BrainScaleS-2 Software
Use Cases, Access and Integration into EBRAINS

Eric Müller
mueller@kip.uni-heidelberg.de

2022-03-31
NICE 2022
BrainScaleS-2

- Physical model, mixed-signal implementation
- AdEx neurons, short-term plasticity
- Structured neurons & nonlinear effects of dendrites
- Accelerated model dynamics ($\sim 10^3$)
- Support for online updates of neuron parameters, synapses (and network topology)
- Programmable plasticity
- Non-spiking operation mode (analog MAC)
BrainScaleS-2 Systems

- Setup types
  - “Lab” — local and remote usage
  - Mobile — embedded operation
  - Multi-chip / “Frankenstein Wafer”

  → (Network-attached) Accelerators

- Software stack providing varying abstraction levels
  - PyNN, hxtorch.snn, …
  - hardware abstraction layers
    (configuration and control)
  - communication

  → APIs for modeling, commissioning and development
Experiments? Configuration & “Protocol”

- Synapses, Neurons
- I/O (On-chip/off-chip)
- Observables, Controllables
- Controllers:
  - Host computer
  - FPGA
  - Embedded processors
Experiments? Configuration & “Protocol”

- Synapses, Neurons
- I/O (On-chip/off-chip)
- Observables, Controllables
- Controllers:
  - Host computer
  - FPGA
  - Embedded processors
Experiment Description → Initial Configuration
- Topology
  → Placement & Routing
- Cell Parameterization
  → Parameter Translation (Calibration)
- Plasticity Kernels

Experiment Description → Experiment Protocol
- Off-chip I/O (input/stimulus, output/recording)
- On-chip I/O (Poisson spike sources, …)
- Other dynamics (e.g., via embedded processors)
Experiment “Execution”

- Initial Configuration
- Execution of the ‘Experiment Protocol’
- Host-centric view here but multiple controllers do co-exist
Time Sharing — Experiment Scheduling

Regular scheduling via SLURM:
- Exp. 1
- Exp. 2
- Exp. 3

Hardware idle despite work

Hardware idle

(time)

Micro-Scheduling via quiggeldy:
- Exp. 1
- Exp. 2
- Exp. 3

(quiggeldy (hardware setup))

(time difference scheduling)

Experiment Setup Update/Analysis

Hardware Run

[Oliver Breitwieser (2021). Learning by Tooling: Novel Neuromorphic Learning Strategies in Reproducible Software Environments. Chapter II-10. Ph.D. thesis. Ruprecht-Karls-Universität Heidelberg]
Coordinator's View on Structured Neurons

```python
coord = halco.AtomicNeuronOnLogicalNeuron  # relative coordinate
def row = halco.NeuronRowOnLogicalNeuron    # 0, 1
column = halco.NeuronColumnOnLogicalNeuron  # 0, 1, ..., 127
morphology = lola.Morphology()

# create compartments: main branch
def morphology.create_compartment([coord(0, 0), coord(1, 0)])
def morphology.create_compartment([coord(2, 0), coord(3, 0), coord(3, 1)])

# create compartments: sub branches
for row_coord in [0, 1]:
    for column_coord in [4, 6]:
        morphology.create_compartment([coord(column_coord, row_coord), coord(column_coord + 1, row_coord)])

# enable conductance to shared line

morphology.connect_resistor_to_soma(coord(1, 0))
for row_coord in [0, 1]:
    for column_coord in [3, 5]:
        morphology.connect_resistor_to_soma(coord(column_coord, row_coord))

# direct connection to shared line
morphology.connect_to_soma(coord(2, 0))
for row_coord in [0, 1]:
    for column_coord in [4, 6]:
        morphology.connect_to_soma(coord(column_coord, row_coord))

# connect somatic shared line
morphology.connect_soma_line(start=column(1), end=column(2), row=row(0))
for row_coord in [row(0), row(1)]:
    morphology.connect_soma_line(column(3), column(4), row_coord)
morphology.connect_soma_line(column(5), column(6), row_coord)

neuron_coordinate, logical_neuron = morphology.done()
```

[Work by Raphael Stock and Jakob Kaiser (2021, 2022)]
class HomeostaticSynapse(pynn.PlasticityRule, 
    pynn.standardmodels.synapses.StaticSynapse):
    # ...
    def generate_kernel(self) -> str: 
        return textwrap.dedent(""
            // C++ ... 
            template <size_t N>
            void PLASTICITY_RULE_KERNEL(
                std::array<SynapseArrayViewHandle, N>& synapses,
                std::array<PPUOnDLS, N> synrams) {
                /* embedded processors have access to a set of 
                * observables and controllables ... */
            """.format(...)
        
