Neuroinform (2013) 11:267–290
DOI 10.1007/s12021-012-9174-x

ORIGINAL ARTICLE

Three Tools for the Real-Time Simulation of Embodied Spiking Neural Networks Using GPUs

Andreas K. Fidjeland · David Gamez · Murray P. Shanahan · Edgars Lazdins

Published online: 30 December 2012

Abstract This paper presents a toolbox of solutions that enable the user to construct biologically-inspired spiking neural networks with tens of thousands of neurons and millions of connections that can be simulated in real time, visualized in 3D and connected to robots and other devices. NeMo is a high performance simulator that works with a variety of neural and oscillator models and performs parallel simulations on either GPUs or multi-core processors. SpikeStream is a visualization and analysis environment that works with NeMo and can construct networks, store them in a database and visualize their activity in 3D. The iSpike library provides biologically-inspired conversion between real data and spike representations to support work with robots, such as the iCub. Each of the tools described in this paper can be used independently with other software, and they also work well together.

Keywords Simulation · 3D visualization · Spike encoding · Robotics · Spiking neural networks · iCub · GPU

Introduction

Large-scale spiking neural networks (SNNs) are being used to construct increasingly realistic models that aim to increase our understanding of the brain. It is also hoped that greater knowledge about the brain could help us to build more intelligent machines. A key problem with this work is that the human brain contains 10^{11} neurons and up to 10^{15} synapses, and so any non-trivial model faces formidable computational challenges.

One approach to the modelling of large numbers of neurons is to distribute neural simulation software across a number of CPUs in a cluster or supercomputer. For example, Izhikevich and Edelman (2008) modelled the mammalian thalamocortical system using a million ∼40 compartment neurons and ∼0.5 billion synapses with a cluster of 60 3 GHz processors, and Ananthanarayanan et al. (2009) constructed a brain-inspired model with 1.6 billion point neurons and 8.87 trillion synapses, which was run on an IBM supercomputer. Tens of thousands of highly biologically realistic multi-compartment neurons have also been simulated on a supercomputer as part of the Blue Brain Project (Markram 2006). The main limitation of these and other regular software simulators (Bernardet and Verschure 2010; Bower et al. 2003; Carnevale and Hines 2006; Djurfeldt et al. 2008; Gewaltig and Diesmann 2007; Goodman and Brette 2009; Hammarlund and Ekeberg 1998) is that they typically have low performance when run across a small number of CPUs or a cluster, which makes them unsuitable for real-time robotics work with large-scale networks. While software can be used to simulate extremely large networks on supercomputers, this hardware is very expensive and only available to a restricted number of researchers. Software simulators have the further limitation that they are typically not power efficient.

To address the problems with software simulators, specialised hardware is being developed to model large numbers of neurons with high energy efficiency in real time. For example, the FACETS project created a silicon wafer that can simulate 4×10^7 synapses and up to 180,000 neurons faster than real time, and these wafers can be connected together to create larger networks (Schemmel et al. 2010).

A. K. Fidjeland (✉) · D. Gamez · M. P. Shanahan · E. Lazdins
Imperial College London, 180 Queen's Gate,
London, SW7 2AZ, UK
e-mail: andreas.fidjeland@imperial.ac.uk

Another example is the SpiNNaker project, which is developing a system that can run a variety of neural models on a custom chip (Rast et al. 2011), with plans to scale up to a billion neurons running in real time. Smaller scale and very fast hardware models have been created using field-programmable gate arrays (FPGAs) (Cheung et al. 2009; Thomas and Luk 2009). Other neural simulators based on specialized hardware are covered in recent reviews (Indiveri et al. 2011; Maguire et al. 2007).

In recent years graphics processing units (GPUs) have become viable platforms for general purpose parallel processing, as they provide a cheap and efficient means of modelling medium-scale networks of spiking neurons. The first tool described in this paper is the NeMo GPU/CPU-based SNN simulator, which is capable of modelling around 100,000 point neurons and 100 million connections in real time (Section "NeMo: GPU and CPU Spiking Neural Simulator")—enough to simulate a fruit fly brain without dendritic trees. Apart from the NeMo simulator described in this paper, a number of other GPU-based spiking simulators have been developed. For example, Nageswaran et al. (2009) and Richert et al. (2011) developed GPU-SNN, which runs on GPUs or CPUs and simulates arbitrary networks of a few hundred thousand Izhikevich neurons with STDP and short-term synaptic plasticity. A simulator supporting more constrained network types has been developed by Tiesel and Maida (2009), which simulated a planar network of 160,000 integrate-and-fire neurons on a GPU. Bernhard and Keriven used early general-purpose GPUs to create a simulator for smaller static planar networks for image segmentation (Bernhard and Keriven 2006) and Han and Taha (2010) created a large-scale simulation with sparse regular connectivity without delays on a cluster of GPUs. Still earlier GPU-based neural network simulation efforts focused on rate-based rather than spiking models, and are reviewed in Meuth and Wunsch (2007). A simulator focusing on numerical accuracy has also been developed by Yudanov et al. (2010), which could model ∼4,000 Izhikevich neurons in real time, and a neural simulation framework that dynamically generates CUDA code (GeNN) has been developed by Nowotny (2011). Of these simulators, GPU-SNN is the closest to NeMo in terms of features, and NeMo outperforms GPU-SNN in terms of spike delivery (Section "NeMo Performance"). Furthermore, NeMo is one of the few GPU-based simulators that supports a variety of neuron and oscillator models.

In network modelling it is often convenient to have a graphical interface that enables the user to control the simulation and visualize neuron activity as it occurs—instead of performing off-line analysis of network data at the end of each run. This kind of interactivity makes exploratory modelling easier, it reveals network dynamics, and it is particularly helpful when working with external data sources, such as robots. The second tool described in this paper is the SpikeStream visualization and analysis environment, which can construct networks, store them in a database, control NeMo and visualize networks' structure and activity in 3D.

With a few exceptions (Bernardet and Verschure 2010; Krichmar et al. 2005), most previous large-scale neural models have attempted to replicate the brain's behaviour without dynamic communication with the environment. While resting state models can help us to understand connectivity and core operations, most of the brain's key functionality is engaged when it is actively interacting with the world, and it is now increasingly thought that the brain cannot be fully understood without taking its embodiment and environment into account (Clark 2008; Noë and Thompson 2004). While a number of increasingly realistic robotic models of animal and human bodies have been built (Ijspeert et al. 2007; Liu and Hu 2006; Marques et al. 2010), there have been few attempts to connect them up to medium or large scale biologically-inspired networks of spiking neurons. This type of robotic work depends on a simulator that can run close to real time, and it requires an interface that can convert sensory data into spikes and convert spiking output from the network back into motor commands that are sent to control the robot. The third tool described in this paper is the iSpike library, which provides biologically-inspired conversion between real data and spike representations and supports communication with the iCub robot (Metta et al. 2008) using the YARP robotics middleware system (Fitzpatrick et al. 2008).

Other software for sensory modelling and spike conversion includes a basic spiking interface for proprioception and the motor system that was constructed for the iCub robot (Bouganis and Shanahan 2010) and an interface to the C2 simulator, also coincidentally called SpikeStream, which was developed to encode visual stimuli from a model retina and provide output to a virtual environment (Ananthanarayanan et al. 2009). Sensory conversion software is included with the iqr simulator (Bernardet and Verschure 2010), which can interface with a wide range of devices, and a visual model for spike conversion has been created by Masquelier and Thorpe (2007). Hardware models of the retina have also been developed, for example Andreou and Boahen (1994) and Linares-Barranco et al. (2007), as well as a Python library that models the auditory system and integrates with neural simulators (Fontaine et al. 2011). A key limitation of much of this work is that the software for spike encoding and decoding is often not made generally available or it is distributed as an integral part of a neural simulator, which makes it difficult to use with other software.

The applications of the tools described in this paper include machine learning, the simulation of biologically-inspired networks embodied in robots and the modelling of different aspects of the brain. In our laboratory SpikeStream has been used to create a network of ∼18,000 spiking neurons that controlled the eye movements of the SIMNOS virtual robot (Gamez 2010), and we have used NeMo to model a global workspace constructed from ∼20,000 spiking neurons, which controlled an avatar in real time within the Unreal Tournament 2004 game environment (Fountas et al. 2011). SpikeStream and NeMo have also been used to explore the applications of spiking neural networks in high frequency financial trading (Buchmann 2011), and NeMo is being used to model the oscillation behaviour of spiking neural networks as part of ongoing work on brain dynamics (Bhowmik and Shanahan 2012). We are also collaborating with the developers of the Brian simulator to enable NeMo to be used as an accelerator within this framework.

Each of the tools described in this paper can be used independently with other software, and they also work well together. They have been made freely available for a variety of different platforms (see Information Sharing Statement below), enabling advanced research on spiking neural networks and robotics to be carried out on a consumer PC equipped with a CUDA graphics card.

This paper starts by describing the design and functionality of NeMo, SpikeStream, and iSpike. Next, some experiments are set out that were used to benchmark the tools' performance and accuracy. We then present the results and a discussion with some suggestions for future work.

NeMo: GPU and CPU Spiking Neural Simulator

Simulator Overview

NeMo (http://nemosim.sf.net) is a high performance spiking neural network simulator for current-based point neurons and oscillator models. It is aimed at real-time simulation of networks of more than 100,000 point neurons on a single computer, and it provides a good basis for the simulation of networks across larger scale clusters. High performance is achieved by targeting modern many-core architectures, which offer the high compute and memory bandwidth that is required for such simulations. NeMo provides multiple front-ends, both programmatic and graphical, to fit in with the variety of workflows that are employed by roboticists and computational neuroscience practitioners (see Fig. 1). Early versions of the simulator have been presented in previous work (Fidjeland et al. 2009; Fidjeland and Shanahan 2010); the version presented in this paper has been substantially reworked to provide better performance, more features and improved usability.

NeMo has been designed to perform well on a particular class of network models. These have guided development, but are not necessarily strict constraints (see Table 1). NeMo is primarily a simulator of point neuron models that can be described by small numbers of ordinary differential equations—for example, the Izhikevich model (Izhikevich 2003) or leaky-integrate-and-fire (LIF) neurons. The synapses are quite basic, with a conductance delay and a weight, and they can be modified using a spike-timing dependant plasticity (STDP) rule. Neurons are modelled

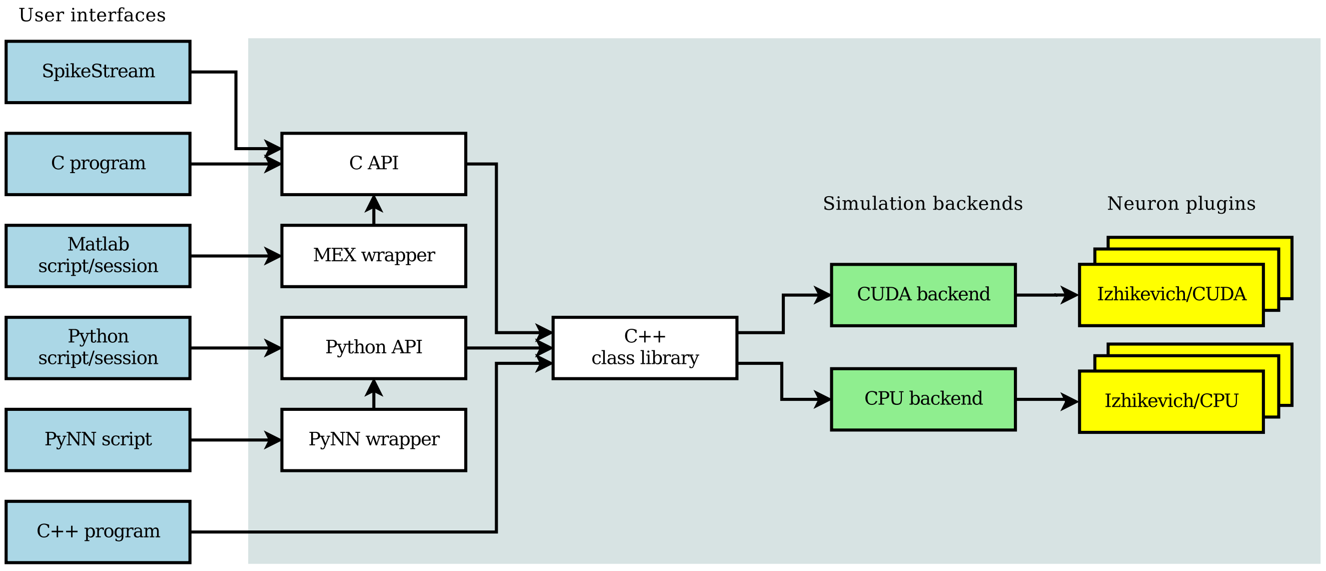


Fig. 1 NeMo overview showing the available user interfaces and simulation backends

Table 1 NeMo features, design assumptions, and limitations

Aspect *notes*	Value
Neuron model	Izhikevich, Input, LIF, Poisson source, Kuramoto
other neuron models can be supported via plugins	
Minimum delay	1 ms
could be reduced to a smaller value	
Maximum delay	64 ms
could be extended by a small factor	
Temporal resolution	1 ms
could be reduced by up to an order of magnitude along with minimum delay	
Plasticity model	pair-based nearest neighbour
support for more generic trace-based model (Morrison et al. 2008) *planned*	
Synapses/neuron	$\geq$1,000
no strict limitation, but throughput reduced for low values	
Integration method	Euler method with 250 μs step size
RK4 planned	

using a fairly coarse temporal resolution (1 ms by default) and conductance delays are assumed to range from one to a few tens of milliseconds. It is assumed that neurons form thousands of synaptic connections with other neurons, and these are assumed, but not required, to be of a clustered nature.

Different neuron models are supported via a plugin architecture. Currently NeMo supports Izhikevich neurons (Izhikevich 2003), sensory neurons (externally driven with no internal dynamics), integrate-and-fire neurons with decaying exponential or alpha-shaped post-synaptic currents or conductances, and Poisson spike sources. Additionally, NeMo can simulate networks of delay-coupled Kuramoto oscillators (Kuramoto 1984), which can be a useful way of modelling neural systems. It is possible to construct networks with a mixture of neuron types.

