Comparing Neuromorphic Systems by Solving Sudoku Problems

Christoph Ostrau*, Christian Klarhorst, Michael Thies, and Ulrich Rückert
Cluster of Excellence Cognitive Interaction Technology
CITEC, Bielefeld University
Bielefeld, Germany
Email: *costrau@techfak.uni-bielefeld.de

***Abstract*—In the field of neuromorphic computing several hardware accelerators for spiking neural networks have been introduced, but few studies actually compare different systems. These comparative studies reveal difficulties in porting an existing network to a specific system and in predicting its performance indicators. Finding a common network architecture that is suited for all target platforms and at the same time yields decent results is a major challenge. In this contribution, we show that a winner-takes-all inspired network structure can be employed to solve Sudoku puzzles on three diverse hardware accelerators. By exploring several network implementations, we measured the number of solved puzzles in a set of 100 assorted Sudokus, as well as time and energy to solution. Concerning the last two indicators, our measurements indicate that it can be beneficial to port a network to an analogue hardware system.**

***Index Terms*—neuromorphic hardware, spiking neural networks, Sudoku, winner-takes-all**

I. Introduction

There is an increasingly growing interest in the field of spiking neural networks. The reason is twofold: new ideas for the efficient use of these event-based systems pop up everywhere, while at the same time hardware accelerators become available and mature. Particular examples are these two large scale systems: the fully digital SpiNNaker system [1] and the mixed-signal BrainScaleS system [2]. Both have been developed as part of the Human Brain Project. Various publications have shown the success of these systems in different areas of computation [3]–[6]. Other neuromorphic systems include e.g. IBM TrueNorth [7], Intel Loihi [8] and the DYNAPs system [9].

However, comparative studies using the same network model across platforms are still rare (see e.g. [10], [11]). The aim of this publication is to show case of an example network architecture running on three neuromorphic platforms and a CPU simulator for a rough efficiency comparison of the systems. At the same time we reveal some challenges encountered when porting a spiking neural network to various target platforms, and how these can be countered in this specific case. The algorithm we chose employs a winner-takes-all architecture for solving Sudoku puzzles. This is an example of a typical constraint satisfaction problem, which belongs to the class of NP-hard problems. The spiking architecture was first proposed in [12], and has been adapted in several publications and deployed to neuromorphic hardware [13], [14]. The contribution of this work is to adapt the existing model for several neuromorphic platforms and comparing the relative efficiency. In contrast to aforementioned publications, we show that the model is capable of solving most of the provided 400 assorted Sudokus, 100 puzzles per size. This reveals that the network parameters have to be adapted to the complexity class of the constraint satisfaction problem. The first section introduces our target platforms and methods applied. Afterwards, parameter sweeps, simulation and measurement results are provided, as well as an outlook on future work.

II. Methods

In this section we describe the neuron model and our target platforms: the NEST simulator, SpiNNaker, Spikey and BrainScaleS. Afterwards we present the network architectures used.

A. Neuron Model

All target platforms presented below implement various neuron models. Simulators are quite flexible in their employed models, but neuromorphic hardware may be limited to only a few models. Hence, the networks introduced in this work rely on Integrate-and-Fire neurons with conductance-based synapses [15], which are supported by all target platforms. The model is defined by the following differential equation for the membrane voltage $V(t)$

$$C_m \frac{dV(t)}{dt} = -g_\mathrm{L}(V(t) - V_\mathrm{rest}) - I(t) \ , \tag{1}$$

where C_m defines the capacitance, g_L the leak conductance, V_rest the resting potential and I the synaptic input current. A spike is triggered as soon as the membrane potential crosses the threshold V_th and the membrane is reset to V_reset:

$$V(t) \leftarrow V_\mathrm{reset} \quad \text{if} \quad V(t) \geq V_\mathrm{th} \ . \tag{2}$$

The synaptic current is defined by

$$I(t) = g_\mathrm{exc}(t)(V(t) - E_\mathrm{exc}) + g_\mathrm{inh}(t)(V(t) - E_\mathrm{inh}) \ , \tag{3}$$

using $g_i(t)$ and E_i as the conductance and reversal potential of the excitatory and the inhibitory channel. The conductance is driven by input spikes and is exponential decaying:

$$-\tau_i \frac{dg_i(t)}{dt} = g_i(t) \tag{4}$$

$$g_i(t) \leftarrow g_i(t) + w_i \ \forall \text{ Spikes at time } t \text{ at synapse } i \tag{5}$$

introducing the additional synapse time constant τ_i, while i represents the two channels (excitatory and inhibitory). This synapse model simplifies computation by using shared synapse parameters (except the connection weight w_i) for all synapses of a post-synaptic neuron. Only inhibitory and excitatory synapses can still be distinguished leading to two distinct sets of parameters. Whether a synapse acts inhibitory or excitatory solely depends on the sign of $(V(t) - E_i)$ (inhibitory for positive values).

