Department of Physics and Astronomy

University of Heidelberg

Master thesis

in Physics

submitted by

Felix Constantin Passenberg

born in Berlin

2019

Improving the BrainScaleS-1 place and route software towards real world waferscale experiments

This Master thesis has been carried out by Felix Constantin

Passenberg

at the

Kirchhoff-Institute for Physics

under the supervision of

Prof. Dr. Karlheinz Meier †

Dr. Johannes Schemmel

Master Thesis

Felix Constantin Passenberg

June 7, 2019

Abstract

The BrainScaleS-1 wafer-scale hardware supports the emulation of large-scale spiking neural networks. Obtaining a valid hardware configuration involves placing and routing the user-defined neural network toplogy onto the hardware. This thesis describes the work performed by the author to improve this process. Previous misbehaviour was corrected as well as the emulation of larger networks was enabled. For example, the lossless mapping results archived on a real-world-like wafer by the benchmark suite are better by a factor of 4 to 15 depending on the topology of the network. Some networks that were not able to be mapped without loss can be mapped now. On that way the calibration database was changed and is queried for hardware limitations. The interface of the placement algorithms were changed to allow the user to select and configure predefined placement strategies or create own strategies. The new placement strategies include simple liner, and also algorithms that cluster populations or neurons based on their connectivity.

Zusammenfassung

Das BrainScaleS-1 Wafer-scale System ermöglicht die Emulation großskaliger spikender neuronaler Netzwerke. Um eine korrekte Konfiguration für Experimente zu erhalten ist eine Abbildung (mapping) der vom Nutzer gegeben gegeben Netzwerke auf die Netzwerkstruktur der Hardware erforderlich. Dise Arbeit handelt von den Verbesserungen an der Abbildungssoftware (mapping-software). Bestehendes Fehlverhalten wurde korrigiert und die Emulation großer Netwerke wurde ermöglicht. Beispielsweise können die Netwerke der Benchmark-suite auf einem Modell-Wafer, welcher der zukünftig zu erwartenden Qualität entspricht, auf die 4- bis 15-Fache Größe skaliert werden, jeweils abhängig von der Topologie des Netzwerks. Netze die zuvor nicht ohne Verluste realisiert werden konnten, können nun realisiert werden. Um die Resultate zu erlangen wurde die Callibrationsdatenbank verändert und nun auch gelesen, um die Hardwarebeschränkungen während des Abbildevorganges abzufragen. Die Architektur des Plazier-Algortihmus wurde verändert um den Nutzern eine Schnittstelle zu bieten, über die vergefertigte Strategien wählbar und konfigurierbar sind, und auch um neue Strategien zu definieren. Die neuen Strategien beinhalten einfache lineare Strategien, aber auch Strategien, die basierend auf Verbindungen von Populationen und Neuronen Gruppierungen vornehmen.

Contents

1. Introduction

The *Human-Brain-Project* (HBP) is an EU Flagship initiative to accelerate science. It provides infrastructure for scientists to research on brain related topics [27].
It is separated into multiple subprojects. One of them, SP9, focuses on Neuromorphic Computing. At the Heidelberg University the BrainScaleS system is located. It emulates *Spiking Neural Networks* (SNN) using custom made devices. They model the characteristics and interaction of biological neurons using the *High Input Count Analog Neural Network* (HICANN) chip which simulates neurons by the adaptive exponential integrate-and-fire (AdEx) [7] model in analogue circuits. The human brain is build of roughly 86×10^9 neurons [3], a single wafer in the BrainScaleS-1 System contains only about 200×10^3 neurons. Ordered by the number of neurons this places the BrainScaleS-1 wafer on a similar level as the Drosophila [11]. Among the main advantages of the mixed-signal system simulation of neural networks compared to simulation on classical computers is the acceleration factor. Biological time is accelerated by a factor of approximately 10^4. While the energy efficiency per synaptic operation is another remarkable advantage of the wafer system compared to classical computers.
In order to simulate neural networks on the system, they need to be translated from the biological description to a hardware configuration. The `marocco` software handles the place-and-route part [2]. It was developed during the PhD work of S. Jeltsch [19], a former group member.
`marocco` was initially developed when the hardware system was still under development, so it assumes the hardware to behave as specified. But alike most real world experiments it turned out there were problems which were unconsidered during `marocco`s development. The software in its old state was not to able handle the real behaviour of the hardware well.
This thesis deals with fixes and improvements to the `marocco` software to allow the up-scaling of neural networks on real world wafer systems.

2. The Wafer-scale system

To run experiments on the BrainScaleS system the biological description of the neural network written in `PyNN` [12] with their `Populations` and `Projectons` has to be translated to a valid hardware configuration. `marocco` handles the place and route part, therefore hardware resources used during mapping are discussed in this chapter.

2.1 Hardware resources

This section is structured by following the signal from a source neuron to another target neuron as pictured in 2.1. The resources used on the way are shortly described. To get a more detailed description consult, e.g., the PhD thesis of S. Jeltsch [19].

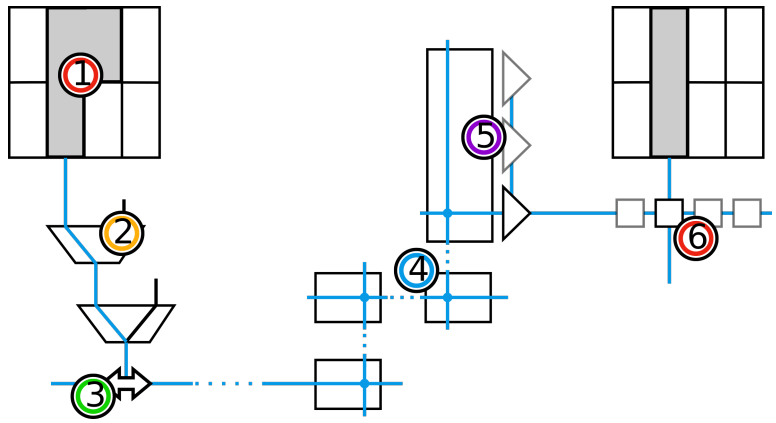


Figure 2.1: The signal of a Sending Neuron (1) is merged in the Merger Tree (2) and injected to the Layer1 network at SPL1 repeaters (3). Layer1 crossbars (4) are used to switch the signal to Synapse drivers (5) where the signal is injected into the synapses (6) [19]

2.1.1 HICANN

The building block of the wafer is the *High Input Count Analog Neural Network* (HICANN) chip. Is it possible to use a single one of these chips to run emulations of Neural Networks. The HICANN is a mixed-signal chip. The hardware (HW) neurons operate as analogue electronics to physically simulate biological (BIO) neurons. Signals between HW neurons is processed digitally. A total of 384 HICANNs are on a single wafer. They are connected to their neighbours to allow the simulations of larger networks.
The HICANN holds 512 NEURONS (sec. 2.1.2) that are grouped into 8 NEURONBLOCKS (sec. 2.1.3). Spikes that are created by Neurons get routed via the MERGER TREE (sec. 2.1.4) to the LAYER1 (sec. 2.1.5) network. After travelling in through the Layer1 network, the signal is injected via SYNAPSEDRIVERS (sec. 2.1.6) into the SYNAPSE ARRAY (sec. 2.1.7) of the targeted neuron.
Connections between neighbouring HICANNs are established on the Layer1 network.

2.1.2 Neuron

Neurons, also referred to as `denmem`, are analogue circuits that are designed to simulate biological neurons as adaptive exponential integrate-and-fire (AdEx) neurons [7].The 512 hardware neurons are split into 8 `NeuronBlocks` containing 64 neurons each. They are ordered as 32 neurons an the top row and 32 on the bottom row. Neighbouring neurons on the same `NeuronBlocks` can be connected to increase the input synapse count. The current software requires that Neurons on the top and bottom are connected. Possible neuron sizes range from two to 64 in steps of two.

2.1.3 Neuron Blocks

`NeuronBlocks` build a logical unit containing 64 Neurons. The signal from neurons on the same `NeuronBlock` are bundled on the same Layer1 bus. To distinguish the signals of the neurons, a different address is used for each neuron. The address consists of 6 bit allowing 64 possible addresses while 5 addresses are reserved for special purposes.

2.1.4 Merger Tree

The signal of multiple Neuron Blocks can be merged in the Merger Tree. At the top of the merger tree are background generators, that are used for signal locking in

the digital layer1 network. At the bottom of the merger tree external spikes can be injected. This is used to stimulate the network during runtime of the experiment.

2.1.5 Layer1 network

The Layer1 network is a circuit switched network consisting of 256 vertical and 64 horizontal buses (`VLine`, `HLine`) which are interconnected by a sparse matrix. In such it is not possible to switch arbitrarily to another bus.
The 8 outputs of the Merger Tree enter the Layer1 network at the HLines $6, 14, 22, 30, 38, 46, 54, 62$. To prevent an immediate blocking of the signal, the buses are rotated by 2 on every transition to a neighbouring HICANN. So after 4 steps along x-direction the signal will reach another possible insertion point. A complete round is done after 32 steps in horizontal direction. In vertical direction the 256 buses are split to the left and right half of the chip. Rotation is done only among 128 buses, thus a full round is done after 64 steps. After two steps in vertical direction the VLine is again accessible from an HLine that might be sending. With the 8 switches and all senders active 16 HICANN might be located vertically before VLines are exhausted.
To feed the signal into the synapse array a switch from a VLine to the synapse drivers has to be activated. It is possible to switch to the synapse drivers on the neighbouring HICANN.

2.1.6 Synapse Drivers

There are 220 Synapse Drivers on a HICANN (version 4). 55 of them are in each corner. Multiple Synapse Drivers can be chained to increase the number of synaptic inputs for the neuron, while putting less capacitive load on the Layer1 network drivers.
One Synapse Drivers has access to two rows of 256 synapses. Two bits of the address are decoded in the Synapse Drivers to select 1 out of 4 synapses. A total of 128 synapses are now active. Additionally to the 4 remaining bits an analogue signal is sent to the synapses.

2.1.7 Synapse Array

The Synapse decodes the other 4 bits of the address. If it matches the line of the analogue signal is connected to the neuron. A neuron of size two (x=1, y=2) has access to 220 synapses in each the top and the bottom synapse array.

2.2 Routing Limitations

Resources might only be used once, so the routing process gets harder with progressed routing, as the resources availability reduces. This is noticeable during routing, because after traversing 4 HICANNs in horizontal direction another signal might be injected by the Merger Tree, which will already occupy the bus.
By the regular order in the switching matrix of the Layer1 network shown in figure 2.2, the periodicity of the signal at a given HICANN is already defined by the sending HICANN and the merger tree configuration. A Similar statement holds for the switches between `VLines` and Synapse Drivers. At time of injection the signal is already destined to access one of 28/26 Synapse Drivers (14/13 per side) of the total 220 drivers at the target.
Nonetheless it is not possible to use any combination of the switches, as it would lead to too large capacities which the layer1 drivers are not able to drive. To keep the signal stable there must not be used more than

- 1 Layer1 Crossbar switch;
- 1 switch into the Synapse Array;
- 3 Synapse Drivers in a chain;

Different timing settings or different hardware might allow to change these constraints.