Synaptic plasticity is supported via a pair-based nearest neighbour STDP mechanism (Song et al. 2000), in which an arbitrary user-defined STDP function is applied to all of the plastic synapses in the network. STDP is implemented as an accumulated eligibility trace, which is used to update the weights of the plastic synapses at the user's discretion, with an optional reward modulation.

A number of different interfaces can be used to control the simulator. NeMo is implemented as a C++ class library, which can be directly linked against in a C++ application. A pure C API is also exposed, which makes it possible to link against the library from a C application or from any programming language with a C foreign function interface. A Matlab interface is available and a Python wrapper exposes the library to scripts written in that language, or to Python interactive and graphical environments, such as IPython and Matplotlib. This Python wrapper forms the basis for NeMo's PyNN bindings.[1] SpikeStream, which is covered in Section "SpikeStream: 3D Visualization, Creation, Storage and Analysis of Neural Networks", offers a full graphical interface.

A network is constructed in NeMo by adding individual neurons and synapses,[2] with higher-level connectivity patterns being the responsibility of the caller or a wrapper library.[3] The simulation interface is similarly low-level: the caller steps through the simulation on a per-time-step basis, reading back firing for every step. Stimulus can be provided both by listing neurons that should be forced to fire during a particular step, or by providing stimulus current for individual neurons. Random Gaussian input current can be generated from within the simulation itself, configurable on a per-neuron basis. The simulation can be queried to get the state of each neuron or synapse.

NeMo can run on a number of different types of hardware. The CUDA backend runs simulations on highly parallel CUDA-enabled GPUs, simulations can run on regular multi-core CPUs (using OpenMP), and an MPI implementation is under development that will enable NeMo to run across a cluster of machines. The mapping of the simulation onto these parallel backends is handled by the library, and so the simulation backend and parallelisation issues are transparent to the user.

Simulator Usage

To illustrate NeMo's programmatic interface, this section presents a short script that constructs and simulates a small network of Izhikevich neurons. This script is written in Python, with the APIs in other languages following a similar structure. Full API documentation is available at http://nemosim.sf.net/docs.html.

NeMo is exposed via the namespace `nemo`.

```
import nemo
```

[1] PyNN (Davison et al. 2008) is a common SNN simulator interface written in Python. By default PyNN is supported by several simulators including NEURON (Carnevale and Hines 2006), NEST (Gewaltig and Diesmann 2007), and Brian (Goodman and Brette 2009). PyNN support for NeMo is currently found in its development branch.

[2] The Matlab and Python interfaces also provide vector versions of the network construction functions.

[3] Both SpikeStream and PyNN provides such connectivity patterns, for example, to create topographic connections.

This namespace contains three classes: `Network`, which contains the network being constructed; `Configuration`, which contains global configuration parameters; and `Simulation`, which encapsulates the network state during simulation.

To use a particular neuron type within NeMo it must first be registered with the network. This returns a neuron type reference which can be used to add neurons of that type. For example, a network of Izhikevich neurons is created in the following way:

```
net = nemo.Network()
n_iz = net.add_neuron_type('Izhikevich')
```

Neurons are added individually to the network, specifying each neuron's type, index, parameters and initial state:

```
# excitatory neurons
for nidx in range(0, 800):
    re = random()**2
    c = -65.0+15*re
    d = 8.0 - 6.0*re
    net.add_neuron(n_iz, nidx,
            0.02, 0.2, c, d, 5.0, 0.2*c, c)
```

The user is responsible for allocating neuron indices, and will be warned of duplicates. The number and order of parameters and state variables depends on the neuron type. In this case the neuron is an Izhikevich neuron which has parameters a, b, c, d, and σ (for random Gaussian input current), and state variables u and v.

The network construction functions also support vector forms, so a number of neurons can be added with a single call:

```
# inhibitory neurons
ri = [random() for x in range(200)]
a = [0.02+0.08*x for x in ri]
b = [0.25-0.05*x for x in ri]
c = -65.0
u = [c*x for x in b]
net.add_neuron(n_iz, range(800,1000),
        a, b, c, 2.0, 2.0, u, c)
```

Like neurons, synapses are added either individually or in groups. Each connection is specified by its source and target indices, a conductance delay (in milliseconds), a weight, and a plasticity flag. For example, to fully connect the network created so far with randomly weighted synapses (plastic excitatory and static inhibitory) and random delays up to 20 ms:

```
targets = range(1000)

# excitatory connections
for source in range(0,800):
     weights = [0.5*random() for tgt
                in targets]
     delays = [randint(1,20) for tgt
                in targets]
     net.add_synapse(source, targets, delays,
             weights, True)

# inhibitory connections
for source in range(800, 1000):
     weights = [-random() for tgt in targets]
     net.add_synapse(source, targets, 1,
            weights, False)
```

A single configuration object is used to set simulation-wide parameters. For example, a global STDP function can be applied to all plastic synapses:

```
# create configuration object
conf = nemo.Configuration()

# set STDP function
ts = range(20)
prefire = [0.08 * exp(-t/20).real
            for t in ts]
postfire = [-0.1 * exp(-t/20).real
            for t in ts]
conf.set_stdp_function(prefire, postfire, 0,
      max_weight)
```

Finally, a simulation is created from a network and a configuration:

```
sim = nemo.Simulation(net, conf)
```

The user steps through the simulation, reading back the neuron firing at each step. For example, to run the simulation for one second:

```
for t in range(1000):
    fired = sim.step()
```

This facilitates fine-grained control over the simulation, enabling the user to provide stimulus to the network every time step—something that is particularly useful if the network is connected to real sensors or actuators. For example, to inject a random spike every time step:

```
for t in range(1000):
    fired = sim.step([randint(0,999)])
```

The state of the neurons in the network can be read back at any time. For example, to get a one second trace of the membrane potential for a single neuron (with index 0):

```
v = []
for t in range(1000):
    sim.step()
    v.append(sim.get_membrane_potential(0))
```

Other state variables and synaptic weights can be queried in a similar fashion.

More detailed information about using NeMo is available in its manual (Fidjeland 2011), which lists NeMo's dependencies and contains full instructions for compiling it on different platforms.

SpikeStream: 3D Visualization, Creation, Storage and Analysis of Neural Networks

Introduction

While large simulations are typically run without visualization for performance reasons, there is an important exploratory stage in almost every set of experiments in which different architectures and parameters are tested while the model is being set up. During this stage it is extremely useful to be able to visualize the behaviour of the model as it runs, and monitoring facilities are often convenient when working with external devices, such as robots.

SpikeStream (http://spikestream.sf.net) is a tool for the creation, visualization, storage and analysis of spiking neural networks, with a plugin architecture that enables it to wrap neural simulators, such as NeMo, and interface with external devices.[4] A previous version of SpikeStream that had an integrated simulator and device spike conversion interface was covered in previous work (Gamez 2007; Gamez et al. 2006). This paper describes the substantially rewritten new version, which is based around a plugin architecture that enables third-party libraries to be used for simulation and communication with external devices. The key features of SpikeStream are as follows:

- Written in C++ using Qt for the graphical user interface.
- Database storage.
- Sophisticated visualisation, editing, archiving and monitoring tools.
- Plugin architecture for all key functions, making it easy to customize and extend.
- NeMo wrapper for GPU and CPU simulation of Izhikevich neurons.
- iSpike wrapper for communication between simulated neurons and the iCub robot.

An overview of the SpikeStream architecture is given in Fig. 2. Detailed information about installing, compiling and using SpikeStream is available in the SpikeStream manual (Gamez 2011b).

Databases

SpikeStream is organized around a number of databases that hold the network model, archives of neuron firing patterns and the results of network analyses. The SpikeStream databases are as follows:

- *SpikeStreamNetwork*. Stores information about the networks, neurons and connections. The available neuron and synapse types and their parameters are also held in this database.
- *SpikeStreamArchive*. Stores neuron firing patterns for each network.
- *SpikeStreamAnalysis*. Holds the results of analyses of an archive of a particular network.

The relationships between these databases are shown in Fig. 3.

These databases are edited by the main SpikeStream application, and they can also be edited by third-party applications—for example, to create custom connection patterns or neuron arrangements—without affecting SpikeStream's ability to visualize and simulate the network. To speed up the creation and editing of networks when archiving is not required, SpikeStream can load networks in a 'prototype' mode in which the network is held in memory and optionally saved to the database at the user's request. Although SpikeStream does not currently support exporting or importing networks to and from XML formats, such as NeuroML, it would be relatively straightforward to write a plugin that could dump the databases to an XML file and upload a network from an XML file to the SpikeStream databases.

3D Graphical User Interface

SpikeStream has an intuitive graphical user interface with the following features (Fig. 4):

- *Editing*. Neuron and connection groups can be created and deleted.
- *3D Visualisation*. Neuron and connection groups are rendered in 3D using OpenGL. They can be rotated,

[4] Another application called SpikeStream was created after the one described in this paper and given this name independently: http://spikestream.bitbucket.org. 'SpikeStream' has also been used to name a component of the C2 simulator (Ananthanarayanan et al. 2009).

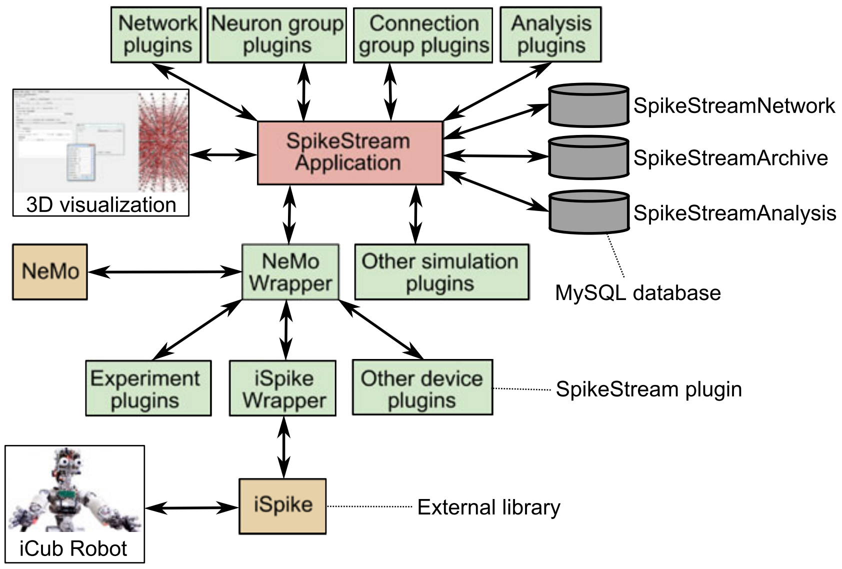


Fig. 2 SpikeStream architecture. The main SpikeStream application is run by the user on their local computer. This provides graphical tools to visualize the network and dynamically loads plugins (shown in *green* in the color version of this paper) that enable it to create, simulate and analyze networks and interface with external devices. One of these plugins interfaces with NeMo for network simulation; another interfaces with iSpike to communicate with the iCub robot. Networks, archived firing patterns and analyses are stored in three MySQL databases, shown on the *right*

selectively hidden or shown, and their individual details displayed. The user can drill down to information about a single synapse or view all of the connections simultaneously.

– *Simulation*. The simulation tab has controls to start and stop simulations, for example using the NeMo wrapper, and vary the speed at which they run. Neuron and synapse parameters can be set, patterns and external devices connected, and noise injected into the system.
– *Monitoring*. Firing and spiking patterns can be monitored, and variables, such as a neuron's membrane potential, graphically displayed.

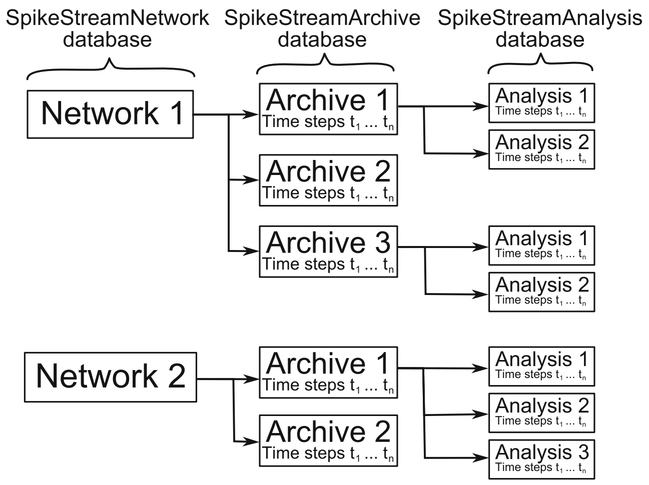


Fig. 3 Relationships between the SpikeStream databases. Each network is associated with multiple archives, which hold firing patterns of that network. Each analysis is specific to a particular archive and network

– *Archiving*. Archived simulation runs can be loaded and played back.

A full description of SpikeStream's many graphical features is available in the SpikeStream manual (Gamez 2011b).

