# Simulation of a Cortical Column Using Leaky Integrate-and-Fire Neurons with Conductance Based Synapses

Moritz Hornung
University of Heidelberg, supervised by Hartmut Schmidt

July 2020

**Abstract**

A cortical column is a neural network which is found in the cerebral cortex of mammals and is considered to be a building block of the brain. It is modeled using cell-type specific connectivity between the neurons that was obtained through anatomical studies. Since this is a well studied model, it would be of interest to simulate it on the BrainScaleS-1 waferscale system to obtain a benchmark for the hardware and to see whether it withstands the imperfections of such a system. Since BrainScaleS-1 uses conductance based leaky integrate-and-fire neurons, the previously existing model that was realised with current based synapses has to be adapted. The goal of this internship was to extend the software simulation to support conductance based synapses.

## Contents

# 1 Introduction

## 1.1 The Leaky Integrate-and-Fire Neuron

The neuronmodel used throughout this work is that of a leaky integrate-and-fire (LIF) neuron. It is a rather simple model that still succeeds at retaining biological relevance and produces reasonable results. The main principle is, as the name suggests, that of a leaky integrator. The dynamics of the membrane voltage are hence given by

$$C_{\mathrm{m}} \frac{du}{dt} = g_{\mathrm{l}}(E_{\mathrm{l}} - u) + I^{\mathrm{syn}} + I^{\mathrm{ext}} \tag{1}$$

Here, $u$ denotes the membrane potential, $C_{\mathrm{m}}$ the membrane capacitance, $g_{\mathrm{l}}$ the leak conductance and $E_{\mathrm{l}}$ the leak potential. In addition, there is an input current which is divided into a synaptic part $I^{\mathrm{syn}}$, modeling synaptic events, and an arbitrary external part $I^{\mathrm{ext}}$, that allows for additional tuning of the model. Taking a look at equation 1, we can identify a leakage ($u \propto -\dot{u}$) and an integration term ($u \propto I$).
Once the membrane potential reaches a certain threshold $\vartheta$, a spike is emitted and it is pulled onto the reset potential $E_{\mathrm{reset}}$. To account for the time it takes a real neuron to reset the ion channels, the refractory time $\tau_{\mathrm{ref}}$ is introduced. After emitting a spike, the neuron is clamped to the reset potential for this time and only then is allowed to propagate according to the above equation. Since the leakage potential and leakage conductance as well as the membrane capacitance are constant, the time evolution of the membrane potential is determined mainly through the synaptic current $I^{\mathrm{syn}}$. There are two commonly used ways of describing the synapses, a current based (CUBA) and a conductance based (COBA) model. As both of them are of relevance, a short summary will be given in the following paragraphs.
In the CUBA case, the main assumption is that most synapses are far away from the soma and thus what reaches the membrane is effectively a current pulse. The synaptic current is then given through

$$I^{\mathrm{syn}} = \sum_k \sum_s \omega_k \varepsilon(t, t_{s,k}) \tag{2}$$

with the synaptic weights $\omega_k$ and the synaptic kernel $\varepsilon(t, t_{s,k})$ that describes the time course of the synaptic event. The first sum is over the synapses and the second sum over the spikes with the respective spiketimes $t_{s,k}$. In this model,

the synaptic currents are independent of each other and the postsynaptic potential (PSP) is solely determined by the synaptic weights.
The mechanisms of the COBA model stay a bit closer to that of the biological neurons, since the assumption of far away synapses is dropped. A synaptic event is described through a change of the conductance $g_{\mathrm{e/i}}$ towards an excitatory or inhibitory reversal potential $E_{\mathrm{rev,e/i}}$. In this case the current is given by

$$I^{\mathrm{syn}} = g_{\mathrm{e}}(t)(E_{\mathrm{rev,e}} - u) + g_{\mathrm{i}}(t)(E_{\mathrm{rev,i}} - u) \tag{3}$$

where the time development of the conductances is controlled through the synaptic weights and the synaptic kernel

$$g_{\mathrm{e/i}}(t) = \sum_k \sum_s \omega_k \varepsilon_{\mathrm{e/i}}(t, t_{s,k}) \tag{4}$$