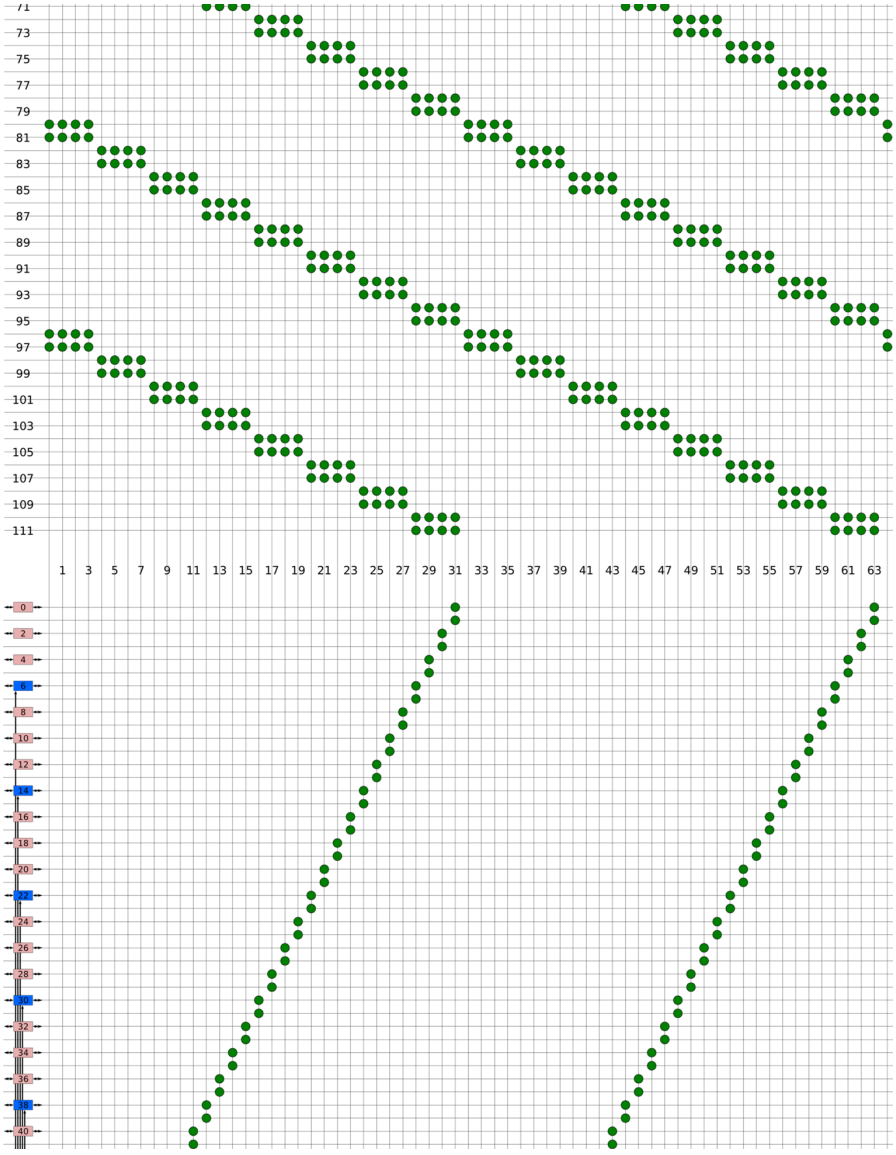


Figure 2.2: A cropped view on the switch matrix. In the top are switches between vertical buses (`VLines`) and wires to the Synapse Drivers. The Bottom shows the crossbars between `HLines` and `VLines`. Every **crossbar** is a **green** dot. The small **lavender** boxes on the left of the horizontal buses are the **Drivers**. On every second line the driver is on the next HICANN. In **blue** are the **SPL1** repeaters, at which the signal from the Mergertree is injected.

3. Encountered state of the place and route algorithm

„Place and route“ or „mapping“ in terms of this work is the process of translating the biological representation of a neural network written in `PyNN` [12] to a valid hardware configuration. Figure 3.1 shows the mapping problem. Hardware neurons, buses and synapses are configured in a way to represent neurons and connections of the described network.
The software stack to solve the mapping problem for the wafer system was a joint effort of multiple PhD thesis. The architecture of the software was provided by E. Mueller [24]. Hardware configuration and calibration was provided in the PhD thesis of C. Koke [22]. M. Kleider [21] did additional work on calibration. The place and route software `marocco` was written during S. Jeltsch PhD thesis[19]. `marocco` splits the mapping problem into three steps:

1. Neuron placement
2. Layer1 routing
3. Synapse routing

The placement itself is not difficult, but it has to provide good characteristics, such that the Layer1 routing will find routes on the limited resources available.
The routing problem is similar to the *rectilinear Steiner tree problem*, that is known to be NP-hard [16]. To generate a mapping in reasonable time the algorithm has to simplify the problem. The properties of the biological network and of the hardware are used by heuristics.
Neuron placement and Layer1 routing affects the global state of the hardware, and therefore cannot be parallelised in a simple way. Synapse routing could be parallelised easily on HICANN granularity.

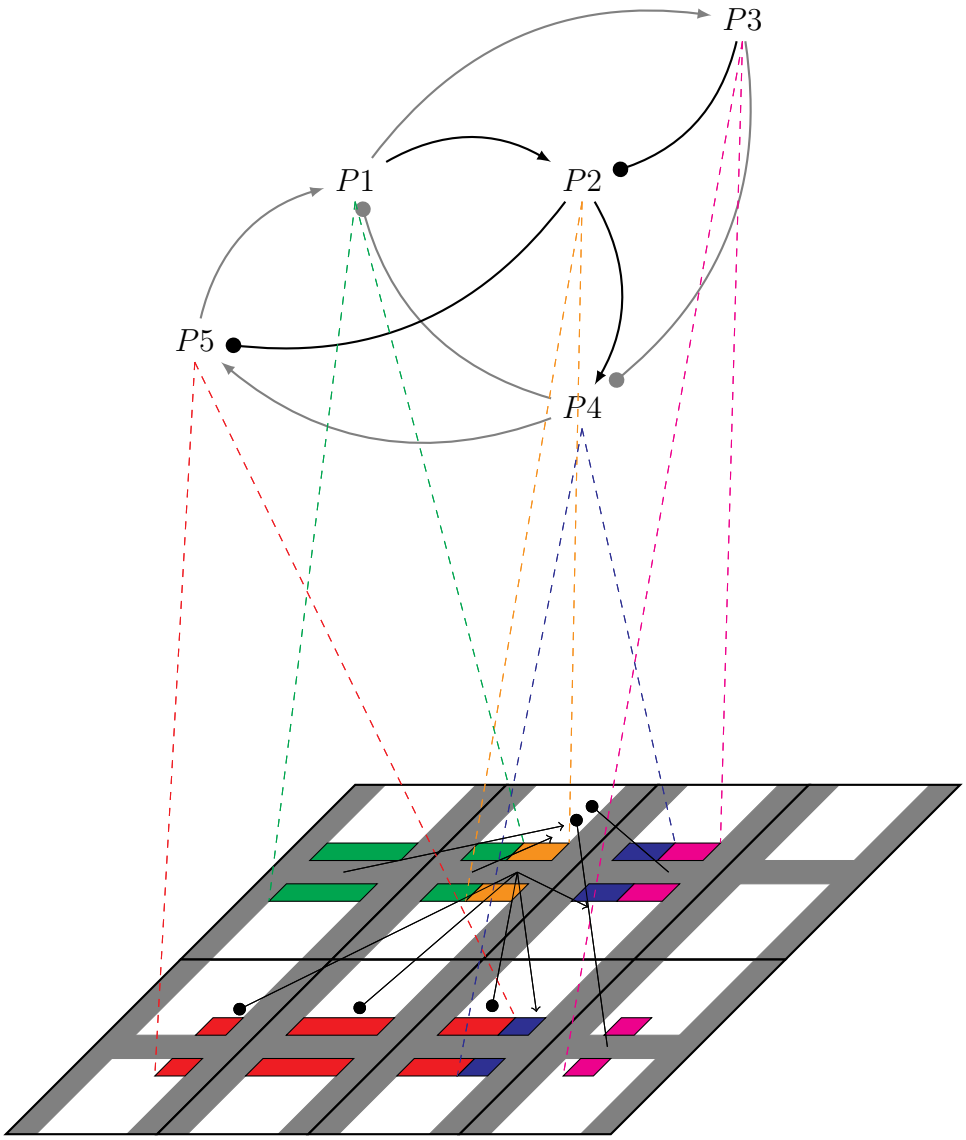


Figure 3.1: Place and route is the translation from the biological representation of a network in the top to a hardware configuration in the bottom. A possible placement configuration is indicated as coloured neurons. A population may span across multiple HICANNs. For simplicity in the *hardware* configuration „only“ airlines to and from population **P2** are shown. On the hardware they have to be routed along the grey **Layer1 Bus** network.

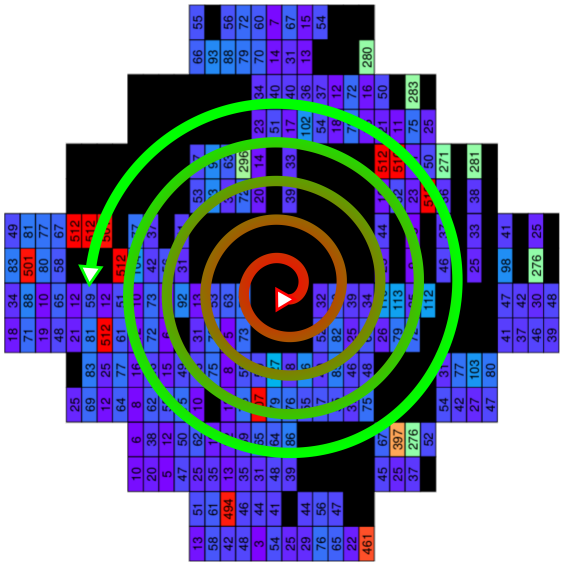


Figure 3.2: the spiral is build by sorting Neuron Blocks first by the number of available Neurons, then by the spiral metric. In the background are blacklisting data for the Wafer 33. One sees that the availability of neurons is scattered, which results in neurons being scattered across the Wafer, as it starts at red via orange and cyan to light and dark blue.

3.1 Neuron placement

When I started working on the software the placement strategy was to sort the `NeuronBlocks` by their available space and if they are equal a spiral order around the centre of the wafer is applied. Neuron Blocks with smaller space available were used first. The populations were sorted by their initialisation order which is equal to their ID.
A figure picturing this strategy is given in 3.2, the order of priority starts at red via orange and cyan to light and dark blue, This method finds a solution to the placement problem in n-log-n time, $\mathcal{O}(\mathrm{NB}\,log(\mathrm{NB}) + \mathrm{Nrn}\,log(\mathrm{Nrn}) + \mathrm{Nrn})$. The initial sorting dominates the runtime.
The strategy turns out to be bad for the currently existing wafer system. As some neurons are blacklisted on the real system the ordering of the `NeuronBlocks` lets the placer jump from location to location. When a population is larger than a `NeuronBlock`, the population is torn apart and placed over multiple `NeuronBlocks` scattered over the wafer. This resulted in two major problems.

1. By starting with `NeuronBlocks` with less space, the first populations are split with a higher probability. This leads to more injection points of the

signal which makes it difficult to realise all connections as more `VLines` are required at the target and in turn more Synapse Drivers are required.

2. The targeted neurons are also scattered across the wafer. This leads to longer connections on the Layer1 network and more branches of the signal path. That will use more network resources and leads to blocking of the network.

3.2 Layer1 routing

For routing two methods are available: a Dijkstra and a Backbone router [19], [14].

Dijkstra The Dijkstra router [13] [6] is not favoured, as it branches quite often and thus uses a lot of the rare `L1Crossbars`, and it does not handle the routing limitations of the hardware well. An additional difficulty is, that we do not have a single target vertex, but there are 512 vertices possible to be used as target. All of the 512 `VLines` can be used to feed the signal into the requested synapse array.

Backbone The second router is similar to a single-trunk Steiner tree router but we call it Backbone router. What [10] calls trunk, is called backbone in our software. We place the backbone on the `HLine` the signal of the neuron is injected on. So there is no optimisation in the placement of the backbone. A minimal single-trunk Steiner tree may be found in linear time [10].
The Backbone router utilised in `marocco` walks the horizontal bus until it reaches the x-coordinate of every target. If it could not reach that far, for example because the bus is already occupied by another route or because there is no horizontal neighbour ,which is a consequence of the wafer being round, a detour is tried by walking a short distance in vertical direction.
After the Backbone has been established the router considers to branch from each segment of the horizontal backbone to a `VLine`. If there are targets at x-coordinate of the segment, the 8 `VLines` are tried to be extended to reach all the targets. The `VLine` that reaches the most targets is set to be used. The synapse arrays in this x-coordinate are marked as reached by the `VLine` and removed from the target list. There are two problems with this implementation of the Backbone router. Figure 3.3 pictures them.

- The Backbone router did not consider the HICANNs on neighbouring x-coordinates as targets even though it is possible to inject signal into the Synapse Arrays of the neighbouring HICANNs. This leads to more branches being made and more buses in use. The chance of congestion is increased.

