Performance Increase of the Visualization Software for the Neuromorphic Hardware System BrainScaleS

Jonas Weidner

University of Heidelberg, supervised by Eric Müller

July 8, 2019

This internship report describes different improvements of the browser based visualization software created and used by the Electronic Vision(s) group at the university of Heidelberg.

After the design of an experiment for the BrainSacelS system, the layout of all neurons and synapses gets mapped to the wafer. The visualization software draws the mapping results for the user to debug and analyze. The current problem is that the visualization is too slow and the memory consumption is too large.

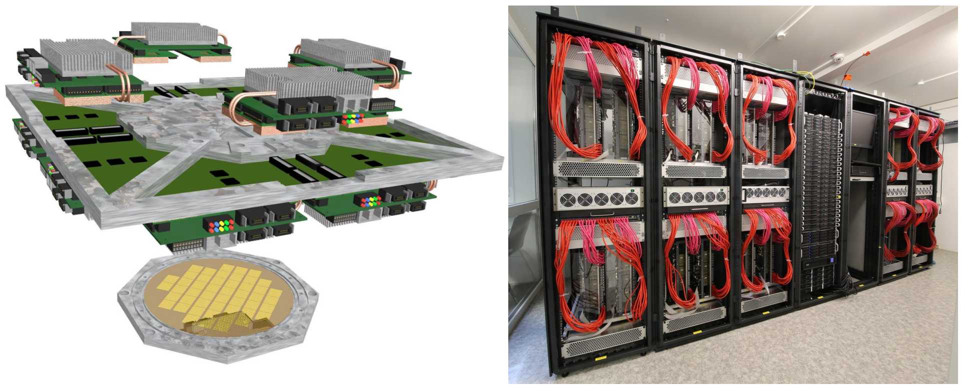
Improvements reduced memory footprint by about 30%. This was mainly achieved by drawing synapses as textures.

Contents

1 Introduction

1.1 BrainScaleS System

The BrainScaleS system is a neuromorphic hardware system developed by the Electronic Vision(s) group at the University of Heidelberg [1]. The system is shown in Figure 1b. It contains 20 silicon wafers shown in Figure 1a. Each wafer contains 384 connected HICANN[1] chips. A HICANN chip consists of 512 neurons and 220 synapses per neuron. This adds up to four million neurons and one billion synaptic connections for the whole system.



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

Figure 1: BrainScaleS - System

The neurons and synapses are realized with analog electronic circuits based on the AdEx - LIF[2] neuron model [6]. The connection between them is digital. Compared to biological timescales a similar emulated neural network is approximately 10,000 times faster on the BrainScaleS system.

1.2 Pixi.JS

Pixi.JS is an open source 2D WebGl graphics library. It is used for the 2D wafer mapping 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 graphics object and draw them as a so called sprite objects. A comparison was done by Richard Boell. [2]

Usually the graphic objects are very fast for a large number of abstract shaped graphics, whereas sprites are good for images. Our idea was to generate a texture out of many graphic objects. Hence, the graphic card has to handle fewer objects. The disadvantage is that the objects are not individually adjustable any more.

[1]High Input Count Analog Neural Network Chip

[2]Adaptive Exponential Leaky Integrate-and-Fire Neuron Model. A detailed explanation is given in "Neuron Circuit Characterization in a Neuromorphic System" [6] by Mitja Kleider in section 2.5

1.3 Building Steps

The building steps are shown in Figure 2. The main part of the web visualization is written in TypeScript [5] and gets transcompiled into JavaScript.

To reduce the code that has to be monitored, the marocco::results class that writes and reads the results file is reused. This is done by Emscripten [4] which automatically 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. It turned out that maintenance-intensive programs need a lot of human resources and will be taken out of service early.

