synaptic delay, a postsynaptic index and a synaptic weight. Depending on whether this storage is indexed by the presynaptic or the postsynaptic neuron, long term depression or long term potentiation can be implemented efficiently but not both at the same time. For this reason, STDP updates are only triggered by presynaptic spikes to allow for efficient memory access and caching mechanisms. Since the postsynaptic spikes required for the STDP update are not yet available at the arrival time of a presynaptic spike, the event handling of this spike is deferred to the future. This is possible because the communication mechanism of SpiNNaker operates at a much faster timescale than the dynamics of the simulated neurons. As soon as all spikes fitting into the current STDP learning window have arrived, the update of the synaptic weights is initiated. This mechanism ensures that the synapse block is always accessed in an efficient way.

Among all the neuromorphic platforms considered in this paper, SpiNNaker is the most versatile one in terms synaptic plasticity rules. Since the neural simulation is executed on general purpose ARM CPUs, it is possible to implement arbitrary learning rules. In the particular case of STDP, the system can simulate learning windows of any shape and arbitrary weight update rules including the additive and multiplicative ones discussed in Section 3.1.2. However, it is important to keep in mind that inefficient implementations of these rules directly affect the performance of the system and might render neural network simulation in biological real-time unfeasible.

Table 1
Overview of the main features of the different neuromorphic hardware architectures presented in this work. All data is taken from the sources referenced in the corresponding sections of the text. Varying numbers of neurons and synapses result from configurable chip topologies. In the column indicating the execution speed of the neural network, BRT refers to biological real-time. Note that the numbers on the power consumption cannot be compared directly between chips due to different measurement procedures. There is no data available in the literature on the power consumption of a single HICANN chip. The wafer-scale system with 384 chip consumes 1 kW of power.

	Architecture	Type	Neurons	Synapses/neuron	Learning rules	Speed	Power
2.1.1	HICANN	Analog + AER	1–512	224–14 336	STP, STDP	10^3–10^5 BRT	n/a
2.1.2	Neurogrid	Analog + AER	1 048 576	$6 \cdot 10^9$ in total	None	BRT	2.7 W
2.1.3	HRL SyNAPSE	Analog + STM	576	128	STDP	BRT	130 mW
2.1.4	ROLLS	Analog + AER	1–256	512 (+2)–131 072	STP, STDP	BRT	4 mW
2.2.1	IBM TrueNorth	Digital	1 048 576	256	None	BRT	63 mW
2.2.2	SpiNNaker	ARM CPU + AER	1 000	1 000	Any	BRT	1 W

5. Neuromorphic chips for robotic applications

So far, the neuromorphic architectures presented in this work have been reviewed mainly in terms of their technical realization. In this section, we will discuss their potential of advancing the field of autonomous robotics. As a guide, the most relevant features and properties of all chips are summarized in Table 1. Probably one of the most pressing issues which should be addressed first is whether all of the presented chips operate precisely enough to reliably simulate the neural network dynamics. This question particularly arises because of the low synaptic weight resolution of only a few bits implemented by the HICANN chip, the HRL SyNAPSE chip and the ROLLS neuromorphic processor. However, an experimental study performed by Pfeil et al. (2012) used a benchmark problem to demonstrate that both low-resolution synaptic weight storage and inaccuracies caused by chip manufacturing processes are uncritical.

As can be seen easily, all listed architectures execute neural networks at very low power consumption, which makes them ideally suited for mobile autonomous robotics. The only exception is the HICANN chip which is only available as a wafer-scale system with a power consumption of 1 kW. However, the extremely high execution speed renders this system inappropriate for real-time applications in robotics anyway. Instead, a very promising use case for systems of this type are off-line simulations for the evaluation of neural network topologies and the optimization of synaptic connectivity. The results of these simulations can then be used to set the parameters of one of the real-time capable chips.

