

# INFORMATION REPRESENTATION ON A UNIVERSAL NEURAL CHIP

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER  
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY  
IN THE FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

2013

By  
Francesco Galluppi  
School of Computer Science

# Contents

<b>Abstract</b>	<b>14</b>
<b>Declaration</b>	<b>15</b>
<b>Copyright</b>	<b>16</b>
<b>Acknowledgements</b>	<b>17</b>
<b>1 Introduction</b>	<b>18</b>
1.1 Background . . . . .	20
1.2 Motivation and Research aims . . . . .	24
1.3 Contributions and Publications . . . . .	25
1.3.1 Journal Papers . . . . .	27
1.3.2 Conference Papers . . . . .	28
1.3.3 Workshops . . . . .	31
1.4 Thesis outline . . . . .	31
<b>2 Computation with Spiking Neurons</b>	<b>33</b>
2.1 Introduction . . . . .	33
2.2 Bioinspired models of neurons and synapses . . . . .	35
2.2.1 Neural Models . . . . .	36
2.2.2 Synaptic Models . . . . .	45
2.3 Spiking Models . . . . .	49
2.3.1 Biologically inspired models . . . . .	51
2.3.2 Rank Order Coding . . . . .	52
2.3.3 Neural Engineering Framework . . . . .	54
2.3.4 Polychronization . . . . .	59
2.3.5 Rank order codes and polychronization combined . . . . .	61

2.4	Summary . . . . .	63
<b>3</b>	<b>Tools for Neural Simulation</b>	<b>64</b>
3.1	Introduction . . . . .	64
3.2	Software Simulators . . . . .	65
3.2.1	NEURON . . . . .	66
3.2.2	GENESIS . . . . .	66
3.2.3	NEST . . . . .	66
3.2.4	Brian . . . . .	67
3.2.5	Nengo . . . . .	68
3.2.6	Towards a Standard . . . . .	70
3.2.7	PyNN . . . . .	72
3.3	Simulation on dedicated platforms . . . . .	74
3.3.1	Supercomputers . . . . .	75
3.3.2	Neuromorphic Hardware . . . . .	77
3.3.3	Field-Programmable Gate Arrays . . . . .	78
3.3.4	Graphics Processing Units . . . . .	80
3.3.5	Address Event Representation . . . . .	80
3.4	SpiNNaker . . . . .	82
3.4.1	Architecture . . . . .	83
3.4.2	Hardware Reconfigurability . . . . .	86
3.4.3	Neural Applications . . . . .	87
3.5	Summary . . . . .	89
<b>4</b>	<b>Configuring a universal neural system</b>	<b>91</b>
4.1	Introduction . . . . .	91
4.2	Model to Hardware . . . . .	92
4.2.1	The mapping approach . . . . .	93
4.2.2	Partitioning and Configuration Manager . . . . .	94
4.2.3	Algorithms . . . . .	98
4.2.4	Data Representation . . . . .	98
4.2.5	Design and implementation choices . . . . .	99
4.2.6	Introducing new types . . . . .	101
4.2.7	Compatible front-ends . . . . .	102
4.3	Results . . . . .	103
4.4	PACMAN for larger systems . . . . .	109

4.5	Revisiting PACMAN for 48-chip boards . . . . .	111
4.6	An open system . . . . .	113
4.7	Summary . . . . .	115
<b>5</b>	<b>Mapping Different Front-ends</b>	<b>117</b>
5.1	Introduction . . . . .	117
5.2	PyNN and PACMAN . . . . .	118
5.2.1	The PyNN.SpiNNaker module . . . . .	118
5.2.2	LIF Neuron . . . . .	120
5.2.3	Izhikevich Neuron . . . . .	121
5.2.4	Multimodel simulations . . . . .	124
5.2.5	Oscillatory Network Dynamics . . . . .	128
5.2.6	Profiling the platform . . . . .	130
5.2.7	Opening SpiNNaker to PyNN models . . . . .	141
5.3	Neural Engineering Framework on SpiNNaker . . . . .	142
5.3.1	Encoding/Decoding approach . . . . .	143
5.3.2	Results . . . . .	145
5.3.3	The Neural Engineering Framework on SpiNNaker . . . . .	152
5.4	Summary . . . . .	153
<b>6</b>	<b>Mapping AER sensors and robotic actuators</b>	<b>155</b>
6.1	Introduction . . . . .	155
6.2	Visual attentional model using a silicon retina . . . . .	156
6.2.1	Hardware setup . . . . .	157
6.2.2	Network Description . . . . .	158
6.2.3	Results . . . . .	163
6.2.4	An event-driven configurable platform . . . . .	164
6.3	Integrating SpiNNaker, NEF and the robot . . . . .	165
6.3.1	Simple model of path integration . . . . .	168
6.3.2	Spiking ratSLAM . . . . .	173
6.4	Summary . . . . .	180
<b>7</b>	<b>Conclusions</b>	<b>185</b>
<b>A</b>	<b>Rank order codes and polychronization combined</b>	<b>190</b>

<b>B</b>	<b>Results from simulations with the LIF neuron</b>	<b>199</b>
B.1	Simulation results with a LIF neuron . . . . .	199
B.1.1	Single Neuron Dynamics . . . . .	199
B.1.2	Spikes Propagation . . . . .	199
B.1.3	Oscillatory Network Activity . . . . .	201
<b>C</b>	<b>Supplementary Material</b>	<b>203</b>
C.1	PyNN Files . . . . .	203
C.1.1	Chapter 3 . . . . .	203
C.1.2	Chapter 4 . . . . .	203
C.1.3	Chapter 5 . . . . .	203
C.1.4	Chapter 6 . . . . .	204
C.2	Nengo Files . . . . .	204
C.2.1	Chapter 3 . . . . .	204
C.2.2	Chapter 5 . . . . .	204
C.2.3	Chapter 6 . . . . .	204
C.3	Videos . . . . .	204
C.3.1	AER Sensors . . . . .	204
C.3.2	Robot . . . . .	204
C.4	SpiNNaker Package Documentation . . . . .	205
	<b>Bibliography</b>	<b>205</b>

Word Count: 43114

# List of Tables

4.1	PACMAN Example. . . . .	97
4.2	Revisiting PACMAN for 48-chip boards. . . . .	112
5.1	Translation methods for the Izhikevich neuron. . . . .	120
5.2	Profiling results for 4-chip boards . . . . .	138
5.3	Profiling results for 48-chip boards . . . . .	141

# List of Figures

1.1	Three drawings by Santiago Ramon y Cajal, taken from the book <i>Comparative study of the sensory areas of the human cortex</i> . . . . .	19
1.2	Neural operations from <i>A logical calculus of the ideas immanent in nervous activity</i> [McCulloch and Pitts, 1943]. . . . .	21
2.1	Neuron structure, from Thompson [1967]. . . . .	35
2.2	Signal transmission in a chemical synapses, from Kandel et al. [1991].	37
2.3	First published intracellular recording of an action potential, from Hodgkin and Huxley [1939]. . . . .	38
2.4	Circuit equivalent model for the HH neuron, from Hodgkin and Huxley [1952]. . . . .	39
2.5	Activation and deactivation of function $n$ , $m$ , $h$ during the generation of an axon potential. . . . .	40
2.6	Phase-plane description of the planer system modelling the resting and spiking attractors, from Izhikevich [2006a]. . . . .	43
2.7	Graphical representation of the parameters in the Izhikevich model from Izhikevich [2006a]. . . . .	44
2.8	Synaptic models, from Dayan and Abbott [2001b]. . . . .	46
2.9	Synaptic models: first order kinetic decay with an instantaneous rise (b) alpha functions (c) difference of two exponentials, after De Schutter [2009]. . . . .	47
2.10	Variation of conductance with post-synaptic membrane potential in NMDA synapses, from Dayan and Abbott [2001b]. . . . .	49
2.11	Speed of processing of the visual system in a recognition task, from Thorpe et al. [1996]. . . . .	50
2.12	Rank order codes. . . . .	53
2.13	Tuning curve examples. . . . .	55
2.14	SPAUN architecture, from Eliasmith et al. [2012]. . . . .	59

2.15	Polychronization. . . . .	60
2.16	The output neuron fires only when the learned code is presented. . .	62
3.1	GENESIS structure, from <a href="#">Bower et al. [1998]</a> . . . . .	67
3.2	Example code and execution results from Brian, from <a href="#">Goodman [2008]</a> . . .	68
3.3	Example of a basal ganglia network model running in Nengo. . . . .	69
3.4	NeuroML structure, from <a href="http://www.neuroml.org/introduction.php">www.neuroml.org/introduction.php</a> . . . . .	71
3.5	PyNN Architecture, from <a href="#">Davison et al. [2008]</a> . . . . .	72
3.6	An example of using PyNN, adapted from <a href="http://neuralensemble.org/trac/PyNN/">http://neuralensemble.org/trac/PyNN/</a> . . . . .	74
3.7	Estimation for a full-brain, real-time simulation from <a href="#">Ananthanarayanan et al. [2009]</a> . . . . .	76
3.8	Design drawing of the BrainScaleS Neural Network Hardware Module from <a href="http://brainscales.kip.uni-heidelberg.de/">http://brainscales.kip.uni-heidelberg.de/</a> . . . . .	78
3.9	A 16-FPGA Bluehive system, from <a href="#">Moore et al. [2012]</a> . . . . .	79
3.10	A working CAVIAR prototype assembled in April 2005 at the CAVIAR workshop and its schematics, from the project resources website <a href="http://www.ini.uzh.ch/~tobi/caviar/">http://www.ini.uzh.ch/~tobi/caviar/</a> . . . . .	81
3.11	SpiNNaker chip diagram, after <a href="#">[Plana et al., 2007]</a> . Each SpiNNaker chip contains 18 ARM968 cores embedded in a programmable, packet based, network on chip. Action potentials are encoded as source-based AER packets, and transmitted through a Multicast Router capable of handling one packet per clock cycle if unimpeded. Every core has a local Tightly-Coupled Memory and access through the System NoC to a dedicated DMA controller, used to access a 1 Gbit SDRAM within every chip. . . . .	85
3.12	Inter-chip connectivity for a 4x4 system without wrap-around. Each SpiNNaker chip (numbered in red) is connected to its neighbours through 6 bidirectional asynchronous links (numbered in black). . . .	85
3.13	Neural Simulation Events. Neural equations are solved during a <i>Neural Event</i> and a <i>Spike Event</i> is produced if a particular condition is met. Spikes are encoded as source-based AER packets and are routed to their destination cores by the MC routers. On the post-synaptic chip synapses are retrieved locally using a source-based lookup to access SDRAM; synaptic inputs are then injected into the target neurons ( <i>Synaptic Event</i> ). . . . .	87

3.14	Single neuron dynamics. The neuron is injected with 4 pulses of current. . . . .	88
3.15	Functional blocks on a section of a SpiNNaker Chip. . . . .	89
4.1	Example neural and connection input lists. Explicit modifications are required for any change in the network model. . . . .	93
4.2	PACMAN structure. . . . .	95
4.3	Hierarchical Approach. . . . .	95
4.4	Example network composed of 2 Populations and a Spike Detector interconnected . . . . .	96
4.5	PACMAN entity-relationship diagram. . . . .	100
4.6	Example Benchmark Network Dynamics. . . . .	106
4.7	Results from a synfire chain of 60 nodes of 100+25 neurons each. . . .	107
4.8	2x2 PACMAN performance analysis. . . . .	108
4.9	Time building results for models run on a 2x2 system. . . . .	109
4.10	Results from mapping and routing models on a 16x16 system. . . . .	110
4.11	PACMAN execution flow example. . . . .	113
5.1	(a) Results of a single LIF neuron compared with PyNN.nest and PyNN.brian (b) Comparison between firing rates . . . . .	120
5.2	Conductance-based LIF neuron simulation on SpiNNaker, compared to Brian (and to Figure 3.6) . . . . .	121
5.3	Results of a single neuron compared with Brian . . . . .	122
5.4	Membrane and network dynamics for simple networks. . . . .	123
5.5	Multimodel network of neural assemblies: large-scale structure and rate plots. . . . .	125
5.6	Activation of a selected triplet of neurons in the mixed-model neural assembly network, with and without top-down priming. . . . .	129
5.7	Simulation with 500 neurons: comparison between NEST and SpiNNaker. (a) Raster Plot for the entire simulation (b) Inter-Spike-Interval (ISI) Distribution (c) Mean activity rate . . . . .	131
5.8	How many neurons can be modelled on a single core of a SpiNNaker chip? Theoretical model. . . . .	132
5.9	Synfire chain test on a test chip board. . . . .	134
5.10	Raster Plot from the simulation of the synfire chain on 4 chips. . . . .	135

5.11	Experiment 1: 16 populations of a 100 complex neurons each firing at 100 Hz are fully interconnected to 5 other random populations. . .	137
5.12	Experiment 2: 16 populations of a 1000 simple neurons each firing at 1 Hz are fully interconnected to 1 other random population. . . . .	137
5.13	Number of neurons that can be modelled by each core, measured limit.	139
5.14	48-node SpiNNaker board ( $10^3$ Machine, 864 cores). . . . .	140
5.15	Example of a single chip containing independent populations, as the model used to test and profile the 48-node boards. . . . .	141
5.16	Approach: Encoding and decoding. . . . .	144
5.17	Structure of the communication channel . . . . .	145
5.18	Representation Principle: Communication channel. . . . .	146
5.19	Transformation Principle: Computing the Square. . . . .	148
5.20	Dynamics: Neural Integrator . . . . .	150
5.21	Cyclic Attractor: Oscillator . . . . .	151
5.22	Non-linear system: Frequency Controlled Oscillator . . . . .	152
6.1	Overview of the hardware setup: 4 SpiNNaker chips board (left), an FPGA translating the AER protocol between the two asynchronous systems (middle) and a DVS Silicon Retina (right). . . . .	158
6.2	. . . . .	159
6.3	Experiment I: tuning curves. . . . .	160
6.4	Experiment II - visual attention task. . . . .	164
6.5	Activity in LIP follows the movements of the biased stimulus. . . . .	165
6.6	The Neural Network used. 512 input neurons from the retina feed into a hidden sub-sampler layer, and then to a competitive output layer to give the robot direction. . . . .	166
6.7	4-chip board/OmniBot configuration for the system used in Telluride 2012. . . . .	167
6.8	Nengo network for the return home robot. . . . .	169
6.9	Basal ganglia model from <a href="#">Stewart et al. [2010b]</a> , based on <a href="#">Gurney et al. [2001]</a> . . . . .	171
6.10	Nengo network for the return home robot with basal ganglia. . . . .	172
6.11	The theta oscillator model for place cells from <a href="#">Blair et al. [2008]</a> . . . . .	174
6.12	Nengo network model for path integration, mono-dimensional case. . . . .	177
6.13	The robot/SpiNNaker platform navigating to the <i>cheese</i> place field . . . . .	178
6.14	Simplified Nengo network for the bi-dimensional ratSLAM model. . . . .	179

6.15	The robot/SpiNNaker platform detecting 3 places . . . . .	180
6.16	The RatSLAM 2D model autonomously navigating through place cells 1 and 2. . . . .	181
6.17	The RatSLAM 2D model autonomously navigating through place cells 3 and 4. . . . .	182
6.18	The SpiNNaker/robot platform. . . . .	183
7.1	Levels of investigation in computational neuroscience, from <a href="#">Churchland and Sejnowski [1992]</a> . . . . .	186
A.1	Model architecture. . . . .	191
A.2	STDP training phase in the polychronous layer . . . . .	192
A.3	The output neuron is trained to respond only when the combination of firings in the polychronous layer appears. This is done by increase the weight until a spike is produced (in this case at sec 10 of the simulation). . . . .	194
A.4	Membrane potential of the output neuron during the presentation of two codes. . . . .	194
A.5	Contribution in terms of population mean fire rate of noise to the normal activity of the input population . . . . .	195
A.6	The output neuron spikes only when the learned code is presented .	196
A.7	The output neuron fires only when the learned code is presented. . .	196
B.1	Single neuron dynamics. The neuron is injected with 4 pulses of current. . . . .	200
B.2	Detection of interpulse interval between spikes generated from neu- rons $a$ and $b$ . Synaptic delay between transponders and detectors is marked as $d$ . . . . .	200
B.3	Network Structure. Arrows represent connections between neurons. Values at the end of each arrow represent synaptic delays, which are set accordingly to detectors placement (cfr. Figure B.2) . . . . .	201
B.4	Spikes Propagation. . . . .	202
B.5	Oscillatory Network Structure. . . . .	202
B.6	Oscillatory Network Raster Plot. . . . .	202

# List of Abbreviations

AER	Address Event Representation protocol, page 78
AMPA	$\alpha$ -amino-3-hydroxy-5-methyl-4-isoxazolepropionic receptor, page 45
ASIC	Application Specific Integrated Circuits, page 75
CAVIAR	Context Aware Vision using Image-based Active Recognition, page 78
DTI	Diffusion Tensor Imaging, page 51
EPSC	Excitatory Post Synaptic Current, page 47
EPSP	Excitatory Post Synaptic Potential, page 47
ERP	Event Related Potentials, page 34
fMRI	Functional Magnetic Resonance Imaging, page 51
FPGA	Field-Programmable Gate Array, page 76
GABA	Gamma-AminoButyric Acid receptors, page 45
GENESIS	GEneral NEural SIMulation System, page 66
GPU	Graphics Processing Units, page 77
HICANN	High Input Count Analog Neural Network, page 76
MC	MultiCast, page 81
NEST	NEural Simulation Tool, page 66
NMDA	N-methyl D-aspartate, page 45
PACMAN	PARtitioning and Configuration MANager, page 92

RISC	Reduced Instruction Set Computer, page 81
SDRAM	Synchronous Dynamic Random-Access Memory, page 80
SPAUN	Semantic Pointer Architecture Unified Network, page 58
STDP	Spike-Timing-Dependent Plasticity, page 51
VLSI	Very-Large-Scale Integration, page 75

# Abstract

How can science possibly understand the organ through which the Universe knows itself? The scientific method can be used to study how electro-chemical signals represent information in the brain. However, modelling it by simulating its structures and functions is a computation- and communication-intensive task. Whilst supercomputers offer great computational power, brain-scale models are challenging in terms of communication overheads and power consumption. Dedicated neural hardware can be used to enhance simulation performance, but it is often optimised for specific models. While performance and flexibility are desirable simulation features, there is no perfect modelling platform, and the choice is subordinate to the specific research question being investigated. In this context SpiNNaker constitutes a novel parallel architecture, with communication and memory accesses optimised for spike-based computation, permitting simulation of large spiking neural networks in real time.

To exploit SpiNNaker’s performance and reconfigurability fully, a neural network model must be translated from its conceptual form into data structures for a parallel system. This thesis presents a flexible approach to distributing and mapping neural models onto SpiNNaker, within the constraints introduced by its specialised architecture. The conceptual map underlying this approach characterizes the interaction between the model and the system: during the build phase the model is placed on SpiNNaker; at runtime, placement information mediates communication with devices and instrumentation for data analysis. Integration within the computational neuroscience community is achieved by interfaces to two domain-specific languages: PyNN and Nengo. The real-time, event-driven nature of the SpiNNaker platform is explored using address-event representation sensors and robots, performing visual processing using a silicon retina, and navigation on a robotic platform based on a cortical, basal ganglia and hippocampal place cells model. The approach has been successfully exploited to run models on all iterations of SpiNNaker chips and development boards to date, and demonstrated live in workshops and conferences.

# Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

# Copyright

- i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.
- ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.
- iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.
- iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see <http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=487>), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see [www.manchester.ac.uk/library/aboutus/regulations](http://www.manchester.ac.uk/library/aboutus/regulations)) and in The University’s policy on presentation of Theses

# Acknowledgements

The three-year journey as a PhD student in the SpiNNaker project has been an incredible experience, which gave me the chance to meet so many great people. My first thanks go therefore to, my supervisor, Steve Furber, and to all my fellow members in the SpiNNaker group: every one of you has taught me so much, personally and professionally, and had a huge impact on my personal development.

Workshops played a key role in my PhD. Many thanks to Jorg Cönradt and his group in Munich for all the work shared in Capocaccia and Telluride, on the robotic platform and on the interconnection with SpiNNaker. Special thanks to Chris Elia-smith and Terry Stewart for all the help in Telluride in implementing the Neural Engineering Framework on SpiNNaker: it was great fun. A big thank you to Bernabé Linares Barranco and his group in Seville for the work on the retina connection, and to Kevin Brohan for all the work done in Capocaccia 2012 in designing the network, running the experiments and analysing results for the retina model. Many thanks to Giacomo Indiveri, Kwabena Boahen, Ralph Etienne-Cummings, Shih-Chii Liu and Toby Delbruck for involving me in the Capocaccia and Telluride workshops, and for all the efforts in organizing them and keeping the neuromorphic community vibrant.

Thanks to EPSRC and BrainScaleS for funding my PhD and travels. Workshops and conferences gave me the chance to meet many great researchers, and most of them turned out to be very nice people as well; thanks for all the moments shared in Capocaccia, Barcelona, Sydney, Brisbane, Munich, Telluride, Leysin, San Diego, San Jose, Stanford, Edinburgh, Paris, Doha, Hsinchu, Singapore and Southampton.

Finally my warmest thanks go to my parents, who have always supported my decisions and encouraged me constantly, and to my friends, the ones I made in Manchester and the ones far away, who I always felt very close.

*What is now proved was once only imagin'd.*

William Blake

# Chapter 1

## Introduction

The application of the scientific method to quantitative study of how physical phenomena relate to human responses can be tracked down to the early days of psychophysics, with Ernst Weber and Gustav Fechner systematically studying the discrimination power of the visual system based on controlled intensity changes. Since then the brain has been identified as the location where perception, cognition and behaviours take place. As a consequence, models of mental activity based on empirical observation of nervous cells and of brain structures have been produced. Anatomical and functional data from neuro-biologists have revolutionised the understanding of how the nervous system processes information from its sensory organs, and how it adapts its behaviour within the environment.

Theories of the mind are progressively taking into account these observations [Denet, 1993, Edelman, 1993]: underneath the self-perception of a unitary conscious state, many parallel, hierarchical systems work in concert or compete [Edelman, 1987] to determine responses of an individual to his environment. This parallelization is expressed both with distinct, functionally specialised areas [Tononi et al., 1998], such as the visual cortex for visual processing or Broca's area for speech production, and in the elements constituting such brain areas: neurons and their connections. Since the beginning of the last century scientists have been observing neurons and the way they are organised, as reported by early drawings of Santiago Ramon y Cajal (Figure 1.1), inspired by the observation of neural tissue using Golgi's method, *la reazione nera*. The observations which led Hodgkin and Huxley to be awarded a Nobel Prize for Medicine and Physiology in 1963 formed the basis of the formalization of mathematical models of the electro-chemical dynamical behaviour of nervous cells, identifying the action potential (or *spike*), and its effects on

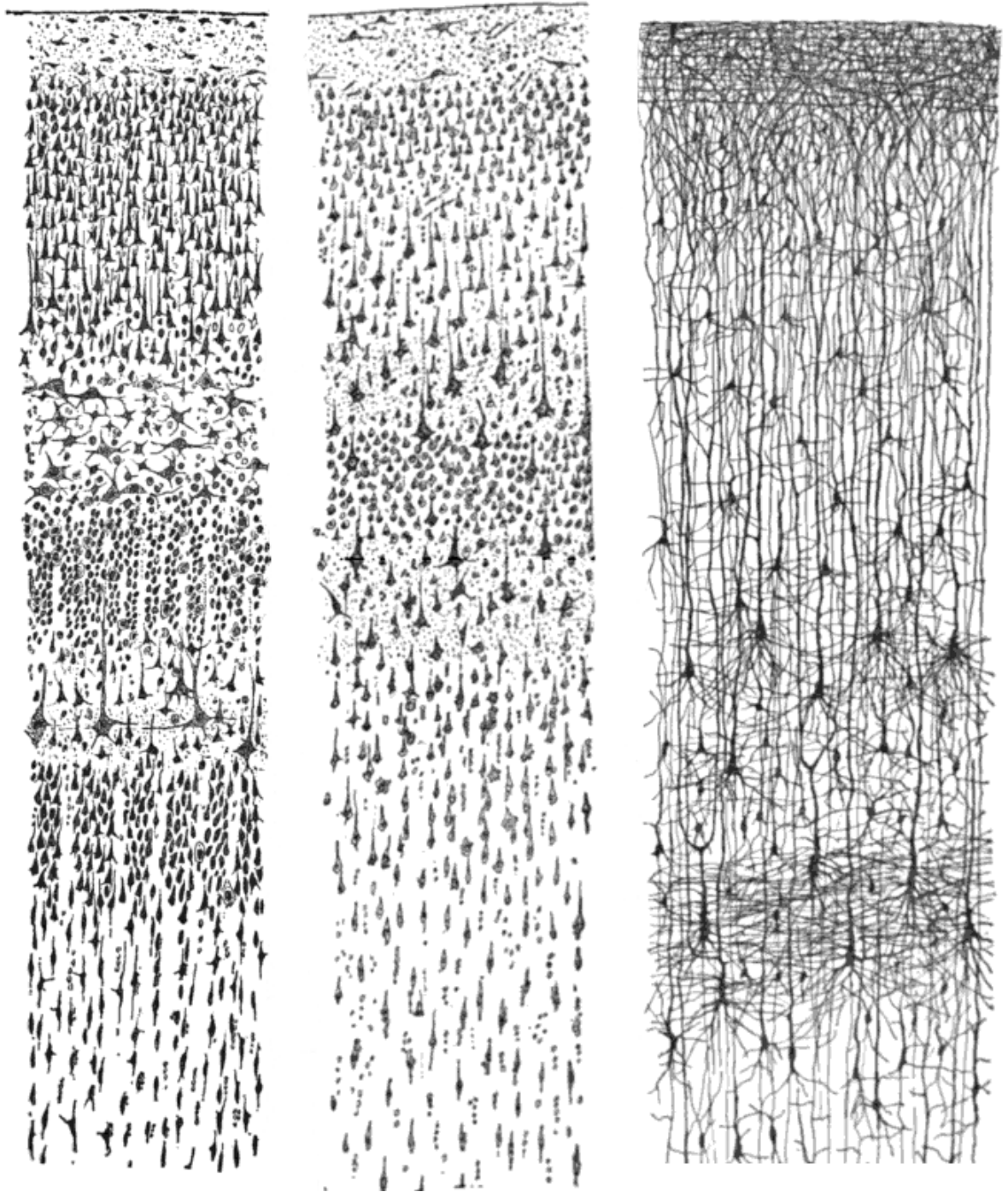


Figure 1.1: Three drawings by Santiago Ramon y Cajal, taken from the book *Comparative study of the sensory areas of the human cortex*.

the synaptic connection between neurons, as the means of communication between neural cells.

The study of a system such as the brain, formed by many elementary components working in parallel and capable of rapidly elaborating information in a fault tolerant way, has attracted the attention of mathematicians, physicists, biologists, psychologists, neurocognitive and computer scientists. This has developed in the *computational neuroscience* discipline, whose ultimate aim is, in the words of [Sejnowski, Koch, and Churchland](#), *"to explain how electrical and chemical signals are used in the brain to represent and process information. This goal is not new, but much has changed in the last decade. More is known now about the brain because of advances in neuroscience, more computing power is available for performing realistic simulations of neural systems, and new insights are available from the study of simplifying models of large networks of neurons. Brain models are being used to connect the microscopic level accessible by molecular and cellular techniques with the systems level accessible by the study of behavior"* [[Sejnowski et al., 1988](#)].

Large-scale models of biologically inspired neural networks [[Markram, 2006](#)] are an essential tool to test hypotheses regarding the mechanisms of information processing in the brain, or to exploit the computational capabilities of networks of neurons [[Furber and Brown, 2009](#), [Maass and Markram, 2004](#)]. Accurate models relating behaviour to brain and neural activity [[Moran et al., 2011](#)] would lead to more accurate understanding of brain-related diseases, and of the functions that robotics and machine learning are trying to mimic with cognitive computing [[Ananthanarayanan et al., 2009](#)].

## 1.1 Background

The study of how information is represented and elaborated in the nervous system opens possibilities for engineers and computer scientists to explore alternative computational paradigms inspired by neuroscience [[Aleksander, 1990](#)]. The history of artificial neural networks brings together researchers from different disciplines: the first artificial neuron, the Threshold Logic Unit proposed in 1943 by [Mcculloch and Pitts](#) [[1943](#)], is a simple unit performing the sum of its inputs and becomes active when that sum exceeds a threshold; a neuron can connect to other neurons to propagate its activity, introducing a neural base algebra (see Figure 1.2). Following Hebb's principle (often simplified as *neurons that fire together, wire together*) published in his

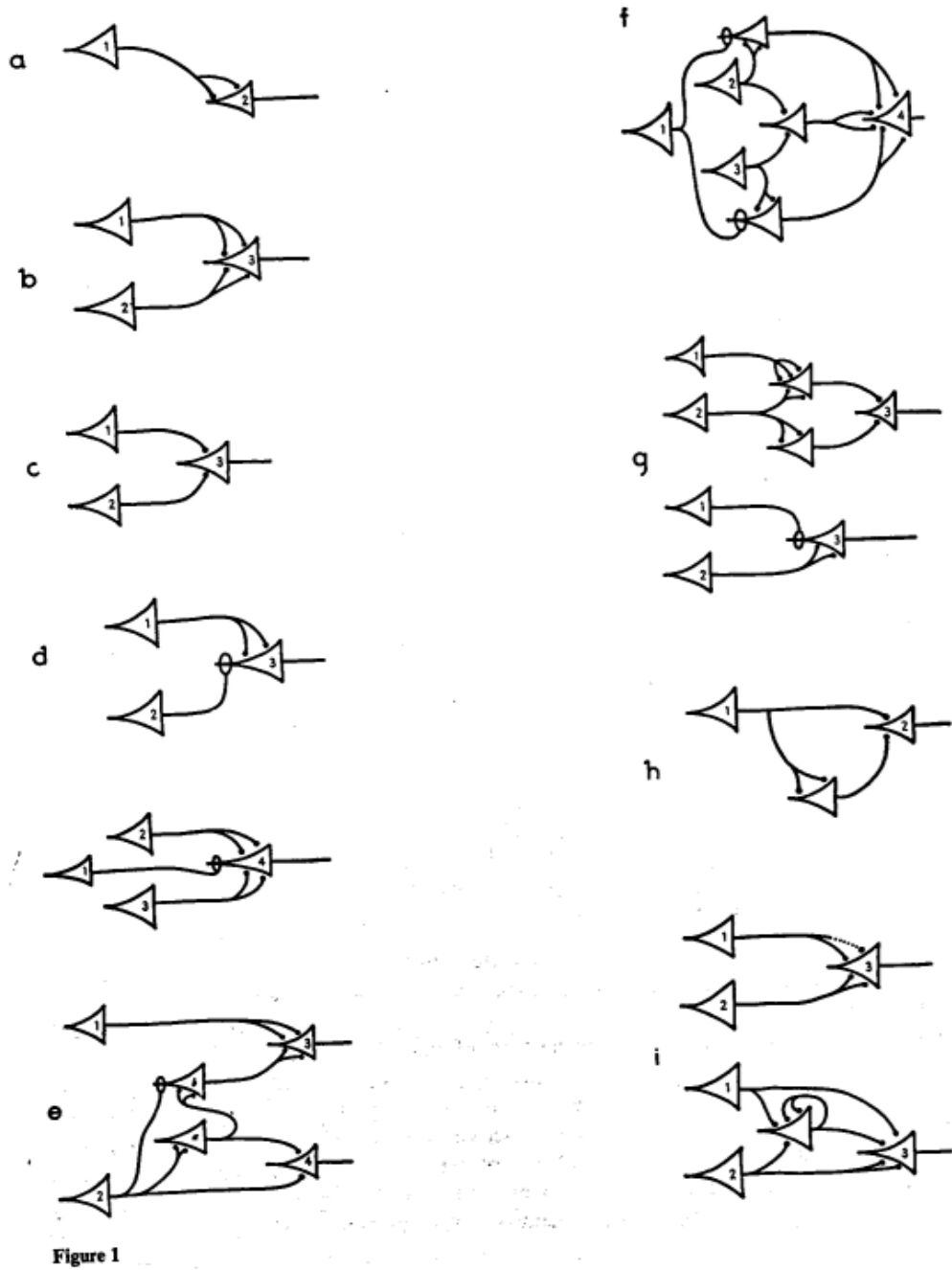


Figure 1.2: Neural operations from *A logical calculus of the ideas immanent in nervous activity* [McCulloch and Pitts, 1943].

*The Organization of Behaviour* book, networks of neurons become plastic, and therefore capable of learning, for example using *back propagation* to classify inputs in a supervised way. In 1958 Rosenblatt introduced the *Perceptron*, a neural network that learns from its inputs to linearly separate them. Models of neurons became increasingly sophisticated, with the binary activation output of neurons being substituted by a continuous value, and therefore enhancing the representation capabilities of neural networks. In 1982 Hopfield's studies of multi-layered, recurrent networks enabled models of network of neurons that can learn to discriminate inputs in an unsupervised way.

While establishing themselves as pillars in the machine learning field, this type of neural network progressively abandoned fidelity with their biological counterparts. Spiking neurons were introduced as more accurate, biologically inspired models of neural cells, which can therefore be used to explain the activity, normal or pathological, of the human brain. The interest in modelling the brain is shared inside and outside the scientific community, as demonstrated by the attention of funding bodies, such as the EU with the *Human Brain Project*<sup>1</sup>, DARPA and IBM with the *SyNAPSE Project*<sup>2</sup> and the US with the *Brain Activity Map* project [Alivisatos et al., 2012].

In spiking models every neuron is characterised by its membrane potential dynamics, as directly observed *in vivo* and *in vitro*, and modelled mathematically by several authors, as reviewed in Izhikevich [2004]. Action potentials are assumed to be stereotypical events, carrying the information in the time of their occurrence to other neurons. Time relations are essential features of spiking neural networks, as information is encoded in spike timing. Methods to encode and decode spiking information have consequently been presented [Izhikevich, 2006b, Thorpe and Gautrais, 1998, Eliasmith and Anderson, 2003], and timing activity has been studied in detailed models of the thalamocortical system Izhikevich and Edelman [2008]. Such timing properties of spiking neurons have also been used to account for the fast response of the visual system, capable of discriminating and appraising visual scenes [Thorpe et al., 1996], or associate visual stimulation to high level concepts to neural activity very rapidly [Quiari Quiroga et al., 2005].

The number of neurons, spikes transmitted in the network and of synapse activations in large networks makes simulation on standard parallel computers challenging, due to synchronization and communication overheads [Plesser et al., 2007].

---

<sup>1</sup><http://www.humanbrainproject.eu>

<sup>2</sup><http://www.ibm.com/synapse>

Study of efficient modelling technologies is therefore a key research area for computational neuroscientists [Morrison et al., 2005]; dedicated hardware emerges then as a natural candidate simulation substrate which exploits parallelism to circumvent software limitations. With the seminal work of Mead [1989] the *neuromorphic* approach was introduced: electronic analog circuits, constituting Very-Large-Scale-Integration systems (VLSI), can be used to replicate membrane potential dynamics of neurons at the transistor level, and can be combined in parallel, fast, and power efficient devices for biologically-inspired computation. The real-time nature of these devices is implicit in the assumption that *time models itself*, as information is represented in the timing of occurrence of an event such as the transmission of an action potential.

Using the address-event-representation (AER) protocol [Mahowald, 1992] devices become event-driven, with parallel elements communicating their address to connected units. This principle has been used to implement VLSI visual systems, known as silicon retinas to denote their biological inspiration. Silicon retina cells respond to local changes in light intensity by emitting an event, signalling the location of the cell in the visual array, in a completely parallel and asynchronous way; within the AER approach there is no concept of a “frame”, and events occur and are processed in real time, taking advantage of the reduced redundancy in the visual information. The use of such fast VLSI systems for vision and computation has been presented in an integrated neuromorphic/AER system, the EU funded project CAVIAR [Serrano-Gotarredona et al., 2009]. CAVIAR is a visual processing system comprising several AER chips and interfaces, modelled with 45k neurons and 5M synapses, capable of fast recognition and tracking. However, only few large-scale parallel systems for real-world real-time sensory processing, learning, and generating complex motor outputs have been implemented.

The cost and effort of producing custom solutions, the availability of just a few specimens, and their steep learning curve for non-hardware experts make them a scarce resource in the computational neuroscience community. More flexible, readily available solutions are offered by off-the-shelf parallel hardware such as Field-programmable gate arrays (FPGAs) and Graphics processing units (GPUs), and can be used to run large scale models in real or accelerated time [Fidjeland et al., 2009, Moore et al., 2012]. Nonetheless due to the specificity of hardware solutions, the vast majority of neural simulations run instead on standard desktop computers, using custom code or domain-specific neural languages [Brette et al., 2007]. This leads to a gap between neural modellers and neuromorphic, parallel hardware, which is often

only used by its creators and a few experts. To be a generic platform for neural exploration, both atomic elements (neurons and synapses) and connection topologies must be easily configurable and extensible. Very few projects focus on a software abstraction layer, which can easily reconfigure the hardware and make it accessible to non-hardware experts [Nageswaran et al., 2007, Brüderle et al., 2011].

In this context the SpiNNaker system [Furber et al., 2006a] offers an alternative set of performances in terms of reconfigurability, scalability and power consumption, thanks to its bespoke packet switched communication infrastructure [Plana et al., 2007] that flexibly connects the ARM cores, the digital computational nodes of the platform. Efficient neural kernels can be programmed for the platform [Jin et al., 2008] to communicate using event address representation and propagate spikes in a network model [Jin et al., 2010]. The methods to distribute and configure such neural kernels on the parallel SpiNNaker system, starting from abstract definitions of the model, are the central contribution of this thesis, which make large, flexible, real-time simulation possible.

