The GeNN ecosystem

James Knight and Thomas Nowotny
J.C.Knight@sussex.ac.uk

GeNN

- Cross-platform C++ library for generating optimised CUDA code for GPU accelerated SNN simulations.
- Hopefully you learnt all about it in Thomas’s talk!

Installation

CUDA on Linux

- Each version of CUDA only supports a subset of GCC versions so if you have a very old or very bleeding edge OS you may need to install an additional version of GCC.
- Installing CUDA via the NVIDIA proprietary packages tends to work best if your OS is supported.

CUDA on Windows

- CUDA is nicely integrated into Visual Studio and provided graphical debugging and profiling tools
- Because Visual Studio is updated (annoyingly) frequently, compiler/CUDA version issues are more prevalent than on Linux
- If installing from scratch we recommend:
  - CUDA 9.2.148
  - Visual Studio 2017 15.6.7
- There are performance issues with CUDA on Windows display devices

https://docs.microsoft.com/en-us/visualstudio/productinfo/installing-an-earlier-release-of-vs2017

CUDA on Mac

- Sadly Apple hasn't built any machines with NVIDIA GPUs since 2014
- However, if you're lucky enough to have:
  - MacBook Pro (Retina, 15-inch, Late 2013)
  - MacBook Pro (Retina, 15-inch, Mid 2014)
  - Equivalent iMac models (probably not with you!)
- You **may** have a NVIDIA GPU that's usable with the current version of CUDA!

GeNN

- Clone genn_4 branch of GeNN from https://github.com/genn-team/genn
- If you have CUDA installed, set the `CUDA_PATH` environment variable to its location
- Add `$GENN_PATH/bin` to your path or, on Linux/Mac, make install)
- Checkout tutorials from https://github.com/neworderofjamie/new_genn_tutorials

Test your GeNN installation

Navigate to `va_benchmark` folder and run code generator:

Linux/Mac with CUDA:

```
genn-buildmodel.sh model.cc
```

Linux/Mac without CUDA:

```
genn-buildmodel.sh -c model.cc
```

Windows with CUDA:

```
genn-buildmodel.bat model.cc
```

Windows without CUDA:

```
genn-buildmodel.bat -c model.cc
```

Build generated code:

Linux/Mac:

```
make
```

Windows (or double click solution file):

```
msbuild va_benchmark.sln /t:va_benchmark /p:Configuration=Release
```

Run model and plot results:

Linux/Mac:

```
./va_benchmark

python plot_spikes.py
```

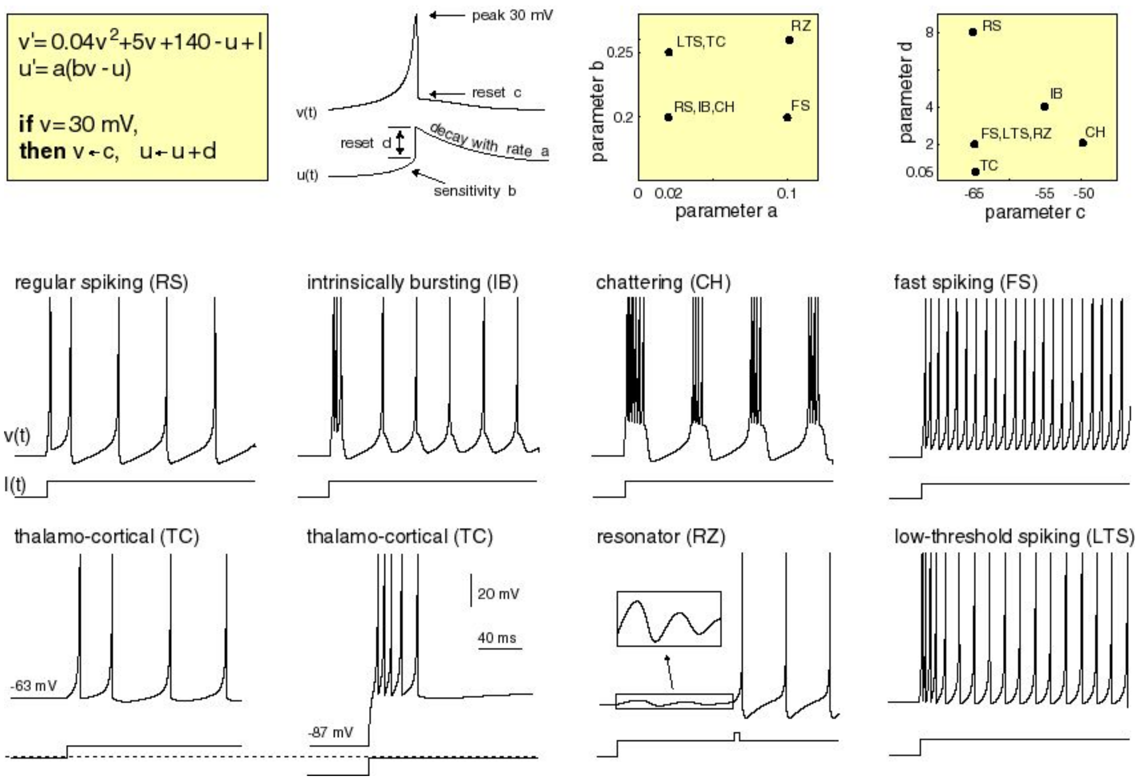
Windows:

```
va_benchmark_Release

python plot_spikes.py
```