        # ...
        synapse_type = HomeostaticSynapse(timer=timer, target=60, weight=0)
        pynn.Projection(pop_input, nrn, pynn.AllToAllConnector(), 
            synapse_type=synapse_type)
        # ...

[Work by Philipp Spilger (2021, 2022)]
Modeling with Hardware in the Loop

```python
from hxtorch import snn

class Model(torch.nn.Module):
    def __init__(...):
        # Create Instance
        instance = snn.Instance(mock=mock)
        # Add HXModules
        self.linear_h = snn.HXSynapse(
            in_features, out_features, instance=self.instance, ...
        )
        self.lif_h = snn.HXNeuron(
            hidden_size, instance=self.instance, ...
        )
        self.linear_o = snn.HXSynapse(
            hidden_size, output_size, instance=self.instance, ...
        )
        self.li_readout = snn.HXReadoutNeuron(
            output_size, instance=self.instance, ...
        )

    def forward(self, input):
        current_i = self.linear_h(input)
        spikes_h = self.lif_h(current_i)
        current_o = self.linear(spikes_h)
        membrane_out = self.li_readout(current_o)
        return membrane_out

    def __call__(self, *args, **kwargs):
        return self.forward(*args, **kwargs)

# Execute
model = Model(...)
inputs = snn.HXTensorHandle(spikes)
membrane = model(inputs)
```

- PyTorch-like description of SNNs
- Handles for tensors (i.e. not using XLA Tensors)
- Same API for software simulation & hardware emulation
- Maintains auto-differentiation functionality
- Flexibility in backward pass by assigning autograd functions to hardware operations
- Future: Integration into Norse?

---

PyTorch-like description of SNNs

Handles for tensors (i.e. not using XLA Tensors)

Same API for software simulation & hardware emulation

Maintains auto-differentiation functionality

Flexibility in backward pass by assigning autograd functions to hardware operations

Future: Integration into Norse?

[Work by Elias Arnold & Philipp Spilger (2022)]
Platform Access & Operation

- We leverage **EBRAINS** central services!
  - AAA, WebIDE hosting (JupyterLab), storage, quota/job reporting, …, user support
- Access to BrainScaleS via EBRAINS (+ SpiNNaker, as well as many other software packages)
- Dedicated BrainScaleS-2 Experiment Service for interactive experimenting (0(10 Hz), limited by specifics of the experiment and I/O)
Platform Access & Operation

- BSS-2 software now integrated into the EBRAINS Software Distribution...
- …enables a native and “natural” integration of BrainScaleS-2 into EBRAINS ‘Collabs’
  - We convinced EBRAINS to adopt spack as a package manager :o)
- Future: Deployments on EBRAINS HPC sites → multi-site workflows
Disclaimer

- Our software deployment on EBRAINS is somewhat ‘stable’… we expect more recent software in a couple of weeks (and more frequent releases afterwards).
- In addition, there will be a ‘testing’ deployment providing a continuous stream of newer software versions (approx. weekly).
- Many features presented here are still work in progress (MC neurons, programmable plasticity, SNN support in hxtorch), will require some more time to stabilize and materialize in a release.
Conclusion

- We work towards multiple goals:
  - Commissioning of recent BSS-2 hardware features, e.g., structured neurons and multi-chip systems
  - Programmable plasticity (code generation for the embedded processors)
  - Providing ML-friendly interfaces
  - Efficiency (fast reconfiguration) in high-level use cases
  - Parameter Translation (SI hardware & bio units) and integration of ‘Calibration’
  - We continue to improve system robustness
  ⇒ Transition towards a flatter learning curve for users (deployment, operation & usage)

- Executable Documentation incl. Examples
- Now: BrainScaleS-2 interactive tutorial → PyNN.brainscales2
  - Link to ‘Collab’ should have been sent via mail
  - https://wiki.ebrains.eu/bin/view/Collabs/ncm-test-SOMEUSERNAME/
A scalable approach to modeling on accelerated neuromorphic hardware

Eric Müller *†, Elias Arnold †, Oliver Breitwieser †, Milena Czierlinski †, Arne Emmel †, Jakob Kaiser †, Christian Mauch †, Sebastian Schmitt ‡, Philipp Spilger †, Raphael Stock †, Yannik Stradmann †, Johannes Weis †, Andreas Baumbach 1, Sebastian Billaudelle 1, Benjamin Cramer 1, Falk Ebert 1, Julian Göltz 1, Joscha Ilmberger 1, Vitali Karasenko 1, Mitja Kleider 1, Aron Leibfried 1, Christian Pehle 1, Johannes Schemmel 1

1Electronic Vision(s), Kirchhoff-Institute for Physics, Heidelberg University, Germany
2Third Institute of Physics, University of Göttingen, Germany

Preprint available on arXiv: https://arxiv.org/abs/2203.11102
BrainScaleS