The dynamics of this model are a lot more complex, since the membrane potential appears in the formula for the synaptic current. Because of this, PSPs are affected by other synaptic events that arrived shortly before. Another important effect that occurs only in the COBA case is the change of the membrane reaction speed. The time constant of a capacitor is generally given as $\tau_{\mathrm{m}} = C/g$. Since COBA synapses change the overall conductance of the membrane, this naturally results in a reduction of the membrane time constant $\tau_{\mathrm{m}}$

## 1.2 The Cortical Microcircuit Model

A cortical column is a network of neurons that can be found in the early sensory cortex of mammals. It is thought to be a building block of the brain that support its functionality. There are many different approaches to model the column. Throughout this work a model based on LIF neurons, which was first proposed by Potjans and Diesman in 2014, will be applied. [1]
The model consists of 4 layers, each one being realised through two neuron populations. These 4 layers are labeled as L2/3, L4, L5 and L6 respectively and are divided into an excitatory (E) and an inhibitory part (I). Anatomical and physiological data of the structure is used to derive a connectivity map between the individual layers. The populations are then connected randomly using the connection probabilities obtained from said map. A schematic view of the resulting connections can be seen in Figure 1. Overall, the model encompasses about 80000 neurons and roughly $3 \times 10^8$ synapses, thereby covering 1 mm$^2$ of the cerebral cortex. A more detailed description of the methods used to derive the connectivity map can be found in the initial publication from Potjans and Diesmann [1].

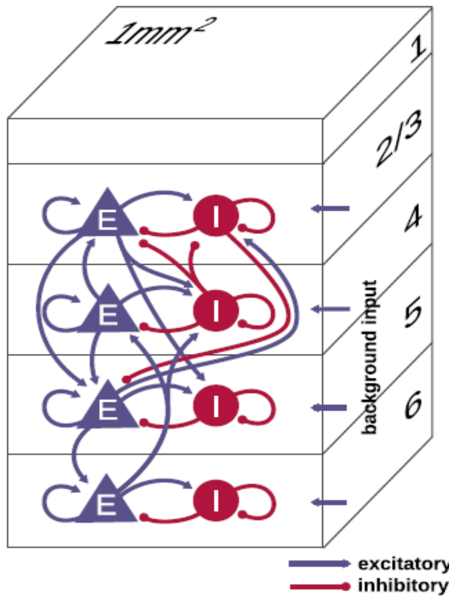


Figure 1: Schematic illustration of the connectivity in a cortical column taken from [2]

## 1.3 The Simulation Software NEST

Since part of this report also deals with speeding up the simulation, this section gives a short overview of the simulation software NEST and how it handles parallelization.
NEST is a simulation software aimed towards simulating large neural networks. It focuses on the dynamics of neural systems instead of single neurons, making it a reasonable choice for simulating a large scale model such as the column. The simulator also supports the python based language for neural networks PyNN, which is also used on the BrainScaleS-1 system. This is useful when comparing results between the software simulation and the implementation on the hardware, since it guarantees an identical experiment.
Given that NEST is aimed at simulating large networks, it already has a built-in parallelization mechanism that can be used to speed up simulation time. It supports multithreading and/or distributed computing to achieve the optimal speed up. Internally, NEST combines the number of running processes $N_\mathrm{p}$ and the number of threads in every process $N_\mathrm{t}$ to a total of $N_\mathrm{vp} = N_\mathrm{p} * N_\mathrm{t}$ virtual processes (VPs). Then the neurons are distributed equally over all of the virtual processes

and integrated individually in time steps of the simulation resolution (0.1 ms per default). Communication with the other VPs only occurs in larger time steps, which are defined by the minimum delay used for the connections. This is possible because any occurring spike can only influence the rest of the simulation after the minimum delay has passed.