## 1.2 Motivation and Research aims

The research presented in this thesis aims to introduce an approach to map arbitrary spiking neural network models on the SpiNNaker system, with the novel set of constraints that need to be evaluated when distributing a model on the hardware platform. Cornelis et al. [2012] introduce the term *monolithic applications* when describing neural network simulators, pointing to the difficulties associated in sharing and maintaining software which integrates user interface, data access code and computational algorithms. This hampers the growth of the field, as pointed out by the authors of PyNN (a domain-specific neural network language), because *Science rests upon the three pillars of open communication, reproducibility of results and building upon what has gone before* [Davison et al., 2008]. If monolithic applications are frequent in software for neural simulation, hardware systems are often tightly optimised for a particular neural or network model, and architectural specificities require the knowledge of custom configuration software and procedures to run models on them.

To avoid building a *monolithic system*, only accessible by its creators, the components of the problem have been modularised in a flexible approach that exposes the reconfigurability of the platform to non-hardware experts through two widely used,

domain-specific simulation languages. The central requirement of the mapping approach is to be open to new hardware or modelling resources, flexibly adapting to multiple neural or network models. Further advantages in integrating the platform with languages used in the community include the possibility to verify the platform and compare the results and performance with other neural simulators. The final aim of the research is to offer SpiNNaker as an open, configurable platform for rapid experimentation of different real-time, real-world embedded complex neural network models.

### 1.3 Contributions and Publications

The main contribution of this work is the design and development of a flexible method to configure a parallel architecture such as SpiNNaker, exposing its programmability and advantages through standard, neural-oriented languages.

The main challenge of allocating models to small SpiNNaker systems is to identify the algorithms and the data structures involved in the simulation [Jin, Galluppi, Patterson, Rast, Davies, Temple, and Furber, 2010]. Using this method it is possible to simulate networks with limited numbers of neurons and neural models: scaling up the system is difficult, as it is configured with lists describing each single neuron and connection. By using neurons and synapses as mapping entities, placing and translating the model is a complex task, as the number of synapses in the system potentially increases quadratically with the number of neurons. The lists are then compiled and allocated to the platform with a *monolithic* software, which implicitly contains translation methods and system descriptions. This makes integration of new neural models [Rast, Galluppi, Jin, and Furber, 2010a], plasticity algorithms [Jin, Rast, Galluppi, Khan, and Furber, 2009] and different network topologies accessible only through a steep learning curve or by SpiNNaker experts.

The introduction of PyNN as a user interface language [Galluppi, Rast, Davies, and Furber, 2010] abstracts the platform from the user, offering the possibility to change neural and connectivity parameters rapidly. It hides all the hardware-dependent, low-level steps such as mapping the model onto a parallel system, compiling the data structures describing it and running the simulation on the platform. Networks can be configured flexibly using PyNN, making simulations with heterogeneous neural models and network topologies possible [Rast, Galluppi, Davies, Plana, Patterson, Sharp, Lester, and Furber, 2011b] (a unique feature in spiking neural hardware, which

is generally optimised for single neural models), and the mapping and translation methods have been used to run models on robots [Davies, Patterson, Galluppi, Rast, Lester, and Furber, 2010] and to validate the platform [Rast, Navaridas, Jin, Galluppi, Plana, Miguel-Alonso, Patterson, Lujan, and Furber, 2011c].

As the size of SpiNNaker machines increases from a few cores to hundreds, the original mapping and translation methods become onerous. The process is therefore redesigned [Galluppi, Davies, Rast, Sharp, Plana, and Furber, 2012d] and modularised: the process itself transforms an initial representation of the model at the neural-language level to an intermediate one where the model is placed onto the system, and then to a final representation where the model is translated and loaded on the platform. Inspired by the PyNN High Level API, *Populations* and *Projections* (rather than neurons and synapses) are identified as the fundamental model-level entities, making placing, routing [Davies, Navaridas, Galluppi, and Furber, 2012b], and translating large models possible. Hardware, software and modelling resources, along with translation methods, are separated from the mapping algorithms, and are accessed dynamically while placing and translating the model. Neural kernels are rewritten to a C-level SpiNNaker API [Sharp, Plana, Galluppi, and Furber, 2011] and become resources available to the system, making the introduction of new neural models simpler.

Such features make the mapping approach very flexible and able to be extended rapidly: exposing the system reconfigurability makes exploring different models, including alternative plasticity algorithms [Davies, Galluppi, Rast, and Furber, 2012a], easily possible. Decoupling the user interface from the mapping and translating algorithms enables the use of other neural languages and neural frameworks, such as demonstrated by the integration of Nengo and the Neural Engineering Framework (NEF) principles on SpiNNaker [Galluppi, Davies, Furber, Stewart, and Eliasmith, 2012c]. Custom NEF neural kernels are integrated in the mapping approach, while the user can interact with the platform transparently through Nengo.

Hardware resources such as larger SpiNNaker systems, AER sensors or robots are also integrated within the mapping approach, exploiting the real-time nature of the SpiNNaker platform. Such integration is demonstrated in a visual selection model with a silicon retina [Galluppi, Brohan, Davidson, Serrano-Gotarredona, Carrasco, Linares-Barranco, and Furber, 2012a], and in a cortical, basal ganglia and hippocampal place cells model running on a robotic platform [Galluppi, Conradt, Stewart, Eliasmith, Horiuchi, Tapson, Tripp, Furber, and Etienne-Cummings, 2012b]. The

mapping approach has been tested with the current generations of SpiNNaker machines (up to 864 cores), making biologically accurate [Sharp, Galluppi, Rast, and Furber, 2012] and large scale [Stromatias, Galluppi, Patterson, and Furber, 2013] simulation possible, and enabling the architecture’s scalability and power consumption [Painkras, Plana, Garside, Temple, Galluppi, Patterson, Lester, Brown, and Furber, 2012] to be evaluated. Scaling up the system required rewriting and optimization of some of the modules, such as the one responsible for writing synaptic structures. This has been optimised and parallelised, hinting at the possibility of using the abstract representation to generate such structures directly on the SpiNNaker system, exploiting its own inherent parallelism.

The research has contributed to place SpiNNaker within the computational neuroscience community by offering a flexible mapping approach, capable of accommodating new neural and computational models, and new hardware resources. The mapping approach and the software infrastructure used to configure the system, the *SpiNNaker Package*, has been used to run many models on all generations of SpiNNaker hardware produced to date, including those already described and the ones in Rast et al. [2010b, 2011a, 2013], Davies et al. [2011], Patterson et al. [2012a] and Khan et al. [2010]. The research has frequently been inspired by collaborations within the SpiNNaker group or with other groups at workshops, as one of its main goals is to integrate different research approaches into a unified translation method. Contributions (peer-reviewed and in workshops) which are relevant for the thesis are highlighted in the following sections.

### 1.3.1 Journal Papers

- A. D. Rast, J. Navaridas, X. Jin, F. Galluppi, L. A. Plana, J. Miguel-Alonso, C. Patterson, M. Lujan, and S. Furber, *Managing Burstiness and Scalability in Event-Driven Models on the SpiNNaker Neuromimetic System* International Journal of Parallel Programming, 2011. Contributed to the *doughnut hunter* experiment (section 6.2.1). Ran the experiments and write sections 6.3.1 (which can be found in this thesis in Section 5.2.6) and 6.3.2.
- A. Rast, F. Galluppi, S. Davies, L. Plana, C. Patterson, T. Sharp, D. Lester, and S. Furber, *Concurrent heterogeneous neural model simulation on real-time neuromimetic hardware*, Neural Networks, vol. 24, pp. 961-978, 2011. Conducted all the experiments and wrote section 5 of the paper; portions of it can be found

in Section 5.2.4.

- T. Sharp, F. Galluppi, A. D. Rast, and S. B. Furber, *Power-efficient simulation of detailed cortical microcircuits on SpiNNaker*, The Journal of Neuroscience Methods, Mar. 2012. Provided the PyNN to SpiNNaker mapping infrastructure. Contributed by designing and running the experiments and analysing the results.
- S. Davies, F. Galluppi, and A. D. Rast, *A forecast-based STDP rule suitable for neuromorphic implementation*, Neural Networks, 2012. Integrated the STDP kernel with PyNN in the mapping approach proposed in Chapter 5. Contributed in designing the experiments and analysing results.
- E. Painkras, L. A. Plana, S. Member, J. Garside, S. Temple, F. Galluppi, S. Member, C. Patterson, D. R. Lester, A. D. Brown, and S. Furber, *SpiNNaker : A 1W 18-core System-on-Chip for Massively-Parallel Neural Net Simulation*, IEEE Journal of Solid State Circuits, August 2013. Contributed to the results reported in sec. VII.B (which can be found in Section 5.2.6).

### 1.3.2 Conference Papers

- F. Galluppi, A. Rast, S. Davies, and S. Furber, *A general-purpose model translation system for a universal neural chip*, in Neural Information Processing. Theory and Algorithms, 2010, pp. 58-65. The paper forms part of Chapter 5.
- F. Galluppi and S. Furber, *Representing and decoding rank order codes using polychronization in a network of spiking neurons*, Neural Networks (IJCNN), The 2011 International Joint Conference on, pp. 943-950, Jul. 2011. Results from this paper can be found in Section 2.3.5 and Appendix A.
- F. Galluppi, S. Davies, S. Furber, T. Stewart, and C. Eliasmith, *Real time on-chip implementation of dynamical systems with spiking neurons*, in Neural Networks (IJCNN), The 2012 International Joint Conference on, 2012, pp. 1-8. Results from this paper can be found in Chapter 5.
- F. Galluppi, S. Davies, A. Rast, T. Sharp, L. A. Plana, and S. Furber, *A Hierarchical Configuration System for a Massively Parallel Neural Hardware Platform*, in Proceedings of the ACM 9th conference on Computing Frontiers, 2012, pp.

183-192. This publication describes the Population/Projection abstraction used to configure the system, and it forms parts of Chapter 4.

- F. Galluppi, K. Brohan, S. Davidson, T. Serrano-Gotarredona, J. A. Carrasco, B. Linares-Barranco, and S. Furber, *A real-time, event-driven neuromorphic system for goal-directed attentional selection*, in Neural Information Processing, 2012, pp. 226-233. This paper forms parts of Chapter 6.
- F. Galluppi, J. Conradt, T. Stewart, C. Eliasmith, T. Horiuchi, J. Tapson, B. Tripp, S. Furber, and R. Etienne-Cummings, *Live Demo: Spiking ratSLAM: Rat hippocampus cells in spiking neural hardware*, in Biomedical Circuits and Systems Conference (BioCAS), 2012 IEEE, 2012, p91. This work forms part of Chapter 6.
- X. Jin, F. Galluppi, C. Patterson, A. D. Rast, S. Davies, S. Temple, and S. Furber, *Algorithm and Software for Simulation of Spiking Neural Networks on the Multi-Chip SpiNNaker System*, in Neural Networks (IJCNN), International Joint Conference on, 2010.
- A. D. Rast, F. Galluppi, X. Jin, and S. Furber, *The Leaky Integrate-and-Fire neuron: A platform for synaptic model exploration on the SpiNNaker chip* in Neural Networks, (IJCNN) International Joint Conference on, 2010, pp. 18-23. Contributed to the result section, which can also be found in Appendix B.
- X. Jin, A. Rast, F. Galluppi, M. Khan, and S. Furber, *Implementing Learning on the SpiNNaker Universal Neural Chip Multiprocessor* in Neural Information Processing, vol. 5863, C. S. Leung, M. Lee, and J. H. Chan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 425-432.
- X. Jin, A. Rast, F. Galluppi, S. Davies, and S. Furber, *Implementing Spike-Timing-Dependent Plasticity on SpiNNaker Neuromorphic Hardware* Neural Networks, International Joint Conference on, pp. 1-8, 2010.
- S. Davies, C. Patterson, F. Galluppi, A. D. Rast, D. Lester, and S. B. Furber, *Interfacing Real-Time Spiking I/O with the SpiNNaker neuromimetic architecture*, in Australian Journal of Intelligent Information Processing Systems, 2010. Contributed in designing the networks and analysing results (see Section 6.3).

- A. D. Rast, X. Jin, F. Galluppi, L. A. Plana, C. Patterson, and S. Furber, *Scalable event-driven native parallel processing: The spinnaker neuromimetic system* in Proceedings of the 7th ACM international conference on Computing frontiers, 2010, pp. 21-30. Contributed to the *doughnut hunter* neural experiment.
- S. Davies, F. Galluppi, A. Rast, and S. Furber, *Maintaining real-time synchrony on SpiNNaker*, in Proceedings of the 8th ACM international conference on Computing frontiers 2011.
- A. Rast, F. Galluppi, S. Davies and S. Furber *An event-driven model for the SpiNNaker virtual synaptic channel*, in Neural Networks, (IJCNN) International Joint Conference on (2011).
- T. Sharp, L. A. Plana, F. Galluppi, and S. Furber, *Event-Driven Simulation of Arbitrary Spiking Neural Networks on SpiNNaker* in ICONIP 2011, vol. 2011, pp. 424-430. Contributed by designing and integrating of this work with the mapping approach proposed in this thesis.
- S. Davies, A. Rast, F. Galluppi and Steve Furber *A forecast-based biologically-plausible STDP learning rule*, International Joint Conference on Neural Networks (IJCNN), 2011.
- C. Patterson, F. Galluppi, A. Rast, and S. Furber, *Visualising large-scale neural network models in real-time*, in Neural Networks (IJCNN), The 2012 International Joint Conference on, 2012, pp. 1-8. Contributed by interfacing PACMAN with the visualiser, running the experiments and analysing results.
- S. Davies, J. Navaridas, F. Galluppi, and S. Furber, *Population-based routing in the SpiNNaker neuromorphic architecture*, in Neural Networks (IJCNN), The 2012 International Joint Conference on, pp. 1-8, 2012. Defined and placed on the system the entities to be routed. Integrated the routing in the mapping approach and software stack presented in Chapter 4.
- E. Stomatias, F. Galluppi, C. Patterson, and S. Furber, *Power analysis of large-scale, real-time neural networks on SpiNNaker* The International Joint Conference on Neural Networks - IJCNN 2013.

### 1.3.3 Workshops

Workshops and engagement in the neuromorphic community had a fundamental role in the research, enabling collaborations and exploration of different models and simulation strategies. Many early results and prototypes for the experiments described in this thesis have been developed during workshops; in particular:

- *Capo Caccia Cognitive Neuromorphic Engineering Workshop 2010*: introduced the PyNN-SpiNNaker integration with the LIF and Izhikevich models <sup>3</sup>; the results from this workshop form parts of Chapter 5.
- *Capo Caccia Cognitive Neuromorphic Engineering Workshop 2011*: interfacing SpiNNaker with a robot through wireless.
- *Telluride Neuromorphic Cognition Engineering Workshop 2011*: integrated the Neural Engineering Framework Principles on SpiNNaker<sup>4</sup>; the results from this workshop form parts of Chapter 5.
- *Capo Caccia Cognitive Neuromorphic Engineering Workshop 2012*: visual selection model with a silicon retina<sup>5</sup>; the results form parts of Chapter 6.
- *Telluride Neuromorphic Cognition Engineering Workshop 2012*: integrating the NEF, SpiNNaker and the robot<sup>6</sup>; first tests with the ratSLAM model<sup>7</sup>; the results from this workshop form parts of Chapter 6.

## 1.4 Thesis outline

The thesis comprises 7 chapters:

- **Chapter 2** describes how neural dynamics can be mathematically modelled, and how biologically inspired computational frameworks and models can be produced with them.
- **Chapter 3** presents different tools for neural simulation, both in hardware and software, available in the computational neuroscience community.

---

<sup>3</sup><https://capocaccia.ethz.ch/capo/wiki/2010/spinn10>

<sup>4</sup><http://neuromorphs.net/nm/wiki/ng11/results/Spinnaker>

<sup>5</sup><https://capocaccia.ethz.ch/capo/wiki/2012/spinnakerpynn12>

<sup>6</sup><http://neuromorphs.net/nm/wiki/act12/results/Combined>

<sup>7</sup><http://neuromorphs.net/nm/wiki/act12/results/OmniSpiNN>

- **Chapter 4** introduces the SpiNNaker platform and the approach to map a model onto the system.
- **Chapter 5** presents the integration of the mapping approach with two domain-specific languages, PyNN and Nengo, and contains the accuracy and performance evaluation of the system and modelling software.
- **Chapter 6** demonstrates how novel network models can be modelled on the system using SpiNNaker, PyNN, Nengo, AER sensors and a robot, within a flexible, open and efficient neural platform.
- The **Conclusions** summarizes the salient parts of the research, and presents how it can be expanded in the future to model complex neural systems.

# Chapter 2

## Computation with Spiking Neurons

### 2.1 Introduction

Improvements in recording instruments and the introduction of increasingly complex paradigms helped neuroscientists accumulate evidence of the importance of the codification mechanisms used by the nervous system to represent and process information. In particular, observation, recording and modelling of action potentials [Hodgkin and Huxley, 1952] has given rise to the hypothesis that the precise time of action potential generation [Shadlen and Movshon, 1999] of a neuron, rather than simply its firing rate [Achard and Bullmore, 2007], could be the information coding in the brain. This hypothesis is supported by a considerable amount of biological evidence related to timing precision of action potentials both *in vivo* [Lindsey et al., 1997, Chang et al., 2000] and *in vitro* [Mao et al., 2001]

Increasing the level of biological fidelity of neurons in a model of a network of neurons is an important step towards understanding how information is processed by the central nervous system, both structurally [Binzegger et al., 2004] and functionally [Dehaene et al., 2003]. A better understanding of how behaviour links to neural circuitry and synaptic transmission [Moran et al., 2011] also makes it possible to study neuro-psychological pathological cases.

Some models aim to find emerging functions from the structural data known from biology, exploiting the biological fidelity and measurable parameters of the neurons. Some quantitative descriptions of the cortex, based on anatomical data [Binzegger et al., 2009], which explore the regularity of the laminar organization of the thalamo-cortical system [Thomson and Lamy, 2007], have been proposed. Large-scale models

of the cerebral cortex based on such data have been produced [Markram, 2006, Ananthanarayanan et al., 2009]. Such models use biological data as constrained parameters for neurons and connections, and aim to reproduce higher level data such as fMRI readings, brain oscillations, and synchronization between different areas [Izhikevich and Edelman, 2008]. Other approaches can be considered more functional, where neural dynamics and quantities act as biological constraints in modelling specific cognitive functions [Dehaene et al., 1998]. Some functions can be modelled using attractor networks [Amit, 1992], networks that can represent information by settling to a (dynamically) stable state with their self-sustained, persistent activity. For example, functions such as memory can be associated to brain areas that are believed to use attractor representation, such as the hippocampus [Wills et al., 2005].

From another perspective spike based computation is also very appealing to engineers because of its computational power [Maass, 1997] and the possibility of building very power- and computationally- efficient silicon neurons [Mead, 1990] which can be used to replicate fast behavioural responses. For example Thorpe et al. [1996] have studied the speed of processing of the visual system in a recognition task, where subjects were asked to discriminate between scenes with or without animals while event related potentials (ERPs) were recorded; they found differential activity in the frontal cortex starting at 150ms and peaking at 180ms, signalling the different functional state associated with the two possible answers. More recently, Quiñ Quiroga et al. [2005] have found medial temporal lobe neurons capable of responding to high-level concepts presented visually (such as the picture of Jennifer Aniston) 300-600ms after the stimulus onset.

This chapter describes some of these models, starting from single components: neurons and synapses. Different mathematical models have been introduced to model biophysical quantities of neural systems with different degrees of accuracy and different dynamics, and are reviewed in the first part of this chapter. The rest of the chapter illustrates some structural and functional models of networks of spiking neurons, showing how information can be encoded and decoded using spike-based strategies. The models presented here form just a small subset of the ones that in the rich research environment of the computational neuroscience community.

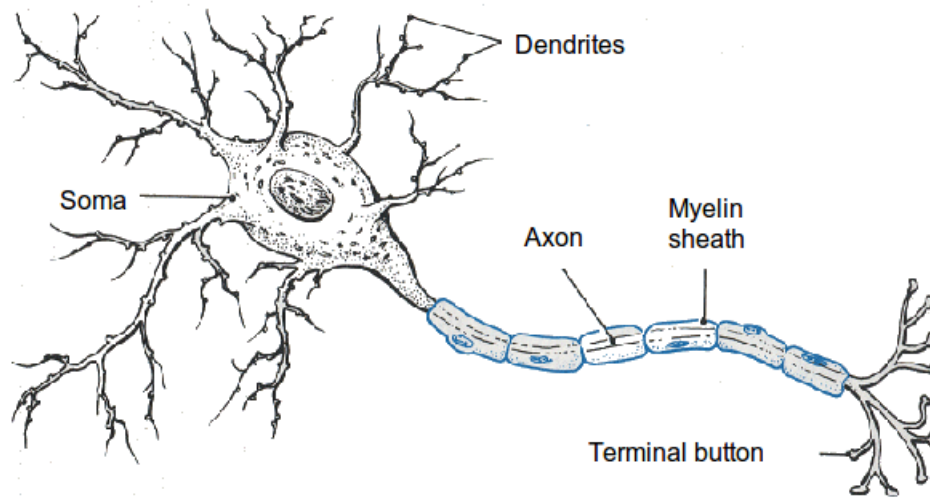


Figure 2.1: Neuron structure, from Thompson [1967].

## 2.2 Bioinspired models of neurons and synapses

Neurons are the cells that process information in the nervous system, by propagating electro-chemical signals through action potentials - stereotypical all-or-nothing signals transmitted as a variation of potential across the nervous cell. Neurons are not electrically neutral, due to the presence of ions in their cell body and thanks to dynamical modifications of the electric permeability of the cell membrane when an electro-chemical stimulation is received. The flux of ions entering and exiting the cell causes a current flow through the membrane, mostly ascribed to  $Na^+$ ,  $K^+$  and  $Cl^-$  ions [Kandel et al., 1991, Thompson, 1967].

While there are many different classes of neuron in the nervous system, each neuron can be subdivided into 4 components (Figure 2.1 from Thompson [1967]):

- **Soma:** comprises the nervous cell body, responsible for its metabolism.
- **Synapse (Terminal Button):** a structure mediating message passing between two nervous cells. Information flows in the form of electro-chemical signals (neurotransmitters) from one cell to the other; the two cells are therefore identified respectively as pre- and post-synaptic neuron.
- **Dendrites:** short nervous terminations which grow from the soma into an arborization, which widely varies accordingly to the neural type. They can be considered the input of the neuron, as they translate chemical signals carried by neurotransmitters released by the pre-synaptic neuron into an electric

signal.

- **Axon:** the nervous fibre which departs from the soma and carries the action potential towards other nervous cells. In order to rapidly carry the action potential at long distances without attenuation some axons are coated with a myelin sheath.

Synapses are the elements used to interconnect neurons in a network, permitting neural signal transmission and elaboration, by altering the electrical state of the post-synaptic cell. There are two kinds of synapses: electric and chemical. Electric synapses, also known as *gap junctions*, communicate through direct bidirectional current injection between two neurons which can be modelled using Ohm's law. The inter-synaptic cleft is usually very narrow, so it is often considered continuous. This kind of communication is very rapid but as there is no gain in the signal amplitude like in the chemical synapses, little current flows through the membrane. In chemical synapses there is no direct contact between the pre-synaptic and the post-synaptic neuron. Signal transmission is mediated by molecules called *neurotransmitters* which travel in the synaptic cleft binding to particular post-synaptic neuron receptors. This kind of transmission is slower, but amplifies the signal and can make the effects of the incoming spike last longer.

The transmission process is illustrated in Figure 2.2: the arrival of the action potential at the end of the axon induces a depolarization of the membrane opening the voltage dependent  $Ca_{2+}$  channels, which enter the cell due to the concentration gradient. Once inside, the calcium ions actively participate in the fusion of the vesicles in the inter-synaptic cleft and the consequent release of the neurotransmitter they contain. Neurotransmitters travel through the synaptic cleft and bind to the corresponding receptors situated on the post-synaptic cell. Neurotransmitters can control the opening of ion channels, hence controlling the variance of conductance of the membrane and its potential.

### 2.2.1 Neural Models

As factors governing neural activity become clearer, neurons modelling biological quantities, with much more fidelity than classical sigmoidal units, have been introduced. In particular, variations in the membrane potential, caused by the opening and closing of ionic channels, are modelled by a set of ordinary differential equations. These equations can model a subportion of the membrane or be representative

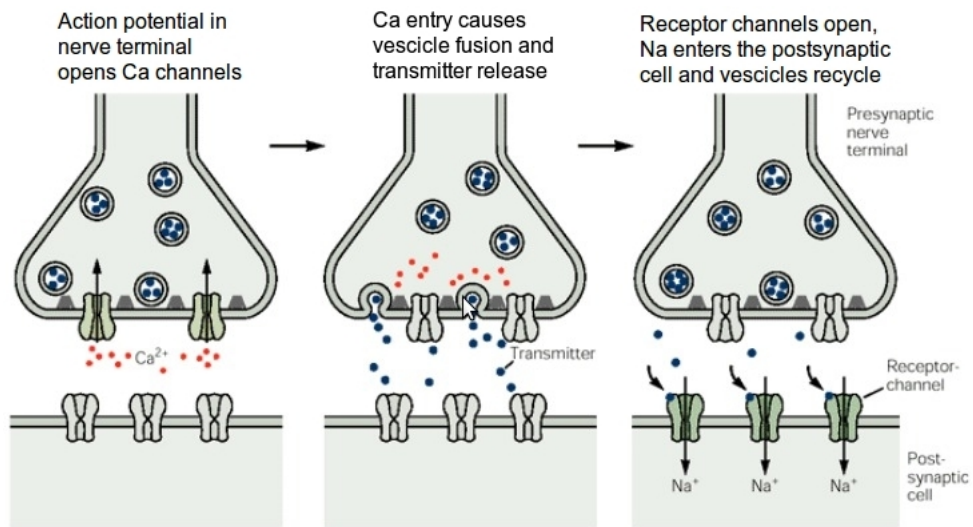


Figure 2.2: Signal transmission in a chemical synapses, from [Kandel et al. \[1991\]](#).

of a whole neuron, considered as a point.

Some models are said to be spiking as they produce action potentials (spikes), pulses of activity carrying information in the time of their occurrence [[Gerstner et al., 1994](#)]. A number of these models are now reviewed.

### Hodgkin-Huxley Model

The section starts by reviewing the Hodgkin-Huxley (HH) model, which takes into account how ionic currents shape the potential of the cellular membrane of the neuron. Based on that, models which simplify the HH model are shown; such models can simplify the original one both spatially, considering the neuron as a point, and computationally, by simplifying the equations needed to simulate the membrane dynamics of a neuron. The section is concluded by different models of synapses as observed phenomenologically.

In 1939 Alan Hodgkin and Andrew Huxley proposed a model explaining the electric fluctuation observed in a giant squid's nervous cellular membrane (Figure 2.3), by modelling the ionic mechanisms governing the generation and transmission of an axon potential as variable conductances [[Hodgkin and Huxley, 1939](#)]. All subsequent spiking neural models are based on this model; it is therefore of interest to examine this model in detail, as it constitutes a point of reference for all the other models used.

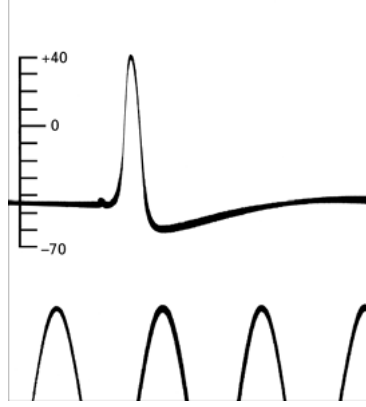


Figure 2.3: First published intracellular recording of an action potential, from [Hodgkin and Huxley \[1939\]](#).

The model divides the currents flowing through the membrane into an ionic current ( $I_{ion}$ ) and a capacitive one ( $C \frac{dV}{dt}$ ), which charge the cell membrane:

$$I = C \frac{dV}{dt} - I_{ion} \quad (2.1)$$

The ionic current takes into account the contribution of the different ionic channels present in the membrane, and can therefore be subdivided according to the ion transported such as:

$$I_{ion} = I_{Na} + I_K + I_{leak} = g_{Na}(V - E_{Na}) + g_K(V - E_K) + g_{leak}(V - E_{leak}) \quad (2.2)$$

where  $I_{Na}$  and  $I_K$  represent the sodium and potassium ionic currents and  $I_{leak}$  represents a leakage current caused prevalently by  $Cl^-$  ions; the former two can be considered as time-variable conductances, while the latter is constant.  $E_{Na}$ ,  $E_K$  and  $E_{leak}$  are called *reverse potentials*, and represent the membrane potential at which the current flowing in a certain type of channels is equal to zero. The term  $V - E_i$  it's called *driving force*, and a current can be expressed as  $g_i(V - E_i)$  [[Dayan and Abbott, 2001a](#)]. The conductance based model, so derived, can be represented in the equivalent circuit in Figure 2.4 from [Hodgkin and Huxley \[1952\]](#).

In such a model conductances can be viewed as a multitude of ionic gates which have a probability to be opened or closed (hence permitting or blocking the passage of a particular ion with its charge). It is possible to calculate the probability  $p$  of the portion of open ionic gates in a population, and their phase transaction in time as:

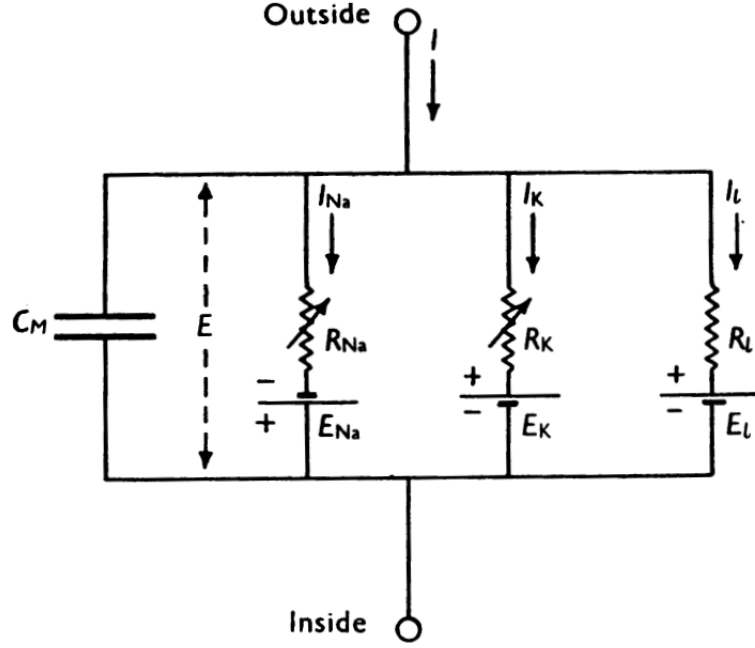


Figure 2.4: Circuit equivalent model for the HH neuron, from [Hodgkin and Huxley \[1952\]](#).

$$\frac{dp}{dt} = \alpha(V)(1 - p) - \beta(V)p \quad (2.3)$$

where  $\alpha$  and  $\beta$  are empirically determined functions of the membrane potential  $V$ , which represent the rate of transaction from a closed to an open state (and vice versa) of a particular ionic gate. A single gate's infinitesimal contribution to the overall total conductance that can be therefore be written as:

$$g = G_{max} \Pi_i p_i$$

where the conductance  $g$  is calculated as the maximum conductance  $G_{max}$  multiplied by product of the infinitesimal probabilities  $p_i$ .

Hodgkin and Huxley modelled  $Na$  conductances with 3 opening ( $m$ ) and 1 closing ( $h$ ) gate and  $K$  conductances with 4 activating gates ( $n$ ), each modelled with equation 2.3. They can therefore be written as

$$\begin{aligned} g_{Na} &= G_{Na} m^3 h \\ g_K &= G_K n^4 \end{aligned}$$

where  $G_{Na}$  and  $G_K$  are the sodium and potassium conductances respectively, and substituted in eq. 2.2

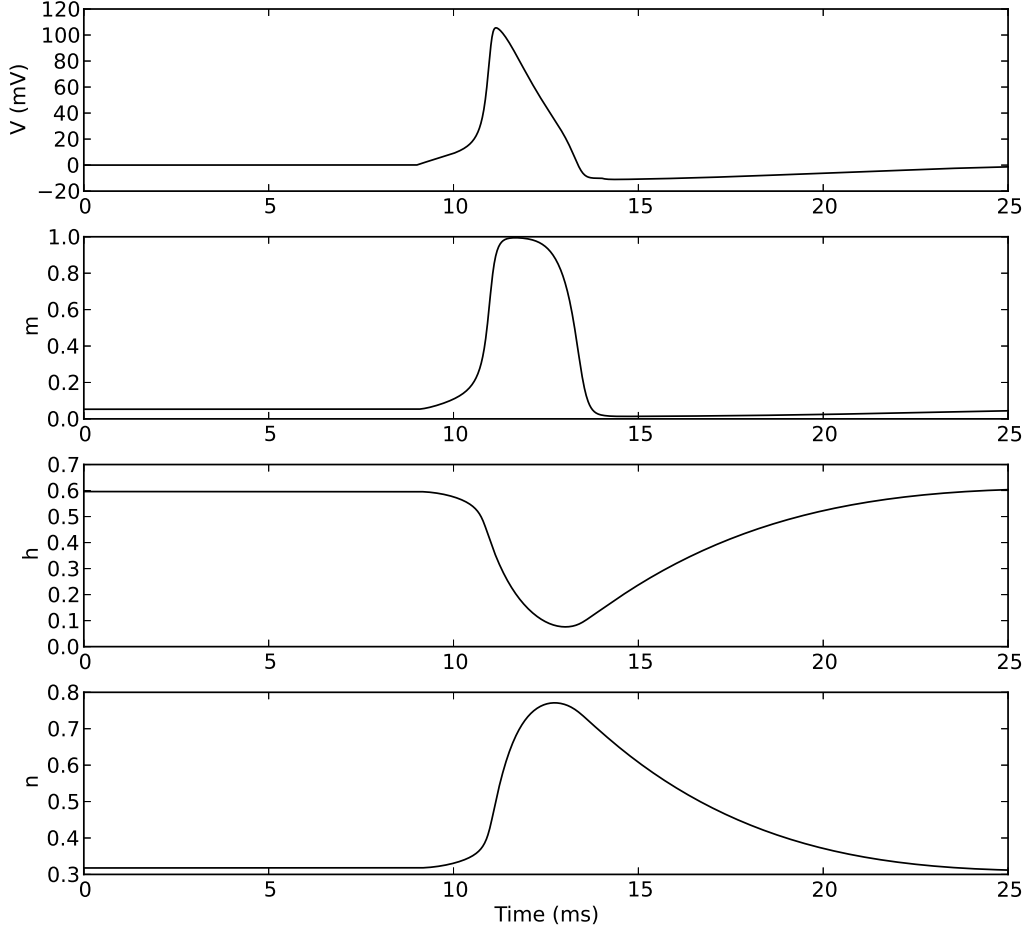


Figure 2.5: Activation and deactivation of function  $n$ ,  $m$ ,  $h$  during the generation of an axon potential. The first plot shows the membrane potential, while the remaining plots show the dynamics of the  $n$ ,  $m$  and  $h$  variables.

$$I = G_{Na}m^3h(V - E_{Na}) + G_Kn^4(V - E_K) + g_{leak}(V - E_{leak}) \quad (2.4)$$

The model can be used to explain the generation of an axon potential, as shown in Figure 2.5. During the rest state the net sum of currents flowing through the membrane potential is 0. If the membrane gets depolarised up to the  $Na$  equilibrium point  $E_{Na}$  (e.g. through synaptic inputs or by means of an electrode), this leads to an increase of opened  $Na$  gates as modelled by  $m$  which increases with  $V$  and depolarizes the membrane, leading to the upstroke of an action potential.  $Na$  gates are subsequently inactivated by the  $h$  function, which tends to 0 when  $V$  increases, pushing the membrane towards the  $E_K$  equilibrium point, hence activating the  $K$  current with a slight delay and re-polarizing the membrane (the down-stroke) below

the rest potential (after-hyperpolarization); while the function  $h$  deactivates the  $Na$  conductance, the neuron is not sensitive to any input (the cell is said to be in its *refractory period*).

The Hodgkin-Huxley neuron models the shape of the action potential explicitly, due to the inactivating function of  $Na$  channels and the activation function of the  $K$  channels. If spikes are considered as stereotypical events, carrying information in the timing of their occurrence, the shape does not need explicit modelling. Some simpler models based on this assumption have been introduced and are treated below.

### Leaky Integrate-and-Fire

As the Hodgkin-Huxley model describes, it is the openings of  $Na$  channels that lead to an action potential; this typically happens whenever the potential of the membrane rises above the voltage-dependent  $Na$  channel equilibrium point. This point can be considered a *threshold potential*, after which the membrane potential goes into the upstroke/downstroke routine generating an action potential. If that is considered an all-or-nothing, stereotypical event, carrying information only in the timing of its occurrence (and not in its shape), it is not necessary to model the exact mechanism underlying the action potential generation, as is done in the HH model by explicit modelling of the  $Na$  and  $K$  conductances; it can be abstracted away. This is the approach taken in the *Leaky Integrate-and-Fire* (LIF) model, initially postulated by Lapicque in 1907 [Abbott, 1999, Dayan and Abbott, 2001b]. The main assumption is that, whenever a *threshold potential*  $V_{th}$  is crossed, an action potential is generated and the neuron is “reset” to a potential  $V_{reset} < V_{threshold}$ , taking into account the after-hyperpolarization of  $K^+$  ions.