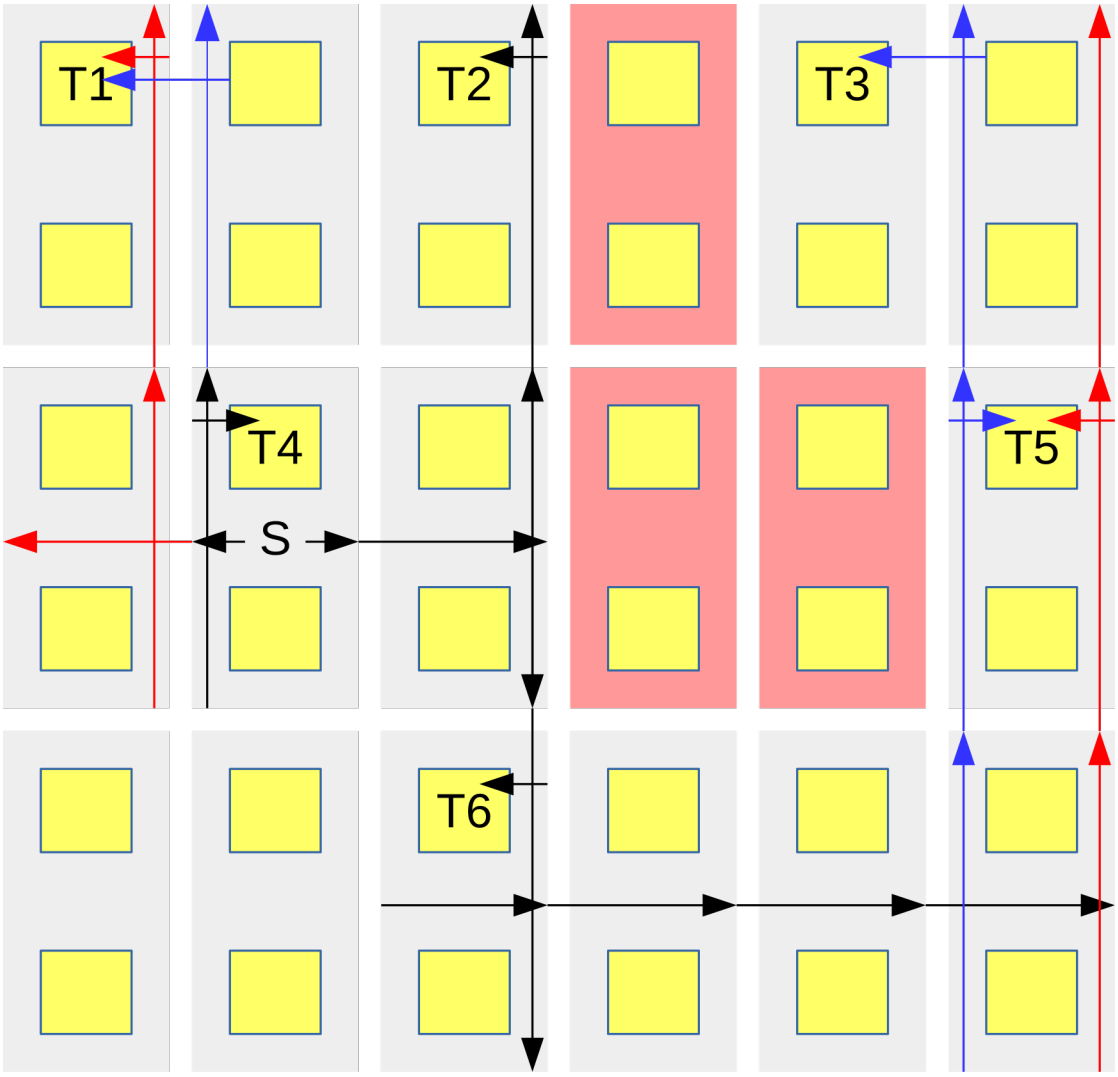


Figure 3.3: Example for routing from a source S to targets $T[0\text{-}6]$. The Backbone router was in a state that could produce a result coloured in **black** and **red**. While it would be possible to produce a result coloured in **black** and **blue**. The **blue** result is better than the **red** as it reaches $T3$, which is blocked by blacklisted or occupied HICANNs coloured in light red, and consumes less resources to reach $T1$.

- In cases where a detour has been performed, the router did not take note of the used `L1Crossbar`. In turn it happened, that another `L1Crossbar` might be set, during the branching step. On the current hardware this violates the constraints dictated by the power of the `L1Repeater`s, which are not capable to drive two crossbars at the current timing settings.

The worst case runtime of the Backbone router is cubic $\mathcal{O}(\mathrm{Nrn} + |V|^2)$ [19]. While in average cases with few detours the expected runtime is $\mathcal{O}(\mathrm{Nrn} + |V|)$. With $|V|$ the number of buses scaling with the number of Neurons.

3.3 User requirements

Some users of `marocco` provided networks for benchmark purposes. The benchmark suite [4] contains:

- a *fully visible Boltzmann machine* [1] figure 3.4a. Every neuron connects excitatory and inhibitory to all other neurons.
  The total number of neurons is scaled.

- an *Ising network* figure 3.4b. It is build of three populations. A two dimensional Ising network with wrapping edges [18]; A noise network by Pfeil (see below) with 500 ; and a bias neuron.
  The linear size of the network is scaled.

- *Pfeil's noise network* [26] figure 3.4c. Every Neuron connects randomly to 20 other neurons with an excitatory connection.
  The total number of neurons is scaled.

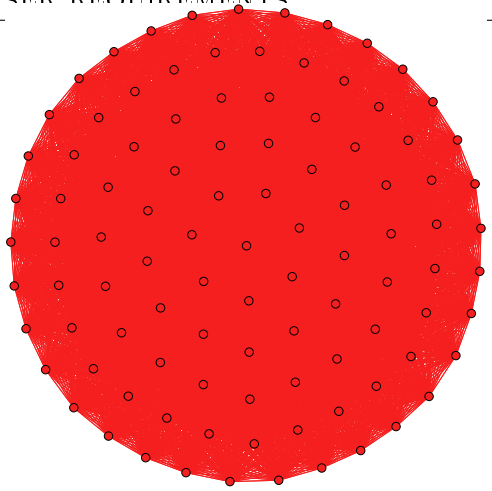
- a *random* network [8] figure 3.4d, Every Neuron connects randomly with a given probability to any other neuron. 10% and 30% connectivity are benchmarked.
  The total number of neurons is scaled.

- a *Restricted Boltzmann machine* [17] figure 3.4e There are two populations of the same size. Each neuron connects to all neurons of the other population using excitatory and inhibitory connections.
  The population size is scaled.

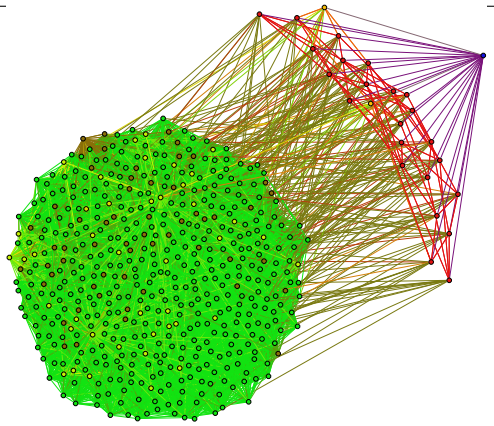
- and a network called *rbmLocalReceptiveFieldsNetwork (net6)* figure 3.4f. There are three populations, a visible layer, a hidden layer, and a label population. Between the hidden and the visible layer are local receptive fields. One edge of the hidden and the full label layer are connected

bidirectionally by excitatory and inhibitory synapses that are redundant by the number of the edge length. This seems to be a bug in the network description. To prevent confusion I will refer to it as `net6`.
The edge length is scaled.

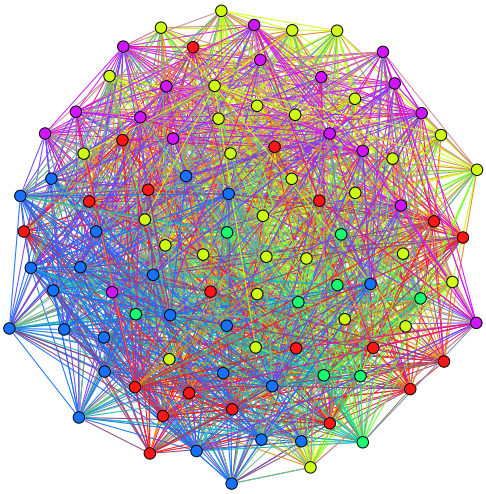
The relative amount of realised synapses and the runtime are used as metrics for a performance analysis of mapping process using these networks. The analysis can be found in section 5.



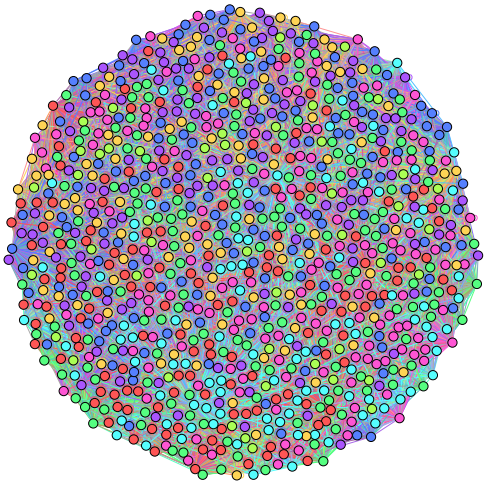
(a) fully visible Boltzmann machine, all neurons connect to all other excitatory and inhibitory



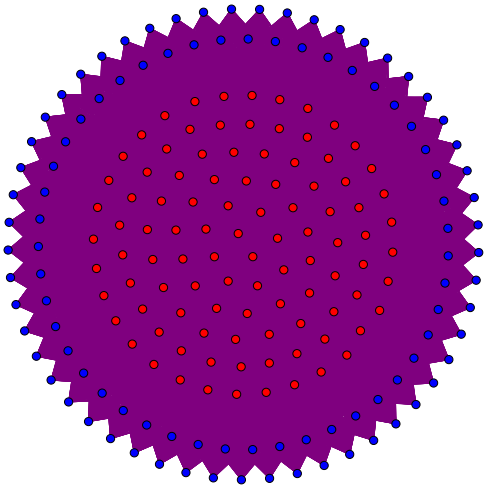
(b) ising network: a green random network is in the left, the red ising part in the middle and in blue on the right is a bias neuron



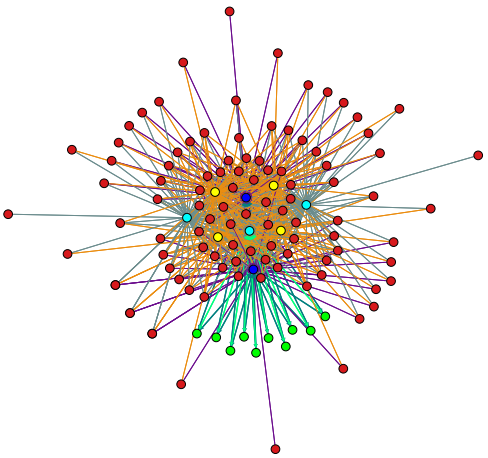
(c) Pfeils noise Network: each neuron connects randomly to 20 other neurons.



(d) random network: each neuron connects to other neurons by a chance of 10%



(e) Restricted Boltzmann Machine: there are two populations. Each neuron is fully connected to the other population, excitatory and inhibitory



(f) net6: there are 3 populations, visible layer partially connects to a hidden layer which is fully connected to a label layer

Figure 3.4: Networks of the benchmark suite. The figures were generated by Gephi [23] using graph drawing algorithms of T. Fruchterman and E. Reingold[15] and T. Kamada and S. Kawai [20].

4. Improvements to the place and route algorithm

To increase the amount of routed synapses, different approaches were followed. At first the `L1Routing` was modified to allow the backbone router to consider neighbouring HICANNs as targets and thus try to use less crossbars and have more branching possibilities. (sec. 4.2)
Second the placement of external sources has been changed to keep the synapse driver chain length within the constraints of the hardware. (sec. 4.4)
Third the architecture of the neuron placement was enhanced to allow different placement strategies. New simple and complex strategies are provided and an interfaces for the user to interact with them, or define their own placement strategies. (sec. 4.3)
In a fourth step a new Merger Tree algorithm was implemented to bundle the output of neurons onto as few `L1Buses` as necessary but still keeping the synapse drivers chain length within the constraints. (sec. 4.5)
Finally a backtracking placement strategy was implemented which re-sizes and re-locates neurons if local synapse driver chain length requirements are not met. (sec. 4.3.2.7)
During all these steps further improvements and deepened insight into the algorithms revealed bugs that did not yet occur to have an influence on the mapping results.

