Department of Physics and Astronomy
University of Heidelberg

Bachelor Thesis in Physics
submitted by

Jonas Weidner

born in Künzelsau (Germany)

2019

Experiment Visualization and Simulations towards a Cortical Microcircuit on the BrainScaleS Neuromorphic Hardware

This Bachelor Thesis has been carried out by Jonas Weidner at the
Kirchhoff Institute for Physics in Heidelberg
under the supervision of
Dr. Johannes Schemmel

Abstract This thesis is part of the long-term goal to run the cortical column network by Potjans and Diesmann (2014b) on the BrainScaleS hardware system of the Electronic Vision(s) group located at the University of Heidelberg. This model was chosen because it is conceptually easy. It is wide spread in the field of neuromorphic hardware and large scale neuromorphic simulations on high performance computers, because it can reproduce biological brain activity.
To keep the results comparable, a simulation was implemented with the Nest simulator and was compared to Potjans and Diesmann (2014b) and Albada et al. (2018). The underlying code provides a solid foundation for the future hardware implementation.
Moreover a hardware mapping analysis was done, showing that for a cortical column network with a size of 10% of the original network on a single wafer, only 5% of the synapses are not realizable.
Furthermore the web visualization of the BrainScaleS-1 system was improved, amongst other things, by implementing synapses and their weights and by decreasing the loading times significantly.

Zusammenfassung Diese Arbeit ist Teil des langfristigen Ziels, das Cortical Column Netzwerk von Potjans and Diesmann (2014b) auf dem BrainScaleS Hardware System der Electronic Vision(s) Gruppe der Universität Heidelberg umzusetzen. Dieses Modell wurde gewählt, da es konzeptionell einfach und im Forschungsgebiet der neuronalen Hardware und der neuronalen Computersimulationen weit verbreitet ist, da es biologisches Verhalten des Gehirnes reproduzieren kann.
Um die Ergebnisse vergleichbar zu halten wurde eine Simulation mit dem Nest Simulator erstellt, welche mit Potjans and Diesmann (2014b) und Albada et al. (2018) verglichen wird. Der zugrudne liegende Code bildet das Fundament für die zukünftige Hardware Implementation.
Zudem wurde eine Hardware Mapping Analyse durchgeführt, welche gezeigt hat, dass ein Cortical Column Netzwerk mit 10% der Orginalgröße auf einem einzelnem Wafer mit 5% Synapsenverlust realisiert werden kann.
Des Weiteren wurde die Web Visualisierung des BrainScaleS-1 Systems verbessert, indem unter anderem Synapsen und deren Gewichte implementiert wurden und die Ladezeit wesentlich verkürzt wurde.

Contents

1 Introduction

1.1 General Motivation

The long-term goal is to run the cortical column network on the BrainScaleS-1 neuromorphic hardware platform. This model is a wide spread benchmark in the field of neuromorphic computing. e.g. Albada et al. (2018), Cain et al. (2016), Merkt, Schüßler, and Rotter (2019). Recently a real time implementation of the cortical column was implemented on the SpiNNaker[1] neuromorphic hardware (Furber et al. 2013), described in Rhodes et al. (2019). Therefore different basic work has to be done. This thesis deals with the simulation of the network and with the visualization of the mapping on the BrainScaleS-1 hardware.

The simulation is important to compare the results to the hardware results. The goal is to start with a fully functional simulation and then change all parameters to hardware realistic ones (size of the network, synapse parameters, neuron parameters) while keeping the overall results of the network constant.

On the other side we want to start with the hardware mapping from small networks and scale up while improving the mapping algorithm.

The visualization is necessary in order to help experiment designers optimize their network on hardware.

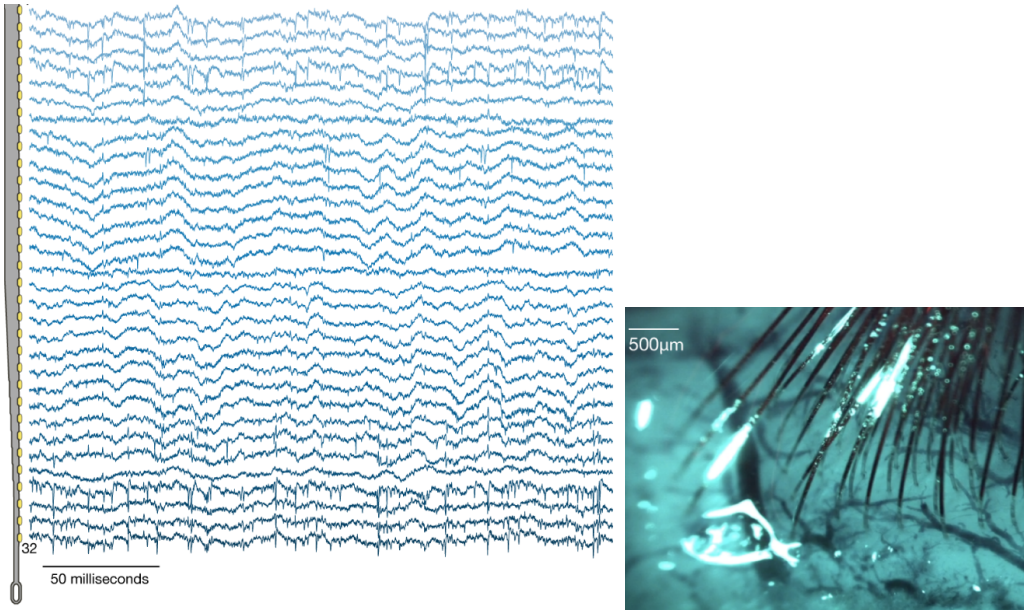
1.2 Neuronal Networks

Artificial neuronal networks are based on a brain inspired structure. In the following only the abstract behavior is discussed.

The smallest unit is a neuron. In the model we used it can be seen as a node with multiple inputs, one output and a scalar value, the membrane potential. If the membrane potential reaches a threshold, the neuron spikes. A spike is parameterized just by the point in time when it occurs. In addition to the explained model of Rosenblatt (1985), neurons have way more parameters that specify the dynamical membrane potential behavior. An example of membrane potentials of neurons located in the cortex of a rat is shown in Figure 1a.

Synapses can be considered as the connections between neurons. Those can either be excitatory or inhibitor, which means that they increase or decrease the membrane potential of a target neuron if the source neuron sends a spike. Additionally they have a scalar value indicating the weight of the connection and also other dynamical parameters.

[1] Spiking Neural Network Architecture



(a) (b)

Figure 1: (a) 32 neural signals of the cerebral cortex of a rat measured with an implemented thread (left) by Musk and Neuralink (2019). It shows membrane potentials with spikes. (b) Threads implemented in the cortical surface of a rat brain. Both images were taken from Musk and Neuralink (2019).

A detailed explanation of different neuron and synapse models can be found in (Petrovici 2015).

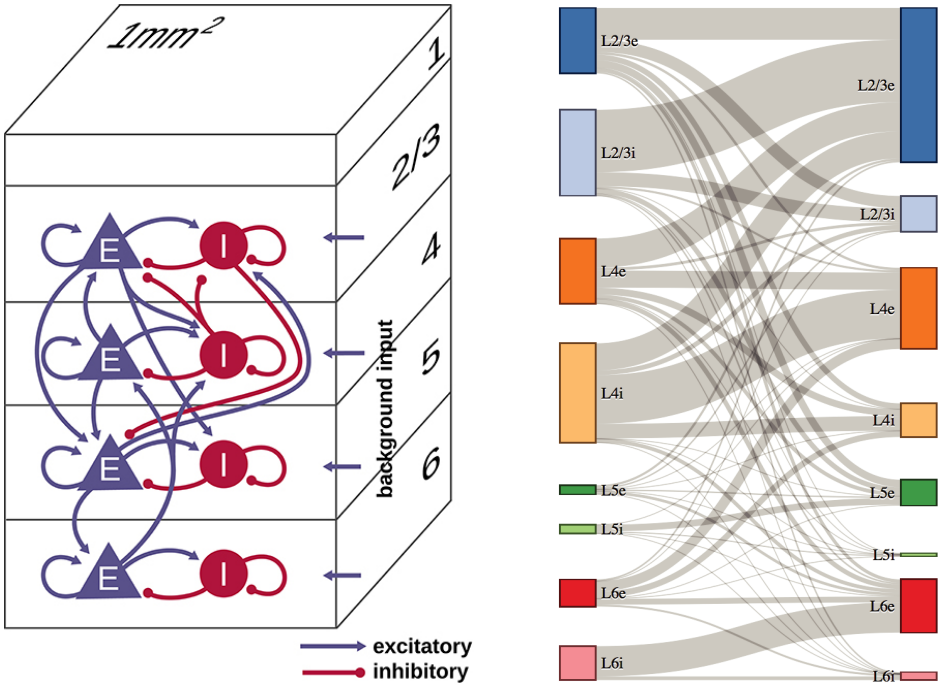
1.3 Cortical Column Network

The cortical column is based on biological experiments. For example by Thomson et al. (2002) measuring connection probabilities by using electrophysiological recordings of rat and cat brains. This was done by implementing sharp electrodes into the brain similar to the picture shown in Figure 1b.

The cortical column model was invented by Potjans and Diesmann (2014b) based on biological experiments. It describes column of 1 mm^2 of the cortical column of a human brain. The network is built out of eight populations, shown in Figure 2a. A population consists of 1,065 to 21,915 neurons. All populations are connected to all other populations with a certain probability (allowing zero as probability), given in Potjans and Diesmann (2014b, Tab. 5). In detail, it is the chance that two specific neurons of different populations are connected.

The great achievement of Potjans and Diesmann (2014b) was to reproduce the biological behaviour by simulation. Because of the known biological results, the cortical column network can be seen as a benchmark for simulation and neuromorphic computing. Benchmarks are needed in this field to reach further improvements as recently mentioned by Davies (2019).