Consequently the model works only around the rest state of the neuron, greatly simplifying the model. The dynamics of the membrane potential can therefore be written as:

$$C \frac{dV}{dt} = I - g_{leak}(V - E_{leak})$$

$$\text{if } V > V_{threshold} \rightarrow V = V_{reset} \quad (2.5)$$

where  $I$  is an external input current, the second term of the first equation represents the dispersion towards the equilibrium potential  $E_{leak}$ ; as  $g_{leak}$  is voltage independent

the equation is usually written in the form:

$$C \frac{dV}{dt} = \frac{V_{rest} - V + RI}{\tau_m}$$

where  $V_{rest}$ ,  $\tau_m$  and  $R$  are called *resting potential*, *membrane time constant* and *membrane resistance* respectively.

The LIF model has the advantage of being a linear model and can therefore be treated analytically; it is also very computational efficient, if compared to the HH model. However it models observed variations in the threshold potential [Badel et al., 2008], and in the upstroke dynamics of an action potential poorly. Nonetheless, being the simplest, most efficient neuron to simulate, it is widely used in large scale simulations.

### Izhikevich

From a dynamical system point of view the neuron can be considered a bistable system, having two stable attractors, one for the resting state and one for the spiking regime [Izhikevich, 2006a]; the former depends on the leakage currents, while the second corresponds to the activation of the  $Na$  channels. Between the two stable attractors there's an unstable attractor, which can be considered as the threshold point of a neuron, the potential where the neuron goes from a resting to a spiking state. The cycle is completed by the outgoing flow of  $K^+$  ions which repolarize the membrane after an action potential, bringing the system towards the resting state.

It is therefore possible to reduce the HH model to a planar system; if the phase plane of the variable  $v$  representing the membrane potential, and the variable  $n$  representing the activation of  $K$  currents are considered, the space can be subdivided into 4 regions, accordingly to the 2 nullclines, as shown in Figure 2.6:

- (a)  $V$  and  $n$  increasing: both  $Na$  and  $K$  currents lead to the depolarization of the membrane and to the generation of an action potential
- (b)  $V$  decreasing,  $n$  increasing:  $Na$  currents start deactivating but  $K$  currents are still active, repolarizing the membrane after the action potential
- (c)  $V$  decreasing,  $n$  decreasing: both currents are not active, leading to an absolute refractory state
- (d)  $V$  increasing,  $n$  decreasing:  $V$  is increased (e.g. because of synaptic inputs) but the leaking currents lead the system towards the rest stable attractor

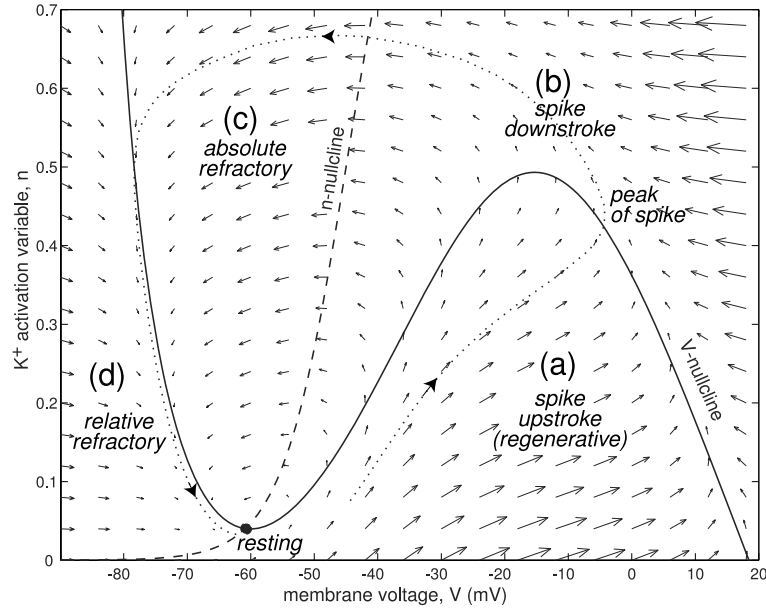


Figure 2.6: Phase-plane description of the planer system modelling the resting and spiking attractors, from [Izhikevich \[2006a\]](#).

It is possible to approximate the system with two variables, the first variable ( $v$ ) representing the membrane potential and a second ( $u$ ) accounting for the activation of  $K$  and the inactivation of  $Na$  currents. The  $v$ -nullcline can be approximated by a parabola, while the  $u$ -nullcline can be approximated with a line. As  $I$  (considered a free parameter taking into account inputs) varies, the system might undergo bifurcations, changing behaviour qualitatively. The study of the system through its bifurcations can be used to obtain a good approximation to the HH model around the rest equilibrium [[Izhikevich, 2003](#)]:

$$\begin{aligned}\frac{dv}{dt} &= 0.04v^2 + 5v + 140 - u + I \\ \frac{du}{dt} &= a(bv - u)\end{aligned}\tag{2.6}$$

with the addition of the reset conditions:

$$\text{if } v > 30\text{mV} \rightarrow v = c, u = d$$

where the parameters have the following meanings (also illustrated in Figure 2.7):  $a$  represents the decay rate of the variable  $u$ ,  $b$  represents the coupling between  $u$  and

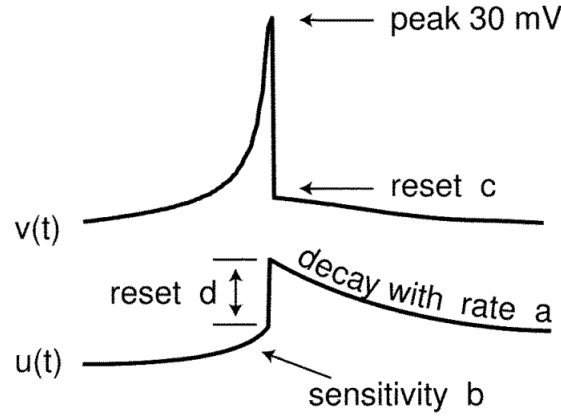


Figure 2.7: Graphical representation of the parameters in the Izhikevich model from Izhikevich [2006a].

$v$ ,  $c$  represents the reset potential for  $v$  and  $d$  represents the reset value for  $u$ .

It must be noted that the value of 30 mV does not represent a spiking threshold, but rather a cutoff value where the neuron is considered to go towards the spiking equilibrium and can then be reset, under the assumption that the shape of the action potential is stereotypical, carries no information, and therefore does not need to be modelled explicitly.

The model is particularly efficient if compared to the HH model, and can therefore be used in large scale simulations with more biological fidelity than the LIF model. In particular, if compared to the LIF neuron, the Izhikevich neuron can model a large variety of spiking behaviours (such as bursting or chattering) observed *in vitro* and *in vivo* [Izhikevich, 2004].

### Adaptive Exponential Integrate-and-Fire

The Izhikevich neuron can be considered to be an adaptive quadratic integrate-and-fire model; the adaptive exponential integrate-and-fire (*AdEx LIF* or just *AdEx* [Brette and Gerstner, 2005]) has an exponential voltage dependence, coupled with a slow variable which models threshold adaptation. The model can be described by the equations:

$$\begin{aligned}
C \frac{dv}{dt} &= -g_L(V - E_L) + g_L \Delta_T \exp \frac{V - V_T}{\Delta_T} - w + I \\
\tau_w \frac{dw}{dt} &= a(V - E_L) - w
\end{aligned} \tag{2.7}$$

with the addition of the reset conditions modelling the downstroke:

$$\text{if } v > 30mV \rightarrow v = v_{reset}, w = w + b$$

where  $w$  is the slow variable taking into account adaptation,  $g_L$  and  $E_L$  are the leaking conductance and reverse potential,  $V_T$  is the rheobase current,  $\Delta_T$  models the sharpness of the  $Na$  channels' activation function and therefore the upstroke of an action potential and  $b$  models the effects of calcium dependent potassium channels during an action potential, coupled with  $a$  which models after-hyperpolarization (subthreshold adaptation). The AdEx model has the advantage of being biologically plausible and capable of reproducing spiking behaviours such as bursting, phasic spiking etc. Its parameters also relate to empirical, measurable quantities in biology, and can therefore be compared to the response of neurons using simple electrophysiological protocols. If compared with the Izhikevich model [Gerstner and Brette, 2009], the AdEx model shares the ability to reproduce firing patterns at a low computational cost; however the AdEx shows better modelling of the spike upstroke and subthreshold dynamics.

### 2.2.2 Synaptic Models

Glutamate and GABA ( $\gamma$ -aminobutyric acid) are the most common excitatory and inhibitory neurotransmitters respectively. Ionotropic receptors for glutamate are AMPA ( $\alpha$ -amino-3-hydroxy-5-methyl-4-isoxazolepropionic) and NMDA (N-Methyl D-Aspartate receptor) [Dayan and Abbott, 2001b]. AMPA receptors are fast and their effects last the order of 10 ms. NMDA receptors by comparison have more prolonged effects on the post-synaptic neuron, on the scale of hundreds of milliseconds; they also have a unique characteristic: they are voltage-dependent. An NMDA receptor will open only if the post-synaptic neuron membrane is already depolarised.

Regarding inhibitory receptors, gamma-aminobutyric acid receptors  $GABA_A$  are fast ionotropic receptors which allow rapid injection of  $Cl^-$  ions into the cell, while

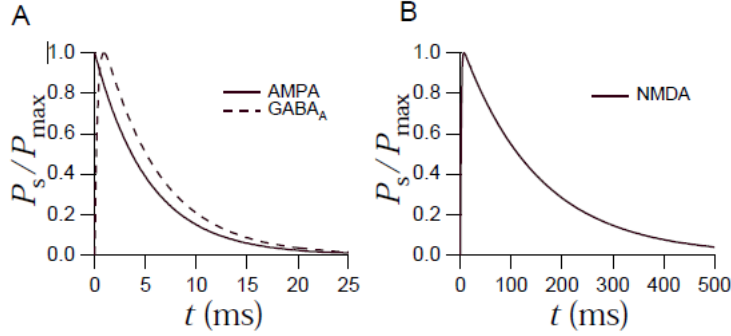


Figure 2.8: Synaptic models, from [Dayan and Abbott \[2001b\]](#).

$GABA_B$  are metabotropic receptors which lead to an outward  $K^+$  current with an effect lasting hundreds of milliseconds [[Destexhe et al., 1994](#)].

A synaptic interaction can be modelled as follows: in the first phase the neurotransmitter binds to a closed receptor and opens it. In the second phase the transmitter unbinds from the receptor, closing it. The neurotransmitter is then degraded and reabsorbed by the pre-synaptic neuron. These can be modelled as a rate of ion channel opening as in the Hodgkin-Huxley model [[Hodgkin and Huxley, 1939](#)], hence as a variation of the conductance  $g$ . When an action potential arrives at the pre-synaptic terminal, the transmitter concentration rises rapidly causing channels to open. Following the release of the neurotransmitter, diffusion out of the cleft, enzyme-mediated degradation, and pre-synaptic uptake mechanisms will lead to a reduction of the transmitter concentration thus closing the ion channels. For ionotropic receptors like AMPA the first phase can occur in the order of fractions of a millisecond, while for NMDA receptors it can take longer as shown in Figure 2.8. The variation of conductance in time can be analytically modelled with different differential equations [[De Schutter, 2009](#)]:

- **First order kinetic** (Figure 2.9a): in this case the first phase (binding of the neurotransmitter) is considered to be instantaneous, and the conductance  $g$  jumps to its maximum value upon the arrival of the spike modelled as an impulse arriving at  $t_0$ , and can therefore be written as  $\delta(t - t_0)$ . This is followed by an exponential decay with a time constant  $\tau$ . This decay can be modeled as  $g_{syn}(t) = g_{max}e^{-(t-t_0)/\tau}$  as shown in Figure 2.8(A). The differential equation describing the change of conductance in time will be

$$\tau \frac{dg_{syn}(t)}{dt} = -g_{syn} + g_{max} \cdot \delta(t - t_0) \quad (2.8)$$

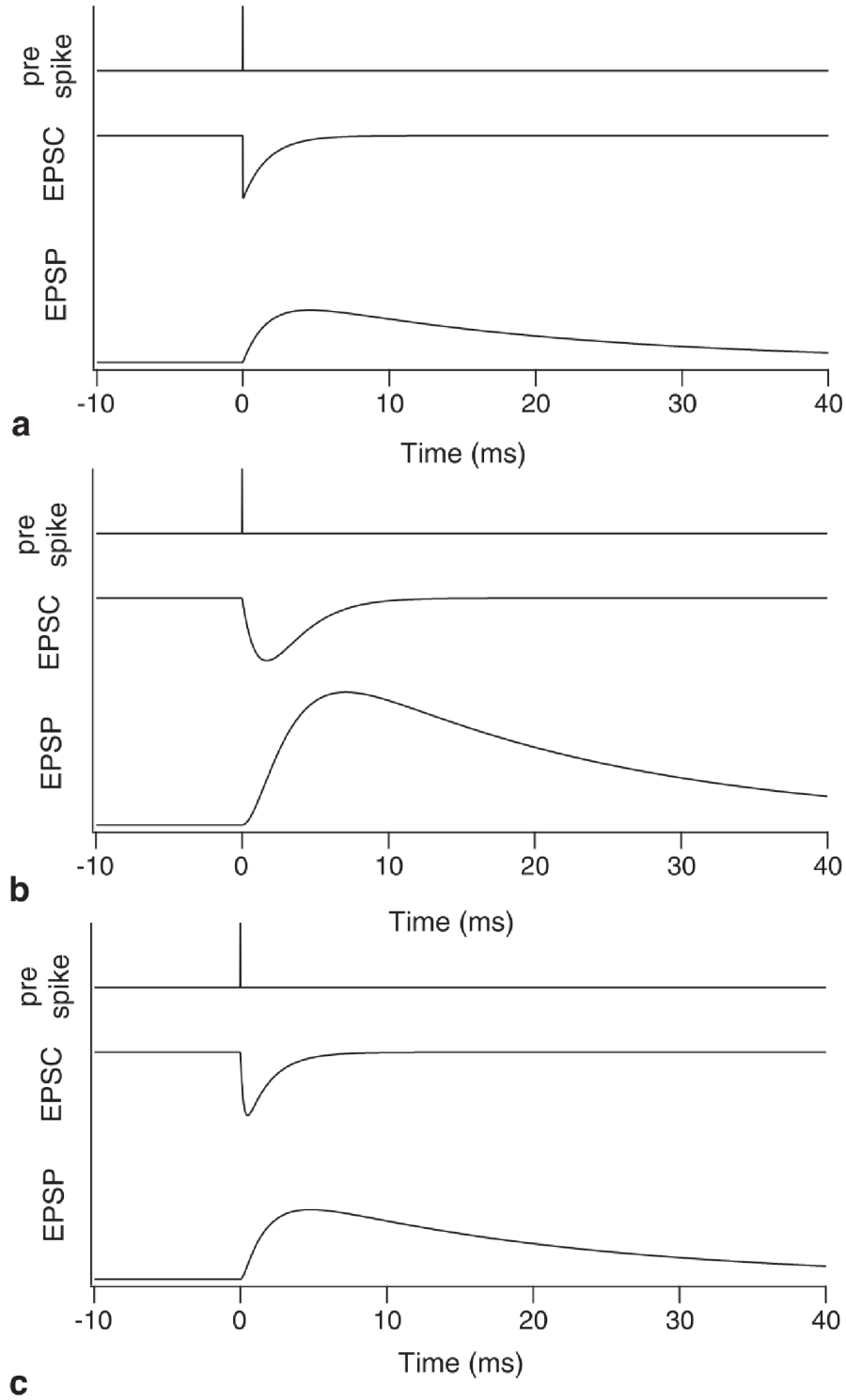


Figure 2.9: Synaptic models: first order kinetic decay with an instantaneous rise (b) alpha functions (c) difference of two exponentials, after [De Schutter \[2009\]](#). For each synapse models the variation of the synaptic current  $g_{syn}$  (EPSC - Excitatory Post Synaptic Potential) and the effects on the membrane potential  $v(t)$  (EPSP - Excitatory Post Synaptic Potential) to an incoming spike are shown.

- **Alpha functions** (Figure 2.9b): for some synapses the binding time of the receptor cannot be modelled as instantaneous, as the rising time can have impact on network dynamics [Vreeswijk et al., 1994]. The alpha functions model a conductance that has a rising phase that is not infinitely fast, but has a certain rise time and then decays exponentially. It can be described by the equation

$$g_{syn}(t) = g_{max} \frac{t - t_0}{\tau} \cdot e^{1-(t-t_0)/\tau} \quad (2.9)$$

and the differential equation will be identical to the one in the previous case.

There is a single  $\tau$  constant controlling the conductance dynamics, hence the binding and unbinding phases are correlated.

- **Difference of exponential** (Figure 2.9c): the two processes can be modelled using two separate time constants  $\tau_{rise}$  and  $\tau_{decay}$  and the conductance can be expressed as a difference between two exponentials each modelling one phase of the process as follows:

$$g_{syn}(t) = g_{max} f(e^{-(t-t_0)/\tau_{decay}} - e^{-(t-t_0)/\tau_{rise}}) \quad (2.10)$$

where the normalization factor  $f$  is included to ensure that the amplitude equals  $g_{max}$ .

NMDA synapses have another feature: their ionic channels are voltage dependent because they are blocked by  $Mg^{2+}$  ions, hence they open only if the post-synaptic neuron membrane is already depolarised. The conductance  $G_{NMDA}$  then varies with the membrane potential and different concentrations of magnesium ions  $[Mg^{2+}]$ , and this relation can be modelled [Jahr and Stevens, 1990]:

$$G_{NMDA} = (1 + \frac{[Mg^{2+}]}{2.57mM} e^{V/16.13mV})^{-1} \quad (2.11)$$

Such a relation is represented in Figure 2.10.

The property of the NMDA receptor to open only if the pre-synaptic cell and the post-synaptic are active together can be used as a coincident neural activity detector to synchronize signals between different areas. More generally speaking, the different synaptic dynamics and their effects on network behaviours are themselves subjects of research [Durstewitz, 2009].

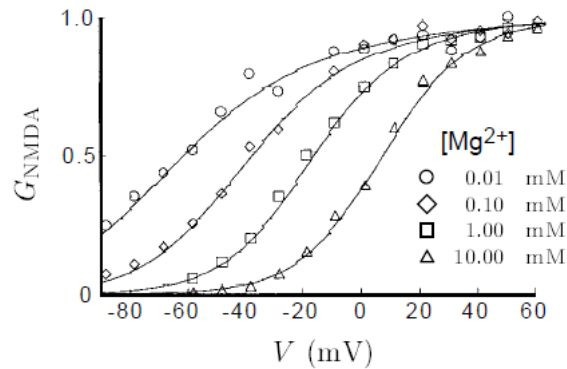


Figure 2.10: Variation of conductance with post-synaptic membrane potential in NMDA synapses, from [Dayan and Abbott \[2001b\]](#).

## 2.3 Spiking Models

The so-called third generation of neural networks [[Maass, 1996](#)] introduces a different set of functions and parameters to model neurons; these both model biological neurons more precisely [[Hodgkin and Huxley, 1952](#)] and increase the computational power of networks of neurons if compared to classical sigmoidal units [[Maass, 1997](#)]. Such networks rely on the propagation of an all-or-none signal, the action potential, which asynchronously carries information to its connected units by means of its timing.

Millisecond precise fire timing has been observed in groups of neurons recorded *in vivo* [[Mao et al., 2001](#)]; time precision and coding of pulses in time and the introduction of axonal delays have been proved to enhance the representational power of networks of spiking neurons [[Maass et al., 1991](#)]. Consequently models taking advantage of the timing distribution of spikes have been produced. Some of them, such as rank order coding [[Thorpe and Gautrais, 1998](#)] exploit the possibility to code information in the timings of the spikes in order to take account of the fast recognition responses of the visual system [[Thorpe et al., 1996](#)]: a high complex visual discrimination task (*is there an animal in the scene?*) can be performed within 150 ms, when differential activity can be detected in the frontal areas. Figure 2.11 shows the difference between correct go and no-go tasks in the ERP readings: after 150 ms from the stimulus onset (third slice) a clear difference appears in the frontal electrodes, showing two different states associated with the answer to the task. From this study

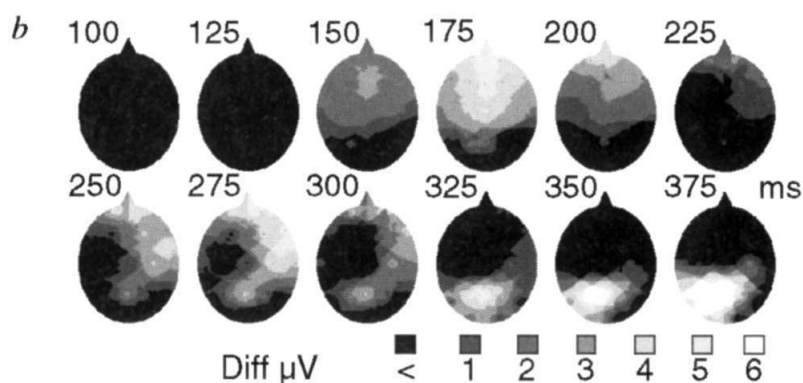


Figure 2.11: Speed of processing of the visual system in a recognition task, from [Thorpe et al. \[1996\]](#). Differential activity in ERP readings during a visual discrimination task. After 150 ms from the stimulus onset (third slice) a clear difference appears in the frontal electrodes, showing two different states associated with the answer to the task (go or no-go).

the authors conclude that at 150 ms the visual system has processed enough information to perform the task and, considering the number of visual areas involved in such processing and the neural dynamics, neurons in each stage have time to emit only one action potential, thus suggesting a time-coding mechanism. With *polychronization* [Izhikevich \[2006b\]](#) has introduced a framework to explore how convergent delays and plasticity can generate time locked group patterns, and has demonstrated its great representational power [[Izhikevich et al., 2004](#)]. It is based on the concept that post-synaptic neurons are sensitive to precise timings of their afferents due to axonal delays, which can compensate for the latencies in the input neurons and enhance post-synaptic response for particular timing patterns.

Such examples of the nervous system's ability to process an input quickly and respond with an adaptive behaviour that can be selected from a vast repertoire, along with its inherent parallelism, hint at a different computational mechanism to be explored. While some authors try to replicate biological structures to understand the underlying computational mechanisms, others focus on more functional models, proposing coding, decoding and computational frameworks based on networks of spiking neurons. In the next section some of these structural and computational models will be reviewed.

### 2.3.1 Biologically inspired models

By increasing the biological fidelity of neurons and synapses it is possible to explore and model biological data and evaluate the quality of the models by comparison. Some authors therefore try to build structures inspired by their biological counterparts and investigate their functionality. A significant number of studies have collected data on the neocortex, the site of higher cognitive activities whose structure is stunningly regular, both across its area and across different mammalian species [Binzegger et al., 2009, Thomson and Lamy, 2007]. Several models of the thalamo-cortical system have been presented in the literature with coarser or finer biological details.

Izhikevich and Edelman [2008] have produced a model inspired by the characteristic circuitry and distribution of neurons and synapses determined by Binzegger et al. [2004] and by original data from DTI studies. The model comprises one million multicompartimental neurons based on the model presented in Izhikevich [2003], different AMPA, GABA and NMDA receptors (as described in the previous section) and dopamine-modulated Spike-timing-dependent plasticity (STDP) [Izhikevich, 2007]. The model shows correlation with activity as reported in literature with Functional magnetic resonance imaging (fMRI) studies [Fox et al., 2005]. One of the results of the model which shows the importance of timing coding is that, by altering only one spike, the activity of the whole system diverges in less than a second. This result is also in accordance with the results in London et al. [2010], which present a model based on in vitro recordings, where the perturbation of a single spike generates 28 extra spikes in the post-synaptic targets; as a consequence of the latter the authors conclude that the cortex must be using a rate code, as spike-based computation is shown to be too noisy by their experiments.

A more detailed thalamo-cortical model has been presented by the Blue Brain Project [Markram, 2006], whose aim is to replicate electro-physiological behaviour observed using a multi-neuron patch-clamp approach developed within the group, on an IBM Blue Gene supercomputer [Gara and Moreira, 2011]. As of today the project has simulated a rat cortical column of 10,000 neurons and  $10^8$  synapses, running 300 times slower than real-time.

Such models explore the mysteries of the brain, but it is not clear how to evaluate their results, as the models differ radically in their level of abstraction components modelled and in the data they compare to. This methodology can be used to evaluate structural models of other areas of the brain also; for example, Humphries et al.

[2009] presented a model of the striatum, the main input of the basal ganglia, a structure involved in planning and decision, based on Izhikevich neurons. Other authors focus more on functionality: for example Dehaene et al. [2003] built a neural architecture based on cortical columns modelled with spiking neurons, and tested the network with a classical neuro-psychological task, the attentional blink Raymond et al. [1992], to link neuro-physiological data to subjective experiences (also known as *qualia*, neural correlates of consciousness [Chalmers, 1995]). Simplified models of the cortical circuitry have also been used as a computational unit for a large system or to abstract its computational properties, as for example with liquid state machines [Maass et al., 2003]. In this model a cortical column is modelled as a stereotypical recurrent circuit of leaky integrate-and-fire neurons which constitute a *liquid*, a medium in which perturbations are representative of present and past inputs. A *readout* neuron can therefore be used to decode such a (dynamical) state. The latter model shows how a structure can be used to perform computation, and serve as a bridge with models introduced in the next paragraphs, which focus more on computation.

### 2.3.2 Rank Order Coding

The rank order coding [Thorpe and Gautrais, 1998] approach has been introduced to explain the ability of the visual system to process information quickly. Spike order is used to carry the information in a feed-forward network rather than by coding it in the rate; for example to explain the fast face recognition observed in monkeys [Van Rullen et al., 1998]. With the first observable differences in the temporal lobe starting after 100 ms and several synaptic stages required to arrive to the infero-temporal cortex (IT) where object recognition happens, a rate code requiring more than one spike per stage cannot be used. Neurons have thus only one spike to code the information, and do so in the timing of emission of an action potential. Information is subsequently passed through a retinal stage comprising ON-OFF ganglion cells and a layer modelling V1 orientation selectively cells [Hubel and Wiesel, 1965]. Information is then passed to a layer trained to respond to increasingly complex combinations of features as seen in higher temporal areas such as V4 and IT.

Rank order coding has proven to be a biological effective explanation in several tasks involving image recognition [Delorme and Thorpe, 2001]; first spikes have also been considered more significant (carrying more information), confirming the possibility of fast recognition and restoration with few spikes [Sen and Furber, 2006].

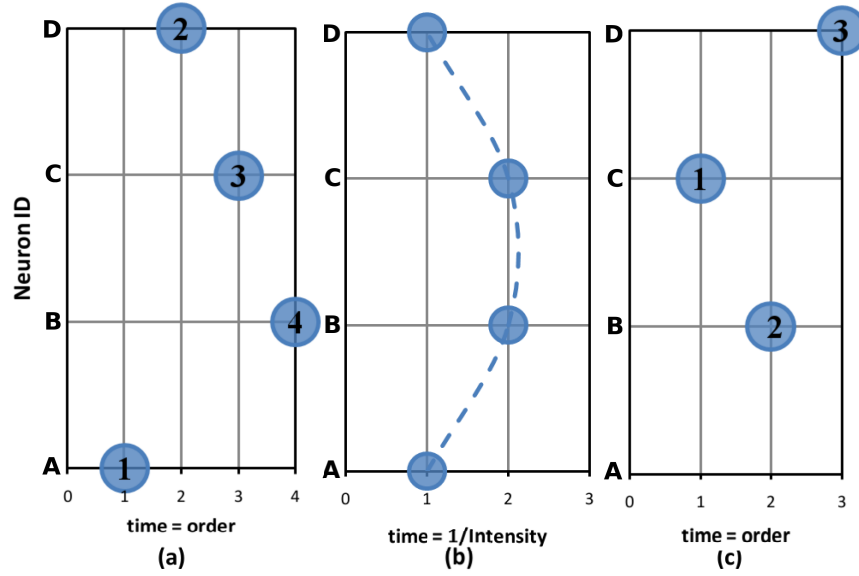


Figure 2.12: Rank order codes. (a) code with no repetition, all neurons firing; (b) intensity coded in the spike time; (c)  $N$ -of- $M$  rank order code (only a subset  $N$  of the  $M$  neurons fire).

[Thorpe et al. \[2004\]](#) have proposed a system for visual processing and object identification with SpikeNet, which is based the principles described in this section. Rank order coding also shares properties with  $N$ -of- $M$  codes which have been coded in a sparse distributed memory [\[Furber et al., 2007\]](#).

Examples of how to code information in the timings of spikes are given in Figure 2.12. A rank order code can be expressed as a sequence (with no repetition) of the input neurons (case (a), neurons firing in the order A-D-C-B). If two neurons can have the same position, the rank order code can be considered to code the intensity of the stimulation: neurons that receive a more intense stimulation fire first [\[Van Rullen et al., 1998\]](#) (case (b), neurons A and D receiving more stimulation than neurons B and C). If only a subset of the input population is considered information can be coded in a  $N$ -of- $M$  rank order code [\[Furber et al., 2007\]](#) (case (c), 3-of-4 code with neurons firing in the order C-B-D). To decode the rank order code, a neuron provided with shunting inhibition is often used [\[Chance and Abbott, 2000\]](#): by decreasing the effectiveness of a synapse as the number of spikes received, order discrimination can be achieved [\[Delorme, 2003\]](#).

### 2.3.3 Neural Engineering Framework

The Neural Engineering Framework (NEF, [Eliasmith and Anderson \[2003\]](#)) describes how biologically relevant variables can be encoded and processed in the dynamic neural activity of recurrently connected networks. This approach can be used to introduce complex control-theoretic models into spiking neural networks, including standard attractor network models [[Eliasmith, 2005](#)]. The NEF is captured by three principles:

1. **Representation** in neurons is defined by the combination of nonlinear encoding (exemplified by neuron tuning curves) and weighted linear decoding.
2. **Transformations** of neural representations are functions of variables that are represented by neural populations. Transformations are determined using an alternately weighted linear decoding which describes the transformation.
3. **Neural dynamics** are characterised by considering neural representations as control theoretic state variables. Thus, the dynamics of neurobiological systems can be analysed using control theory.

Nonlinear encoding is obtained by translating an analog value to a spike train for each neuron  $i$  in the encoding population as follows:

$$\delta(t - t_{in}) = G_i[\alpha_i \langle x \cdot e_i \rangle + J_i^{bias}] \quad (2.12)$$

where the spiking activity  $\delta(t - t_{in})$  is given by  $G_i$ , a nonlinear function describing the neural model, a gain factor  $\alpha_i$  (which in the scalar case is either 1 or -1), the value  $x$  to be encoded, the encoder  $e$  and a bias current  $J_i^{bias}$ . Encoding can be viewed as capturing the characteristic response (tuning curve) of a neuron to a specific stimulus space, demonstrated, for example, by tuning curves found in the visual system for orientation [[Celebrini et al., 1993](#)]. Tuning curves express the relation between the value of the stimulus and the spike response of a neuron, according to its preference (tuning) to the stimulus value.

An example of encoding is illustrated in Figure 2.13; this depicts the response of two example neurons out of a large population to an input stimulus  $x$ : the first neuron (blue tuning curve on the left) responds to negative values of  $x$ , by increasing its firing rate as the input tends to -1; the second neuron (green tuning curve on the right) responds to positive values of the input  $x$  instead. The response of the two neurons to

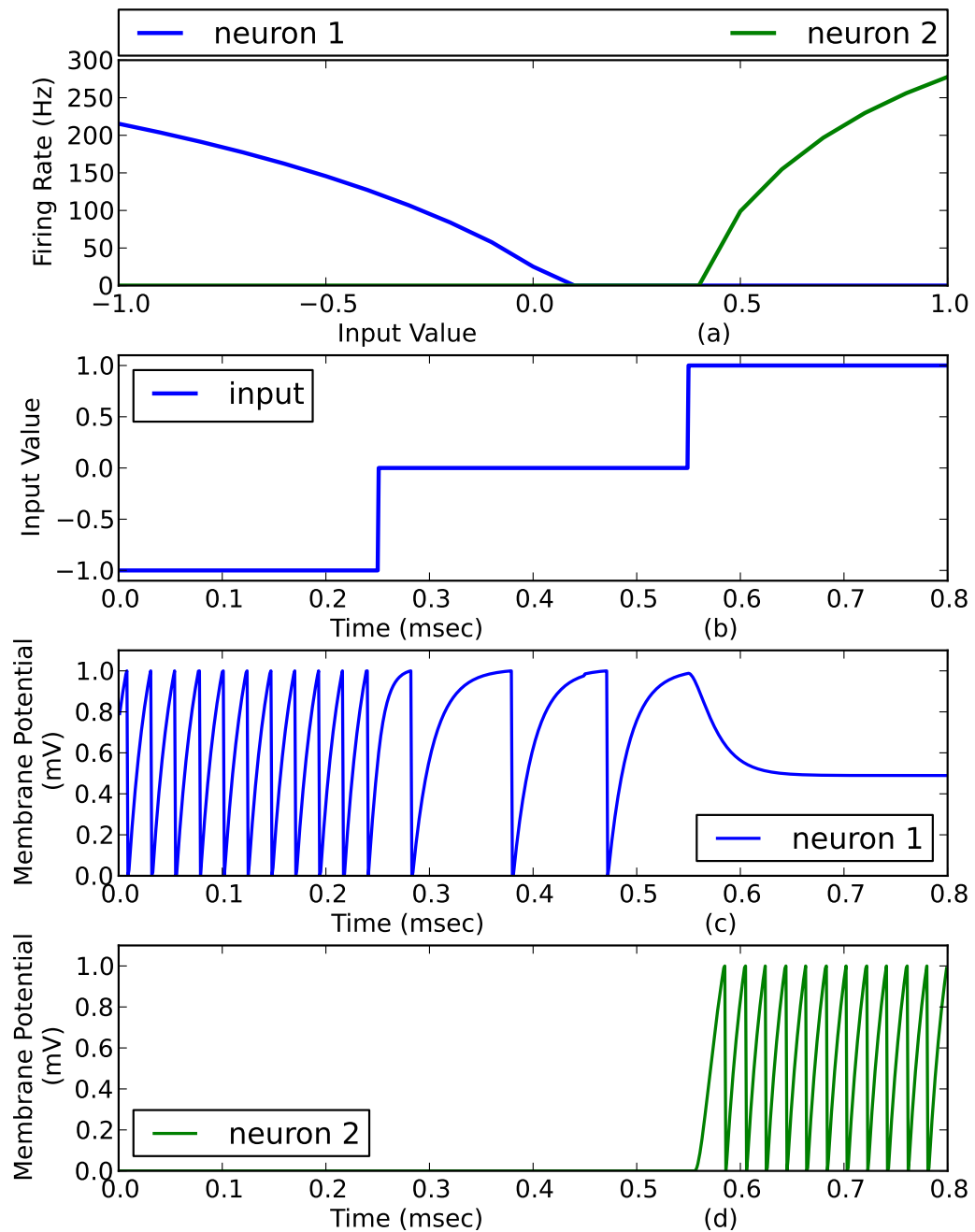


Figure 2.13: Tuning curve examples: (a) Two example neurons out of a heterogeneous encoding population to an input stimulus  $x$ : the first neuron (blue tuning curve) responds to negative values of  $x$ , by increasing its firing rate as the input tends to  $-1$ ; the second neuron (green tuning curve) responds to positive values of  $x$  by increasing its firing rate as the input tends to  $+1$  (b) input stimulation (c, d) The sub-threshold voltage response of the two neurons to a step input is shown: the first neuron fires steadily when the input is negative, while it goes silent when the input is positive, as encoded by the second neuron which starts firing steadily.

a step input is shown in Figure 2.13(a): the first neuron fires steadily when the input is negative, while it goes silent when the input is positive, as encoded by the second neuron which starts firing steadily. The original stimulus vector can be estimated by using decoding vectors  $d$  which can be found by a least square method [Eliasmith and Anderson, 2003], and coupled with a simple model of the post-synaptic current  $h(t)$  (a decaying exponential). Together, these give the following decoding scheme:

$$\hat{x} = \sum_{i,n} h(t - t_n) d_i \quad (2.13)$$

where  $h(t - t_n)$  is the convolution of the original spike train with the post-synaptic current (PSC) [De Schutter, 2009].

Encoding and decoding together define the neural population code. Conveniently, encoding and decoding can be used to compute neural connection weights. Specifically, connection weights can be calculated knowing the decoders for the source population and the encoders of the target population. Suppose the function  $y = x$  is to be computed, representations for  $x$  and  $y$  are defined as above. Connection weights to compute this function in the  $y$  population are determined as:

$$\begin{aligned} \delta(t - t_{jm}) &= G_j[\alpha_j \langle (y = x) \cdot e_j \rangle + J_j^{bias}] \\ &= G_j[\alpha_j \langle \hat{x} \cdot e_j \rangle + J_j^{bias}] \\ &= G_j[\alpha_j \langle \sum_{i,n} h(t - t_n) d_i \cdot e_j \rangle + J_j^{bias}] \\ &= G_j[\alpha_j \sum_{i,n} \omega_{ij} h(t - t_n) + J_j^{bias}] \end{aligned}$$

So, in a fully spiking network with connection weights  $\omega_{ij} = \langle d_i \cdot e_j \rangle$ , we can decode the output of the  $y$  population to determine the input values  $x$  (i.e., it is computing  $y = x$ ). Critically, decoders can also be estimated to compute an arbitrary function  $f(x)$  other than identity, thus permitting a broad class of computations through transformational decoders  $d_i^{f(x)}$ . All such computations will be feedforward, however, and it should be noted that the accuracy of the computation is dependent on the number of neurons in the two populations and their neural properties. In particular, mean squared error is proportional to  $1/N$ , where  $N$  is the number of neurons in the neural population [Eliasmith, 2005].