4.1 Functional bug fixes

Several problems were present in the software stack and produced invalid results or let the software crash in a segfault. These Changesets fix them:

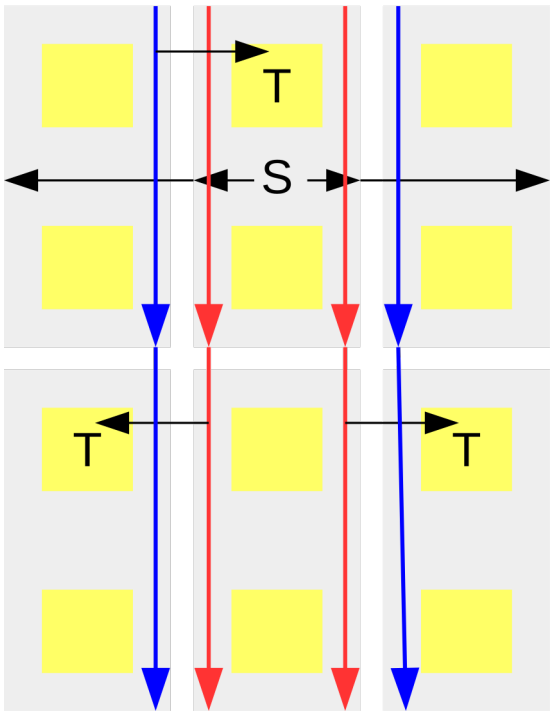


Figure 4.1: With the current hardware configuration it is not possible to set two switches from the same `HLine`. Therefore it is not allowed to use both **red** `VLines`. Though it is fine to use both **blue** buses or one of each colour.

4.1.1 Restrict L1Crossbar

Changeset: 4251

Using allowed configurations of the setup is crucial for an experiment. Therefore a new function was added to the `L1Routing` which evaluates allowed `L1Crossbars` from a predecessor list and the `HICANNManager`, who accesses the calibration database to query the constraints. The `L1BackboneRouter` and `L1GraphWalker` were modified to make use of the `L1Crossbar` restrictioning. Previous to CS4189 the crossbar configurations were invalid only in rare cases, like when a detour has been made directly at the source HICANN and there are also targets on the same x-coordinate. As both directions (east and west) are handled separately it happened that two crossbars were used from the same `HLine` to make the detour and to reach the target. Figure 4.1 pictures such a case in red. An allowed configuration includes at most one red line.

4.1.1.1 Restrict L1Crossbar after detours

Changeset: 6601

In cases where a detour due to the wafer edge was made, the predecessor list was not updated correctly. This led to multiple crossbars being used on the HICANN where the detour was initiated, if another `VLine` could reach more targeted HICANNs in the same x-coordinate.

4.1.2 Handle neighbouring HICANN injection correctly

Changeset: 4778

The Improved Backbone router 4189 increases the probability of `VLines` injecting the signal into the neighbouring HICANN significantly. This led to cases, where the same `VLine` index was used from the same HICANN and the neighbouring one to reach the same Synapse Array. Programmes interpreting the results and using `marocco` calls, like the WebVisu, failed in this case.

4.1.3 Augment L1Routes only with local synapses

Changeset: 5837

Another Bug affecting the result processing using `marocco` calls, is fixed by this changeset. Routes ending at the target `VLine` shall be augmented with the synapses they connect to. But all Synapses were appended to the route, this added Synapses on other chips as well. In consequence tools like the WebVisu crashed. This change checks if source and target neuron of the route matches them of the projection, before adding the synapse.

4.1.4 Skip disabled DNCs for external Sources

Changeset: 5842

The blacklisting database is able to mark DNCs as disabled for external spike sources, but the mapping did not use that data and used the blacklisted DNCs nonetheless. Disabling DNCs is important on HICANNs that can only be acessed via JTAG but not via the highspeed interface. Such a HICANN can be configured and used for routing, but no spike data can be transferred to and from this HICANN.

4.1.5 Wrong initialisation of the Predecessor list

Changeset: 5858

During routing the available resources are represented in a graph of type `boost::adjacency_list` [5]. An `std::vector` is used to store the predecessor of each vertex. Enumerating from 0 is common practice in informatics and so in this case. This means 0 is a valid vertex of the graph [6]. The predecessor list was initialised with 0 which lead to the problem, that the `L1Crossbar` restriction introduced in 4251 could not find a valid configuration for vertices connected to vertex 0. According to documentation [6] the right way to initialise the predecessor list is to set predecessors itself $p[v] = v$. This was implemented.

4.1.6 VLineUsage

Changeset: 6317

As there are 16 `VLines` competing for 14 synapse drivers the usage of those can be queried in the `VLineUsage` function. During branching the returned value is used in a scoring function to penalise high competition. When more than 12 `VLines` are set as target, the score for the following `VLines` is decreased. Until now the VLineUsage was queried, but not updated with new information, resulting in the same result for all requests.

4.2 Improved Backbone router

Changeset: 4189

The old implementation of the backbone router only checks for branches to `VLines` if there are targets on the same x-coordinate. And then only targets on this x-coordinate are marked as reached.
But as it is possible to inject the signal from the neighbouring HICANN, the router was changed to branch also if targets are lying on $x \pm 1$. And to inject the signal into the neighbouring HICANN, if there switch constraints and `VLineUsage` (see prev. CS 6317) allows it. The scoring function was changed to return a zero score if 14 `VLines` are set as targeting this synapse array. As the target is not set if a zero scare is retrieved, this shifts synapse loss due to insufficient synapse drivers to the Layer1 routing loss category.
Figure 4.2 shows in blue the new lines that are considered during routing from a source S to a target T. Adding the new branch functionality potentially doubles

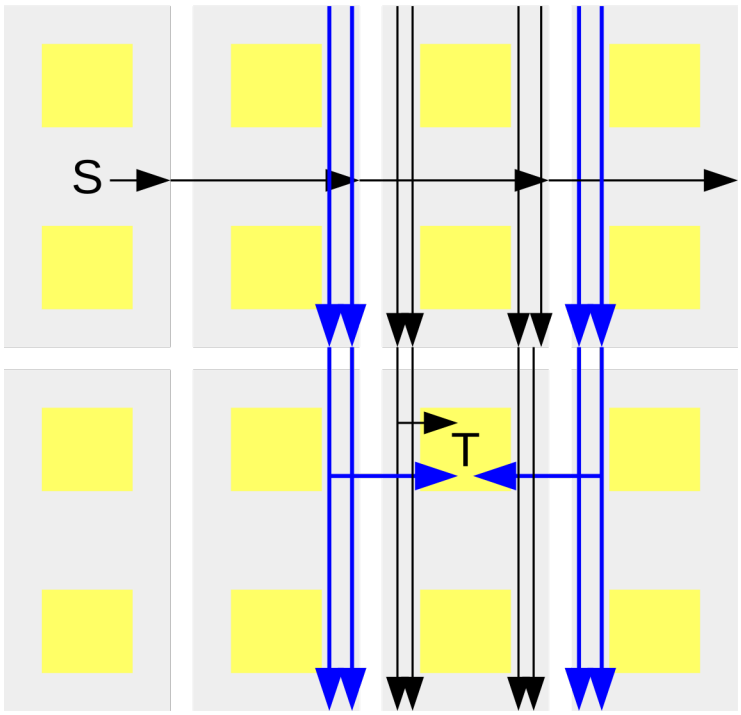


Figure 4.2: For simplicity only two instead of four `VLines` per side are sketched. The improved Backbone router considers also the buses on the neighbouring HICANNs coloured in **blue** to reach the Target T. Previously it was possible to switch from a given `HLine` to 8 `VLines` on the same HICANN. Now additional 4 `VLines` on both neighbouring HICANNs are available. All considered `VLines` are able to feed into T.

the amount of usable crossbars to circumvent congestion, but still the synapse drivers and switches into the synapse array limit the router.
If the last HICANN that shall be reached by the router is missed by only one in the x-coordinate a special *missed by one* handling kicks in, where it tries to feed the missed target from the side. This handling is especially useful close to the edge of the wafer, when target populations are not placed far away. Figure 4.3 visualises the behaviour. It saves a detour and thus two switches and two buses, but requires more synapse drivers on the closer side of the HICANN.

4.3 Placement algorithms

In the placement part larger changes have been made to the architecture of the algorithm. The change allows user to select from a predefined placement strategy,

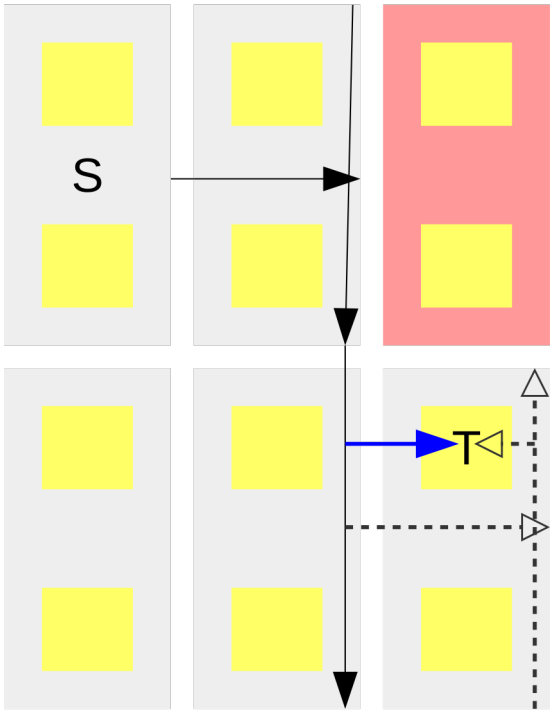


Figure 4.3: The improved backbone router allows to save switches in the special *missed by one* case. A blocked/occupied HICANN is sketched in **magenta**. Used buses are **black**, saved buses are dashed, and the new injection is done via the **blue** synapse line.

customise the selected one to suit their needs, or even define their own strategy. In addition to the conversion of placement strategy used until now, new strategies have been added.

4.3.1 Architectual change

In order to modularise the placement procedure the architecture has been changed towards a an Object-oriented class interface. The new design allows the easy extension and modification of placement strategies by deriving from existing ones. Additionally the classes are provided via the python interface `PyMarocco` to the user. To influence the behaviour of a placement class the user can create an instance of it and either do simple changes by the python interface of that class, or in order to have larger influence on the placement, inherit from it and and overwrite function-calls to suite their needs.

4.3.2 Algorithms

To give the user examples and provide good defaults the following placement strategies have been developed.

- base class: `PlacePopulationsBase` → 4.3.2.1
- placement strategy until now: `bySmallerNeuronBlockAndPopulationID` → 4.3.2.2
- linear placement: `byNeuronBlockEnumAndPopulationID` → 4.3.2.3
- linear ascending placement: `byNeuronBlockEnumAndPopulationIDasc` → 4.3.2.4
- cluster populations: `ClusterByPopulationConnectivity` → 4.3.2.5
- cluster neurons: `ClusterByNeuronConnectivity` → 4.3.2.6
- cluster neurons with chain length constriction: `ConstrainedNeuronClusterer` → 4.3.2.7

4.3.2.1 PlacePopulationsBase

Changeset: 4641

This Placer is designed as an base class to derive from it. Functions are provided as hooks for derived classes to call custom code at defined times. It would have been an abstract base class, but wrapping to python prohibits pure virtual functions.
The parameters passed to the `run` function-call are saved to local variables for the use in the later placement process.Only the recurring placement task of the last `PopulationSlice` on the last `NeuronBlock` is handled by the base-class. The last element is accessed because `std::vectors` provide removal of the last element in constant time [9]. If it is not possible to place the whole population on the current `NeuronBlock`, the Population is sliced into two parts and both are pushed back to the placement queue. If it is not possible to place more neurons on a `NeuronBlock` the `NeuronBlock` is popped from the queue.
Deriving classes shall sort the `PopulationSlice` queue and `NeuronBlock` queue according their needs.
The actual placement is called in a loop during the `run` method. Functions to be overwritten by derived classes are called before, during and after the loop, see the following code 4.1.

```
38 MAROCCO_TRACE("init:");
39 initialise(); // this shall be used as hook by derived classes
40
41 MAROCCO_TRACE("place_one_pop:");
42 while (place_one_population()) { // one hook is hidden in
43             // place_one_population after a successful placement.
44     MAROCCO_TRACE("loop:");
45     loop(); // this shall be used as hook by derived classes
46 };
47
48 MAROCCO_TRACE("finalise:");
49 finalise(); // this shall be used as hook by derived classes
```

Listing 4.1: Code sniplet of the run function of `PlacePopulationsBase` to show the hooks

`initialise` This hook is called in the beginning, so that derived classes can prepare the structure they require.

`loop` This hook is called repeatedly while `PopulationSlices` and `NeuronBlocks` are still in the queues. It might be called if no population was placed. The population might have been split, or a `NeuronBlock` was removed.

`finalise` Some derived classes manipulate the member variables in a way, that a termination will result in wrong assumptions by code following to the placement. Derived classes have to clean up what they have done, and must leave the Base class in an ordered way.

`update_relations_to_placement` This hook is called once a `NeuronPlacementRequest` has successfully been assigned to a `NeuronBlock`. Both the `NeuronPlacementRequest` and the `NeuronBlock` are passed as arguments to the call. It can be used by derived classes to have a simple way to handle following placements depending of the current placement.

