GPU enhanced neuronal networks (GeNN)

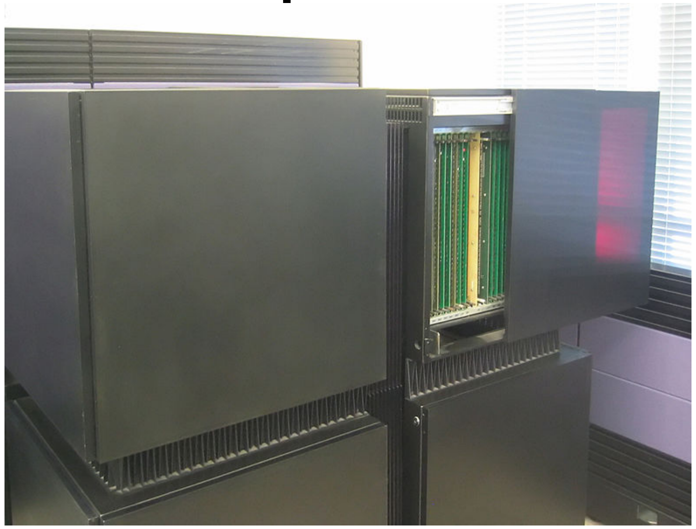
BRANDY School
Val di Sole

Prof. Thomas Nowotny (@drtnowotny)
CCNR and Sussex Neuroscience, School of Engineering and Informatics

UNIVERSITY OF SUSSEX

Introduction

- Parallel computers go as far back as the 50s and early 60s.
- The “connection machine” had up to 65536 processors (CM-1,2)



Thinking Machines CM-1 at the Computer History Museum in Mountain View. (from Wikipedia)

Introduction

- The desktop revolution however occurred with serial CPUs (and PCs remain largely serial to date)
- Fuelled by the games industry, graphics adaptors have become very powerful and are now using massively parallel graphical processing units (GPUs)

NVIDIA® GRID K2, 2x Kepler GPUs, 3072 cores total, 8GB GDDR5 memory.
(Image from NVIDIA)

The beginnings of GPGPU

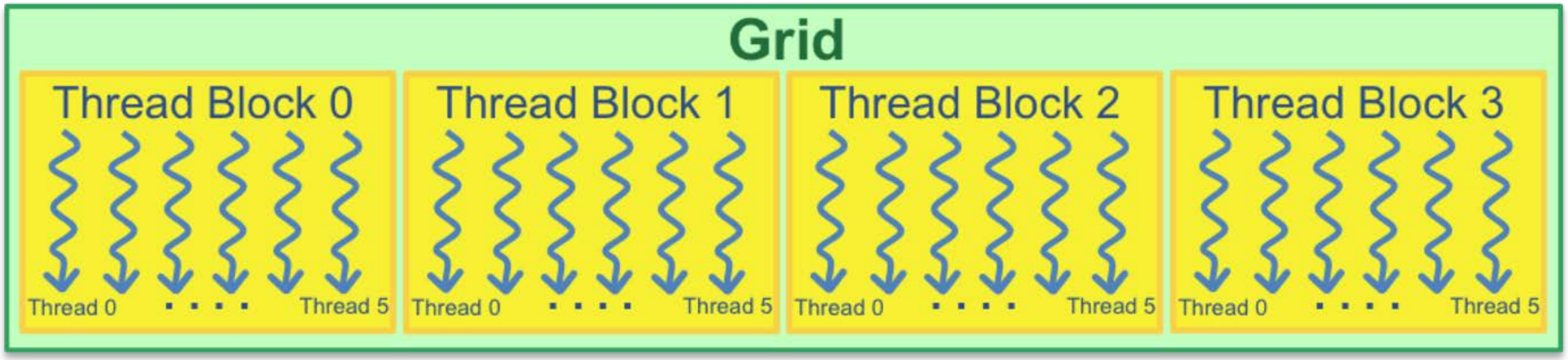
- Performance seekers started to (mis)use GPUs for general purpose computation:
- First examples around 2004, e.g.
  - neural networks
  - surface waves
- Using Cg shader language (OpenGL) or HLSL (Direct X)
- Impressive speedups of 20x and more over CPU based simulations

CUDA

The real Game-Changer: Nvidia® CUDA™

- CUDA™= "Common Unified Device Architecture"
- It was introduced by NVIDIA® to allow main stream developers to use massively parallel graphics chips for GPGPU **without the restrictions of shaders**
- The first CUDA SDK was released Feb 2007 (according to Wikipedia)
- CUDA™ is supported on all newer NVIDIA cards
- More general alternative: OpenCL

SIMT programming

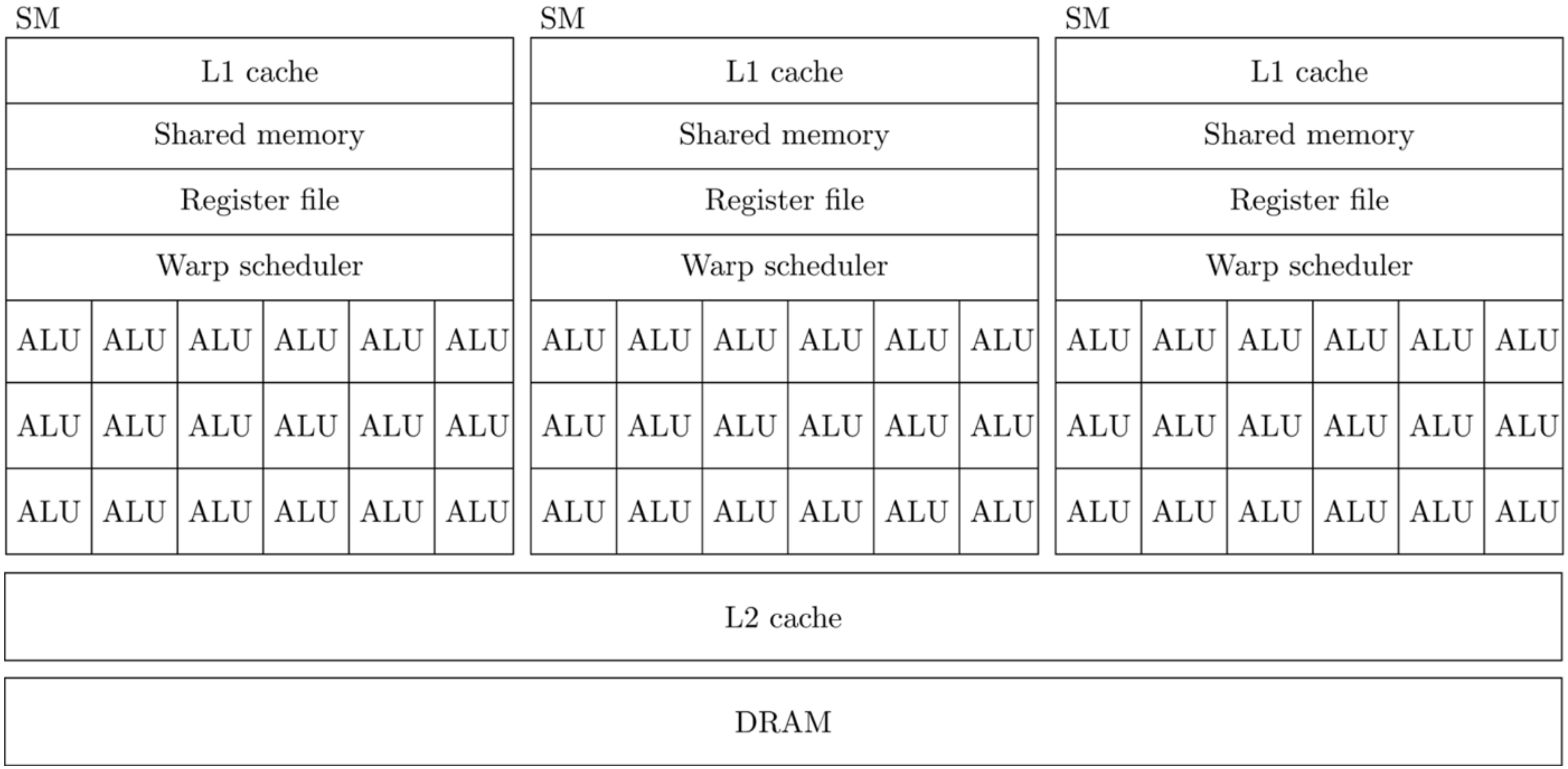


```
__global__ void saxpy(int N, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x+ threadIdx.x;
    if(i<N) {
        y[i] = a*x[i] + y[i];
    }
}
```

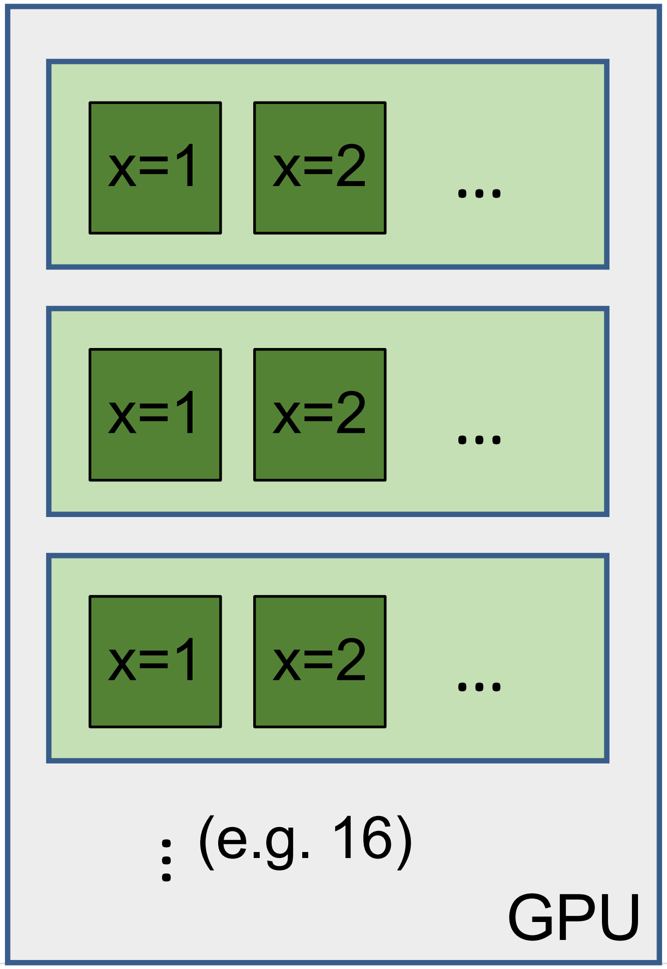
The CUDA^{TM} API

- Each thread executes what is called a “kernel”, defined by a C-like function
- This is a Single-Instruction-Multiple-Data (SIMD) environment
- Kernels within a blocks can share memory and can be synchronised
- Different blocks may execute in parallel or consecutively
- Maximal block sizes typically 512 (1024) threads
- Grid sizes up to 65535 x 65535 blocks