The introduction of neural dynamics allows variables represented by a neural population to be control theoretic state variables and hence to apply modern control

theory [Lewis and Kamen, 1994] methods to engineer and analyse time dynamics of network models, thus integrating standard attractor models and complex control [Eliasmith, 2005]. Notably, the post synaptic currents dominate the dynamics of the neural system, so a general mapping between any dynamical system and a recurrently connected network that accounts for the difference in dynamics between our PSC model and ideal integration can be determined. For the simple case of an exponential PSC and a linear time-invariant (LTI) system, recurrent and feedforward weights can be computed from the input and dynamics matrices of the LTI system.

Specifically, a widely used model for the PSC with a time constant  $\tau$  is the exponential function (as described in sec. 2.2.2) in the form  $h(t) = e^{-t/\tau}/\tau$ , which has as a Laplace transform  $h'(s) = 1/(1+s\tau)$ . The input and dynamics matrices (respectively  $\mathbf{A}$  and  $\mathbf{B}$ ) of LTI systems are transformed to their neural equivalents, obtaining:

$$\mathbf{A}' = \tau\mathbf{A} + \mathbf{I} \quad (2.14)$$

$$\mathbf{B}' = \tau\mathbf{B}$$

where  $A'$  and  $B'$  are transformation matrices describing the dynamics in the neural system. These can be included directly in the recurrent and feedforward connection weights derived above. For example, the recurrent weights would be  $\omega_{ij} = \langle d_i \mathbf{A}' \cdot e_j \rangle$ .

More generally weights are the result of the multiplication of 3 matrices:

$$\omega_{ij} = \mathbf{d}_i^{F\beta} \cdot \mathbf{M}^{\alpha\beta} \cdot \mathbf{e}_j^\alpha \quad (2.15)$$

where the terms are: the *decoding matrix* from an input population  $\beta$  which can be written as  $d_i^{F\beta}$ , decoding some function  $F$ ; the *encoding matrix* for the target population  $\alpha$  written as  $e_j^\alpha$ ; the *transformation matrix*  $M^{\alpha\beta}$  that defines the transformation between populations. In the case of feed-forward computations  $M^{\alpha\beta}$  is the identity matrix and the function is computed by the transformational decoders  $d_i^{F\beta}$ ; in the case of recurrent connections,  $\alpha$  and  $\beta$  index the same population of neurons; in the case of mono-dimensional representation encoders and decoders are scalar; in the case of multidimensional representation  $d_i^{F\beta}$  and  $e_j^\alpha$  are vectors. Functions and transformations are then defined only by weights in the neural space.

The Neural Engineering Framework offers a unified approach to building complex dynamical systems in the neural space, using only spiking neuron models, PSC models, and connection weights. These models can represent arbitrary functions,

while considering biological characteristics (e.g. tuning curves) as constraints. All parameters are estimated directly from neural data, or using the representation method described above. In this sense no parameters need be tuned, as they are either calculated using the framework or estimated from neurobiological data. The NEF has been successfully used in modelling a wide variety of neural systems including those involved in sensory processing [Eliasmith et al., 2002], motor control [J et al., 2011], and cognitive functions [Eliasmith, 2007] such as decision making, both matching experimental data (neural and behavioural), and making a variety of novel predictions [Stewart and Eliasmith, 2010].

In particular in Stewart et al. [2010b] the NEF has been used to model how the basal ganglia performs action selection and planning. The model parameters and structure are inspired by Gurney et al. [2001], and are biologically plausible. The model is tested with the Tower of Hanoi task [Stewart and Eliasmith, 2011], showing reaction times and error rates comparable to those of human subjects. The framework can also be used to manipulate language-like structures [Stewart et al., 2011] by using Holographic Reduced Representation [Plate, 1995], a type of vector symbolic architecture. Basic features of an entity (e.g. form or colour) are bounded to a realisation of the feature (e.g. square for form and red for colour) and can be combined together with a superimposition operator, leading to a compositional structure.

The functional approach taken with the NEF makes building a complex architecture in neurons possible. Recently SPAUN (Semantic Pointer Architecture Unifed Network), a cognitive architecture based on the framework principles, has been proposed [Eliasmith et al., 2012]. The model is capable of solving some classical cognitive tasks by using computational sub-modules which perform perception, cognition and action, using the same architecture. The basal ganglia module recruits some of the cognitive and action modules as the task is selected and performed. Figure 2.14 shows the architecture of SPAUN: the model consists of a working memory system, an action selection (basal ganglia) system, a visual input and a motor output system, and five general neural information processing subsystems, which are dynamically recruited by the basal ganglia to perform a particular task.

Using the same architecture and parameters, the model can perform different tasks, such as hand-written digit recognition, serial working memory recollection, addition by counting and simple question answering about an input sequence [Stewart et al., 2012]. SPAUN is a distributed architecture capable of induction, as shown in the rapid variable creation task, where a pattern is completed by generalising from

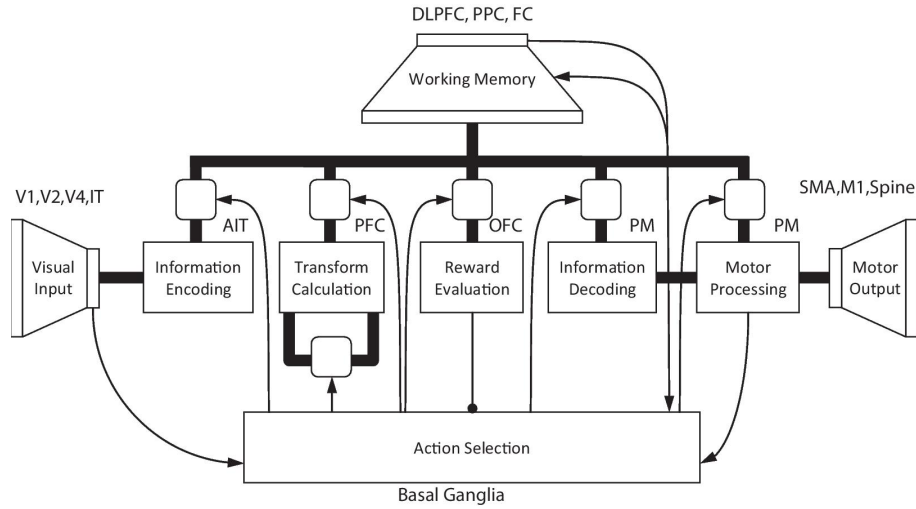


Figure 2.14: SPAUN architecture, from [Eliasmith et al. \[2012\]](#). The model consists of a working memory system, an action selection (basal ganglia) system, a visual input and a motor output system, and five general neural information processing subsystems which are recruited dynamically by the basal ganglia to perform a particular task.

previous examples. This task requires the capability of rapidly creating a symbolic variable in a syntactic structure, and binding it to unseen elements. In SPAUN the experiment is performed by presenting three example lists associated with a correct answer (eg.  $0094 \rightarrow 94$ ,  $0014 \rightarrow 14$ , and  $0024 \rightarrow 24$ ) to the model. Generalisation by induction is tested by presenting a new list ( $0074$ ), to which the model produces the right answer ( $74$ ). The task shows generalisation after a few input examples, and does not require neural rewiring as the syntactic structure and the variable are represented in the working memory and in the Transform Calculation cortical component, which drives the basal ganglia information routing to solve the task.

### 2.3.4 Polychronization

While rank order coding is used to exploit temporal properties in the source (pre-synaptic) neurons, polychronization can be used to organize and tune post-synaptic neurons to respond by compensating time differences in the inputs with the axonal delay, so as to increase their response [[Izhikevich, 2006b](#)]. Polychronous neurons will then be sensitive to certain time locked input patterns, but not to others.

The idea of polychronization is illustrated in its simplest form in Figure 2.15. The neurons are interconnected with different delays so that only precise time patterns of the input neurons will have their inputs converge into the output neurons at the

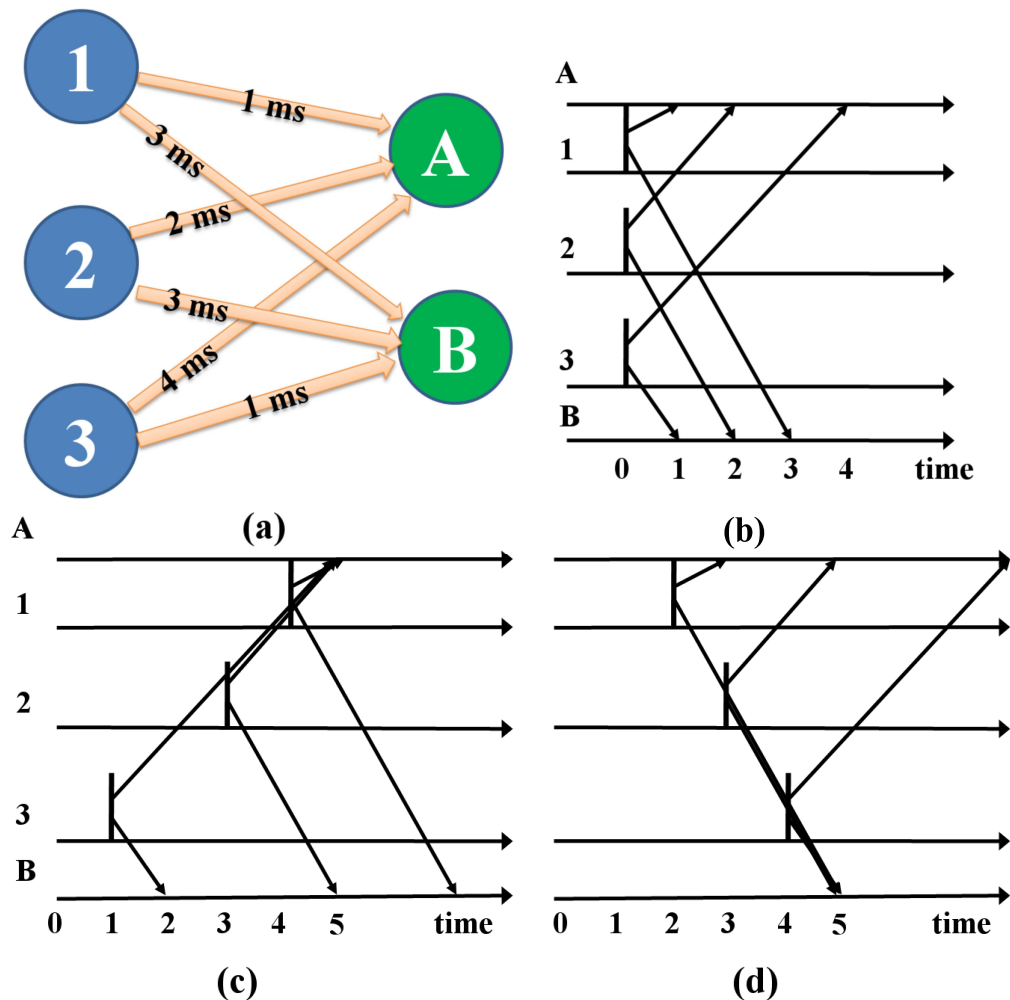


Figure 2.15: Polychronization: (a) example network: neurons 1-2-3 connect to neurons A-B with conduction delays indicated on the arrows; (b-d) vertical lines denote pre-synaptic neuron (1-2-3) spike timing; arrows point to the spike time arrival at the post-synaptic neurons (A-B); (b) synchronous firing of the pre-synaptic neurons does not elicit a strong PSP in either of the post-synaptic neurons, because the spikes arrive at different times due to different conduction delays; (c) if pre-synaptic neurons fire in the order 3-2-1 their spikes converge to neuron A due to compensation through the axonal delay, and have no effect on neuron B; (d) if input neurons fire in the reverse order (1-2-3) their spikes will converge onto neuron B and arrive in sparse order to neuron A.

same moment.

An example network is presented in Figure 2.15: neuron A receives input from neurons 1, 2 and 3 with delays 1, 2 and 4 ms; neuron B receives inputs with delays 3, 2 and 1 ms respectively (b) if input neurons fire together their contribution arrives in sparse order to the post-synaptic neuron (green) eliciting a weak response (c) if neurons fire in a precise time pattern (at ms 4, 3 and 1 respectively) their spikes converge to neuron A due to compensation through the axonal delay while having no effect on neuron B (d) if input neurons fire at 2, 3 and 4 ms their spikes will converge onto neuron B and arrive in sparse order to neuron A.

The emergence of groups in regular structures has also been explored in [Izhikevich et al. \[2004\]](#). When a plastic randomly interconnected spiking network with delays is stimulated, the activity self organizes into polychronous groups thanks to the interplay between delays and plasticity. This mechanism leads to an increased representational power of the network, which can have more groups than neurons, as each neuron can be a subpart of a different polychronous group, while also considering time dynamics and the ability to represent an internal state as the persistent activity of some groups, as observed *in vivo* [[Chang et al., 2000](#), [Tetko and Villa, 2001](#)].

The abstraction of polychronization from neurons into a computational framework is shown in [Izhikevich and Hoppensteadt \[2009\]](#). In this work pulses of temporal and spatial patterns of activity propagate as circular waves in a medium comprising transponders - agents which generate a pulse upon the receipt of more coincident pulses. Transponders are reminiscent of neurons in a network, but their biological properties have been stripped away, going towards an event based computational method called *Polychronous Wavefront Computation*. As the author remarks this framework cannot be characterised as a neural network, but rather as a framework where "*the effect of each pulse depends exclusively on its timing relative to other pulses and the location of transponders*"; in other words, it's a completely event-driven framework where information is encoded in timing.

### 2.3.5 Rank order codes and polychronization combined

As a conclusion to this section an approach where rank order coding and polychronization are considered as complementary is presented [[Galluppi and Furber, 2011](#)]; the first model is used by an input population to code information and the second by a higher dimensional population of post-synaptic polychronous neurons. This population is able to represent the codes presented in the pre-synaptic input layer, and a

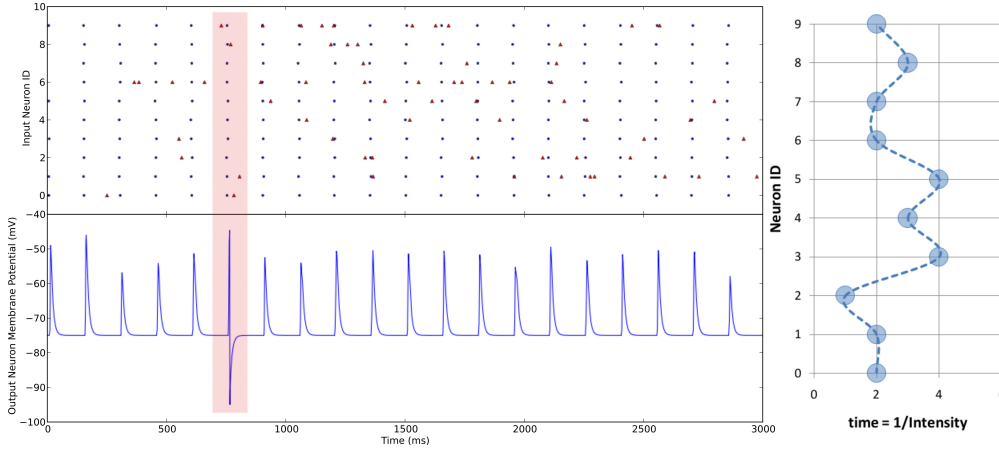


Figure 2.16: The output neuron fires only when the learned code is presented.

neuron can be trained to discriminate an arbitrary selected code. Rank order coding can be used to encode information in the spike timings of an input population and polychronization can be used by a post-synaptic neuron to decode a temporal pattern emitted by a source population by compensating the latencies with the axonal delays.

Polychronization, then, seems the natural candidate to decode rank order coded inputs. By observing again Figure 2.15 firing patterns in neurons 1, 2 and 3 can be modelled by two distinct rank order codes, and the two output neurons A and B as responders to a particular rank order code. In theory, a network exploiting connection delay values can be devised to compute the number of neurons requested and their combination. Instead, in the approach proposed, a polychronous layer is connected to the input source with random delays. This layer acts as a spiking neural implementation of a sparse distributed memory [Furber et al., 2007] in spiking neural networks which is able to store and recall rank order codes.

Results show that the mechanism can be used to encode an input with a rank order which can be decoded by a neuron connected to a polychronous layer, even in the presence of noise, as shown in Figure 2.16. The model details are presented in Appendix A.

## 2.4 Summary

This chapter has presented how biophysical properties of neural components can be characterised mathematically into models of neurons and synapses. Computational models were then presented along with discussions of the mechanisms used to encode and decode information in spiking neurons. The models reviewed in this chapter are just a small subset of a very large number of different models proposed within the modelling computational neuroscience community, working at different levels of abstraction - from molecules to behaviour [Sejnowski, 2003]. Basic elements - neurons and synapses - were introduced and it was discussed how models relate to their biological counterparts. Some computational properties that emerge when such neurons are coupled together in networks were presented, first by examining models that try to mimic the structure of the thalamocortical system, then by shifting progressively towards models where structure supports computation, showing how the selection of a particular neural, synaptic or network model is dependent on the scientific question being investigated.

The variety of approaches in neural modelling are reflected in the growing number of different emerging tools for spiking neural simulation. Some tools are oriented to the simulation of neurons in great detail [Hines and Carnevale, 2001, Bower and Beeman, 2007], others are more oriented to the simulation of large networks [Gewaltig and Diesmann, 2007], others try to capture a simulator-independent description of the network [Raikov et al., 2011, Davison et al., 2008, Goddard et al., 2001]. Some models are too computationally demanding to be simulated on a standard computer, and therefore a variety of different hardware solutions have been proposed.

The next chapter reviews the diversity of these approaches and challenges, and solutions proposed to simulate networks of spiking neurons, in hardware and software.

# Chapter 3

## Tools for Neural Simulation

### 3.1 Introduction

Simulation of networks of spiking neurons can follow two distinct paths. The first is simulation on standard computers, using proprietary neural code (written for example in C) or domain specific simulation languages and engines (such as NEST [Plesser et al., 2007] or Brian [Goodman, 2008]). The advantages of using standard simulators are in interoperability and ease of use, since they abstract all the network instantiation and programming management from end users; this comes at the cost of efficiency [Brette et al., 2007]. Since simulating large scale networks of biologically plausible neurons is a challenging task requiring scalable computational and communication resources, a second approach is to run the simulation on dedicated hardware platforms. As a consequence, different hardware approaches have been proposed to simulate neural networks. Some models are simulated taking advantage of recent developments in computational infrastructure that can be scaled up. Therefore simulations usually take place on supercomputers [Ananthanarayanan et al., 2009], general purpose hardware such as FPGAs [Maguire et al., 2007] or dedicated neuromorphic hardware [Schemmel et al., 2010, Wijekoon and Dudek, 2008]. However, access to hardware resources is often limited, and tailored to a specific platform or model [Choi et al., 2004]; every approach has different scalability, programmability, precision and power consumption characteristics.

This chapter will present the different approaches for simulating spiking neural networks, by first introducing different software simulators oriented to different modelling needs, and some description languages, which try to standardize model definitions to encourage sharing between research groups. The rest of the chapter

describes different approaches to simulate spiking neural networks, on dedicated or off-the-shelf parallel hardware, showing differences and characteristics of each system.

## 3.2 Software Simulators

The range of behaviours and details that computational neural scientists need to model is very large and, not surprisingly, many scientists prefer to build their own simulation environment using Matlab, C++ or Python, as for example SpikeNet, a visual processing system proposed by Thorpe et al. [2004]. This tendency is also reflected in a survey done by the NeuroDebian Initiative<sup>1</sup>, with Matlab and C++ being the most commonly used languages. However the lack of a standard way to describe neural networks and results [Crook et al., 2012] limits the possibility for the field to grow by discouraging reuse of software and replication of modelling results, as well as validation of existing models [Cannon et al., 2007, Nordlie et al., 2009]. To support modellers some domain-specific, neural-oriented languages and simulators have been developed; such simulators have the advantage of being reusable and exchangeable between different groups, promoting sharing of knowledge and validation of models. These languages and simulators offer standard conceptual entities (neuron, synapses, connections, probes etc.) to build a neural network, either with a scripting language or with a GUI, and to control the entities, run experiments, collect and analyse data. They often come with implementations of standard models such as those presented in the previous chapter, and some have the possibility to extend the neural model library. Their key-concept is to offer the user abstracted, domain-specific, conceptual entities such as neurons and synapses, translating them into the appropriate mathematical form to solve such equations numerically, offering an abstract entry point for writing neural network models which are then easy to share and build upon. In the last years the introduction of standard packages [Davison et al., 2008, Raikov et al., 2011] has increased the standardisation in the field, towards a unified way to describe spiking neural network models and run experiments with reproducible, shareable data. The most used simulators for networks of spiking neurons accordingly to the NeuroDebian survey cited beforehand will be reviewed in this sections.

---

<sup>1</sup>For complete results check <http://neuro.debian.net/survey/2011/results.html>

### 3.2.1 NEURON

NEURON [Carnevale, 2007] is a modelling and simulation tool which is oriented to the simulation of very fine detailed neurons which can then be connected in complex networks [Hines and Carnevale, 1997]. It lets the user model the surface of multi-dendritic neurons, yielding complex, detailed neuron models. The simulator offers users the possibility to work with abstract models, while the internal engine offers the infrastructure to simulate it. NEURON is written in a proprietary language called NMDOL which has Python bindings open for extension [Hines and Carnevale, 2000]. It also offers a GUI which lets the user select the parameters of the model and run them on parallel hardware. It is a very appealing instrument for neuroscientists because it helps model the physical properties of the neurons, but also for network modelling if expanded through the Python bindings; it also supports some degree of parallelization [Brette et al., 2007]. NEURON is a widely used tool with a large userbase, with more than 860 publications recorded by 2009<sup>2</sup>.

### 3.2.2 GENESIS

GENESIS (the GEneral NEural Simulation System - Bower et al. [1998]) is a tool for realistic simulation, based on physical structures and biological properties [Bower and Beeman, 2007]. Users can configure abstract objects and can interact with the simulation, even with the use of a GUI, while the computation is carried on in the underlying GENESIS simulation engine written in C. The structure of GENESIS is shown in Figure 3.1, where it can be seen that the script interpreter can be used to drive the internal simulation engine and communicate with the user front-end, hence separating the different concerns involved in the simulation. GENESIS has a long history of simulation and is now being renewed with the GENESIS-3/Neurospaces initiative, using separation of concerns as a key design principle [Cornelis et al., 2012].

### 3.2.3 NEST

NEST (NEural Simulation Tool) [Dupuy et al., 1990, Gewaltig and Diesmann, 2007] is a tool able to simulate efficiently networks composed of standard neuron models as described in Section 2.2.1. It is oriented to the simulation of networks with more than

---

<sup>2</sup>[http://www.neuron.yale.edu/neuron/what\\_is\\_neuron#userbase](http://www.neuron.yale.edu/neuron/what_is_neuron#userbase)

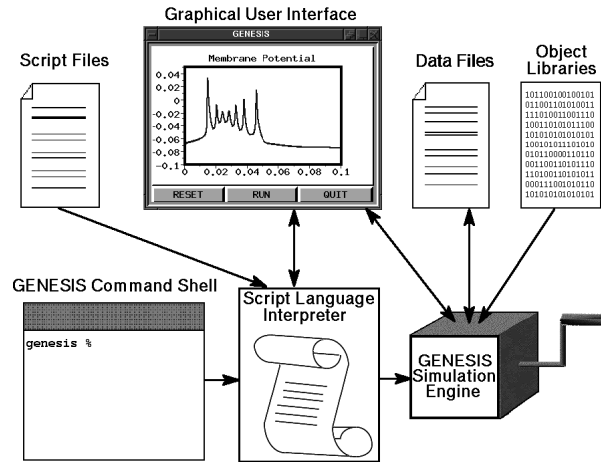


Figure 3.1: GENESIS structure, from Bower et al. [1998]. Different modules performing user control, data representation visualisation and transformation are separated following the separation of concerns principle [Cornelis et al., 2012].

$10^4$  neurons and  $10^7 - 10^9$  synapses. NEST can be programmed using SLI, a stack-based custom language; however recently a Python interface has been introduced to make the simulator more accessible [Eppler et al., 2008]. NEST uses nodes (neurons, groups of neurons and devices) and connections between nodes to model networks of spiking neurons interconnected arbitrarily, along with synaptic models and plasticity methods. The simulator has a good user base and publications, demonstrating its use as a simulation tool<sup>3</sup>; in Potjans et al. [2011] for instance, NEST is used to model the effects of dopamine on temporal difference learning in a biologically plausible architecture. Nest offers a Message Passing Interface (MPI) optimisation to distribute the model on cluster computers [Plesser et al., 2007], exploiting the parallelism of the architecture to run large networks efficiently [Morrison et al., 2005]. This approach is however limited by the collective communication functions adopted for the MPI data exchange scheme [Eppler et al., 2007].

### 3.2.4 Brian

Brian [Goodman, 2008] is a neural network simulator entirely written in Python oriented to the simulation of large networks of point neurons. The choice of Python as a native language (rather than using C/C++ like GENESIS and NEST) lets the user run simulations and plot and analyse data in the same environment; this is also equipped with efficient Python packets for numeric and vector-based computation

<sup>3</sup><http://www.nest-initiative.org/index.php/Publications>

```

from brian import *
eqs = '''
dV/dt = (ge+gi-(V+49*mV))/(20*ms) : volt
dge/dt = -ge/(5*ms) : volt
dgi/dt = -gi/(10*ms) : volt
'''
P = NeuronGroup(4000, model=eqs,
               threshold=-50*mV, reset=-60*mV)
Pe = P.subgroup(3200)
Pi = P.subgroup(800)
Ce = Connection(Pe, P, 'ge')
Ci = Connection(Pi, P, 'gi')
Ce.connect_random(Pe, P, p=0.02,
                 weight=1.62*mV)
Ci.connect_random(Pi, P, p=0.02,
                 weight=-9*mV)
M = SpikeMonitor(P)
P.V = -60*mV+10*mV*rand(len(P))
run(.5*second)
raster_plot(M)
show()

```

$$\tau_m \frac{dV}{dt} = -(V - E_L) + g_e + g_i$$

$$\tau_e \frac{dg_e}{dt} = -g_e$$

$$\tau_i \frac{dg_i}{dt} = -g_i$$

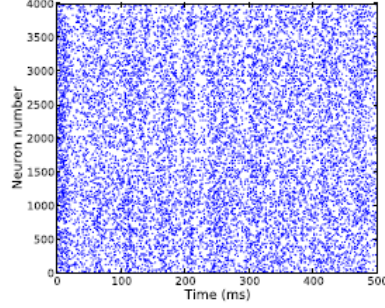


Figure 3.2: Example code and execution results from Brian, from [Goodman \[2008\]](#). The Brian code implementing the model is shown on the left: 3,200 excitatory and 800 inhibitory neurons, organised in 2 subgroups, are recurrently and sparsely connected. The right part of the figure shows the mathematical form of the equations implemented and the results of the script execution.

such as Scipy and Numpy; while the integration with PyLab lets the user easily create graphics, integration with SymPy introduces dimensionality checks. With Brian a user can directly input the equations to be solved by the simulator in standard notation or use a standardised neuron model, making the simulator very flexible.

Brian is an excellent choice as an entry simulator as it offers a very expressive Python based language which is also very efficient (25% slower than C for large networks) and also supports the use of cluster computers or GPUs [[Brette and Goodman, 2012](#)] as a parallel platform for simulation. Figure 3.2 shows an example Brian program, the mathematical methods used to model neurons and the result of the simulation. The network comprises 3,200 excitatory and 800 inhibitory neurons recurrently and sparsely connected, and is often used as a *neural benchmark* [[Brette et al., 2007](#)].

### 3.2.5 Nengo

Nengo [[Stewart et al., 2009](#)] is the tool used to implement the Neural Engineering Framework principles (see sec. 2.3.3), giving the user the possibility to map a wide range of neuro-computational dynamics to spiking neural networks; it is a software specific to a modelling framework rather than general purpose. Nengo offers a graphical user interface and a scripting language to build neural groups and interconnects

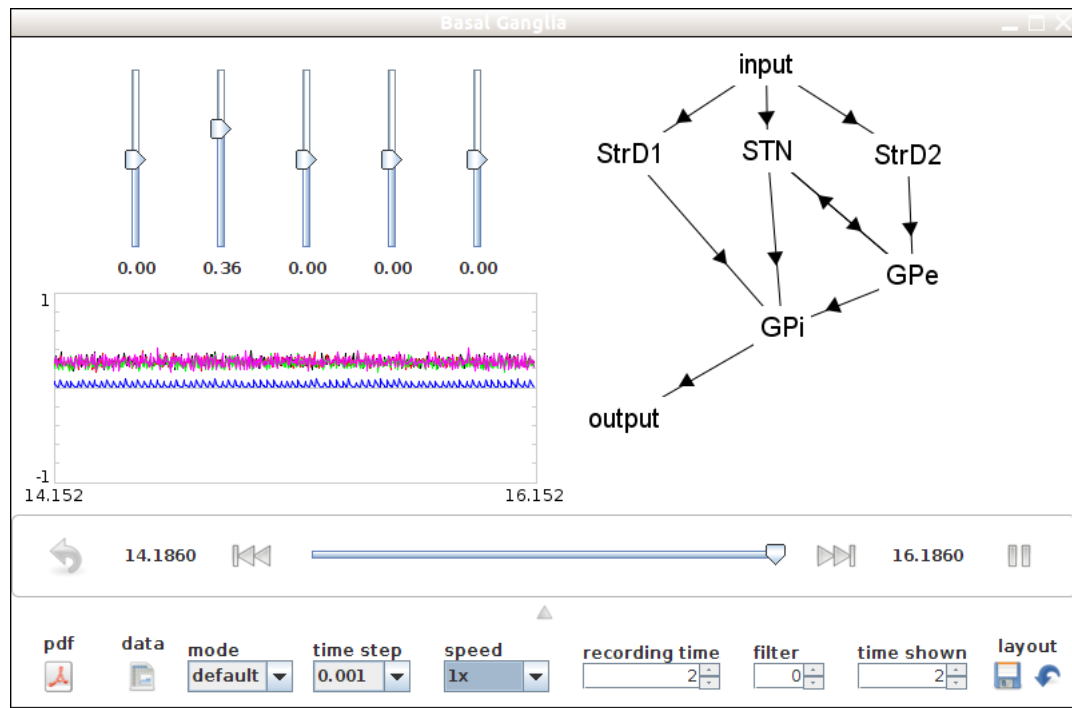


Figure 3.3: Example of a basal ganglia network model running in Nengo. Action selection is performed by a basal ganglia module by evaluating the value of five different actions.

them based on the desired function, computing all the neural parameters and connection weights. It also offers a benchmark to interact with running simulations and save data as shown in Figure 3.3, where the simulation workbench is used to control the input into a basal ganglia model [Stewart and Eliasmith, 2011]. Nengo is very easy to use overall, and coupled with the ability of the NEF to calculate appropriate connection weights for functions and dynamical systems, it allows the rapid building of complex networks. A simple network, where a neural population  $B$  is performing the square of the value represented by another neural population  $A$ , is described in Nengo with the following code<sup>4</sup> (see Appendix C.2.1):

```
import nef

# Create the network object
net = nef.Network('Squaring')

# Create a controllable input function with a starting value of 0
net.make_input('input', [0])
```

<sup>4</sup><http://ctnsrv.uwaterloo.ca/docs/html/demos/squaring.html>

```

#Make a population with 100 neurons, 1 dimension
net.make('A',100,1)

#Make a population with 100 neurons, 1 dimensions
net.make('B',100,1,storage_code='B')

net.connect('input','A') # Connect the input to A

# Connect A and B with the defined function approximated in that connection
net.connect('A','B',func=lambda x: x[0]*x[0])
net.add_to_nengo()

```

The model shown in Figure 3.3 can be written in Nengo as follows<sup>5</sup> (see Appendix C.2.1):

```

import nef
import nps

D=5
net=nef.Network('Basal Ganglia') #Create the network object

# Create a controllable input function
# with a starting value of 0 for each of D dimensions
net.make_input('input',[0]*D)

# Make a population with 100 neurons, 5 dimensions, and set
# the simulation mode to direct
net.make('output',1,D,mode='direct')

#Make a basal ganglia model with 50 neurons per action
nps.basalganglia.make_basal_ganglia(net,'input','output',D,
    same_neurons=False, neurons=50)

net.add_to_nengo()

```

A five dimensional input, representing the value of five different actions, is created and connected to a basal ganglia module which performs action selection.

### 3.2.6 Towards a Standard

The proliferation of different user and standard languages, and the difficulties in exchanging models and verifying results have led some research groups towards the creation of representations for spiking neural network models.

One of the first moves in that direction has been introduced with NeuroML [Goddard et al., 2001], an XML based language able to describe models, data and methods in neurocomputational disciplines, storing the model specification and its formalism.

---

<sup>5</sup><http://ctnsrv.uwaterloo.ca/docs/html/demos/basalganglia.html>

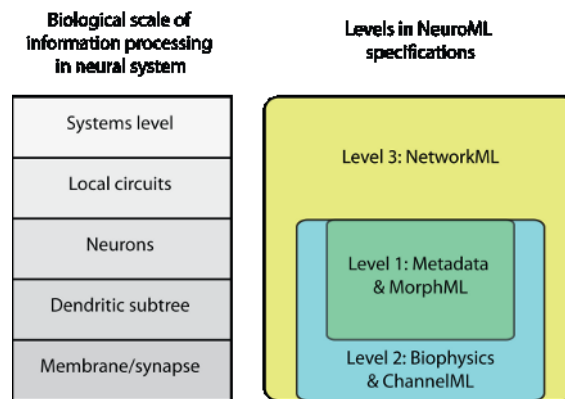


Figure 3.4: NeuroML structure, from [www.neuroml.org/introduction.php](http://www.neuroml.org/introduction.php).

NeuroML is a declarative language in the sense that it describes components of the model, including imperative portions of code to be inserted in standard interfaces. Its main goal is to be a vehicle of information between scientists. It aims therefore to be clear, portable and modular, and is represented at different levels which describe the following properties:

- **MorphML**: neuroanatomic properties of the neuron, such as morphology of the soma and of its dendritic tree[S et al., 2005]
- **ChannelML**: synaptic interactions, ionic channels and extracellular ionic concentration models
- **NetworkML**: describing a 3D network of neurons and their interconnections

This need has more recently been addressed by the Multiscale Modeling program of the International Neuroinformatics Coordinating Facility (INCF<sup>6</sup>) with the introduction of NineML (Network Interchange for Neuroscience Modeling Language [Raikov et al., 2011]), a declarative language for describing network models of point spiking neurons. It identifies as conceptual entities neurons, synapses, populations of neurons and patterns of interconnection between them (projections) and defines their mathematical abstractions. NineML comprises an *Abstraction Layer* which defines the entities and a *User Layer* which lets a user build a model using such entities. Production of interfacing facilities with Python and XML are provided, to be supported by the available simulation packages already described.

<sup>6</sup><http://www.incf.org/>

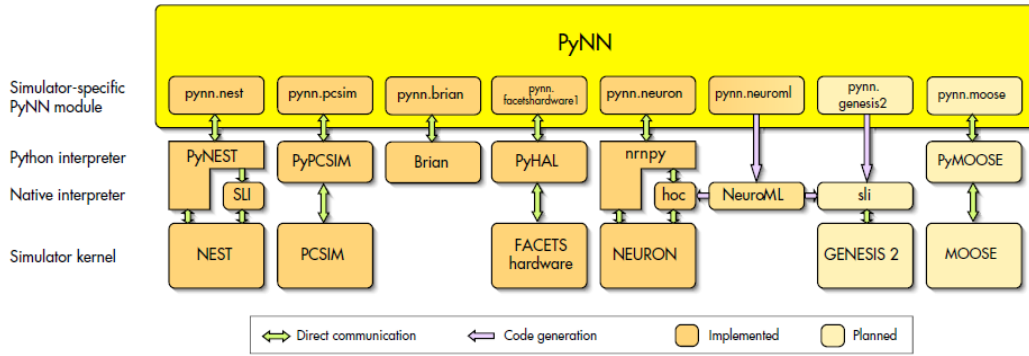


Figure 3.5: PyNN Architecture, from Davison et al. [2008].

### 3.2.7 PyNN

The declared object of PyNN is to “write the code for a model once, run it on any supported simulator without modification” [Davison et al., 2008]. It is a standard description language written in Python to model networks of point spiking neurons. If compared with the specification languages described above, PyNN is able to run network models with any supported simulator, from those mentioned in the previous chapter, and in neuromorphic hardware [Brüderle et al., 2009]. Figure 3.5 shows the simulators supported by PyNN; the neural kernels are separated from the user-language interface through the creation of simulator-specific PyNN modules and their Python bindings.

PyNN offers, natively, a standard description language and the interface to several back-end simulators, making it an ideal choice when running and comparing models with different simulators and platforms or that need to be shared with groups using different architectures.

It comprises a standardised set of neural models (including the conductance and current based LIF models with exponential and alpha currents synapses and the AdEx neuron) and connections (along with several interconnection pattern algorithms) equipped with fast and slow synapse dynamics (plasticity). It also has facilities for distributed computing and handling of pseudo-random numbers.