The eight populations are assigned to four layers, which are called layer 2/3, 4, 5 and 6, as shown in Figure 2a. Each layer includes one excitatory and one inhibitory population. Excitatory means that spikes sent from this population lead to a rise of the membrane potential of the spike receiving target neuron. Spikes sent from an inhibitory population decrease the membrane potential of the target neuron.



(a) Overview (b) Connections

Figure 2: (a) Illustration of a 1 mm^2 cortical column network with about 80,000 neurons and 300 Mio. synapses. The structure of the network was derived from multiple biological experiments. It consists of four layers 2/3, 4, 5 and 6 with one excitatory and one inhibitory population each. Excitatory external input and main internal probabilistic connections are drawn. Figure taken from Albada et al. (2018, Fig. 1). (b) Visualization of connections from source(left) to target(right) populations. Exact connection probabilities can be found in Potjans and Diesmann (2014b, Tab. 5). Figure taken from Cain et al. (2016, Fig. 1a).

1.4 BrainScaleS-1 System

The BrainScaleS-1 System is a neuromorphic hardware system developed by the Electronic Vision(s) group at the Kirchhoff-Institute for Physics at Heidelberg University (*BrainScaleS System - Neuromorphic Computer Coming Online* 2016). The system is shown in Figure 3b. It contains 20 silicon wafers shown in Figure 3a. Each wafer contains 384 HICANN[2] chips, which are interconnected via an on-wafer network. A HICANN chip consists of 512 neurons (256 in the upper and lower part each) and 220 synapses per neuron. This adds up to four million neurons and one billion synaptic connections for the whole system.

The spike input and output is handled by vertical connections linking all HICANN chips to a set of 48 FPGA[3]s.

(a) Wafer with Controlling (b) 20 Built-in Wafers

Figure 3: BrainScaleS-1 - Neuromorphic Hardware System. Images were taken from (*Twitter Account of Electronic Vision(s) group* 2019).

The neurons and synapses are realized with analog electronic circuits based on the AdEx - LIF[4] neuron model. This model is a simplification of the Nobel Prize winning HodgkinHuxley neuron model by Hodgkin and Huxley (1952), which is based on four differential equations. A detailed explanation is given by Naud et al. (2008). Inter wafer spike communication is digital. Compared to biological timescales a similar emulated neural network on the BrainScaleS-1 System is approximately 10,000 times faster.

[2]High Input Count Analog Neural Network Chip
[3]Field Programmable Gate Array
[4]Adaptive Exponential Leaky Integrate-and-Fire Neuron Model

1.5 Nest Simulation

With the NEST[5] simulator it is possible to simulate a large variety of different spiking neural networks. One can chose models out of a wide range of different neuron and synapse models. For more details refer to Plesser et al. (2007). We use it to compare the results to the neuromorphic hardware by using the Python interface PyNN (Davison et al. 2009). We used Nest version 2.2.2.

1.6 PyNN

PyNN is a Python interface for common neuronal network simulators (Davison et al. 2009) ("PyNN Documentation" 2019). It offers a generalized python interface for the BrainScaleS hardware, for the Nest simulation software and for other simulators.

In the following example, the Python code connects a source population with three neurons to a target population with four neurons. All neurons of the source population are connected to all neurons of the target population. The synaptic weight is 0.4. The simulator can be chosen in the first line. Import "nest" to make a Nest simulation or "pyhmf" to run on the BrainScaleS hardware. A visualization is shown in Figure 4.

```
import nest_or_phymf as pynn
sourcePopulation = pynn.Population(3)
targetPopulation = pynn.Population(4)
conncetor = pynn.AllToAllConnector()
synapse = pynn.StaticSynapse(weight = 0.4)
pynn.Projection(sourcePopulation, targetPopulation,
                connector, synapse)
```

[5]Neural Simulation Tool

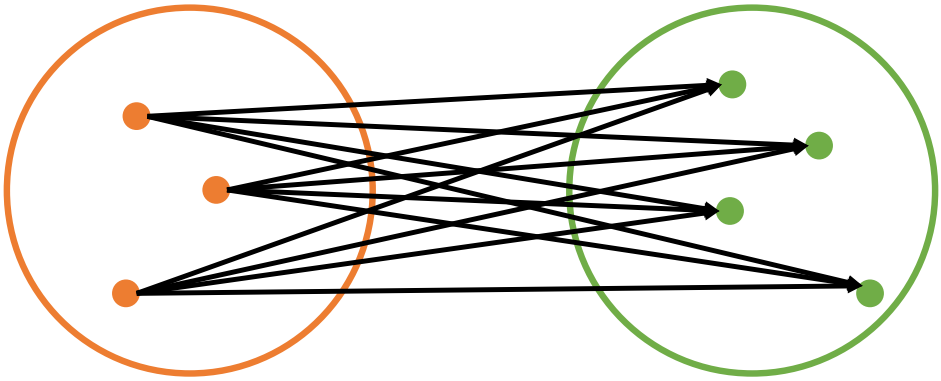


Figure 4: Visualization of an All-to-All network described in section 1.6. Each of the three neurons of the source population (left, orange) is connected to all four neurons of the target population (right, green).

1.7 Placement and Routing on Hardware

The network, specified in PyNN, has to be mapped on hardware. This is done by the program marocco[6]. Therefore all specified neurons and synapses have to be placed on hardware and need to be connected. There are a lot of different hardware properties and limitations, making it a complex and non-trivial problem to solve. Hence, I want to focus on finding the best mapping variables and algorithm for the cortical column network.

1.7.1 Different Mapping Algorithms

To get more inputs on a model neuron, it is possible to merge multiple hardware neurons to one model neuron. The size of the hardware neuron can be varied from 2 to 64.

The mapping is split into multiple parts. We focus on the placement, which places all neurons and the routing, which connects them. The easiest placing algorithm is the so called linear placer. It starts at a specified starting point and places one neuron after another. Afterwards the routing algorithm connects them.

The "byNeuron" placer also places one neuron after another, but it takes into account that the neuron has to be connected to all of his already placed source and target neurons. Therefore it checks if there are globally enough routes available between the HICANN chips.

When placing a neuron, the "constrained" placer checks if there are locally enough synapses available to get all input. If this is not the case, the neuron is replaced at

[6]Mapping and Routing Software for the BrainScaleS-1 System

another location. The "constrained" placer builds on the "byNeuron" placer.

A very detailed explanation is given by Jeltsch (2014) in his PhD thesis. Major improvements were made by Passenberg (2019) and by Klähn (2017).

1.8 Web Visualization

The BrainScaleS-1 System enables users to emulate neural networks on a hardware system. The marocco mapping and routing software translates the network so that the hardware can emulate it. Results of this mapping process can be visualized with the web visualization software. Therefore a hardware run is therefore not required. It is enough to just run the mapping. Figure 5 shows the user interface of the web visualization software.

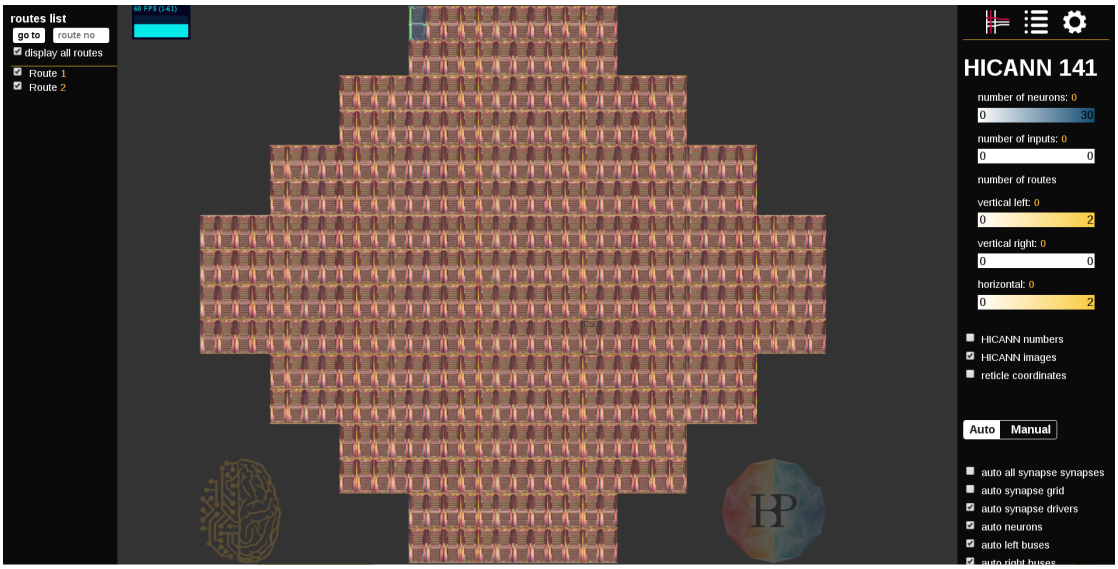
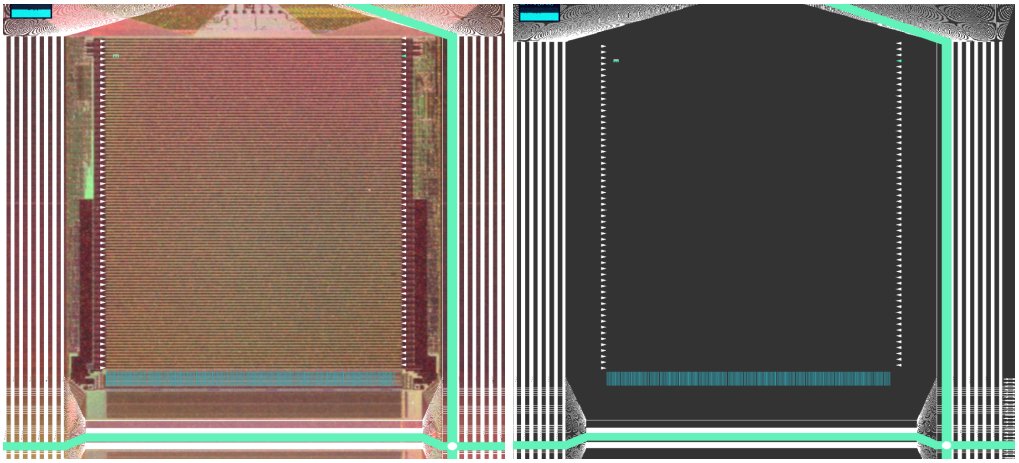


Figure 5: Web Visualization Software. On the left side is a list of the realized connections between neurons. So called routes. All settings are located on the right. In the middle one can see a zoomable wafer representation.