Tutorial 1: Neurons

Tutorial 1: Introduction

- Basics of using GeNN
- Explore the dynamics of the Izhikevich neuron model
- Simple recording



```cpp
#include "modelSpec.h"
void modelDefinition(NNmodel &model)
{
}
```

Include GeNN definitions

Function gets linked against code generator

```
#include "modelSpec.h"
void modelDefinition(NNmodel &model)
{
    model.setDT(0.1);
    model.setName("tutorial1");
}
```

Simulation time step [ms]

Model name - mostly used as the name of the code generator output directory

```cpp
#include "modelSpec.h"
void modelDefinition(NNmodel &model)
{
    model.setDT(0.1);
    model.setName("tutorial1");

    NeuronModels::IzhikevichVariable::ParamValues paramValues;
    NeuronModels::IzhikevichVariable::VarValues initValues(
        -65.0,              // 0 - V
        -20.0,              // 1 - U
        uninitialisedVar(), // 2 - A
        uninitialisedVar(), // 3 - B
        uninitialisedVar(), // 4 - C
        uninitialisedVar()); // 5 - D
}
```

Parameters are constant across population and do not change during simulation (this model doesn't have any)

Var values specify how model's state variables are initialised - in this case we want to manually set A, B, C and D on a per-neuron basis

```cpp
#include "modelSpec.h"
void modelDefinition(NNmodel &model)
{
    model.setDT(0.1);
    model.setName("tutorial1");

    NeuronModels::IzhikevichVariable::ParamValues paramValues;
    NeuronModels::IzhikevichVariable::VarValues initValues(
        -65.0,                  // 0 - V
        -20.0,                  // 1 - U
        uninitialisedVar(),     // 2 - A
        uninitialisedVar(),     // 3 - B
        uninitialisedVar(),     // 4 - C
        uninitialisedVar());    // 5 - D
    model.addNeuronPopulation<NeuronModels::IzhikevichVariable>("Neurons", 4,
                                                                paramValues, initValues);
}
```

Adds a population called "Neurons" consisting of 4 Izhikevich neurons (1 for each regime) with these parameters and initial state to network

```
#include "modelSpec.h"
void modelDefinition(NNmodel &model)
{
    model.setDT(0.1);
    model.setName("tutorial1");

    NeuronModels::IzhikevichVariable::ParamValues paramValues;
    NeuronModels::IzhikevichVariable::VarValues initValues(
        -65.0,                  // 0 - V
        -20.0,                  // 1 - U
        uninitialisedVar(),     // 2 - A
        uninitialisedVar(),     // 3 - B
        uninitialisedVar(),     // 4 - C
        uninitialisedVar());    // 5 - D
    model.addNeuronPopulation<NeuronModels::IzhikevichVariable>("Neur
                                                                param

    CurrentSourceModels::DC::ParamValues currentSourceParamVals(
        10.0);  // 0 - magnitude
    model.addCurrentSource<CurrentSourceModels::DC>("CurrentSource", "Neurons",
                                                    currentSourceParamVals, {});
}
```

Define DC current source parameters (magnitude)

Attach a DC current source called “CurrentSource” to our neuron population with these parameters

Tutorial 1: Generate model code

Linux/Mac with CUDA:

```
genn-buildmodel.sh model.cc
```

Linux/Mac without CUDA:

```
genn-buildmodel.sh -c model.cc
```

Windows with CUDA:

```
genn-buildmodel.bat model.cc
```

Windows without CUDA:

```
genn-buildmodel.bat -c model.cc
```

```
#include "tutorial1_CODE/definitions.h"
int main()
{
    allocateMem();
    initialize();
    return EXIT_SUCCESS;
}
```

Include generated code

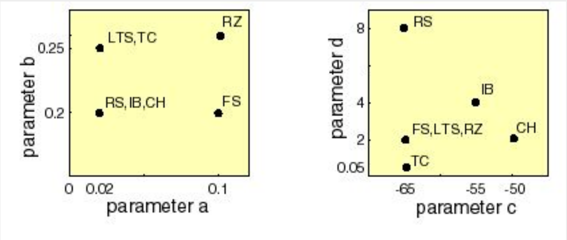
Call generated function to make necessary memory allocations

Call generated function to perform first stage of initialisation (V and U will be initialised here)

```
#include "tutorial1_CODE/definitions.h"
int main()
{
    allocateMem();
    initialize();
    aNeurons[0] = 0.02; bNeurons[0] = 0.2;  cNeurons[0] = -65.0;    dNeurons[0] = 8.0;  // RS
    aNeurons[1] = 0.1;  bNeurons[1] = 0.2;  cNeurons[1] = -65.0;    dNeurons[1] = 2.0;  // FS
    aNeurons[2] = 0.02; bNeurons[2] = 0.2;  cNeurons[2] = -50.0;    dNeurons[2] = 2.0;  // CH
    aNeurons[3] = 0.02; bNeurons[3] = 0.2;  cNeurons[3] = -55.0;    dNeurons[3] = 4.0;  // IB
    initializeSparse();
    return EXIT_SUCCESS;
}
```