With PyNN it is straightforward to compare different simulating platforms, as shown in Figure 3.6, as it offers, also, a standard way to save data. The model can be represented in PyNN and run on different simulators with a script such as the one presented here (and in Appendix C.1.1). The simulator name can be passed as a parameter to the Python script, and then can be initialised with the `setup()` call:

```

simulator_name = get_script_args(1)[0]
exec("from pyNN.%s import *" % simulator_name)

p.setup(timestep=1.0,min_delay=1.0,max_delay=10.0)

```

An integrate-and-fire, conductance based neuron (*IF\_cond\_exp*) receives connections from two spike sources. The LIF neuron can be instantiated as a *Population* of 1, and the parameters can be defined in a dictionary as follows:

```

ifcell = p.Population(1, p.IF_cond_exp, cell_params, label='IF_cond_exp')
cell_params = {
    'i_offset' : .1,      'tau_refrac' : 3.0, 'v_rest' : -65.0,
    'v_thresh' : -51.0,   'tau_syn_E'  : 2.0,
    'tau_syn_I' : 5.0,    'v_reset'   : -70.0,
    'e_rev_E'  : 0.,     'e_rev_I'   : -80.}

```

Two *SpikeSourceArray* populations are created using a predefined list of spike times as a parameter to control their activity.

```

spike_sourceE = p.Population(1, p.SpikeSourceArray,
    {'spike_times': [[i for i in range(5,105,10)],]}, label='spike_sourceE')

spike_sourceI = p.Population(1, p.SpikeSourceArray,
    {'spike_times': [[i for i in range(155,255,10)],]}, label='spike_sourceI')

```

The spike sources are then connected to the LIF neuron using the *Projection* method:

```

connE = p.Projection(spike_sourceE, ifcell,
    p.OneToOneConnector(weights=0.006, delays=2), target='excitatory')
connI = p.Projection(spike_sourceI, ifcell,
    p.OneToOneConnector(weights=0.02, delays=4), target='inhibitory')

```

Finally recording methods for the neurons in the model are set, and the simulation is executed for 200 ms:

```

ifcell.record_v()
ifcell.record_gsyn()
ifcell.record()

spike_sourceE.record()
spike_sourceI.record()

p.run(200.0)

```

Results obtained running the model with different simulators are shown in Figure 3.6.

Some standardised data analysis and plotting tools are offered by NeuroTools [\[Yger](#)

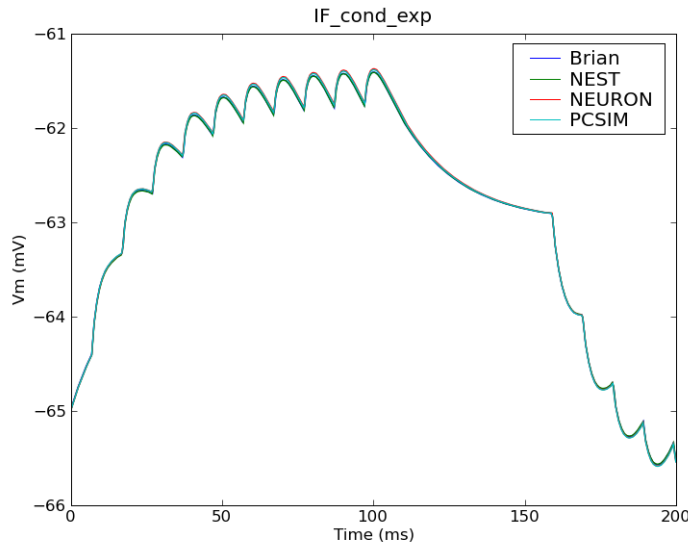


Figure 3.6: An example of using PyNN, adapted from <http://neuralensemble.org/trac/PyNN/>.

et al., 2009], in an effort to unify and promote code sharing for data analysis as well. For all these reasons PyNN has rapidly acquired popularity in the neuromorphic community as it offers a way to check and validate neuromorphic platforms, and their precision and operative ranges, with standard neural languages [Brüderle et al., 2011, Galluppi et al., 2010].

### 3.3 Simulation on dedicated platforms

While simulation in software makes flexible and exploratory modelling accessible to a wide community, larger models require more computational resources. Simulating large number of nodes (neurons) and interconnections (synapses) on standard computers is a very challenging task, both for computation and for communication. To overcome such difficulties, alternative hardware architectures and approaches to spiking network simulation have been proposed; each tries to exploit the intrinsic parallelism of neural networks directly in hardware. Efficiently mapping a neural network onto a hardware substrate is a nontrivial task, due to the massive parallelism and communication bandwidth required by the high connectivity. Scaling on supercomputers has proven to be challenging [Morrison et al., 2005]: two of the biggest and more biologically inspired models of the thalamocortical system

[Markram, 2006, Ananthanarayanan et al., 2009] run on an IBM Blue Gene [Gara and Moreira, 2011]. However such a methodology is not optimal, and IBM itself, through the SyNAPSE initiative, is investigating alternative approaches such as the creation of a *neuromorphic core* [Merolla et al., 2011], a very power efficient digital neuromorphic chip. Alternative approaches include using more ready-available hardware such as FPGAs [Cassidy et al., 2011, Moore et al., 2012] or GPUs [Fidjeland and Shanahan, 2010]. These solutions try to exploit the inherent parallelism and communication characteristics of neural networks in order to explore non-classical computing architectures and computational methods. The rest of the section analyses these different solutions.

### 3.3.1 Supercomputers

The IBM Blue Gene has been used in brain modelling at different levels of abstraction, from the ion-channel level to point neurons. The Blue Brain Project [Markram, 2006] at EPFL, for instance, uses it for simulating plastic neural networks on 8,000 CPUs. The project has produced a detailed model of a 10,000 neuron cortical column based on a 14 day old rat somatosensory non-barrel cortex. This network simulates neurons with a precise biological fidelity down to the ion-channel level, and its simulation language is based on NEURON. The model is continually refined and released to the community every 6 months.

Ananthanarayanan et al. [2009], within the IBM/DARPA SyNAPSE Program, have taken a simpler approach to simulate only point neurons by building a massive cortical simulator called C2. They have simulated a large thalamo-cortical system on a Dawn Blue Gene/P supercomputer with 147,456 CPUs and 144 TB of memory. Their largest simulation is a system with 1.6 billion neurons and 8.87 trillion synapses, corresponding to a cat's brain or 4.5% of the human brain. The total time for simulation is 173 seconds (for 1 second of model time at 3.89 Hz firing rate), with the (MPI based) communication component consuming 71 seconds. From their predictions (reported in Figure 3.7) a 100% scale, real-time human-brain simulation could be achieved in 2018 by a supercomputer with 4 petabytes of memory and running at >1 exaflops. This is predicted by extrapolating the trends for the top 500 supercomputers (purple and green) and their simulation approach (cyan). At June 2013 the twice-yearly TOP500 list of the world's most powerful supercomputers<sup>7</sup> reports the Tianhe-2 (MilkyWay-2) of the National University of Defense Technology in China

---

<sup>7</sup>[www.top500.org](http://www.top500.org)

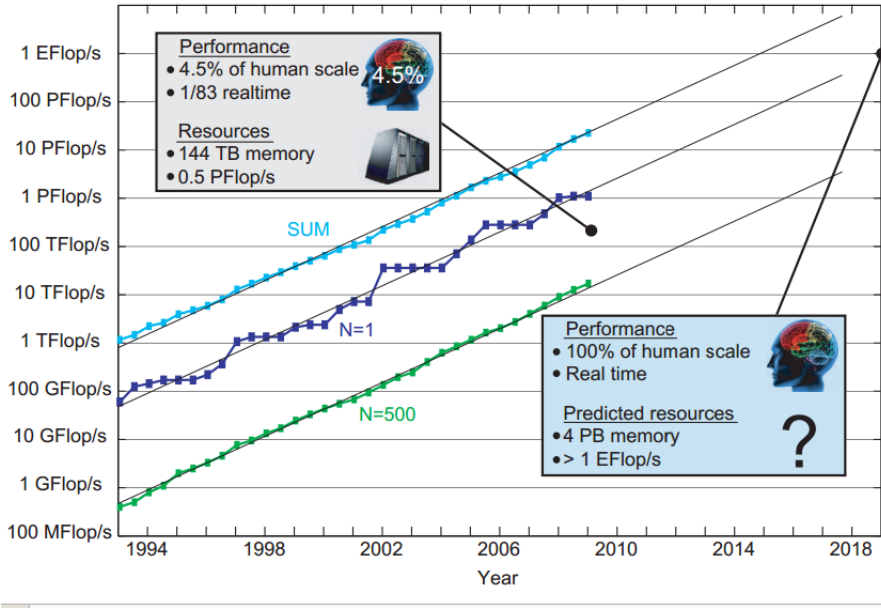


Figure 3.7: Estimation for a full-brain, real-time simulation from [Ananthanarayanan et al. \[2009\]](#).

as the fastest supercomputer, with 3,120,000 processors achieving 33,862 TFlops/s (in line with IBM’s prevision, purple line in Figure 3.7) and consuming 17,808 kW. The SyNAPSE group has recently announced<sup>8</sup> a simulation of  $53 \times 10^{10}$  neurons and  $1.37 \times 10^{14}$  synapses, running  $1542 \times$  slower than real time on the Sequoia IBM BlueGene/Q system installed at DOE’s Lawrence Livermore National Laboratory, a supercomputer ranked as third in the TOP500 list with 17,173 TFlop/s and consuming 7,890 kW.

Both the approaches described run hundreds of times slower than real time. Such a use of supercomputers is generally tightly optimised to the model simulated, and accessibility to such computational resources is not widespread. One attempt to make neural modelling more flexible and accessible has been done with an implementation of NEST for MPI [[Plesser et al., 2007](#)]. Distributing and mapping arbitrary models to a cluster is a non-trivial task, due to the necessity to balance the load accurately through the nodes, which need to be synchronised [[Morrison et al., 2005](#)]. Since (blocking) communication is costly, spikes are buffered and sent at every time step, corresponding to the minimum delay in the model. Further, in general, MPI frames with their size introduce overheads, making meeting the communication constraints

<sup>8</sup>IBM Report:  $10^{14}$ : <http://www.modha.org/blog/SC12/RJ10502.pdf>

in spiking neural networks harder. Finally, the power requirements make the scaling of models challenging.

### 3.3.2 Neuromorphic Hardware

Challenges in simulating large scale models on standard and super computers have pushed some researchers to explore alternative architectures, exploiting the possibility of building fast and efficient neurons in silicon [Mead, 1990, Indiveri et al., 2011] by developing neuromorphic Application Specific Integrated Circuits (ASIC) , constituting Very-Large-Scale Integration systems (VLSI).

Such systems can be used to effect a specific neural model in silicon [Indiveri et al., 2006] but, while such an approach is very power- and compute- efficient [Hynna and Boahen, 2006], it comes at the price of reconfigurability, since the neural model or the connectivity [Choi et al., 2004] is hard-wired. While direct wiring of local neurons is often used, some systems [Merolla et al., 2007, Brüderle et al., 2011] use an FPGA-like lookup to overcome connectivity limitations, combining analogue neural simulation with digital packet-based spike communication to support more general connectivity patterns, albeit at lower density. Another way to enhance reconfigurability of such hardware is to use an FPGA for storing connectivity information in a multi-chip analogue SNN system [Vogelstein et al., 2007].

The DARPA SyNAPSE project itself not only simulates on supercomputers, but to meet power and performance needs has produced a *digital neurosynaptic core* [Merolla et al., 2011]. This neuromorphic chip contains 256 leaky integrate-and-fire neurons arranged in a 16x16 array and implemented in CMOS hardware. Each neuron has 1,024 synapses interconnected with the neurons through a crossbar, making a total of 262,144 synapses per core, consuming 45pJ per spike. The chip is able to match simulations run in software.

After producing a “Spikey” mixed-signal chip [Pfeil et al., 2013] capable of modelling 128,000 synapses, as part of the BrainScaleS project the University of Heidelberg has developed the HICANN (High Input Count Analog Neural Network) chip supporting wafer scale integration for improved connectivity [Schemmel et al., 2010]. Several neural network models implemented on PyNN have been simulated on the former architecture [Brüderle et al., 2011], and they establish effectively a set of test cases to apply to different neural network platforms. Different wafers can communicate with a bandwidth up to 44 billion events/second. A full system with 352 HICANN chips can simulate up to 180,000 adaptive exponential IF neurons and

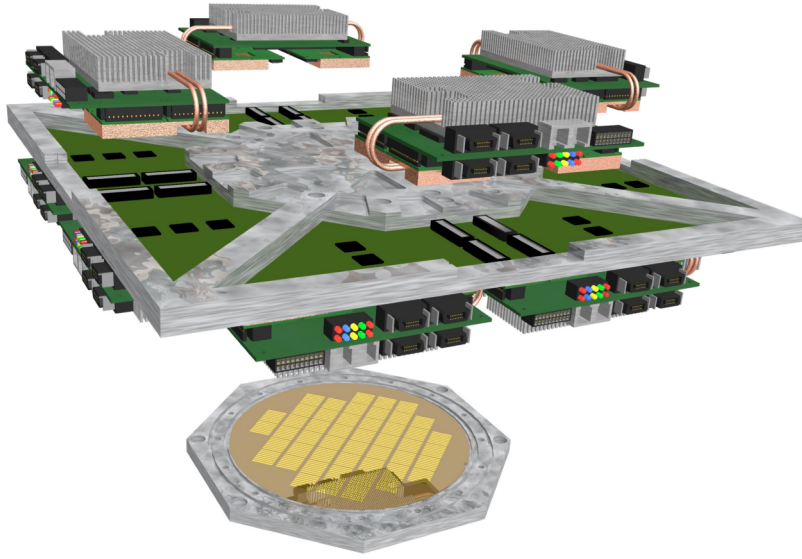


Figure 3.8: Design drawing of the BrainScaleS Neural Network Hardware Module from <http://brainscales.kip.uni-heidelberg.de/>.

40 million 4-bit synapses, equipped with a digital circuit for STDP, running 10,000 faster than real time. A concept design of the system is shown in Figure 3.8.

The Brain in Silicon Project at Stanford University has built Neurogrid [Boahen, 2006, Gao et al., 2012], a platform based on programmable analog chips. Each chip can model 65,000 two-compartment neurons. Neurogrid uses an AER-based packet switched network based on a binary tree connecting 16 chips, scaling up to a system with 1 million neurons running in real time at 1W. With the exception of a subset of connections that can be arbitrarily routed through an FPGA, interconnectivity patterns are generally local between connected chips; a space decay constant (diffusion term) can be used in order to adjust the propagation radius of a spike and enable it to reach more surrounding neurons [Lin et al., 2006].

### 3.3.3 Field-Programmable Gate Arrays

Developing a custom chip is a time and resource consuming research effort. Field-Programmable Gate Arrays (FPGA) can be used as exploratory standalone tools, as their reconfigurability addresses the low flexibility of neural models cast into silicon - a process which is more power efficient and embeddable, but also more costly and time consuming if compared to mixed hardware/software codesign [Reyneri, 2003].

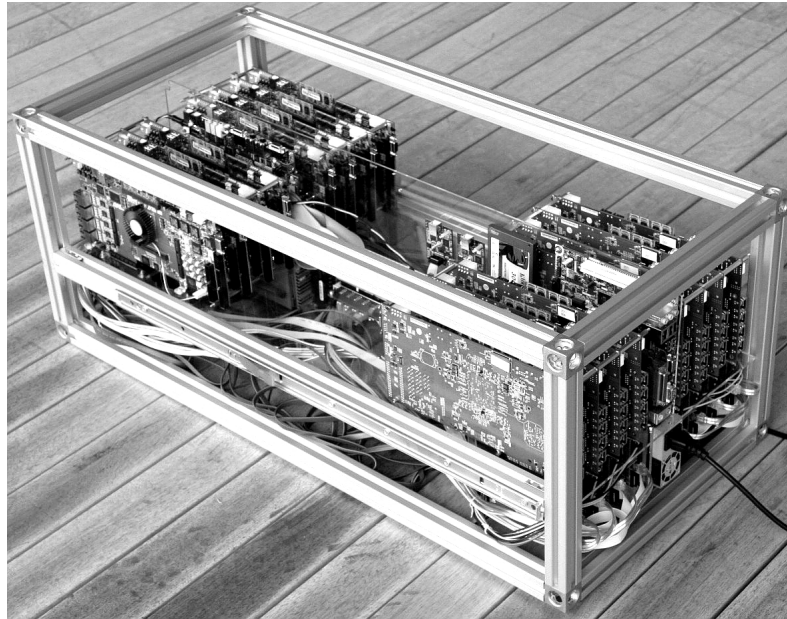


Figure 3.9: A 16-FPGA Bluehive system, from [Moore et al. \[2012\]](#).

[Cassidy et al. \[2011\]](#) have presented a 1 million neuron FPGA system oriented to real-time scene analysis. The system is equipped with 2 GSI Technologies 36Mb QDR SRAMs that hold the mapping and the synaptic weights and effectively limits the interconnectivity in the system. FPGAs have proven to be good exploration tools; however mapping very high connection densities in FPGAs is challenging due to the circuit-switched architecture, and cost and power consumption make scalability difficult [[Maguire et al., 2007](#)].

As an alternative approach the University of Cambridge has recently proposed Bluehive [[Moore et al., 2012](#)], a system of 64 FPGAs each capable of simulating 64,000 neurons and 64 million synapses. The system is designed to deliver high-bandwidth and low latency communication; this is obtained through the use of 12 SATA3 bidirectional links with a bandwidth of 6 Gb/s each - sufficient to guarantee latencies « 1 ms with a lightly loaded network for FPGA-to-FPGA communication. The system supports models of Izhikevich neurons built with arbitrary topologies. As communication with memory limits the number of neurons that can be modelled in real-time on the platform, each FPGA is equipped with 2 DDR2 slots which can hold up to 8 GiB of off-chip memory with high-access bandwidth peaking at 12.8 GB/s. Such a memory system gives the advantage that the platform can be reconfigured with a different network description without resynthesizing the FPGA. An example of a 16 FPGA Bluehive system is presented in Figure 3.9

### 3.3.4 Graphics Processing Units

Graphical Processing Units (GPUs) are mass-market devices comprising hundreds of cores with high memory-access bandwidth; with their inherent parallelism they offer a reasonable alternative for medium-scale neural modelling. GPUs are frequently used to increase performance in parallel programming; speed-ups from tens to thousands times over standard CPUs have been reported; however some of these studies lack in methodology and fail to replicate such results on more rigorous tests with benchmark kernels [Lee et al., 2010], only achieving a 2.5x speedup. Another major limiting factor is power, which affects scalability: a typical GPU consumes  $\sim 200$  W/chip, even more than an FPGA [Tse et al., 2009].

Programming a GPU is a non trivial task. The NeMo platform [Fidjeland et al., 2009] is a GPU-based system able to run large-scale models up to 40,000 Izhikevich neurons delivering 400 million spikes per second in real-time on an nVidia Tesla C1060 GPU, consuming 187 W (6 mW/neuron or alternatively 467 nW/spike). The system can arbitrarily be configured to different topologies; Bhuiyan et al. [2010] present a study of performance improvements using GPUs to run spiking neural networks of Izhikevich and Hodgkin-Huxley neurons. They report their limiting factor for the simulation as the access to memory, as GPUs are primarily suited to high computation-to-communication-ratio models. However as reported in Moore et al. [2012], neural network simulation is communication bound. In fact the maximum available bandwidth for GPUs can be accessed only using ideal memory access patterns. Programming a GPU is a very specialised task, as the architecture needs to be taken carefully into consideration to use it efficiently. Mapping arbitrary models to a GPU is then a non-trivial task, due to the requirement of having coalescent memory data structures to minimize memory access overheads [Brette and Goodman, 2012], and it is therefore a niche for experts. In order to open GPU-based modelling to a wider community the NeMo project has proposed a method to configure and run arbitrary models on GPUs by interfacing them with a PyNN-inspired C++ interface [Fidjeland et al., 2012], making it accessible to non hardware experts.

### 3.3.5 Address Event Representation

Address Event Representation (AER - Mahowald [1992]) is a spike-inspired real-time communication mechanism, where *time models itself*, as the information is encoded in the timing of occurrence of the event. Following this abstraction different sensing,

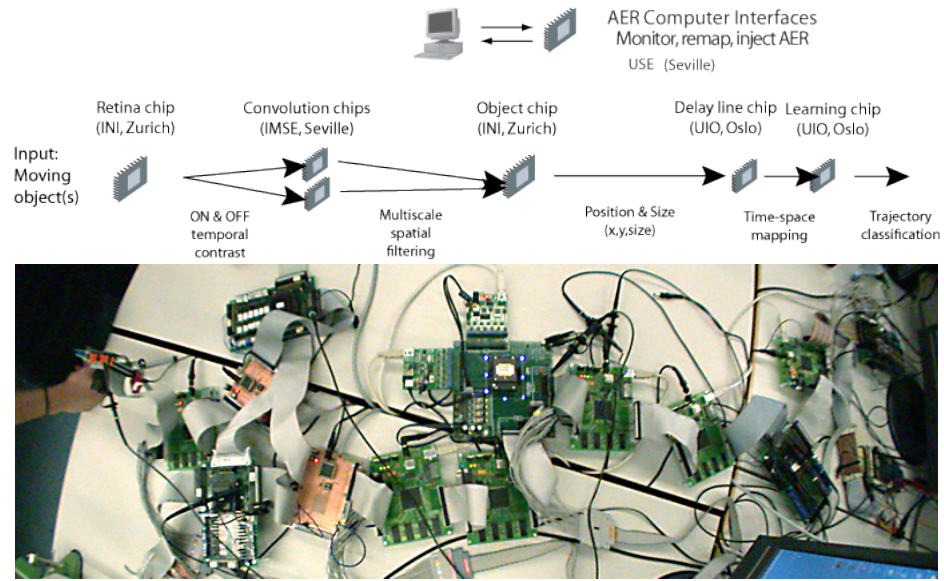


Figure 3.10: A working CAVIAR prototype assembled in April 2005 at the CAVIAR workshop and its schematics, from the project resources website <http://www.ini.uzh.ch/~tobi/caviar/>.

processing and actuating spike-based custom systems have been developed.

The use of AER as a standard communication protocol enables such systems to be interconnected, as done in the EU funded project *Context Aware Vision using Image-based Active Recognition* (CAVIAR) project - [Serrano-Gotarredona et al., 2009]. The project shows the advantages of using event based representation rather than conventional image (frame) based approaches in terms of performance and power consumption by building a visual processing system composed of four custom AER chips and five AER interfaces (see Figure 3.10).

The sensory system comprises a neuromorphic silicon retina [Lichtsteiner et al., 2008], an asynchronous vision sensor which emits events signalling the location of contrast changes. The sensor has a very low latency (down to  $15 \mu s$ ) in emitting an event and a wide dynamic range of operation due to the logarithmic representation of the intensity.

To support the silicon retina, other custom mixed-signals chips were produced. A programmable kernel 2-D convolution processing chip was used to detect circular objects of various size by convolving the events with a circular filter. Such convolutional chips [Serrano-Gotarredona et al., 2006] are fast feature extractors which work on the principle of convolutional networks [LeCun and Bengio, 1995]. The event flow produced by the silicon retina is convolved with a kernel representing a

spatial feature and connected to a feature map. Every neuron in the feature map is connected to a subgroup of units of the retina - its *receptive field* - and implements the convolution kernel in the interconnection weights. The feature map represents the result of this operation, the matching of the receptive field with the convolution kernel. Different features can be extracted in parallel as was done, for example, with oriented bars in the primary visual cortex [Hubel and Wiesel, 1962].

Feature and spatial competition is implemented in a 2-D winner-take-all (WTA) object chip composed of integrate-and fire neurons [Oster et al., 2008], which filters the convolutional signal enhancing contrast. The learning of spatio-temporal activity patterns, representing the trajectory of an object, is performed by a chip introducing delay lines to represent different time instants in space and by a learning chip capable of Hebbian Learning. Finally a tracking system, controlled by the WTA module, keeps the object in the field of view. Overall the system comprises 45k spiking neurons and 5M synapses, is capable of fast recognition and tracking, and in particular is able to discriminate two distinct shapes rotating on a disk, whose positions can be extracted at level of the object chip. Such a project presents many of the characteristics that show why the neuromorphic approach is appealing for fast, asynchronous response time and its low power consumption. The developed ASICs and interfaces are presented in Figure 3.10.

### 3.4 SpiNNaker

This section presents SpiNNaker (a contraction of “Spiking Neural Network Architecture”), a novel architecture developed by the Advanced Processor Technologies Group at the University of Manchester [Furber et al., 2006b]. This offers a different sets of trade-offs if compared to standard computers, VLSI neuromorphic chips, FPGAs and GPUs. It proposes a specialised architecture for simulating networks of spiking neurons while maintaining flexibility in neural model and network topology specification.

SpiNNaker is a digital multi-core multi-chip architecture, comprising a mixture of off-the-shelf, programmable digital components (ARM968 cores) and a bespoke reconfigurable packet switched network infrastructure. Each SpiNNaker chip contains 18 identical ARM968 processors designed to model various types of neural and synaptic dynamics, up to  $\sim 1000$  neurons per core [Furber and Temple, 2008], each receiving inputs from 1000 neurons firing at 10Hz. Chips are interconnected using 6

asynchronous links in a toroidal mesh that forms the full version of the system, scalable to 60,000 chips for a total of more than a million ARM cores. Spikes propagate in the system, using a multicast routing mechanism, through a packet-switched link fabric. Synaptic information is kept in an Synchronous Dynamic Random-Access Memory (SDRAM) chip which can also be used to store simulation results.

The use of many RISC-based processors brings the advantages of reconfigurability and power consumption (1W for a single SpiNNaker chip equipped with 18 ARM cores), while the network infrastructure is designed to overcome the limitations in communication encountered when scaling up architectures mentioned in the previous chapter. The strongest points of the SpiNNaker architecture are the low-power consumption (as energy - not the number of cores - is the main cost in running large-scale models), the network infrastructure (designed for efficiently multicasting small packets through the system), and its programmability (both in terms of neural dynamics and network topology). In this sense SpiNNaker hints at supercomputer approaches for their programmability, but offers a much low-power, real-time oriented system; while not reaching the performance and power consumption typical of custom analog neuromorphic hardware, it offers greater reconfigurability and scalability over such systems.

Exposing such reconfigurability comes at a cost: there must be a process which maps a neural network model to the system, configuring the platform. Such a process needs to be capable of efficiently distributing the load across a SpiNNaker machine, while also exposing its reconfigurability. Information about the model is distributed across different components and different chips of the SpiNNaker system. The network model needs to be translated into SpiNNaker data structures that will then be used to configure different parts, such as the Tightly-coupled memory (TCM) of the ARM cores, the SDRAM and the Multicast Router content-addressable memory (CAM) of each chip. In the next paragraphs the architecture will be analysed more in depth, with special regards to the ways to configure it.

### 3.4.1 Architecture

The SpiNNaker System is a programmable, asynchronous, massively-parallel multi-core system oriented to the simulation of heterogeneous, large-scale, models of spiking neural networks [Furber et al., 2006b]. Each SpiNNaker chip contains 18 ARM968 cores embedded in a programmable, packet based, network on chip [Plana et al.,

2007], offering a novel combination of programmability, low-power and communication performance. Each SpiNNaker chip contains 100 million transistors in a  $102\text{ mm}^2$  die, stitch-bonded together with a 128 MByte SDRAM chip in a single BGA package. Action potentials are encoded as source-based AER [Lazzaro et al., 1993, Mahowald, 1992] packets, and transmitted through a Multicast (MC) Router capable of handling one packet per clock cycle if unimpeded. Every core has a local Tightly-Coupled Memory (TCM - 32KByte for instructions and 64KByte for data), while each core has access through a dedicated DMA controller to a 1 Gbit SDRAM shared by the 18 ARM cores within a single chip; this has an aggregate memory bandwidth of 5.6 Gbit/s. SDRAM is partitioned into regions containing core-specific synaptic information, eliminating the issue of memory sharing across the system. The memory system is thus locally to every chip, circumventing the challenges needed on, e.g. GPUs to access memory [Bhuiyan et al., 2010] and maintain process coherency [Nageswaran et al., 2007], while keeping power consumption lower than on such systems. This makes every node independent, as the execution needs only local available data, while communication with other nodes is completely asynchronous. A chip diagram is presented in Figure 3.11.

Each chip can be connected to 6 adjacent neighbours in a toroidal mesh using bi-directional asynchronous links yielding an aggregate spiking bandwidth of 1.5 Gbit/s [Patterson et al., 2012b], supporting reconfigurable arbitrary connectivity [Furber et al., 2006a]. An example 4x4 system without wrap-around is presented in Figure 3.12: each SpiNNaker chip (numbered in red) is connected to its neighbours through 6 bidirectional asynchronous links (numbered in black).

Vainbrand and Ginosar [2011] have analysed different interconnection strategies for spike-based communication, and conclude that a MC mesh Network-on-Chip is the most suitable interconnect architecture for reconfigurable neural network implementations. The custom packet-switching network [Plana et al., 2007] is easily reconfigurable by changing the MC routing tables and more wire-efficient than a circuit-switched architecture [Maguire et al., 2007, Cassidy et al., 2011].

The SpiNNaker System offers an alternative set of trade-offs to neuromorphic chips [Hynna and Boahen, 2006] and programmable, high-performance computing systems [Ananthanarayanan et al., 2009]. While being outperformed in absolute power consumption [Indiveri et al., 2006, Merolla et al., 2007] or speed [Schemmel et al., 2010] by dedicated analog hardware, and in representational flexibility

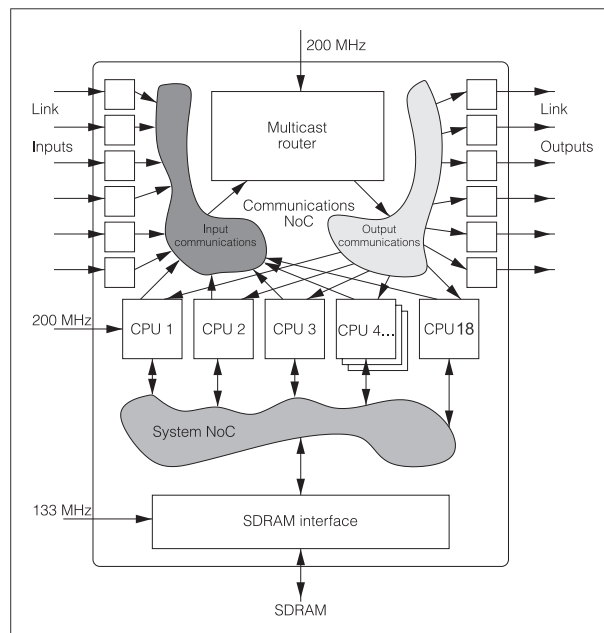


Figure 3.11: SpiNNaker chip diagram, after [Plana et al., 2007]. Each SpiNNaker chip contains 18 ARM968 cores embedded in a programmable, packet based, network on chip. Action potentials are encoded as source-based AER packets, and transmitted through a Multicast Router capable of handling one packet per clock cycle if unimpeded. Every core has a local Tightly-Coupled Memory and access through the System NoC to a dedicated DMA controller, used to access a 1 Gbit SDRAM within every chip.

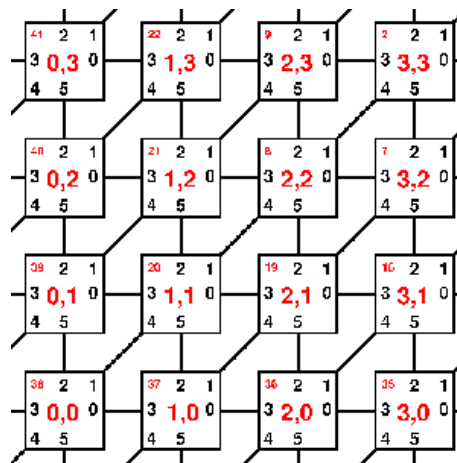


Figure 3.12: Inter-chip connectivity for a 4x4 system without wrap-around. Each SpiNNaker chip (numbered in red) is connected to its neighbours through 6 bidirectional asynchronous links (numbered in black).

by general-purpose computers, SpiNNaker offers a scalable and completely reconfigurable platform for exploration of a wide range of network models, within a low power budget of 1 W/Chip [Furber and Brown, 2009]. The full system is designed to contain up to 65,536 chips and more than a million cores.

### 3.4.2 Hardware Reconfigurability

From a computational and communication point of view each ARM core offers flexibility in the complexity and number of neurons modelled and the way they are integrated in a configurable network based on the Multicast Router. The number of neurons which can be modelled on a single core depends on factors including the activity of the neurons, the computational power needed to solve the equations describing the neural dynamics in real time, the number of synapses and the memory occupancy. The software that allocates neurons on board must facilitate exploring such trade-offs to experiment with different configurations. As introduced in the previous section, the SpiNNaker system is an architecture with multiple dimensions of reconfigurability, in particular:

- **Application reconfigurability:** individual cores in the system can be configured to run different applications, for instance different neural, synaptic or plasticity kernels, or non-neural applications such as data collection and on-board analysis. Each application needs to be configured with its parameters (e.g. governing the neural equations). Seamless integration of new applications into the existing framework is critical to have them as available resources in system deployment.
- **Connection reconfigurability:** the MC router system permits arbitrary network topologies to be mapped through a 2-stage routing and lookup system which is able to route a spike efficiently to many different destination neurons placed anywhere in the system. Spikes are encoded as source-based AER events: The MC routers manage the first routing stage through the leading field of the routing key (processor coordinates in the system); the lookup phase deals with the second field of the routing key (neuron id within a processor). The field definitions themselves are not fixed but programmable through mask bits in the router.

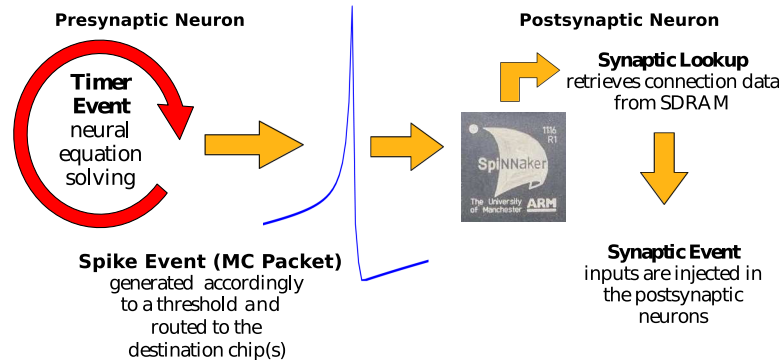


Figure 3.13: Neural Simulation Events. Neural equations are solved during a *Neural Event* and a *Spike Event* is produced if a particular condition is met. Spikes are encoded as source-based AER packets and are routed to their destination cores by the MC routers. On the post-synaptic chip synapses are retrieved locally using a source-based lookup to access SDRAM; synaptic inputs are then injected into the target neurons (*Synaptic Event*).

### 3.4.3 Neural Applications

While the SpiNNaker system is a general purpose scalable parallel machine, its architecture is designed for the simulation of large networks of spiking neurons; these can be modelled efficiently by representing spikes as stereotypical events encoded in an MC packet. Neural applications (or kernels) are based on a set of APIs [Sharp et al., 2011] written in C which expose the event-driven nature of the system. From a user perspective 3 events can be distinguished (Figure 3.13):

- **Timer Event (Neural Event):** each core contains a programmable timer generating interrupts at configurable time steps, for example at every millisecond. At each time step, for each neuron, the equations are solved and spikes generated as required.
- **Spike Event (MC packet):** a spike is produced if during the neural evaluation a condition is met (e.g. the membrane potential crosses a threshold).
- **DMA Done (Synaptic Event):** once a spike arrives at its destinations, DMA retrieves the relative synaptic information from SDRAM and the input is injected into the post-synaptic neurons.

Figure 3.13 shows how events interact: at each time step a *Timer Event* is produced, locally to the chip. Neural equations are solved (according to the neural application, parameters and state variables) and a *Spike Event* is produced if a condition is

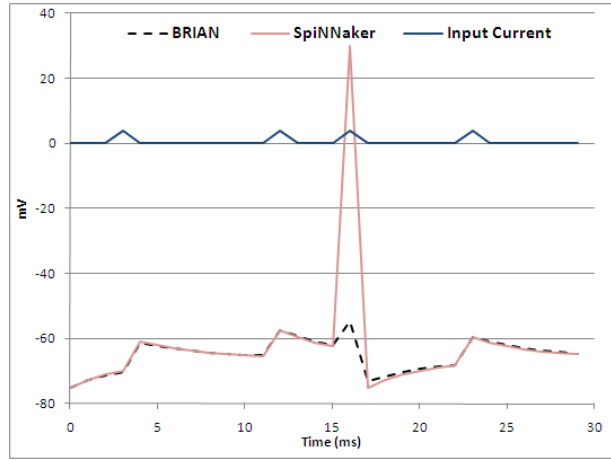


Figure 3.14: Single neuron dynamics. The neuron is injected with 4 pulses of current.

met (e.g. the membrane potential crosses a threshold). Spikes are encoded as source-based AER packets and are routed to their destination cores by the MC routers. On the post-synaptic chip synapses are retrieved locally using a source-based lookup to access SDRAM; synaptic inputs are then injected into the target neurons (*Synaptic Event*) according to the synaptic model used [Jin et al., 2010]. Sharp et al. [2011] describe how to build a neural application on top of the API that abstracts the SpiNNaker hardware layer. Some other aspects need to be taken into consideration however when building an application kernel for SpiNNaker: most notably the absence of a Floating Point Unit, support for complex mathematical operations on the ARM968 cores and their limited local memory. Figure 3.14 from Rast et al. [2010a] shows the simulation of a Leaky-Integrate-and-Fire neuron compared to the Brian simulator (more results can be found in Appendix B).

Neural applications use run-time, model-dependent data to configure neural dynamics and connectivity. These need to be compiled for the system, translating the model (neurons and synapses with associated parameters) into SpiNNaker data structures. Such data structures are presented in Figure 3.15 and include the individual neural parameters (such as the threshold potential for the LIF neuron), the routing tables to route spikes from source to destinations, the lookup table to retrieve synaptic data from memory and the synaptic data itself (weights, delays, type of synapse, etc.). To load a model on the system the model must first be mapped, allocating model resources (neurons, synapses) to system resources (TCM, SDRAM, MC Router CAM); the data structure representing the instantiation of the model on a physical system must then be compiled and loaded into the machine.