4.3.2.2 bySmallerNeuronBlockAndPopulationID

Changeset: 4641

This class is the conversion of the placement strategy used until now to the new class based architecture. See section 3.1 for a more detailed description of the algorithm. It sorts the `NeurnonBlocks` by their available space first and then a spiral ordering starting from the centre of the wafer is applied among

`NeuronBlocks` with the same available space. `NeuronPlacementRequests` are handled in descending order of their ID.
It is a static placement method, that only hooks into the `initialise` function. It calls sort algorithms on both the `NeuronPlacementRequests` and the `NeuronBlocks` as described above.

4.3.2.3 byNeuronBlockEnumAndPopulationID

Changeset: 4641

This new placement method is simple and fast. Populations are sorted by their ID order and the `NeuronBlocks` by their Enum, a linear progressing number over the whole wafer. This results in a linear placement from `HICANN 0 NB 0` via `HICCANN 0 NB 7` and `HICANN 1 NB 0` to `HICANN 383 NB 7` starting with Population n and ending with population 0.
it hooks only into the `initialise` functions.

4.3.2.4 byNeuronBlockEnumAndPopulationIDasc

Changeset: 6297

By request of the users this class is provided to place the Populations in ascending order, starting from population 0 and ending with population n. The order of the `NeuronBlocks` is not altered compared to `byNeuronBlockEnumAndPopulationID` 4.3.2.3. This gives the users a simpler understanding of where their populations are placed.
As the performance (sec. 5.2.5) and simplicity of this method surpasses the old `bySmallerNeuronBlockAndPopulationID` 4.3.2.2 method, it was chosen to be the new default placement strategy.
The runtime is calculated by sorting both queues and placing neurons. $\mathcal{O}(\mathrm{NB}\, log(textNB) + \mathrm{Nrn}\, log(\mathrm{Nrn}) + \mathrm{Nrn})$

4.3.2.5 ClusterByPopulationConnectivity

Changeset: 4580

Inspired by Graph drawing algorithms of Fruchtman and Reingold [15] and Kamada and Kawai [20], which are also part of the Boost Graph Library, a Clustering of the populations was introduced to keep the connections short. The Model cannot be directly applied to the problem of placement of the populations, because the `BioGraph` is a directed graph, but the algorithms require undirected

graphs. Additionally sending and receiving are quite different in terms of the position. Different resources are used with different properties, sending is limited to 8 possible outputs, depending on the merger tree configuration, while during receiving a single synapse driver can access any neuron same HICANN. And populations spanning multiple `NeuronBlocks` were difficult to represent.
The idea to use one of these algorithms was held back, instead a simple clustering heuristic was invented which adds populations that are connected to an already placed population to a priority list. This priority list is sorted by the degree to the placed populations. The average position of the already placed communication partners is used to centre the ordering function applied to the `NeuronBlocks`. Figure 4.4 depicts the behaviour of the clustering.
The class provides a python interface which allows the setting of some parameters to influence the placement.

- the order on a HICANN can be selected.
- to spiral using full HICANNs or to spiral for each Neuron Block 0 on all HICANNs then `NeuronBlock` 1 on all HICANNs. and so forth.
- to use a spiral, or vertical ordering.
- to centre the spiral close to targets, sources or both types of populations.
- Weights to prioritise populations by their in- or out-degree to already placed populations.
- to move sending, targeted or both types of populations from the „place-later“-queue to the placement queue based on connections to placed populations.

`initalise` The `initialise` call moves all `NeuronPlacementRequest`s to a new queue which will contain all requests, that do not connect to an already placed population. This „unconnected“-queue is sorted by their degree and the last element is moved tho the placement queue. A first ordering of the `NeuronBlocks` is also conducted.

`loop` The `NeuronPlacemenRequests` are sorted by their degree to already placed populations. The centre of placed communication partners for the next `NeuronPlacementRequest` is calculated and the `NeuronBlock` queue sorted around this location.
The Ordering function passed to `std::sort` is obtained by `std::bind`ing to a virtual function that and can be overwritten by further derivations of the class.

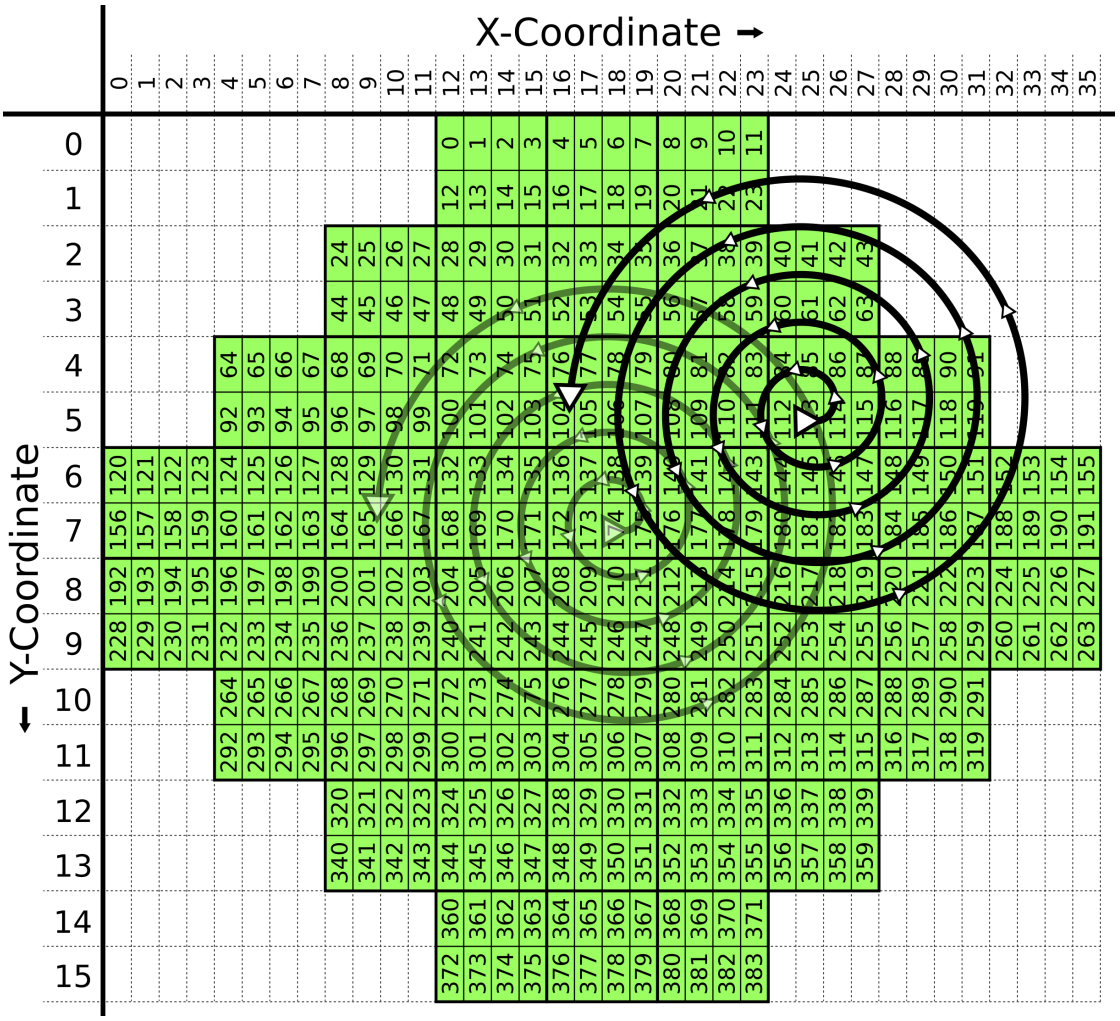


Figure 4.4: The `ClusterByPopulationConnectivity` strategy starts the spiral at the centre of the wafer. It does disregard the availability of the `NeuronBlocks` which will prevent scattering. To increase locality of the placements, future spirals are centred at the average location of connected and already placed populations.

`finalise` The list of unplaced Populations gets moved back to its original queue, such that later steps will notice that not all neurons have been placed.

`update_relations_to_placement` The book keeping which population has been placed on which `NeuronBlock` is handled in this function. Additionally populations connected to the just placed population are moved from the „unconnected"-queue to the placement queue.
further derivations from this class may also overwrite:

- `sort_population_priority`: sorts the `NeuronPlacementRequest`s by the degree to placed ones.
- `add_first_population_to_priority_list`: moves a single `NeuronPlacementRequest` from the „unconnected"-queue to the placement-queue.
- `update_population_priority_list`: it moves connected `NeuronPlacementRequest`s from the „unconnected"-queue to the placement-queue.
- `sort_neuron_blocks`: sorts the `NeuronBlocks` for the next placement request
- `nb_order_function`: defines an order for `NeuronBlockOnHICANN`s
- `hicann_order_function`: defines an order for `HICANN`s
- `neuron_block_comparator_function`: compares two `NeuronBlockOnWafer`
- `center_of_partners`: calculates the location of the partners for the next `NeuronPlacementRequest`.
- `degree_to_placed`: calculates the degree of a placement request to placed communication partners.

It is a dynamic placement strategy that, sorts the populations and `NeuronBlock`s before every placement. The runtime is dominated by the `update_relations_to_placement`:
$\mathcal{O}\left(\text{placements} \cdot \left((\text{pops}_{\text{targets+sources}}) \cdot \text{not placed} \cdot \text{not placed}\right)\right) \leq \mathcal{O}(\text{Nrn}^3 \cdot \text{pops})$
This is quite bad for larger numbers of populations. But usually there are less than 100 populations which in turn contain larger amounts of neurons.

4.3.2.6 ClusterByNeuronConnectivity

Changeset: 4728

This class is very similar to the `ClusterByPopulationConnectivity` 4.3.2.5 class, in fact it is derived from it, but this class operates on neuron level instead of population level.
During the `initialise` call all `PlacementRequests` are split into single neurons. Thus the sorting and placement is done on neuron level rather than for full Populations. This requires some more computations and requires to overwrite some methods to work on neuron level, but it has advantages where stochastic connectors are used, that do not connect all neurons between the populations. additionally the following functions were written

- `toBioNeuron`: converts `NeuronPlacementRequests` to `BioNeuron`
- `is_connected`: checks if two `BioNeurons` communicate with each other
- `getTargetNeurons`: returns a vector of target neurons for a `BioNeuron`
- `getSourceNeurons`: returns a vector of source neurons for a `BioNeuron`

the complexity of this class is equal to that of the population clustering, but on neuron level, thus $\mathcal{O}(\mathrm{Nrn}^4)$.

4.3.2.7 ConstrainedNeuronClusterer

Changeset: 5720 (WIP)

The `ConstrainedNeuronClusterer` is a further derivation of the `ClusterByNeuronConnectivity` 4.3.2.6. It is still work in progress.
The idea behind this placement strategy is to keep the synapse driver chain length within the constrained limits. To archive this the chain length is calculated after every single neuron placement, and checked against the `HICANNManager` who loads the calibration. This guarantees the validity after every placed neuron. As the calculation of driver chain lengths is the most expensive part in this placement strategy it is calculated only if required. This involves the chain length at targets of the new placed neuron and, and all other sources targeting the new neuron. If an overuse is detected, different behaviours are triggered.

- If the new neuron produces a violation of the chain lengths restriction on another HICANN it depends on the usage of the `NeuronBlock` the neuron was placed on.

  - If it is alone on that `NeuronBlock`, there are no other sources. This means that there are too many synapses emerging from this single neuron to the neurons on other chips. Enlargements of the targeted neurons are required. To archive this, the targeted neurons have to be relocated.
  - If other neurons are present on this `NeuronBlock`, the new neuron causes the violation. The new neuron has has to be relocated, it is put into a waiting queue. If the neuron was also targeting itself, then then it is also enlarged. If no other neurons could be placed on this `NeuronBlock` and the neuron queue is empty the `NeuronBlock` is removed from the placement opportunities and the waiting queue is put back to the neuron queue.

- If other neurons targeting the new placement produce a violation at the new neuron, the placed neuron might be to small or the sources are placed in an unfortunate way to allocate `L1Addresses` that require more drivers during decoding.
  - If the size of the neuron is smaller than a full `NeuronBlock` it is unplaced, enlarged and put back to the queue. This will result in a new try for the same neuron on the same `NeuronBlock`.
  - If the neuron can not be enlarged, the sources needs to be relocated. The neurons size is restored to original value.

In the current state of the changeset it is possible that the algorithm runs into loops that are not possible to solve. It requires tracking information to decide when to give up on neurons and and warn the user. To speed up the procedure of finding the correct size, the possibility to resize all neurons of the same population shall to be evaluated. It is promising as neurons of the same populations share the same properties.
in the current state it is only suitable for smaller networks, or to notify the user of expected synapse loss and advice them to use larger neurons.