Another huge advantage of neuromorphic chips is their inherent support for massively parallel information processing, which makes them highly capable of handling multi-modal sensory input efficiently and responsively. For example, Qiao et al. (2015) have demonstrated the feasibility of implementing an image classification algorithm on the highly efficient ROLLS neuromorphic processor based on input signals from a silicon retina. Similarly, in another study, the SpiNNaker system enabled the control of a small mobile robot by presentation of visual cues (Furber et al., 2014). These examples clearly prove that neuromorphic chips have a huge potential of efficiently implementing algorithms from computer vision which could previously only be executed on workstations. In recent years, the power of these workstations has increased drastically with the advent of powerful GPUs with general purpose programming interfaces for scientific computing tasks, bringing huge momentum to the field of *deep learning* in artificial neural networks (Raina, Madhavan, & Ng, 2009). In the field of spiking neural networks, GPU acceleration is implemented by the simulator *CARLsim* (Richert, Nageswaran, Dutt, & Krichmar, 2011) using NVIDIA CUDA (NVIDIA Corporation, 2015). In a study by Richert et al. (2011), CARLsim was able to simulate a cortical network with 138 K neurons and 30 M synapses slightly faster than real-time. These results render GPUs a strong competitor for neuromorphic hardware designs. However, they neither offer the compact size of a small SpiNNaker system nor the high maximum speedup of the HICANN chip. Moreover, most neuromorphic designs are based on highly modular architectures and can be easily extended to support the simulation of larger networks. Nevertheless, as pointed out by Krichmar, Coussy and Dutt (2015), GPUs can be a useful tool for easily testing neural network designs before running them on neuromorphic chips.

Besides power-efficient operation and parallel information processing, neuromorphic hardware can also simplify the programming of robots. With neural networks being very general and abstract descriptions of system behavior, they can be easily ported between different platforms without having to pay attention to low-level details of underlying processors or operating systems. Moreover, the AER protocol enables seamless integration of other neuromorphic hardware. Finally, two even more important arguments in favor of employing neural networks and neuromorphic chips in autonomous robots are their inherent support for learning and their proven fault tolerance. In the SpiNNaker architecture, for example, the latter aspect is addressed by reserving one of the 18 cores in each chip as a spare (Furber et al., 2013). Additionally, the redundant toroidal connection topology adds fault tolerance to the communication system. Both of these two special architectural properties enable the system to compensate for errors occurring over its lifetime, which is another notable advantage over GPUs.

6. Conclusion & outlook

In this work, we have reviewed how neurobiological learning is implemented on neuromorphic hardware designs which enable the efficient execution of large-scale spiking neural networks with millions of neurons. Neurobiological learning is essential in the emerging field of neurorobotics in order to achieve a close correspondence to biological nervous systems.

In a first step, we presented six different neuromorphic architectures based on analog and digital computation. The HICANN chip developed by the BrainScaleS project implements analog silicon neurons. The focus of the design clearly lies on maximum configuration flexibility. At the same time, the system runs up to 10^5 times faster compared to biological real-time, making it especially suited for extensive experimental studies. Neurogrid is another analog neuromorphic hardware which is composed of 16 individual Neurocores. Unlike the HICANN chip, these cores run in biological real-time and are thus suited for neurorobotics. The system is optimized for extremely energy-efficient operation. As it turned out, this comes at the price of less flexibility since maximum performance is only achieved for layered neural network architectures. In a recent publication by Menon, Fok, Neckar, Khatib, and Boahen (2014), Neurogrid was used to implement a robot controller. The HRL SyNAPSE chip uses memristors to store synaptic weights. It is not based on the AER protocol but incorporates STM and thus avoids the need for large arrays of synaptic circuits. The most notable feature of the ROLLS neuromorphic processor is its support for a bistable synapse dynamics model which has proven to reliably retain memories even in the presence of

background noise. The TrueNorth chip from IBM is based on digital silicon neurons. However, it is mainly designed for commercial applications and thus only implements very abstract neuron and synapse models. Moreover, there is no support for multicast routing. The SpiNNaker architecture achieves maximum flexibility regarding the types of supported neuron models and network topologies by interconnecting standard ARM microprocessors in an efficient event-based toroidal routing mesh.