GPU architecture

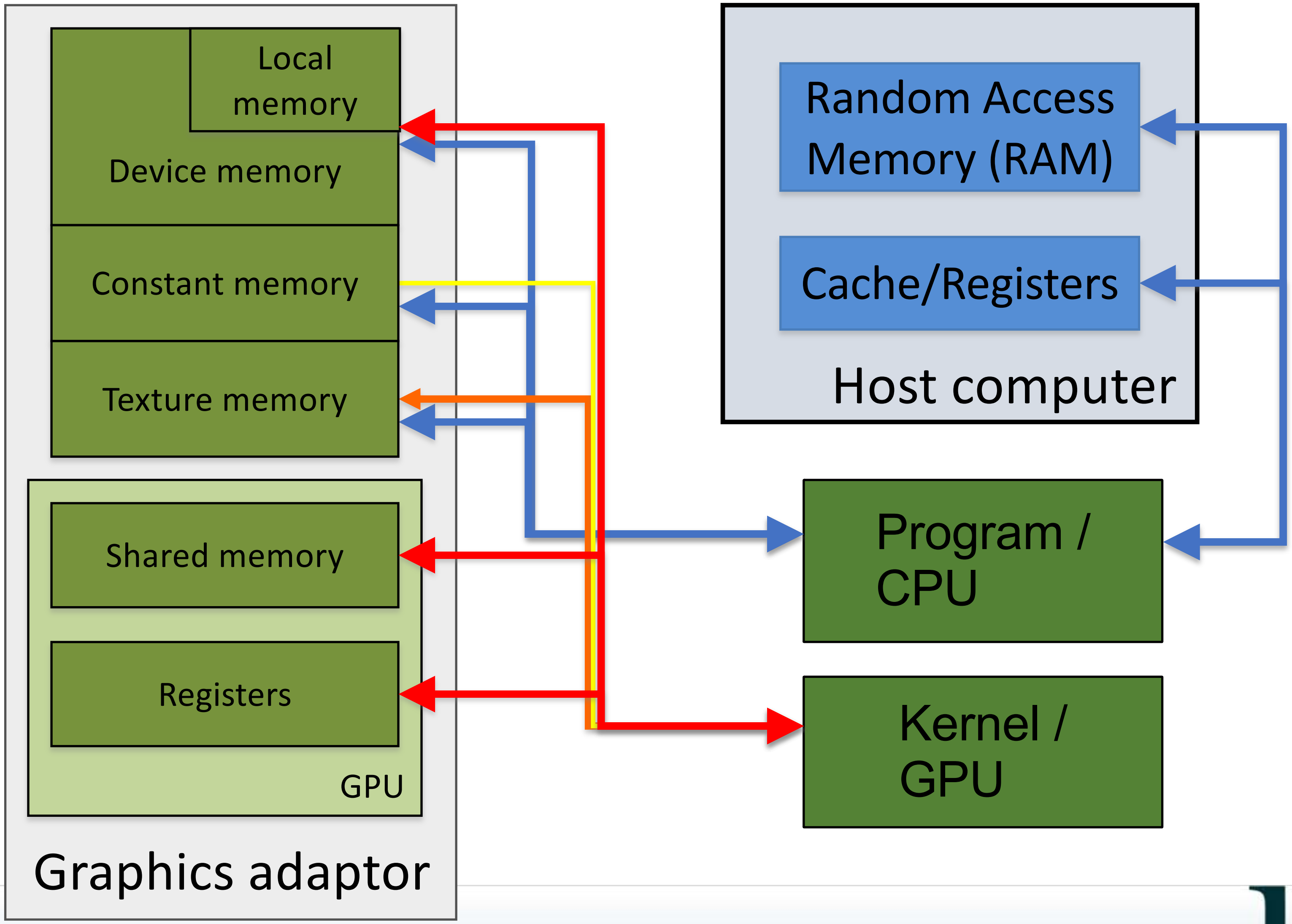


What happens on the GPU



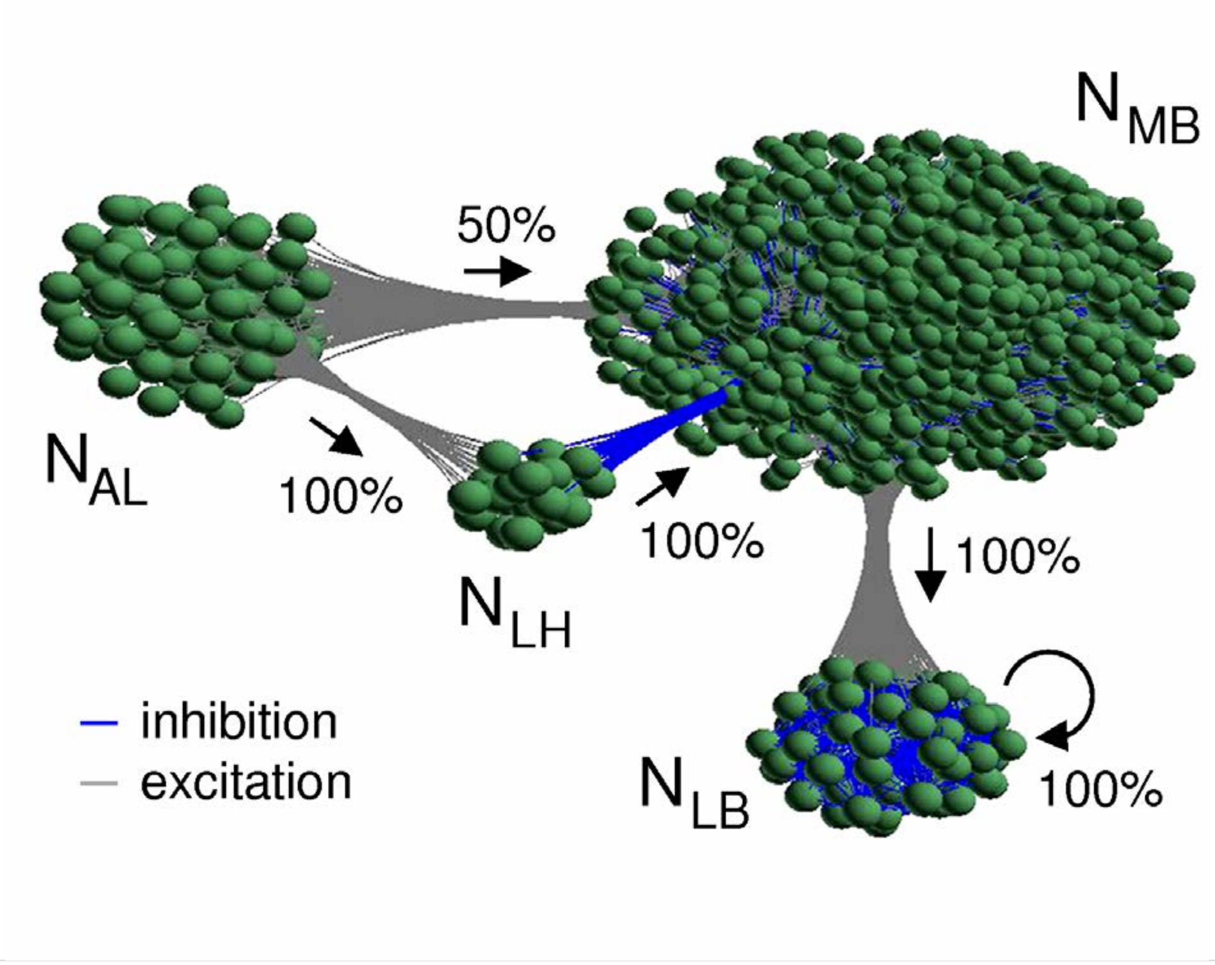
- GPUs have several streaming multi-processors (typically 2-16)
- Each block occupies one multi-processor (but several may occupy the same)
- Within the block, threads are executed in (half) warp sizes of (16) 32
- Other thread warps are swapped in and out

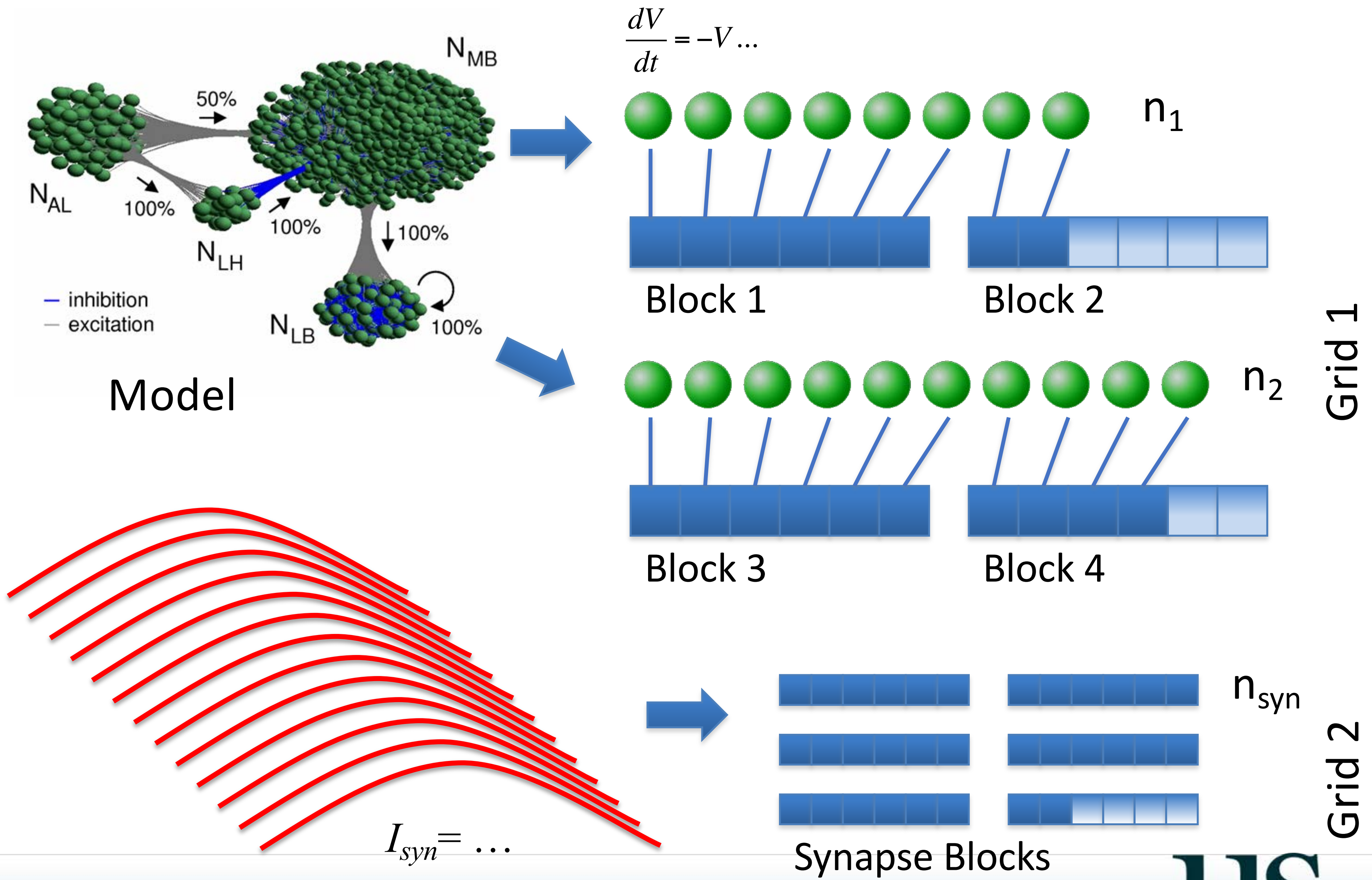
CUDA memory architecture



Prof. Thomas Nowotny (@drtnowotny)
CCNR and Sussex Neuroscience, School of Engineering and Informatics
UNIVERSITY OF SUSSEX