Using current-based synapses is a second possibility, and we only employ these where explicitly stated.

$$-\tau_i \frac{dI_i(t)}{dt} = I_i(t) \tag{6}$$

$$I_i(t) \leftarrow I_i(t) \pm w_i \ \ \forall \text{ Spikes at time } t \text{ at synapse } i \tag{7}$$

Here, two current variables for excitatory and inhibitory currents are directly altered by the connection weights, and the exponential decay is similar to the one in conductance-based models. Reversal potentials are not necessary in this model, thus the model is less complex.

B. NEST

The software simulator NEST [16] was used in version 2.14. Recently it was shown that the simulator can be efficiently scaled to larger networks [17]. In this publication, we will only make use of rather small networks, which is why we avoid the usage of MPI, and only use four threads for parallel simulation of our networks. All networks are simulated using the `iaf_cond_exp` (`iaf_currr_exp`) model, which implements the model described above.

C. SpiNNaker

A single chip of the fully digital many-core architecture SpiNNaker [1] is composed of 18 general purpose ARM968 cores. Systems are available in different sizes, beginning with a four chip board (called SpiNN-3), a larger 48-chip board (SpiNN-5) and up to the large scale neuromorphic cluster provided in the Human Brain Project (see e.g. [18]. To access the hardware, the platform developers provide the software tool-chain sPyNNaker, which is used in the current release version 4.0.0 and provides an API based on PyNN version 0.8.3 [19]. The system supports several neuron and synapse models while most of our experiments make use of PyNN's `IF_cond_exp` model only. Here, the default setup allows to simulate up to 255 neurons per core in real-time, which yields roughly 16,000 neurons for the smallest board. This model implementation has the disadvantage that using large inhibitory inputs leads to numerical artefacts, which can be countered by reducing the time-step of the system from $1.0\,\mathrm{ms}$ to $0.1\,\mathrm{ms}$. In the current release software, this will lead to a slow-down of the simulation by a factor of 10 compared to real-time. In some experiments, these artefacts still appear despite the increased accuracy of the reduced time-step. We will make use of the current-based `IF_curr_exp` model instead, which does not suffer from these issues. Furthermore, we reduced the number of neurons per core to avoid network bottlenecks and allow higher firing rates in general.

D. Spikey

Spikey is a single chip system [20] and is the predecessor to BrainScaleS (see below). It implements a physical (analogue) VLSI replacement for the above mentioned Integrate-And-Fire models, which is restricted in the number of parameters available through the front-end software (an early version of PyNN). Thus, some neuron model parameters are fixed (like the membrane capacitance) while all others are limited in the available range. The communication between neurons is however realized in a digital way. The full system composed of two chips supports the emulation of 384 neurons each providing 256 synapses accelerated by a factor $10,000$ compared to biological real-time.

E. BrainScaleS

Similar to the Spikey system, BrainScaleS provides a physical model for accelerated emulation of the Adaptive Exponential Integrate-And-Fire model [2]. The adaptive part can be turned off such that the circuits emulate the behaviour of the above mentioned `IF_cond_exp` model. This system uses waferscale-integration [21] to combine 352 HICANN chips in a single system, with each of them containing 512 neuron circuits with 220 synapses, respectively. Up to 64 circuits are combined to form a single neuron, and in this publication we always combine 4 circuits to a single neuron. These neurons communicate via a digital communication fabric. In contrast to the Spikey system, the front-end software allows to set all aforementioned parameters.

F. Network Description

All introduced systems support the PyNN API [19]. However, the supported version of PyNN differs from simulator to simulator necessitating platform specific network descriptions. In [10] the framework Cypress has been proposed, which allows to access all aforementioned platforms with a single network description[1]. In this framework, all network descriptions are implemented using the C++ programming language. Furthermore, the library provides all kind of auxiliaries like a C++ plotting interface.