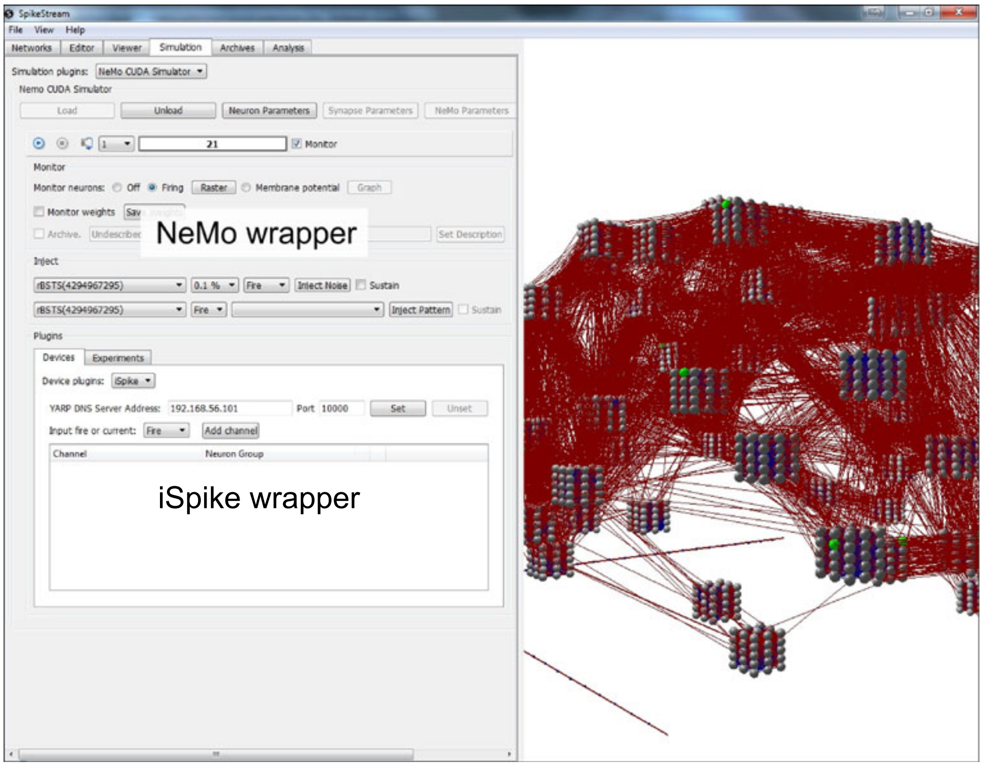
3D visualization of large networks is a computationally expensive operation that can severely compromise the simulator's ability to run in real time. To address this issue SpikeStream comes with a number of user-configurable performance optimizations:

– *Fast vs high quality render*. Neurons can be drawn as simple vertices or in a high quality mode using spheres. Connections can be drawn as lines (the colour indicates whether it is positive or negative) or as double cones whose width is proportional to the connection weight.
– *Connection thinning*. When the number of connections exceeds a user-configurable threshold, only a random selection of the connections are shown.
– *Simulation monitoring*. The NeMo wrapper has a variety of options for monitoring that can be selectively switched on. It is possible to monitor the state of every neuron and connection or to switch off all graphical monitoring during a simulation.

Measurements of SpikeStream's performance with different optimization strategies are given in Section "SpikeStream Performance".

Plugins

Most of the key functions of SpikeStream are implemented as C++ libraries that are dynamically loaded at runtime.



Simulation plugins: NeMo CUDA Simulator
Monitor neurons: Firing
Device plugins: iSpike
YARP DNS Server Address: 192.168.56.101 Port 10000
Input fire or current: Fire

Fig. 4 Screenshot of SpikeStream showing the 3D visualization, NeMo wrapper and iSpike wrapper. SpikeStream's graphical features are documented in detail in the SpikeStream manual (Gamez 2011b)

These plugins are compiled against the core SpikeStream binaries and a couple of simple functions need to be implemented to facilitate the dynamic loading.[5] The following sections give an overview of some of the plugins that are included with the SpikeStream release.

Network Plugins

A plugin can be used to create an entire network and its associated archives. This is useful when networks need to be constructed for a particular experiment and the neurons and connection patterns are known. Network plugins are also used to import networks from files created by other applications. For example, SpikeStream includes a plugin that imports networks from NRM files[6] and a plugin that constructs a network from a connection matrix by placing a group of neurons on each node and rewiring the connections with a probability that depends on the connection strength.

Neuron Group Plugins

Neuron group plugins are used add groups of neurons to the network. SpikeStream includes a neuron group plugin that enables the user to add cuboid neuron groups with a mixture of excitatory and inhibitory neurons.

Connection Group Plugins

This type of plugin enables the user to add connections between two neuron groups or within a single neuron group. The current plugins support the addition of random connections or a topographic 3D projection from one group to the centre of another, which can be spherical or cuboid, with hard edges or a Gaussian dropoff.

Simulation Plugins

Simulation plugins enable a network to be simulated, either directly in the plugin or by wrapping an existing third-party simulator. A wrapper plugin has been developed for NeMo (Section "NeMo: GPU and CPU Spiking Neural Simulator"), which enables the user to select the hardware, set the neuron and synapse parameters and run the simulation (Fig. 4). When NeMo is used with SpikeStream, the NeMo simulation is stepped in synchrony with the graphical interface and the update frequency varies with

[5]Instructions for writing SpikeStream plugins are given in the SpikeStream manual (Gamez 2011b).

[6]NRM stands for Neural Representation Modeller, a simulator of weightless neurons that was developed by Barry Dunmall and Igor Aleksander (Aleksander 2005).

the network activity and the level of graphical monitoring (Section "SpikeStream Performance"). Visualization of firing neurons, spike rasters and membrane potentials are supported, and the user can record firing patterns and inject noise or patterns into the network as the simulation is running. The SpikeStream NeMo wrapper also has its own plugins for carrying out *experiments* and for interfacing with *external devices*.

Experiment plugins are dynamically loaded C++ libraries that enable the user to carry out a particular experiment on a network within SpikeStream. The user can programatically step through the simulation, inject firing patterns and current into the network and access the spikes. A dialog is available that enables the user to select the experiment that they want to run and set the parameters. Template code for an experiment is included in the SpikeStream release.

Device plugins enable a network simulated within SpikeStream to communicate with an external device. At each time step, spikes generated by the network are passed to the device plugins, and spikes from the device plugins are passed to the network—causing current to be injected into the neurons or neurons to fire. Device plugins can be written to interface with any data source, such as a robot or financial data feed. SpikeStream currently includes a wrapper plugin for iSpike (Section "iSpike: Biologically-Inspired Spike Encoding and Decoding"), which enables it to communicate with the iCub robot.

Analysis Plugins

Analysis of networks for functional connectivity, effective connectivity or other properties plays an important role in network modelling linked to the brain (Sporns 2007). There has also been increasing interest in the analysis of networks for information integration, which has been claimed to be linked to consciousness (Tononi 2008). The current plugins enable the user to analyze networks of weightless neurons for state-based ϕ and liveliness (Gamez and Aleksander 2011).

iSpike: Biologically-Inspired Spike Encoding and Decoding

Introduction

iSpike (http://ispike.sf.net) is a C++ library that has been developed to interface between a spiking neural simulator and an external device or data source. A wrapper for this library has been created within SpikeStream (Section "Simulation Plugins"), which enables networks simulated using SpikeStream/NeMo to be connected to the iCub robot using the YARP robotics middleware platform—converting the robot's visual and proprioceptive data into spikes that are passed to the network in NeMo, and converting spikes from the network into motor values that are sent to control the robot. The YARP interface enables iSpike to communicate with a variety of other YARP-enabled robots and robot software, such as Player/Stage, Orocos, Urbi and ROS.[7] The iSpike library can also be used to convert other types of data, such as stock prices, to and from spikes, to enable them to be processed by a spiking neural network.

The central objective of iSpike is to provide an approximate imitation of the biological senses that can play a useful role in robotics and whose encoding and decoding of data is compatible with several interpretations of the neural code. The architecture of iSpike is shown in Fig. 5. A brief summary of the neural encoding and decoding is given in the following sections.

Encoding Visual Information into Spikes

iSpike's visual encoding is intended to provide a model of the spike encoding that occurs at the human retina, which is usable for robotics work, works in close to real time and is reasonably biologically realistic—it is not intended to be an exact replica of the human visual system. The first stage in the conversion of visual data into spikes is the resampling of the image to mimic the distribution of cones in the human retina.[8] The central part of the image is sampled on a 1-to-1 basis to replicate the high density of cones in the fovea and the parafoveal areas are sampled using a logarithmic function with a user-configurable base (a $\log(z+a)$ representation) (Bolduc and Levine 1998; Schwartz 1980). To match the mapping between the human retina and visual cortex (Grill-Spector and Malach 2004) the foveated image is output in log-polar format.

The next stage in the retinal processing is the simulation of colour-opponent ganglion cells using the difference-of-Gaussians method created by Enroth-Cugell and Robson (1966). For example, in the case of Red$^+$Green$^-$ (R$^+$G$^-$) cells, red and green versions of the foveated image are created, and then a Gaussian filter is applied using different

[7]Player/Stage: http://playerstage.sf.net; Orocos: http://www.orocos.org; Urbi: http://gostai.com/products/urbi/; Robot Operating System (ROS): http://www.ros.org. While iSpike's YARP interface should make it relatively straightforward to interface with these other systems, it has only been tested with the iCub robot.

[8]Rods are not included in the current implementation of iSpike because it is designed for daylight conditions, during which the light intensity is too high for rod photoreceptors to operate.

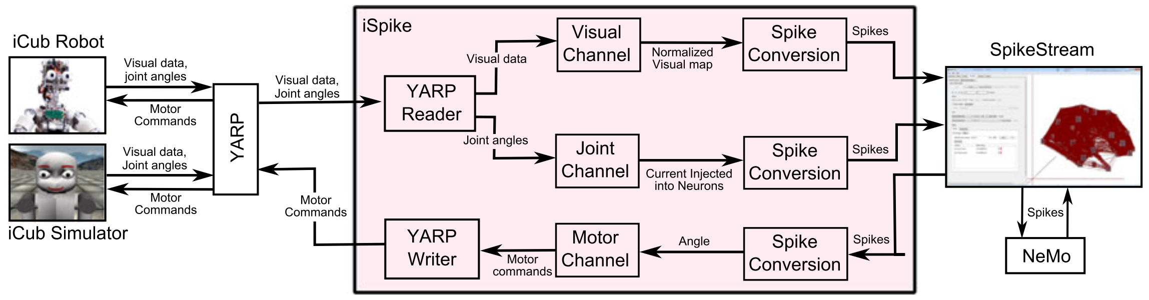


Fig. 5 iSpike architecture and its interface with YARP/iCub and SpikeStream/NeMo. The central part (in *pink* in the colour version of the paper) shows the internal iSpike components, including modules for converting data to and from spikes, pre-processing channels and the network interface with YARP. The *right side* of the figure shows the communication between iSpike and SpikeStream, which simulates the neural network using NeMo. The *left side* of the diagram illustrates how iSpike uses YARP to communicate with the real and virtual iCub

standard deviations for each image. The final R^+G^- image is obtained by subtracting the Gaussian green from the Gaussian red image. A similar procedure is used to generate Green$^+$Red$^-$, Blue$^+$Yellow$^-$ and Grey$^+$Grey$^-$ images.

The final stage of visual processing is the conversion of the foveated colour-opponent maps into spikes. In the retina the early stages of visual processing are analogue, with the final spiking output being generated by the ganglion cells. This is modelled by feeding the analogue visual maps into a two-dimensional array of Izhikevich neurons (Izhikevich 2003), with the current entering each neuron being proportional to the intensity of the corresponding pixel. This approach has the advantage that it is compatible with temporal, rank order and rate-based codes.[9] The stages in iSpike's visual encoding are illustrated in Fig. 6.

A basic model of the motion-sensitive magnocellular channel in the retina has also been created, which calculates a map of the difference between successive images (with optional logarithm), which is fed into a two-dimensional array of Izhikevich neurons. The changing parts of the image cause neuron activity, with moving objects typically producing the largest number of spikes around their leading and lagging edges where the difference between images is greatest.

Encoding Proprioceptive Data into Spikes

Most of the muscles in the human body contain specialized structures called muscle spindles that send information about muscle length to the brain, and it has been shown that the output from the spindles operates as a population code that accurately represents the direction of movement of a limb (Bergenheim et al. 2000; Jones et al. 2001; Roll et al. 2000, 2004) and its static position (Ribot-Ciscar et al. 2003), which is directly related to joint position. There also evidence that joint receptors (Ferrell et al. 1987; Macefield et al. 1990) and cutaneous receptors (Collins and Prochazka 1996; Collins et al. 2005; Edin and Johansson 1995) play a role in proprioception.

In the iSpike model, each item of proprioceptive data from the robot is converted into a vector of input currents by convolving it with a Gaussian function. These input currents are multiplied by a parameterized weight and fed into a population of neurons that span the range of possible values, producing a spike pattern that is delivered to the neural simulator. The use of currents pumped into a population of Izhikevich neurons is compatible with several interpretations of the neural code, since either the rate, temporal difference or rank order in which the neurons fire can be used to reconstruct the joint angle from the population .

Decoding Spikes into Motor Output

Biological motor control can be roughly divided into two components:

– *Voluntary muscle contraction*. Populations of neurons in the motor cortex specify the contraction of groups of muscles to move a part of the body, such as an arm or leg, into a particular position (Georgopoulos et al. 1986). Circuitry within the spinal cord ensures that antagonists of the contracted muscles are relaxed and maintains the body in a particular position.
– *Motor programs in the spinal cord*. Neural circuitry in the spinal cord executes sequential muscle contractions that lead to simple behaviours, such as the withdrawal reflex or running (Grillner et al. 2005). The brain does not control the order or nature of the individual commands, but sends a trigger to the spinal cord to indicate that a particular movement pattern should be carried

[9]The neuron that receives the most current will be the first to spike and it will also fire at the highest rate. The neuron with the most current will also be likely to fire before the one with the second highest current, and so on, producing a rank order code.