The purpose of the visualization is to show experimenters which regions of the wafer are in use and how they interconnect. It is basically a 2D surface where one can zoom in and out and switch the visibility of different elements on or off. Zooming in, one gets a more detailed view on the neurons, synapses, connections and other elements on the chip. This is shown in Figure 6.

The web visualization software was created by Richard Boell as part of his bachelor thesis (Boell 2018) and is maintained by the Electronic Vision(s) group.



(a) Upper HICANN half

(b) Upper HICANN half, background image is turned of.

Figure 6: Web visualization zoomed into a HICANN-Chip with and without background image is shown. The example from Section 1.6 is visualized, an All-To-All network from 3 to 4 neurons. The blue stripes symbolize the neurons. Only one connection is needed, because all source neurons address all target neurons. The green line represents this connection. The signal starts at a neuron (one of the blue stripes) and travels down to the green line. From there it travels via the MergerTree to the right and up along the green line. The green triangle is the synapse driver that connects the signal to the synapse grid (the rectangle above the neurons). The used synapses are the few green dots (top left). From there the signal goes back down to the target neurons, which are also located on the same chip (blue stripes).

1.8.1 Comparison to other Visualization Tools

There are already different visualizations existing in the field of neuromorphic computing. The VIOLA software is a web based Nest visualization tool developed by Senk et al. (2018). It provides different views on the result of layer simulated spiking neural networks. The visualization tool developed by Nowke et al. (2015) focuses on large brain like models. It links the Nest model directly to brain regions.

In contrast to the presented Nest visualization tools, we primarily wanted to have a visualization, which is closer to hardware to reveal chip specific concerns.

This year, Rowley et al. (2019) presented their SpiNNTools software package. It enables users to work with graphs, as shown in Figure 2a, on the SpiNNaker hardware platform. A similar tool, to create graphs as shown by Passenberg (2019, Fig. 3.4), is

currently under development and will be included into the marrocco program.

As the HICANN chip has a unique design, so it also needs a special visualization tool.

1.8.2 Pixi.JS

PixiJS is an open source 2D WebGl graphics library (*PixiJS 5* 2019). It is used for the 2D wafer mapping web visualization. There are different ways to draw graphics on the screen. The first way is to draw them as graphic objects. The other way is to generate a texture out of a graphic object and draw them as a so called sprite object. The graphic object is the right way for most objects in the web visualization. A comparison was done by Boell (2018).

1.8.3 Emscripten

Emscripten is a tool to wrap C++ code to JavaScript. The main reason to use it, is to reuse C++ code in the Web Visualization. Classes and functions are getting transferred automatically, but they still have to be specified so that they can be used. The following code shows how the class named `marocco::results::Marocco` and the function named `marocco::results::Marocco::load` can be used in JavaScript as `MaroccoInJavaScript` and `loadInJavaScript`.

```
EMSCRIPTEN_BINDINGS ( marocco_results ){
emscripten :: class_ < marocco :: results :: Marocco >("MaroccoInJavaScript")
. function ("loadInJavaScript", & marocco :: results :: Marocco :: load )
}
```

A more detailed explanation can be found in Boell (2018, Sec. 3.2).

1.8.4 Building Steps

The workflow is shown in Figure 7. The main part of the web visualization is written in TypeScript (*TypeScript* 2019) and is transcompiled into JavaScript.

The `marocco::results` class contains all mapping results. This class can read and write an XML[7] results file, which is loaded into the web visualization. The `marocco::results` class is reused in the web visualization. This is done by Emscripten (*Emscripten* 2019), which rewrites the class in JavaScript. Compared to rewriting the code by hand in TypeScript, this makes it way easier to maintain the code when changing something in marocco. In the Electronic Vision(s) group it turned out that maintenance-intensive programs will brake soon after the maintainer is gone.

[7] Extensible Markup Language

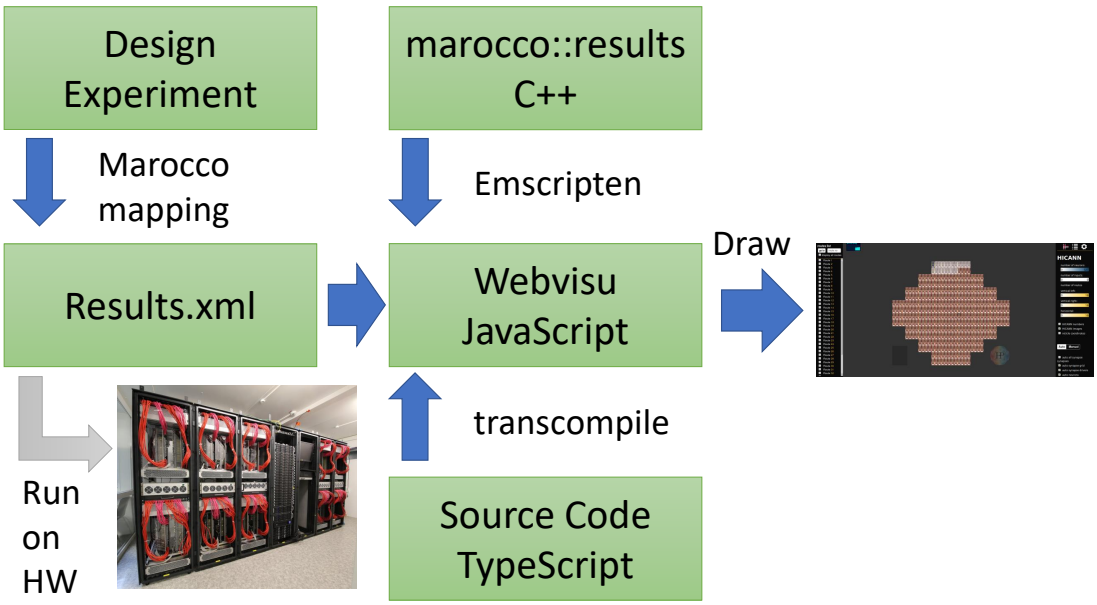


Figure 7: Building steps and usage of the web visualization software. At first, the experiment has to be designed. The marocco mapping tool creates an XML results file that can be run on the BrainScaleS-1 hardware. The web visualization can read the same file. The source code is written in TypeScript and is translated into JavaScript. To read the results file, the `marocco::results` class gets embedded into JavaScript via Emscripten (*Emscripten* 2019).

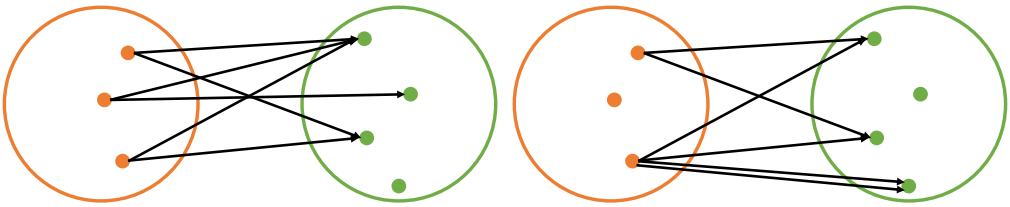
2 PyNN Reimplementation of the Cortical Column Network

We already had some code of the cortical column network from Albada et al. (2018). Because of four different reasons we decided to reimplement the network in PyNN. The first reason was that the already existing code was hard to understand and so it would not be much faster to use it. Additionally, the idea was to get familiar with the network and develop some intuition by writing the code on our own. Another point is that we can focus on Nest and especially the BrainScaleS system and also include the mapping process. Furthermore we can start from the original model Potjans and Diesmann (2014b) and get independent results.

We decided to start with a light version that only contains the connections and the mapping as benchmark for the marocco routing tool and for further improvements. The other version, containing all specific neuron and synapse parameters, is used for the Nest simulation and for hardware experiments simultaneously.

2.1 Benchmark

For the benchmark we implemented the eight populations, the internal connections and the external input of the column network accordingly to Potjans and Diesmann (2014b, Tab. 5), in PyNN (Davison et al. 2009). To realize the internal connections we chose the `pyNN.FixedProbabilityConnector`. In Figure 8 it is compared to the `pyNN.FixedTotalNumberConnector`, which was not working and had to be implemented manually.



(a) PyNN Fixed Probability Connector (b) PyNN Fixed Total Number Connector

Figure 8: Visualization of two PyNN Connectors. The source population (left, orange, 3 neurons) connects to the target population (right, green, 4 neurons). (a) Each source neuron connects to every target neuron with a certain probability (50% in this case). (b) The total amount of connections is fix (6 in this case). Compared to (a) it can happen that two neurons are connected multiple times.

As external input we chose the row named "(reference)" from Potjans and Diesmann

(2014b, Tab. 5). A `PyNN.FixedNumberPostConnector`, which connects all target neurons to a fixed number of source neurons from a virtual external input population, realizes the external input.

2.2 Hardware and Nest

Compared to the benchmark implementation, we focused on the Nest version at first. We added the synaptic weights and delays, as well as the neurons as current based neurons with all parameters accordingly to (Potjans and Diesmann 2014b, Tab. 5). This neuron was chosen, knowing that it has to be replaced with a conductance based one, which is implemented on the BrainScaleS hardware. The reason for that is that they are quite similar and we wanted to reproduce the results of (Potjans and Diesmann 2014b) before.