G. Network Architecture

Here, we describe the network architecture for our spiking Sudoku solver. We will use the term connection for a single synaptic connection, while projection refers to the connectivity between populations (a projection consists of at least one connection). The architecture is based on a Winner-Takes-All

[1] https://github.com/hbp-unibi/cypress

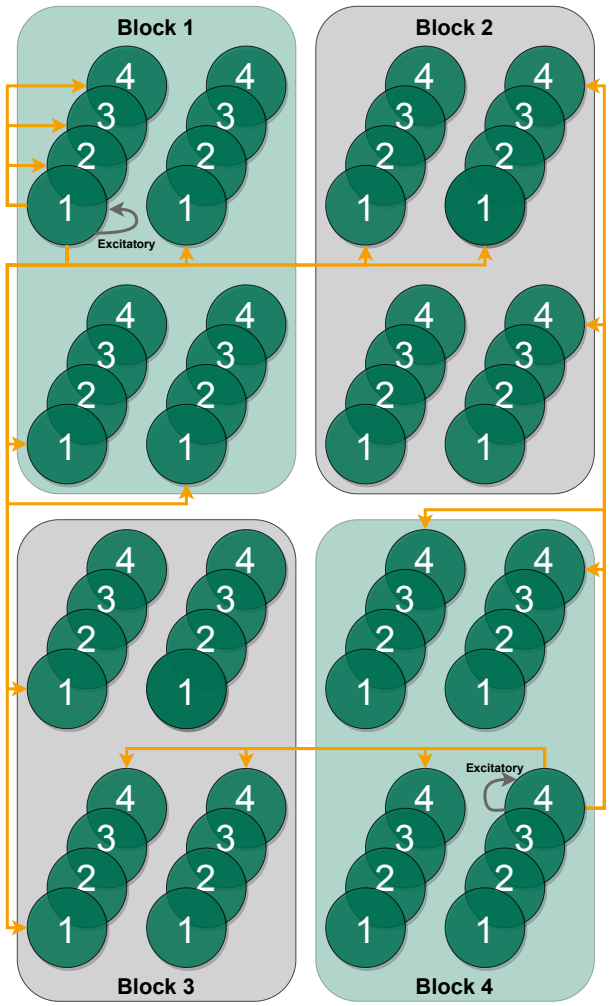

Fig. 1. Network architecture of the spiking Sudoku solver for a small scale Sudoku only containing the numbers *1-4*. Green circles represent populations of neurons, boxes express sub-squares of the Sudoku. Every cell of the puzzle is shown as four populations placed next to each other. Arrows depict projections, with orange arrows for inhibitory projections, black arrows for excitatory projections. Only the outgoing projections of the population representing *1* in the upper left cell and *4* in the lower right cell of the Sudoku are shown.

architecture [22]. Populations are connected such that their activity is in competition. Lateral inhibition between different populations allows only a single population to be activated at a time. Through self activation a population can sustain its activity for an extended period of time. The last ingredient for a working Winner-Takes-All setup is independent random Poisson distributed spike input to all of the neurons. As soon as there is enough random input to one population, it gets activated, and stays active due to its self-excitation. At the same time, lateral inhibition will suppress all other competing populations from becoming activated.

This architecture can now be employed to solve a Sudoku problem. First, every assignable number in a specific cell is represented by a dedicated population. For a small scale example using only the numbers *1* to *4* the Sudoku problem consists of $4 \cdot 4$ cells, while the corresponding spiking neural network uses $4 \cdot 4 \cdot 4$ populations to encode all possible solutions to this Sudoku problem (compare Fig. 1). Second, every population has self-excitation as described above. The only difference from the general winner-takes-all architecture is related to the inhibition: a population is not inhibiting all other populations, instead the constraints of the Sudoku rules lead to exactly these inhibitory projections:

1) Every cell can only have a single value. Populations have to inhibit all other populations situated in the same cell
2) A value can only appear once in a row: every population is inhibiting all other populations in the same row that are representing the same number as the source population
3) A value can only appear once in a column: every population inhibits all other populations in the same column if they represent the same value as the source population
4) A value appears only once in a sub-block: every population inhibits populations representing the same number in a sub-block of the Sudoku

In the small scale example this leads to $3+3+3+1$ outgoing inhibitory projections per population and thus to 640 inhibitory connections (see also Fig. 1). On top of that, there is one self-excitatory projection and one random input projection per population.

This completes the architecture for solving arbitrary Sudokus (of size 4×4). To solve a specific Sudoku puzzle we further excite those populations that represent the numbers given in the specific Sudoku puzzle, which is the only puzzle specific part in the architecture. This architecture well be referred to as `architecture#1`.