The second part of this work provided an overview of different types of learning rules for spiking neural networks. A conceptual distinction was made between local methods which rely completely on information available at every neuron and global methods which require additional external input. Almost all of the local rules considered in this review are based on STDP. However, we also reviewed a spike-based plasticity rule with bistable synaptic dynamics which was specifically designed for neuromorphic hardware. Global rules are much more diverse and often involve the computation of gradients. The Neural Engineering Framework was introduced as an alternative approach for implementing neural networks without learning by precomputing fixed synaptic weights.

Based on the results from the previous sections, we illustrated in Section 4 how the neuromorphic chips presented at the beginning implement neurobiological learning. Although the Neurogrid platform is in principle capable of adapting synaptic weights, it does currently not support any type of synaptic plasticity and therefore requires precomputed weights. This also applies to the IBM TrueNorth chip. Like in the case of Neurogrid, future releases could be extended to support on-chip learning via STDP. The HICANN chip implements both STP and STDP. Differently from the STDP protocol of the HRL SyNAPSE chip, it relies on lookup tables to compute weight-dependent synaptic updates and therefore supports many different update rules. While the ROLLS neuromorphic processor also supports STP, its most interesting feature is a spike-driven plasticity rule based on bistable synapses. The SpiNNaker system turned out to be the most versatile platform in terms of learning since it can be programmed to support arbitrary plasticity rules. However, we illustrated at the example of an existing STDP update rule that efficient implementations are not trivial and require additional considerations.

In the last section, we finally put our results of the review into the context of robotics and argued that the low power consumption and the massively parallel processing of neuromorphic chips has not only the potential of advancing neurorobotics but the field of autonomous robotics in general. Table 1 can serve as a starting point for the selection of an appropriate neuromorphic architecture for the targeted application. From a theoretical perspective, the maximum number of simulated neurons, synapses and spikes as well as simulation speed and power consumption seem to be sufficient to completely characterize the performance of a neuromorphic system. However, representative benchmarks could definitely contribute to a more practically oriented performance estimation. To the best knowledge of the authors, such a standardized benchmark suite for neuromorphic hardware does not exist yet. An initial step towards this goal has been made by Brüderle et al. (2011) who defined a set of model experiments to support the development of the HICANN chip. Currently, the definition of standardized neuromorphic benchmarks and corresponding datasets is an active topic of discussion in the neuromorphic research community (Rast et al., 2015). However, the huge diversity of different approaches and paradigms in neuromorphic hardware designs makes it difficult to benchmark different platforms consistently. In any case, as stated by Rast et al. (2015), a neuromorphic benchmark suite should focus on problems with constraints on time and power to clearly demonstrate the advantages of neuromorphic chip designs over conventional hardware architectures.

Future releases of the reviewed neuromorphic chips will benefit from both new findings in neuroscience as well as novel manufacturing processes. From a neurorobotics point of view, it is very desirable that all architectures support biologically plausible mechanisms of synaptic plasticity. Only then detailed and realistic simulations of the brain will be possible. The SpiNNaker system with its freely programmable learning rules is a first step in this direction. In any case, neuromorphic chips are an enabling technology for neurorobotics and allow researchers to investigate the neural foundations of complex cognitive functions and processes.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007–2013) under grant agreement no. 604102 (HBP).

References

Amir, A., Datta, P., Risk, W.P., Cassidy, A.S., Kusnitz, J.A., & Esser, S.K. et al. (2013). Cognitive computing programming paradigm: A Corelet Language for composing networks of neurosynaptic cores. In *2013 international joint conference on neural networks. IJCNN 2013 - Dallas* (pp. 1–10).

APT Group, 2015. SpiNNaker: Project Description. URL: http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project/.

Bartolozzi, C., & Indiveri, G. (2007). Synaptic dynamics in analog VLSI. *Neural Computation*, *19*(10), 2581–2603.

Benjamin, B. V., Gao, P., McQuinn, E., Choudhary, S., Chandrasekaran, A. R., Bussat, J.-M., et al. (2014). Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, *102*(5), 699–716.