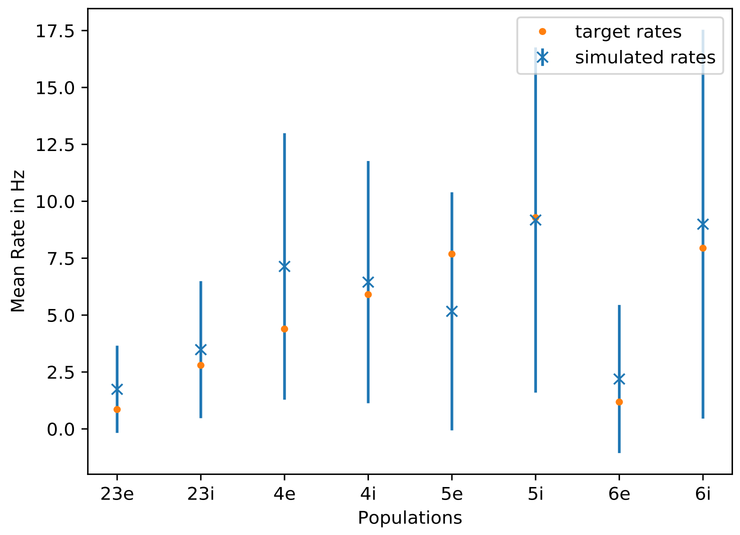


Figure 2: Mean firing rates of the downscaled 10% CUBA model. The additional scaling parameters are set to exc = 0.5 and inh = 1.5 and the simulation was run up to 5000ms with a timestep of 0.1ms. The first second of the simulation is excluded since the network needs time to achieve stable firing rates. The error bars that are plotted give an insight over the spread of the firing rates throughout the neuron populations and are not to be taken as an error of the mean value itself.

## 2 Implementation

The model as described in section 1.2 is too large to apply it directly to the hardware, since the amount of neurons and synapses on the waferscale system is limited. Therefore, the simulation has to be downscaled to a fraction of its initial size. Unfortunately, keeping the amount of synapses per neuron constant when scaling down is not possible, since that would still take too many synapses. To

bypass this problem, the indegree for the neurons is decreased and simultaneously the weights of the synapses are increased. This is done to keep the overall input to the neurons constant. Using this approach to the scaling, the variance of the model grows, since now single neurons have more influence. To balance this, the external poisson input used in the original model is substituted with a DC input, which counteracts the change in variance to a certain degree. Because this is not enough to guarantee the same firing rates as in the full model, the two additional multiplicative parameters ”exc” and ”inh” were added. They scale the excitatory and inhibitory weights independently and can be set manually when running a simulation. To obtain the best settings, a parameter sweep was performed resulting in exc = 0.7 and inh = 1.1. The problem with the variance still persists, so for now only the first order statistics are of relevance in the results.
The starting point for this internship was a properly working, scaled CUBA simulation of the Cortical Column, that yields per population firing rates similar to the ones of the full-scale model (Figure 2).

## 2.1 Transition towards COBA Simulation

As mentioned in the beginning, a software simulation based on CUBA neurons is not sufficient. To obtain an impression of what the hardware simulation is going to look like, the model has to be adapted to match the conditions of the BrainScaleS-1 hardware. This means that the synapse model needs to be changed to support COBA synapses.
The main aspect that needs to be taken care of when transitioning the model are the synaptic weights, since they change from being a current to being a conductance. Starting from the known weights of the CUBA simulation, we can estimate those of the COBA model using a rather simple approach.
As a first step, a mean membrane potential of the neurons is calculated. To this end, the number of synapses per neuron $K$ together with the average firing rates $r$ and the known synaptic weights $w_{\text{e/i}}$ of the CUBA model are combined, which yields an estimate for the average excitatory and inhibtory synaptic input $x_{\text{e/i}}$ to the neurons

$$x_{\text{e/i}} = w_{\text{e/i}} * K * r \tag{5}$$

This can easily be converted to the mean membrane potential $V_{\text{mean}}$ by combining it with the synaptic time constant and the membrane resistance. Note that the external input is included in the leak potential, since the poisson input was substituted with a DC input, which is equivalent to an increased leak potential.