For evaluation, spikes are sorted into time-bins and the population with the highest number of spikes per bin is marked as the activated population for that bin.

For most of the target simulators, run-time termination as soon as the Sudoku is solved is not available. Thus, we need to specify the run-time before the beginning of the simulation. This means that possibly not all Sudokus are solved, as statistical outliers might need more solving time than provided.

H. Alternative Implementations for neuromorphic hardware

When trying to execute the above described network on SpiNNaker, the middle-ware will make use of more cores than actually necessary for simulating networks of that size. This is due to the fact that currently the software maps at most one population to a single core. In the above mentioned example of a 4×4 Sudoku, the network itself consists of 64 populations, and additional populations for the random input. This unnecessarily exceeds the capacity of a small SpiNN-3 board consisting of 4×18 cores of which 4×16 cores are used for the simulation of the network itself. One remedy is to employ the bigger SpiNN-5 board, which provides 49×16 cores for simulating neurons. The alternative is the merging of all populations into a single virtual population, and to manually select individual sub-populations. This however requires us to expand projections on a higher level instead of using

predefined connectors that are provided by the backend. Thus, set-up time might increase with the merged virtual population. We will reference this approach as `architecture#2`.

A very different problem arises when employing the mixed-signal Spikey hardware. This platform restricts us to use bio-realistic connection schemes in a manner that all outgoing connections of a neuron can only be either excitatory or inhibitory. This conflicts with our approach of using self-excitation. The network structure is thus changed to make up for this limitation: Every population requires two additional neurons, which are both excited by the population itself. One of these neurons is then projected back to the population and therefore implements the self excitation. The second neuron is responsible for the inhibition according to the above mentioned rules. This effectively restores both mechanisms and makes it possible to solve Sudokus on this specific platform and will be referred to as `architecture#3`.

III. Results

This section presents the achieved results on all introduced platforms. The architecture is analysed and chosen parameters are motivated. We begin with small scaled Sudokus, which can be used for extensive parameter sweeps and showcase basic functionality. In the next step we show larger examples, before looking at the second architecture suited for the Spikey system. We conclude with the evaluation on all platforms, provide performance measures and roughly compare the energy consumption of the platforms. The software that has been used to create the results can be found online[2].

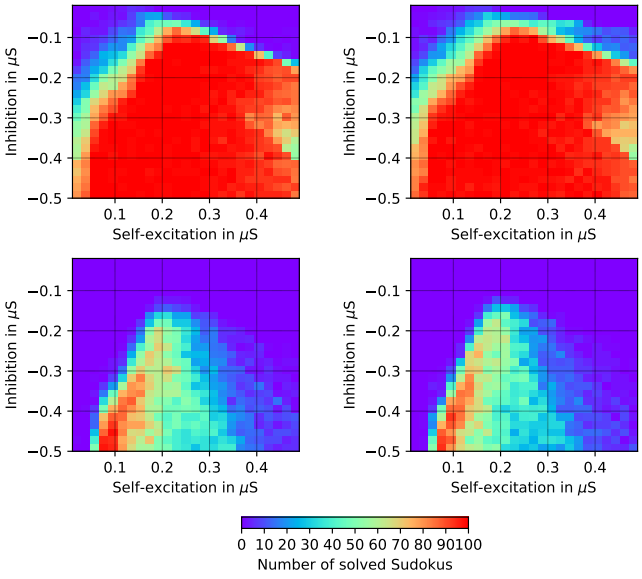

Fig. 2. Parameter Sweep for (left to right) 4×4 with 2 neurons, 4×4 with 3 neurons, 6×6 with 2 neurons and 6×6 with 3 neurons. y-axis represents the inhibitory weight that is shared for all connections, x-axis is the self-excitation weight, and the z-axis depicts the number of unsolved Sudokus of 100 assorted Sudokus. All images were created using the Nest simulator.

[2]https://github.com/hbp-unibi/SpikingSudokuSolver

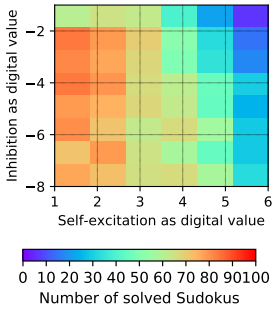

Fig. 3. Parameter Sweep for 4×4 Sudokus with 2 neurons on the BrainScaleS system. Weights are set as low-level digital 4-bit value without unit.