Call generated function to perform second stage of initialisation (will upload initial state to GPU)

Manually initialise each neuron's state variables to match different regimes



```
#include "tutorial1_CODE/definitions.h"
int main()
{
    allocateMem();
    initialize();
    aNeurons[0] = 0.02; bNeurons[0] = 0.2;  cNeurons[0] = -65.0;    dNeurons[0] = 8.0;  // RS
    aNeurons[1] = 0.1;  bNeurons[1] = 0.2;  cNeurons[1] = -65.0;    dNeurons[1] = 2.0;  // FS
    aNeurons[2] = 0.02; bNeurons[2] = 0.2;  cNeurons[2] = -50.0;    dNeurons[2] = 2.0;  // CH
    aNeurons[3] = 0.02; bNeurons[3] = 0.2;  cNeurons[3] = -55.0;    dNeurons[3] = 4.0;  // IB
    initializeSparse();

    while(t < 200.0f) {
        stepTime();
    }

    return EXIT_SUCCESS;
}
```

Loop until 200ms of simulated time has elapsed (t is provided by GeNN)

Call generated functions to advance simulation state

```
#include "tutorial1_CODE/definitions.h"
#include <fstream>
int main()
{
    allocateMem();
    initialize();
    aNeurons[0] = 0.02; bNeurons[0] = 0.2;  cNeurons[0] = -65.0;    dNeurons[0] = 8.0;  // RS
    aNeurons[1] = 0.1;  bNeurons[1] = 0.2;  cNeurons[1] = -65.0;    dNeurons[1] = 2.0;  // FS
    aNeurons[2] = 0.02; bNeurons[2] = 0.2;  cNeurons[2] = -50.0;    dNeurons[2] = 2.0;  // CH
    aNeurons[3] = 0.02; bNeurons[3] = 0.2;  cNeurons[3] = -55.0;    dNeurons[3] = 4.0;  // IB
    initializeSparse();

    std::ofstream stream("state.csv");
    while(t < 200.0f) {
        stepTime();
        pullNeuronsStateFromDevice();

        stream << t << "," << VNeurons[0] << "," << VNeurons[1];
        stream << "," << VNeurons[2] << "," << VNeurons[3] << std::endl;
    }

    return EXIT_SUCCESS;
}
```

Include standard header for file IO

Open CSV file for writing

Download the state of “Neurons” population from GPU

Write current time and membrane voltage of each of the 4 neurons to CSV file

Tutorial 1: Building on Linux/Mac

Create Makefile

```
genn-create-user-project.sh tutorial1 simulator.cc
```

Build using:

```
make
```

Then run with: `./tutorial1`

Tutorial 1: Building on Windows

Create MSBuild projects

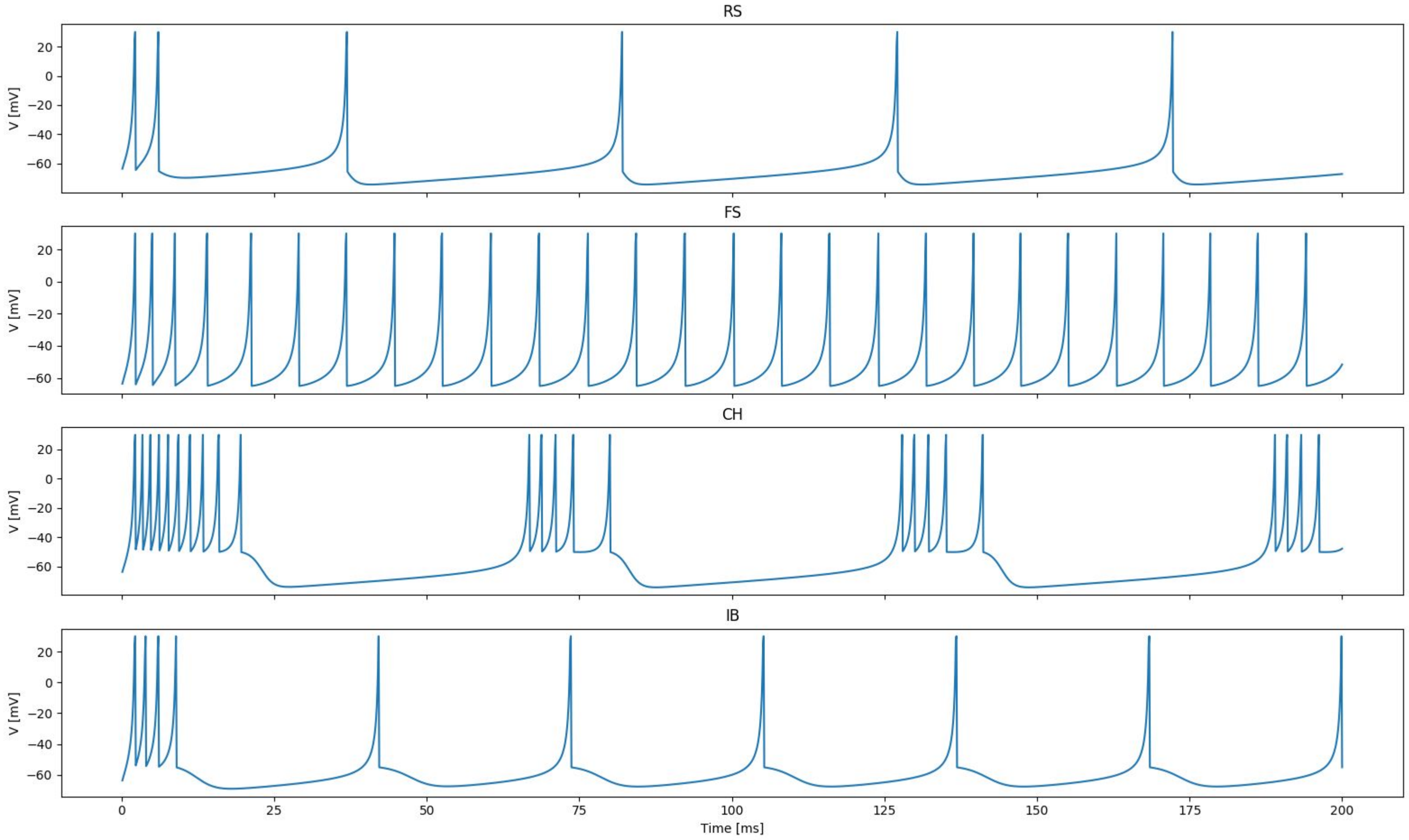
```
genn-create-user-project.bat tutorial1 simulator.cc
```