Bi, G.-Q., & Poo, M.-M. (2001). Synaptic modification by correlated activity: Hebb's postulate revisited. *Annual Review of Neuroscience*, *24*(1), 139–166.

Bishop, C. M. (2006). *Information science and statistics*, *Pattern recognition and machine learning*. New York: Springer.

Boahen, K. A. (2000). Point-to-point connectivity between neuromorphic chips using address events. *IEEE Transactions on Circuits and Systems Part II: Analog and Digital Signal Processing*, *47*(5), 416–434.

Boahen, K. (2015). Brains in silicon: Project homepage. URL: http://web.stanford.edu/group/brainsinsilicon/.

Bohte, S. M. (2004). The evidence for neural information processing with precise spike-times: A survey. *Natural Computing*, *3*(2), 195–206.

Bohte, S. M., Kok, J. N., & La Poutré, H. (2002). Error-backpropagation in temporally encoded networks of spiking neurons. *Neurocomputing*, *48*(1–4), 17–37.

Brader, J. M., Senn, W., & Fusi, S. (2007). Learning real-world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Computation*, *19*(11), 2881–2912.

Brette, R., & Gerstner, W. (2005). Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *Journal of Neurophysiology*, *94*(5), 3637–3642.

Brüderle, D., Müller, E., Davison, A. P., Muller, E., Schemmel, J., & Meier, K. (2009). Establishing a novel modeling tool: a python-based interface for a neuromorphic hardware system. *Frontiers in Neuroinformatics*, *3*.

Brüderle, D., Petrovici, M., Vogginger, B., Ehrlich, M., Pfeil, T., Millner, S., et al. (2011). A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. *Biological Cybernetics*, *104*(4–5), 263–296.

Cassidy, A.S., Merolla, P., Arthur, J.V., Esser, S.K., Jackson, B., & Alvarez-Icaza, R. et al. (2013). Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In *2013 international joint conference on neural networks. IJCNN 2013—Dallas* (pp. 1–10).

Choudhary, S., Sloan, S., Fok, S., Neckar, A., Trautmann, E., Gao, P., et al. (2012). Silicon neurons that compute. In A. Villa, W. Duch, P. Érdi, F. Masulli, & G. Palm (Eds.), *Lecture notes in computer science: Vol. 7552. Artificial neural networks and machine learning—ICANN 2012* (pp. 121–128). Berlin, Heidelberg: Springer.

Cruz-Albrecht, J. M., Derosier, T., & Srinivasa, N. (2013). A scalable neural chip with synaptic electronics using CMOS integrated memristors. *Nanotechnology*, *24*(38), 384011.

Cruz-Albrecht, J. M., Yung, M. W., & Srinivasa, N. (2012). Energy-efficient neuron, synapse and STDP integrated circuits. *IEEE Transactions on Biomedical Circuits and Systems*, *6*(3), 246–256.

DARPA, 2015. Systems of neuromorphic adaptive plastic scalable electronics (SYNAPSE): Project homepage. URL: http://www.darpa.mil/Our_Work/DSO/Programs/Systems_of_Neuromorphic_Adaptive_Plastic_Scalable_Electronics_(SYNAPSE).aspx.

Delbruck, T., Linares-Barranco, B., Culurciello, E., & Posch, C. (2010). Activity-driven, event-based vision sensors. In *2010 IEEE international symposium on circuits and systems—ISCAS 2010* (pp. 2426–2429).

Fieres, J., Schemmel, J., & Meier, K. (2008). Realizing biological spiking network models in a configurable wafer-scale hardware system. In *2008 IEEE international joint conference on neural networks. IJCNN 2008—Hong Kong* (pp. 969–976).

Floreano, D., Ijspeert, A. J., & Schaal, S. (2014). Robotics and neuroscience. *Current Biology*, *24*(18), R910–R920.

Furber, S., & Brown, A. (2009). Biologically-inspired massively-parallel architectures—computing beyond a million processors. In *Ninth international conference on Application of concurrency to system design, 2009. ACSD'09*.