A. Small Scale Sudokus

In the small scale case we make use of the `fixed_fan_in` connector (PyNN's `FixedNumberPreConnector`) for all inhibitory and excitatory projections, which fixes the number of actual in-going connections between two populations. First, this reduces the actual number of connections in comparison to a fully connected network. Second, the number of in-going connections for a neuron is independent of the number of neurons representing a number. Fig. 2 shows parameter sweeps for connection weights for 4×4 and 6×6 Sudokus, where the number of connections per target neuron is set to one. The parameter sweeps show that the solving probability is independent of the number of neurons, which is related to the type of the connector we chose. Furthermore, the parameter range for solving all 100 Sudokus is quite broad in both cases. However, one can already see that with increasing Sudoku size the probability of solving a random Sudoku is decreasing.

For the BrainScaleS system we had to adapt the parameters found for the simulator. This is shown in Fig. 3. Limited parameter ranges and analogue mismatch between different neurons and synapses yield a degradation in solving capabilities.

B. Larger Sudokus

In Fig. 4 parameter sweeps for larger Sudokus have been executed. The images show that the range of valid parameters can be described by a thin line, and only small spots of the parameter space perform best. Furthermore, the number of solved Sudokus is significantly reduced compared to small scale Sudokus. This is partly due to the fixed simulation time because we are not able to interrupt the simulation as a soon as a Sudoku is solved. For parameter evaluation (see below), run-times are significantly increased. The decreased robustness in parameter space is most probably due to the increased complexity range of larger Sudokus.

C. 3rd Network Architecture

The `architecture#3` developed for the Spikey system is different to those discussed before, thus we treat it as a separate case. On the Spikey system we use maximal redundancy to reduce the effects of neuron to neuron variations,

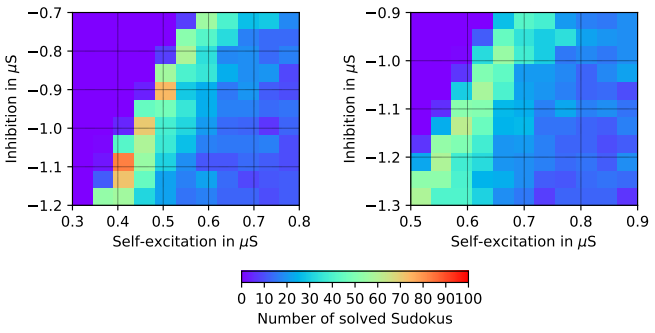

Fig. 4. Parameter Sweep for (left to right) 8×8 and 9×9 Sudokus with 2 neurons.

which is due to the analogue nature of the system. This means that populations encoding numbers consist of three neurons, while external populations responsible for self-excitation and inhibition consist of a single neuron. Hence, we use fully connected projections between populations and helper neurons. For the random input noise, we chose to have two random noise sources per population. This network does not yet fully utilize the available neurons on the Spikey system. However, all synapse drivers are in use such that addition neurons could not be connected to the network in any meaningful way.

The same network was evaluated in NEST, only using two neurons per population. The results of the respective parameter sweep are shown in Fig. 5. Again, rather broad parameter ranges are available to solve nearly all tested Sudokus. Nevertheless, the analogue platform is not able to solve all of the given Sudokus puzzles.

D. Comparison of platforms

For direct comparison of all platforms, we evaluate the three architectures on all platforms where applicable. For measuring the energy consumption during the simulations, we used a Ruideng UM25C inserted into the $5/12\,\mathrm{V}$ supply lane of the hardware systems. We cross checked the accuracy using a Keithley 2450 SourceMeter and deviations were in the range of the measurement accuracy. NEST simulations were executed using four threads of four-core Intel Core i7-4710MQ, while power was measured using a PeakTech 9035 with idle values subtracted. The respective power usage was then multiplied with the average wall-clock time to solution, which yields the average energy that was used to solve a random Sudoku. However, we were not able to measure the power-consumption directly for the BrainScaleS system. To still be able to give a rough approximation, we will estimate the used power by falling back to published measurements. In [3] the authors report an energy consumption of $0.1 - 10\,\mathrm{nJ}$. Thus, we record the number of spikes per population and assume a consumption of $5\,\mathrm{nJ}$ per pre-synaptic event.

Real-time to solution was calculated using the respective speed-up/slow-down factor of the simulator. For NEST this is non-trivial, because the simulator does not give any guarantee about simulation speed compared to real-time. We approximate it by using the real-time of the full simulation, and scale accordingly to biological time scales.