Build using (or double click solution file):

```
msbuild tutorial1.sln /t:tutorial1
/p:Configuration=Release
```

Then run with: `tutorial1_Release.exe`

Tutorial 1: Results

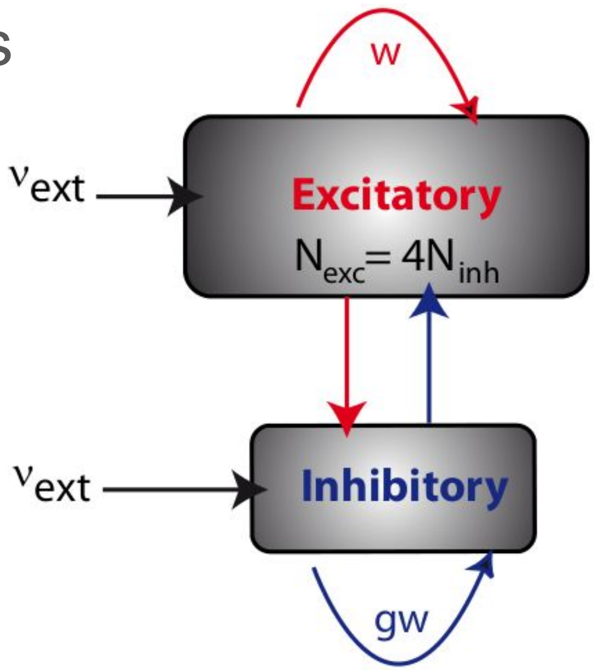


Tutorial 1: Exercises

- Expand the example to include more of the regimes from the diagram
- Look in “tutorial1_CODE” at neuronUpdate.cc and see what GeNN is doing!

Tutorial 2: Building networks

- Defining synapse populations
- Using initialisation snippets to configure parameters and connectivity
- Recording spikes



```cpp
#include "modelSpec.h"
void modelDefinition(NNmodel &model)
{
    model.setDT(1.0);
    model.setName("tutorial2");
}
```

Hopefully this is now all familiar

```
#include "modelSpec.h"
void modelDefinition(NNmodel &model)
{
    model.setDT(1.0);
    model.setName("tutorial2");

    NeuronModels::Izhikevich::ParamValues izkParams(
        0.02,   // 0 - A
        0.2,    // 1 - B
        -65.0,  // 2 - C
        8.0);   // 3 - D
    InitVarSnippet::Uniform::ParamValues uDist(
        0.0,    // 0 - min
        20.0);  // 1 - max
    NeuronModels::Izhikevich::VarValues ikzInit(
        -65.0,                                 // 0 - V
        initVar<InitVarSnippet::Uniform>(uDist));  // 1 - U
    model.addNeuronPopulation<NeuronModels::Izhikevich>("Exc", 8000, izkParams, ikzInit);
    model.addNeuronPopulation<NeuronModels::Izhikevich>("Inh", 2000, izkParams, ikzInit);

    CurrentSourceModels::DC::ParamValues currentSourceParamVals(4.0);  // 0 - magnitude
    model.addCurrentSource<CurrentSourceModels::DC>("ExcStim", "Exc", currentSourceParamVals, {});
    model.addCurrentSource<CurrentSourceModels::DC>("InhStim", "Inh", currentSourceParamVals, {});
}
```