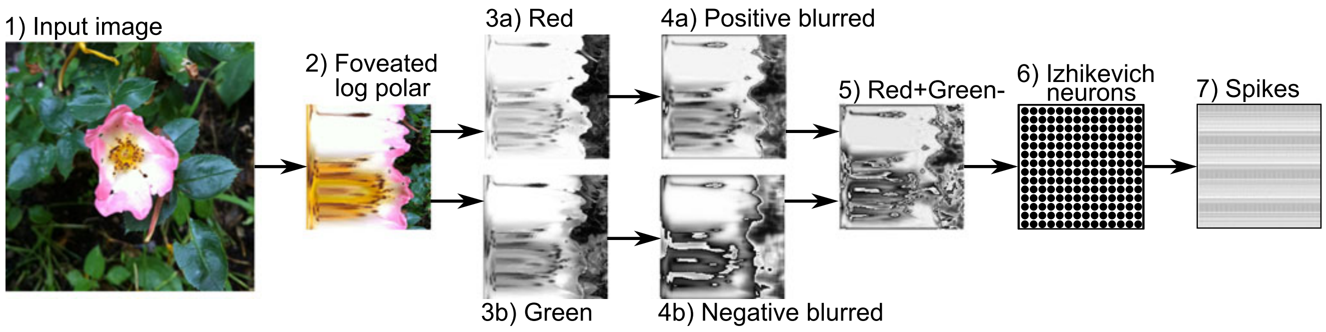


Fig. 6 Example of iSpike's visual processing stages. *1*) Input image; *2*) Input image converted into a foveated log$(z + a)$ representation in log polar format; *3a, b*) *Red* and *green* colour maps; *4a, b*) Colour maps are blurred by passing them through a Gaussian filter; *5*) Gaussian maps subtracted from each other to produce a $\mathrm{R}^+\mathrm{G}^-$ map; *6*) Analogue values in the $\mathrm{R}^+\mathrm{G}^-$ map are converted into spikes by feeding them into a two-dimensional array of Izhikevich neurons; *7*) Spikes from the neurons can be passed to a neural simulator

out. The spinal cord then executes the individual actions specified by the requested motor program.

In iSpike voluntary motor control is modelled using a population of variables, which span the range of a requested joint angle or other motor output value. These variables decay exponentially over time and are mapped onto a population of neurons in the simulator, so that each spike from a neuron in this population increases its corresponding variable by a fixed amount. The variables that receive spikes more often will have higher values and variables receiving no spikes will eventually decay to zero. The joint angle value is extracted from the variables by taking their weighted average.

iSpike Usage

Internally iSpike consists of the following components:

- *Reader*. Extracts sensory data of a given type from a given location. The current release of iSpike has readers that extract data from files, as well as readers that use YARP to access visual data and joint angles from the iCub robot.
- *Writer*. Outputs data of a given type to a given destination. The current release has writers that output data to a file, as well as writers that send motor commands to the iCub robot using YARP.
- *Input Channel*. Receives sensory data from a Reader and transforms it into a spike representation that can be passed to a neural simulator.
- *Output Channel*. Receives a spike pattern from the neural simulator and uses a Writer to deliver it to its destination.

To use iSpike to convert real valued data into spikes, one starts by creating a reader, which interacts with YARP or reads data from a file. For example, the following code (in pseudo-Python—the real API is in C++) creates a reader that loads an image from the file 'input.ppm'. Next a visual input channel is created that converts the image provided by the reader into spikes.

```
visual_reader = FileVisualReader(
        filename='input.ppm')
visual_channel = VisualInputChannel(
        reader=visual_reader,
        fovea_radius=50)
```

The spike conversion process depends on simulated Izhikevich neurons, which are stepped with a temporal resolution of 1 ms per time step in synchrony with the neural network simulator. At each time step the spikes resulting from the conversion of the visual image are extracted and passed to the neural simulator:

```
for t in range(1000): # 1 second
  visual_channel.step()
  fired = visual_channel->get_firing()
  sim.step(fired) # pass spikes to
    neural simulator
```

The construction of writers and output channels works in a similar way to the construction of readers and input channels. The key difference is that spikes from the neural simulator are passed into iSpike at each time step:

```
for t in range(1000): # 1 second
  fired = sim.step()
  motor_output_channel->set_firing(fired)
```

More detailed information about using iSpike is available in the iSpike manual (Gamez 2011a), which lists iSpike's dependencies and contains full information about compiling iSpike on different platforms.

Performance and Accuracy Measurements

NeMo Performance

The performance of NeMo (v0.7.1) was measured using a network whose properties were varied using parameters controlling the number of neurons (p), target synapses per neuron (m), and a locality parameter (σ). This network was a ring torus composed of an integer number of cylindrical patches of 32×32 neurons evenly spaced on a grid. The torus diameters were therefore 32 and $32p$, and the number of neurons in each network was $1024p$. Each patch consisted of 80 % excitatory and 20 % inhibitory neurons, which were placed randomly on the grid. The neuron parameters, including the random input current, were taken from Izhikevich (2003).

The connection distances (2D Cartesian distance along the torus surface) between neurons were normally distributed, as suggested by Hellwig (2000), using $\mathcal{N}(0, \sigma)$ for excitatory connections and $\mathcal{N}(0, 16)$ for inhibitory connections. The network parameter σ thus controlled the locality of excitatory connections in the network with self-connections being disallowed. Conductance delays were scaled linearly with distance, so that delays were in the range 1–20 ms. Excitatory weights were uniformly random in the range (0.0, +0.5), and inhibitory weights were uniformly random in the range (0.0, −1.0). All neurons received normally distributed random input current with different parameters for excitatory and inhibitory neurons, and the network firing rate was constant regardless of the network size. For most experiments the random input currents were drawn from $\mathcal{N}(0, 5)$ (excitatory) and $\mathcal{N}(0, 2)$ (inhibitory), resulting in a mean firing rate of around 7.3 Hz.

The two main parts of any spiking neuron simulation are the neuron state update and the spike delivery, the relative importance of which changes with the ratio of neurons to synapses and the connection density—synapses with similar characteristics (nearby targets, same conductance delay) can be processed simultaneously, leading to higher throughput for more densely connected networks. Simulator performance also changes with the network activity—more activity entails a higher work load as more synaptic events need to be processed—which is also linked to the balance between neuron update and spike delivery.

We performed four sets of experiments to assess NeMo's performance on different networks. The first experiment varied the number of neurons in the network. The second measured system performance for a network with a constant number of neurons (30K), with the number of synapses per neuron being varied from a very low value of 100–5,000, which is approximately the connection density found in parts of the mammalian brain (Binzegger et al. 2004). The third experiment varied the firing rate in the network while keeping the neuron count (30K) and number of synapses per neuron (1,000) constant. The firing rate was altered by varying the amount of random Gaussian current in the excitatory neurons. Variations in standard deviation in the range 3–20 resulted in firing rates ranging from 0.6 to 51.6 Hz.

The fourth experiment varied the frequency of weight update when STDP was enabled to asses the cost of accumulating the eligibility trace and the weight update. These STDP experiments used the toroidal network described above with $p = 30$ and $m = 1{,}000$. All excitatory synapses were plastic and all inhibitory synapses were static. The STDP function was $\alpha \exp(\Delta t/\tau)$. For pre-post pairs $\alpha = +1.0$, for post-pre pairs $\alpha = -0.8$, and $\tau = 20$ was used for both. We simulated the network for 60 s and updated the weights with periods of 60 s (i.e., once), 1,000 ms, 100 ms, and 10 ms. To eliminate performance artifacts arising from an alteration in network dynamics due to weight change, we used a modified version of the simulator, which accumulated the eligibility trace correctly, but updated the weights to the same value that they had before.

NeMo's performance was measured using two metrics: throughput in terms of millions of post-synaptic potentials per (wall-clock) second (MPSP/s), and speedup with respect to real time. A user will typically want to know the latter measure, but as the sole measure of performance it suffers from several drawbacks. The main issue is that speedup is highly sensitive to both the firing rate and density of connectivity in the network, and it does not necessarily reflect the effect of these two factors on the most demanding part of the simulation, namely the spike delivery. A simulation with a low firing rate will report a lower speedup than one with a high firing rate, even though the simulator capabilities are the same for the two simulations.

All experiments were run on three NVIDIA CUDA-capable GPUs and one Intel multi-core processor (Table 2). The Tesla C2070 and GTX580 are from the same generation of GPUs, whereas the Tesla C1060 is from the previous generation. The C2070 and C1060 are high-performance computing (HPC) grade devices, whereas the GTX580 is aimed at the consumer market. The HPC-grade devices have more memory, additional error checking, stricter fabrication quality, and are considerably more expensive. In Table 2, the number of cores refers the number of traditional Intel cores and to the number of CUDA multiprocessors. The number of hardware threads refers to the number of hyperthreads on the CPU and the number of floating point units on the GPU. The number of software threads that are active on the GPU is much larger: 256 per multiprocessor.

To compare NeMo's performance with other simulators we used the throughput and speedup measures from the above experiments. We chose not to compare NeMo's performance with all of the other GPU-based simulators in the literature, because many of them were designed to solve

Table 2 Devices used for NeMo performance measurements

Device	GTX580	C2070	C1060	i7-950
Cores	16	14	30	4
Hardware threads	512	448	240	8
Software threads	4,096	3,584	7,680	8
Clock (GHz)	1.544	1.15	1.3	3.07
Memory				
Capacity (GB)	1.5	6	4	24
Bandwidth (GB/s)	192.4	144	102	25.6

different problems. For example, the simulator of Yudanov et al. (2010) is not a performance-oriented simulator as its main focus is on accurate numerical integration during the neuron update. The simulator of Han and Taha (2010) was also not appropriate for comparison because it only works with networks that have a small number (on a per-neuron basis) of highly regular connections without delays. This left GPU-SNN (Nageswaran et al. 2009; Richert et al. 2011) as the most suitable comparison. This simulator fills much the same niche as NeMo, has a similar feature set and it targets the same type of unconstrained networks. In these experiments we compared NeMo with the version of GPU-SNN described in earlier work (Nageswaran et al. 2009) because this is reported to have higher performance (due to a more restricted plasticity model) than the later version. For GPU-SNN we ran a series of experiments mimicking the synthetic torus network described above, with some changes to accommodate the slightly different programming interface. The most notable change was a maximum neuron out-degree of 500. We compared networks with static synapses only, so we were primarily measuring the performance of the spike delivery algorithm.

NeMo Accuracy

Accuracy tests on different neural simulators are unlikely to produce spike-for-spike equivalence, as tiny variations in neuron state resulting from minor implementation differences quickly cause variations in spike timing, leading to divergent network behaviour. However, different simulators should produce similar large-scale results on networks with stable dynamics. A number of benchmark networks for comparing neural simulators have been put forward by Brette et al. (2007), based on the Vogels–Abbot network (Vogels and Abbott 2005). Our accuracy tests on NeMo used their second benchmark, which consists of 4,000 current-based leaky integrate-and-fire neurons that exhibit self-sustained irregular activity. The benchmark code was taken directly from the PyNN source code, which was modified to use a larger time step and minimum delay (both 1 ms) to match the minimum requirements of NeMo. Using this benchmark we compared the output from both the CPU and GPU backends of NeMo with NEST (Gewaltig and Diesmann 2007) and Brian (Goodman and Brette 2009), using the same random seeds in all cases. The spike times were recorded and a set of per-neuron inter-spike intervals computed.

SpikeStream Performance

The ability of SpikeStream to visualize networks in 3D and display neuron firing patterns, membrane potentials and connection weights is limited by the CPU and GPU processors and the system and graphics memory. System and graphics memory also limits the size of the network that can be simulated using SpikeStream because NeMo constructs a copy of the network in system RAM, which is loaded onto the graphics card.

To measure the performance of SpikeStream with different levels of visualisation and monitoring, two networks were created: one with 2,970 neurons and 133,650 connections, the other with 14,256 neurons and 3,079,296 connections. These networks were distributed over 66 nodes, with the number of excitatory connections between nodes being proportional to the connection weights in the matrix developed by Hagmann et al. (2008). A 4:1 mixture of excitatory and inhibitory neurons was used, and the parameters were taken from Izhikevich (2003). Neurons in the small network fired at an average rate of 10 Hz and in the large network they fired at approximately 77 Hz. When learning was used the connection weights were updated every 100 ms.

These two networks were simulated with the levels of monitoring shown in Tables 3 and 4. The update frequency was measured for 10 runs of 1,000 time steps, and the results are given in Section "SpikeStream Performance". The measurements were carried out on a 32 bit Windows system with a NVIDIA Quadro FX 580 (512 MB graphics memory) for the visualization, a Tesla C1060 (4 GB graphics memory)

Table 3 Levels of monitoring for network with 2,970 neurons and 133,650 connections

Experiment	1	2	3	4	5	6	7	8	9	10	11	12
Monitor time step	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Learning (every 100 ms)	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
Monitor neuron firing	✗	✗	✗	✓	✓	✗	✗	✓	✓	✓	✓	✓
Monitor membrane potential	✗	✗	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗
Draw connections	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓
Update connection weights	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓
Full render neurons	✗	✗	✗	✗	✓	✗	✓	✗	✓	✓	✓	✓
Full render connections	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Table 4 Levels of monitoring for network with 14,256 neurons and 3,079,296 connections. No learning, connection visualization or monitoring was used

Experiment	1	2	3	4	5	6
Monitor time step	✗	✓	✓	✓	✓	✓
Monitor neuron firing	✗	✗	✓	✗	✓	✗
Monitor membrane potential	✗	✗	✗	✓	✗	✓
Full render neurons	✗	✗	✗	✗	✓	✓

for the network simulation, a Xeon quad core 2.4 GHz CPU, and 4 GB of system RAM.

iSpike Performance

The performance of iSpike's proprioceptive encoding was measured by encoding a joint angle with neuron populations ranging from 10 to 100 neurons, with the encoding time per time step averaged over ten runs of 100,000 time steps. The time taken to encode the angle was measured independently of the time taken to communicate with the robot or other device. A similar approach was used to measure the performance of the decoding of motor output from spikes.

With visual encoding one of the main computational costs is the calculation of the visual maps whose pixel values are fed into the two dimensional array of neurons to perform the spike conversion (Section "Encoding Visual Information into Spikes"). iSpike optimizes this process by calculating the maps only when a new image arrives. The cameras in the iCub robot operate at 30 frames per second (Metta et al. 2008), which should lead to the visual maps being recalculated approximately every 33 ms. This intermittent recalculation of visual maps is biologically realistic because the human eye moves in a series of rapid movements known as saccades, which fixate for 20–200 ms on a particular part of the visual field (Jiirgens et al. 1981; Robinson 1964). This suggests that it would not be necessary to update the visual maps more than once every 20 ms to produce a high level of biological accuracy.