It turned out that the fixed probability connector assumes that each neuron in the source population is connected to each neuron of the target population with the given probability. In contrast, (Potjans and Diesmann 2014b) meant the probability that two neurons of the different populations are connected, which differs in total connections by about 10%. For the Nest simulation this was corrected by calculating the total amount of connections (Potjans and Diesmann 2014b, Eq. 1,2). More details can be found in (Potjans and Diesmann 2014a, Sec. 1). With that both connectors were implemented. Comparisons of the realized connections showed that the difference between the different connectors is very small. This might be helpful later. We can choose the one that fits better on hardware.

The external input mechanism was changed to get an individual input for each neuron by a `PyNN.OneToOneConnector` from the virtual external input population to each population. The reason therefore is that all neurons get input. The virtual external input population emits Poisson distributed spikes.

These changes are no problem for the benchmark, because it is only a small difference and we focus on the relative changes in different mapping algorithms. Furthermore, for the final column network on the hardware, there will change a lot after implementing the right scaling behaviour.

3 Mapping Results

3.1 Different Mapping

For all mapping analysis we used the benchmark version of our cortical column PyNN implementation. Thereby we assumed to have a perfect wafer, meaning all components

are working. Via calibration not working components can be blacklisted and can be considered later in the mapping process. For a better comparison we chose to stick to a perfect wafer for the mapping tests.

In Figure 9 different mapping algorithms are compared. The amount of lost synapses is compared to the network size. As there is no working scaling algorithm yet, we mapped a smaller network by linearly scaling down the amount of neurons per population and keeping the connection probability the same.

As already mentioned, hardware neurons can be connected to get more input. The maximum size is 64. With previous testing we found out that 4, 8 and 16 are the best sizes. This makes sense because all of the 64 hardware neurons can be used.

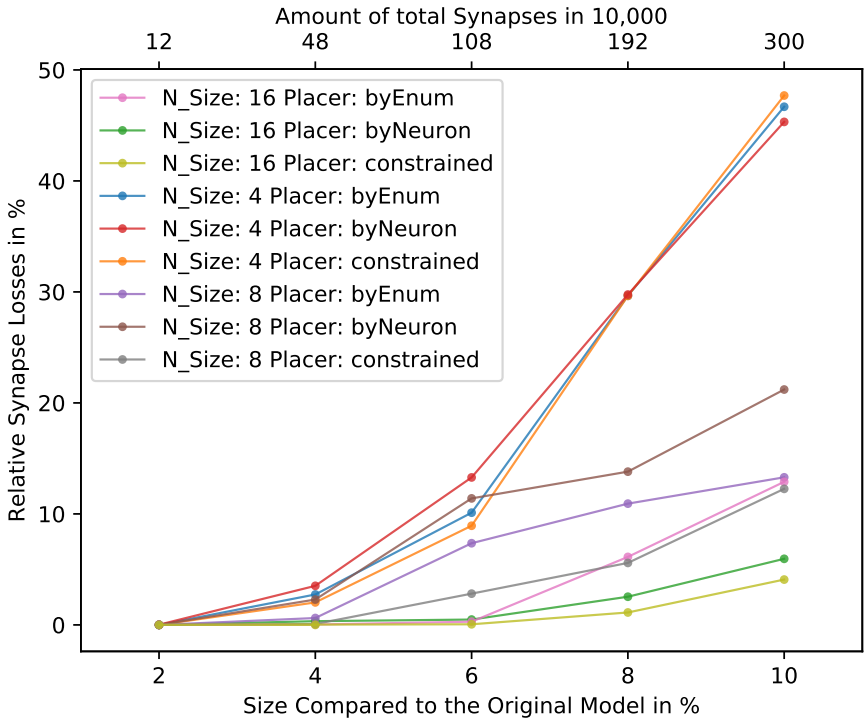


Figure 9: All not realized connections (synapse losses) of different mapping algorithms are compared to the network size in total synapses and relative to the original model of Potjans and Diesmann (2014b). Different hardware neuron sizes and placement algorithms are used.

The best result that we can get is a 5% loss with network size of 10%, as shown in Figure 9. This results in about 130,000 used neurons of about 200,000 available ones. Of the available 41 Mio. synapses 3 Mio. are occupied.

The most important factor is the hardware neuron size. It turned out that a mapping with 16 hardware neurons per model neuron is the optimum case. Additionally the network would not fit on one wafer with a network size of 10% and a hardware neuron size of 32, as it would take twice as many hardware neurons in total. A hardware neuron size of 20, which is the largest neuron size fitting three neurons on one neuron block, was tested as well.

The "constrained" placer has the fewest losses. This was expected, as it includes the "byNeuron" placer. But with a neuron size of 16 the difference between those two placers is small because of the already large amount of possible input synapses per model neuron.

It still might still be better on a realistic wafer to use smaller hardware neurons. This would lead to a smaller network on the chip and would leave more spare components for eventually blacklisted components.

In Figure 10 the distribution of synapse losses over all population connections is shown. It turns out that the loss is equally spread. As mainly the whole populations are regarded in the network analysis, it is also important that those receive equal synapse losses, which is also given. This is important to receive the same results as Potjans and Diesmann (2014b). It is assumed that small equally distributed losses can be compensated by larger synaptic weights.

This mapping analysis should not be considered as a precise statement of how large the cortical column network can be realized on a single wafer, but rather as a rough estimation. As we have not considered blacklisting we will get more losses, but the real scaling algorithm will also affect the mapping.

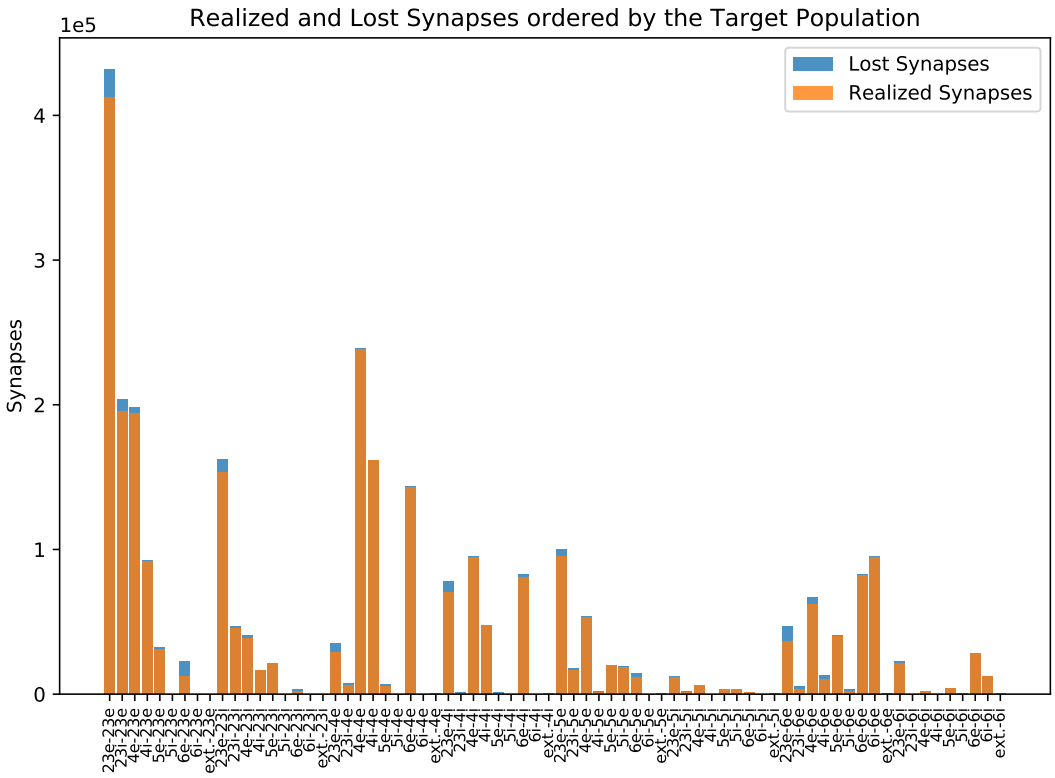


Figure 10: Loss distribution of all internal and external connections of the best mapping result of Figure 9. The network size is 10%, the hardware neuron size is 16 and the "constrained" placer is used.

3.2 Visualization of the Mapping

In Figure 11, a cortical column network with a size of 10% is visualized in the web visualization. A hardware neuron size of 16 and the "constrained" placer were chosen. This setting is currently the cortical column network with the fewest losses as shown in Figure 9.

It is shown that about 70% of the wafer is used by the cortical column network. If this could be implemented on hardware, it would be a great achievement. As shown in Figure 11, there is still some space on the left. It might be possible to use this space as well, but loss will rise in this case because of increasing difficulty to connect the neurons.

This is also a good example of how the web visualization can help during the mapping process.