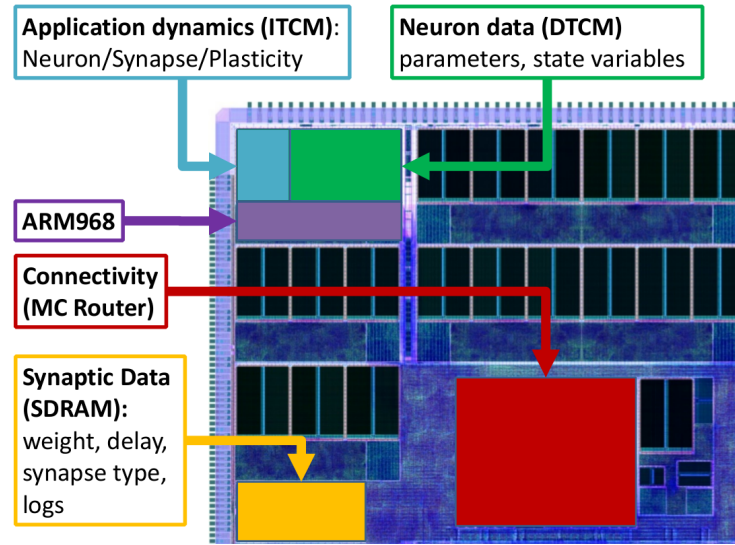


Figure 3.15: Functional blocks on a section of a SpiNNaker Chip.

Since SpiNNaker cores can be configured as desired any synaptic dynamics can be implemented, within the constraints of the simulation speed (more complex synapses will take more instructions to process, hence lowering the number of neurons per core or decreasing the simulation speed). To reconfigure and expand a system of thousands of chip and up to about a million cores rapidly, data structure creation needs to be as flexible and efficient as possible. The next section presents the approach proposed to solve this problem.

### 3.5 Summary

This chapter reviewed the principal approaches to spiking neural simulation, by introducing different software approaches and presenting hardware solutions to run fast, large-scale models. The richness of instruments and tools replicates the variability of modelling approaches which was tentatively captured in the previous chapter.

To overcome memory-access and bandwidth-limitation problems, alternative architectures for neural simulation have been proposed. However such approaches are rarely interoperable, making model sharing hard within the community and negatively impacting the development of the field [Crook et al., 2012, Nordlie et al., 2009]. With alternative architectures the problem becomes more evident, as hardware solutions typically have their own fixed architecture which does not support arbitrary interconnectivity [Choi et al., 2004] or neural models [Millner et al., 2010], rarely

supports general description languages and does not have the reconfigurability of standard computers. However running simulations on supercomputers has its own challenges in communication [[Morrison et al., 2005](#)] and power consumption. While this vast choice of different, optimised approaches is beneficial for researchers, it also constitutes a hurdle to greater collaboration between research groups. Therefore some standard ways to describe networks have been identified; such languages are a step forward towards enabling scientific collaboration.

In this context this chapter has introduced the SpiNNaker system; a novel configurable digital platform that supports the simulation of large real-time neural network models with an alternative set of constraints to standard or neuromorphic architectures. The central contribution of this thesis can be found in the next chapter, which presents an approach to represent models on a digital distributed platform in order to expose its reconfigurability and ability to meet the standards emerging in the spiking networks community; such an approach will constitute the basis for running on SpiNNaker all the models presented in this thesis.

# Chapter 4

## Configuring a universal neural system

### 4.1 Introduction

The previous chapter analysed the proliferation of different tools and approaches for spiking neural network simulation from a software view and discussed alternative architectural approaches. It also presented the SpiNNaker platform with its alternative set of trade-offs in power consumption, scalability and reconfigurability if compared to neural simulations running on off-the-shelf computing platforms or on ASICs.

SpiNNaker is a digital, programmable platform and the model features, such as neural model kernel, network topology, neural parameters and synaptic informations, are all specified in software. While the system architecture offers a low-power, scalable, fully configurable platform for modelling neural networks, its innate parallelism and distribution of information raises challenges regarding application mapping, loading and running [[Khan et al., 2010](#), [Rast et al., 2010b](#)].

This chapter addresses these challenges by describing an open system to map models onto the SpiNNaker platform. The approach proposed defines the entities involved in translating a model into compiled data structures to be distributed on a parallel system, and the algorithms used in the process. The approach separates the model from the system through an intermediate, generic representation; it also separates the data (entities and different representations) from the algorithms performing the translation from the user interface.

As a result, the reconfigurability of SpiNNaker is exposed in a user-friendly hardware abstraction of the platform through interfaces with domain-specific neural languages, described in the next chapter. The flexibility of the proposed translation method is the key feature that enables all the experimental contributions presented in this work.

## 4.2 Model to Hardware

So far, how the biological observations of nervous cells can be modelled mathematically has been examined, and the different approaches to software and hardware infrastructures that can be introduced to facilitate simulations by offering high-level conceptual entities. The SpiNNaker system and its architecture has been presented, along with neural kernels that need to be configured to run simulations. What is missing is a way to transform a conceptual network model into a simulation running on parallel hardware. Further, in general, the problem can be seen as a way to encode a model, map it and allocate it on a physical, parallel system. Conceptual entities are encoded into neural kernels and need to be configured to be allocated; once they are allocated, they consume a specific resource on the system such as the TCM of a core, SDRAM or connectivity resources; these can be modelled as conceptual entities as well: the system produces resources that are consumed by model entities.

In this context an approach to decouple the model specification stage and the generation of data structures representing the model on the SpiNNaker platform is presented. This approach can be used to configure different types of computational nodes (neurons) connected in an arbitrary way and map them to a dedicated architecture. The complexity of the problem becomes rapidly intractable if allocation of single neurons and synapses on the system is attempted, as done by [Jin, Galluppi, Patterson, Rast, Davies, Temple, and Furber \[2010\]](#). The input to the system itself is a list of neural parameters and connections (as shown in Figure 4.1), making even slight modifications to the model onerous, as explicit modifications to the list are required. The system topology and the translation methods are embedded in a *monolithic* software, which uses neurons and synapses as entities for placing and routing, making scalability difficult.

Instead, by organising neurons into homogeneous populations, the system maintains hierarchical information about the model, using it to drive the allocation on the system in a simpler way than by considering single neurons. To demonstrate

```

neural list:
ID 0: IZK -80, 0, 0.02, 0.20, -65, 8, 0
ID 1: IZK -80, 0, 0.02, 0.20, -65, 8, 0
ID 2: IZK -80, 0, 0.02, 0.20, -65, 8, 0
ID 3: IZK -80, 0, 0.02, 0.20, -65, 8, 0
ID 4: IZK -80, 0, 0.02, 0.20, -65, 8, 0
ID 5: IZK -80, 0, 0.02, 0.20, -65, 8, 0
ID 6: IZK -80, 0, 0.02, 0.20, -65, 8, 0
ID 7: IZK -80, 0, 0.02, 0.20, -65, 8, 0
ID 8: IZK -80, 0, 0.02, 0.20, -65, 8, 0
ID 9: IZK -80, 0, 0.02, 0.20, -65, 8, 0
ID 10: IZK -80, 0, 0.02, 0.20, -65, 8, 0
...
...

connection list:
ID 0
: 1, 20.00, 4
: 2, 20.00, 3
: 6, 20.00, 1
: 7, 20.00, 1
: 10, 20.00, 6
ID 1
: 11, 8.11, 1
: 7, 10.00, 1
: 8, 20.00, 7
ID 2
: 12, 6.55, 1
: 3, 22.00, 8
: 4, 1.00, 4
: 5, 23.00, 8
...
...
```

Figure 4.1: Example neural and connection input lists. Explicit modifications are required for any change in the network model.

the proposed approach an implementation of the abstraction layer, which translates a model from different front-ends and allocates it on the SpiNNaker machine, is presented in the following sections. Results, obtained using this configuration approach, are in the following sections. Finally, the chapter ends by discussing the challenges and potential of such an approach and its overall implications.

### 4.2.1 The mapping approach

By introducing a translation level (*map* or system level) from one to the other layer, the approach effectively decouples the device from the neural network model level. This layer needs to reflect the availability of a variety of models (neural, synaptic, plasticity), their allocation on the system, and the configuration of the connections between neurons, in order to transform the high-level representation into an on-chip representation. It also needs to be able to represent the resources present in the system, from both a software and hardware point of view. This layer then becomes central to any model-to-machine interaction: at boot time for configuring the system, at runtime for real-time interaction and for data analysis. Therefore it needs to be efficient, as the potential of a real-time configurable device is wasted if the time or complexity to configure and interact with it becomes unmanageable. It also needs to scale up to configure networks with millions of neurons and billions of connections. Finally, it must be compatible with different programming languages or modelling approaches to leverage the hardware reconfigurability fully.

We represent information at the *Population* (group of neurons) and *Projection* (bundle of single connections between Populations) level, as most of the languages considered use this representation natively [Davison et al., 2008, Stewart et al., 2009, Gewaltig and Diesmann, 2007]. This greatly simplifies the allocation, mapping and generation processes as against flattening out the whole network at a neuron level, using hierarchical information to effect the translation from the model and place it on the device. Atomic reconfigurability (single neurons and synapses) is maintained, but is pushed to the data file generation step in a way which can easily be ported on-chip and parallelised. The 2-stage routing/lookup system lets the router deal only with Populations, greatly saving routing entries and lowering the complexity through interval routing [Van Leeuwen, 1987] (where a routing interval can be considered as either the set of routing keys in a Population or in a core, depending on the granularity of the mapping algorithm). This greatly simplifies the routing problem, as illustrated in Davies et al. [2012b], and reconfigurability both for arbitrary connectivity between Populations and for easy integration of new cell types constituting Populations. To maximise system scalability, all the data of this layer is represented in SQL format, taking advantage of a “language agnostic” representation which can easily be accessed from different languages with high performance database technologies. The following sections explain the structure and function of this layer by presenting an implementation which is able to translate models from different front-ends and drive data structure generation.

### 4.2.2 Partitioning and Configuration Manager

The function of PACMAN - the PArtitioning and Configuration MANager, is to transform the high-level representation of a neural network into a physical on-chip implementation. PACMAN is based on a database that holds three representations of the neural network (Figure 4.2), which can be thought as directed graphs:

- **Model Level**, the network as specified in the high-level language (such as PyNN or Nengo).
- **System Level**, the network as partitioned into *partPopulations* that can fit into a single computing core, Projections, probes and inputs being split accordingly.
- **Device Level**, a map distributing the model to system components.

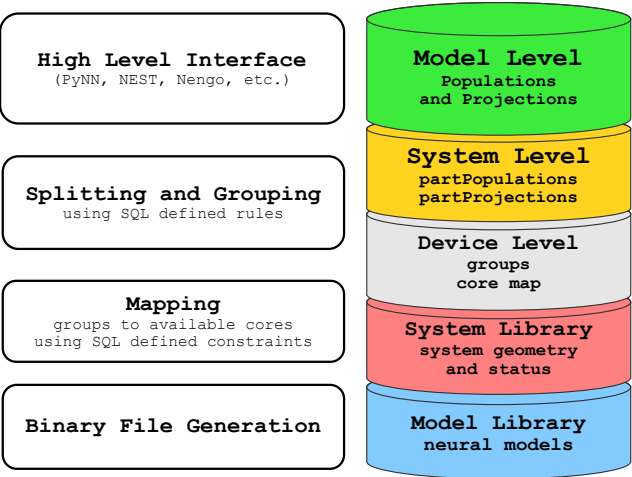


Figure 4.2: PACMAN structure. PACMAN is based on a database that holds three representations of the neural network, and a process that transforms a model written in a domain specific neural language such as PyNN or Nengo into binaries that can be loaded and executed on the system.

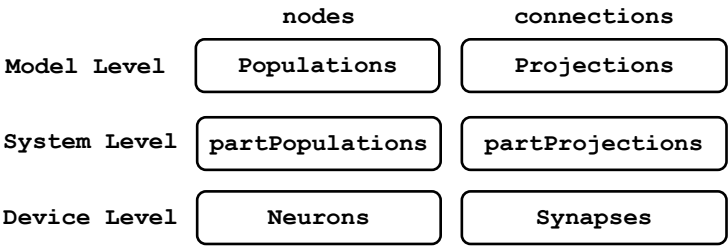


Figure 4.3: Hierarchical Approach. At the Model level the entities considered are Population of neurons and Projections between them; at the system level Populations and Projections are split into their corresponding partPopulations and partProjections, grouping together neurons from a single population that can be modelled on a single ARM core; at the Device level, after the mapping and routing process, neural and synaptic entities are expanded, flattening out the hierarchy.

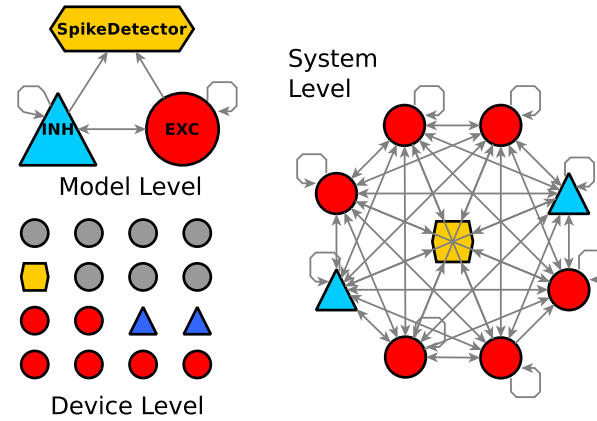


Figure 4.4: Example network composed of 2 Populations and a Spike Detector interconnected

These representations are specific for every network instantiated, while the *Model* and *System Libraries* are imported to represent information about the system resources and geometry and the models available. Such translations enable the network to be mapped and deployed on the SpiNNaker system, by generating the binaries needed to configure the simulation components and topology, implementing the hierarchical approach described in Figure 4.3. PACMAN itself (Figure 4.2) is divided into 4 different steps:

- **Splitting**, responsible for splitting neural *Populations* which will not fit in a single core (because of memory or computational complexity limitations) into *partPopulations* that will fit in a core.
- **Grouping**, responsible for collating *partPopulations* which can be run using the same application code to fit more of them onto a single core. The result of this operation is a *group*, a collection of neurons which may belong to one or more *partPopulations* and that can fit in a single core.
- **Mapping**, responsible for allocating neural groups to processors and calculating routing. At this stage the model is placed onto the device.
- **Binary file generation**, which creates the actual data binaries from the partitioned and mapped network.

Figure 4.4 illustrates an example, showing a common scenario where one excitatory (red circle) and one inhibitory (blue triangle) population of neurons are interconnected [Brette et al., 2007]; a spike detector device is placed to record spike

activity. PACMAN splits Populations into *partPopulations* according to their size (in the example the result is 6 excitatory and 2 inhibitory *partPopulations*), and *Projections* between them into *partProjections*, maintaining connectivity unaltered. Finally, it maps *partPopulations* to physical cores at the Device level.

population	neurons/core	offset	x	y	p	ID range	key range
Excitatory	500	0	0	0	1	0-499	0x0800-0x09F4
Excitatory	500	500	0	0	2	500-999	0x1000-0x11F4
Excitatory	500	1000	0	0	3	1000-1499	0x1800-0x19F4
Excitatory	500	1500	0	0	4	1500-1999	0x2000-0x21F4
Excitatory	500	2000	0	0	5	2000-2499	0x2800-0x29F4
Excitatory	500	2500	0	0	6	2500-2999	0x3000-0x31F4
Inhibitory	500	0	0	0	7	0-499	0x3800-0x39F4
Inhibitory	500	500	0	0	8	500-999	0x4000-0x41F4

Table 4.1: PACMAN Example: an excitatory population of 3000 neurons and an inhibitory population of 1000 neurons are interconnected. The excitatory population is divided into 6 *partPopulations* of 500 neurons each, while the inhibitory population is divided into 2 *partPopulations* of 500 neurons each. The mapping stage assigns every *partPopulation* to a specific core, and assigns to each neuron a unique ID within the core and a unique routing key.

Table 4.1 shows an example, where an excitatory population of 3000 neurons and an inhibitory population of 1000 neurons are interconnected. The excitatory population is divided into 6 *partPopulations* of 500 neurons each, while the inhibitory population is divided into 2 *partPopulations* of 500 neurons each. The mapping stage assigns every *partPopulation* to a specific core, and consequently every neuron to a unique routing key within the routing key space interval assigned to that core, enabling the possibility of using interval routing [Van Leeuwen, 1987]. The advantages of using a hierarchical approach, by using *partPopulations* and *partProjections* as mapping and routing entities, is evident even in a small-scale example such as this. If we suppose a 10% connection probability between (and within) the populations we have a total of  $3000 + 1000 = 4000$  neurons and  $4000 \times 4000 \times .1 = 1,600,000$  synapses. The mapping entities are reduced in number from 4K neurons to 8 *partPopulations*, and the number of entities to be routed from 1.6M synapses to 64 *partProjections* (as every *partPopulation* is connected to each other with a 10% probability). The hierarchical approach introduced in this Chapter makes mapping large-scale problems such as the ones described in the following Chapters possible.

### 4.2.3 Algorithms

Rules for Splitting and Grouping can be defined in SQL format to represent arbitrary constraints, which can be easily incorporated in the framework. For example, splitting can be done on the basis of the maximum number of a certain neural type a core can model in real time, while grouping can be applied to neurons using the same neural/plasticity model and having compatible mapping constraints. This information can be passed to the Mapper stage along with model-specific data provided by the high-level generation tool, giving it all the information needed to generate each portion of the network locally. The Mapping stage can take into account constraints also defined in SQL on a Population basis. This makes interaction with external tools and user-entered constraints simple; if none are specified, groups are assigned to available cores on a first-come first-serve basis. The *System Library* (Figure 4.2) represents available resources of the system, holding a representation of the system size, geometry and functional status so as to map around malfunctioning resources. This process can also differentiate between multiple users accessing separate parts of the machine independently. When users request resources, PACMAN will grant access only to chips allocated to them.

Output from the Mapper is a hierarchical physical description of the entire network. At this point the description still contains abstract objects rather than single neurons or synapses. The mapper organises this information in a way such that binary file generation can be easily parallelised, evaluating the table produced by the mapper core by core. This is in turn passed to an Object File generator (which for the moment resides on the Host but could eventually be migrated to an on-SpiNNaker implementation) which flattens the network and generates the actual data binaries for the system. The neural and connectivity structures are computed by retrieving the translation, size and position of the parameters in the neural structure from the Model Library. Parameters can be defined as single values, random distributions or lists explicitly defining the parameter values for each neuron or synapse.

### 4.2.4 Data Representation

The entities involved in the three representations are shown in the entity-relationship diagram in Figure 4.5. Particularly important entities and relationships are:

- **Populations** of neurons and **Projections** between them, as described in the

model, are the fundamental entities at the *Model Level* representation. Available translation methods for neurons and connections are mapped in the model library. Neural parameters for the instantiated Populations are represented in the **cell\_instantiation**, which links it to a specific neural model (and translation method) contained in the model library through **cell\_methods**. Projections, their parameters and plasticity algorithms are linked to their translation methods in the model library in an analogous way.

- **partPopulations** and **partProjections** are a partitioned translation of the *Model View* representation, and constitute the *System Level* representation. Populations and Projections are partitioned, accordingly to customizable splitting and grouping rules, into entities that can be mapped on single processors.
- The **processors** entity describes the geometry and size of the available SpiNNaker system, by listing all the available chips and cores, arranged in an XY grid of chips with 16 . Malfunctional cores can be reported at this stage, and are mapped out during the mapping phase.
- Groups of **partPopulations**, and their corresponding **partProjections** are linked to available processors through the **map** entity. The **Device Level** representation is obtained in this way, and the model is mapped to an instantiation of a SpiNNaker system, and it is possible to use the map to translate system level entities (e.g. routing keys) to model entities (e.g. a neuron in a Population) and viceversa.

Instantiation of single neurons and synapses on specific chips and cores can occur now that the model has been placed on the system, using the high-level, hierarchical Populations and Projections structures contained in the model structure.

#### 4.2.5 Design and implementation choices

The SpiNNaker system is situated in the computational neuroscience ecosystem; this is steadily producing novel models, approaches, and measurement and data analysis techniques. The design is an open system which can rapidly accommodate different types of model, network, neural framework and hardware configuration available. PACMAN does this by using a data model which represents the conceptualization of the entities involved in the process: system resources, modelling resources, and the three representations of the problem (model, system, device). PACMAN comprises

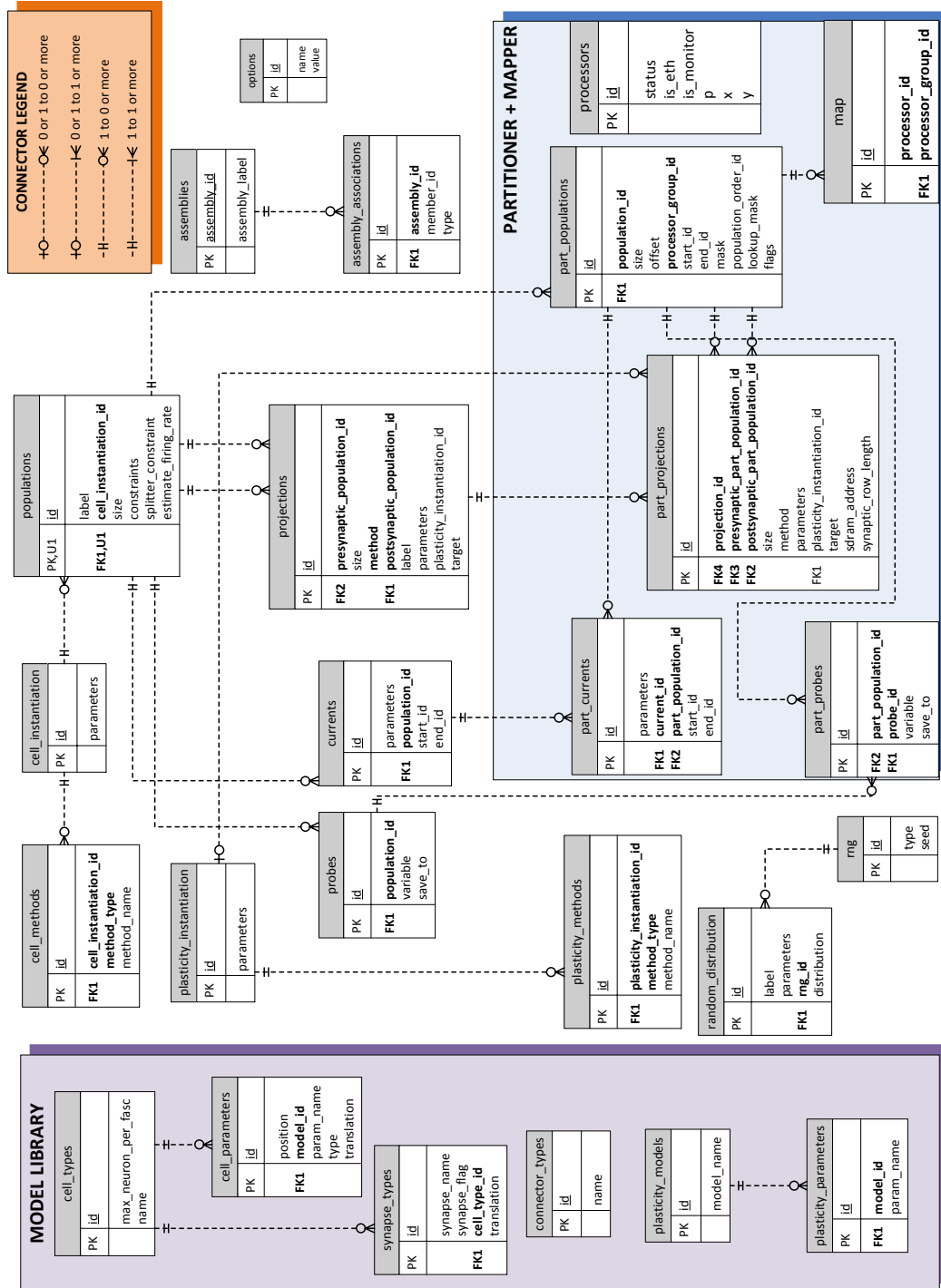


Figure 4.5: PACMAN entity-relationship diagram.

the process which accesses this data, and performs a transformation of the model between such representations, hence creating a map from the model to the system and vice versa.

Following the separation of concerns principle [Parnas, 1972] the process has been modularised, and plug-ins can be inserted between PACMAN's modules for application-specific requirements, customizing the control flow of PACMAN. Data is stored in a database, hence separating it from the control flow; modules access data through an abstract interface; this hides the information about data organization, making exploring different data storage techniques possible. The process can then be modularised, as shown in the previous section, and new system or modelling resources can rapidly be added as translation methods for standard neural languages. More importantly, it is possible to alter the control flow of the process by conditionally executing modules, meeting the requirement of flexibility.

Python has been chosen as the implementation language as many of the tools described are written in Python or have Python bindings [Davison et al., 2008, Gewaltig and Diesmann, 2007, Stewart et al., 2009, Carnevale, 2007, Goodman, 2008, Yger et al., 2009]. An SQLite database has been chosen to represent the data, and PACMAN interacts with it through a data access object (DAO) module.

#### 4.2.6 Introducing new types

The previous section presented how cells can be divided into populations and placed on the system according to their connectivity. Within this framework, new neural [Galluppi et al., 2012c], synaptic [Sharp et al., 2011] and plasticity [Davies et al., 2012a] models can easily be integrated. All these elements have a representation in the *Model Library* which contains the translation methods for those objects into the SpiNNaker data structure. Using this approach, new models based on the event-driven API can easily be incorporated in a neural application framework presented in Sharp et al. [2011], and generated using the approach described. PACMAN can be used to specify the translations (order, data types, size) of the parameters describing the neural model. This makes exploration of new models with the machine rapid, exploiting the ARM cores' programmability.

### 4.2.7 Compatible front-ends

Typically, dedicated hardware platforms have their own proprietary front-end, or offer compatibility with a single standard one [Brüderle et al., 2011, Nageswaran et al., 2007] or don't address compatibility with a standard description language at all [Bhuiyan et al., 2010, Cassidy et al., 2011]. An intermediate model-to-system translation layer which effectively decouples the front-end model language from the hardware back-end configuration has been proposed. As a result of this operation it is possible to plug in different front-ends, analysis or representation tools to the system by interfacing with PACMAN, as the process of configuring and compiling the data structures is the same regardless of the language used. The Population/Projection abstraction is already built into many of the front-ends considered, making the integration seamless. The *Model Library* stores the neuron, synapse, plasticity and connector models, containing language-neutral translation methods for the models available on the system.

- **PyNN:** as an emerging standard, PyNN is a central tool that can be used to exchange and validate models and results between different groups or compare neuromorphic hardware [Nageswaran et al., 2007, Brüderle et al., 2011]. The High Level API is implemented as a natural extension of the *Population* and *Projection* objects used in the language. *OneToOne*, *AllToAll*, *FixedProbability* and *FromList* connectors are supported (which subsumes the remaining connector types, e.g. *DistanceDependent*). As discussed in Section 3.2.7, PyNN supports standard cells and plasticity methods; this is extended to incorporate the Izhikevich neural model and to test novel plasticity algorithms [Davies et al., 2012a].
- **Nengo:** in contrast to the other front-ends considered, Nengo [Stewart et al., 2009] is specific to the Neural Engineering Framework, which computes connection and weights between groups of neurons to achieve the desired neural computation [Eliasmith and Anderson, 2003]. It demonstrates the flexibility of the system, in that it proves possible to translate a model from Nengo to PACMAN and generate the correct structures for the model, by implementing an *encoding* and a *decoding* cell type [Galluppi et al., 2012c].
- **NEST:** a preliminary NEST plugin based on PyNEST [Eppler et al., 2008] has been developed which can be used to create populations of neurons and connections between them. Population and Projection concepts are modelled in

NEST through the *Create* and *Connect* commands, while the *Device* object in NEST nicely maps to a generic application core doing parallel data collection and analysis rather than neural simulation.

To analyse the system or the output data collected from it, different types of tool can also be integrated within the current framework:

- **I/O mapping:** in addition to being used at model generation time, PACMAN can be used to translate information flowing from/to the system at run time, and also makes the mapping of AER devices [Lazzaro et al., 1993] in the system manageable as another system resource. This lets the end user work exclusively at the model level; PACMAN manages the translation to the Device level when sending/retrieving information to specific populations or neurons when, for example, interfacing with data visualisation software [Patterson et al., 2012a], with a peripheral [Galluppi et al., 2012a] or both [Galluppi et al., 2012b].
- **Network Analysis tools:** it is possible to consider *Populations* and *Projections* (or their *part* versions) as nodes and edges of a graph. A weight can then be introduced on the edges, representing the connection density between two populations. It is also possible to represent the Device level in this way, considering cores and routes between cores as nodes and edges, and using the connection density metric to optimize the placement of cores on the system, for example, by clustering the most connected cores or by using existing graph representation/analysis tools (such as Networkx or GraphML). Those can directly be interfaced with the Mapping stage at SQL level to improve placement of the model on the device.

## 4.3 Results

The approach proposed, and particularly the integration with PyNN (see Section 3.2.7), makes running tests with different sets of models on the platform straightforward, and, in fact, it has been used to configure the three generations of SpiNNaker system produced to date: the first test chip (containing 2 ARM cores per chip), a 4-chip board and a 48-node board. The following section describes some basic testing which has been run to validate both hardware and software infrastructure as they have been developed and introduced.

### Example Benchmark Network

The network, presented as an example in Section 4.2.1 (Figure 4.4), can be used to study the balance of excitation and inhibition in a neural network, and is commonly considered a spiking neural network benchmark [Brette et al., 2007], as it can easily be scaled up while maintaining the oscillatory network dynamics, presented in Figure 4.6 for a model with 5000 neurons, intact. The model [Vogels and Abbott, 2005] consists of a network of excitatory and inhibitory neurons, connected via current-based first order synapses (injected current with instantaneous rise and exponential decay, *IF\_curr\_exp* in PyNN's terms) with a 2% probability of interconnection. The network is an example distributed with PyNN<sup>1</sup>, and it can be executed on different simulators by changing the arguments to this script (see Appendix C.1.2):

```
usage = """Usage: python VAbenchmarks.py <simulator> <benchmark>
    <simulator> is either neuron, nest, brian or pcsim
    <benchmark> is either CUBA or COBA."""
simulator_name, benchmark = get_script_args(2, usage)
exec("from pyNN.%s import *" % simulator_name)
```

The excitatory and inhibitory populations are instantiated with a single command line each, and their sizes can be derived by desired model size, keeping a 4:1 ratio between excitatory and inhibitory cells in order to maintain the network dynamics stable across different network size.

```
n          = 5000  # number of cells
r_ei       = 4.0   # number of excitatory cells:number of inhibitory cells
n_exc     = int(round((n*r_ei/(1+r_ei)))) # number of excitatory cells
n_inh     = n - n_exc          # number of inhibitory cells

exc_cells = Population(n_exc, celltype, cell_params, label="Excitatory_Cells")
inh_cells = Population(n_inh, celltype, cell_params, label="Inhibitory_Cells")
```

The sets of parameters *cell\_params* for the 2 populations modelled need to be expressed only once, and these parameters are saved in a dictionary and are applied to all the neurons in the population:

```
celltype = IF_curr_exp

# Cell parameters
area     = 20000. # (um^2)
tau_m    = 20.    # (ms)
```

---

<sup>1</sup>The full script is available at <http://neuralensemble.org/trac/PyNN/wiki/Examples/VogelsAbbott>

```

cm      = 1.      # (uF/cm^2)
g_leak  = 5e-5     # (S/cm^2)
v_thresh = -50.    # (mV)
v_reset = -60.    # (mV)
t_refrac = 5.     # (ms) (clamped at v_reset)
v_mean  = -60.    # (mV) 'mean' membrane potential, for calculating CUBA weights
tau_exc  = 5.     # (ms)
tau_inh  = 10.    # (ms)

area    = area*1e-8
cm      = cm*area*1000
Rm      = 1e-6/(g_leak*area)

cell_params = {
    'tau_m'      : tau_m,      'tau_syn_E' : tau_exc,  'tau_syn_I' : tau_inh,
    'v_rest'     : E_leak,     'v_reset'   : v_reset,  'v_thresh'   : v_thresh,
    'cm'         : cm,         'tau_refrac' : t_refrac, 'i_offset'   : 0}

# setting connection parameters

Gexc = 0.27      # (nS)
Ginh = 4.5       # (nS)

w_exc = 1e-3*Gexc*(Erev_exc - v_mean) # (nA) weight of excitatory synapses
w_inh = 1e-3*Ginh*(Erev_inh - v_mean) # (nA)
delay  = 1.0

# defining connectors
exc_conn = FixedProbabilityConnector(pconn, weights=w_exc, delays=delay)
inh_conn = FixedProbabilityConnector(pconn, weights=w_inh, delays=delay)

# creating Projections
connections={}
connections['e2e'] = Projection(exc_cells, exc_cells, exc_conn, target='excitatory')
connections['e2i'] = Projection(exc_cells, inh_cells, exc_conn, target='excitatory')
connections['i2e'] = Projection(inh_cells, exc_cells, inh_conn, target='inhibitory')
connections['i2i'] = Projection(inh_cells, inh_cells, inh_conn, target='inhibitory')

```

Figures 4.8 and 4.9 present the performance of PACMAN with different network sizes. This network represents a worst-case scenario, as every partPopulation (and consequently every core in the system) is interconnected and every partProjection is probabilistic. The overhead of creating partProjections is maximal as they are then utilised with only 2% connection density.

It is worth noticing that the length of the input script to be compiled does not change when scaling the network up, conversely to the approach to configure the

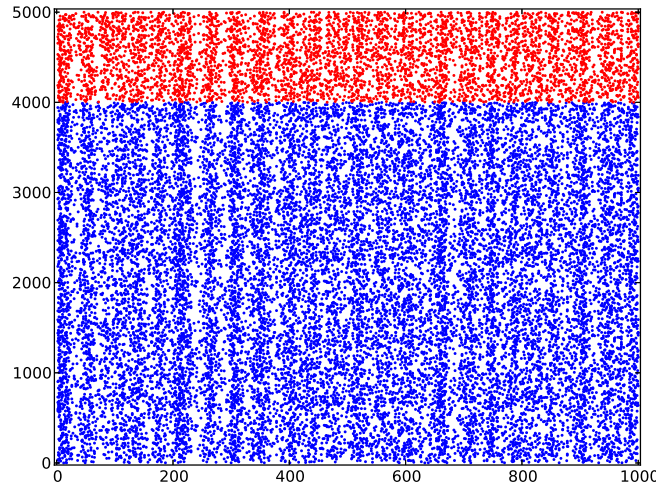


Figure 4.6: Example Benchmark Network Dynamics. An excitatory (blue) and an inhibitory (red) population are interconnected, giving rise to the oscillatory activity reported in the raster plot.

platform presented in [Jin et al. \[2010\]](#), where every neuron and every synapse needed to be explicitly parametrised, mapped and routed.

### Synfire Chain

Synfire chains are well-studied systems of accurate signal propagation in networks of spiking neurons [[Vogels and Abbott, 2005](#)]; a simplified model offers an ideal mapping benchmark, since scaling the number of units while keeping the network dynamics intact is a relatively easy task. As the network is scaled up the interconnection probability is scaled down so as not to change the dynamics [[Brüderle et al., 2011](#)]. Figure 4.7 shows the general connectivity pattern of the synfire chain: each layer is composed of a population of excitatory neurons (red circles) and a population of inhibitory neurons (blue triangles); each excitatory population is interconnected with both the inhibitory and excitatory population in the next layer in a feed-forward inhibition fashion [[Kremkow et al., 2010](#)] to make it stable; the inhibitory population suppresses the corresponding excitatory population. Figure 4.7 presents the results for a run of a synfire chain composed of 60 nodes of 100 excitatory + 25 inhibitory neurons each; the first population stimulates the single propagation through the chain; figure 4.9 summarises build times for different nodes/network sizes. The placing process (splitting, grouping and mapping populations and projections to the

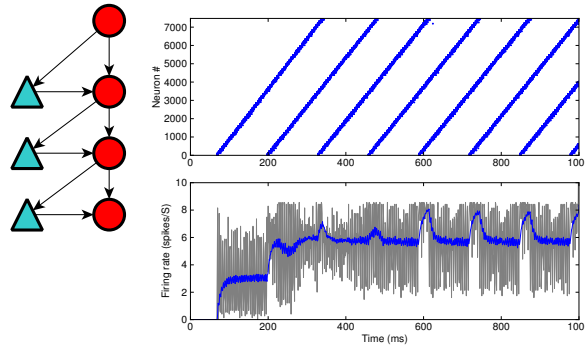


Figure 4.7: Results from a synfire chain of 60 nodes of 100+25 neurons each, interconnected as shown in the left part of the figure. Activity is propagated from one layer to the other, as shown in the raster plot (top right). The mean network firing rate is shown in the bottom-right part of the figure.

system) is fast for every experiment, with a maximum in Synfire #2 which contains more nodes and populations.

### Randomly Connected Network

The previous two examples have dealt with very different interconnectivity patterns: in the first, any given neuron has a finite probability to connect to any other neuron in the network. In the second there is a fixed feed-forward connection topology, where each population projects to the next level or receives projections from the inhibitory cells in the same layer.

To study intermediate classes of topologies a network where populations are randomly interconnected is introduced; neurons are injected with random currents as so to sustain activity. While neural dynamics are of limited interest, this model can be used to explore different mappings on the system as the number of populations and their interconnections varies, and it can, for instance, represent how signals from a group of functionally specialised neural populations are integrated [Tononi et al., 1998]. This network topology takes advantage of fewer (but denser) partProjections between cores and can therefore be considered as an intermediate case.