The speed of visual processing was tested using images with resolutions of 640 $\times$ 480 (the same as the iCub cameras) and 200 $\times$ 200. The first image was resampled by iSpike down to a 200 $\times$ 200 image using the $\log(z + a)$ method with a central fovea area of 50 $\times$ 50; the second image was resampled by iSpike down to 50 $\times$ 50 with a fovea of 20 $\times$ 20. The average spike encoding time per time step was measured over ten runs of 1,000 ms with the visual maps being recalculated every 25, 50, 100, 200, 300 and 1,000 time steps. These experiments were carried out on a Windows system with 4 GB of RAM and an AMD Phenom 6-core 2.8 GHz CPU.

iSpike Accuracy

The accuracy of iSpike's proprioceptive encoding and motor decoding was measured by converting a series of proprioceptive values into spikes and converting the spikes back into motor output. This should identify errors introduced by noise and pick up most of the deficiencies in the encoding and decoding processes.

To evaluate whether the whole angle space is encoded with the same quality, we encoded and decoded a large number of angles while keeping the number of neurons/receptors constant (25 or 100) and a fixed normal distribution for the receptive field ($\sigma = 0.5$ neurons). The angle space was sampled at half degree intervals and the combined encoding/decoding error was measured as the system adapted to an instantaneous change from an initial angle of 0 to the new angle, measured over a period of 50 ms. We also explored the relationship between the size of the encoding/decoding population and the accuracy by measuring the combined encode/decode error for a varying number of neurons ranging from 2 to 100. The effect of the neurons' receptive fields on accuracy was evaluated by calculating the encode/decode root mean square error for a population of 50 neurons with standard deviation (in neurons) ranging from 0.05 to 25, averaged per time step over 50 runs with different initial values drawn from a uniform distribution.

Results

NeMo Performance

The results from the first experiment on NeMo's performance shown in Fig. 7 indicate that the GTX580 achieves the highest performance both in terms of throughput (up to 1,450 MPSP/s) and the highest speedup of the GPUs (2$\times$ wrt. real time). However, the network size does not scale as far as with the other GPUs, due to limited memory. The shapes of the curves for the GPU backends indicate that they have similar performance characteristics, with peak throughput only being achieved for networks of around 50K neurons or more. With further increases in network size the throughput stays fairly constant while the speedup decreases. The shape of this performance curve is explained by the partial utilization of the computational resources for small networks. Since the networks are split into 1K-neuron sized partitions, tens of partitions are needed for all cores to be active. Even with all cores active, the load may not be fully balanced, and so additional increases in the network size increases throughput further. The CPU backend balances the load between the cores, even for small networks, and so the throughput is invariant with respect to network size.

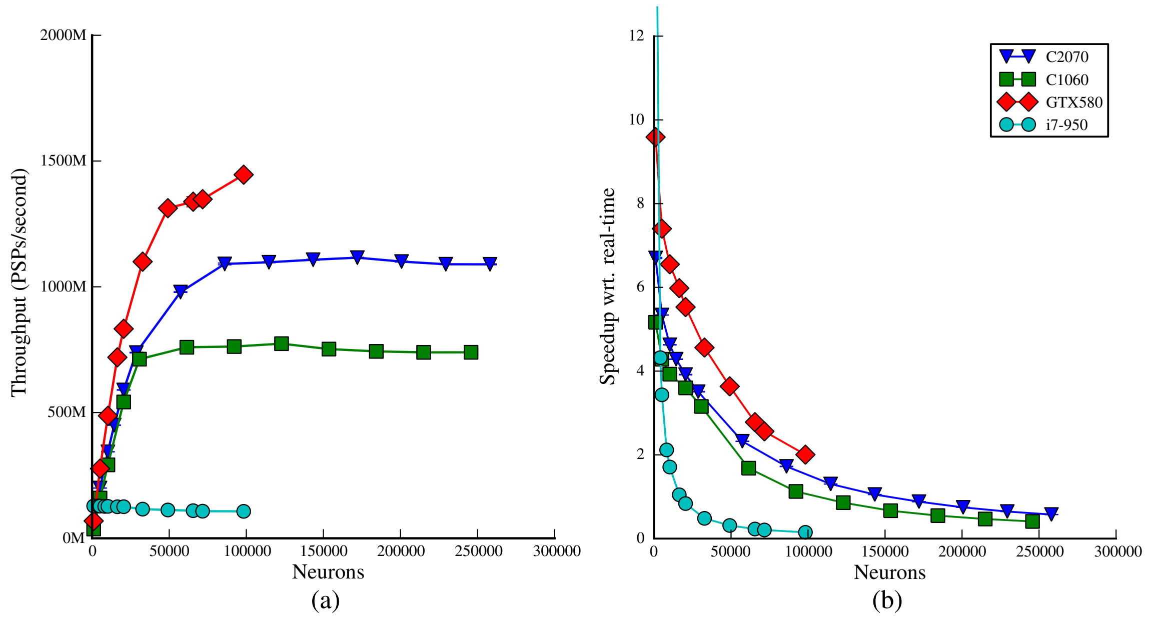


Fig. 7 System performance for variations in the number of neurons for toroidal networks with 1000 synapses per neuron showing **a** throughput in terms of post-synaptic potentials per second and **b** speedup with respect to real time for simulation with 1 ms temporal resolution

The speedup results shown in Fig. 7b demonstrate that most of the networks run faster than real time, with all of the networks running on the GTX580 at greater than real-time, and the C2070 and C1060 GPUs simulating networks in real time for up to 140K and 90K neurons. The speedup of the CPU backend is very high for small networks, but decreases exponentially, with real-time performance achieved on networks with up to 16K neurons.

The results from the second experiment, which varies the level of connectivity, are given in Fig. 8. This shows that for sparsely connected networks the throughput is low, and throughput peaks for networks where neurons connect to 2,000 or more other neurons. NeMo's spike delivery algorithm delivers spikes to nearby synapses (in both space and time) in parallel. For low levels of connectivity there are few opportunities for such parallelism, and then beyond a

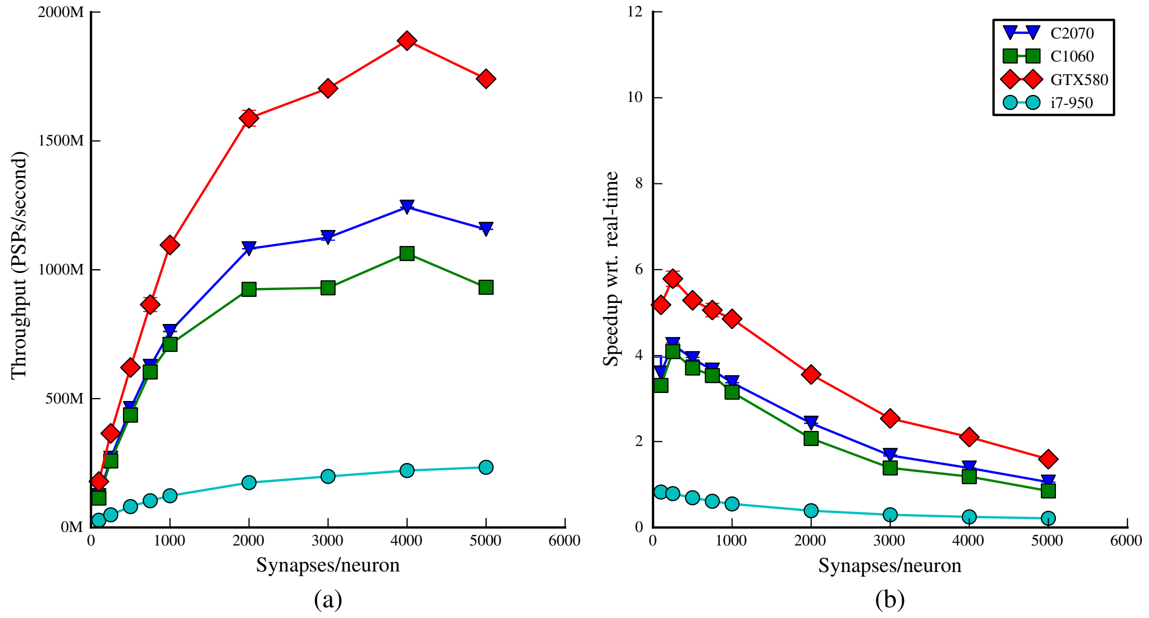


Fig. 8 System performance for variations in the number of synapses per neuron for toroidal networks with 30K neurons showing **a** throughput in terms of post-synaptic potentials per second and **b** speedup with respect to real time for simulation with 1 ms temporal resolution

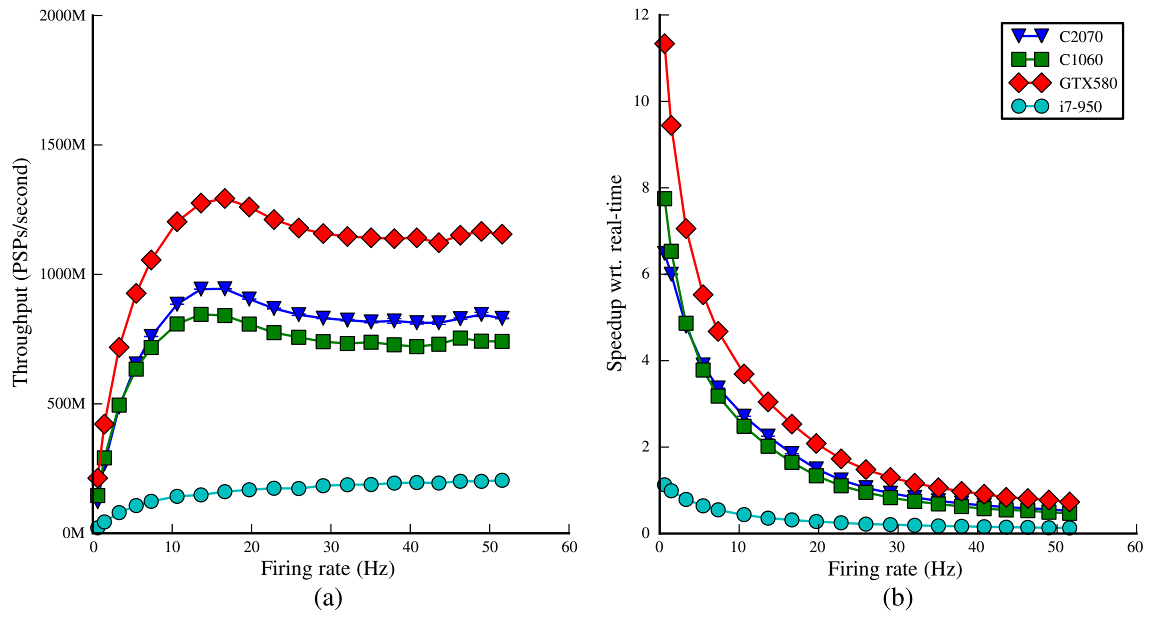


Fig. 9 System performance for variations in network activity for toroidal networks with 30K neurons and 1000 synapses/neuron showing **a** throughput in terms of post-synaptic potentials per second and **b** speedup with respect to real time for simulation with 1 ms temporal resolution

certain point this form of parallelism is fully exploited, which explains the flattening of the throughput curve.

Figure 9 shows both throughput and speedup for networks with different firing rates. Again the GTX580 displays the highest performance, with the throughput peaking and stabilising after a firing rate of ∼10 Hz. Throughput is generally high and stable for high firing rates, but low firing rates lead to low throughput. The latter observation is unsurprising since the simulation becomes dominated by the neuron update, which is not directly measured by the throughput. For similar reasons, speedup falls exponentially with increasing firing rate.

Figure 10 shows that when STDP is enabled the performance drops by around 50 % compared with static synapses. The leftmost value in each of the plots (labelled ∞) shows the performance for the case where STDP is not used, and

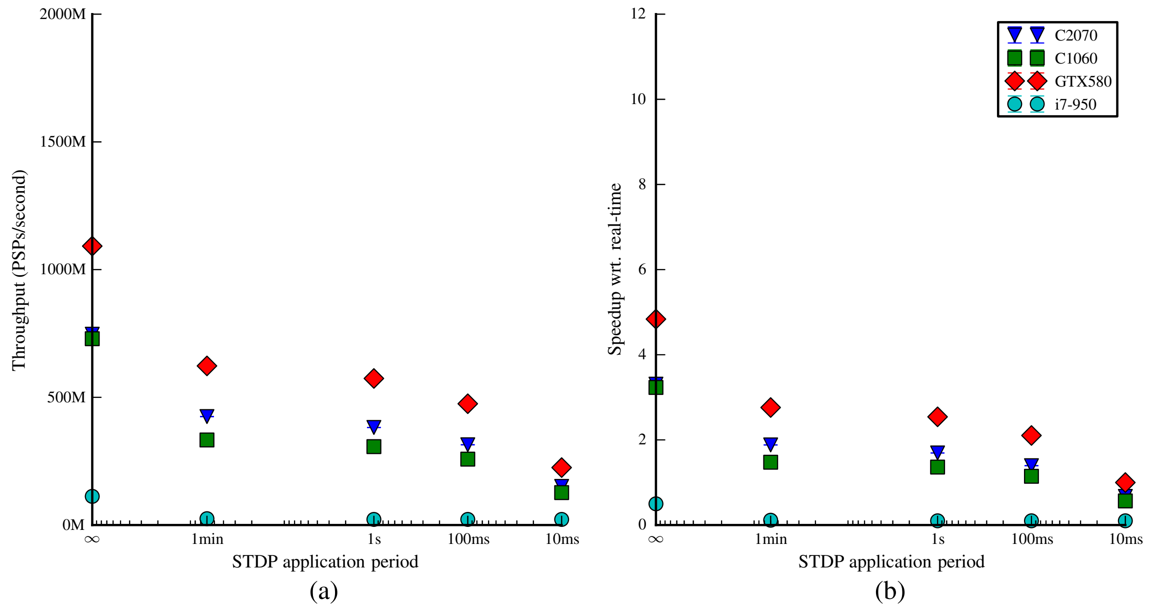


Fig. 10 System performance for variations in STDP application period for toroidal networks with 30K neurons and 1000 synapses/neuron showing **a** throughput in terms of post-synaptic potentials per second and **b** speedup with respect to real time for simulation with 1 ms temporal resolution

the next value shows the performance when the weights are updated exactly once during the 60 s simulation. This is dominated by the accumulation of the STDP eligibility trace and the two leftmost values show that this slows the simulation down by a factor of around two. This is unsurprising as there is roughly twice the amount of work to be done for each spike. The update of the weights is also a costly operation that involves the whole connectivity matrix. This has negligible impact when the weights are updated every second, but it becomes significant when the weights are updated every 10 ms. This indicates that the current algorithm is not ideal if frequent weight updates are required.