Configure our Izhikevich neurons into the RS regime

Define a uniform random **distribution**

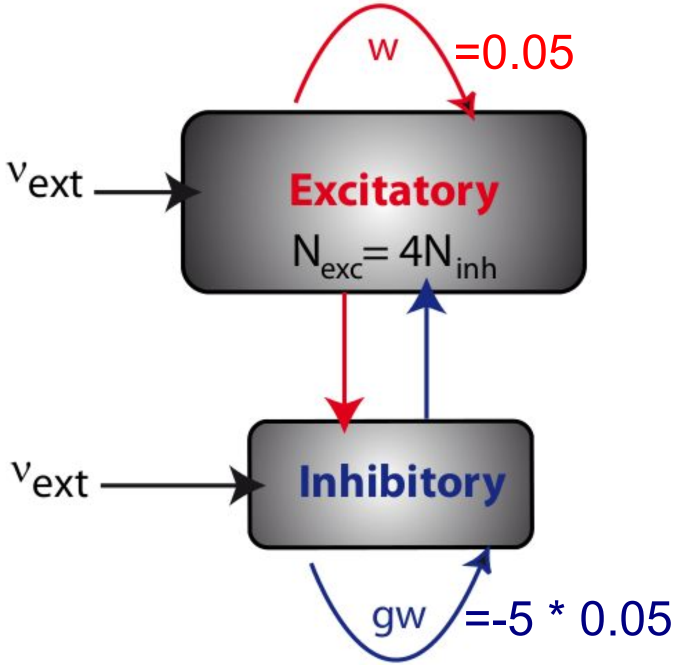
Use distribution to initialise u

Add neuron populations and current sources as before

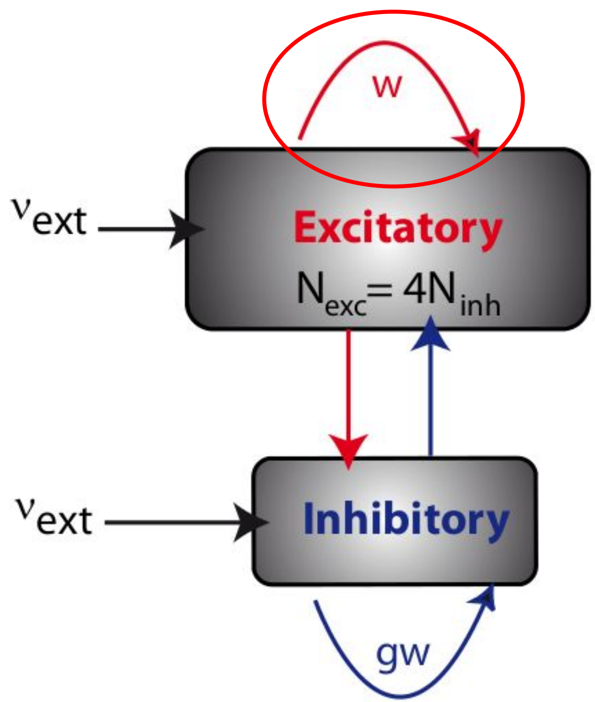
```
    WeightUpdateModels::StaticPulse::VarValues excSynInitValues(0.05);
    WeightUpdateModels::StaticPulse::VarValues inhSynInitValues(-5 * 0.05);

    InitSparseConnectivitySnippet::FixedProbability::ParamValues fixedProb(0.1); // 0 - prob
}
```

Set strength of connections

Configure parameters for connectivity initialisation.



```
    WeightUpdateModels::StaticPulse::VarValues excSynInitValues(0.05);
    WeightUpdateModels::StaticPulse::VarValues inhSynInitValues(-5 * 0.05);

    InitSparseConnectivitySnippet::FixedProbability::ParamValues fixedProb(0.1); // 0 - prob

    model.addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::DeltaCurr>(
        "Exc_Exc", SynapseMatrixType::SPARSE_GLOBALG, NO_DELAY,
        "Exc", "Exc", {}, excSynInitValues, {}, {},
        initConnectivity<InitSparseConnectivitySnippet::FixedProbabilityNoAutapse>(fixedProb));
}
```



```
    WeightUpdateModels::StaticPulse::VarValues excSynInitValues(0.05);
    WeightUpdateModels::StaticPulse::VarValues inhSynInitValues(-5 * 0.05);

    InitSparseConnectivitySnippet::FixedProbability::ParamValues fixedProb(0.1); // 0 - prob

    model.addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::DeltaCurr>(
        "Exc_Exc", SynapseMatrixType::SPARSE_GLOBALG, NO_DELAY,
        "Exc", "Exc", {}, excSynInitValues, {}, {},
        initConnectivity<InitSparseConnectivitySnippet::FixedProbabilityNoAutapse>(fixedProb));
}
```