Results from the script presented in Appendix C.1.2 are reported in Figure 4.8 and 4.9 and demonstrate that as the indexing overhead is reduced (few partProjections) and the probability of synaptic connection is high (50%, making dense Projections), synapse generation is considerably faster than in the Example Benchmark

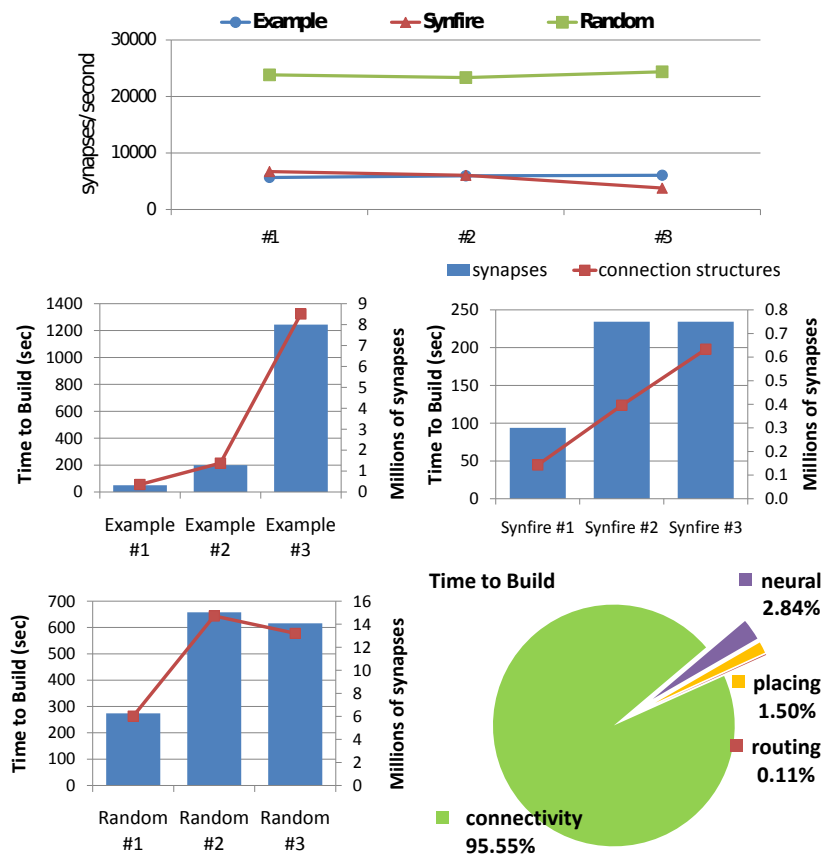


Figure 4.8: 2x2 PACMAN performance analysis. For a given model topology, the number of synapses created per second is constant, leading to a linear scaling up of execution time when adding more synapses. The time to build synapses is the dominant cost in all simulated models.

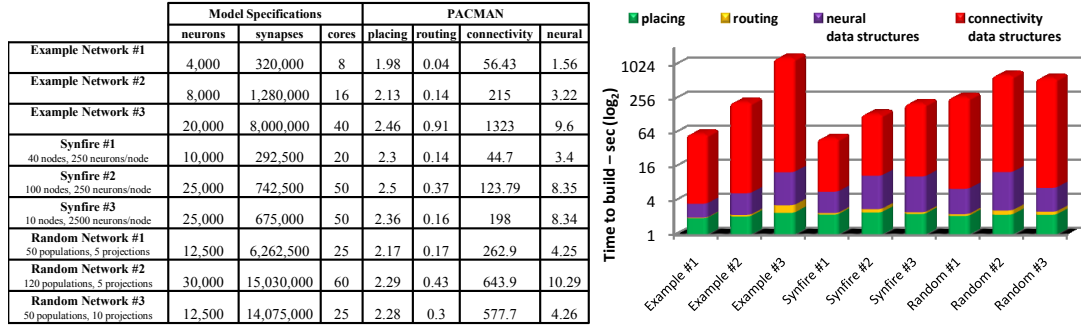


Figure 4.9: Time building results for models run on a 2x2 system. The time to build synapses is the dominant cost in all simulated models. Mapping takes advantages of the Population/Projection abstraction: in the last network reported in the table, instead of mapping 12,500 neurons, 25 partPopulations (cores) need to be mapped; instead of routing 14M synapses, only  $50 \times 10 = 500$  partProjections need to be routed, as each of the 50 populations is connected randomly to the other 10 populations.

Network. It is worth noting that, for a given model topology, the number of synapses created per second is constant (see Figure 4.8), leading to a linear scaling up of execution time when adding more synapses.

## 4.4 PACMAN for larger systems

In previous sections an approach for configuring 2x2 systems has shown that the limiting factor for large systems is the generation of complex data structures (see Figure 4.8) that need to be indexed by every core in every chip. The same approach can be used to test bigger systems without simulating the data structure creation. As models that maintain their dynamics under scaling have been chosen, the number of neurons per population can simply be increased to fill a much larger system. System size and geometry can easily be modified by adding entries in the *Model Library* representing new chips: PACMAN will then consider them in the mapping phase as available resources. To test how the population-based placing approach works on bigger systems, a network of 16x16 SpiNNaker chips is simulated, assuming 16 cores available for neural modelling per chip for a total of 4,096 cores available for neural simulation.

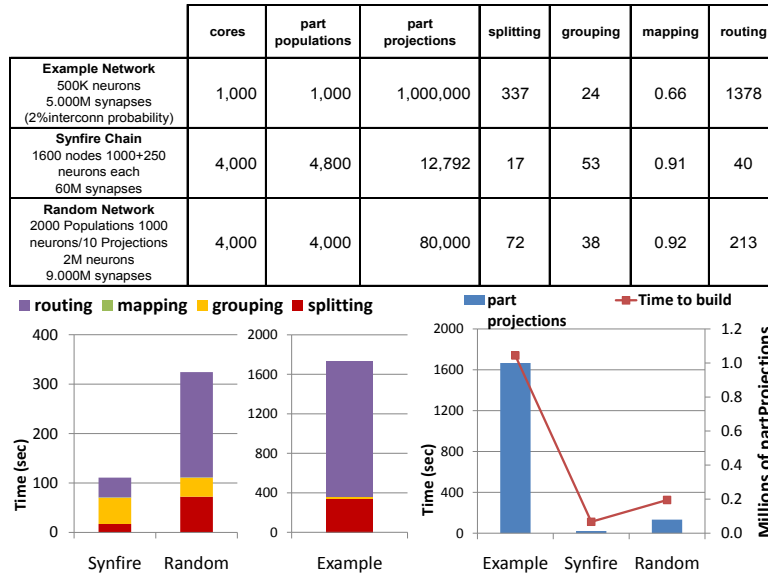


Figure 4.10: Results from mapping and routing models on a 16x16 system. Scaling up the system shows further advantages of the mapping approach proposed, based on the Population/Projection abstraction: in the last network reported in the table, instead of mapping 2,000,000 neurons, only 4,000 partPopulations (cores) need to be mapped; instead of routing 9,000,000,000 synapses, only 80,000 partProjections need to be routed.

## Results

Figure 4.10 presents the build times for each PACMAN stage and model specification; as in the previous section, cores model 500 LIF neurons each. The Splitting stage is responsible for re-indexing Projections into partProjections, and as their number increases the process takes longer, as does the routing necessary to route such partProjections. This is evident from the last plot: the time to build is directly proportional to the number of partProjections in the system. As the previous section discusses, the example benchmark network is worst-case, both for the placing strategy and for the system itself. It is no surprise then that, regardless of the neuron and synapse count, this is the model that takes longest to build. A model with fewer, but more dense, Projections (such as the Random Network) takes less time to build despite having  $4\times$  more neurons and twice as many synapses as the Example Network. The Synfire Chain Network is the fastest to build, as it has fewer Projections/synapses, but stresses the Grouper, since inhibitory Populations can be grouped in pairs together in a core. The Mapping stage is trivial in all cases as no constraints are imposed, and hence it just needs to allocate groups to cores serially. These results show the

potential of the system with different connectivity patterns, and hint at the ones that are easier to model on the machine.

## 4.5 Revisiting PACMAN for 48-chip boards

The results presented in the previous section show the limits of the first implementation of PACMAN: the component responsible for writing the synapses in SDRAM is under-performing for a 4-chip board and cannot be scaled up to configure the next generation of SpiNNaker machine, the 48-node board with 864 ARM cores ( $10^3$  *machine*); the splitter is shown to be the critical component when systems are scaled up. While the neural data creation rate is constant (as expected since the number of neurons to be modelled by a single core does not change) synaptic data generation under-performs as the mapping gets more complex and the number of synapses increases.

The bottleneck of the actual system has been identified - the generation of complex synaptic structures that must be indexed by every core in every chip. In order to meet the challenges imposed by hardware scaling up such components have been rewritten using the Python numerical library NumPy.

The PACMAN mapping approach makes generation of synapses a parallel process, where each chip can independently compute its own SDRAM entries. This is due to the fact that the model is explicitly mapped and allocated at the end of the mapping process; sub-portions of the model (partPopulations) have been distributed on the system and have their own set of contiguous IDs (routing entries). Data structure creation is an “embarrassingly parallel” problem (each core needs only to configure its local portion of memory independently) and eventually this stage of PACMAN will be executed directly on the SpiNNaker system itself, where it is possible to parallelize the process by distributing jobs to different chips.

As a demonstration of the fact that data generation can be parallelised, the execution of the synapse writer on a host system, generating the SDRAM data structures, has been parallelised using Python threading Queue objects: the process assigns different chips to each job queue that then interrogates the database to retrieve the information to compute the data structures. Table 4.2 shows the result of building some models with the original version of PACMAN and the newly introduced one with or without parallelization, showing performance improvements from  $14\times$  more than  $300\times$ .. The networks (whose parameters and build results can be observed in

the table) used for testing are:

1. a network used to spatially sub-sample an input receptive field of  $128 \times 128 \times 2$  neurons to a  $16 \times 16$  population, as done by the FPGA in the retinal model presented in Section 6.2. This network uses a list connector, where every connection needs to be specified (which is the common case for all the models presented in Chapter 6).
2. a network similar the one used to profile the system in Section 5.2.6, where single-core populations are randomly interconnected by all-to-all connections.
3. the example network used in Section 4.3, using a fixed-probability connector.

Table 4.2: Revisiting PACMAN for 48-chip boards. The process of writing synapses has been optimised and parallelised, showing performance improvements from  $14 \times$  more than  $300 \times$ .

	Network 1	Network 2	Network 3
Neurons	33,024	6,000	6,400
Synapses	32,768	720,000	3,200,000
Cores	17	60	64
Chips	2	4	4
PACMAN (original) (sec)	1,015	314	157
Synapse writer (original) (sec)	993	303	148
PACMAN (new) (sec)	9	31	26
Synapse writer (new) (sec)	3.2	22	21
PACMAN (parallel) (sec)	N/A	21	14
Synapse writer (parallel) (sec)	N/A	13	10

The models have been tested with the original version of PACMAN and compared to the newly introduced one, with and without parallelization of synapse generation on 4 different threads.

Finally, the PACMAN process has been tightly integrated in Python, by letting external users or developers insert plug-ins at each stage of the process; as a consequence control flow can be altered by conditionally executing such plug-ins. An example of the PACMAN process used to produce the results presented in the next chapters is presented in Figure 4.11:

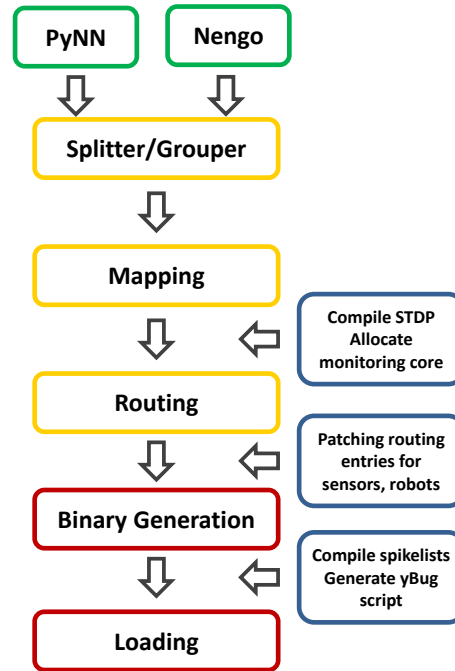


Figure 4.11: PACMAN execution flow example. Green boxes represent the front-end modules, responsible of representing the model; yellow boxes represent the PACMAN modules, responsible of mapping the model onto the system; red boxes represent the machine-specific binaries generation and loading stages; blue boxes represent plugins that can be conditionally inserted at different stages of the process.

- Neural models are written in PyNN or Nengo (green).
- Partitioning, mapping and routing are performed in the PACMAN stage (yellow).
- Data structures are compiled and loaded on the system (red).
- Specific plugins for allocating monitoring instrumentation, mapping robots and aer sensors etc. can be inserted at different stages of the process (blue).

## 4.6 An open system

The software infrastructure presented above can be used to configure a parallel digital hardware system for simulation of spiking neural networks by exploiting its

arbitrary remapping capability, both on a single computational node and at a connectivity level. The approach takes advantage of hierarchical information embedded in the model at a neural *Population* level to drive the configuration of the system in an efficient way; this defers the data file compilation stage until after resource mapping on the system, thus simplifying its parallelization (and hence the generation of the configuration data structures on the machine itself). This is a key result of the methodology chosen: by using a hierarchical approach and considering Populations and Projections rather than flattening single neurons and synapses, all the mapping and placing algorithms have lower complexity, while still maintaining a full low-level representation at the device level. The complexity of the proposed approach is proportional to the number of *partPopulations* (as limited by the number of cores) in the model. For larger systems, as processors are freely available, grouping should be applied only if the number of neurons in a population  $n_{pop}$  is smaller than the number of neurons  $n_{core}$  that can be modelled by a single core; the introduction of this stage makes the process more complex, and in the mentioned case resource savings will not be significant, as only few more cores will be used by bypassing grouping altogether. The results presented in the previous section show interesting aspects of the problem and of the approach proposed to solve it. It is evident that, for the scale of systems considered in this section, the most complex task is compiling the data structures, especially for synaptic data. Indeed, the allocation and mapping on a system of this size is trivial, as is routing, since it is done on a core/*partPopulation* basis rather than on a single neuron/synapse level.

A key result of this approach is that, at the end of the Mapping stage, information is already represented in a local, core-based way and can therefore be parallelised and distribute on the machine itself, greatly speeding up the most critical processes and exploiting the system architecture in the configuration phase. At this point each neuron has been allocated to a core, and it is possible to route the network, associating neurons to routing keys, even though the data structures do not as yet exist. Therefore, this part of the process can be done on the host machine, and the hierarchical representation permits the use of interval routing [Van Leeuwen, 1987], simplifying the problem greatly. Allocating neurons to system resources can be done by sending the Population parameters to the corresponding core. Each core can then receive information about the incoming *partProjections* (parameters, range of routing keys considered, connectivity pattern) from PACMAN, and initialise, compile, organize

and index its own synaptic data structures in SDRAM, having all the necessary information represented locally. Hence the host system needs only to send abstract information to the machine which self-configures by populating its local portion of memory.

The proposed approach has been integrated in a software infrastructure, called the *SpiNNaker package*, a self-contained package comprising all the software tools needed to run spiking neural simulations on SpiNNaker. The SpiNNaker package has been the developing platform for many models, including ones which explore new plasticity methods [Davies et al., 2012a] or cortex micro-circuitry [Sharp et al., 2012]. Creating new neural applications, which can then be used seamlessly as computational resource nodes, is an approach that greatly exploits the reconfigurability of a digital architecture [Sharp et al., 2011] and can be used to introduce new computational frameworks rapidly for the architecture [Galluppi et al., 2012c]. For general purpose use, a user will, in future, be able to create his own neural model specifying dynamical equations as is done with Brian [Goodman, 2008]. As the system is completely event-driven, this neural model creation tool will map dynamics to events [Rast et al., 2010a], making use of code generation techniques [Goodman, 2010] either from the modelling language or from a standard XML format [Goddard et al., 2001]. Tutorials on how using SpiNNaker for neural network simulations with the SpiNNaker package have been offered in Telluride and Capocaccia workshops<sup>234</sup>, and the documentation is available on the SpiNNaker website<sup>5</sup> and attached in Appendix C.4.

## 4.7 Summary

This chapter presented a large parallel system and the hierarchical approach proposed to host neural models on it. The proposed layer effectively decouples the model, written in a user-selected front-end language, and the device level, abstracting the hardware from the end user and letting non-hardware experts exploit the reconfigurability of the architecture in a seamless way. The proposed method has been tested by configuring the current generation of SpiNNaker systems in a variety of networks with thousands of neurons and millions of synapses. The limitations have

---

<sup>2</sup>See Section 1.3.3

<sup>3</sup><https://capocaccia.ethz.ch/capo/wiki/2013/spinnaker13>

<sup>4</sup><http://neuromorphs.net/nm/wiki/2013/uns13/resources>

<sup>5</sup>[http://spinnaker.cs.man.ac.uk/docs/spinnaker\\_package/](http://spinnaker.cs.man.ac.uk/docs/spinnaker_package/)

been explored and some solutions have been proposed and implemented, discussing the capabilities of an approach which promises to be able to scale up to configure systems of millions of neurons and thousands of processors.

A system which combines off-hardware hierarchical decomposition of the application with on-board generation of the data binaries is a compelling and efficient model for very large-scale parallel applications that maximises the utilisation of available hardware resources. This approach is a natural one whenever the application, like networks of neurons, has its own internal structure which naturally suggests the on-hardware mapping. This kind of representation is also commonly used by standard neural languages, making it the natural choice for guiding resource allocation on the system. The system has been designed to be open to alternative control flows, incorporate model-specific changes and new models by the conceptualization of three levels of abstractions, from the model to the machine; the system is therefore able to incorporate changes and different research frameworks rapidly. The next chapter will show how the approach can be used to build models using standard languages and running them on SpiNNaker.

# Chapter 5

## Mapping Different Front-ends

### 5.1 Introduction

Having introduced an approach to configure a general-purpose neural multi-chip system, this chapter describes in more detail how emerging standard neural network modelling languages can be interfaced to the platform. The process of writing and loading neural network models on the SpiNNaker hardware is described, using examples from two neural languages and showing results obtained with them.

The first section focuses on the integration of PyNN (see Section 3.2.7), a simulator-independent language for neural network modelling, with PACMAN, by using the mapping approach presented in the previous chapter. Integration with PyNN makes cross-platform verification of the platform possible, and opens up network modelling on SpiNNaker to non-hardware experts. The rest of the section shows some standard tests and models simulated with the architecture, including multi-model networks, and ways to evaluate the performance of the platform by building controlled PyNN models.

The rest of the chapter describes how the Neural Engineering Framework (see Section 2.3.3) can be ported onto SpiNNaker, by interfacing it with Nengo, the simulator used to implement the NEF principles; this lets a user rapidly build complex neural models without detailed knowledge of the Framework principles or the hardware architecture. Basic models exemplifying NEF principles are simulated on SpiNNaker, and results are presented at the end of the section.

## 5.2 PyNN and PACMAN

Simulations on dedicated hardware are more efficient than those on general-purpose computers but, due to the proprietary nature of dedicated hardware, writing a network model and running it becomes a task accessible only to the system creators. A user can be discouraged by the amount of time required to learn the architecture and configuration modalities of a dedicated hardware system [Steinkraus et al., 2005]. To simplify this process we develop a PyNN [Davison et al., 2008] interface with PACMAN. It abstracts the hardware from the end-user point of view by implementing a module for writing models in PyNN, a simulator-independent language for building neural networks described in Section 3.2.7. The system represents an attempt towards a universal neural network simulation system. The rest of the section describes the role of the PyNN module in the platform architecture and the implementation of the interface. Results in PyNN are presented and compared against other simulators, reproducing single neuron and network dynamics, successfully testing the hardware and validating the implementation.

### 5.2.1 The PyNN.SpiNNaker module

For PACMAN to be used to guide the placement of a PyNN model through different components and different chips of the SpiNNaker system, this needs to be translated to a Model level PACMAN view. PACMAN is then responsible for translating the Model view into SpiNNaker data structures that will then be used to configure different parts of the system, such as the Tightly-coupled memory (TCM) of the ARM cores, the SDRAM and the Multicast Router of each chip. The PyNN high-level API exposes the Populations and Projections natively, making mapping straightforward. Neural, connector and plasticity models are described in PACMAN's model library and can be mapped directly in PyNN by specifying automatic translation methods to drive the data structure generation process.

The PyNN.SpiNNaker module for PACMAN is also responsible for scripting the execution of low-level tools, in order to automate and abstract those configuration steps from the final user. Those steps include generating extra files describing the system architecture and SpiNNaker specific parameters, dynamically calling low level compilers, interacting with the ethernet interface to load files on the chip, starting the simulation, and retrieving simulation results.

## Populations

SpiNNaker is a universal platform for spiking neurons in the sense that no neural model is embedded in the hardware, but applications that take advantage of the parallel distributed architecture and communication infrastructure can be developed, integrated and switched rapidly. Some neural models are currently available for SpiNNaker and can be used through the PyNN interface: Izhikevich [Jin et al., 2008] and Leaky Integrate-and-Fire (LIF) [Rast et al., 2010a] modules, both current and conductance based synapses, and various spike source population types.

PACMAN takes a description of the network compiled with PyNN and allocates Populations to chips, splitting them into partPopulations if needed. Hence, neurons in populations share a contiguous portion of the routing space and sequential neural IDs, greatly simplifying allocation and generation of binary structures from the model description. The neural application along with state variables and parameters are placed in the TCM of each core, since the neural application continuously needs them in order to update neuron equations, process new inputs, plasticity etc. PyNN is used to model neurons as parameter place-holders, since the simulation will run on-chip. In this way, customization of PyNN with non-standard models can be very rapid, and PyNN can be used as a configuration interface for dedicated hardware. Translation methods can be described in the Model Library, making translation automatic. An example of the translation of an Izhikevich neuron in SpiNNaker data structures by this automatic process is given in the portion of the Model Library relative to the Izhikevich neuron, and reported in Table 5.1, along with the size (type of variable) and position in the final compiled structure. *params* is a matrix containing the parameters for all the neurons in the population; *p1* and *p2* are the scaling factors, required to represent floating point numbers on SpiNNaker. For the details about the neural model and the transformations needed to represent it on SpiNNaker refer to Jin et al. [2008].

## Projections

Neural connections are configured in two different places in the SpiNNaker system. Routing tables route spikes from pre- to post-synaptic neurons, while synaptic information (weight, delay, etc.) is stored in the SDRAM of the post-synaptic neuron(s). Routing tables and memory maps for each chip are generated by PACMAN, starting from the Model view of Populations and Projections and their parameters as

Table 5.1: Translation methods for the Izhikevich neuron.

Parameter Name	(C) Type	Translation method	Position
v_init	int	$\text{int}(\text{value} * p1)$	1
u_init	int	$\text{int}(\text{params}[i][\text{'u\_init'}] * p1)$	2
a	short	$\text{int}(\text{params}[i][\text{'a'}] * \text{params}[i][\text{'b'}] * p2)$	3
b	short	$\text{int}(-\text{params}[i][\text{'a'}] * p2)$	4
c	short	$\text{int}(\text{params}[i][\text{'c'}] * p1)$	5
d	short	$\text{int}(\text{params}[i][\text{'d'}] * p1)$	6
tau_syn_E	ushort	$\text{int}(p2 / \text{params}[i][\text{'tau\_syn\_E'}])$	7
tau_syn_I	ushort	$\text{int}(p2 / \text{params}[i][\text{'tau\_syn\_I'}])$	8
i_offset	int	$\text{int}(\text{params}[i][\text{'i\_offset'}] * p1)$	9

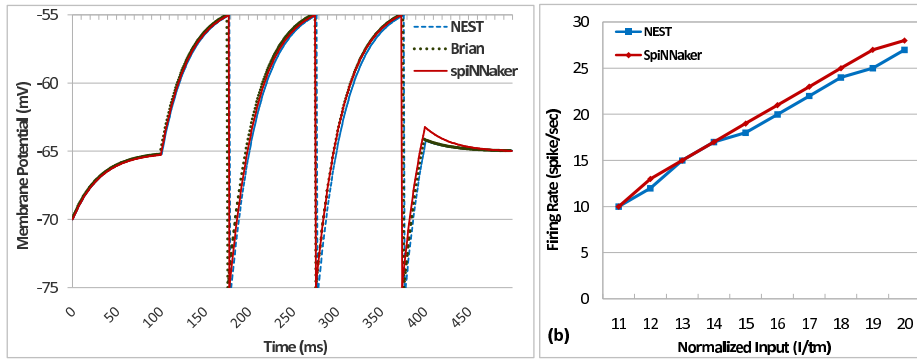


Figure 5.1: (a) Results of a single LIF neuron compared with PyNN.nest and PyNN.brian (b) Comparison between firing rates

extracted from the PyNN network. Projections support both standard and custom plasticity models, through the introduction of a set of dedicated interfaces in PAC-MAN [Davies et al., 2012a].

### 5.2.2 LIF Neuron

The implementation of the LIF SpiNNaker module described in Rast et al. [2010a] is tested by modelling a single LIF neuron. Figure 5.1 displays the results from injecting it with a DC current source strong enough to make it exceed the threshold potential (-55 mV) and fire 3 times in the interval considered

The same script is simulated with two other simulators supported by PyNN (Brian

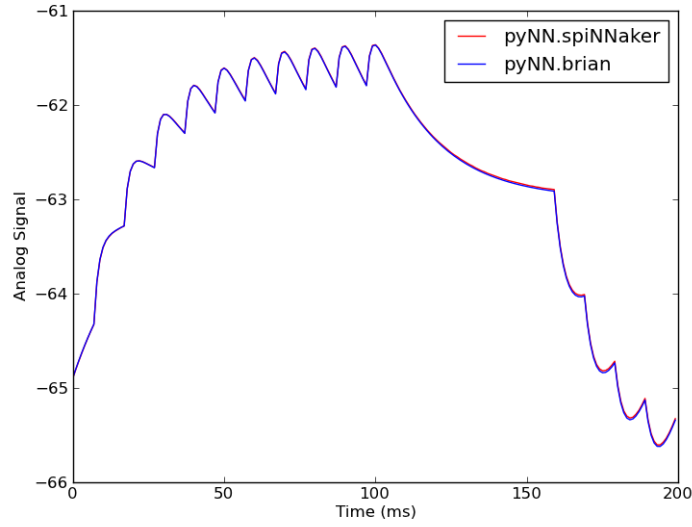


Figure 5.2: Conductance-based LIF neuron simulation on SpiNNaker, compared to Brian (and to Figure 3.6)

and NEST) in order to compare results. As shown in the figure, the membrane potential dynamics are very close in all three simulations; the difference in timing after a spike can be considered to be due to refractory periods not being explicitly modelled on the SpiNNaker LIF module at the time of the test. This can also be observed in Figure 5.1 (b), where the activity rate is plotted against the amplitude of the input current (normalised to the membrane time constant). The number of spikes generated per second is comparable between NEST and SpiNNaker, with SpiNNaker producing spikes more frequently because of the absence of a refractory period in this specific test (see Appendix C.1.3).

A conductance-based LIF module has also been implemented to test the platform with the standard script provided on the PyNN website; the results, if compared against Brian, are illustrated in Figure 5.2; this can be compared with Figure 3.6 presented in the PyNN section (3.2.7).

### 5.2.3 Izhikevich Neuron

Simulation of the Izhikevich neural kernel required an extension of PyNN to support a new neuron model - Izhikevich's. A neuron is stimulated with a DC current source, and spikes are recorded (see Appendix C.1.3 for the full script).

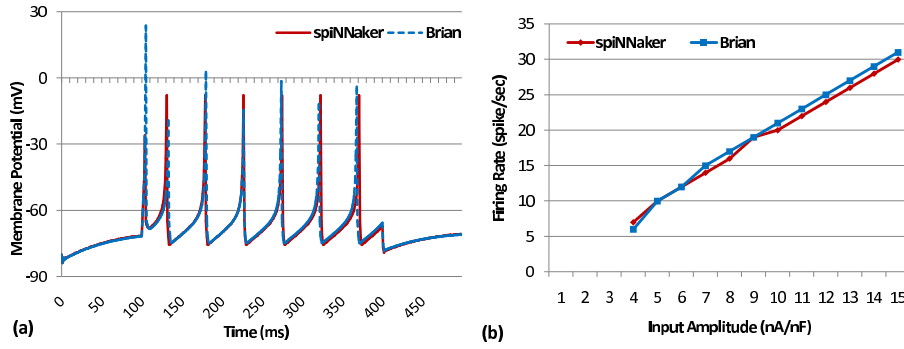


Figure 5.3: Results of a single neuron compared with Brian

```

cell_params_izh = { 'v_init'      : -80.0,    # Initial V
                    'u_in'       : 0,        # Initial U
                    'a'          : 0.02,
                    'b'          : 0.2,
                    'c'          : -65,
                    'd'          : 8,
                    'i_offset'   : 0         # Bias current
                  }

Layer1 = Population(1,                # size
                   Izhikevich,        # Neuron Type
                   cell_params_izh,   # Neuron Parameters
                   label='Layer1'
                  ) # Label

Layer1.record()
Layer1.record_v()

cs = DCSource(amplitude=10, start=100, stop=runtime)
cs.inject_into(Layer1)                # DC Current Electrode inputs to all neurons in i1

```

Since the simulation runs on hardware, no further specification/modelling is required at this stage in PyNN, which is used purely as a configuration tool for lower level hardware; this approach could easily be extended to any other neural model running on hardware that relies on parameter configuration. Since the Izhikevich neuron is not a standard cell type in PyNN, a Brian script has been used to test the implementation. Figure 5.3 shows that the SpiNNaker fixed-point implementation of the Izhikevich neuron follows the dynamics obtained with Brian. Results of the simulation show that the membrane potential dynamics of a single Izhikevich regular spiking neuron [Izhikevich, 2003] injected with a DC source current are similar for Brian and SpiNNaker, despite the fixed point implementation used in the SpiNNaker model, and firing rates are qualitatively the same.

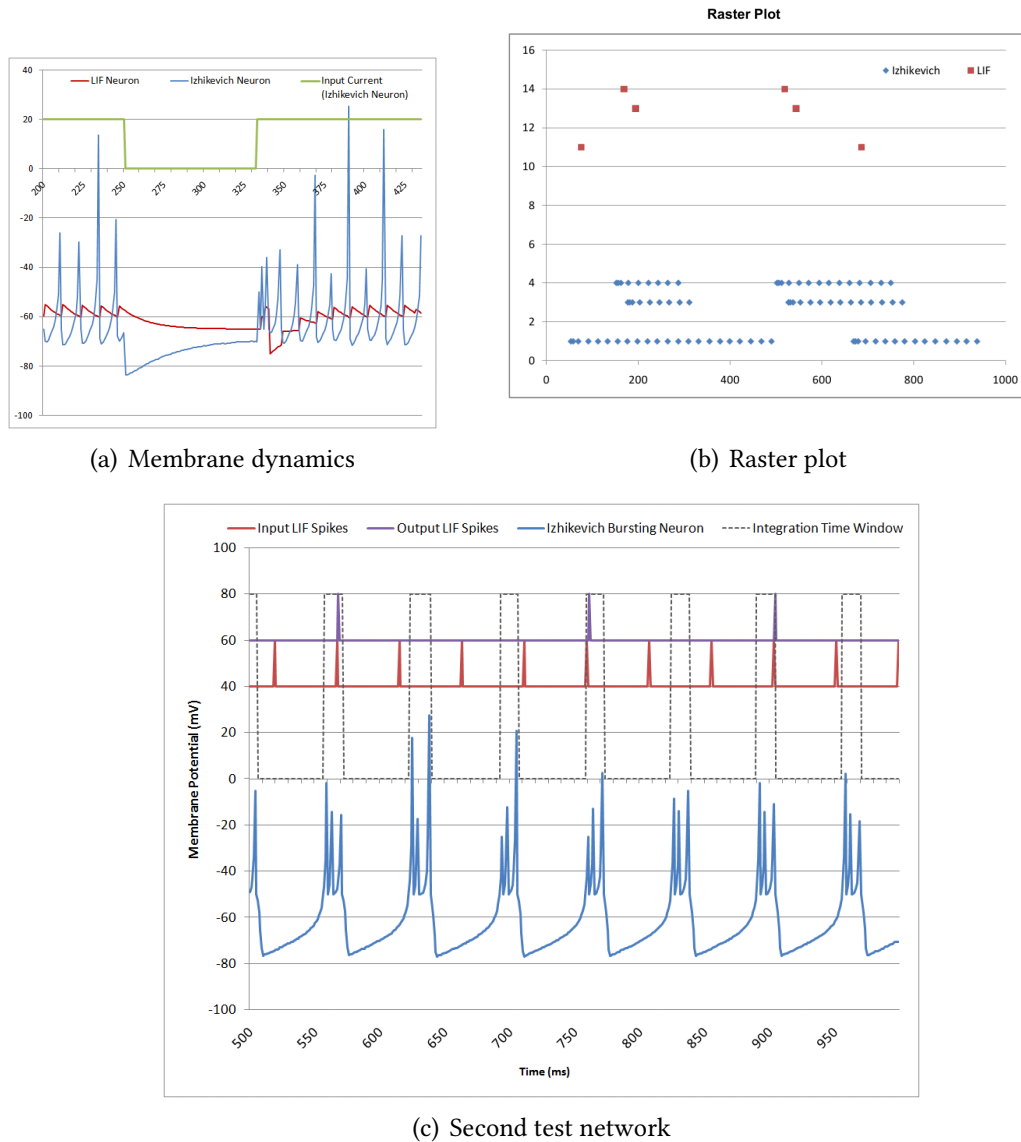


Figure 5.4: (a and b) The first layer serves as an input layer using Izhikevich Intrinsic Bursting neurons. The second layer is an LIF layer, configured so that neurons will emit spikes only if stimulated over a certain frequency. Each Izhikevich neuron emits a burst of 3 spikes, then spikes regularly; in turn each LIF neuron thus spikes only when it receives the initial burst from the lower level population (c) The Izhikevich neurons are Chattering types that generate a burst “clock” to gate input from the Layer 1 LIF neurons, so that the Layer 3 LIF neurons will only fire when an input from Layer 1 is coincident with the burst window.

### 5.2.4 Multimodel simulations

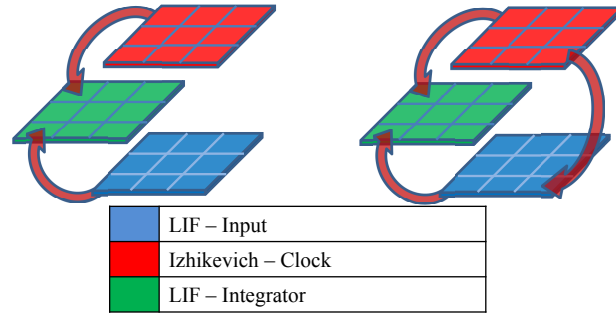
This section demonstrates how SpiNNaker can run simulation concurrently involving different neural models [Rast et al., 2011b], something which is very hard to do on other dedicated platforms such as neuromorphic chips, FPGAs and GPUs, which are typically optimised to run a single neural model. Networks of a few neurons which implement a mixed-model neural network exhibiting complex dynamics are demonstrated.

The first simulated network uses two populations, containing respectively LIF neurons and Izhikevich neurons. The network comprises two layers of 15 neurons each, linked in a one-to-one mapping with the corresponding neuron in the other layer. The first layer serves as an input layer using Izhikevich Intrinsic Bursting neurons. The second layer is an LIF layer, configured so that neurons will emit spikes only if stimulated over a certain frequency. A modulated stimulus of 20 mV is injected into each input neuron. Each Izhikevich neuron emits a burst of 3 spikes, then spikes regularly (Figure 5.4(a)). In turn each LIF neuron thus spikes only when it receives the initial burst from the lower level population (Figure 5.4(b)). Parameters for the simulated test network are as follows: LIF neurons  $\tau_m = 32ms$ ,  $V_r = -65mV$ ,  $V_s = -75mV$ ,  $a = 0.02$ ,  $b = 0.2$ ,  $c = -65$  mV, neurons  $\tau_m = 16ms$ ,  $V_r = -49mV$ ,  $V_s = -70mV$ ,  $V_t = -50mV$ ; Izhikevich neurons,  $a = 0.01$ ,  $b = 0.2$ ,  $c = -50mV$ ,  $d = 5$ ; LIF output neurons  $\tau_m = 16ms$ ,  $V_r = -65mV$ ,  $V_s = -75mV$ ,  $V_t = -55mV$ .