Spiking neural network simulation is a memory-bound problem, with only a small amount of computation being required for each synapse-related memory operation. The available memory bandwidth and the degree to which this can be exploited will therefore largely determine the achievable performance, and the differences in performance between the three GPU architectures can be largely explained by differences in their memory bandwidth. Figure 11 shows the memory bandwidth utilisation as measured by the number of PSPs per word of available memory bandwidth for the experiments above with variations in neuron count. For the networks which saturate the device we find very similar figures. Note that while the utilisation figures may seem low, the memory bandwidths used in the computations are the maximum *theoretical* values, while the actual number of memory operations required for a PSP is

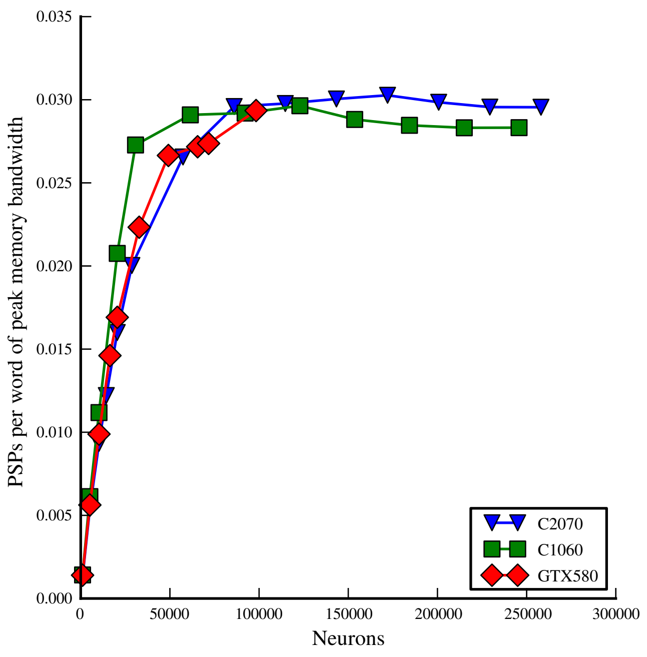


Fig. 11 Utilisation of available memory bandwidth as measured by number of PSPs per word of available memory bandwidth for experiments with variations in number of neurons

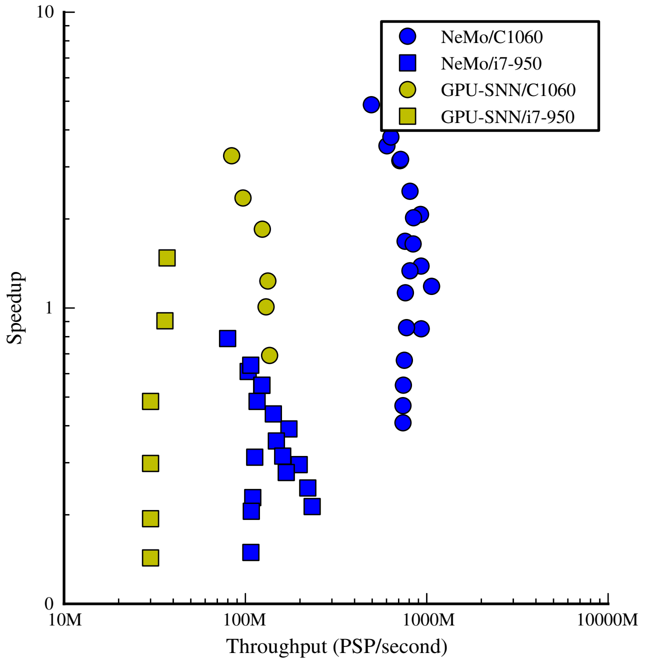


Fig. 12 Comparison of NeMo with GPU-SNN using both a GPU and a CPU for a range of different networks

greater than one word. The current version of NeMo uses the same algorithm on all of the different GPU architectures. This algorithm assumes that the amount of scratchpad memory available is that which is found on the older Tesla device (C1060). Additional performance gains should be possible for the newer 'Fermi' devices (GTX580 and C2070) where there are additional possibilities for exploiting data locality, as these devices have a larger scratchpad memory and an additional L1 cache.

NeMo performs well when compared with the GPU-SNN simulator. Figure 12 shows the throughput and speedup of a range of simulations of NeMo and GPU-SNN (Nageswaran et al. 2009) running on the C1060 GPU and the i7-950 CPU. The speedup of GPU acceleration is similar for the two simulators, but NeMo has around six times the throughput of GPU-SNN.

NeMo Accuracy

In the accuracy experiments the GPU and CPU versions of NeMo showed similar behaviour to Brian and NEST on the benchmark network (Fig. 13). To quantify this further we constructed the empirical cumulative distribution functions (CDFs) for the inter-spike interval histograms (ISIs) and computed the maximum absolute difference between them (i.e. the test statistic from the Kolmogorov–Smirnov test). Table 5 shows that the two NeMo backends produced identical results for this network, with the small differences

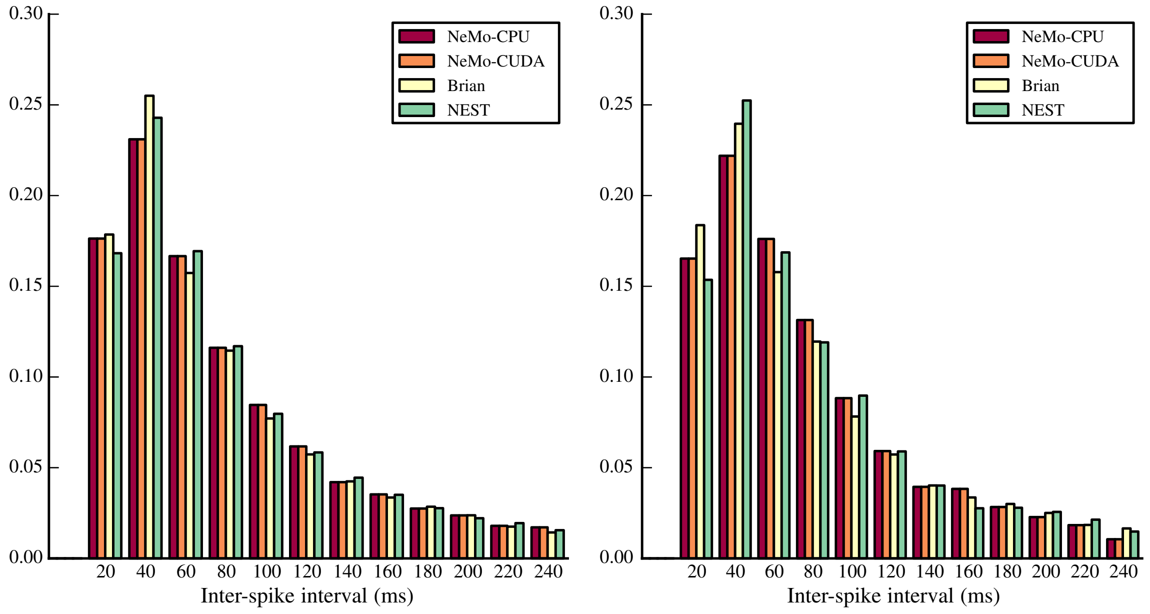


Fig. 13 Inter-spike interval histograms for **a** excitatory and **b** inhibitory neurons from the same simulation running on three simulators

between NeMo and Brian and NeMo and NEST being of the same order as the difference between Brian and NEST.

SpikeStream Performance

The results shown in Fig. 14 indicate that with all monitoring switched off SpikeStream has relatively little impact upon NeMo's performance, and it has a severe impact when all monitoring is switched on. Even simple graphical operations, such as time step monitoring, slowed the simulation down by a factor of ∼20, and visualization of firing neurons or membrane potentials reduces the performance of the larger network to around 70× real time.

Table 5 Pairwise absolute difference between empirical ISI CDFs for both excitatory and inhibitory neurons populations

	NeMo-CPU	NeMo-CUDA	Brian	NEST
Excitatory				
NeMo-CPU		0.0000	0.0252	0.0099
NeMo-CUDA	0.0000		0.0252	0.0099
Brian	0.0252	0.0252		0.0268
NEST	0.0099	0.0099	0.0268	
Inhibitory				
NeMo-CPU		0.0000	0.0376	0.0202
NeMo-CUDA	0.0000		0.0376	0.0202
Brian	0.0376	0.0376		0.0270
NEST	0.0202	0.0202	0.0270	

While real time monitoring of a simulation running at 1,000 Hz would be a nice option to have, in practice most of this data would be invisible on a monitor refreshing at 60 Hz, and it would be impossible for human users to process information at this speed. Instead, it is anticipated that SpikeStream will be used to construct and visualize networks and carry out preliminary tests with full monitoring to understand the networks' behaviour. The monitoring will then be switched off to enable the robotics experiment to be carried out in real time. The most useful form of monitoring is the neuron firing and the results suggest that this can be carried out at 50 Hz for a network of ∼3,000 neurons and at 14 Hz for a network of ∼14,000 neurons, which is fast enough for the projected applications of this software.

iSpike Performance

The iSpike performance results for the encoding and decoding of joint angles in Fig. 15a demonstrate that iSpike is capable of encoding all of the iCub's 53 degrees of freedom using populations of 20 neurons faster than real time. It can also convert spike information into motor output for all 53 degrees of freedom using populations of 100 current variables significantly faster than real time.

The vision results in Fig. 15b are less impressive, with it taking an average of 35 ms processing time per 1 ms simulation time step to convert the 640 × 480 image (resampled by iSpike down to 200 × 200) into spikes with a map recalculation every 25 time steps, i.e. an update frequency of around 29 Hz. The figure for 1,000 time steps per image update indicates that around 19 ms of the processing time per

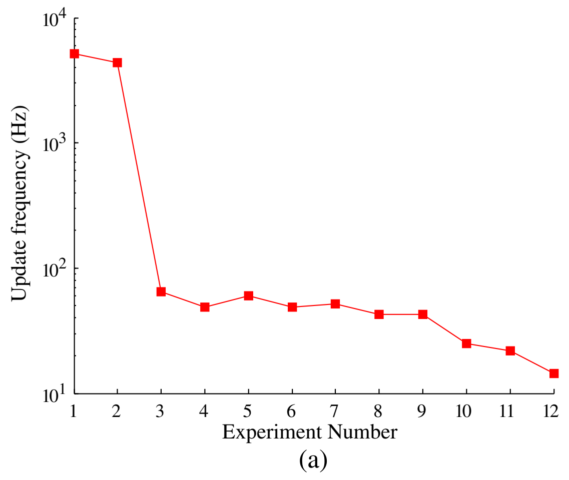


Fig. 14 **a** Update frequency for the network with 2,970 neurons and 133,650 connections, with the different levels of optimization shown in Table 3; **b** Update frequency for the network with 14,256 neurons and 3,079,296 connections, with the different levels of optimization shown in Table 4

simulation time step is due to the update of the 40,000 neurons that are used to convert the pixel values into spikes, with the remaining time being due to the visual maps recalculation. The results for the 200 × 200 image (resampled by iSpike down to 50 × 50) were more promising, running at approximately 1.7× real time with a visual map recalculation every 50 time steps.

iSpike Accuracy

The results of the accuracy experiments are shown in Fig. 16. When a small number of neurons was used, there was some periodicity in the result, which was reduced by increasing the size of the neuron population. The results also show that there was a delay between a change in the input

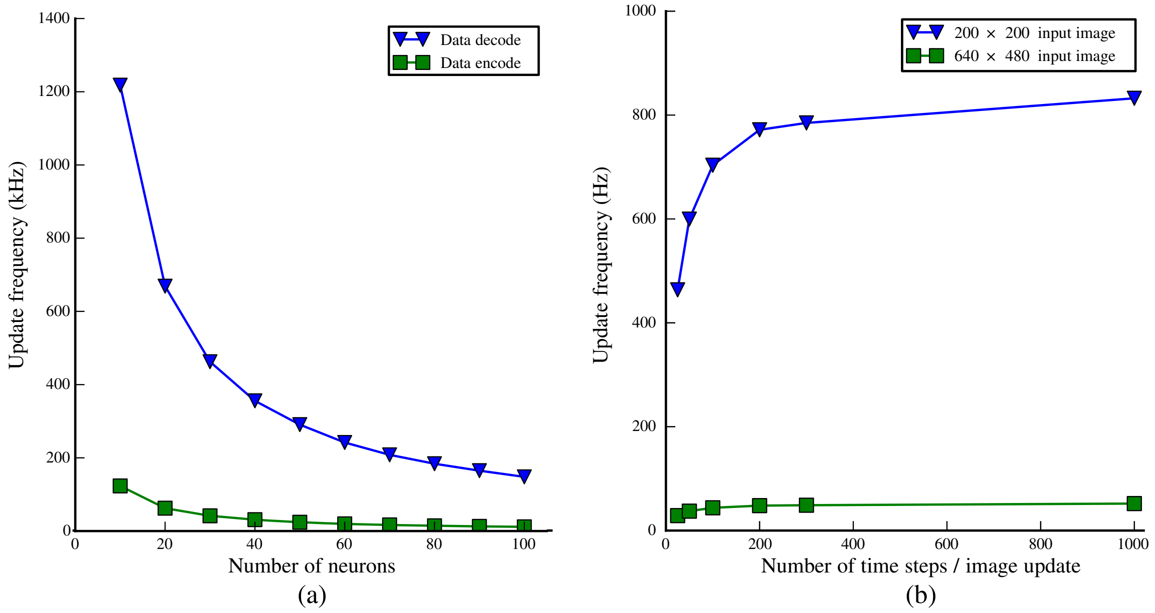


Fig. 15 Encoding and decoding performance of iSpike. **a** Update frequency for the encoding/decoding of joint angles using different numbers of neurons/current variables; **b** Encoding of visual information for two different images, with the visual maps being recalculated after different numbers of time steps. The duration in simulation time of each iSpike time step is 1 ms