The results are presented in Table I, where simulation times have been increased compared to the simulations in the parameter sweeps above. All target systems show general capability of solving most of the provided puzzles. Average solving-times differ between systems due to accuracy and implementation differences.

The SpiNNaker system was used with an increased time resolution: the default time-step of the Euler-integrator, allowing to execute simulations in real-time, is $1\,\mathrm{ms}$. Because the neuron model in use is not yet fully optimised for the platform, larger deviations in the simulation results appear. This difference is negligible in many cases, but not when using larger inhibitory input as in our model. The issue reappears with larger Sudokus, which is why we evaluated these simulations using the simpler current-based model. Thus, we set the time-step to $0.1\,\mathrm{ms}$ in all simulations on SpiNNaker, which slows down the simulation by a factor of 10. The work done in [13] shows, that using the current-based Integrate-And-Fire model, SpiNNaker is able to solve Sudokus in real-time. In their network, the authors do not use self-excitation connections, which reduces the workload for the hardware system. Nevertheless, an analysis whether their implementation is able to solve a broad range of Sudokus is missing. All in all, it should be kept in mind that running in real-time reduces the used energy by a factor of 10 for finding the solution. Furthermore, `architecture#1` was evaluated on the SpiNN-5 board, which already needs more energy at idling.

For the BrainScaleS system, we were able to solve a reasonable amount of small 4×4 Sudokus. Scaling up to 6×6 was unproblematic, and only caused a very small ($< 1\%$) synapse loss during mapping process. Nevertheless, we were not able to find a suitable parameter set allowing to solve a reasonable number of Sudokus. Extended parameter space evaluation and increasing redundancy should reveal that it is possible to solve larger Sudokus on the BrainScaleS system.

`Architecture#3`, which was especially designed for the Spikey system, performs on similar level as the other

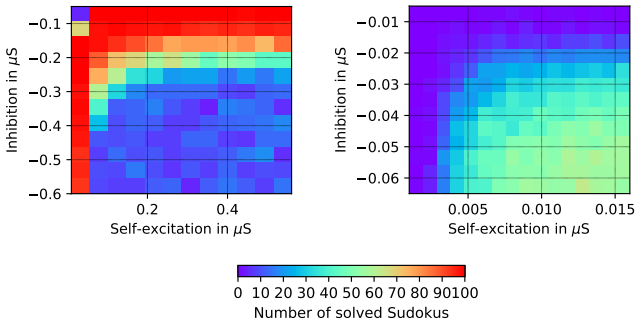

Fig. 5. Parameter sweep for `architecture#3` using the simulator (left) and the Spikey system (right). Both platforms differ in neuron parameters significantly, which is responsible for the different look of the parameter space.

TABLE I

WE EVALUATED 100 ASSORTED SUDOKUS FOR THREE ARCHITECTURES AND CALCULATED THE FIRST POINT IN SIMULATION TIME, WHERE NETWORK ACTIVITY SHOWED THE SOLUTION TO THE RESPECTIVE SUDOKU. HARDWARE TIME TO SOLUTION IS CALCULATED BY TAKING INTO ACCOUNT THE HARDWARE ACCELERATION/SLOW-DOWN FACTOR. † CALCULATED APPROXIMATE VALUE.

Platform	#Solved Sudokus	Bio-time to sol. in ms	Standard Deviation in ms	Real-time to sol. in s	Power in W	Energy to Solution in J
4×4 *Sudokus using* `architecture#1`						
NEST	100	214.6	263.1	0.03	45	1.4
SpiNN-5	97	357.1	688.9	3.57	23.3	83.2
BrainScaleS	86	3,241.9	4,573.1	$3.24 \cdot 10^{-4}$	NA	†0.0062
4×4 *Sudokus using* `architecture#2`						
NEST	100	214.6	263.1	0.03	45	1.4
SpiNN-3	99	241.2	250.0	2.41	2.7	6.5
4×4 *Sudokus using* `architecture#3`						
NEST	100	286.0	377.6	0.12	45	5.4
SpiNN-3	100	319.0	437.3	3.19	2.8	8.9
Spikey	75	3,745.8	6,041.1	$3.75 \cdot 10^{-4}$	5.6	0.0021
6×6 *Sudokus using* `architecture#1`						
NEST	98	1,769.2	1,909.1	0.62	45	27.9
SpiNN-5	99	2,084.8	2,703.3	20.85	23.5	490.0
6×6 *Sudokus using* `architecture#2`						
NEST	98	1,769.2	1,909.1	0.62	45	27.9
SpiNN-3	91	1,641.1	1,463.0	16.41	2.7	44.3
8×8 *Sudokus using* `architecture#2` *(current-based synapses)*						
NEST	87	10,893.8	8,524.	1.43	45	64.4
SpiNN-3	91	11,892.5	8,484.1	118.93	2.8	333.0
9×9 *Sudokus using* `architecture#2` *(current-based synapses)*						
NEST	88	17,725.2	15,602.6	4.87	45	219.2
SpiNN-3	81	12,402.6	11,618.7	124.02	2.8	347.3