Furber, S. B., Galluppi, F., Temple, S., & Plana, L. A. (2014). The SpiNNaker project. *Proceedings of the IEEE*, *102*(5), 652–665.

Furber, S. B., Lester, D. R., Plana, L. A., Garside, J. D., Painkras, E., Temple, S., et al. (2013). Overview of the SpiNNaker system architecture. *IEEE Transactions on Computers*, *62*(12), 2454–2467.

Galluppi, F., Davies, S., Furber, S., Stewart, T., & Eliasmith, C. (2012). Real time on-chip implementation of dynamical systems with spiking neurons. In *2012 international joint conference on neural networks. IJCNN 2012—Brisbane* (pp. 1–8).

Gerstner, W., & Kistler, W. M. (2002). *Spiking neuron models: single neurons, populations, plasticity*. Cambridge, UK, New York: Cambridge University Press.

Graf, H. P., Jackel, L. D., & Hubbard, W. E. (1988). VLSI implementation of a neural network model. *Computer*, *21*(3), 41–49.

Heemskerk, J. N. H. (1995). *Overview of neural hardware: Neurocomputers for brain-style processing, design, implementation and application*. (Ph.D. thesis), Leiden University.

Indiveri, G., & Fusi, S. (2007). Spike-based learning in VLSI networks of integrate-and-fire neurons. In *2007 IEEE international symposium on circuits and systems* (pp. 3371–3374).

Indiveri, G., Linares-Barranco, B., Hamilton, T. J., van Schaik, A., Etienne-Cummings, R., Delbruck, T., et al. (2011). Neuromorphic silicon neuron circuits. *Frontiers in Neuroscience*, *5*.

Jeltsch, S. (2014). *A scalable workflow for a configurable neuromorphic platform*. (Ph.D. thesis), Universität Heidelberg.

Jin, X., Rast, A., Galluppi, F., Khan, M., & Furber, S. (2009). Implementing learning on the SpiNNaker universal neural chip multiprocessor. In C. Leung, M. Lee, & J. Chan (Eds.), *Lecture notes in computer science: Vol. 5863. Neural information processing* (pp. 425–432). Berlin, Heidelberg: Springer.

Kandel, E. R., & Siegelbaum, S. A. (2013). Cellular mechanisms of implicit memory storage and the biological basis of individuality. In E. R. Kandel, J. H. Schwartz, T. M. Jessel, S. A. Siegelbaum, & A. J. Hudspeth (Eds.), *Principles of neural science* (pp. 1461–1486). New York: McGraw-Hill.

Krichmar, J. L., Coussy, P., & Dutt, N. (2015). Large-scale spiking neural networks using neuromorphic hardware compatible models. *ACM Journal on Emerging Technologies in Computing Systems*, *11*(4), 36:1–36:18.

Legenstein, R., Pecevski, D., & Maass, W. (2008). A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. *PLoS Computational Biology*, *4*(10), e1000180.

Liu, S.-C., & Delbruck, T. (2010). Neuromorphic sensory systems. *Sensory Systems*, *20*(3), 288–295.

Lukoševičius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, *3*(3), 127–149.

Maass, W. (1997). Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671.

Markram, H., Lübke, J., Frotscher, M., & Sakmann, B. (1997). Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, *275*(5297), 213–215.

Meier, K. (2013). The FACETS project: Project homepage. URL: http://www.facets-project.org/.

Meier, K. (2015). BrainScaleS: Project homepage. URL https://brainscales.kip.uni-heidelberg.de/.

Memmesheimer, R.-M., Rubin, R., Ölveczky, B. P., & Sompolinsky, H. (2014). Learning precisely timed spikes. *Neuron*, *82*(4), 925–938.

Menon, S., Fok, S., Neckar, A., Khatib, O., & Boahen, K. (2014). Controlling articulated robots in task-space with spiking silicon neurons. In *2014 5th IEEE RAS & EMBS international conference on biomedical robotics and biomechatronics*.

Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R., & Modha, D.S. (2011). A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *2011 IEEE custom integrated circuits conference—CICC 2011* (pp. 1–4).