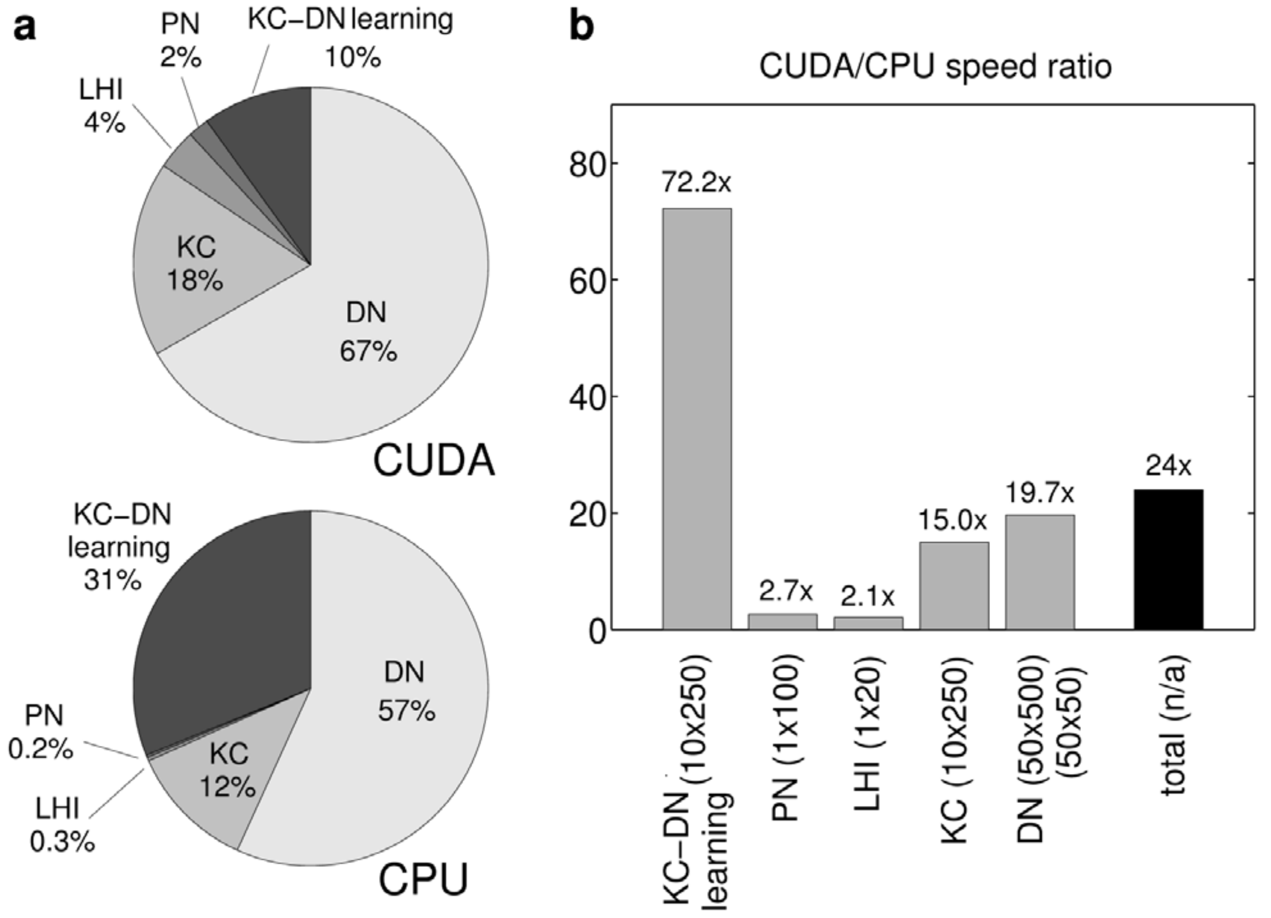
Example: Insect olfaction model





Prof. Thomas Nowotny (@drtnowotny)
CCNR and Sussex Neuroscience, School of Engineering and Informatics
UNIVERSITY OF SUSSEX

Hand-tuned network simulation 2009



NVIDIA Tesla C870,
240 cores

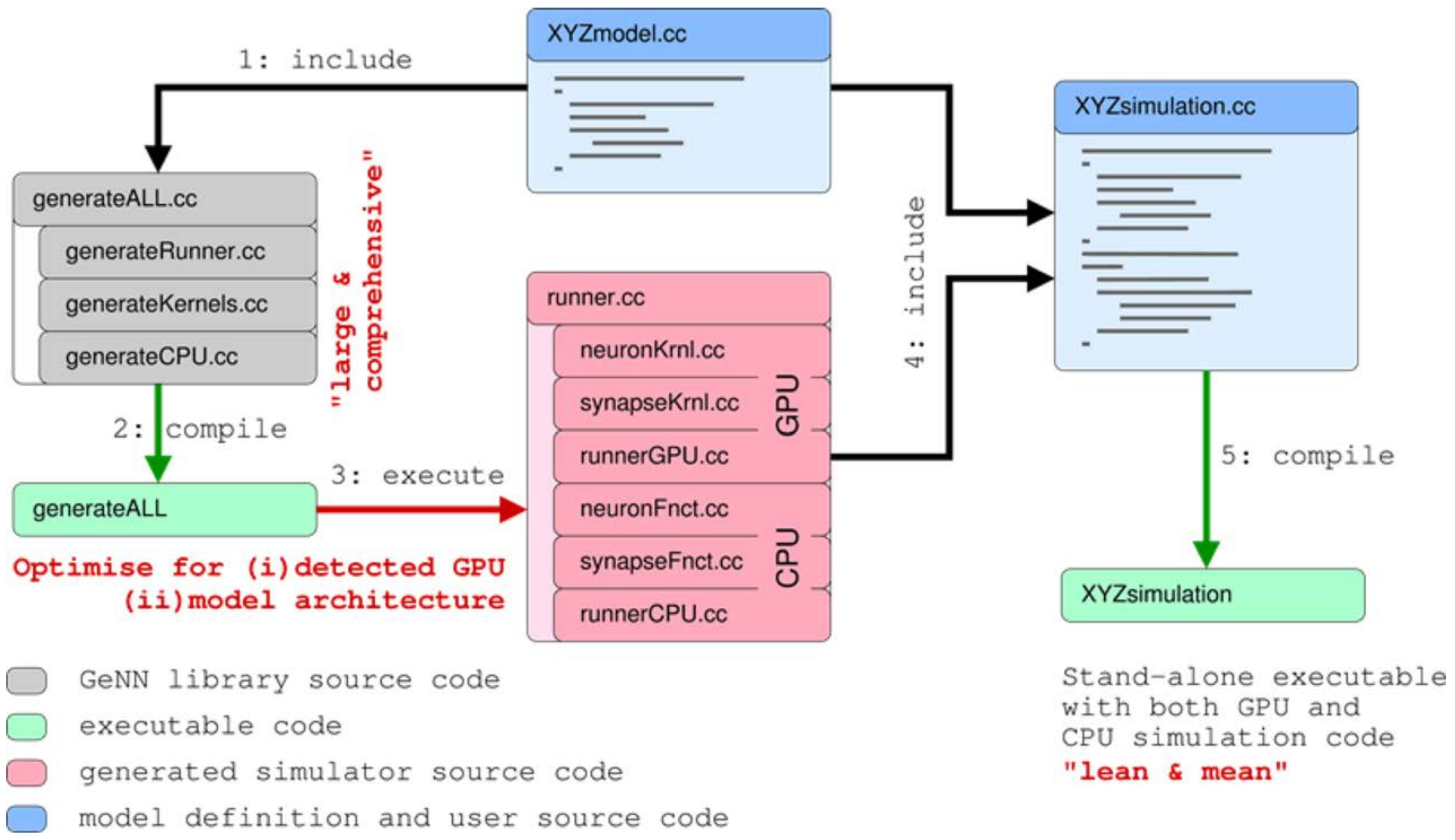
T. Nowotny,
WCCI 2010 Barcelona

GeNN

GeNN: GPU enhanced Neuronal Network simulator

- Based on code generation
- Provides a simple C++ API for specifying a neuronal network of interest
- Generates optimised C++ and CUDA code for the model and for the detected hardware at compile time (e.g. grid/block organisation, HW capability, model parameters)
- GeNN can offer a large variety of different models – only the used ones actually enter the generated code
- Users can add their own neuron models
- The generated code is compiled with the native NVidia compiler (and all its optimisations).