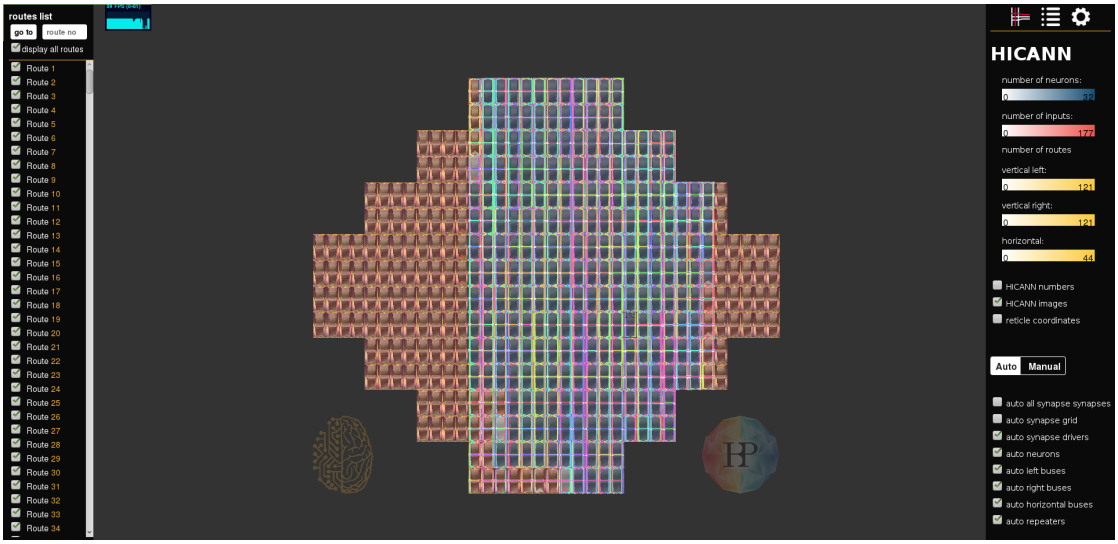


Figure 11: Web visualization of 10% of a cortical column network. The best mapping (Figure 9), containing the "constrained" placer and a hardware neuron size of 16, with 5% loss is shown. The selected routes (right) indicate the used HICANN chips.

4 Nest Simulation of the Cortical Column network

In the following the different behaviors of the cortical column networks are compared to the results of Potjans and Diesmann (2014b) and Albada et al. (2018).

4.1 Mean Rates of all Populations

The spike rates of all populations are shown in Figure 12. They were compared to different random seeds for the normal delay and weight distributions of the synapses, as well as for random external inputs. Moreover the rates from (Albada et al. 2018) and from (Potjans and Diesmann 2014b, Fig. 6b) are plotted.

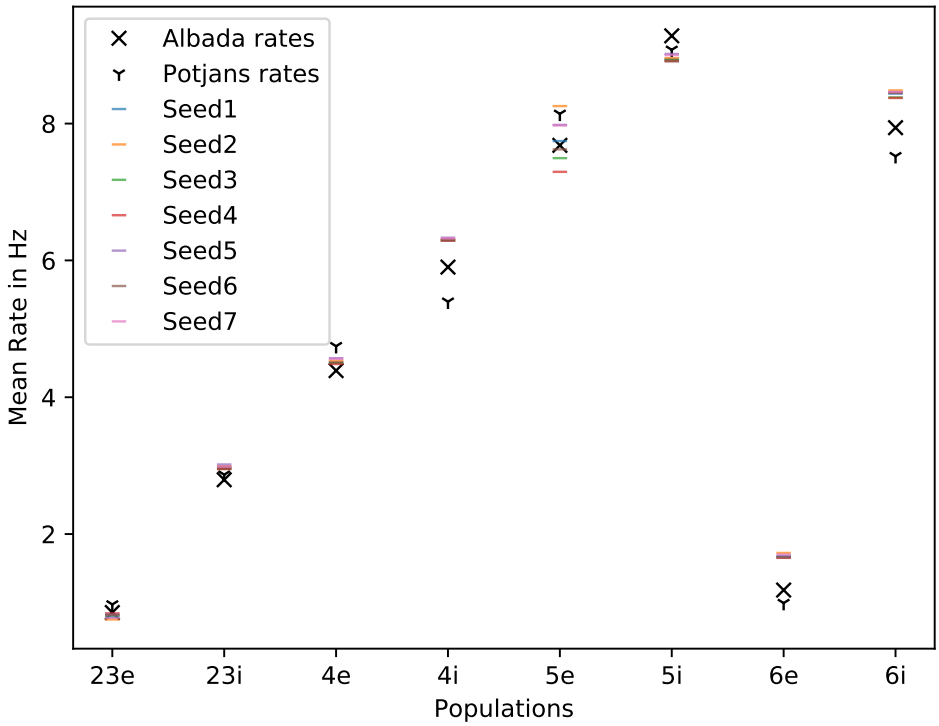


Figure 12: Mean rates of all populations with seven different random seeds compared to Albada et al. (2018) and Potjans and Diesmann (2014b, Fig. 6b). The simulation time was 60s. The first second was not considered and all neurons of the populations were measured.

First of all, one can see that the rates fit well to the results of Albada et al. (2018) and Potjans and Diesmann (2014b). Additionally, it can be shown that the different random seeds do not effect the mean rates much, accept for population 5e. This fits to Albada et al. (2018, Fig. 6), which shows a similar behavior for different seeds as well.

Small differences appear in population 4i, 6e and 6i. This can be explained by the different simulators and numerical difficulties. Even Potjans and Diesmann (2014b) and Albada et al. (2018) differ a bit.

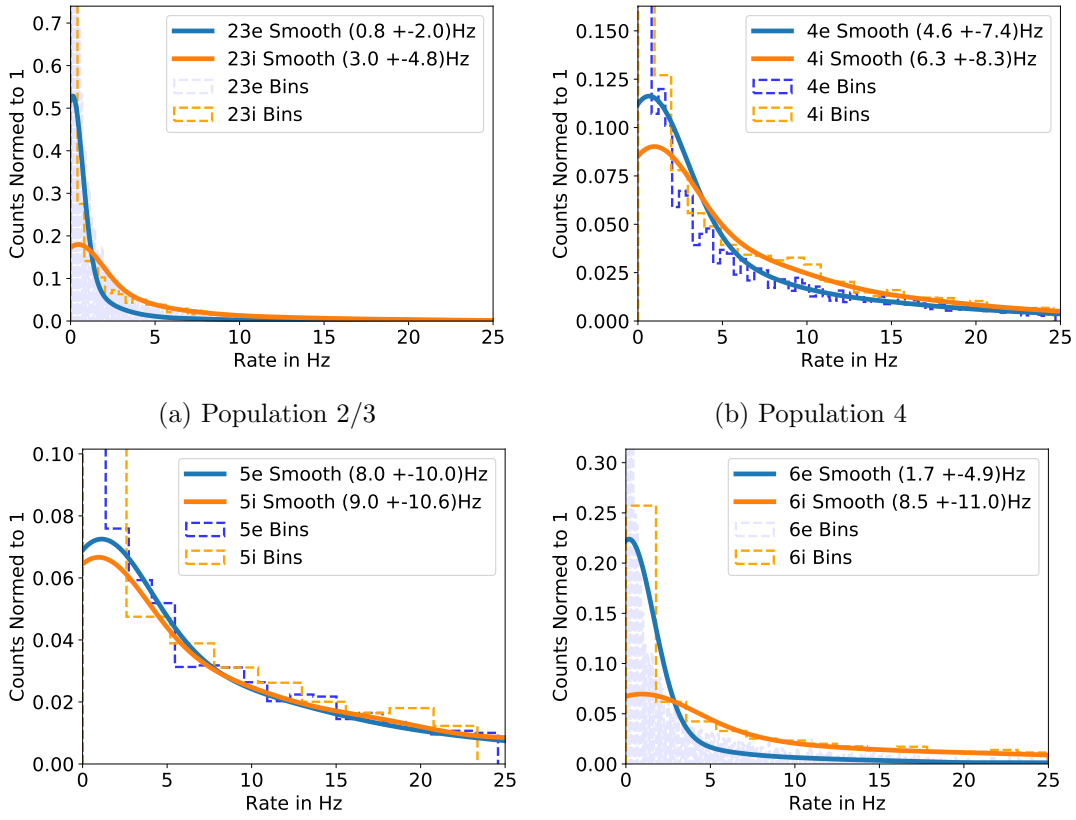
4.2 Rate Distributions

More detailed results of the population internal rate distributions are shown in Figure 13. Those results can be compared with Albada et al. (2018, Fig. 5d). The bin width

B_w was chosen with the rule of Freedman and Diaconis (1981).

$$B_w = 2\frac{\mathrm{IQR}(X)}{\sqrt[3]{n}} \tag{1}$$

X describes the sample, IQR the interquartile range and n the number of measurements. For comparison we smoothed the histogram bins in the same way as Albada et al. (2018, Sec. 2.4) did. Therefore we used the `scipy.stats.gaussian_kde` function with bandwidth $0.3Hz$ to perform a Gaussian kernel density estimation.



(a) Population 2/3

(b) Population 4

(c) Population 5

(d) Population 6

Figure 13: Spike rate distributions for all eight populations over all neurons of one random seed sample. Blue describes the excitatory and orange the inhibitory population. The mean rate and the standard deviation is shown in the legend. The bin width was chosen with the Freedman and Diaconis (1981) rule and the smoothing is explained in Section 4.2. The same parameters as in (Albada et al. 2018, Fig. 5d) were chosen. The last 9s of a 60s simulation were considered. In (a) and (d) the excitatory bins are very dense, which explains the blue area.

Compared to (Albada et al. 2018, Fig. 5d), we get very similar results, as shown in Figure 13. The main difference appears for population 4 and 5. Albada et al. (2018) measure a narrower, but therefore a more right shifted, distribution.

The smoothed lines vary quite a bit from the bins. Especially the first bin, with a rate of zero, seems way smaller than it is. On the other side, the smoothing provides a better overview for very small bins as shown in Figure 13a and 13d. It is questionable why Albada et al. (2018) chose a Gaussian kernel density estimation, as our bins do not appear normal distributed, but rather exponentially decreasing.