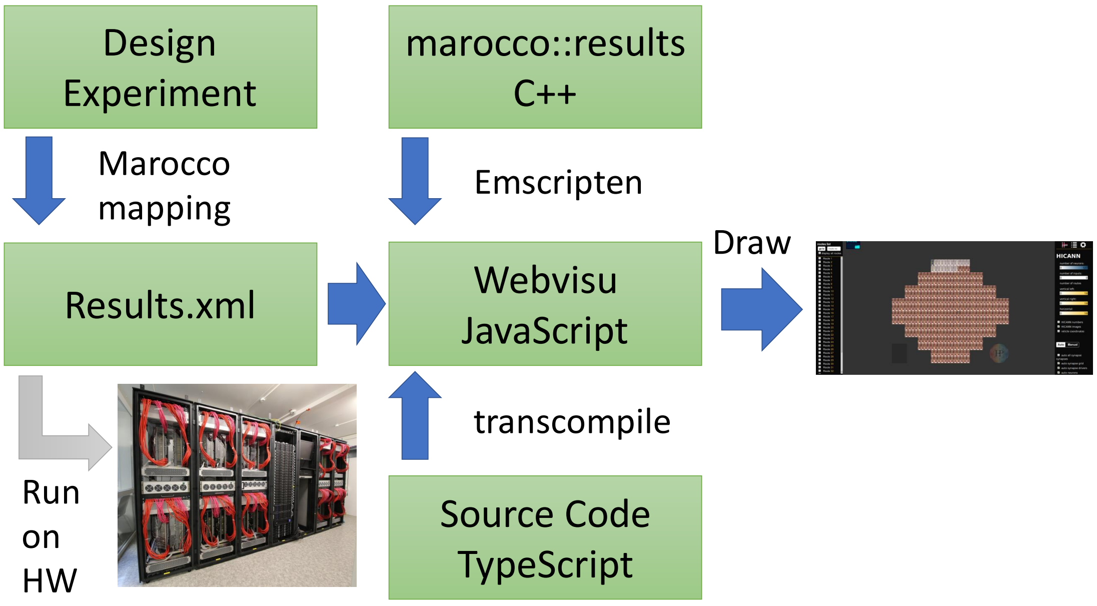
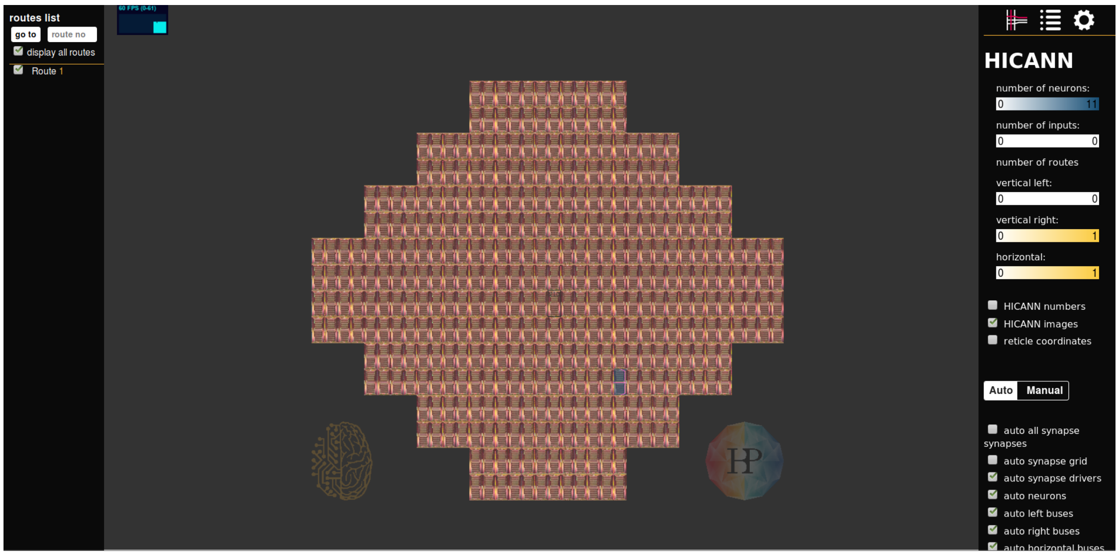


Figure 2: Building steps and usage of the web visualization software. At first, the experiment has to be designed. The marocco mapping tool creates a results file which can run on BrainScaleS hardware. The web visualization can read the same file. The source code is written in TypeScript and gets translated into JavaScript. To read the results file, the marocco::results class gets embedded into JavaScript via Emscripten [4].

1.4 Web Visualization Software

The BrainSacleS 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 in the web visualization software. Therefore, a hardware run is not required. It is enough to run just the mapping. Figure 3 shows the user interface of the web visualization software.