4.4 Splitting external sources

Changeset: 4706

The external sources are placed similar to the clustered neurons, but are handled after the neurons have been placed. For some sources where the rate is known the sources already get split if the expected rate exceeds the rate of the FPGA or HICANN.

So far external sources were not checked if they will violate hardware constraints given by the required synapse drivers chain length. As the placement will have been performed at this time the size of the targeted neurons shall not be changed. The other possibility to reduce the synapse driver chain length is to use less addresses on the L1Bus. To archive this the external sources are split and thus the synapse driver chain length is reduced by using more primary synapse drivers at the cost of more Layer1 routes.
The driver requirement is calculated from the placement results datastructure. If a violation of the Driver requirement is recorded, the placement of the external source is undone. Then the placement request is split into two equal halves and the placement is tried again. This leads to a logarithmic backoff which is an acceptable runtime increase.

4.5 New Merger Tree strategy

Changeset: 5789

Currently there are two merger tree Strategies. the first which is a *one-to-one* mapping of the tree, and the second will configure the merger tree to bundle as many neurons as possible by considering only the `L1Addresses` thus it will merge up to 59 neurons together (some addresses are reserved).
In the latter case the synapse drivers chain length is often violated. As a precaution for synapse loss due to chain length restrictions a new merger tree strategy was written. It calculates the driver requirements during merging and splits mergers again, when the configuration would lead to synapse loss.
The expected firing rate of neurons should also be considered during merging but as it is not possible to guess the firing by the topology of the network the user would need to provide such information. So the recommended strategies are `one-to-one` mapping if high firing rates are expected and the new `minimise-as-possible` for low firing rates.

4.6 Hardware constraint calibration

Changeset: 4734 4903 4939 5077
Changeset: 4733 4857 4928 5253

To have hardware constraints available during placement and routing changes to `calibtic`, the calibration library interface used in the wafer software, have been made. The HICANN calibration now holds information on the values for

`L1CrossbarSwitches`, `SynapseSwitches` and `SynapseDriverChainLenth`. The values are influenced by the strength of the `L1Drivers` and the time scaling of the experiment. Slower executions would give the drivers more time to move the electrons and thus allow more switches.
During place and route the `HICANNManager` loads these information from the database. The values are cached to reduce IO- and computational-load.

5. Performance analysis

The main goal of this thesis is to improve the mapping performance archived by `marocco`.

5.1 Measurement procedure

While it is expected, that the state of the hardware improves, it is not expected to become a perfect wafer. Therefore I modelled an expected wafer to be real-world-like. The benchmark suite was run on the model of a perfect wafer and the model of a real-word-like wafer.
The model of a real-world wafer will pay attention to fabrication fluctuations and the missing highspeed connection on the centring 16 HICANNs [28]. To simulate fabrication fluctuations, 10% of the neurons are blacklisted on random. The actual number might be different, but scattered neuron blacklisting is expected as the users require specific characteristics of the neurons. The missing highspeed connections is simulated by by disabling neuron placement on the 16 centre HICANNs. These HICANNs are configurable only via JTAG, which is annoyingly slow and does not allow the transfer of spike data.
The default neuron size is set to 4, that means 4 hardware neurons (`denmems`) are connected together to resemble a single bio-neuron. In this case perfect wafer with 512 Neurons per HICANN and 384 HICANNs, can therefore hold a maximum of 49152 bio-neurons. On the real-world-like wafer model 16 HICANNs less are used and 10% of the neurons are blacklisted, so an approximate of $\frac{512\,\mathrm{HwNrn}}{4\,\mathrm{HwNrn/BioNrn}} \cdot (384 - 16)\mathrm{HICANNs} \cdot (1 - 0.1\,\mathrm{blacklisting}) \approx 42\,394\,\mathrm{BioNrn}$ can be assigned. The actual number of 33530 available bio-neurons is extracted from the logs, and is smaller than the calculated number, because the scattered blacklisting blacklists only single hardware-neurons, that prohibit the assembly of quad-neurons. Therefore not all whitelisted neurons are available to represent bio-neurons.
The networks of the benchmark suite (sec. 3.3), are scaled up in a parameter sweep over the network size. The following sizes were tested:

- fullyVisibleBm: 10; 100; 200; 400; 600; 800; 1000; 1500; 2000; 2500; 3000; 3500; 4000; 4500; 5000; 5500
- ising: 901; 2101; 4101; 6901; 10501; 14901; 20101; 26101
- pfeilsNoise: 10; 100; 200; 400; 600; 800; 1000; 1500; 2000; 2500; 3000; 3500; 4000; 4500; 5000; 5500; 6000
- random10: 10; 100; 200; 400; 600; 800; 1000; 1500; 2000; 2500; 3000; 3500; 4000; 4500; 5000; 5500; 6000
- random30: 10; 100; 200; 400; 600; 800; 1000; 1500; 2000; 2500; 3000; 3500; 4000; 4500; 5000; 5500; 6000
- net6: 119; 179; 255; 455; 719; 1047; 1235; 1439; 2019; 2699; 3479; 4359; 6419; 8879; 11739
- rbm: 20; 100; 200; 400; 600; 800; 1000; 1200; 1400; 1600; 1800; 2000; 2500; 3000; 4000; 5000; 6000

The results are checked for validity with respect to the hardware constraints (sec. 2.2).
The following tables show the largest network, that was successfully mapped with less than the stated number of synapse loss. The left half of the tables show the largest networks restricted by the overall synapse loss, while the right half shows network sizes selected by loss due to failed Layer1 routes. A Layer1 route is considered failed if no `VLine` was allocated next to the targeted synapse array. Additional sources of loss, like unavailable synapse drivers, restrictions to the synapse-driver-chain-length and errors in the synapse array routing are counted into the total loss.

5.2 Algorithms

5.2.1 Bug fixes

Tables 5.1 and 5.3 show the results of a software stack containing only a minimum of my changes. For compatibility with new software dependencies a few of my changes had to be included. Changes of other contributors were all included.
The bug fixes provided by me in changesets 4251, 4778, 5837, 5842, 5858, 6601 were included in the software stack used to generate tables 5.2 and 5.4.
Having a look on the perfect wafer without fixes 5.1 and with fixes 5.2, we see some improvements provided by the bug fixes. Especially if higher synapse losses

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	2500	2500	2500
ising2d	4101	4101	4101	4101	4101	4101
pfeilsNoise	2500	2500	2500	2500	2500	2500
random10	1000	1500	2500	2500	2500	2500
random30	200	1000	1500	2500	2500	2500
net6	0	0	0	4359	4359	4359
rbm	0	0	0	3000	3000	3000

Table 5.1: On a perfect wafer using a software stack **without** most of my changes, for compatibility some were required. The number is the amount of neurons that can be routed with $[0, 5, 10]\%$ total loss on the left and the network sizes restricted only by Layer1 routing on the right.
†: no larger network was tested

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	3000	3500	4500
ising2d	26101†	26101†	26101†	26101†	26101†	26101†
pfeilsNoise	3000	3500	4000	3000	3500	4500
random10	1000	1500	3500	3000	3500	4500
random30	200	1000	1500	3000	3500	4500
net6	0	0	0	4359	6419	11739†
rbm	0	0	0	3000	3000	4000

Table 5.2: On a perfect wafer **with bugfixes** (6601 5858 5842 5837 4251 4778) applied. Colours compare to 5.1 **Further changes will compare to this table.**
†: no larger network was tested

are tolerated larger networks are possible. The `ising` network profits a lot, and can be scaled to the maximum step size tested in the benchmarks. This slight improvements for networks without loss explains why most of the bugs were hidden so far.
Looking at the real-world-like wafer for a run without bug fixes, as shown in table 5.3, one sees that only small networks of around 200 neurons are possible. Most of the time hardware constraints were violated at the end of the mapping, that prevent larger networks to be accepted. The bug fixes do prevent invalid configurations an allows the setup of networks 4 times the size. The results can be found in table 5.4.

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	10	10	10	200	200	200
ising2d	0	0	0	0	0	0
pfeilsNoise	200	200	200	200	200	200
random10	200	200	200	200	200	200
random30	200	200	200	200	200	200
net6	0	0	119	455	455	455
rbm	0	0	0	200	200	200

Table 5.3: On a a real-world-like wafer with 10% blacklisting using a software stack **without** most of my changes, for compatibility some were required. †: no larger network was tested

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	10	10	10	800	1500	2000
ising2d	2101	6901	10501	2101	6901	10501
pfeilsNoise	800	1500	2000	800	1500	2000
random10	800	1500	2000	800	1500	2000
random30	600	1000	1500	800	1500	2000
net6	0	0	119	1235	4359	6419
rbm	0	0	0	800	1600	2000

Table 5.4: On a real-world-like wafer with 10% blacklisting **with bugfixes** (6601 5858 5842 5837 4251 4778) applied.
Colours compare to table 5.3
Further changes will compare to this table.
†: no larger network was tested

5.2.2 4706: splitting external sources

No changes in the performance can be measured, since external sources are not used in the benchmark suite.

5.2.3 4189: feed left and right

This change also allows the usage of the neighbouring HICANNs for the signal injection into the synapse array, this doubles the amount of `VLines`, but the number of synapse drivers stays the same. To solve the competition between the 32 `VLines` for the 14 synapse drivers the `VLineUsage` is evaluated to prevent overallocation of the drivers and allow the backbone router to try to feed the signal from the neighbouring HICANN. This will, in case of overallocation, cause the routing to stop early. As a consequence the synapse loss by insufficient synapse drivers is shifted from the category of total to the category of Layer1-loss.

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	1500	1500	1500
ising2d	26101†	26101†	26101†	26101†	26101†	26101†
pfeilsNoise	1500	1500	1500	1500	1500	1500
random10	1000	1500	1500	1500	1500	1500
random30	200	1000	1500	1500	1500	1500
net6	0	0	0	4359	8879	11739†
rbm	0	0	0	1600	3000	3000

Table 5.5: On a perfect wafer with 4189. For this Changeset only the total loss should be consulted for comparisons, as Synapse loss by unavailable Synapse Drivers is shifted from total loss to L1-loss. Colours compare to 5.2
†: no larger network was tested

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	10	10	10	200	600	800
ising2d	0	6901	10501	0	6901	10501
pfeilsNoise	200	1000	1000	200	1000	1000
random10	200	800	800	200	800	800
random30	200	800	800	200	800	800
net6	0	0	119	2019	6419	6419
rbm	0	0	0	400	1200	1400

Table 5.6: On a real-world-like wafer with 10% blacklisting with 4189. For this Changeset only the total loss should be consulted for comparisons, as Synapse loss by unavailable Synapse Drivers is shifted from total loss to L1-loss. Colours compare to table 5.4
†: no larger network was tested

Previously the `VLines` were marked as reached even if it was known, that there will not be enough synapse drivers to inject the signal into the synapse array. So for reasons of comparison the total loss should be observed in this case. Table 5.5 and 5.6 show the results of Change-Set 4189.
The results for the total loss on the real-world-like wafer are disappointing. but can be explained by the irregular placement pattern. This leads to `VLines` being extended longer than before, to feed a neighbouring HICANN. This uses more synapse drivers on that specific side of the HICANN. Which then reduces the score for following routes, until no score is archived. The scoring function is only based on the `VLineUsage`, but not the amount of neurons that can be reached. So for some sources the free side is not attractive enough because the score on the blocked side is kept high by targeted HICANNs requiring the other side. They usually have only a few neurons, that are targeted by just a few sources.

Still this change allowed the deeper evaluation of the routing methods and lead the finding of most of the fixed bugs.

5.2.4 5789: synapse driver chain length aware merger tree

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	3000	3500	4500
ising2d	26101†	26101†	26101†	26101†	26101†	26101†
pfeilsNoise	2500	6000†	6000†	5500	6000†	6000†
random10	1000	1500	3500	3000	3500	4500
random30	200	1000	1500	3000	3500	4500
net6	0	0	0	4359	6419	11739†
rbm	0	0	0	3000	3000	4000

Table 5.7: On a perfect wafer with 5789.
Colours compare to 5.2
†: no larger network was tested

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	10	10	10	800	1500	2000
ising2d	4101	26101†	26101†	4101	26101†	26101†
pfeilsNoise	1000	2000	3500	1000	2000	3500
random10	1000	1500	2000	1000	2000	3500
random30	600	1000	1000	1000	1500	2000
net6	0	0	119	1235	4359	6419
rbm	0	0	0	800	1600	2000

Table 5.8: On a real-world-like wafer with 10% blacklisting with 5789.
Colours compare to table 5.4
†: no larger network was tested

The results of changeset 5789 are shown in table 5.7 and 5.8.
On a perfect wafer only the results of `pfeil's noise` network have changed. The performance of the Layer1 routing is improved from 3500 neurons to 5500 neurons without loss on the Layer1 network. However the total performance is decreased by one step in the parameter scan and starts at 2500 neurons instead of 3000, but with a 5% loss up to 6000 neurons can be mapped, which is the maximum that was tested during benchmark runs. The loss could be explained by bundled signals requiring a driver chain length, that steals synapse drivers from neighbouring periods. The `VLineUsage` does not track synapse driver chains.