4.3 Irregularity

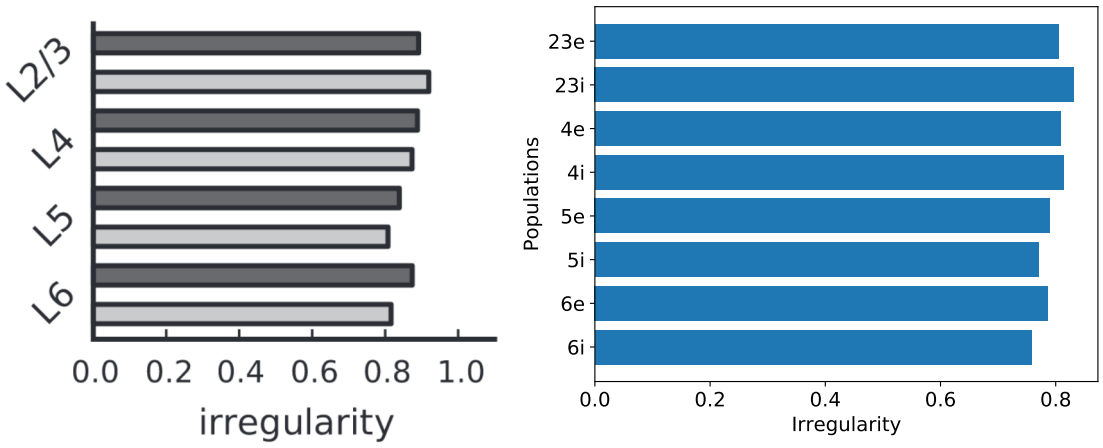
The irregularity is a measurement of how irregular spikes of one neuron appear. Therefore, the inter spike time intervals ISI of one neuron, the times between one spike and the following, are considered. As measurement of the irregularity, the coefficient of variation C_V is defined equivalent to Potjans and Diesmann (2014b, Fig. 6c).

$$C_v = \frac{\sigma(P_{ISI})}{\mu(P_{ISI})} \tag{2}$$

P_{ISI} is the distribution of all ISI's of one neuron. μ and σ characterize the mean and standard deviation.

Intuitively, if the neuron spikes with a nearly constant rate, this leads to a small standard deviation of the ISI's and therefore to a small irregularity. The $C_V(P_{ISI})$ of a completely random uniform spike distribution, a Poisson distribution is equal to 1. For more detail see (Gerstner et al. 2014, Sec. 7.3).

The mean of 1000 neurons C_V's per population for a simulation time of 60 seconds is shown in Figure 14.



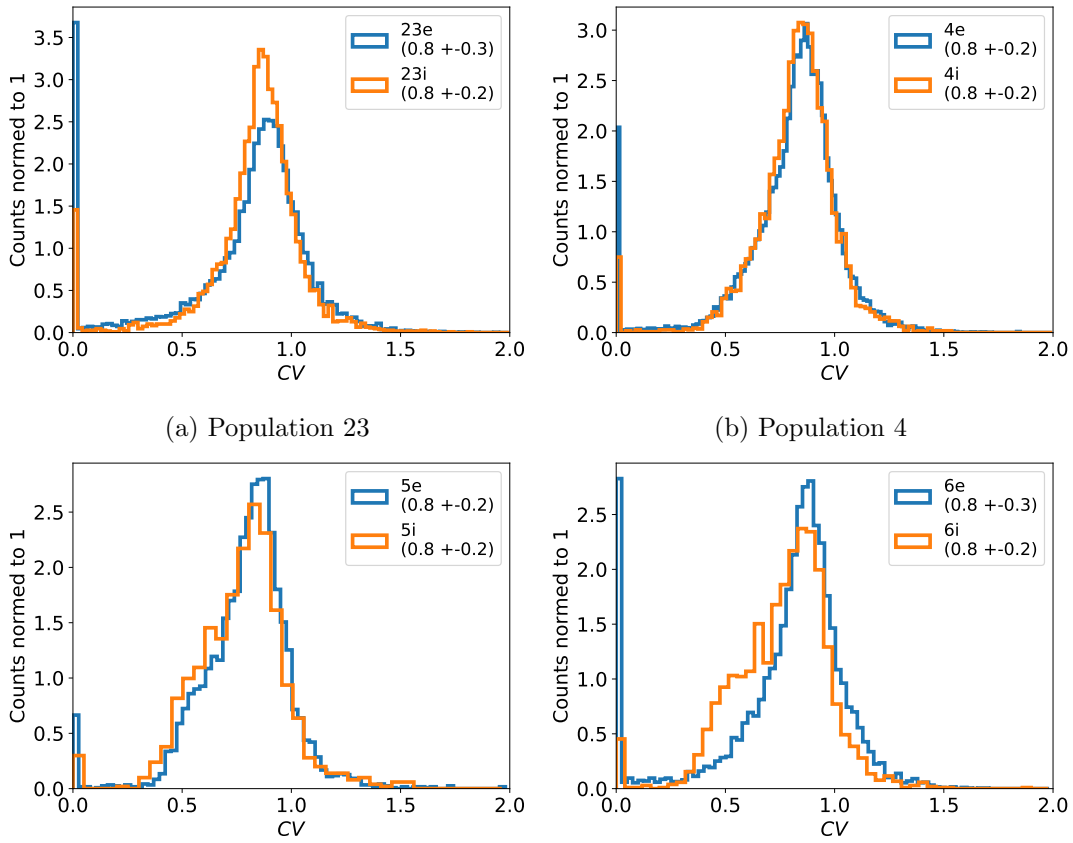
(a) (b)

Figure 14: Irregularity of all populations of the cortical column network. (a) Results of Potjans and Diesmann (2014b, Fig. 6d). Dark stands for excitatory and bright for inhibitory populations. (b) Our Results. Measurements were taken from 1000 neurons per population. The simulation time was 60s, where the first second was not considered.

Compared to Potjans and Diesmann (2014b, Fig. 6d) we get very similar results. The irregularity is close to one, which is close to a Poisson process. For the last populations, the irregularity is a bit smaller, but this can be explained with statistical and numerical variations.

4.4 Irregularity Distribution

The C_V distributions are inspired by Albada et al. (2018, Fig. 5e). Our comparable results, with the same parameters, are shown in 15. Compared to the rate distributions (4.2), the smoothing Albada et al. (2018) did, was not applied, as the bins, calculated with Equation 1, already have a smooth size. Additionally it is easier to see the first bin, which contains all neurons spiking exactly twice. As two spikes only have one inter spike time interval (ISI), the coefficient of variation (C_V) is zero (Equation 2). Because of the inclusion of the first bin by Albada et al. (2018), we also plotted it for better comparison.



(a) Population 23

(b) Population 4

(c) Population 5

(d) Population 6

Figure 15: Irregularity (Coefficient of Variation) distributions of inter spike times of all eight populations. Blue describes the excitatory and orange the inhibitory population. The mean and the standard deviation are shown in the legend. The same parameters as chosen by Albada et al. (2018, Fig. 5e) were used. The simulation time was 60s, but the first second was not considered. All neurons that have more than one spike were measured. The bin width was chosen accordingly to the Freedman-Diaconis rule explained in Equation 1. Compared to Albada et al. (2018), smoothing was not applied, because the distributions are already recognizable.

Compared to Albada et al. (2018, Fig. 5e), Figure 15 shows that the first bin is usually way larger in our results. This is caused by the smoothing process (explained in Section 4.2), Albada et al. (2018) used.

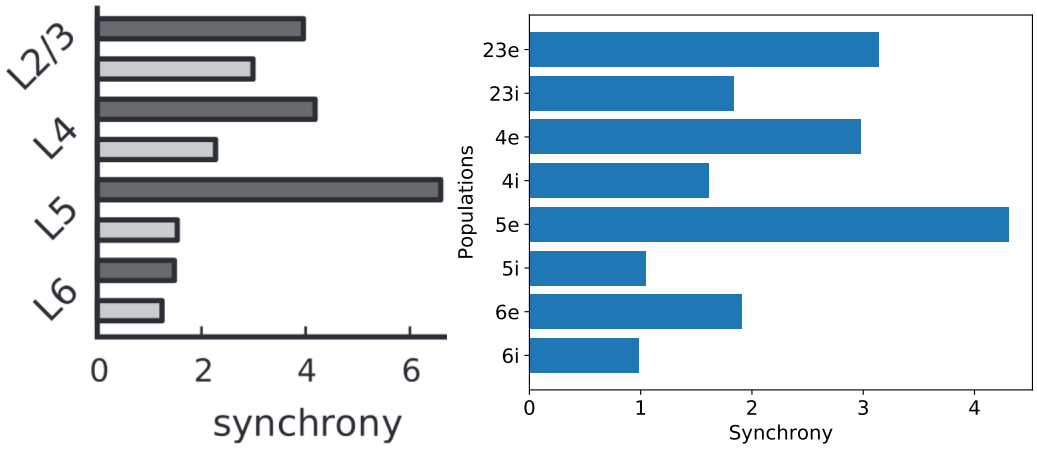
The relative differences between the excitatory and inhibitory distributions is similar to Albada et al. (2018), as well as the absolute distributions of population 4 and 5. The peaks of the irregularity distributions of population 2/3e, 2/3i and 6e are higher compared to Albada et al. (2018). Those deviations can be explained by the statistical and numerical variations.

4.5 Synchrony

Synchrony is a measurement for simultaneity of spikes of multiple neurons in one population. Therefore, all spikes of all considered neurons in a certain time step t are counted. The distribution of counts in all time steps is called $P_C(t)$. The synchrony S is defined equivalent to Potjans and Diesmann (2014b, Fig. 6d) as the variation of P_C.

$$S = \boldsymbol{var}(P_C) \tag{3}$$

The synchrony of all populations of the column network is shown in 16



(a) (b)

Figure 16: Synchrony of all populations of the cortical column network. (a) Results of Potjans and Diesmann (2014b, Fig. 6d). Dark stands for excitatory and bright for inhibitory populations. (b) Our results. Measurements were taken from 1000 neurons per population. The simulation time was 60s and the time step t was 3 ms. The measurement was started after one second.

The results of the synchrony plot differ from Potjans and Diesmann (2014b, Fig. 6d). The main difference appears for population 5e. This population is the most fragile one, which can also be seen in Figure 12, in Potjans and Diesmann (2014b, Fig. 8c) and in Albada et al. (2018, Fig. 6). So it can partly be explained by statistical deviation.