implementations. On the Spikey system we were able to achieve a reasonable solving rate. For the remaining unsolved puzzles, device mismatch will lead to nearly defect and also to overly active populations, which deactivates or activates certain numbers, resulting in the reduced accuracy.

All in all, we can make the following conclusions from the results in Table I: the analogue platforms Spikey and BrainScaleS allow us to solve Sudokus highly efficient at the cost of a slightly reduced solving capability. The SpiNNaker system, which is slowed down in this set-up, runs less efficient than the simulator on a non-specialized CPU. The gap between both approaches narrows when simulating larger networks due to the fact that the NEST simulation will slow-down with increasing network size. Not all cores of the SpiNNaker system have been utilized in contrast to the NEST simulation, which on the other hand applies a more accurate integrator. However, a factor ten lies within the aforementioned slow-down of the system, which will be solved with future software releases. An additional factor is given by the more modern manufacturing process of the Intel CPU, which accounts for another jump in efficiency. Here, the next generation SpiNNaker 2 system will also yield a jump in efficiency of an order of magnitude [23]. Noticeable is also the rather large empirical standard deviation found in the bio-time to solution measures. This is related to the fact that the statistical distribution is not normal. This can be verified by observing the Violin-plots in Fig. 6. It reveals that most Sudokus can actually be solved within a small interval of the mean value, however, single outliers have unexpectedly long solving times. This justifies the approach of reduced simulation times for parameter space exploration, as most solutions are found within the first second of the simulation.

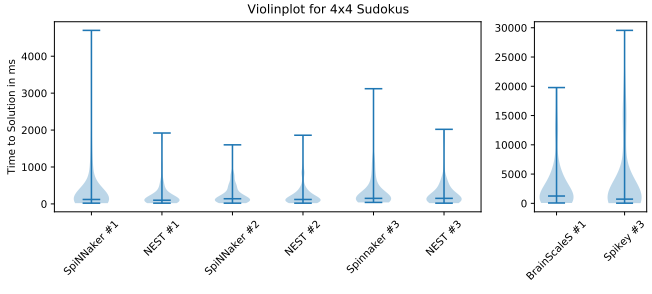

Fig. 6. Violin-plot for all 4x4 Sudokus. Analogue platforms are separated due to the higher maximal simulation time.

IV. Discussion and Outlook

In this contribution we have used Sudokus as a representative constraint satisfaction problem. Using a winner-takes-all architecture, we demonstrated that all neuromorphic target platforms were able to solve most of the given problem instances. This fact can be used to actually compare platforms in their efficiency of simulating/emulating spiking neural networks. The main performance indicator is the number of Sudokus that could be solved in a fixed simulation time window. Here, all platforms show their capability to solve most of the given puzzles. The second indicator, bio-time to solution, is however hardly comparable, as parameters for the BrainScaleS and Spikey systems differ from those of the digital simulators. Still, energy-to-solution is an important part of the comparison of platforms. We showed that the analogue systems are highly efficient in simulating these networks. Furthermore, the SpiNNaker system becomes more and more efficient with increasing network sizes. A future software update allowing to increase the accuracy of the simulation on SpiNNaker, while preserving the real-time capability the system is designed for, is an important step towards a highly efficient neuromorphic platform. This has been discussed with the hardware maintainers and is planned for a future release. On the BrainScaleS system, larger Sudokus were not solved satisfactorily. This is basically not a hardware issue, but more an issue of increasing redundancy and finding correct parameters. A pilot study showed that these networks map well onto the platform, and we expect to process larger Sudokus in the future.

The next possible step is to exploit adaptability in several directions: using adaptive integrate-and-fire neurons or using STDP might bring some benefits, so might abating input noise during simulation. Furthermore, as the algorithm allows direct comparison of all target platforms, it is suitable to be used as a benchmark for neuromorphic hardware platforms. Here, we continuously improve our approach, trying to extend the availability of network models to all target platforms, and to augment the list of target platforms as well.