On the model wafer general improvements on the `ising`, `pfeil's noise` and `random` network can be observed. The `random10` networks was increased from 800 to 1000 neurons without any losses. Special note should also be on the `ising` network, that was mapped twice the size without any losses, and could be mapped with losses less than 5% to the maximum step of the benchmarks with 26101 neurons.

5.2.5 4641: class interface for placement

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	4000*	4000	4500
ising2d	14901	26101†	26101†	14901	26101†	26101†
pfeilsNoise	3000	3500	4000	4000	4000	4500
random10	800	1500	3500	4000	4000	4500
random30	800	1000	1500	4000	4000	4500
net6	0	0	0	8879	11739†	11739†
rbm	0	0	0	4000	4000	4000

Table 5.9: On a perfect wafer with 4641 and the new simple byNeuronBlockEnumAndPopulationIDasc selected 4.3.2.4. Colours compare to 5.2
Other Placement strategies are compared to this results, as it was decided to be the new default strategy
∗: the technical maximum is 4096 Synapses for a fully connected network
†: no larger network was tested

The old placement strategy converted to a class and provided in 4641 does not alter the results therefore the tables are not shown. This is expected behaviour, as only the interface changed but the strategy stayed the same.
The simple placement strategy `byNeuronBlockEnumAndPopulationIDasc` 4.3.2.4 archives results for the total loss that are comparable between the perfect wafer 5.9 and the real-world-like wafer 5.10. The network size restricted by loss caused during Layer1 is nearly twice the size on the perfect wafer. Actually it reaches the step size with the largest network of the `fullyVisibleBM` that is expected to be possible to route on a perfect wafer without loss on the Layer1 network. With a default neuron size of 4, a `NeuronBlock` contains 16 bio-neurons. As there are 256 `VLines` on a HICANN that each carry the signal of a full `NeuronBlock` 256 VLines · 16 BioNeurons/VLine = 4096 BioNeurons can be routed via the L1 network to a single HICANN. The results on the modelled wafer suffer from a lower neuron occupation on the Layer1 buses.
`byNeuronBlockEnumAndPopulationIDasc` was chosen to be the new default

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	2500	3000	3000
ising2d	14901	14901	14901	14901	14901	14901
pfeilsNoise	2000	2500	3000	2500	3000	3000
random10	1000	2500	2500	2500	3000	3000
random30	800	1000	1500	2500	3000	3000
net6	0	0	0	4359	6419	8879
rbm	0	0	0	2500	3000	3000

Table 5.10: On a real-world-like wafer with 10% blacklisted neurons with 4641 and the new simple `byNeuronBlockEnumAndPopulationIDasc` selected 4.3.2.4. Colours compare to table 5.4
Other Placement strategies are compared to this results, as it was decided to be the new default strategy
†: no larger network was tested

placement strategy also because it is easier to predict by the user. Subsequent placement strategies are compared to this algorithm.

5.2.6 4580: cluster by population connectivity

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	4000*	4000	4500
ising2d	2101	26101†	26101†	2101	26101†	26101†
pfeilsNoise	3000	3500	4000	4000	4000	4500
random10	1000	1500	3500	4000	4000	4500
random30	800	1000	1500	4000	4000	4500
net6	0	0	0	3479	6419	11739†
rbm	0	0	0	4000	4000	4000

Table 5.11: On a perfect wafer with 4580 and vertical+neighbour+11+targetsANDsources
Colours compare to 5.9
†: no larger network was tested
∗: the technical maximum is 4096 Synapses for a fully connected network

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	2500	2500	2500
ising2d	4101	6901	20101	4101	6901	20101
pfeilsNoise	2000	2500	2500	2500	2500	2500
random10	1000	2500	2500	2500	2500	2500
random30	800	1000	1500	2500	2500	2500
net6	0	0	0	2019	8879	8879
rbm	20	20	600	2500	2500	2500

Table 5.12: On a real-world-like wafer with 10% blacklisted neurons with 4580 and vertical+neighbour+11+targetsANDsources
Colours compare to table 5.10
†: no larger network was tested

Different settings for the cluster strategy has been tested. The default `spiral-neighbour-1T-1S-targetANDsource` has been altered for each parameter once.

- The ordering function for sort `NeuronBlock`s:
  - **spiral** ordering
  - **vertical** ordering
- The centre for the order function was set to:
  - consider **all** placed communicating populations

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	3000	3500	4500
ising2d	14901	26101†	26101†	14901	26101†	26101†
pfeilsNoise	2500	3500	4000	3000	3500	4500
random10	1000	1500	3500	3000	3500	4500
random30	200	1000	1500	3000	3500	4500
net6	0	0	0	2019	11739†	11739†
rbm	0	0	0	3000	3000	4000

Table 5.13: On a perfect wafer with 4580 and spiral+target+11+targetsANDsources
Colours compare to 5.9
†: no larger network was tested

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	2500	3000	3000
ising2d	10501	26101†	26101†	10501	26101†	26101†
pfeilsNoise	2000	2500	3000	2500	3000	3000
random10	1000	2500	2500	2500	3000	3000
random30	600	1000	2000	2500	3000	3000
net6	0	0	0	2019	11739†	11739†
rbm	20	20	20	2500	3000	3000

Table 5.14: On a real-world-like wafer with 10% blacklisted neurons with 4580 and spiral+target+11+targetsANDsources
Colours compare to table 5.10
†: no larger network was tested

  - consider only placed **targeted** populations
  - consider only placed **sourcing** populations

- The Sorting Priority for the `PlacementRequest`s:
  - 1T-1S: **equal** weigths to sources and targets
  - 0T-1S: only calculate the degree by counting **sources**.
  - 1T-0S: only calculate the degree by counting **targets**.

- A Setting to decide which populations are added to the priority queue for the next placement:
  - add **targets and sources** of just placed Populations
  - only add **sources** of just placed Populations

– only add **targets** of just placed Populations

Similar to the `byNeuronBlockEnumAndPopulationIDasc` 4.3.2.4 the vertical ordering of connected populations is also able route the maximum step size of the `fullyVisibleBM` without loss on the Layer1 network, refer to table 5.11. On the expected Wafer 5.12 the performance is not reproducible, mainly because of the blacklisted centre HICANNs, where long `VLines` could be used. The disabling of the 16 centre HICANNs also interrupts the driver periodicity because 4 y-coordinates in the centre are missing, which leads to unevenly distributed periods for the required Synapse Drivers.
The best results for the modelled wafer were acquired by a `spiral` ordering with centring at the `targets` and given sources and targets the **same priority** and **adding both** to the priority queue. The results for the modelled wafer are in 5.14 and for the perfect wafer in 5.13.
As the Clustering analysis is done on population level only networks with multiple populations are able to profit by this placement strategy. This includes the `ising`, `rbm` and the `net6` network. For the first time on the expected wafer the `rbm` is now able to be routed with 20 neurons without any loss. The `ising` and `net6` networks remain to be affected by some losses, but they can be scaled to larger sizes within 5% loss. The `net6` suffers from a high fan-in at some neurons, that require driver chain length, that would violate the hardware constraints, and thus the total loss remains high.

5.2.7 4728: cluster by neuron connectivity

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	3000	3500	4000†
ising2d	6901	26101†	26101†	6901	26101†	26101†
pfeilsNoise	3000	3500	4000	3500	4000	4500
random10	1000	1500	3500	3000	4000	4500
random30	100	1000	1500	3000	3500	4500
net6	0	0	0	6419	11739†	11739†
rbm	0	0	0	3000	3000	4000

Table 5.15: On a perfect wafer 4728 and spiral-neighbours-10-targetANDsource
Colours compare to table 5.9
†: no larger network was tested

With clustering by neuron connectivity the results are generally comparable to the clustering by population connectivity.

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	2500	3000	3000
ising2d	4101	26101†	26101†	4101	26101†	26101†
pfeilsNoise	2000	2500	3000	2500	3000	3000
random10	1000	2500	2500	2500	3000	3000
random30	600	1000	2000	2500	3000	3000
net6	0	0	0	4359	11739†	11739†
rbm	20	600	800	2500	3000	3000

Table 5.16: On a real-world-like wafer with 10% blacklisted neurons with 4728 and spiral-neighbours-10-targetANDsource
Colours compare to table 5.10
†: no larger network was tested

Results of the run on the perfect wafer are shown in table 5.15. The results of the simple placement method are not regained, but `pfeils` and `net6`, perform better than on the population based clustering algorithm, however the `ising` network performs worse.
On the real-world-like wafer the behaviour is are comparable, the `ising` and `net6` suffer from some early loss in the Layer1 network but can be scale to the maximum step size with less than 5% loss. With different parameters set through the `python` interface 10501 neurons could be mapped in the `ising` network without loss on the real-world-like wafer.
Notable is the high connectivity `rbm` network which can be mapped with less than 5% loss with up to 600 neurons and up to 800 neurons with less than 10% loss.

5.2.8 5720: synapse driver chain length aware placemnt

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	200	600	800†	600	800†	800†
ising2d	2101	10501	26101†	2101	10501	26101†
pfeilsNoise	3000	3500	4000	4000	4000	4000
random10	1000	1500	3500	4000	4000	4500
random30	800	2000	2000	2000	2500	2500
net6	719	2019†	2019†	2019†	2019†	2019†
rbm	200	800	800	800	1000†	1000†

Table 5.17: On a perfect wafer with 5720
Colours compare to 5.9
†: no larger network was tested

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	200	600	800†	800†	800†	800†
ising2d	2101	6901	14901†	2101	6901	14901†
pfeilsNoise	2000	2500	2500	2000	2500	3000
random10	1000	2500	2500	2500	2500	2500
random30	800	1000	2000	2500	2500	2500
net6	179	455†	455†	255	455†	455†
rbm	200	400†	400†	400†	400†	400†

Table 5.18: On a real-world-like wafer with 10% blacklisted neurons with 5720 (chain length aware placement)
Colours compare to table 5.10
†: no larger network was tested

Changeset 5720 introduces a dynamic placement strategy derived from the neuron clustering algorithm provided in 4728. If the synapse driver chain length of 3 is not sufficient the affected neurons are either enlarged or relocated. In tables 5.17 and 5.18 notable changes are the improvements on the high degree networks (`fullyVisibleBm`, `net6`, `rbm`).
The enlarged neurons will occupy more space and reduce the number of neurons per `NeuronBlock`, this reduces the Layer1 occupancy, which causes more buses to be used and in turn reduce the in Layer1 performance, but it allows to reduce the synapse driver chain length, which is required for the high degree networks.
The time required by the dynamic placement has no upper boundary, because the same neuron might be relocated again and again, which causes larger networks to timeout. But it demonstrates, that the high degree networks are now routable with network sizes of around 200 neurons, which is at least an increase of 10 times the size for the `rbm`.

5.3 Bundled result analysis

Here a patch stack containing the improvements, and 5789 and 4728 is analysed. The `ClusterByNeuronConnectivity` is set to prioritise only the targeted neurons. The results on a perfect wafer can be found in table 5.19 and the model wafer is shown in table 5.20. `ising`, `pfeil's` and `net6` were run with more steps to test the software towards full wafer usability.
Impressive is the result for the `ising` which is very archives a size of 32901 bio-neurons, 98% of the placeable bio-neurons 33530 are placed and successfully routed.