Popualtion 4i, 5i, 6e and 6i match the results of Potjans and Diesmann (2014b).

A possible explanation might be the very important and small bin size of 3 ms compared to the the simulation time step of 0.1 ms. Moreover, the measurement is specifically depending on the simulation time, the amount of measured neurons and the mentioned bin, making it hardly comparable.

5 Web Visualization

Before optimizing the web visualization, it was in a fluid and fast state for small networks with up to about 100 connections. For larger networks it was very slow in the loading phase and also choppy during zooming.

Therefore, different performance improvements were implemented, as well as additional functional features.

5.1 Functionality

5.1.1 Synapse

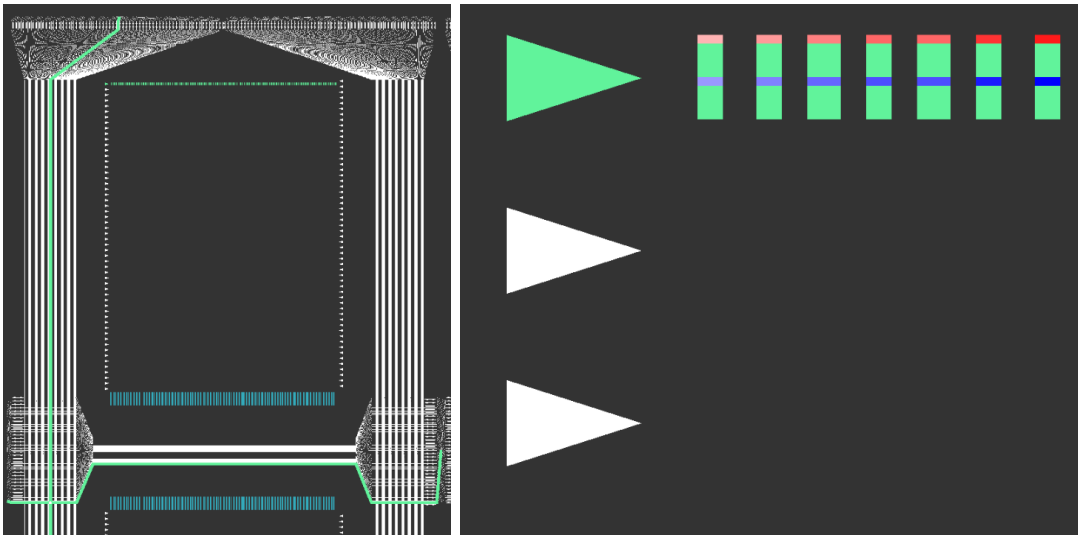
Synapses can be considered as the connection between neurons. The synapses from the new web visualization are shown in Figure 17. They are located in the large rectangle above the neurons, called synapse grid, as a simple rectangle. On the hardware they are located at the same position. Additionally there is another synapse grid on the lower half of the HICANN chip. The synapse color is the same as the color of the route for better distinction.

5.1.2 Synapse Type Indication

Synapses are either excitatory or inhibitory. This describes the behavior of a spike at the post synaptic neuron. To indicate this in the new web visualization a red/blue rectangle is drawn on top of the synapse, signalizing an excitatory/inhibitory synapse. A screenshot of the visualized synapse type is shown in Figure 17b.

5.1.3 Synaptic Weight

The synaptic weight defines how strong the impact of a spike from a certain synapse on the membrane potential of the post synaptic neuron is. This means that a high weight of an excitatory synapse leads to a higher membrane potential of the target neuron, which leads to a higher probability of a spike from the target neuron. In the web visualization the weight is indicated as the intensity of the blue or red color in the box on top of the synapse that indicates the synapse type (Figure 17b).



(a) The upper half of the HICANN chip is shown in the web visualization. The green line indicates the connection.

(b) 14 new synapses (green rectangle) and three synapse drivers (triangle) are shown. The rectangle on top of the synapse indicates the type (excitatory or inhibitory) by color (red or blue) and the weight of the synapse by intensity. The weight increases from left to right.

Figure 17: New web visualization of a network that connects one neuron to 126 other neurons (blue stripes) with an excitatory and an inhibitory connection. The source neuron is located in the left most blue stripe in (a). From there the spike travels along the green line to the left and up to the synapse driver (green triangle) in (b). The synapse driver sends the spike to the 126 inhibitory and 126 excitatory synapses. From there the spike travels back down to the different neurons.

5.2 Performance Improvements

5.2.1 Identifying main Performance Bottlenecks

Before improving we had to find out which objects and functions cause the major performance issues. This was done by inspecting different parameters like the used memory and the frames per second by using the build in tools of the Firefox[8] web browser. The main investigation was done as part of my internship (Weidner 2019).

It turned out that the memory consumption causes the major performance problem. It came from a lot of PixiJS graphic objects, especially for larger networks. Those were used to build up the whole wafer. Due to the relatively large amount of synapses in a typical network, those were the main performance limiters.

5.2.2 Improvements

At first we removed some unnecessary or doubled calculations. Moreover we set the synapse grid, which was a white grid behind the synapses, invisible by default, because it is not necessarily needed and contains a lot of graphic objects as well.

As mentioned before, synapses are the main problem. Therefore we chose a simple style. Two rectangles were implemented, one shows the color of the route and the other one signalizes the synapse type as shown in Figure 17b. To reduce the amount of graphic objects, we combined all synapses of one route to one texture which is called sprite in PixiJS. Therefore one cannot address them individually after they are drawn once, but it increases the performance substantially.

For large networks we reached a memory reduction of about 30%. The main improvement is a more fluid zooming. For the user experience this is more important than the initial loading time.

[8] *Firefox* `https://www.mozilla.org/de/firefox/`

6 Outlook

6.1 Simulation

As already mentioned at the beginning, the overall goal is to get the cortical microcircuit running on the BrainScaleS hardware. We could reproduce the major results from Albada et al. (2018) and Potjans and Diesmann (2014b) in a new PyNN Nest simulation. The PyNN script can now be modified to run on hardware. Therefore I recommend to define three to five benchmark parameters that specify how good the network on hardware fits the initial network behaviour of Potjans and Diesmann (2014b). Those benchmark parameters can be derived from the rate distributions, the synchrony and the irregularity. Furthermore they should quantify the current state of the network.

The following steps should all happen while keeping benchmark parameters as good as possible. At first, the network should be scaled down to a size that fits on one wafer, to about 5% to 10%. Then the synapses should be changed to conductance based synapses, which are integrated, in the BrainScaleS-1 System. Afterwards all hardware parameters should be simulated as real as possible. For example, the weights are only discreetly adjustable (4 Bit and offset) and the delay depends on the distance on the chip and might be faster even when reaching a multiple wafer cortical column network. Furthermore, the jitter from hardware has to be simulated too. The synapses lost due to the mapping can also be simulated. It might be possible to compensate the synapse losses by increasing the synaptic weights.

Parallel, a simple version of the network should be implemented on hardware as soon as possible to figure out yet unknown problems early. By continuously testing of hardware and simulation with the defined benchmark parameters steady improvements can be reached.

After the limits of one wafer are reached, the cortical column network should be scaled back up again and a multiple wafer cortical column should be implemented. The inter-wafer communication is still under development.

On the other side, the benchmark parameters are a good tool to verify the hardware state by running the experiment nightly.

6.2 Mapping

The mapping is now in a state where about 10% of the whole cortical column network can be mapped on a single wafer with about 5% synapse loss. It is assumed that it does not matter much how many synapses are lost, but rather how the realized ones are

distributed over the populations. As this is quite equal in our case, this might be an advantage.

After all hardware adaptions to the simulation it might be useful to manually improve the mapping by varying different not yet tested placement parameters. At the moment it makes no sense to do so, because it is assumed that there will change a lot until the cortical column will run on hardware.

It also has to be considered that some parts of the BrainScaleS-1 Wafer are defect due to different production issues. Via calibration, the not working components are getting blacklisted, which becomes important concerning larger networks. The mapping can use this blacklisting data to only use working components. The mapping with blacklisting was not tested in this thesis. When doing so with the finally scaled network, the distribution of the loss has to be tested again.

Because of hardware limitation, the absolute maximum size of a column network that can be realized is 10 Mio. synapses on a single wafer. With the current scaling algorithm, which might be changed, this leads to a cortical column network with 18% in size.

6.3 Web Visualization

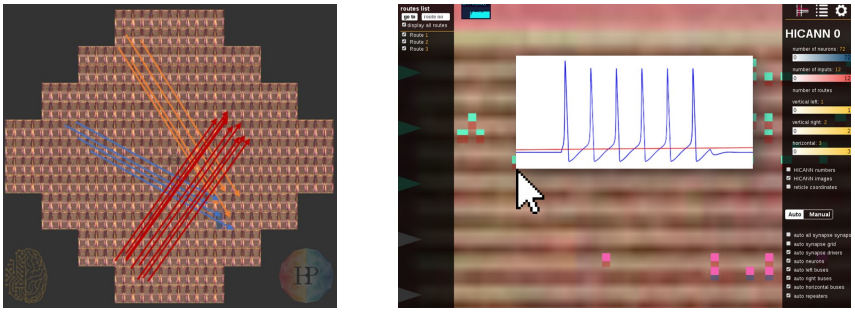
There are still a lot of things that can be improved in the web visualization. The zooming is still not perfectly fluid and the loading still takes too long. The synapses are currently not drawn perfectly in the same size.

Especially for the cortical column network, but also for other large networks, it would be nice to see which synaptic connection could not be implemented. Figure 18a shows what this could look like. Moreover it would also be great to see not only lines but to get statistical information about the losses, similar to Figure 10.

The next step would be to access the mapping via the visualization. With that it will be possible to change the mapping by mouse and with a good overview.