Merolla, P., Arthur, J., Alvarez, R., Bussat, J.-M., & Boahen, K. (2014a). A multicast tree router for multichip neuromorphic systems. *IEEE Transactions on Circuits and Systems. I. Regular Papers*, *61*(3), 820–833.

Merolla, P. A., Arthur, J. V., Alvarez-Icaza, R., Cassidy, A. S., Sawada, J., Akopyan, F., et al. (2014b). A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, *345*(6197), 668–673.

Minkovich, K., Srinivasa, N., Cruz-Albrecht, J. M., Cho, Youngkwan, & Nogin, A. (2012). Programming time-multiplexed reconfigurable hardware using a scalable neuromorphic compiler. *IEEE Transactions on Neural Networks and Learning Systems*, *23*(6), 889–901.

Morrison, A., Diesmann, M., & Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biological Cybernetics*, *98*(6), 459–478.

Mortara, A., & Vittoz, E. A. (1994). A communication architecture tailored for analog VLSI artificial neural networks: Intrinsic performance and limitations. *IEEE Transactions on Neural Networks*, *5*(3), 459–466.

Murray, A. F., Del Corso, D., & Tarassenko, L. (1991). Pulse-stream VLSI neural networks mixing analog and digital techniques. *IEEE Transactions on Neural Networks*, *2*(2), 193–204.

Murray, A. F., & Smith, A. V. W. (1987). A novel computational and signalling method for VLSI neural networks. In D. Seitzer (Ed.), *ESSCIRC'87, thirteenth european solid-state circuits conference* (pp. 19–22). Berlin: VDE-Verlag.

Navaridas, J., Miguel-Alonso, J., Plana, L. A., Lujan, M., & Furber, S. (2009). Understanding the interconnection network of SpiNNaker. In *Proceedings of the 23rd international conference on Supercomputing* (pp. 286–295). Yorktown Heights, NY, USA: ACM.

NVIDIA Corporation (2015). Parallel programming and computing platform — CUDA. URL: http://www.nvidia.com/object/cuda_home_new.html.

Painkras, E., Plana, L.A., Garside, J., Temple, S., Davidson, S., Pepper, J., Clark, D., Patterson, C., & Furber, S. (2012). SpiNNaker: A multi-core System-on-Chip for massively-parallel neural net simulation. In *2012 IEEE custom integrated circuits conference. CICC*.

Pfeil, T., Potjans, T. C., Schrader, S., Potjans, W., Schemmel, J., Diesmann, M., et al. (2012). Is a 4-bit synaptic weight resolution enough?—Constraints on enabling spike-timing dependent plasticity in neuromorphic hardware. *Frontiers in Neuroscience*, *6*.

Ponulak, F., & Kasiński, A. (2009). Supervised learning in spiking neural networks with ReSuMe: Sequence learning, classification, and spike shifting. *Neural Computation*, *22*(2), 467–510.

Ponulak, F., & Kasinski, A. (2011). Introduction to spiking neural networks: Information processing, learning and applications. *Acta Neurobiologiae Experimentalis*, *71*(4), 409–433.

Preissl, R., Wong, T. M., Datta, P., Flickner, M., Singh, R., & Esser, S. K. (2012). Compass: A scalable simulator for an architecture for cognitive computing. In *2012 SC—international conference for high performance computing, networking, storage and analysis* (pp. 1–11).

Qiao, N., Mostafa, H., Corradi, F., Osswald, M., Stefanini, F., Sumislawska, D., et al. (2015). A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Frontiers in Neuroscience*, *9*.

Raina, R., Madhavan, A., & Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, ICML'09. (pp. 873–880). New York, NY, USA: ACM.

Rast, A., Alibart, F., Azghadi, M.R., Bamford, S., Bennett, C., & Celiker, O. et al. (2015). Benchmarking neuromorphic systems: Discussion group. In *The 2015 CapoCaccia cognitive neuromorphic engineering workshop*.