GeNN overview



Prof. Thomas Nowotny (@drtnowotny)
CCNR and Sussex Neuroscience, School of Engineering and Informatics
UNIVERSITY OF SUSSEX

Design choices

- The user has maximal control:
  - The user defines all equations/update code
  - GeNN generates kernels and data transfer ‘convenience functions’
  - The user decides what data to transfer and when
  - The user does their own I/O to disk etc.
- The user can interfere:
  - By providing neuron, synapse, learning models in the form of ‘integration time-step code’
  - Generated code is human-readable & can be ‘interfered’ with

Example GeNN model code

Definition of a neuron model

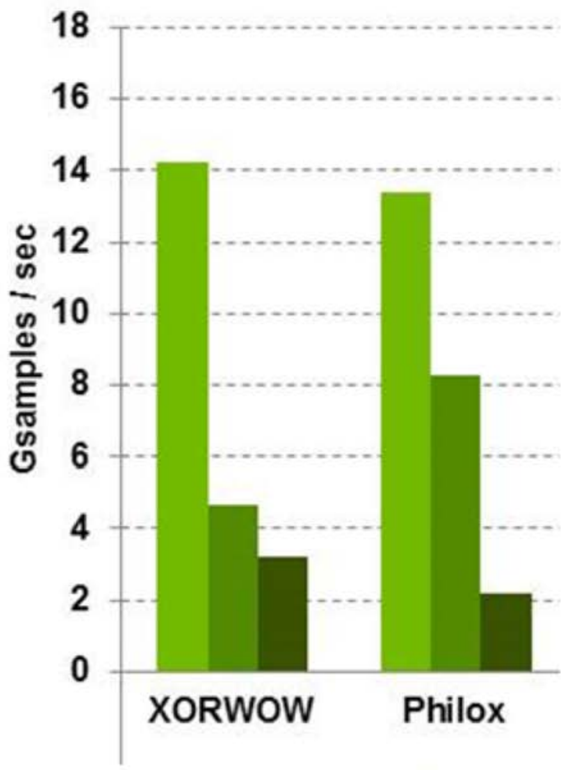
```
class LIF:public NeuronModels::Base
{
public:
  DECLARE_MODEL(LIF,1,1);
  SET_SIM_CODE("$(V)=($(Isyn)*$(TauM)*(1.0-$(ExpTC)))+($(ExpTC)*$(V));\n");
  SET_THRESHOLD_CONDITION_CODE("$(V)>=1.0");
  SET_RESET_CODE("$(V)=0.0;");
  SET_PARAM_NAMES({"TauM"});
  SET_DERIVED_PARAMS({
      {"ExpTC",[](const vector<double> &pars,double dt)
               {return exp(-dt/pars[0]);}}});
  SET_VARS({{"V","scalar"}});
};
IMPLEMENT_MODEL(LIF);
```

Population creation

```
network.addNeuronPopulation<LIF>("pop",1000,params,initState);
```

New Developments

GPU random number generation



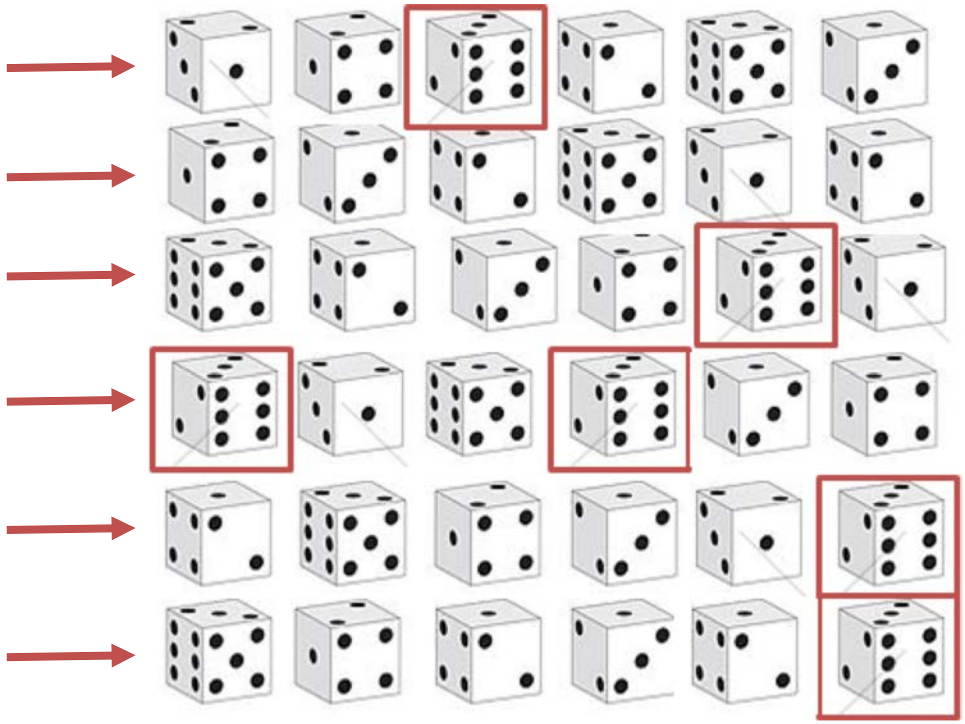
🟩 Uniform Distribution 🟩 Normal Distribution ■ Log-Normal Distribution

cuRAND 6.0 on K40m, ECC ON, double precision input and output data on device

```
__global__ void initializeDevice()
{
  const unsigned int id = 64 * blockIdx.x +
threadIdx.x;
  if (id < 1000) {
    curand_init(1234, id, 0, &dd_rng[id]);

    ...

    dd_VE[id] = curand_uniform(&dd_rng[id]);  }
}
```

Prof. Thomas Nowotny (@drtnowotny)
CCNR and Sussex Neuroscience, School of Engineering and Informatics
UNIVERSITY OF SUSSEX

Initialising 'fixed probability' connectivity



- Whether there is a connection between any pair of neurons can be modelled as Bernoulli Distribution
- 6.4x10^9 'dice' throws if connecting 80x10^3 neurons to themselves
- Inefficient for sparse connectivity

Prof. Thomas Nowotny (@drtnowotny)
CCNR and Sussex Neuroscience, School of Engineering and Informatics
UNIVERSITY OF SUSSEX

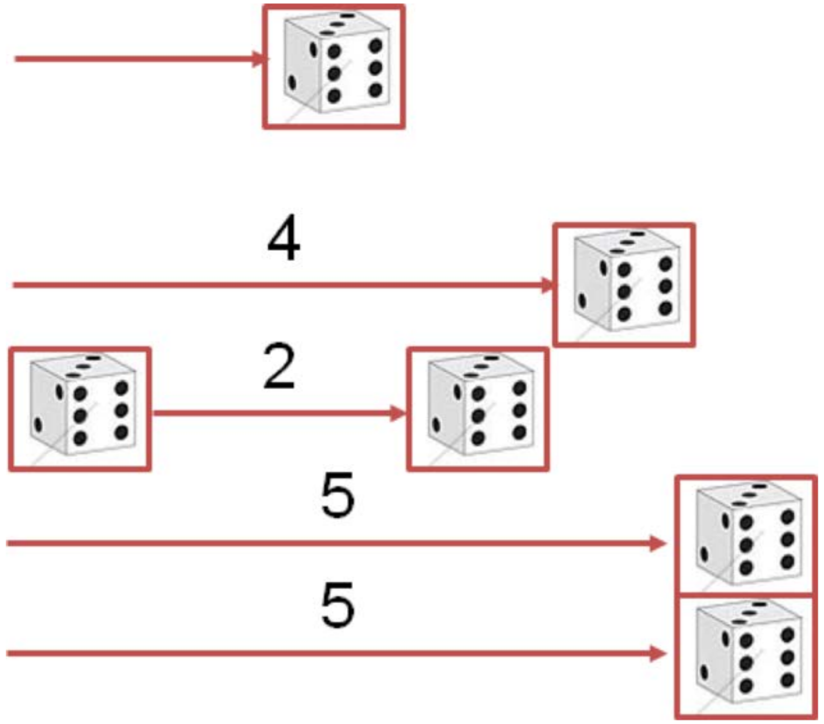
Initialising 'fixed probability' connectivity

- Alternative is to sample from the geometric distribution **Geom[p]**
- This distribution describes how many Bernoulli trials until the next success
- The geometric distribution can be sampled in constant time by inverting the cumulative density function (CDF) of its equivalent continuous distribution (the exponential distribution) to obtain for the 'distance' to the next existing synapse

$$\Delta = \frac{log(\mathrm{Unif}[0,1])}{log(1-p)}$$

Devroye, L. (1986). Non-Uniform Random Variate Generation. New York: Springer.

Initialising 'fixed probability' connectivity



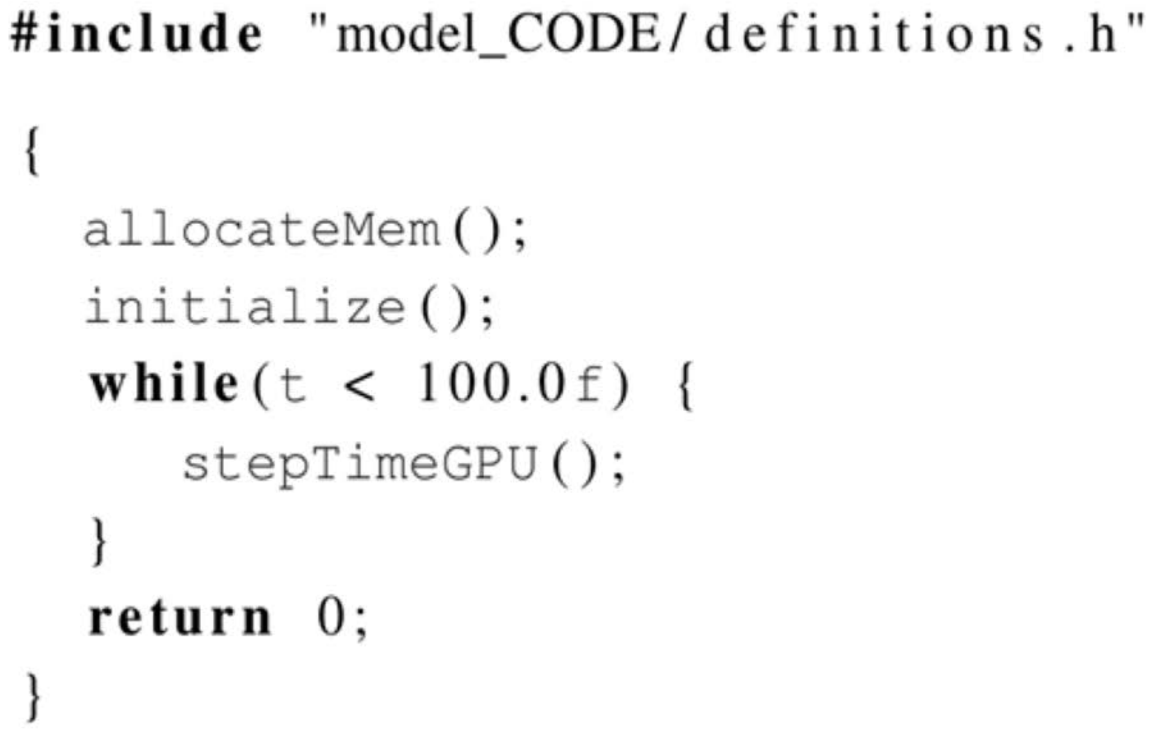
- Generating fixed probability connectivity can now be performed entirely in parallel by initializing each row of connectivity using an independent CUDA thread