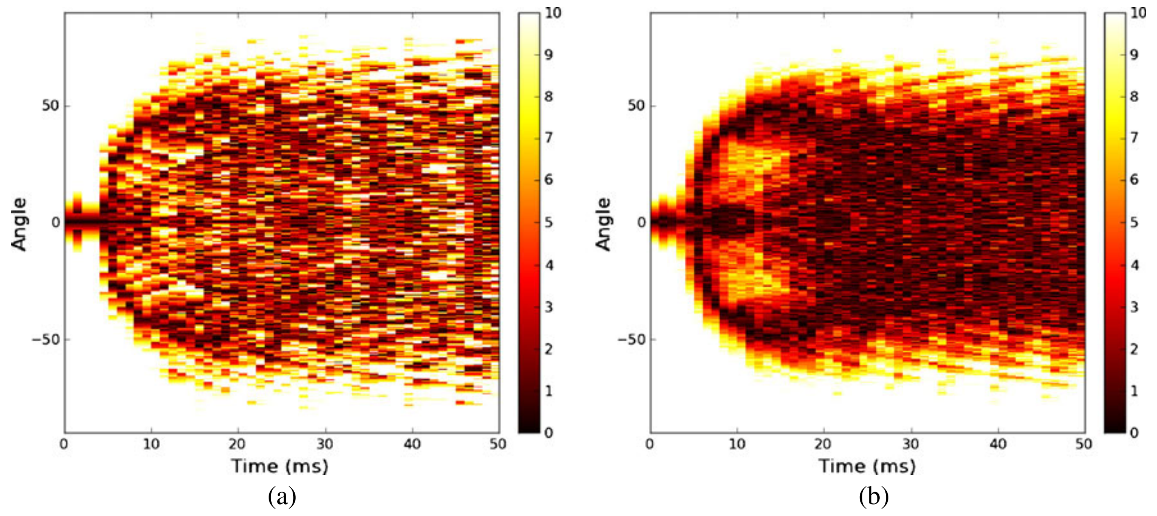


Fig. 16 Encoding and decoding accuracy (absolute error in degrees) for different parts of the angle space, when transitioning from an initial angle of zero. **a** 25 neurons in population; **b** 100 neurons in population

angle and a corresponding change in the decoded angle. When converting from spikes to angles each spike had an effect over a time period (due to the exponential decay), causing a delay before the new spike pattern dominated. The delay before a stable value was achieved increased with the magnitude of the change in angle, but even large changes in input gave errors of less than five degrees after 10 ms and less than one degree after 15 ms.

Figure 17a shows the root mean square error in degrees over time for a number of runs using different population sizes. The error for each time/size pair was averaged over 40 runs with transitions between pairs of uniformly random angles in the range [−45°, 45°]. The figure shows accuracy improving both with increasing population size and with time. Populations of less than ten neurons displayed error rates in excess of 10 degrees, whereas the error of a population size of 50 was generally less than one degree. For large populations the error settles at this low value after approximately 20 ms. Figure 17b shows that fairly narrow fields produce better results.

In previous work we used a normalized measure of the quality of reconstruction known as the coding fraction to evaluate the encoding accuracy of joint angles that varied dynamically with frequencies ranging from 1 to 20 Hz (Gamez et al. 2012). These experiments demonstrated that iSpike had poor reconstruction quality for populations of two and three neurons, whereas the reconstruction quality of populations of five or more neurons remained consistently high, with low frequency signals being reproduced with greater fidelity than higher frequencies. There was also

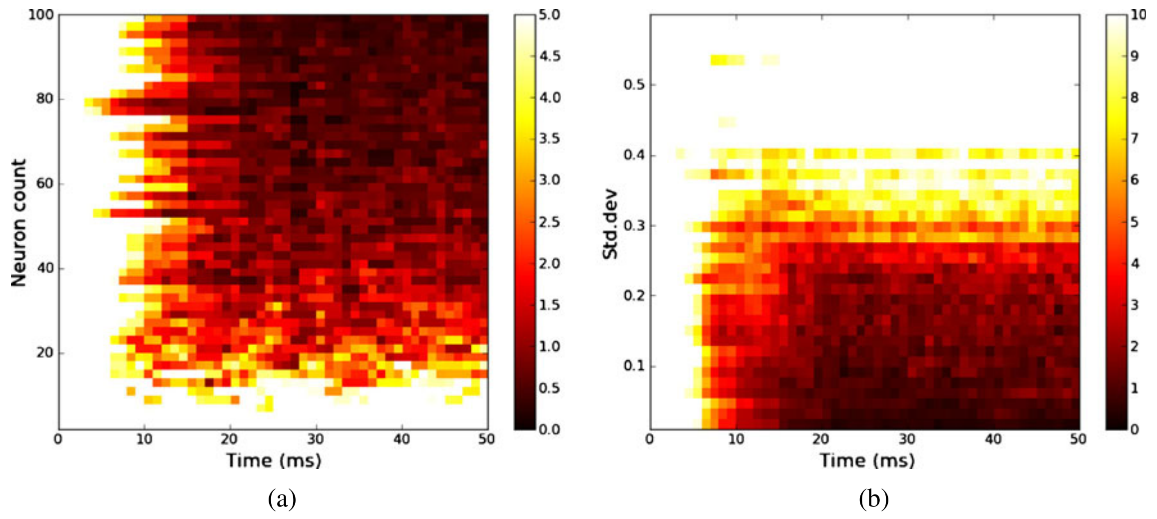


Fig. 17 Accuracy of encoding/decoding measured by root mean square error (in terms of angle) for **a** variations of population size, averaged over angles, and **b** variations in receptive field size, averaged over angles

some delay in the response of the encoding and decoding populations to changes in the signal, which introduced greater errors for higher frequencies.

Discussion and Future Work

The results presented in Section "Results" show that the tools described in this paper are capable of simulating networks with tens of thousands of point neurons and millions of connections that can interface with robots in close to real time. The NeMo simulator outperforms its closest competitor for static networks and has similar accuracy results to other simulators for networks of the same class. While NeMo's restriction to point neurons limits its networks' biological plausibility, more accurate models of tens of thousands of compartmental neurons cannot currently be simulated in real time, and they are typically run on supercomputers that are unavailable to the majority of researchers (Markram 2006). As GPUs increase in performance it might become possible to use them to build real time embodied systems based on more biologically realistic neurons, such as the multicompartment neurons used by Izhikevich and Edelman (2008).

One significant limitation of NeMo is that the simulation time steps have a relatively coarse resolution of 1 ms. While the reduction of the time step size is in principle simple, it would introduce a number of problems. First, the rotating queue structures used for spike delivery, which vary with the longest network delay, would increase in size to handle the increased number of time steps. Second, the throughput would decrease since the algorithm exploits 'clustering' of synapses with the same conduction delay, of which there will be fewer. A third problem is that a reduction in the time step would effectively narrow the time window in which synaptic plasticity has an effect. In practice the temporal resolution could be reduced by a small factor (to 0.5 ms or 0.25 ms, say), but finer temporal resolutions call for different algorithms, such as an event-based approach.[10]

A second shortcoming of NeMo is that its synaptic plasticity is currently limited to a global pair-based nearest-neighbour scheme. However, since the software contains a variety of connectivity data structures, it would be reasonably straightforward to add a more elaborate model. For example, the trace-based approach suggested by Morrison et al. (2008), would be a viable extension that would support the modelling of interactions (for example, triplets of spikes) and plasticity rules that depend on a neuron's state, such as its voltage. This approach to modelling synaptic plasticity would also remove the effect that a finer temporal resolution would have on the STDP window size.

A third potential shortcoming is that there is no built-in mechanism to deal with variations in processing time due to dynamic variations in network activity—for example, during periods of high network activity each time step is likely to take longer to process. In some circumstances more consistent real time performance could be achieved by pausing between simulation updates, so that each time step in NeMo took approximately the same amount of real time. In the future NeMo could be extended with mechanisms that would enable the user to select tradeoffs between performance and accuracy—for example, dropping randomly selected spikes when the performance was slower than real time.

A general problem with GPU-based spiking neural simulators is that the amount of memory on each card limits the size of networks that can be run and the number of cores constrains the simulation throughput of larger networks. While steady growth in card performance is anticipated, it is unlikely to become possible to simulate networks containing millions of neurons and billions of connections on a single card in the next few years. An obvious way of addressing this limitation is to distribute the network across a number of computers, with NeMo simulating part of the network on each machine and a communication interface, such as MPI, being used to exchange spikes at each time step. An efficient implementation of this approach depends on an optimal parcellation of the network that minimizes the exchange of spikes across machines. While it can be difficult to do this automatically, some network models lend themselves to a manual approach. For example, one or a number of cortical columns could be simulated on each machine, with the high level of intra-column connectivity being handled efficiently within each copy of NeMo, and only the inter-column connections generating spikes that are passed between machines. We are currently in the process of developing an MPI version of NeMo—an implementation with a simple network parcellation scheme is available on request.

While NeMo already has a functional API, more user-friendly APIs have been developed. For example, the Brian simulator (Goodman and Brette 2009) has some support for code generation (Goodman 2010), which can be used to combine flexible neuron and synapse models with high performance. Work is currently under way to integrate NeMo with Brian, so that NeMo can be used as an accelerator within this framework.

The performance issues with iSpike's visual spike conversion could be addressed by using NeMo to run the neural simulations within iSpike, which should greatly reduce the time taken to model the neurons that are encoding the visual maps. It would also be possible to use CUDA to speed up the

[10]These effects relate to changes to the temporal grid on which the spike events take place. A reduction of the step size used in the numerical integration within each simulation time step would only have a minimal effect on performance.

calculation of the visual maps—with these improvements it seems likely that the output from the iCub's cameras could be converted into a spiking retinal representation in close to real time. In the future iSpike could also be improved by increasing the biological realism of its sensory models and extending the senses that it supports. For example, audition could be added by modelling the cochlea with Lyon's method (Lyon 1982), and converting the output from this model into spikes using Izhikevich neurons. Spike conversion for robots with skin (Cannata et al. 2008; Hoshi and Shinoda 2006) could also be added, and it would be relatively straightforward to extend iSpike to communicate with other robots, such as ECCE (Marques et al. 2010), either using YARP or another network protocol.

Information Sharing Statement

NeMo is available as a stand-alone library under the terms of the GPLv2 license from http://nemosim.sf.net. A source package is available as well as binary packages for Windows and Mac OS X.

SpikeStream is available under the terms of the GPLv2 license from http://spikestream.sf.net. A source code package is available as well as pre-built binaries (with NeMo and iSpike embedded) for Windows and Mac OS X. A full manual with installation instructions is also available for SpikeStream.

iSpike is available as a stand-alone library under the terms of the GPLv2 license from http://ispike.sf.net. A source code package is available as well as pre-built binaries for Windows and Mac OS X.

Acknowledgements This work was supported by EPSRC grant EP/F033516/1. Some of the GPUs used were funded through the NVIDIA University Program. We would also like to thank the reviewers of this paper for their helpful comments.

References

Aleksander, I. (2005). *The world in my mind, my mind in the world*. Exeter: Imprint Academic.

Ananthanarayanan, R., Esser, S.K., Simon, H.D., Modha, D.S. (2009). The cat is out of the bag: cortical simulations with 10^9 neurons, 10^{13} synapses. In *Proc. conf. high performance computing networking, storage and analysis* (pp. 1–12). New York: ACM.

Andreou, A., & Boahen, K. (1994). A 48,000 pixel silicon retina in current-mode subthreshold cmos. In *37th midwest symposium on circuits and systems* (pp. 97–102).

Bergenheim, M., Ribot-Ciscar, E., Roll, J.P. (2000). Proprioceptive population coding of two-dimensional limb movements in humans: I. Muscle spindle feedback during spatially oriented movements. *Experimental Brain Research*, *134*(3), 301–310.

Bernardet, U., & Verschure, P.F. (2010). iqr: a tool for the construction of multi-level simulations of brain and behaviour. *Neuroinformatics*, *8*(2), 113–134.

Bernhard, F., & Keriven, R. (2006). Spiking neurons on GPUs. In *Proc. 6th int. conf. computational science* (pp. 236–243).

Bhowmik, D., & Shanahan, M. (2012). How well do oscillator models capture the behaviour of biological neurons? In *Proc. int. joint conf. neural networks*.

Binzegger, T., Douglas, R.J., Martin, K.A. (2004). A quantitative map of the circuit of cat primary visual cortex. *Journal of Neuroscience*, *24*(39), 8441–8453.

Bolduc, M., & Levine, M. (1998). A review of biologically motivated space-variant data reduction models for robotic vision. *Computer Vision and Image Understanding*, *69*, 170–184.

Bouganis, A., & Shanahan, M. (2010). Training a spiking neural network to control a 4-DoF robotic arm based on spike timing-dependent plasticity. In *Proc. int. joint conf. neural networks* (pp. 4104–4111).

Bower, J.M., Beeman, D., Hucka, M. (2003). The GENESIS simulation system In M. Arbib (Ed.), In *The handbook of brain theory and neural networks* (2nd ed., 475–478). Cambridge: MIT Press.