Simplest **weight update model** - no learning etc

Simplest **postsynaptic model** - no ‘shaping’ of input current

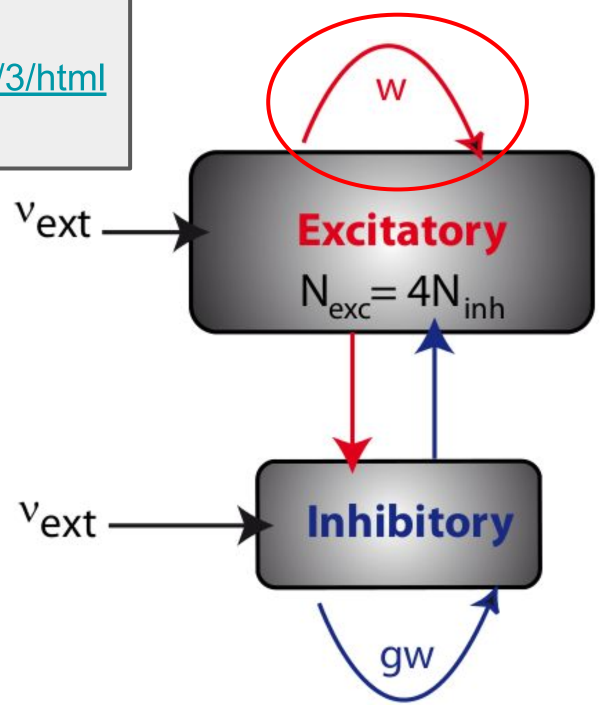
```
    WeightUpdateModels::StaticPulse::VarValues excSynInitValues(0.05);
    WeightUpdateModels::StaticPulse::VarValues inhSynInitValues(-5 * 0.05);

    InitSparseConnectivitySnippet::FixedProbability::ParamValues fixedProb(0.1); // 0 - prob

    model.addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::DeltaCurr>(
        "Exc_Exc", SynapseMatrixType::SPARSE_GLOBALG, NO_DELAY,
        "Exc", "Exc", {}, excSynInitValues, {}, {},
        initConnectivity<InitSparseConnectivitySnippet::FixedProbabilityNoAutapse>(fixedProb));
}
```

No synaptic delays

Name of synapse population

Sparse matrix with the same parameters for each synapse

http://genn-team.github.io/genn/documentation/3/html/subsect34.html



```
    WeightUpdateModels::StaticPulse::VarValues excSynInitValues(0.05);
    WeightUpdateModels::StaticPulse::VarValues inhSynInitValues(-5 * 0.05);

    InitSparseConnectivitySnippet::FixedProbability::ParamValues fixedProb(0.1); // 0 - prob

    model.addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::DeltaCurr>(
        "Exc_Exc", SynapseMatrixType::SPARSE_GLOBALG, NO_DELAY,
        "Exc", "Exc", {}, excSynInitValues, {}, {},
        initConnectivity<InitSparseConnectivitySnippet::FixedProbabilityNoAutapse>(fixedProb));
}
```

Name of **source** population

Name of **target** population

Weight update model parameters and initial state

Postsynaptic model parameters and initial state (DeltaCurr has none)

```
    WeightUpdateModels::StaticPulse::VarValues excSynInitValues(0.05);
    WeightUpdateModels::StaticPulse::VarValues inhSynInitValues(-5 * 0.05);

    InitSparseConnectivitySnippet::FixedProbability::ParamValues fixedProb(0.1); // 0 - prob

    model.addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::DeltaCurr>(
        "Exc_Exc", SynapseMatrixType::SPARSE_GLOBALG, NO_DELAY,
        "Exc", "Exc", {}, excSynInitValues, {}, {},
        initConnectivity<InitSparseConnectivitySnippet::FixedProbabilityNoAutapse>(fixedProb));
}
```

Use parameters to initialise connectivity of synapse population

```
    WeightUpdateModels::StaticPulse::VarValues excSynInitValues(0.05);
    WeightUpdateModels::StaticPulse::VarValues inhSynInitValues(-5 * 0.05);

    InitSparseConnectivitySnippet::FixedProbability::ParamValues fixedProb(0.1); // 0 - prob

    model.addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::DeltaCurr>(
        "Exc_Exc", SynapseMatrixType::SPARSE_GLOBALG, NO_DELAY,
        "Exc", "Exc", {}, excSynInitValues, {}, {},
        initConnectivity<InitSparseConnectivitySnippet::FixedProbabilityNoAutapse>(fixedProb));
    model.addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::DeltaCurr>(
        "Exc_Inh", SynapseMatrixType::SPARSE_GLOBALG, NO_DELAY,
        "Exc", "Inh", {}, excSynInitValues, {}, {},
        initConnectivity<InitSparseConnectivitySnippet::FixedProbability>(fixedProb));
    model.addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::DeltaCurr>(
        "Inh_Inh", SynapseMatrixType::SPARSE_GLOBALG, NO_DELAY,
        "Inh", "Inh", {}, inhSynInitValues, {}, {},
        initConnectivity<InitSparseConnectivitySnippet::FixedProbabilityNoAutapse>(fixedProb));
    model.addSynapsePopulation<WeightUpdateModels::StaticPulse, PostsynapticModels::DeltaCurr>(
        "Inh_Exc", SynapseMatrixType::SPARSE_GLOBALG, NO_DELAY,
        "Inh", "Exc", {}, inhSynInitValues, {}, {},
        initConnectivity<InitSparseConnectivitySnippet::FixedProbability>(fixedProb));
}
```