Renaud, S., Tomas, J., Bornat, Y., Daouzli, A., & Saighi, S. (2007). Neuromimetic ICs with analog cores: an alternative for simulating spiking neural networks. In *2007 IEEE international symposium on circuits and systems* (pp. 3355–3358).

Richert, M., Nageswaran, J. M., Dutt, N., & Krichmar, J. L. (2011). An efficient simulation environment for modeling large-scale cortical processing. *Frontiers in Neuroinformatics*, *5*, 19.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408.

Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., & Millner, S. (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE international symposium on circuits and systems—ISCAS 2010* (pp. 1947–1950).

Schemmel, J., Bruderle, D., Meier, K., & Ostendorf, B. (2007). Modeling synaptic plasticity within networks of highly accelerated I&F neurons. In *2007 IEEE international symposium on circuits and systems* (pp. 3367–3370).

Schemmel, J., Fieres, J., & Meier, K. (2008). Wafer-scale integration of analog neural networks. In *2008 IEEE international joint conference on neural networks. IJCNN 2008—Hong Kong* (pp. 431–438).

Schemmel, J., Grubl, A., Meier, K., & Mueller, E. (2006). Implementing synaptic plasticity in a VLSI Spiking Neural Network Model. In *International joint conference on neural networks, 2006. IJCNN'06*.

Schmuker, M., Pfeil, T., & Nawrot, M. P. (2014). A neuromorphic network for generic multivariate data classification. *Proceedings of the National Academy of Sciences of the United States of America*, *111*(6), 2081–2086.

Senn, W., & Pfister, J.-P. (2014). Reinforcement learning in cortical networks. In D. Jaeger, & R. Jung (Eds.), *Encyclopedia of computational neuroscience* (pp. 1–9). New York: Springer.

Seo, J.-s., Brezzo, B., Liu, Y., Parker, B.D., Esser, S.K., & Montoye, R.K. et al. (2011). A 45nm CMOS neuromorphic chip with a scalable architecture for learning in networks of spiking neurons. In *2011 IEEE custom integrated circuits conference—CICC 2011* (pp. 1–4).

Seth, A. K., Sporns, O., & Krichmar, J. L. (2005). Neurorobotic models in neuroscience and neuroinformatics. *Neuroinformatics*, *3*(3), 167–170.

Sivilotti, M. A., Emerling, M. R., & Mead, C. A. (1986). VLSI architectures for implementation of neural networks. In *AIP conference proceedings, Vol. 151* (pp. 408–413).

Sjöström, P. J., Turrigiano, G. G., & Nelson, S. B. (2001). Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron*, *32*(6), 1149–1164.

Srinivasa, N., & Cruz-Albrecht, J. (2012). Neuromorphic adaptive plastic scalable electronics: Analog learning systems. *IEEE Pulse*, *3*(1), 51–56.

Stefanini, F., Neftci, E. O., Sheik, S., & Indiveri, G. (2014). PyNCS: a microkernel for high-level definition and configuration of neuromorphic electronic systems. *Frontiers in Neuroinformatics*, *8*.

Stewart, T. C., & Eliasmith, C. (2014). Large-scale synthesis of functional spiking neural circuits. *Proceedings of the IEEE*, *102*(5), 881–898.

Stromatias, E., Galluppi, F., Patterson, C., & Furber, S. (2013). Power analysis of large-scale, real-time neural networks on SpiNNaker. In *2013 international joint conference on neural networks. IJCNN 2013—Dallas*.

The Human Brain Project, 2015. Project Homepage. URL: https://www.humanbrainproject.eu.

Walter, W. G. (1950). An imitation of life. *Scientific American*, *182*(5), 42–45.

Xin, J., Lujan, M., Plana, L. A., Davies, S., Temple, S., & Furber, S. B. (2010). Modeling spiking neural networks on SpiNNaker. *Computing in Science & Engineering*, *12*(5), 91–97.

Xu, Y., Zeng, X., & Zhong, S. (2013). A new supervised learning algorithm for spiking neurons. *Neural Computation*, *25*(6), 1472–1511.