Example GeNN connectivity model

$$\Delta = \frac{log(\mathrm{Unif}[0,1])}{log(1-p)}$$

```
class FixedProbability: public InitSparseConnectivitySnippet::Base
{
public:
    DECLARE_SNIPPET(FixedProbability, 1);

    SET_ROW_BUILD_CODE(
        "const scalar u = $(gennrand_uniform);\n"
        "prevJ += (1 + (int)(log(u) * $(probLogRecip)));\n"
        "if(prevJ < $(num_post)) {\n"
        "   $(addSynapse, prevJ);\n"
        "}\n"
        "else {\n"
        "   $(endRow);\n"
        "}\n");

    SET_ROW_BUILD_STATE_VARS({{"prevJ", "int", -1}});

    SET_PARAM_NAMES({"prob"});
    SET_DERIVED_PARAMS({{"probLogRecip",
                            [](const std::vector<double> &pars, double)
                            {
                                return 1.0 / log(1.0 - pars[0]);
                            }}});
};
```

Example GeNN simulation code



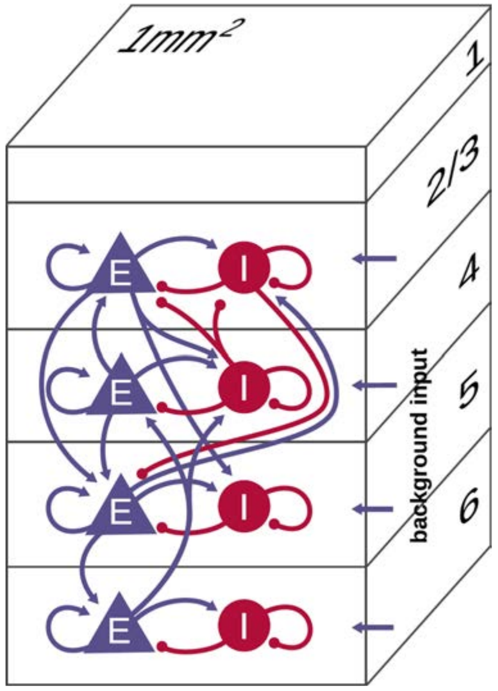
```
#include "model_CODE/definitions.h"

{
  allocateMem();
  initialize();
  while(t < 100.0f) {
     stepTimeGPU();
  }
  return 0;
}
```

Microcolumn Benchmark

Model

Jamie Knight

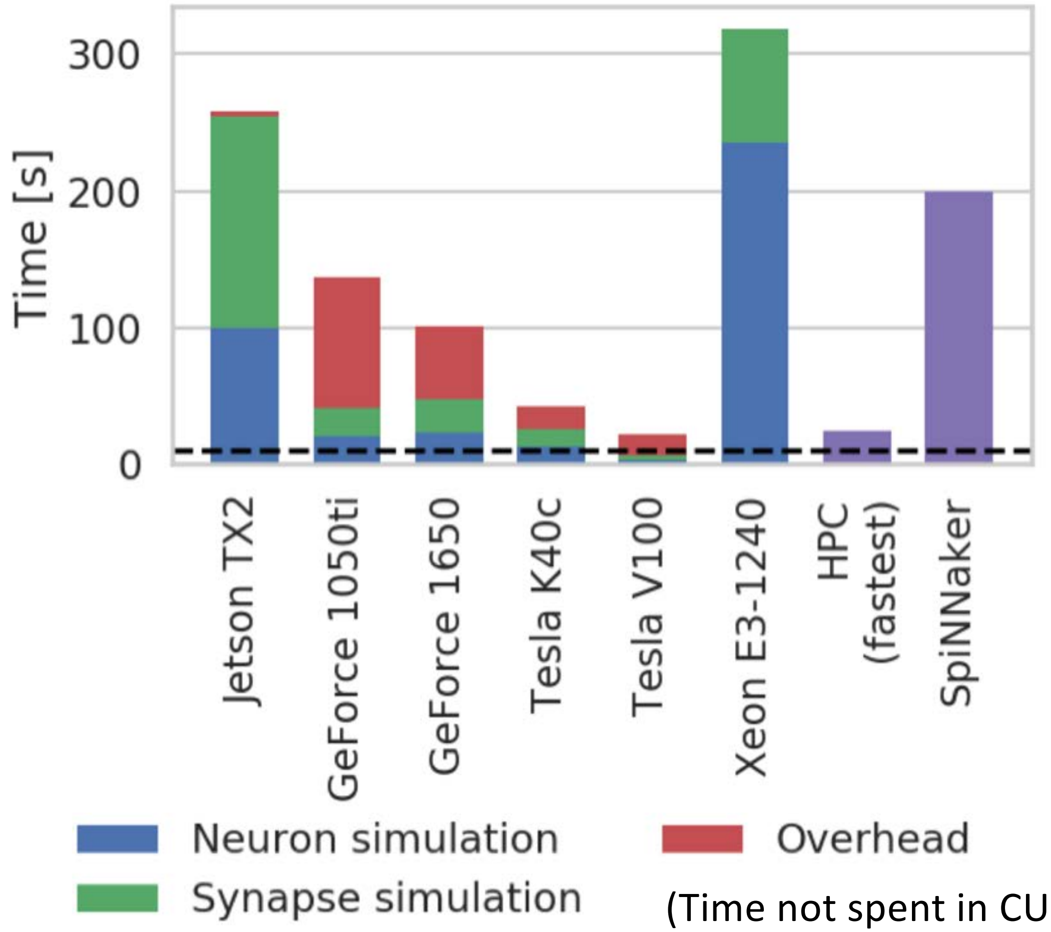
- $1mm^3$ of cortex
- 80×10^3 (LIF) neurons
- 0.3×10^9 (static) synapses
- Probabilistic connectivity
- Weights and delays sampled from normal distributions
- Recent benchmark
  - NEST on HPC - 0.33× real-time (25kW)
  - SpiNNaker - 0.05× real-time (277W)



Potjans, T. C., & Diesmann, M. (2012). The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model. Cerebral Cortex

Van Albada, S. J., Rowley et al. (2018). Performance Comparison of the Digital Neuromorphic Hardware SpiNNaker and the Neural Network Simulation Software NEST for a Full-Scale Cortical Microcircuit Model. 1–20.

Simulation performance

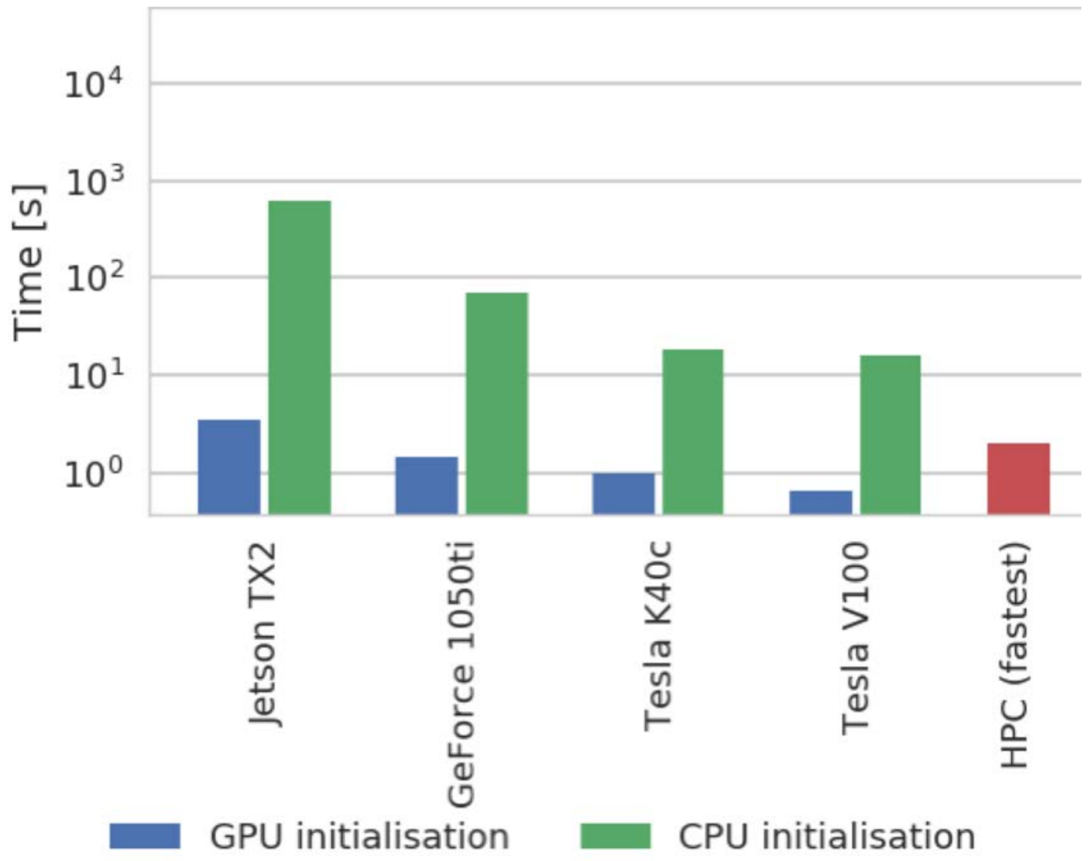


(Time not spent in CUDA kernels)

- Using latest Volta GPUs, GeNN is faster than HPC
- Consumer GPUs still faster than SpiNNaker and offer significant speedup over single CPU core

Prof. Thomas Nowotny (@drtnowotny)
CCNR and Sussex Neuroscience, School of Engineering and Informatics
UNIVERSITY OF SUSSEX