$$V_{\mathrm{mean}} = V_{\mathrm{l}} + \frac{\tau_{\mathrm{m}}}{C_{\mathrm{m}}}(\tau_{\mathrm{syn}}(x_{\mathrm{e}} + x_{\mathrm{i}})) \tag{6}$$

Now the formulas for the synaptic current described in section 1.1 can be compared to calculate the synaptic weights for the COBA model from those of the CUBA case.

$$g_{\mathrm{e/i}} = \frac{w_{\mathrm{e/i}}}{V_{\mathrm{rev,\ e/i}} - V_{\mathrm{mean}}} \tag{7}$$

Using the weights obtained from equation 7, we find the firing rates shown in Figure 3. They are already very close to the rates in the CUBA simulation, thereby justifying the ansatz used. In both the COBA and the CUBA model, the main deviations from the target rates are within populations 4e and 5e. The 4e population is spiking at an increased rate where as the rate of the 5e population is too small. To further improve the results, the thresholds of the different populations could be adjusted.

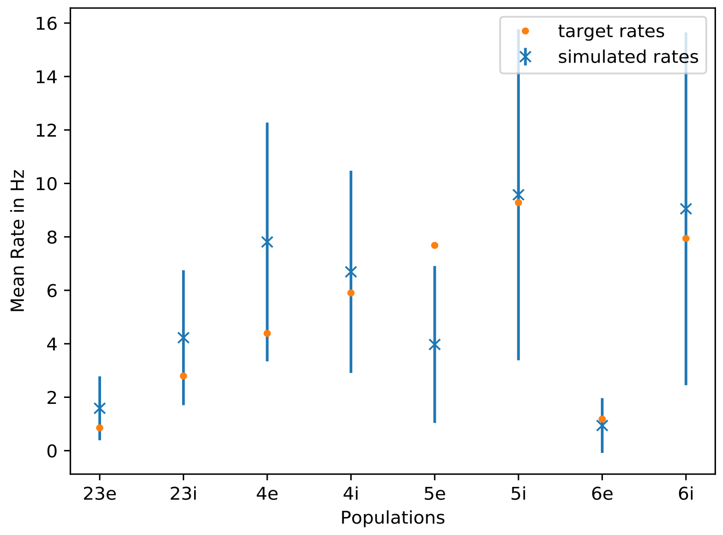


Figure 3: Firing rates of the 10% COBA simulation using the synaptic weights obtained with equation 7. The simulation was run up to 5000 ms with a timestep of 0.1 ms, once again excluding the first second when calculating the rates. The optimal additional scaling parameters differ from the CUBA case with exc = 0.9 and inh = 1.5

## 2.2 Speed Up

Transitioning to the COBA model increases the simulation time to roughly 5 times that of the CUBA case, the reason being the more complex dynamics of the COBA equations. Hence it is convenient to do a speed up using the NEST intern parelleliztion mechanisms.
A considerable speed up can be achieved, using the multithreading aspect of parallelization. Since NEST already has an internal option to do this, the number of threads used can be specified when setting up the simulator. The possibility to use distributed computing through MPI was neglected, since the NEST version was not compiled to support it and the speed up using multithreading was enough to run the simulations without having to wait too long.
Figure 4 shows the simulation time as a function of the number of threads used. Due to a trial to trial variation, the measurements are repeated 50 times and the ensuing mean and standard deviation are depicted. The Plot shows a strong de-