- ✓ display all routes
- ✓ Route 1

number of neurons: 0 – 11
number of inputs: 0 – 0
number of routes
vertical left: 0 – 0
vertical right: 0 – 1
horizontal: 0 – 1

- ☐ HICANN numbers
- ✓ HICANN images
- ☐ reticle coordinates
- ☐ auto all synapse synapses
- ☐ auto synapse grid
- ✓ auto synapse drivers
- ✓ auto neurons
- ✓ auto left buses
- ✓ auto right buses
- ✓ auto horizontal buses

Figure 3: 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 the whole wafer.

The purpose of the visualization is to show experimenters which regions of the wafer are in use and how they interact. Therefore, it is basically a 2D surface where one can zoom in and out and switch drawing 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 4.

The web visualization software was started by Richard Boell as part of his bachelor thesis [2] and maintained by the Electronic Vision(s) group.

2 Optimization

2.1 Search for Optimization Potential

The first task was 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.

It turned out that, especially large networks are very slow. The reason for this is that it is easily possible to get a very large number of synapses and those cause memory problems (about 40MB for all synapses). The investigated network had roughly a half HICANN(4b) filled with synapses, which are about 50,000 ones. So, this was the first thing to improve.

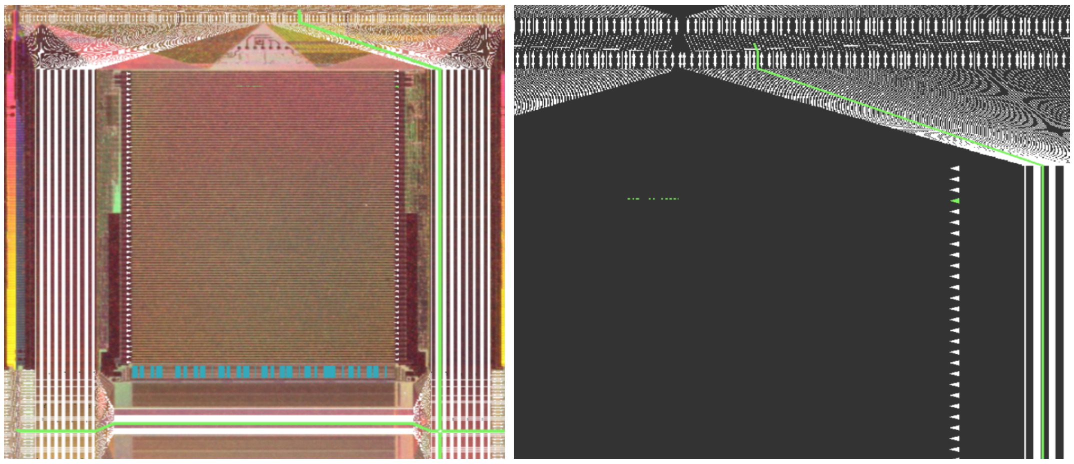

(a) Close look on HICANN (b) Synapse Grid (green dots are used synapses). HICANN image is turned of.

Figure 4: A detailed view of the HICANN-Chip is shown by zooming in. The blue stripes symbolize the neurons. In this case one neuron connects to a few other ones. 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. From there the signal goes back down to the neurons.