Rinse and repeat for the other three connections

Tutorial 2: Generate model code

Linux/Mac with CUDA:

```
genn-buildmodel.sh model.cc
```

Linux/Mac without CUDA:

```
genn-buildmodel.sh -c model.cc
```

Windows with CUDA:

```
genn-buildmodel.bat model.cc
```

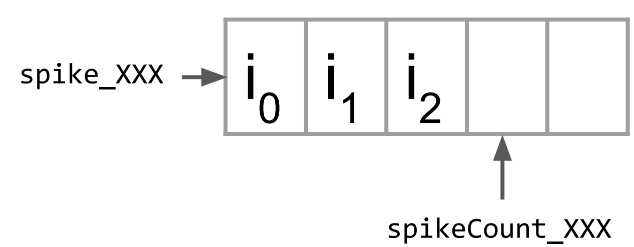
Windows without CUDA:

```
genn-buildmodel.bat -c model.cc
```

```cpp
#include "tutorial2_CODE/definitions.h"
#include <fstream>
#include <iostream>

int main()
{
    allocateMem();
    std::cout << "Initialising" << std::endl;
    initialize();
    initializeSparse();

    std::cout << "Simulating" << std::endl;
    std::ofstream stream("spikes.csv");
    while(t < 1000.0f) {
        stepTime();
        pullExcCurrentSpikesFromDevice();
        pullInhCurrentSpikesFromDevice();

        for(unsigned int i = 0; i < spikeCount_Exc; i++) {
            stream << t << ", " << spike_Exc[i] << std::endl;
        }
        for(unsigned int i = 0; i < spikeCount_Inh; i++) {
            stream << t << ", " << 8000 + spike_Inh[i] << std::endl;
        }
    }
    return EXIT_SUCCESS;
}
```

Open CSV file for writing

Download the spikes emitted by the "Exc" and "Inh" populations this timestep from GPU



Tutorial 2: Building on Linux/Mac

Create Makefile

```
genn-create-user-project.sh tutorial2 simulator.cc
```

Build using:

```
make
```

Then run with: `./tutorial2`

Tutorial 2: Building on Windows

Create MSBuild projects

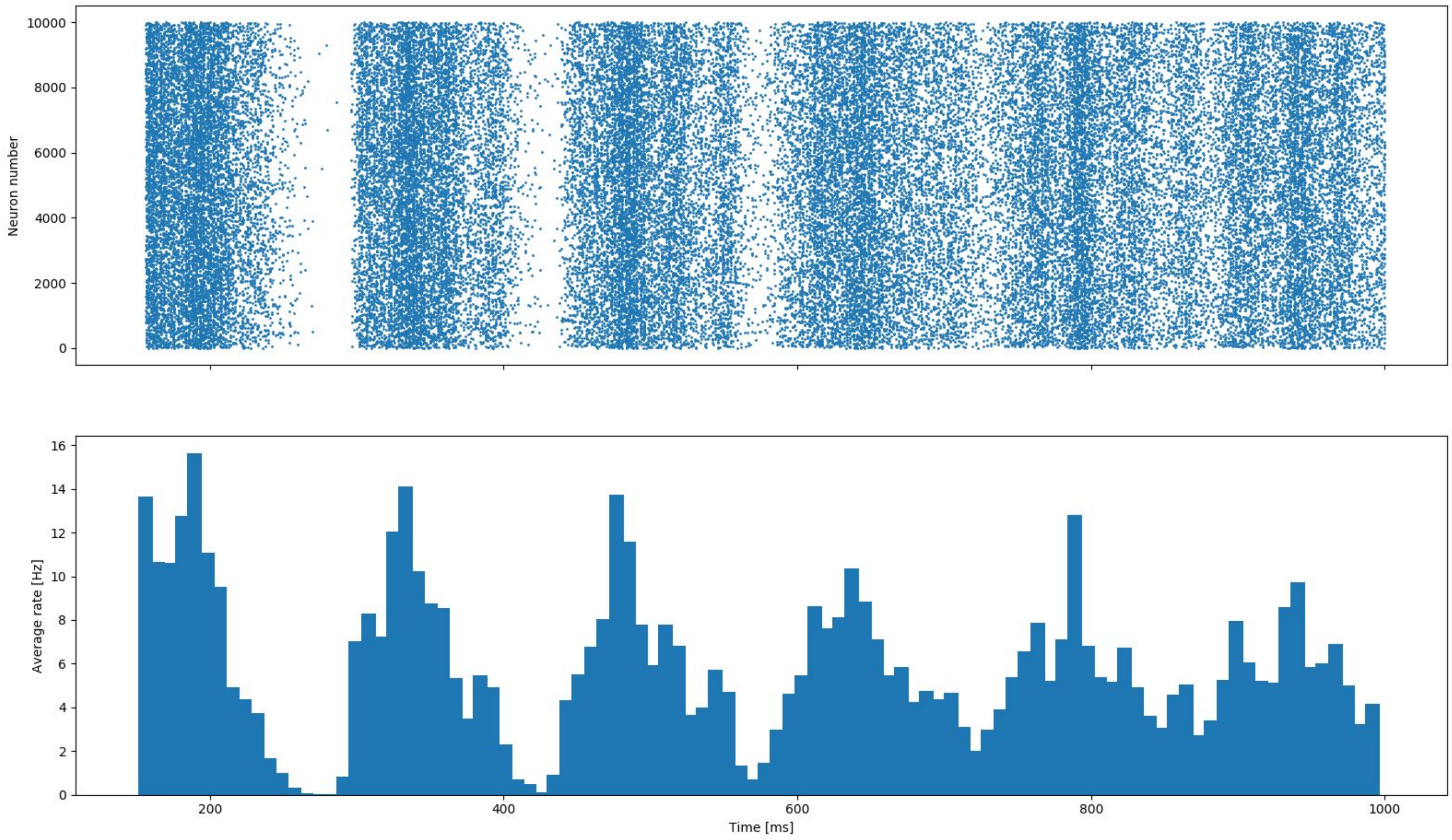
```
genn-create-user-project.bat tutorial2 simulator.cc
```