Initialisation performance



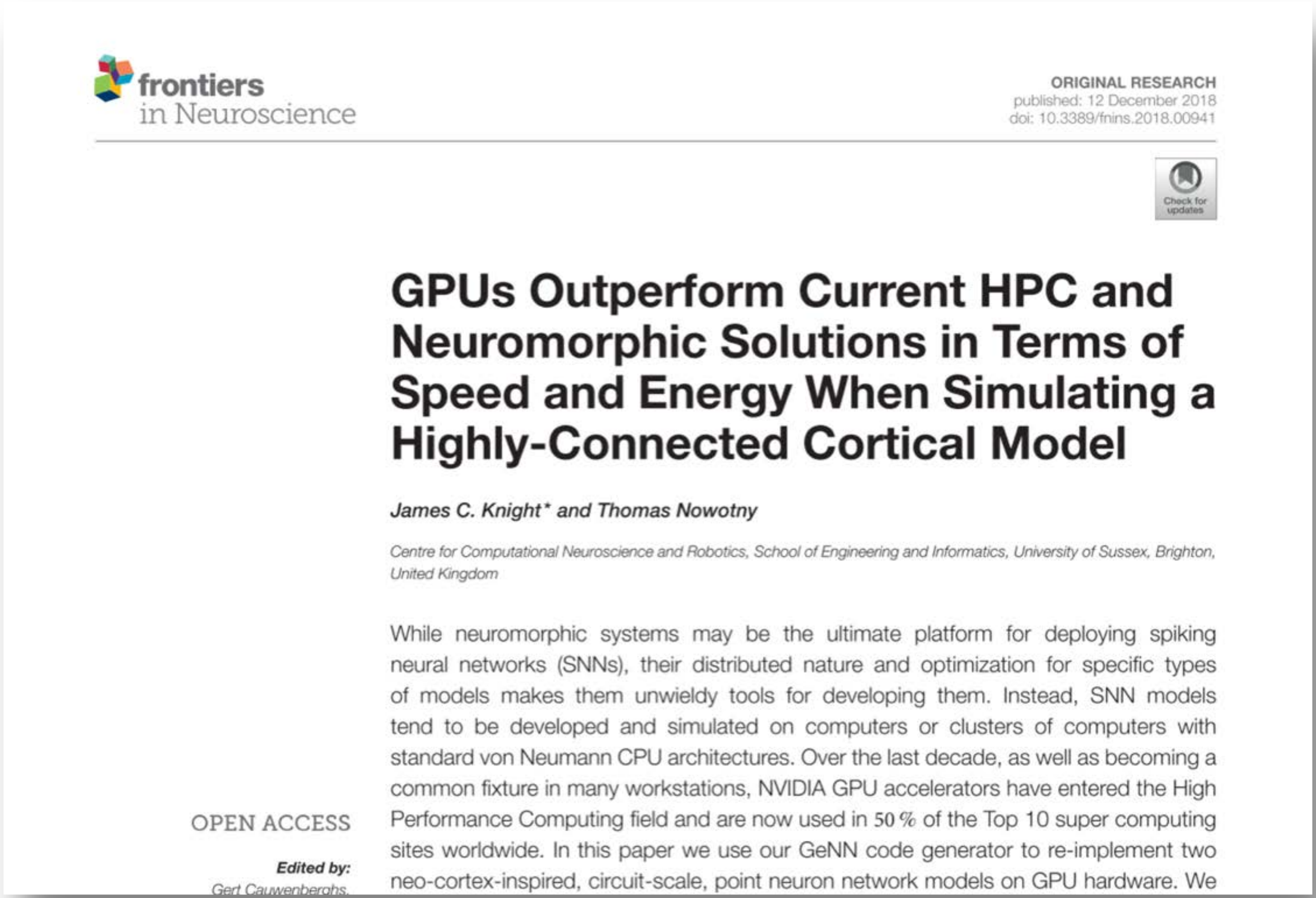
- Rapid initialisation of simulations is important
- Although initialisation also scales strongly on HPC cluster, GPU initialisation is still faster

Energy efficiency

- Latest Volta GPU uses an order of magnitude less energy than CPU-based HPC and SpiNNaker
- Even though it's significantly slower, Jetson TX2 remains very energy efficient

Device	**Simulation energy [kWh]**
GeForce 1050 Ti	0.0051
Jetson TX2	0.00078
Tesla K40c	0.0028
Tesla V100 (estimated)	0.0012
SpiNNaker	0.017
HPC (lowest energy)	0.012

Publication



frontiers in Neuroscience

ORIGINAL RESEARCH
published: 12 December 2018
doi: 10.3389/fnins.2018.00941

Check for updates

GPUs Outperform Current HPC and Neuromorphic Solutions in Terms of Speed and Energy When Simulating a Highly-Connected Cortical Model

James C. Knight and Thomas Nowotny*

Centre for Computational Neuroscience and Robotics, School of Engineering and Informatics, University of Sussex, Brighton, United Kingdom

OPEN ACCESS

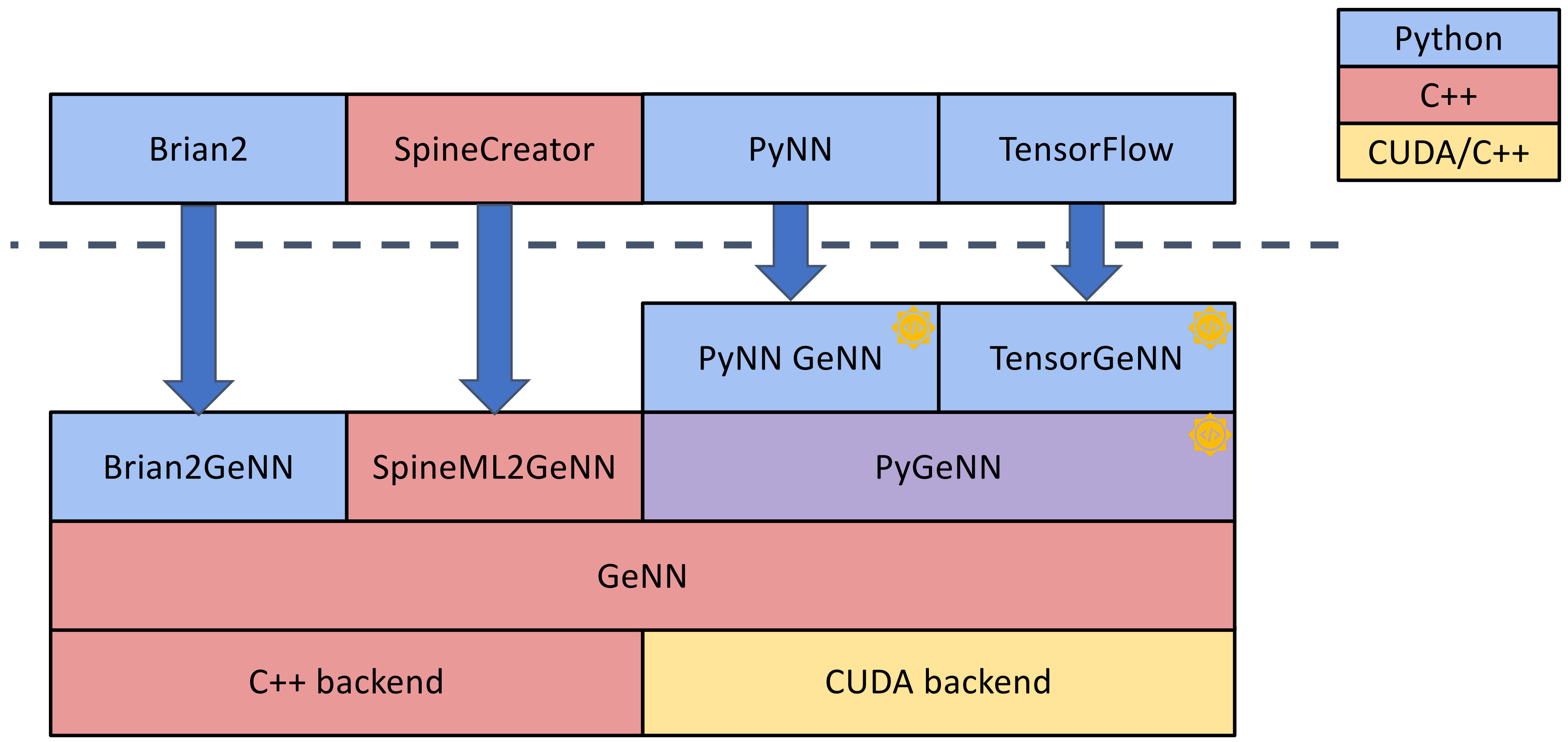
Edited by:
Gert Cauwenberghs,

While neuromorphic systems may be the ultimate platform for deploying spiking neural networks (SNNs), their distributed nature and optimization for specific types of models makes them unwieldy tools for developing them. Instead, SNN models tend to be developed and simulated on computers or clusters of computers with standard von Neumann CPU architectures. Over the last decade, as well as becoming a common fixture in many workstations, NVIDIA GPU accelerators have entered the High Performance Computing field and are now used in 50 % of the Top 10 super computing sites worldwide. In this paper we use our GeNN code generator to re-implement two neo-cortex-inspired, circuit-scale, point neuron network models on GPU hardware. We

UNIVERSITY OF SUSSEX

The GeNN 'ecosystem'

Overview



Prof. Thomas Nowotny (@drtnowotny)
CCNR and Sussex Neuroscience, School of Engineering and Informatics
UNIVERSITY OF SUSSEX

Courtesy of Dr. Alex Cope

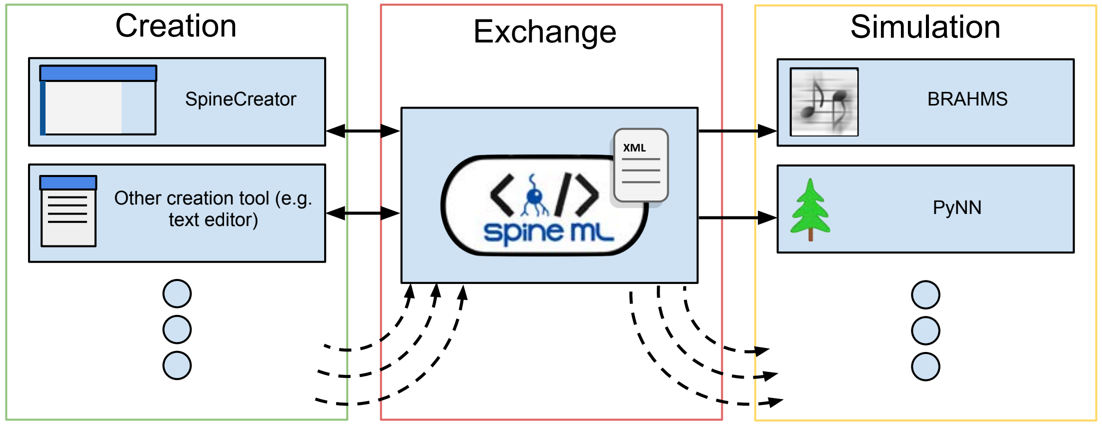
SpineML

SpineML is an XML-based description format for networks of point neurons

It is a proposed extension of the NineML format, including the description of neural dynamics, network structure, and experimental procedures.