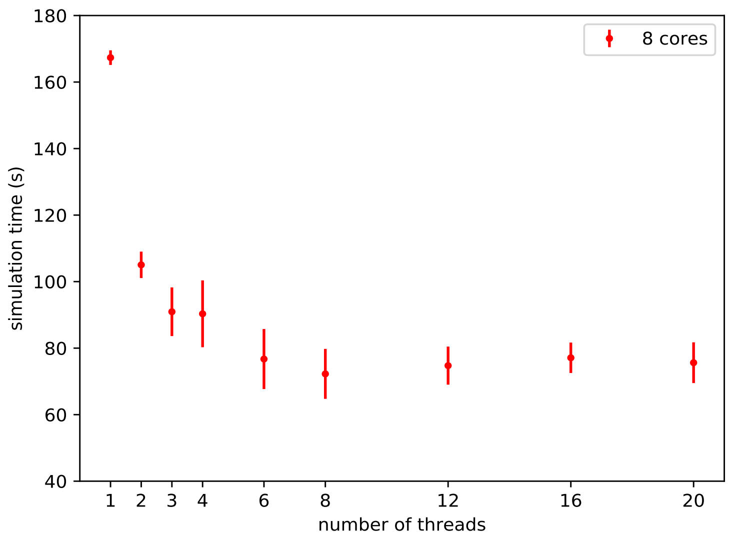


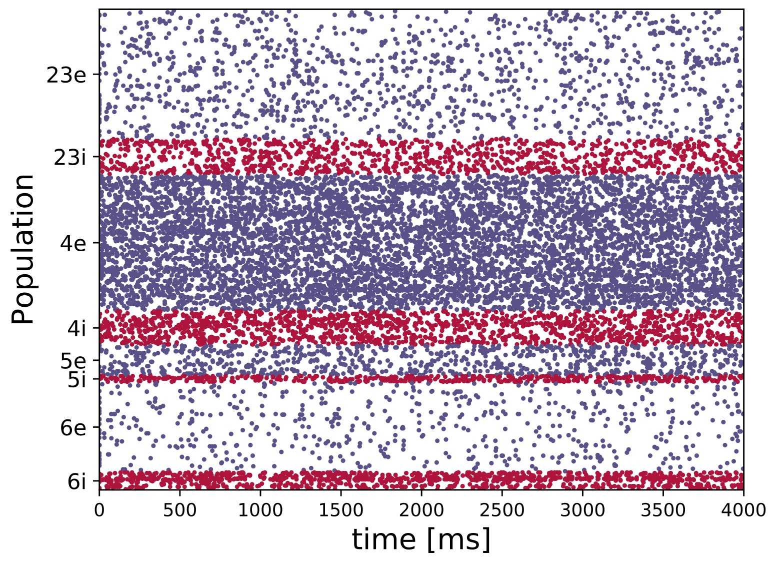
Figure 4: Simulation time depending on the number of threads used in the simulation. The measurements were taken using a single host with 4 physical and 8 virtual cores and averaged over 50 runs

crease over the first few threads up to the number of cores used, where it reaches a minimum at roughly one third of the initial simulation time. In the NEST documentation, it is suggested to try oversubscribing (using more threads than cores), as this might decrease the simulation time further [3]. However this did not work here, even leading to a slightly worse performance. It should be noted that the simulation time also depends heavily on the host and its current workload.