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	3000	3500	4000†
ising2d	44601	48901†	48901†	44601	48901†	48901†
pfeilsNoise	3000	7000	7500	5500	7000	8500
random10	1000	1500	3500	3000	4000	4500
random30	100	1000	1500	3000	3500	4500
net6	0	0	0	6419	27179	32039†
rbm	0	0	0	3000	3000	4000†

Table 5.19: On a perfect wafer with 5789 and 4728 and spiral-neighbours-10-targetANDsource
Colours compare to table 5.9
†: no larger network was tested

network	no loss	<5%	<10%	no L1-loss	<5%	<10%
fullyVisibleBm	0	0	0	2500	3000	3000
ising2d	32901	33262†	33262†	32901	33262†	33262†
pfeilsNoise	3000	6000	7500	5000	7000	8500
random10	1000	2000	3000	3500	4500	4500
random30	600	1000	2000	2500	3000	3000
net6	0	0	0	4359	14999	14999
rbm	20	600	800	2500	3000	3000

Table 5.20: On a real-world-like wafer with 10% blacklisted neurons with 5789 and 4728 and spiral-neighbours-10-targetANDsource
Colours compare to table 5.10
†: no larger network was tested

6. Summary

network	buggy	fixed	improved	buggy	fixed	improved
fullyVisibleBm	0	0	200^{*}	10	10	200^{*}
ising2d	4101	$26101^{\dagger}$	$44601^{\ddagger}$	0	2101	$32901^{\ddagger}$
pfeilsNoise	2500	3000	3000	200	800	$3000^{\ddagger}$
random10	1000	1000	1000	200	800	1000^{+}
random30	200	200	800^{+}	200	600	800^{+}
net6	0	0	719^{*}	0	0	179^{*}
rbm	0	0	200^{*}	0	0	200^{*}

Table 6.1: The maximum network that was mapped without any loss on a perfect wafer left, and on a real-world-like wafer right.
Without most of my changes: buggy (sec. 5.2.1), with bug fixes: fixed (sec. 5.2.1), and with improvements. The superscript indicates which changeset was the „simplest“ to produced the result
†: no larger network was tested ‡: bundled stack 5.3 $*$: constraint aware placer 5.2.8 $+$: simple placer 5.2.5

The place and route software `marocco` has been fixed and improved. It is now more tolerable towards real world hardware with blacklisting. A real-world-like wafer was modelled which includes 10% random blacklisting and disabled placement on 16 centring HICANNs on which the highspeed interface is unavailable. The performance analysis was carried out on both the perfect wafer and the real-world-like wafer. Noticeable improvements are archived on the both wafer models, see table 6.1.
The `ising` model can be scaled to span over nearly the full wafer on both models (90% on the perfect wafer and 98% of the available hw-quad-neurons (or 66% of the hw-neurons including blacklisted) on the real-world-like wafer). On the real-world-like wafer most networks can be scaled up lossless to the size that is also possible as on a perfect wafer. The higher the connectivity in the network, the smaller is the maximum network size that can be mapped without loss. `Pfeil's noise` network can be scaled to 3000 neurons, a `random` network with 10% connectivity to 1000 neurons, a `random` network with 30% connectivity only

to 800 neurons. Networks with higher connectivity can profit from the still under development changeset 5720. It resizes and relocates neurons on demand, which solves issues of the synapse driver chain length requirements. A `fullyVisibleBm` and an `rbm` can be scaled to 200 neurons without any synapse loss, and the benchmark network `net6` can be scaled to 179 neurons on the real-world-like wafer and to 719 neurons an a perfect wafer.
The fact that the networks start to lose synapses by Layer1 later than by the total loss, indicates, that there are to few synapse drivers available.
My fixes and improvements are especially useful on the real-world-like wafer, where the network size could be increased depending on the network topology by a factor greater than four and up to fifteen.
On the way to archive this results the calibration database is extended to hold information on hardware restrictions, which are loaded during mapping to generate valid results. This is also a step towards hardware agnostic routing which will be useful in translating the library to the next generation of hardware.
A new `MergerTree` algorithm was written to minimise the required Layer1-buses while still maintaining the synapse driver chain length constraints given for the hardware. The increased Layer1 occupation reduces the bus allocation which prevents congestion and keeps more buses available for other sources. In cases where only a few neurons per `NeuronBlock` require synapses the bundling reduces the number required synapse drivers at the target.
Changes on the design of the neuron placement architecture allow the user to choose either from a predefined placement strategy or define their own strategy. The old strategy was converted and remains available to the users. New linear placement strategies and strategies that cluster by connectivity of populations and neurons were written and are available to the user.

7. Outlook on Further improvement possibilities

7.1 Synapse Driver allocation

After applying my changes the main source contributing to the total synapse loss is the competition of 16+16 `VLines` for 14 Synapse Drivers. The synapse driver period at the target is determined by the `DNCMerger` the signal emerges from. To change the `DNCMerger` of a signal the placement algorithms has to calculate the driver period and chain length requirements at in advance, and use a different `NeuronBlock` or `DNCMerger` mapping in case of Synapse Driver collisions. To archive this in the placement algorithms a function for period precalculation has to be written, and the `MergerTree`s-routing will be required during placement.
Synapse drivers chaining influences the neighbouring periods, so also a preallocation class has to be written and used during placement.

7.2 L1 crossbars and switches

If the L1-drivers were better characterised it would be possible to calculate the electric load on them and decide which switches would be allowed to use. For example it might be allowed to use 2 Switches into the Synapse Arrays if no Crossbar would be used. Further use cases can be imagined. Aside from better knowledge of the L1-drivers this would require a class used during L1-routing, which would keep track of the electric load and switches. It would also require changes to the current implementation of the `L1Crossbar` restrictioning (4251).

7.3 Synapse Driver chain length

A neuron build by the top and bottom neuron can have 2 synapses per hemisphere. Therefore to increase the number of synapse more bits have to be decoded by the

Synapse Drivers, which are a rare resource, as explained earlier. It is possible to chain up to three Synapse Drivers, which will threefold the number of synapses. To use more Synapse Drivers the signal would need to be routed multiple times around the targeted Synapse Array, which allows the use of multiple `VLines`, and thus more primary Synapse Drivers, but on the same period (except if there is a change by one to the neighbouring HICANN, which will produce problems with the chaining).
Extending the Neuron horizontally is a c̈heaperälternative to multiple target definitions during routing. A first implementation of a resizing placement algorithm has been provided in 5720, but the current implementation is still slow. It still requires better notekeeping during backtracking, and could profit by assuming the same driver requirements for all neurons of the same population.

7.4 L1 congestions

Congestion of the Layer1 network have been reduced by providing better placement methods (4641 4580 4728), and bundling more signals (5789). But the backbone can not extend past the edges of the wafer, this requires the use of a detour. If there are also targets in the vertical column where the detour is started, the current implementation cannot fulfil both needs, the need for a large extension of the detour and the need to be on the correct side for the Synapse Drivers.
Simple methods would place the detour earlier in cases where the driver and extension requirements are not met. A more advanced option would be to adopt methods of the Refined-Single-Trunk-Steiner-Tree algorithm described in [10] which has a time complexity of $\mathcal{O}(n \log(n))$. Another extension is a dynamic routing with backtracking and dynamic costs. would allow all routes to find their ideal way, and then set

7.5 Parallelism

The Synapse routing can be parallelised in a simple way by using HICANN granularity. As it currently makes up a notable part of the mapping process it would be worth a look.

7.6 Modularity

Currently it is only possible to define one strategy for each step, that is used during the whole mapping process. It would be nice to be able to define the placement

strategy for each population on its own. A network might consist of different parts, that can profit from different strategies.

7.7 Hardware agnostics

Future chips (BrainScaleS-2) also require mapping software, but will have different topologies on Layer1 network, MergerTree network an neuron configurations. The mapping software shall be able to load these values from a database and find a solotion to the mapping problem.

7.8 FPGA like routing

In a discussion with C. Pehle [25], he mentioned the pathfinding algorithm, which at first tries to route all signals without removing the resources from the graph, to get an estimation of hot areas, and assign costs to these resources, that are in subsequent calls more expensive to take. After multiple iterations the routing shall settle on a solution.

7.9 Multi wafer routing

The current implementation of `marocco` can only act on a single wafer, but the hardware is designed to provide a wafer crossing Layer2 network, which would allow the use of 20 wafers for a single experiment.

Bibliography

[1] Dave Ackley, GE Hinton, and Terrence Sejnowski. "A Learning algorithm for Boltzmann machines". In: Cognitive Science: A Multidisciplinary Journal 9 (Jan. 1985), pp. 147–169. DOI: 10.1207/s15516709cog0901_7.

[2] Sebastian Jeltsch et al. . marocco: Mapping and Routing Software for the BrainScaleS-1 System. June 6, 2019. URL: https://github.com/electronicvisions/marocco.

[3] Frederico Azevedo et al. "Equal Numbers of Neuronal and Nonneuronal Cells Make the Human Brain an Isometrically Scaled-Up Primate Brain". In: The Journal of comparative neurology 513 (Apr. 2009), pp. 532–41. DOI: 10.1002/cne.21974.

[4] benchmarks for the BrainScaleS system. June 6, 2019. URL: https://github.com/electronicvisions/brainscales-benchmarks.

[5] Boost Graph Library, adjacency list. May 21, 2019. URL: https://www.boost.org/doc/libs/1_69_0/libs/graph/doc/adjacency_list.html.

[6] Boost Graph Library, Dijkstra. May 21, 2019. URL: https://www.boost.org/doc/libs/1_69_0/libs/graph/doc/dijkstra_shortest_paths.html.

[7] Romain Brette and Wulfram Gerstner. "Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity". In: J Neurophysiol 94: 3637–3642, 2005. (2005). DOI: doi : 10.1152/jn.00686.2005..

[8] Nicolas Brunel. "Dynamics of Sparsely Connected Networks of Excitatory and Inhibitory Spiking Neurons". In: Journal of Computational Neuroscience 8.3 (May 2000), pp. 183–208. ISSN: 1573-6873. DOI: 10.1023/A:1008925309027. URL: https://doi.org/10.1023/A:1008925309027.

[9] C++ Reference. June 3, 2019. URL: https://cppreference.com/.

[10] Hongyu Chen et al. "Refined Single Trunk Tree: A Rectilinear Steiner Tree Generator For Interconnect Prediction". In: Conference: SLIP'02, April 6-7, 2002, San Diego, California, USA. (2002). DOI: 10.1145/505348.505366.

[11] Ann-Shyn Chiang et al. "Three-Dimensional Reconstruction of Brain-wide Wiring Networks in Drosophila at Single-Cell Resolution". In: Current Biology 21, 1–11 (January 11, 2011). DOI: 10.1016/j.cub.2010.11.056.

[12] AP Davison et al. "PyNN: a common interface for neuronal network simulators". In: (2009). DOI: doi:10.3389/neuro.11.011.2008.

[13] E. Dijkstra. "A note on two problems in connexion with graphs." In: Numerische Mathematik, 1:269-271 (1959).

[14] Johannes Fieres, Johannes Schemmel, and Karlheinz Meier. "Realizing Biological Spiking Network Models in a Configurable Wafer-Scale Hardware System". In: Proceedings IJCNN2008, IEEE Press (2008).

[15] T. Fruchterman and E. Reingold. "Graph drawing by force-directed placement." In: Software–Practice & Experience, 21 (11), pp. 1129-1164 (1991).

[16] Michael R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman, 1979. ISBN: ISBN 0-7167-1045-5.

[17] Geoffrey E. Hinton. "Training Products of Experts by Minimizing Contrastive Divergence". In: Neural Computation 14.8 (2002), pp. 1771–1800. DOI: 10.1162/089976602760128018. eprint: https://doi.org/10.1162/089976602760128018. URL: https://doi.org/10.1162/089976602760128018.

[18] E. Ising. "Beitrag zur Theorie des Ferromagnetismus". In: Zeitschrift fur Physik 31 (Feb. 1925), pp. 253–258. DOI: 10.1007/BF02980577.

[19] Sebastian Jeltsch. "A Scalable Workflow for a Configurable Neuromorphic Platform". PhD thesis. Ruperto Carola University of Heidelberg, 2014.

[20] T. Kamada and S. Kawai. An algorithm for drawing general undirected graphs. 1989.

[21] Mitja Kleider. "Neuron Circuit Characterization in a Neuromorphic System". PhD thesis. Universität Heidelberg, 2017.

[22] Christoph Koke. "Device Variability in Synapses of Neuromorphic Circuits". PhD thesis. Ruperto Carola University of Heidelberg, 2017.

[23] Bastian Mathieu, Heymann Sebastien, and Jacomy Mathieu. "Gephi : An Open Source Software for Exploring and Manipulating Networks". In: Association for the Advancement of Artificial Intelligence (2009).

[24] Eric Christian Müller. "Novel Operation Modes of Accelerated Neuromorphic Hardware". PhD thesis. Ruperto Carola University of Heidelberg, 2014.

[25] Christian Pehle. private communication.

[26] Thomas Pfeil et al. "Effect of Heterogeneity on Decorrelation Mechanisms in Spiking Neural Networks: A Neuromorphic-Hardware Study". In: Phys. Rev. X 6 (May 2016), p. 021023. DOI: 10.1103/PhysRevX.6.021023. URL: http://link.aps.org/doi/10.1103/PhysRevX.6.021023.

[27] Human Brain Projct. "HUMAN BRAIN PROJECT FRAMEWORK PARTNERSHIP AGREEMENT Annex 1 Part B". In: (2015).

[28] Sebastion Schmitt. private communication.

Erklärung:

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

Heidelberg, den (Datum) .....