SpineML acts as an exchange format between creation tools and simulators, as well as between collaborators working on a shared model

```
<SpineML
 xmlns="http://www.shef.ac.uk/SpineMLComponentLayer">
 <ComponentClass name="IF">
  <AnalogReducePort reduce_op="+" name="I_Syn" dimension="nA"/>
  <EventSendPort name="spike"/>
  <Parameter dimension="nF" name="cm"/>
  <Parameter dimension="nA" name="i_offset"/>
  <Parameter dimension="mV" name="v_thresh"/>
  <Parameter dimension="mV" name="v_rest"/>
  <Parameter dimension="mV" name="v_reset"/>
  <Parameter dimension="ms" name="tau_m"/>
  <Parameter dimension="ms" name="tau_refractory"/>
  <Dynamics initial_regime="integrating">
     ... Regimes ...
    <StateVariable dimension="mV" name="v"/>
    <StateVariable dimension="ms" name="t_spike"/>
  </Dynamics>
 </ComponentClass>
</SpineML>
```



For more details see:
Richmond P, Cope A, Gurney K, Allerton DJ. "From Model Specification to Simulation of Biologically Constrained Networks of Spiking Neurons" Neuroinformatics. 2013

Courtesy of Dr. Alex Cope

SpineCreator

SpineCreator is one tool that can be used to author SpineML models

A Graphical User Interface allows models to be specified without any programming

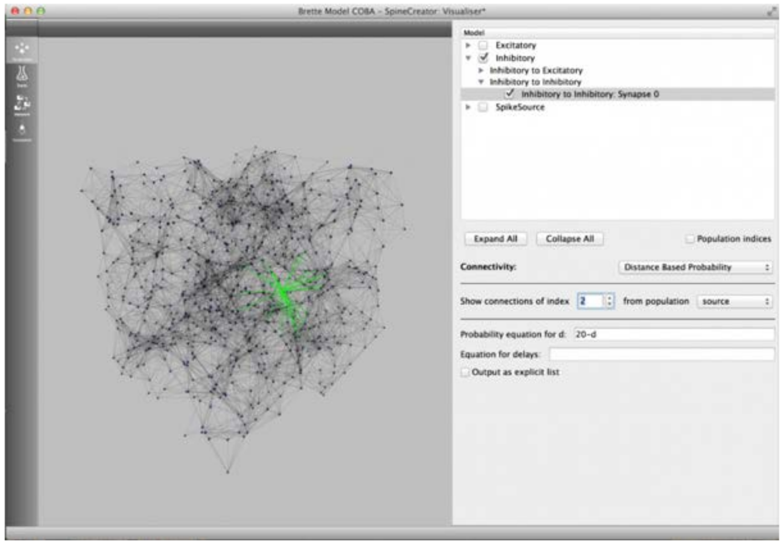
Some features:

Graphing

3d visualisation

Extensible simulator support

Version control support


Brette Model COBA – SpineCreator: Visualiser*

Model
- Excitatory
- ✓ Inhibitory
  - Inhibitory to Excitatory
  - Inhibitory to Inhibitory
    - ✓ Inhibitory to Inhibitory: Synapse 0
- SpikeSource

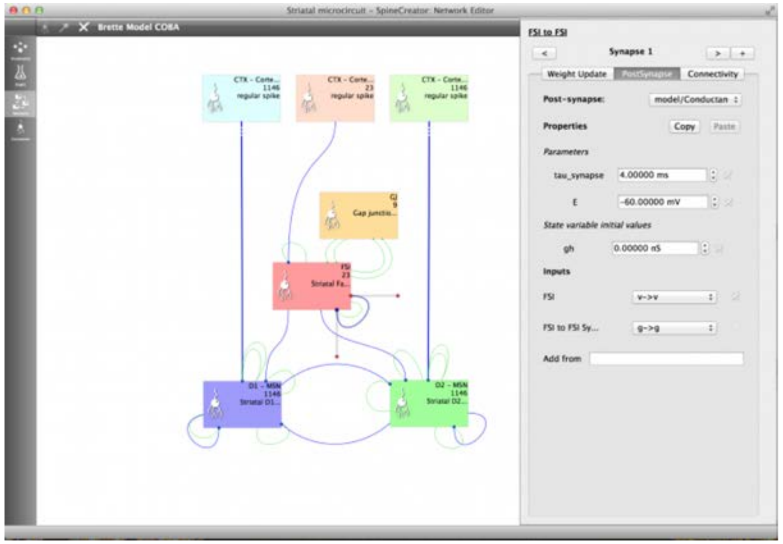
Connectivity: Distance Based Probability

Show connections of index 2 from population source

Probability equation for d: 20-d

Equation for delays:

Output as explicit list


Striatal microcircuit – SpineCreator: Network Editor

FSI to FSI

Synapse 1

Post-synapse: model/Conductan

Properties

Parameters

tau_synapse 4.00000 ms

E -60.00000 mV

State variable initial values

gh 0.00000 nS

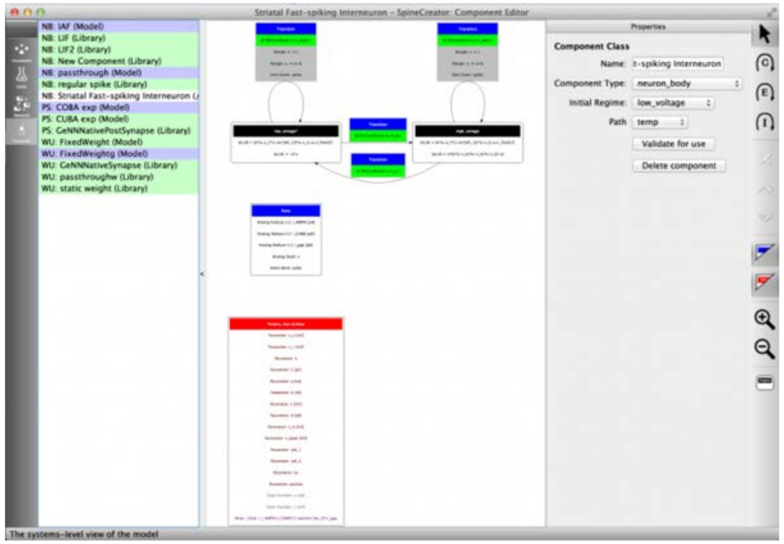
Inputs

FSI v->v

FSI to FSI Sy... g->g

Add from

Brette Model COBA
Weight Update
PostSynapse
Connectivity
Copy
Paste
CTX - Corte... 1146 regular spike
CTX - Corte... 23 regular spike
CTX - Corte... 1146 regular spike
Gap junctio...
FSI 23 Striatal Fa...
D1 - MSN 1146 Striatal D1...
D2 - MSN 1146 Striatal D2...


Striatal Fast-spiking Interneuron – SpineCreator: Component Editor

NB: IAF (Model)
NB: LIF (Library)
NB: LIF2 (Library)
NB: New Component (Library)
NB: passthrough (Model)
NB: regular spike (Library)
NB: Striatal Fast-spiking Interneuron (
PS: COBA exp (Model)
PS: CUBA exp (Model)
PS: GeNNNativePostSynapse (Library)
WU: FixedWeight (Model)
WU: FixedWeightg (Model)
WU: GeNNNativeSynapse (Library)
WU: passthroughw (Library)
WU: static weight (Library)

Properties

Component Class

Name: t-spiking Interneuron

Component Type: neuron_body

Initial Regime: low_voltage

Path temp

The systems-level view of the model

Available at https://github.com/SpineML/SpineCreator

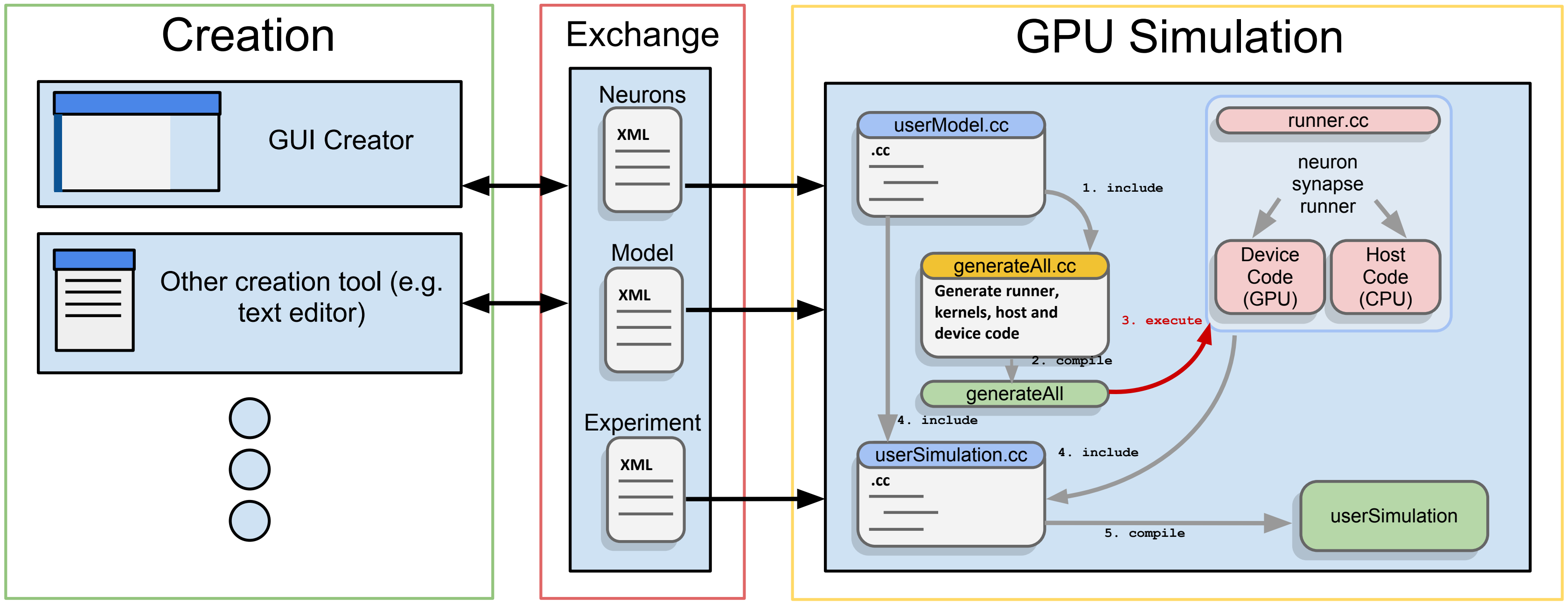
Courtesy of Dr. Alex Cope

SpineML and GeNN

SpineML can use code generation for simulator support, so it is very easy to add support for GeNN

XSLT translation scripts transform the model into GeNN user files, and the GeNN standard toolchain compiles and runs the simulation

Support for new neuron types (and soon synapses and weight update rules) is supported through XSLT translation to GeNN system files



Brian2GeNN interface

Brian is a popular simulator for neuronal networks (http://briansimulator.org/)