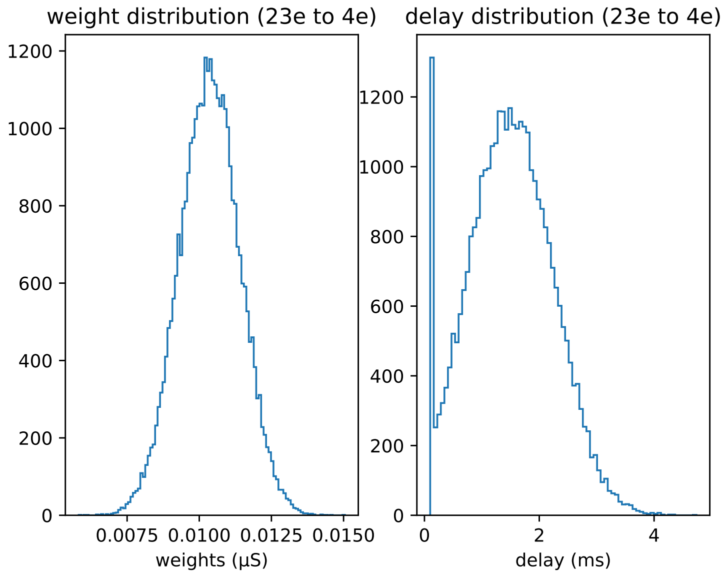
## 2.3 Analysis Tools

This last section covers part of the analysis tools used to investigate the column. When doing the transition to the COBA model, there were some problems setting the weights correctly. To generate the weight distributions for the inhibitory connections in the CUBA simulation, a random number generator with the boundaries $-\infty$ and 0 was used. After the transition, all the weights are now positive. However the boundary restriction still applied, which led to the weights being set to zero for the inhibitory connections. Since there was no logging of the drawn weights, this was overlooked. To avoid errors like these and help with further debugging in the

future, the options to print a raster plot of the individual spiketimes for the neurons in every population and to plot the drawn weight distributions were added.
Figure 5 (a) shows a spike time raster plot. For every spike a neuron sends out, a dot is generated in the plot at the corresponding time. To keep the data manageable, this is only done for 10 % of the neurons in the downscaled simulation. This kind of plot helps noticing irregularities in the spiking pattern that might hint at different error sources. It also helps with identifying behaviour were only very few neurons in every population spike and the others are inactive, which is a case that occurred before and is also not desired.
5(b) shows a histogram of the weight and delay distribution that are used for a specific connection. We clearly see the gaussian shape underlying the distributions. The peak that can be seen for small delays close to zero originate from the boundary condition that only non negative delays are used. When drawing a negative number, it will automatically be set to 0.1 ms.



(a) Spike time raster plot for 10% of the neurons in every population



(b) Weight and delay distribution for the connection between layers 23E and 4E

Figure 5: Exemplary look of the plots added to the analysis tool

## 3 Summary and Outlook

The cortical column is a rather well known structure found in the early sensoric cortex of the brain. There are already many studies concerning it, using a variety of neuronmodels. In particular, there are already several studies based on CUBA LIF neurons.
The main purpose of this internship was to get used to the existing software implementation of the CUBA LIF model and to conduct the transition towards a model based on COBA LIF neurons. This was done successfully with the approach described in section 2.1. On top of that, the simulation was sped up and a raster spiketime plot as well as the option to plot the weight distributions between the neuron populations were added to the analysis tool.
With the simulation running properly, the next step now is to conduct some stability tests of the network, e.g. checking the influence of some variation between the neuron parameters of individual neurons. This distribution of the parameters is expected on the hardware due to the manufacturing process. Therefor a NEST simulation that includes parameter variation might help with predicting some of the effects that might occur when running the model on the BrainScaleS-1 system.

## References

[1] Tobias C. Potjans, Markus Diesmann, The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model, Cerebral Cortex, Volume 24, Issue 3, March 2014, Pages 785–806, https://doi.org/10.1093/cercor/bhs358

[2] Sacha J. van Albada, Andrew G. Rowley, Johanna Senk, Michael Hopkins, Maximilian Schmidt, Alan B. Stokes, David R. Lester, Markus Diesmann, Steve B. Furber Performance Comparison of the Digital Neuromorphic Hardware SpiNNaker and the Neural Network Simulation Software NEST for a Full-Scale Cortical Microcircuit Model, https://www.frontiersin.org/article/10.3389/fnins.2018.00291

[3] NEST documentation, https://nest-simulator.readthedocs.io/en/nest-2.20.1/guides/index.html