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J.M., Diesmann, M., Morrison, A., Goodman, P.H., Harris, F.C. Jr., Zirpe, M., Natschlager, T., Pecevski, D., Ermentrout, B., Djurfeldt, M., Lansner, A., Rochel, O., Vieville, T., Muller, E., Davison, A.P., El Boustani, S., Destexhe, A. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of Computational Neuroscience*, *23*(3), 349–398.

Buchmann, T. (2011). *Stock market trading with spiking neural networks*. MSc thesis, Imperial College London.

Cannata, G., Maggiali, M., Metta, G., Sandini, G. (2008). An embedded artificial skin for humanoid robots. In *IEEE int. conf. multisensor fusion and integration for intelligent systems* (pp. 434–438).

Carnevale, N.T., & Hines, M.L. (2006). *The NEURON book*. Cambridge: Cambridge University Press.

Cheung, K., Schultz, S.R., Leong, P.H.W. (2009). A parallel spiking neural network simulator. In *Proc. IEEE. int. conf. field-programmable technology* (pp. 247–254).

Clark, A. (2008). *Supersizing the mind: Embodiment, action, and cognitive extension*. New York: Oxford University Press.

Collins, D.F., & Prochazka, A. (1996). Movement illusions evoked by ensemble cutaneous input from the dorsum of the human hand. *Journal of Physiology*, *496*(Pt 3), 857–71.

Collins, D.F., Refshauge, K.M., Todd, G., Gandevia, S.C. (2005). Cutaneous receptors contribute to kinesthesia at the index finger, elbow, and knee. *Journal of Neurophysiology*, *94*(3), 1699–1706.

Davison, A.P., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., Yger, P. (2008). PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, *2*, article 11.

Djurfeldt, M., Lundqvist, M., Johansson, C., Rehn, M., Ekeberg, O., Lansner, A. (2008). Brain-scale simulation of the neocortex on the IBM Blue Gene/L supercomputer. *IBM Journal of Research and Development*, *52*(1–2), 31–41.

Edin, B.B., & Johansson, N. (1995). Skin strain patterns provide kinaesthetic information to the human central nervous system. *Journal of Physiology*, *487*(Pt 1), 243–251.

Enroth-Cugell, C., & Robson, J. (1966). The contrast sensitivity of retinal ganglion cells of the cat. *Journal of Physiology*, *187*(3), 517–552.

Ferrell, W.R., Gandevia, S.C., Mccloskey, D.I. (1987). The role of joint receptors in human kinaesthesia when intramuscular receptors cannot contribute. *Journal of Physiology*, *386*, 63–71.

Fidjeland, A.K. (2011). *NeMo manual*. http://nemosim.sf.net/manual.pdf.

Fitzpatrick, P., Metta, G., Natale, L. (2008). Towards long-lived robot genes. *Robotics and Autonomous Systems*, *56*(1), 29–45.

Fidjeland, A.K., Roesch, E.B., Shanahan, M.P., Luk, W. (2009). NeMo: a platform for neural modelling of spiking neurons using GPUs. In *Proc. IEEE int. conf application-specific systems, architectures and processors* (pp. 137–144).

Fidjeland, A., & Shanahan, M. (2010). Accelerated simulation of spiking neural networks using GPUs. In *Proc. int. joint conf. neural networks* (pp. 536–543). Piscataway: IEEE.

Fontaine, B., Goodman, D., Benichoux, V., Brette, R. (2011). Brian hears: online auditory processing using vectorization over channels. *Frontiers in Neuroinformatics*, *5*, 9.

Fountas, Z., Gamez, D., Fidjeland, A. (2011). A neuronal global workspace for human-like control of a computer game character. In *IEEE conf. computational intelligence and games* (pp. 350–357).

Gamez, D. (2007). Spikestream: a fast and flexible simulator of spiking neural networks. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*. Lecture Notes in Computer Science (Vol. 4668, pp. 360–369).

Gamez, D. (2010). Information integration based predictions about the conscious states of a spiking neural network. *Consciousness and Cognition*, *19*(1), 294–310.

Gamez, D. (2011a). *iSpike manual*. http://ispike.sf.net/doc/ispike-manual-2.1.pdf.

Gamez, D. (2011b). *SpikeStream manual*. http://spikestream.sf.net/pages/documentation.html.

Gamez, D., & Aleksander, I. (2011). Accuracy and performance of the state-based phi and liveliness measures of information integration. *Consciousness and Cognition*, *20*(4), 1403–1424.

Gamez, D., Fidjeland, A., Lazdins, E. (2012). iSpike: a spiking neural interface for the icub robot. *Bioinspiration and Biomimetics*, *7*, 025008.

Gamez, D., Newcombe, R., Holland, O., Knight, R. (2006). Two simulation tools for biologically inspired virtual robotics. In *Proc. IEEE 5th chapter conf. on advances in cybernetic systems* (pp. 85–90).

Georgopoulos, A.P., Schwartz, A.B., Kettner, R.E. (1986). Neuronal population coding of movement direction. *Science*, *233*(4771), 1416–1419.

Gewaltig, M.O., & Diesmann, M. (2007). NEST. *Scholarpedia*, *2*(4), 1430.

Goodman, D.F. (2010). Code generation: a strategy for neural network simulators. *Neuroinformatics*, *8*(3), 183–196.

Goodman, D.F., & Brette, R. (2009). The Brian simulator. *Frontiers in Neuroscience*, *3*(2), 192–197.

Grill-Spector, K., & Malach, R. (2004). The human visual cortex. *Annual Review of Neuroscience*, *27*, 649–677.

Grillner, S., Hellgren, J., Menard, A., Saitoh, K., Wikstrom, M.A. (2005). Mechanisms for selection of basic motor programs–roles for the striatum and pallidum. *Trends in Neuroscience*, *28*(7), 364–370.

Hagmann, P., Cammoun, L., Gigandet, X., Meuli, R., Honey, C.J., Wedeen, V.J., Sporns, O. (2008). Mapping the structural core of human cerebral cortex. *PLoS Biol*, *6*(7), e159.

Hammarlund, P., & Ekeberg, O. (1998). Large neural network simulations on multiple hardware platforms. *Journal of Computational Neuroscience*, *5*(4), 443–459.

Han, B., & Taha, T.M. (2010). Neuromorphic models on a GPGPU cluster. In *Proc int. joint conf. neural networks* (pp. 3050–3057). Piscataway: IEEE.

Hellwig, B. (2000). A quantitative analysis of the local connectivity between pyramidal neurons in layers 2/3 of the rat visual cortex. *Biological Cybernetics*, *82*(2), 111–121.

Hoshi, T., & Shinoda, H. (2006). Robot skin based on touch-area-sensitive tactile element. In *Proc. IEEE int. conf. robotics and automation* (pp. 3463–3468).

Ijspeert, A.J., Crespi, A., Ryczko, D., Cabelguen, J.M. (2007). From swimming to walking with a salamander robot driven by a spinal cord model. *Science*, *315*(5817), 1416–1420.

Indiveri, G., Linares-Barranco, B., Hamilton, T.J., van Schaik, A., Etienne-Cummings, R., Delbruck, T., Liu, S.C., Dudek, P., Hafliger, P., Renaud, S., Schemmel, J., Cauwenberghs, G., Arthur, J., Hynna, K., Folowosele, F., Saighi, S., Serrano-Gotarredona, T., Wijekoon, J., Wang, Y., Boahen, K. (2011). Neuromorphic silicon neuron circuits. *Frontiers Neuroscience*, *5*, 73.

Izhikevich, E.M. (2003). Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, *14*, 1569–1572.

Izhikevich, E., & Edelman, G. (2008). Large-scale model of mammalian thalamocortical systems. *Proceedings of the National Academy of Science of the United States of America*, *105*(9), 3593–3598. doi:10.1073/pnas.0712231105.

Jiirgens, R., Becker, W., Kornhuber, H. (1981). Natural and drug-induced variations of velocity and duration of human saccadic eye movements: evidence for a control of the neural pulse generator by local feedback. *Biological Cybernetics*, *39*, 87–96.

Jones, K.E., Wessberg, J., Vallbo, A.B. (2001). Directional tuning of human forearm muscle afferents during voluntary wrist movements. *Journal of Physiology*, *536*(2), 635–647.

Krichmar, J.L., Nitz, D.A., Gally, J.A., Edelman, G.M. (2005). Characterizing functional hippocampal pathways in a brain-based device as it solves a spatial memory task. *Proceedings of the National Academy of Science of the United States of America*, *102*(6), 2111–2116.

Kuramoto, Y. (1984). *Chemical oscillations, waves, and turbulence*. Berlin: Springer.

Linares-Barranco, A., Gomez-Rodriguez, F., Jimenez-Fernandez, A., Delbruck, T., Lichtensteiner, P. (2007). Using FPGA for visuo-motor control with a silicon retina and a humanoid robot. In *IEEE int. symp. circuits and systems* (pp. 1192–1195).

Liu, J.D., & Hu, H. (2006). Biologically inspired behaviour design for autonomous robotic fish. *Internation Journal of Automation and Computing*, *3*, 336–347.

Lyon, R. (1982). A computational model of filtering, detection, and compression in the cochlea. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, *7*, 1282–1285.

Macefield, G., Gandevia, S.C., Burke, D. (1990). Perceptual responses to microstimulation of single afferents innervating joints, muscles and skin of the human hand. *Journal of Physiology*, *429*, 113–129.

Maguire, L.P., McGinnity, T.M., Glackin, B., Ghani, A., Belatreche, A., Harkin, J. (2007). Challenges for large-scale implementations of spiking neural networks on FPGAs. *Neurocomputing*, *71*(1–3), 13–29.

Markram, H. (2006). The blue brain project. *Nature Reviews Neuroscience*, *7*(2), 153–160.

Marques, H., Jäntsch, M., Wittmeier, S., Alessandro, C., Holland, O., Diamond, A., Lungarella, M., Knight, R. (2010). ECCE1: the first of a series of anthropomimetic musculoskelal upper torsos. In *Proc. IEEE int. conf. humanoid robotics* (pp. 391–396).

Masquelier, T., & Thorpe, S. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Computational Biology*, *3*(2), e31.

Metta, G., Sandini, G., Vernon, D., Natale, L., Nori, F. (2008). The iCub humanoid robot: an open platform for research in embodied cognition. In *Proc. workshop on performance metrics for intelligent systems*.

Meuth, R.J., & Wunsch, D.C. (2007). A survey of neural computation on graphics processing hardware. In *2007 IEEE 22nd* *international symposium on intelligent control* (pp. 524–527). Piscataway: IEEE.

Morrison, A., Diesmann, M., Gerstner, W. (2008). Phenomenological models of synaptic plasticity based on spike timing. *Biological Cybernetics*, *98*(6), 459–478.

Nageswaran, J.M., Dutt, N., Krichmar, J.L., Nicolau, A., Veidenbaum, A.V. (2009). A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks*, *22*, 791–800.

Noë, A., & Thompson, E. (2004). Are there neural correlates of consciousness? *Journal of Consciousness Studies*, *11*(1), 3–28.

Nowotny, T. (2011). Flexible neuronal network simulation framework using code generation for NVidia CUDA. *BMC Neuroscience*, *12*(1), 239.

Rast, A., Galluppi, F., Davies, S., Plana, L., Patterson, C., Sharp, T., Lester, D., Furber, S. (2011). Concurrent heterogeneous neural model simulation on real-time neuromimetic hardware. *Neural Networks*, *24*(9), 961–978.

Ribot-Ciscar, E., Bergenheim, M., Albert, F., Roll, J.P. (2003). Proprioceptive population coding of limb position in humans. *Experimental Brain Research*, *149*(4), 512–519.

Richert, M., Nageswaran, J.M., Dutt, N., Krichmar, J.L. (2011). An efficient simulation environment for modeling large-scale cortical processing. *Frontiers in Neuroinformatics*, *5*, 19.

Robinson, D.A. (1964). The mechanics of human saccadic eye movement. *Journal of Physiology*, *174*, 245–264.

Roll, J.P., Albert, F., Ribot-Ciscar, E., Bergenheim, M. (2004). “Proprioceptive signature” of cursive writing in humans: a multi-population coding. *Experimental Brain Research*, *157*(3), 359–368.

Roll, J.P., Bergenheim, M., Ribot-Ciscar, E. (2000). Proprioceptive population coding of two-dimensional limb movements in humans: II. Muscle-spindle feedback during “drawing-like” movements. *Experimental Brain Research*, *134*(3), 311–321.

Schemmel, J., Brüderle, D., Grübl, A., Hock, M., Meier, K., Millner, S. (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proc. IEEE int. conf. circuits and systems* (pp. 1947–1950).

Schwartz, E.L. (1980). Computational anatomy and functional architecture of striate cortex—a spatial-mapping approach to perceptual coding. *Vision Research*, *20*(8), 645–669.

Song, S., Miller, K.D., Abbott, L.F. (2000). Competitive hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, *3*(9), 919–926.

Sporns, O. (2007). Brain connectivity. *Scholarpedia*, *2*(10), 4695.

Thomas, D.B., & Luk, W. (2009). FPGA accelerated simulation of biologically plausible spiking neural networks. In *Proc. IEEE symp. field-programmable custom computing machines*.

Tiesel, J.P., & Maida, A.S. (2009). Using parallel GPU architecture for simulation of planar I/F networks. In *Proc int. joint conf. neural networks* (pp. 754–759).

Tononi, G. (2008). Consciousness as integrated information: a provisional manifesto. *Biological Bulletin*, *215*(3), 216–242.

Vogels, T.P., & Abbott, L.F. (2005). Signal propagation and logic gating in networks of integrate-and-fire neurons. *Journal of Neuroscience*, *25*(46), 10786–10795.

Yudanov, D., Shaaban, M., Melton, R., Reznik, L. (2010). GPU-based simulation of spiking neural networks with real-time performance and high accuracy. In *Proc. int. joint conf. neural networks*.