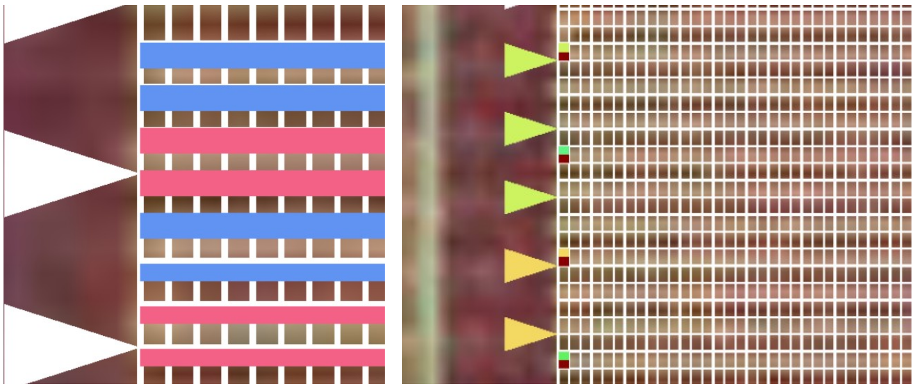
2.2 Synapses as Sprites

The idea was to combine synapses in a large sprite. For this we created a texture of all the graphic objects and inspected the used memory. To our surprise, it was still the same memory used. It turned out that unused graphic objects are not automatically deleted. After deleting the unused graphic objects, the memory consumption decreased by about 30% in this particular case with lots of synapses. The memory used by the synapses decreased to a negligible amount of less than 3%. Furthermore, synapses are now only drawn once at the beginning.

In Figure 5a the synapses are drawn as sprites for each route. The problem here is that the synapses have a different shape. This changes depending on the set resolution, but it is never perfect. So, this still has to be improved.

2.3 Excitatory and Inhibitory Synapses

Another feature that I experimented with, is the marking of synapses as excitatory or inhibitory. How this could look like is shown in Figure 5b. There the upper half of a neuron is colored accordingly to the route color and the lower half is colored red for excitatory and blue for inhibitory synapses. One can also think of a changing transparency depending on synaptic weights. In general, this would double the memory needed for synapses, but as they are drawn as large sprites this would not lead to performance issues.


(a) Synapses drawn as texture; Different thickness because of the resolution. Many synapses next to each other.

(b) Synapses drawn as graphic object; Additional marker to show whether a synapse is inhibitory(blue) or excitatory(red). Only excitatory in this figure.

Figure 5: Different drawn Synapses.

2.4 Further Possible Improvements

After getting rid of the large synaptic impact on memory consumption, there are still lots of things to work on. At first, the web visualization is still very slow at the initialization and is sometimes, especially with large networks, a bit jerky while zooming.

Secondly, as already mentioned, the synapses are not drawn perfectly in the same size. One could either figure out a way to draw them right or draw them below a sprite grid (the white grid in Figure 5).

Additionally, one could improve other elements similar to the synapses. I assume by deleting unused graphic objects and drawing sprites instead, one could decrease the memory consumption by another 5% to 15%. Moreover, there is definitely potential for improvement by drawing and calculating elements only once and make them invisible while not needed.

Other minor improvements could be a global color theme, as most colors are still hard coded or drawing sprites of synapses per HICANN instead of per route. The latter has already been implemented but not all synapses were drawn. This might become important if experiments have lots of routes.

3 Summary

The goal of this internship was to become familiar with the current software and to improve it's performance. At the beginning it was quite hard to become familiar with the work flow in the group. Particularly to follow the communications with many unknown words like HICANN, phymf, Marocco, halbe, spack and sthal is difficult. Moreover, a lot of the software needs much effort to get started with. For example, for PyNN[3] there are a few examples available and also a short documentation, but a short beginner's tutorial would be great. Another nice thing to have would be a very easy and short description of the whole BrainScaleS system of about one page that describes the key figures and the underlying model in an understandable way.

After thoroughly inspections of the current web visualization I could improve it noticeably. Nevertheless, there is still a wide range of possible improvements. This will be part of my following bachelor thesis.

Finally, it was very interesting to understand a bit more about neuromorphic hardware and it is a lot of fun to be part of the team.

[3]Python Interface for the Spikey Neuromorphic Hardware System

References

[1] BrainScaleS system - Neuromorpher Computer geht online `https://www.uni-heidelberg.de/presse/news2016/pm20160316_neuromorpher-computer-geht-online.html`

[2] Visualization of Mapping and Routing of the BrainScaleS System, Bachelor Thesis of Richard Boell, 2018, `http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3760`

[3] Twitter Account of Electronic Vision(s) group `https://twitter.com/brainscales`

[4] Emscripten LLVM to Java Script compiler `https://github.com/emscripten-core/emscripten/wiki#programming-languages`

[5] TypeScript `https://en.wikipedia.org/wiki/Microsoft_TypeScript`

[6] Neuron Circuit Characterization in a Neuromorphic System, Mitja Kleider, 2017 `http://www.kip.uni-heidelberg.de/Veroeffentlichungen/details.php?id=3657`