Moreover results of experiments or even live experiments could be shown in the web visualization. The activity of neurons and synapses could be displayed to really see what happens on the chip. An example is shown in Figure 18b.

Additionally the web visualization should support multi wafer networks, as soon as they are realizable, as well as future large scale chip layouts.



(a) (b)

Figure 18: (a) Example of what it could look like in the future to indicate not realized synaptic connections between neurons in the web visualization by airlines. (b) Example for experiment results in the visualization. Shown is the membrane trace of the neuron in the mouse-hovered column.

In my opinion it would not make sense to design the whole experiment in this web visualization, because it shows the mapped network on the chip, which does not reflect the abstract network structure. Therefore another graphical tool would be nice, to make the BrainScaleS system accessible for everybody. If realized, it would make sense to use this abstract tool to create networks for other platforms like with PyNN.

Personally I think, a web based visualization software including all parts of the experiment, from modeling over hardware mapping to plotting results, similar to google's TensorBoard (Martín Abadi et al. 2015) for machine learning, should be the long term goal.

7 Discussion

Several things were improved to realize the cortical column network from Potjans and Diesmann (2014b) on the BrainScaleS hardware platform.

First of all, the mapping to the hardware was investigated. It turned out that a network with about 3 Mio. synapses is mappable on one wafer chip, which is equivalent to a cortical column network of 10%. Thereby it is important to mention that the amount of needed synapses depends strongly on the not yet finished network size scaling algorithm. Specific improvements can increase the realizable network size even further.

The results of the simulated cortical column network are similar to the ones of Albada et al. (2018) and Potjans and Diesmann (2014b) and can serve as a good basis for further, more hardware orientated, simulations. The small differences concerning the synchrony and the rate distributions should be further investigated. In addition, the PyNN code can also be used to run on hardware.

To the web visualization, I added various important information like synaptic weights and types. The memory was reduced by about 30% leading to a more fluid zooming. This helps to get a better overview of the mapped network.

I think it would be nice to have an easier way to get started in the Electronic Vision(s) group. During my time here a few documentations for different software have already been added, but some more basic and easier examples would be nice. As there are so many confusing names of different hardware and software, it would be helpful to create a short overview at one place.

In my opinion it is also quite hard for external users to run experiments on the BrainScaleS hardware. In general, I think a very simple graphical interface to run small networks on the hardware to lower the entry threshold for biologists and other interested people is worth aspiring. Thereby it has to be taken into account that compared to software simulations neuromorphic hardware is way more complicated and specific, but it also has big advantages in run time and energy consumption.

In conclusion, I can say that I enjoyed it a lot getting an introduction into the very interesting field of neuromorphic computing.

Bibliography

Albada, Sacha J. van et al. (2018). "Performance Comparison of the Digital Neuromorphic Hardware SpiNNaker and the Neural Network Simulation Software NEST for a Full-Scale Cortical Microcircuit Model". In: *Frontiers in Neuroscience* 12, p. 291. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00291.

Boell, Richard (2018). "Visualization of Mapping and Routing of the BrainScaleS System". Bachelorarbeit. Universität Heidelberg. URL: http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3760.

BrainScaleS System - Neuromorphic Computer Coming Online (2016). URL: https://www.uni-heidelberg.de/presse/news2016/pm20160316-neuromorphic-computer-coming-online.html.

Cain, Nicholas et al. (2016). "The Computational Properties of a Simplified Cortical Column Model". In: *PLOS Computational Biology* 12.9, pp. 1–18. URL: https://doi.org/10.1371/journal.pcbi.1005045.

Corporation, Mozilla (2019). *Firefox*. URL: https://www.mozilla.org/de/firefox/.

Davies, Mike (2019). "Benchmarks for progress in neuromorphic computing". In: *Nature Machine Intelligence*. URL: https://www.nature.com/articles/s42256-019-0097-1.

Davison, Andrew et al. (2009). "PyNN: a common interface for neuronal network simulators". In: *Frontiers in Neuroinformatics* 2, p. 11. URL: https://www.frontiersin.org/article/10.3389/neuro.11.011.2008.

Emscripten (2019). URL: https://emscripten.org.

Freedman, David and Persi Diaconis (1981). "On the histogram as a density estimator:L2 theory". In: *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* 57.4, pp. 453–476. URL: https://doi.org/10.1007/BF01025868.

Furber, Steve et al. (2013). "Overview of the SpiNNaker System Architecture". In: *Computers, IEEE Transactions on* 62, pp. 2454–2467.

Gerstner, Wulfram et al. (2014). *Neuronal Dynamics - From single neurons to networks and models of cognition*. URL: https://neuronaldynamics.epfl.ch/online/index.html.

Hodgkin, Alan L. and Andrew F. Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of Physiology*. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1392413/.

Jeltsch, Sebastian (2014). "A Scalable Workflow for a Configurable Neuromorphic Platform". PhD thesis. Universität Heidelberg. URL: http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3052.

Klähn, Johann (2017). "Training Functional Networks on Large-Scale Neuromorphic Hardware". Master. Universität Heidelberg. URL: http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3641.

Martín Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. URL: http://tensorflow.org/.

Merkt, Benjamin, Friedrich Schüßler, and Stefan Rotter (2019). "Propagation of orientation selectivity in a spiking network model of layered primary visual cortex". In: *PLOS Computational Biology* 15.7. URL: https://doi.org/10.1371/journal.pcbi.1007080.

Musk, Elon and Neuralink (2019). "An integrated brain-machine interface platform with thousands of channels". In: *bioRxiv*. URL: https://www.biorxiv.org/content/early/2019/07/17/703801.

Naud, Richard et al. (2008). "Firing patterns in the adaptive exponential integrate-and-fire model". In: *Biological Cybernetics*. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2798047/.

Nowke, Christian et al. (2015). "Integrating Visualizations into Modeling NEST Simulations". In: *Frontiers in Neuroinformatics* 9, p. 29. URL: https://www.frontiersin.org/article/10.3389/fninf.2015.00029.

Passenberg, Felix Constantin (2019). "Improving the BrainScaleS-1 place and route software towards real world waferscale experiments". Masterarbeit. Universität Heidelberg. URL: http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3914.

Petrovici, Mihai A. (2015). "Form vs. Function - Theory and Models for Neuronal Substrates". PhD thesis. Universität Heidelberg. URL: https://www.springer.com/de/book/9783319395517.

PixiJS 5 (2019). URL: https://www.pixijs.com/.

Plesser, Hans E. et al. (2007). "Efficient Parallel Simulation of Large-Scale Neuronal Networks on Clusters of Multiprocessor Computers". In: *Euro-Par 2007 Parallel Processing*. Ed. by Anne-Marie Kermarrec, Luc Bougé, and Thierry Priol. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 672–681.

Potjans, Tobias C. and Markus Diesmann (2014a). "Supplemental Material The cell-type specific cortical microcircuit: relating structure and activity in a full-scale spiking

network mode". In: *Cerebral Cortex*. URL: https://academic.oup.com/cercor/article/24/3/785/398560#supplementary-data.

Potjans, Tobias C. and Markus Diesmann (2014b). "The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model". In: *Cerebral Cortex*. URL: https://doi.org/10.1093/cercor/bhs358.

"PyNN Documentation" (2019). In: URL: http://neuralensemble.org/docs/PyNN/.

Rhodes, Oliver et al. (2019). "Real-Time Cortical Simulation on Neuromorphic Hardware". In: *The Royal society - Philosophical Transactions A*. URL: https://arxiv.org/abs/1909.08665.

Rosenblatt, Frank (1985). "The perceptron: a probabilistic model for information storage and organization in the brain". In: *Psychological Reviews*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf.

Rowley, Andrew G. D. et al. (2019). "SpiNNTools: The Execution Engine for the SpiNNaker Platform". In: *Frontiers in Neuroscience* 13, p. 231. URL: https://www.frontiersin.org/article/10.3389/fnins.2019.00231.

Senk, Johanna et al. (2018). "VIOLA-A Multi-Purpose and Web-Based Visualization Tool for Neuronal-Network Simulation Output". In: *Frontiers in Neuroinformatics* 12, p. 75. URL: https://www.frontiersin.org/article/10.3389/fninf.2018.00075.

Thomson, Alex M. et al. (2002). "Synaptic Connections and Small Circuits Involving Excitatory and Inhibitory Neurons in Layers 2-5 of Adult Rat and Cat Neocortex: Triple Intracellular Recordings and Biocytin Labelling In Vitro". In: *Cerebral Cortex* 12.9, pp. 936–953. URL: https://doi.org/10.1093/cercor/12.9.936.

Twitter Account of Electronic Vision(s) group (2019). URL: https://twitter.com/brainscale.

TypeScript (2019). URL: https://www.typescriptlang.org.

Weidner, Jonas (2019). *Performance Increase of the Visualization Software for the Neuromorphic Hardware System BrainScaleS*.

8 Acknowledgements

I would like to thank...

... Dr. Johannes Schemmel for giving me the chance of working on this interesting topic.

... my direct supervisor Eric Müller for introducing me to the group and for the great support, as well as for prove reading my thesis.

... Sebastian Schmitt for reading my internship report.

... Hartmut Schmidt for helping me a lot with all the Nest and PyNN stuff and also for prove reading my thesis.

... Felix Passenberg for all the mapping algorithms and explanations.

... Oliver Breitwieser, Jakob Kaiser, Malte Wehrheim, Philipp Dauer, Aron Leibfried, Christian Mauch and Quirinus Schwarzenböck for answering all of my questions and for the interesting discussions in the office.

... Daniel Lange for the overnight prove reading of this thesis.

... the rest of the Electronic Vision(s) group for the great working atmosphere.

... my girlfriend for correcting many of my numerous spelling mistakes and overall for the moral support.

... my family and friends for the great support during my whole studies.

Statement of Originality (Erklärung)

I certify that this thesis is the product of my own work and no other than the cited sources were used.

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, den 7.10.2019
Jonas Weidner