Acknowledgment

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7) under grant agreement no 604102 and the EU's Horizon 2020 research and innovation programme under grant agreements No 720270 and 785907 (Human Brian Project, HBP). It has been further supported by the Cluster of Excellence Cognitive Interaction Technology "CITEC" (EXC 277) at Bielefeld University, which is funded by the German Research Foundation (DFG). Furthermore, we thank the Electronic Vision(s) group from Heidelberg University and Advanced Processor Technologies Research Group from Manchester University for access to their hardware systems and continuous support. The first implementation of the spiking Sudoku solver has been developed as part of a students team project at Bielefeld University.

References

[1] S. B. Furber *et al.*, "Overview of the SpiNNaker system architecture," *Computers, IEEE Transactions on*, vol. 62, no. 12, pp. 2454–2467, 2013.

[2] M. A. Petrovici *et al.*, "Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms," *PloS one*, vol. 9, no. 10, p. e108590, 2014.

[3] S. Schmitt *et al.*, "Neuromorphic Hardware In The Loop: Training a Deep Spiking Network on the BrainScaleS Wafer-Scale System," *arXiv preprint arXiv:1703.01909*, mar 2017. [Online]. Available: http://arxiv.org/abs/1703.01909

[4] T. Wunderlich *et al.*, "Demonstrating advantages of neuromorphic computation: A pilot study," *arXiv preprint arXiv:1811.03618*, 2018.

[5] S. J. van Albada *et al.*, "Performance comparison of the digital neuromorphic hardware spinnaker and the neural network simulation software nest for a full-scale cortical microcircuit model," *Frontiers in neuroscience*, vol. 12, p. 291, 2018.

[6] P. A. Bogdan, A. G. D. Rowley, O. Rhodes, and S. B. Furber, "Structural plasticity on the spinnaker many-core neuromorphic system," *Frontiers in Neuroscience*, vol. 12, p. 434, 2018.

[7] A. S. Cassidy *et al.*, "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 2013, pp. 1–10.

[8] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[9] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 1, pp. 106–122, 2018.

[10] A. Stöckel, C. Jenzen, M. Thies, and U. Rückert, "Binary associative memories as a benchmark for spiking neuromorphic hardware," *Frontiers in computational neuroscience*, vol. 11, p. 71, 2017.

[11] J. C. Knight and T. Nowotny, "GPUs Outperform Current HPC and Neuromorphic Solutions in Terms of Speed and Energy When Simulating a Highly-Connected Cortical Model," *Frontiers in Neuroscience*, vol. 12, no. December, pp. 1–19, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00941/full

[12] W. Maass, "Noise as a resource for computation and learning in networks of spiking neurons," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 860–880, 2014.

[13] G. A. Fonseca Guerra and S. B. Furber, "Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems," *Frontiers in neuroscience*, vol. 11, p. 714, 2017.

[14] J. Binas, G. Indiveri, and M. Pfeiffer, "Spiking analog vlsi neuron assemblies as constraint satisfaction problem solvers," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 2094–2097.

[15] R. Brette and W. Gerstner, "Adaptive exponential integrate-and-fire model as an effective description of neuronal activity," *Journal of Neurophysiology*, vol. 94, no. 5, pp. 3637–3642, 2005.

[16] A. Peyser *et al.*, "Nest 2.14.0," Oct. 2017. [Online]. Available: https://doi.org/10.5281/zenodo.882971

[17] J. Jordan *et al.*, "Extremely scalable spiking neuronal network simulation code: from laptops to exascale computers," *Frontiers in neuroinformatics*, vol. 12, p. 2, 2018.

[18] S. Furber, "Large-scale neuromorphic computing systems," *Journal of neural engineering*, vol. 13, no. 5, p. 051001, 2016.

[19] A. Davison *et al.*, "Pynn: a common interface for neuronal network simulators," 2009.

[20] T. Pfeil *et al.*, "Six networks on a universal neuromorphic computing substrate," *Frontiers in Neuroscience*, vol. 7, p. 11, 2013.

[21] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*. IEEE, 2008, pp. 431–438.

[22] W. Maass, "On the computational power of winner-take-all," *Neural computation*, vol. 12, no. 11, pp. 2519–2535, 2000.

[23] S. Hoeppner *et al.*, "Dynamic power management for neuromorphic many-core systems," *arXiv preprint arXiv:1903.08941*, 2019.