Build using (or double click solution file):

```
msbuild tutorial2.sln /t:tutorial2
/p:Configuration=Release
```

Then run with: `tutorial2_Release.exe`

Tutorial 2: Results



Tutorial 2: Exercises

- Experiment with what parameters you can change to scale the network while keeping it in the same regime.
- Look in “tutorial2_CODE” at synapseUpdate.cc and see what GeNN is doing!

Brian 2 frontend

```python
from brian2 import *
import brian2genn
set_device('genn')

n = 1000
duration = 1*second
tau = 10*ms
eqs = '''
dv/dt = (v0 - v) / tau : volt (unless refractory)
v0 : volt
'''
group = NeuronGroup(n, eqs, threshold='v>10*mV', reset='v=0*mV', refractory=5*ms, method='exact')
group.v = 0*mV
group.v0 = '20*mV * i / (n-1)'
monitor = SpikeMonitor(group)

run(duration)
```

- Probably simplest way of using GeNN
- For installation instructions talk to Thomas or see: https://brian2genn.readthedocs.io/en/latest/introduction/index.html#installing-the-brian2genn-interface

build unstable

Google Summer of Code

Python interface and PyNN frontend

Installation with CUDA

1. Make sure you have swig installed
2. From GeNN directory, build as a dynamic library using:
   `make -f lib/GNUMakefileLibGeNN DYNAMIC=1 LIBGENN_PATH=pygenn/genn_wrapper/`
3. On Mac OS X, set your newly created library's name with
   `install_name_tool -id "@loader_path/libgenn_DYNAMIC.dylib" pygenn/genn_wrapper/libgenn_DYNAMIC.dylib`
4. Install python module with setuptools using:
   `python setup.py develop`

Installation without CUDA

1. Make sure you have swig installed
2. From GeNN directory, build as a dynamic library using:
   `make -f lib/GNUMakefileLibGeNN CPU_ONLY=1 DYNAMIC=1 LIBGENN_PATH=pygenn/genn_wrapper/`
3. On Mac OS X, set your newly created library's name with
   `install_name_tool -id "@loader_path/libgenn_CPU_ONLY_DYNAMIC.dylib" pygenn/genn_wrapper/libgenn_CPU_ONLY_DYNAMIC.dylib`
4. Install python module with setuptools using:
   `python setup.py develop`

```python
import numpy as np
import matplotlib.pyplot as plt
from pygenn import genn_wrapper, genn_model

model = genn_model.GeNNModel("float", "tutorial1_pygenn")
model.dT = 0.1

izk_init = {"V": -65.0, "U": -20.0,
            "a": [0.02,     0.1,    0.02,   0.02], "b": [0.2,      0.2,    0.2,    0.2],
            "c": [-65.0,    -65.0,  -50.0,  -55.0], "d": [8.0,      2.0,    2.0,    4.0]}
pop = model.add_neuron_population("Neurons", 4, "IzhikevichVariable", {}, izk_init)
model.add_current_source("CurrentSource", "DC", "Neurons", {"amp": 10.0}, {})

model.build()
model.load()

voltage_view = pop.vars["V"].view
v = None
while model.t < 200.0:
    model.step_time()
    model.pull_state_from_device("Neurons")
    v = (np.copy(voltage_view) if v is None else np.vstack((v, voltage_view)))

figure, axes = plt.subplots(4, sharex=True)
for i, t in enumerate(["RS", "FS", "CH", "IB"]):
    axes[i].plot(np.arange(0.0, 200.0, 0.1), v[:,i])
plt.show()
```

More Pythonic parameters

Numpy views for efficient io

PyNN 🌲

PyNN is a simulator-independent language for building neuronal network models in Python

Installation

Once you have managed to install PyGeNN, this should be easy

1. Clone master branch of PyNN-GeNN from https://github.com/genn-team/pynn_genn
2. Install with setuptools using `python setup.py develop`

Thank you

J.C.Knight@sussex.ac.uk