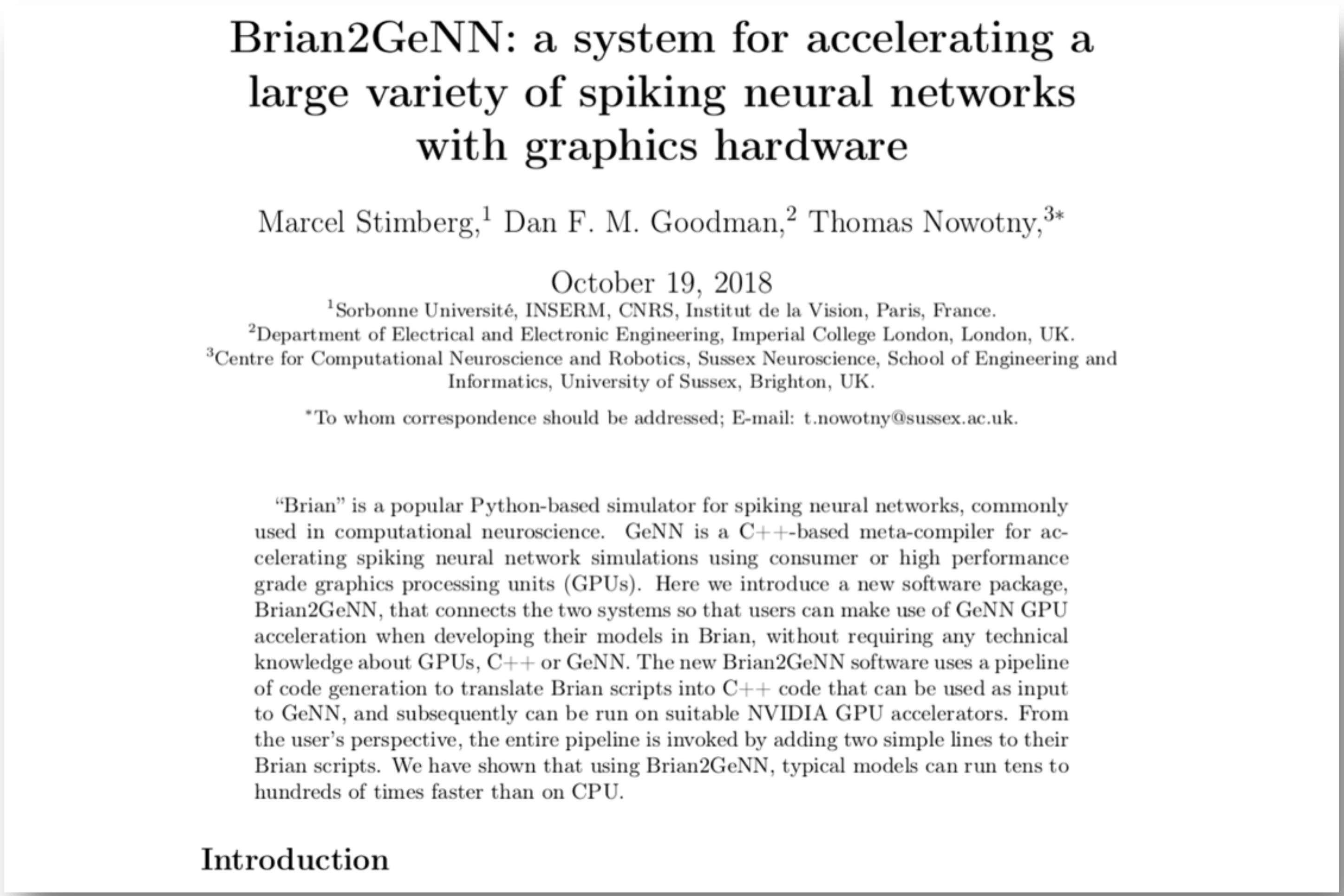
Brian 2 is entirely code-generation based

Users are able to choose target devices, including Python (as before), C/C++, Android and GPU/GeNN

BRIAN2

```
from brian2 import *
from brian2.devices.genn import *
set_device('genn');

##### Define the model
tau = 10*msecond
eqs = '’’dV/dt = (-40*mV-V)/tau : volt # (unless-refractory)'''
threshold = 'V>-50*mV'
reset = 'V=-60*mV'
refractory = 5*msecond
N = 1000

##### Generate genn code

G = NeuronGroup(N, eqs, reset=reset, threshold=threshold,
name='gp')
M = SpikeMonitor(G)
G2 = NeuronGroup(1, eqs, reset=reset, threshold=threshold,
name='gp2')
# Run the network for 0 seconds to generate the code
net = Network(G, M, G2)
net.run(1*second)
```

Brian2GeNN



Brian2GeNN: a system for accelerating a large variety of spiking neural networks with graphics hardware

Marcel Stimberg,[1] Dan F. M. Goodman,[2] Thomas Nowotny,[3*]

October 19, 2018

[1]Sorbonne Université, INSERM, CNRS, Institut de la Vision, Paris, France.
[2]Department of Electrical and Electronic Engineering, Imperial College London, London, UK.
[3]Centre for Computational Neuroscience and Robotics, Sussex Neuroscience, School of Engineering and Informatics, University of Sussex, Brighton, UK.

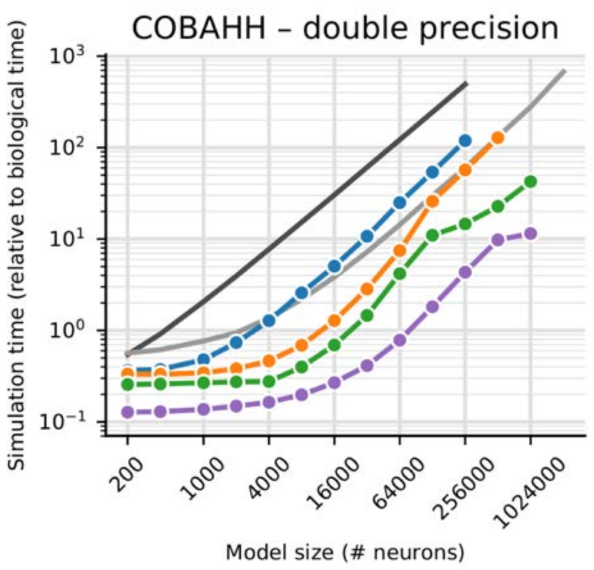
[*]To whom correspondence should be addressed; E-mail: t.nowotny@sussex.ac.uk.

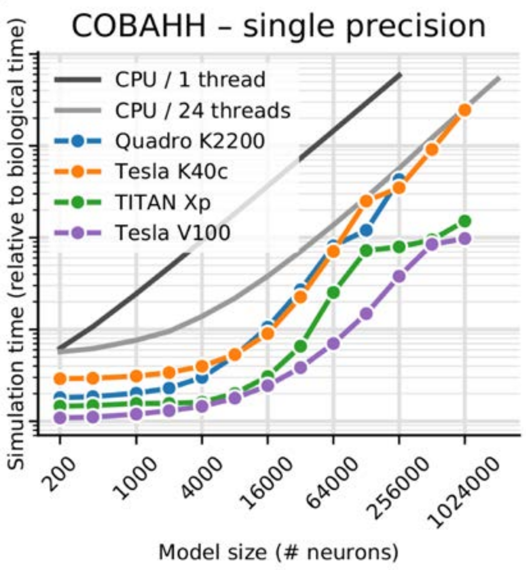
"Brian" is a popular Python-based simulator for spiking neural networks, commonly used in computational neuroscience. GeNN is a C++-based meta-compiler for accelerating spiking neural network simulations using consumer or high performance grade graphics processing units (GPUs). Here we introduce a new software package, Brian2GeNN, that connects the two systems so that users can make use of GeNN GPU acceleration when developing their models in Brian, without requiring any technical knowledge about GPUs, C++ or GeNN. The new Brian2GeNN software uses a pipeline of code generation to translate Brian scripts into C++ code that can be used as input to GeNN, and subsequently can be run on suitable NVIDIA GPU accelerators. From the user's perspective, the entire pipeline is invoked by adding two simple lines to their Brian scripts. We have shown that using Brian2GeNN, typical models can run tens to hundreds of times faster than on CPU.

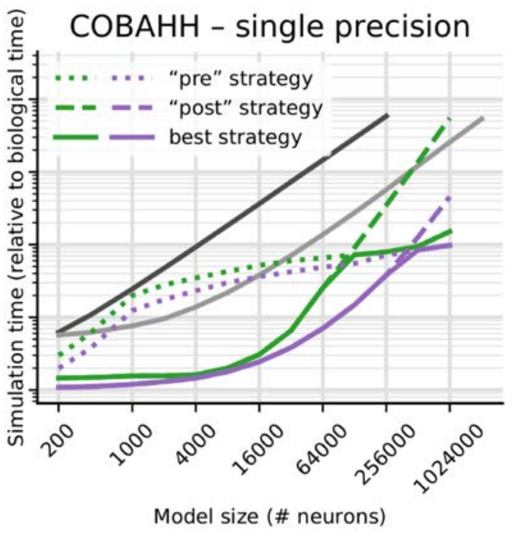
Introduction

Available on bioarxiv

Brian2GeNN performance







- Very large models can be simulated on single GPUs
- Tesla GPUs perform well, even when double-precision accuracy is required
- When models are large enough, higher performance can be achieved by parallelising synaptic update over incoming spikes rather than postsynaptic neurons.

Future

- New backends
  - OpenCL
  - Intel ISPC
- Going larger
  - Support for NVLink multi-GPU systems
  - Support for GPUDirect RDMA
- Going smaller
  - Low-precision types reduce load on memory and memory bandwidth and offer increased throughout on newer devices
  - New backends may allow us to efficiently target devices too small to have an NVIDIA GPU

Some Conclusions

- Using a C++/CUDA code generation approach has several advantages:
  - Model specific optimisations at compile time
  - Hardware specific optimisations at compile time
  - Can provide unlimited number of different models but actual simulations stay lean and mean
- GeNN is freely extendible with few constraints
- Low level code is accessible if desired/needed
- New hardware capability can be accommodated
- https://github.com/genn-team/genn

Other GPU solutions (here: Neural Networks)

- Nageswaran et al. (UC Irvine, 2009): General simulator for Izhikevich neurons with delay, optimized for Izhikevich's thalamo-cortical model (C++ library)
- Fidjeland et al. (Imperial, 2010) - Nemo: General simulator for Izhikevich neurons with delay, optimized for Izhikevich's thalamo-cortical model (C++ library)
- Mutch et al. (MIT, 2010) CNS simulator: Simulator for layered "cortical networks", models can be defined by the user (used exclusively through a MatLab interface)
- Minkovich et al., (HRL Laboratories LLC, 2013) HRLSim: A High Performance Spiking Neural Network Simulator for GPGPU Clusters (LIF & Izhikevich)
- …

Acknowledgments

ANR AGENCE NATIONALE DE LA RECHERCHE

PheroSys

BBSRC bioscience for the future

2008-2011

EPSRC Engineering and Physical Sciences Research Council

GREEN BRAIN

2012-2016

brains on BOARD

2017-2021

HFSP HUMAN FRONTIERS SCIENCE PROGRAM

odor objects

2015-2019

Human Brain Project

2015-2021

Thank you!

Prof. Thomas Nowotny (@drtnowotny)

CCNR and Sussex Neuroscience, School of Engineering and Informatics

UNIVERSITY OF SUSSEX