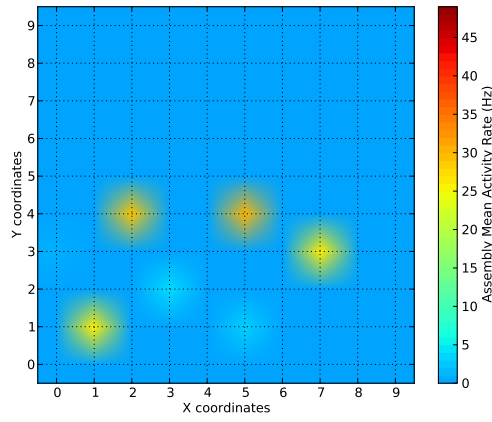
Populations of N neurons can be instantiated and connected with *OneToOneConnectors* in PyNN as follows (see Appendix C.1.3 for the full script):

```
# cell_params will be passed to the constructor of the Population Object
cell_params_izk = {'v_init'      : -65.0,          # Initial V
                  'u_in'        : 0,              # Initial U
                  'a'            : 0.02,
                  'b'            : 0.2,
                  'c'            : -65,
                  'd'            : 1,
                  'i_offset'     : 0               # Bias current
                  }

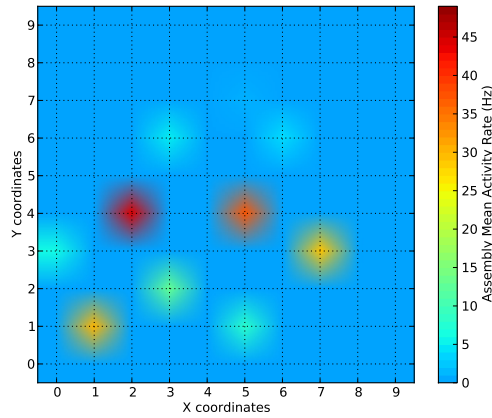
cell_params_lif = {'tau_m'       : 32,             # Membrane time constant
                  'v_rest'       : -65.00,         # Rest Potential
                  'v_reset'      : -75.00,         # Reset Potential
                  'v_thresh'     : -55.00,         # Threshold
                  'v_init'       : -70.00,         # Initial potential
                  'i_offset'     : 0               # Bias current
                  }
```



(a) Network structure



(b) Rate plot of the multimodel network, No top-down influence



(c) With top-down influence

Figure 5.5: Multimodel network of neural assemblies: large-scale structure and rate plots: (a) The network comprises 3 layers: the LIF layer is the 2D structure of neural assemblies. The Izhikevich layer and the LIF integrator layer have the same microscale structure as the networks presented above, but they are also organised into a 2D structure, where each neuron maps a corresponding assembly coordinate. With no top-down connection; (b) the activity of four stimulated assemblies is comparable, while with top-down connections (c) the primed assembly is the most active one.

```

    } # they can be different within/between populations

Layer1 = Population(N,          # size
                    Izhikevich, # Neuron Type
                    cell_params_izk, # Neuron Parameters
                    label="Layer1") # Label
Layer2 = Population(N,          # size
                    IF_curr_exp, # Neuron Type
                    cell_params_lif, # Neuron Parameters
                    label="Layer2") # Label

# Randomize Initial membrane potential
uniformDistr = RandomDistribution('uniform', [-80,-60], rng)
Layer1.randomInit(uniformDistr)
Layer2.randomInit(uniformDistr)

print "connecting"

# Creates feedforward projectionms with random delay
delayDistr = RandomDistribution('uniform', [1,10], rng)

c1_2 = Projection(Layer1, Layer2, OneToOneConnector(), target='excitatory', rng=rng)
c1_2.setDelays(1)
c1_2.setWeights(w/2)      # Weights

c2_3 = Projection(Layer2, Layer1, OneToOneConnector(), target='inhibitory', rng=rng)
c2_3.setDelays(1)
c2_3.setWeights(-w/2)     # Weights

```

In the second network, we configured 3 layers of 15 LIF, Izhikevich and LIF neurons respectively, similarly connected in a 1-1-1 mapping. Thus a single triplet of neurons can be considered as a modular element or “unit”, much like the pairs of neurons in the previous network. The Izhikevich neurons are Chattering types that generate a burst “clock” to gate input from the Layer 1 LIF neurons, so that the Layer 3 LIF neurons will only fire when an input from Layer 1 is coincident with the burst window (Figure 5.4(c)). This network demonstrates the important ability to generate control signal-like inputs to integrating layers, and is easily scalable by using a unit as a modular “drop-in” component which can be instantiated in building-block fashion.

In the final example, we combine results obtained in the previous experiments into a scalable mixed LIF/Izhikevich-neuron model. The network comprises 3 layers: the LIF layer is the 2D structure of neural assemblies. The Izhikevich layer and the LIF integrator layer have the same microscale structure as the networks presented above, but they are also organised into a 2D structure, where each neuron maps a corresponding assembly coordinate (Figure 5.5(a)).

Every assembly is a Python class composed of a layer of excitatory neurons and a layer of inhibitory neurons. The neural type and their parameters, number of neurons and connections between each assembly layer (intra-assembly connections) are dynamically passed to the assembly's constructor as follows:

Layer creation:

```
self.p_exc = Population(n_exc, neuron_model, exc_params, label='%s_exc' %
    assembly_label)
self.p_inh = Population(n_inh, neuron_model, inh_params, label='%s_inh' %
    assembly_label)
```

Connection initialiser:

```
self.exc_exc_conn = Projection(self.p_exc, self.p_exc,
    exc_exc_connector,
    target='excitatory',
    label='%s_exc_exc_conn' % (assembly_label)
)

self.exc_inh_conn = Projection(self.p_exc, self.p_inh,
    exc_inh_connector,
    target='excitatory',
    label='%s_exc_inh_conn' % (assembly_label)
)

self.inh_exc_conn = Projection(self.p_inh, self.p_exc,
    inh_exc_connector,
    target='inhibitory',
    label='%s_inh_exc_conn' % (assembly_label)
)
```

Each assembly consists of an excitatory layer of 2 LIF neurons and an inhibitory layer of 2 LIF neurons, using the above parameters and constructed as follows.

```
assembly(2, 2, cell_params, cell_params,
    exc_exc_connector=AllToAllConnector(weights=w/n_exc, delays=delayDistr,
    allow_self_connections=False),
    exc_inh_connector=AllToAllConnector(weights=20, delays=2),
    inh_exc_connector=AllToAllConnector(weights=-20, delays=2),
    assembly_label='A_%d_%d' % (x, y))
)
```

where x/y specifies the coordinates of the assembly in the 2D mesh.

Assemblies are organised in a 10x10 2D mesh (for a total of 10x10x4=400 neurons) and are interconnected using a distance base algorithm. An assembly connects to all surrounding assemblies within a distance radius of 3. Excitatory cells of the selected

assembly receive a `step_current_source` input, using a construct such as the one below, which specifies an injected current of 2mA into the assembly at coordinates (4,7) starting at millisecond 1 of the simulation.

```
current_source = StepCurrentSource([0,1],      # Time
                                   [0,2])      # Amplitude

cells = []
for i in p[4][7].p_exc:
    cells.append(i)
current_source.inject_into(cells)
```

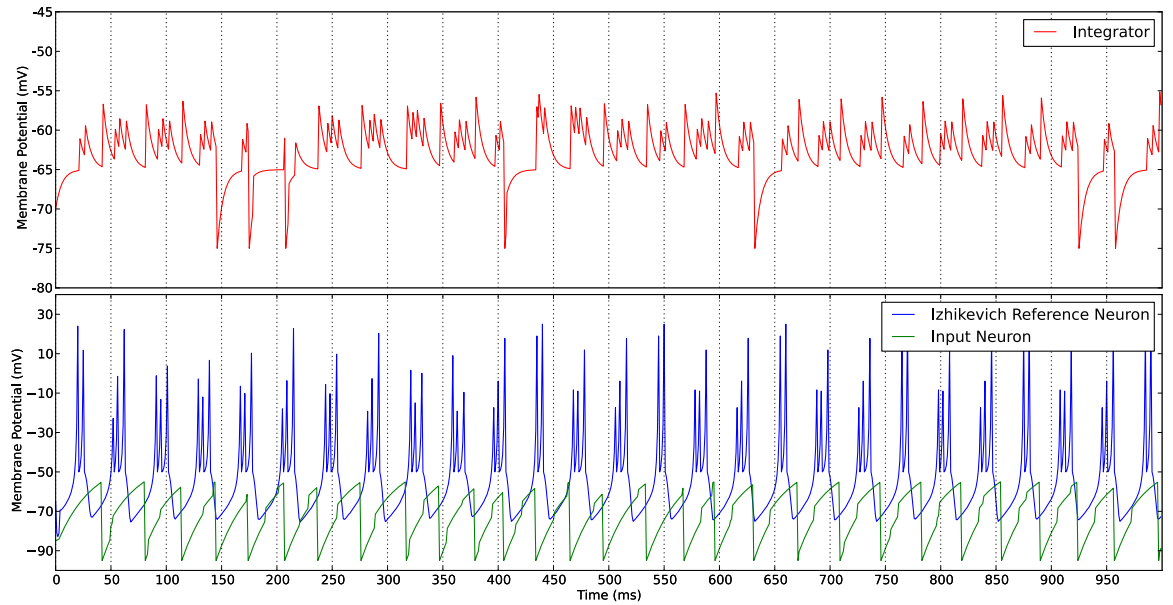
Results from the experiments with no top-down influence are first reproduced. The integrator neuron fires only if there is coincident activity in the input LIF layer and in the Izhikevich “clock” layer. Then a top-down connection is added from the Izhikevich layer back to the LIF input layer which boosts (primes) and synchronizes the activity of the assembly with the corresponding neuron in the clock layer (Figure 5.6). The rate plots clearly show the priming effect: with no top-down connection (Figure 5.5(b)) the activity of four stimulated assemblies is comparable, while with top-down connections (Figure 5.5(c)) the primed assembly is the most active one. This network demonstrates the scalability of a SpiNNaker model at assembly, map dimension and hierarchical levels, along with the advantages of having mixed model networks, such as SpiNNaker is able to support in hardware.

The script describing the network can be found in Appendix C.1.3.

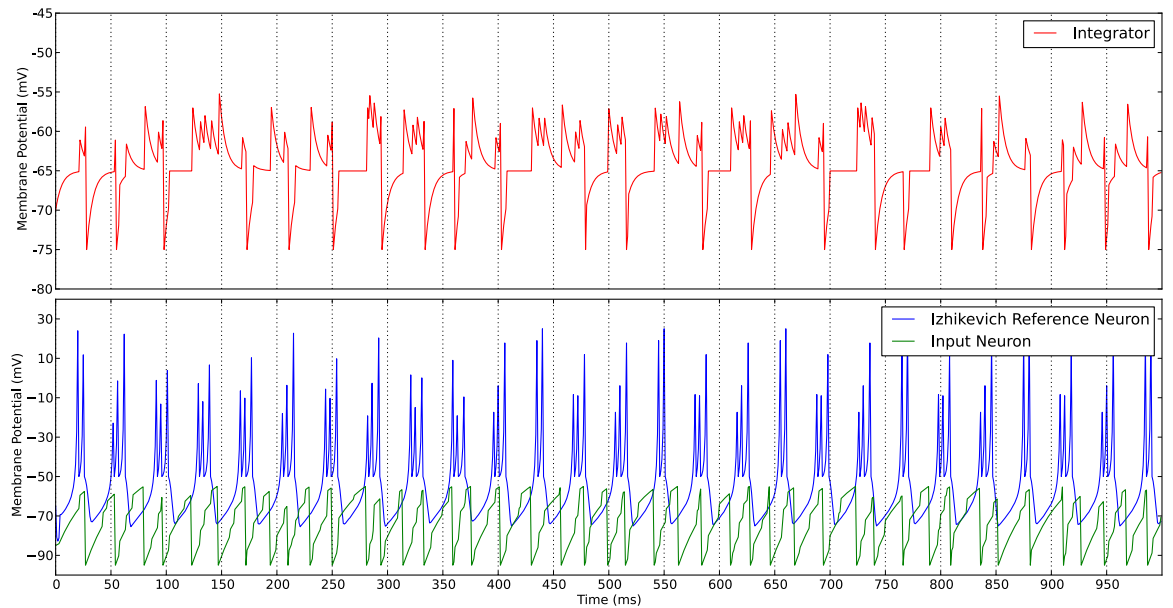
### 5.2.5 Oscillatory Network Dynamics

To test network dynamics a network, inspired by the fourth benchmark proposed in [Brette et al. \[2007\]](#), was implemented using a PyNN script from ModelDB [[Migliore et al., 2003](#)] to verify consistency of results and compare them between different simulators. The network is modelled as 500 randomly interconnected excitatory and inhibitory (ratio 4:1) LIF neurons which interact through voltage deflections [[Vogels and Abbott, 2005](#)]. To compare results against other simulators synaptic time constants were set to be very short, so as to mimic the voltage-jump synapses used by the SpiNNaker LIF module at the time of the test. We compared the results of the simulation on SpiNNaker with the one produced by the pyNN.nest module.

```
n_exc = 400      # number of excitatory cells
n_inh = 100      # number of inhibitory cells
cell_params = {
```



(a) No top-down signal



(b) With top-down signal

Figure 5.6: Activation of a selected triplet of neurons in the mixed-model neural assembly network, with and without top-down priming. With no top-down priming, the integrator layer fires (as seen by the membrane potential recovery downstroke) only intermittently due to random correlations of the Izhikevich and LIF neuron activity. With top-down priming, the integrator layer fires with strong synchrony to the Izhikevich clock layer.

```

'tau_m'      : 32,      'cm'      : 1,      'v_init'     : -60,
'v_rest'     : -49,     'v_reset'   : -60,     'v_thresh'   : -50,
'tau_syn_E'  : 1,      'tau_syn_I' : 1,      'tau_refrac' : 1

```

Neurons are modelled as above, interconnected with a 2% probability. Other network parameters are inspired by [Brette et al. \[2007\]](#) to compare results: weights 0.25 mV and -2.25 for excitatory and inhibitory synapses, connection delay randomly distributed in the interval [1,14] ms.

The results of the simulation are presented in Figure 5.7. As neurons cross the threshold potential they start spiking, building a background activity through random connections. A current is injected into a subset of the population to increase its activity at 1/3 of the simulation time and double it at 2/3 (observed as the patch of neurons with high activity in figure). Despite choosing a standard cell in PyNN (exponential current-based LIF neuron with first-order synapse kinetics) whose synapse dynamics SpiNNaker did not model at the time of the test, a comparison of raster plots and mean population frequency rates using the PyNN.nest module shows that the qualitative behaviour of the network is maintained; the same number of spikes are produced (the difference is in the order of 2%) and the inter-spike interval distributions have their peaks close (48-50 ms) with their distance comparable to the 1 ms sampling rate, although the SpiNNaker distribution has a wider spread than obtained with NEST.

### 5.2.6 Profiling the platform

The tools presented in this and in the previous chapters have been used to profile all the generations of SpiNNaker boards and SpiNNaker chips produced to date. The SpiNNaker system design is carefully balanced to offer a mixture of computation, memory access and communication performance. The architecture's target is the simulation of a billion neurons and a trillion synapses with a firing rate of 10 Hz, over a million cores. At an individual core level, this corresponds to 1000 neurons, each receiving 1000 synapses firing at 10 Hz per core. Under the assumption that the platform is *synaptically* bound and the rest of the processes (including neural updates) are second order, the maximum number of synaptic events (connections activated)  $c_{max}$  per second per core is:

$$c_{max} = 1000 \times 1000 \times 10Hz = 10M/s$$

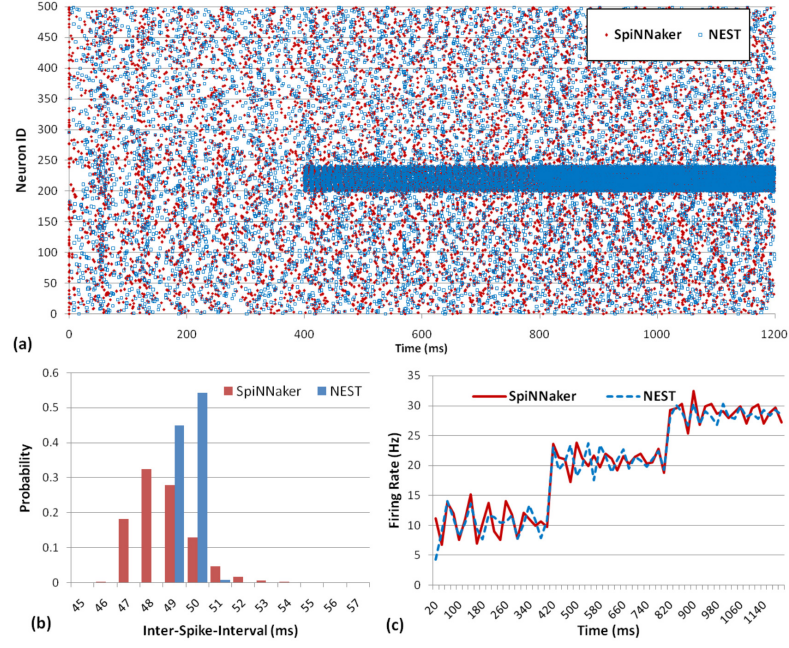


Figure 5.7: Simulation with 500 neurons: comparison between NEST and SpiNNaker. (a) Raster Plot for the entire simulation (b) Inter-Spike-Interval (ISI) Distribution (c) Mean activity rate

or, more generally,

$$c_{max} = n \times s \times fr = 10M/s \quad (5.1)$$

where  $n$  is the number of neurons,  $s$  is the number of synapses each neuron receives and  $fr$  is the mean firing rate.

Considering that each synapse is represented by 4 bytes in SDRAM, the bandwidth requirement can be translated in terms of SDRAM bandwidth (640 MByte/s shared by 16 cores per chip):

$$4 \times 16 \times c_{max} < 640MByte/s$$

The bandwidth effectively measured on SpiNNaker chips is reported in [Patterson et al. \[2012b\]](#), and memory requests can run concurrently to computing cores. On the computational side this corresponds to a core running at 200 MHz, processing 10 M connections/sec if each connection takes less than 20 instructions to be evaluated; this disregards the time needed to update the neural equations and the infrastructural costs to schedule events.

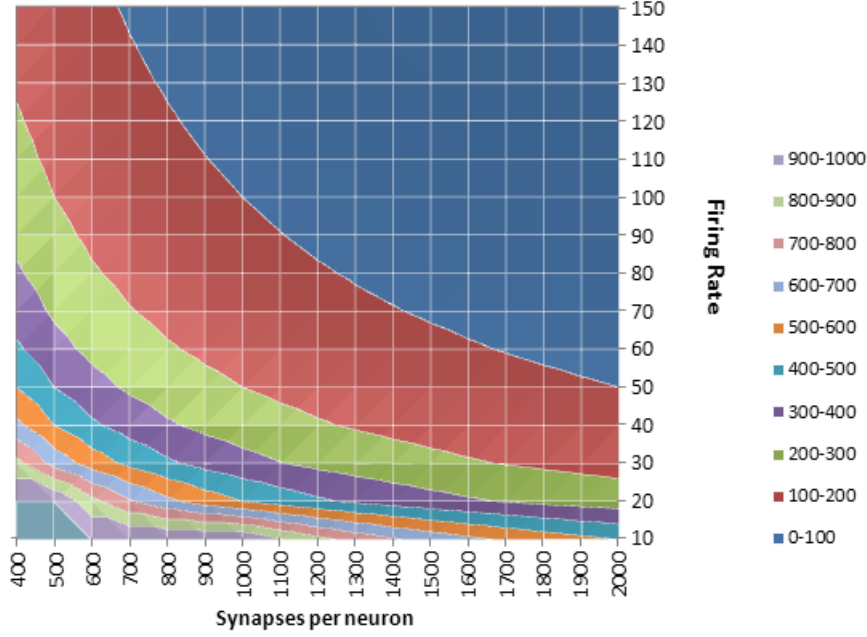


Figure 5.8: How many neurons can be modelled on a single core of a SpiNNaker chip? Theoretical model.

If these are assumed to be the limiting factors for the platform, it is possible to calculate the number of neurons each core can model by introducing 2 variables: the number of synapses per neuron and the (incoming) firing rate. These quantities must obey the following relation:

$$n \times s \times fr = c_{max}$$

or

$$n = \frac{c_{max}}{(s \times fr)}$$

where  $c_{max}$  is the maximum number of connections, calculated from the SDRAM bandwidth above. The relation between these two variables can be plotted, hence determining the maximum number of neurons (see Figure 5.8).

In its original implementation PACMAN was only aware of neural types when splitting populations into partPopulations. Since the number of synapses per neuron is known, and the firing rate is roughly estimable beforehand when a model is designed, this information can be used by the splitter in PACMAN to determine how to split populations and balance load across cores. The relative attributes for the Population object have then been introduced, letting PACMAN split different Populations with different criteria, taking into consideration the estimated number of synapses

that will be activated in a second.

### Test Chip Results

The following two models have been tested on a PCB equipped with 4 SpiNNaker test chips interconnected as shown in Figure 5.9 [Rast et al., 2011c]. Each chip is connected with the others through one of the 6 available physical links (black lines in the figure). Each test chip is equipped with two cores, one for “monitor” functionalities and another for simulating up to 1000 spiking neurons. The tests investigate the behaviour of a real multi-chip SpiNNaker system modelling networks of spiking neurons; for these tests LIF neurons were used Rast et al. [2010a]. To test a scalable network and verify network dynamic with bursts of activity, a synchronous synfire chain model [Abeles, 1982] was implemented. A synfire chain is a feed forward neural network comprising groups of neurons or “pools”, where every pool is connected to the next one in the hierarchy. This mechanism is used to propagate bursts of activity through the network. Four pools comprising 250 neurons each for each chip were simulated in a 4 chip testing environment, for a total of 16 pools and 4000 neurons, as shown in Figure 5.9.

Neuron pools are enumerated from 1 to 16, and every neuron in a pool is connected to the corresponding neuron of the subsequent pool.

The PyNN code instantiating and connecting the populations can be written in a complete parametric way (and hence be scaled to a different number of pools transparently) as follows:

```
for i in range(pools_per_core*xChips*yChips):
    populations.append(Population(pool_size,
                                  IF_curr_exp,
                                  cell_params,
                                  label='pop_%d' % i)
    )
    populations[i].randomInit(uniformDistr)

for i in range(pools_per_core*xChips*yChips-1):
    connections.append(Projection(populations[i],
                                  populations[i+1],
                                  OneToOneConnector(weights=fwd_weights,
                                  delays=delayDistr),
                                  target='excitatory',
                                  label='pop_%d->pop_%d' % (i, i+1)
                                  )
    )
```

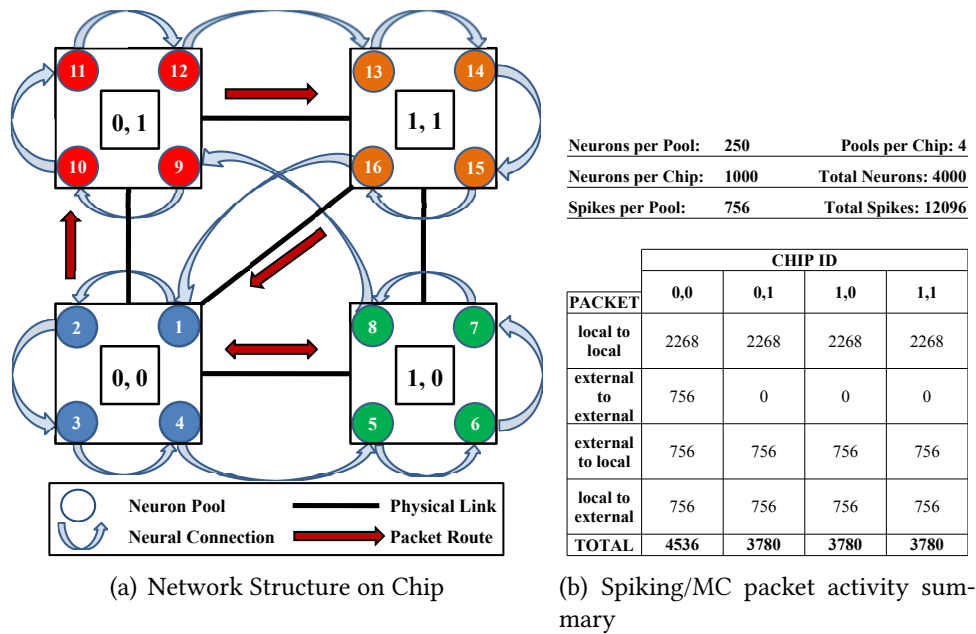


Figure 5.9: Synfire chain test on a test chip board: (a) Interconnection on a 4 chip board with no wrap-around. Black segments identify bidirectional physical links (only 5 links are available on a board with 4 chips and no wrap-around), red arrows identify routes. Neuron pools are enumerated from 1 to 16, and every neuron in a pool is connected to the corresponding neuron of the subsequent pool. Neural connections are depicted by the blue curved arrows; the last pool is connected back to the first providing inhibitory feedback. (b) Simulation metrics.

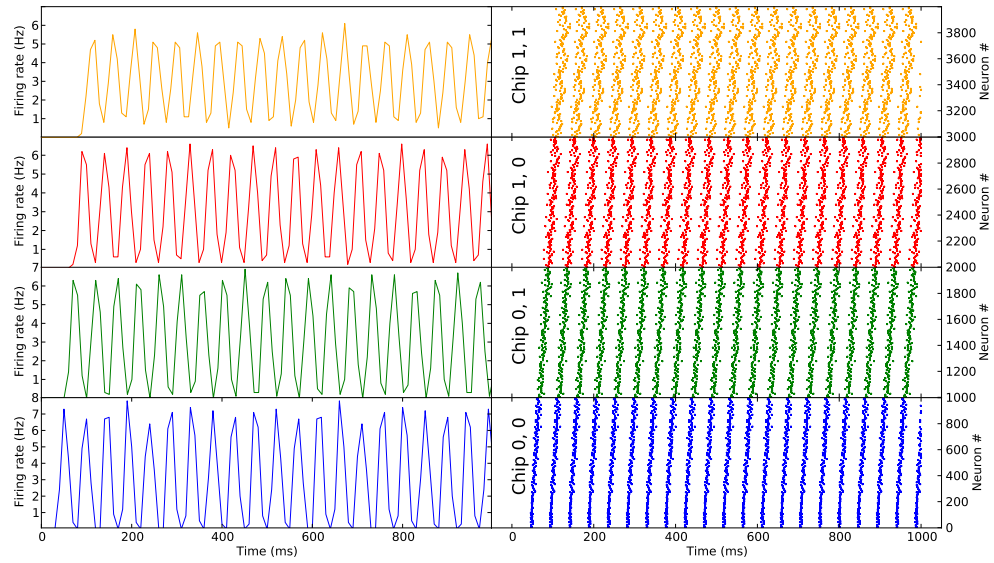


Figure 5.10: Raster Plot from the simulation of the synfire chain on 4 chips. Activity is propagated in a feed forward way in the neural pools simulated on the 4 test chips and it is shown in the raster plot on the right part of the figure, with different colours identifying different chips. The firing rate, averaged per chip, is reported on the left side.

Neural connections are depicted by the blue curved arrows; the last pool is connected back to the first providing inhibitory feedback. Weights are set so that a single spike received will make the neuron fire, hence propagating the activity through the network. These connections are virtual: they exist only at a model level; at the physical level they are transformed into routes through the black physical links. The pre-synaptic and post-synaptic neurons may reside on different chips (e.g. connections from pool #4 and pool #5) or locally if they reside on the same chip (e.g. connections from pool #1 to pool #2). Red straight arrows indicate physical routes. 35 random neurons were simulated in the first population by injecting them with a current strong enough to make them fire at 20 Hz. The activity was then propagated, with random delays in the range 1-8 ms, through the other pools.

Results of the simulation are showed in Figure 5.10, where the raster plots and the mean activity firing rate are divided by chip. The firing rate is averaged across all neurons in the chip (1000), and is hence a population firing rate. Due to the nature of the network structure the activity is bursting, oscillating with peaks of 7 Hz (activity averaged on the whole population with a sliding window of 10 ms). The

spike activity and MC packet count for each chip are shown in Figure 5.9(b). During the simulation every pool emits 756 spikes, for a total of 3024 spikes per chip. Since  $3/4$  of the connections are local,  $3/4$  of the MC packets produced will be consumed locally (local to local packets), while  $1/4$  will be routed to the next chip. Every chip will then send (local to external) and receive (external to local) 756 MC packets. Chip 0, 0 will also route packets from pool #8 (chip 0, 1) to pool #9 (chip 1,0), explaining the value of external to external (transit) packets in the table for chip 0, 0.

### Full chip results, 4-chip board

In order to show the flexibility in programming SpiNNaker and how different models perform on it, a controlled neuron model based on [Sharp et al. \[2011\]](#) has been implemented, where events (rather than neural quantities) are experimental variables. This approach allows control of the dynamics of the experiment and evaluation of the results in a precise way: in particular, the number of packets emitted by every core and the time to process a neural (timer) event can be varied, and the number of spikes and synaptic events can be measured. The neural model is integrated in the mapping approach which configures the number of neurons, synapses and the topology of the network.

Results from 3 experiments, one with many simple neurons firing at low firing rates and a two with more complex neurons and topology and with higher population firing rates are shown in Table 5.2.6. In the simple model each neuron takes 5 instructions per ms to update, while in the complex model it takes 15 instructions. All experiments run in real time.

**Experiment 1 (Figure 5.11):** 16 populations of 100 neurons are interconnected randomly so that all neurons from a population receive input from all neurons of 5 different populations for a total of 800K synapses ( $16 \times 100 \times 100 \times 5$ ). Each population emits 10 spikes per ms, corresponding to a 100Hz mean population firing rate over a population of 100 neurons, producing 160K spikes/s (MC packets/s) ( $10 \text{ spikes} \times 16 \text{ populations} \times 1000 \text{ ms}$ ) and 16M synaptic events/s. Each population is mapped to a core for a total of 16 cores in a single SpiNNaker chip.

**Experiment 2:** the model is scaled up to a 4 chip simulation by increasing the number of populations modelled to 64. This model produces a total of 640K spikes/s and 320M synaptic events/s ( $10 \text{ spikes} \times 64 \text{ populations} \times 1000 \text{ ms}$ ) running on a prototype SpiNNaker 4-chip board.

**Experiment 3 (Figure 5.12):** 16 populations of 1000 neurons are interconnected

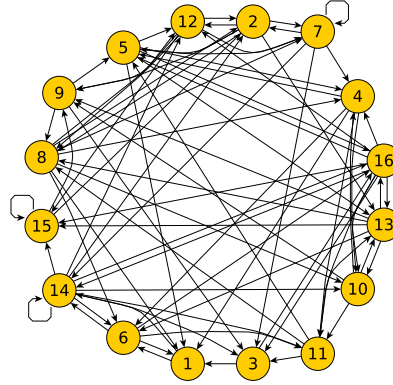


Figure 5.11: Experiment 1: 16 populations of a 100 complex neurons each firing at 100 Hz are fully interconnected to 5 other random populations.

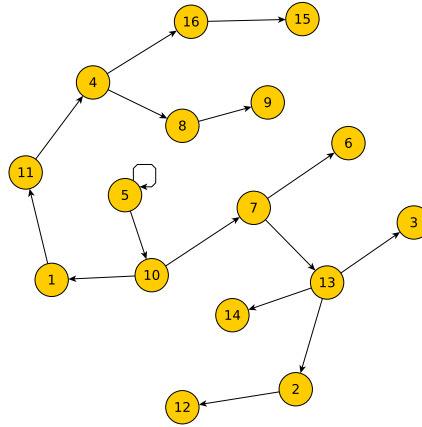


Figure 5.12: Experiment 2: 16 populations of a 1000 simple neurons each firing at 1 Hz are fully interconnected to 1 other random population.

randomly so that all neurons in one population receive input from all neurons of another population, a total of 16M synapses. Each population emits 4 spikes per ms, corresponding to a 4Hz mean population firing rate over a population of 1000 neurons, producing 64K spikes/s (MC packets/s) (4 spikes x 16 populations x 1000 ms) and 64M synaptic events/s ( 256 MB/s transferred from SDRAM). Each population is mapped to a core, for a total of 16 cores in a single SpiNNaker chip.

The experiment proposed analyses the performance of the initial implementation of the system, while exposing its programmability by modelling networks with different characteristics, regimes, dynamics and topologies.

If the results for experiment 1 are compared with the theoretical design limit of 10 M connections/s, than it is evident that overall the simulation exploits 40%

Table 5.2: Profiling results for 4-chip boards

	Exp. 1	Exp. 2	Exp. 3
Neurons per population	1,000	100	100
Populations	16	16	64
Incoming projections/pop.	1	5	5
Synapses/population	$10^6$	50,000	50,000
Total number of neurons	16,000	1,600	6,400
Total number of synapses	$16 \times 10^6$	800,000	$3.2 \times 10^6$
Spikes/population/s	4,000	10,000	10,000
Total spikes/s	64,000	160,000	640,000
Total connections/s	$64 \times 10^6$	$80 \times 10^6$	$320 \times 10^6$
Mapping			
Chips used	1	1	4
Cores used	16	16	64
Processing and communications per core			
Neurons	1,000	100	100
Neuron updates/s	$10^6$	100,000	100,000
Instructions/neuron update	33	93	93
(Neuron up.) instructions/s	$33 \times 10^6$	$9.3 \times 10^6$	$9.3 \times 10^6$
Synapses	$10^6$	50,000	50,000
Connections/s	$4 \times 10^6$	$5 \times 10^6$	$5 \times 10^6$
Instructions/connection	6	6	6
(Connection) instructions/s	$24 \times 10^6$	$30 \times 10^6$	$30 \times 10^6$
Total instructions/s	$57 \times 10^6$	$39.3 \times 10^6$	$39.3 \times 10^6$
Packets processed/s	4,000	50,000	50,000
Energy efficiency			
Power/neuron (nJ/ms)	12	44	45
Energy/connection (nJ)	4	4	4

of the maximum performance out of the machine; it should be noted however that this ideal limit does not take into consideration infrastructural costs (of running the API) and neural updates, but only a budget of 20 assembly instructions for activated connections as a first order approximation.

Power consumption has also been measured during the experiments above, in order to estimate the amount of energy consumed by each component of the system: the infrastructure (idle power, API), the energy/neuron/ms and the energy/connection. As the neural model used to profile does not have any membrane dynamics (or,

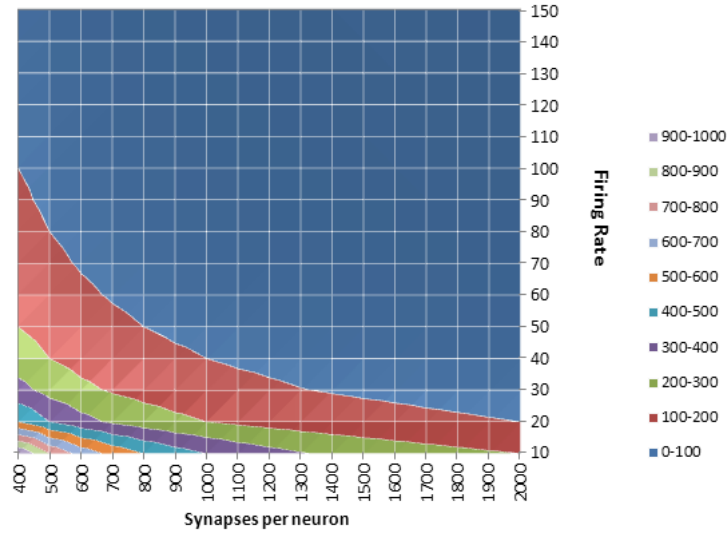


Figure 5.13: Number of neurons that can be modelled by each core, measured limit.

in other words, is not effectively simulating any neural equation) it is straightforward to separate the energy/neuron/ms from the synaptic one, first by running the simulation with all callbacks and events running, and then disabling the sending of packets, hence cutting out all spike-related processing (communication of a spike, lookup in SDRAM, retrieval and computation of synapses). The power used by synaptic events is therefore the differential power between these two simulations. Results for neuron power consumption reflect the complexity of the different models in experiment 1 with respect to experiments 2 & 3, while the connection energy stays constant as the number of instructions per synapse does not change.

#### 48-node board

As new SpiNNaker machines become available, it is important to have software which readily scales up and makes running preliminary tests possible, to catch early problems and have the chance to improve the infrastructure for larger SpiNNaker machines. To test the 48-node board (Figure 5.14), some models with known dynamics (such as the synfire-like model presented in the previous chapter) have been executed on the board. This section presents a model where populations (sized as so to fit a single core) are self-connected with an all-to-all connectivity pattern (with  $n$  neurons in a population having therefore  $n^2$  synapses). Populations (cores) are therefore run independently, as shown by the diagram of interconnections in a single chip presented in Figure 5.15. Weights are set to 0 so that they are processed regularly by

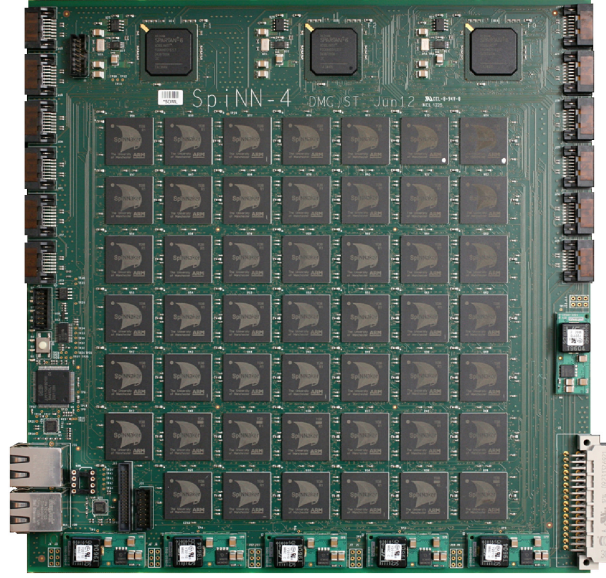


Figure 5.14: 48-node SpiNNaker board ( $10^3$  Machine, 864 cores).

the software on SpiNNaker, but without impacting the neural simulation dynamics. To check that all synaptic processing is taking place in real time, at the beginning of every timer event the number of packets sent (received) is checked against the number of DMA requests processed: as every core is sending packets only to itself, such numbers will match if processing is done in real time. The network comprises LIF neurons injected with a bias current; with the weights set to 0 the network activity is purely a function of the bias current. The results from different models are described in Table 5.3 and show the number of neurons, synapses and synaptic events (connections) for single cores and the whole system. The DMA bandwidth utilised by a single chip is computed under the assumption that every connection requires 4 bytes; performance is then evaluated as a percentage of the SDRAM memory bandwidth compared to the 640 MByte/s theoretical limit; this is equivalent to estimating the performance by comparing the designed number of connections per chip (1000 neurons per core receiving 1000 inputs at 10 Hz = 160 M connections/chip/second) with the ones achieved during the experiments.

The advantages of the hierarchical mapping approach proposed in the previous Chapter become particularly evident when the network size is scaled up: for the third network presented in Table 5.3, instead of placing 65,536 neurons only 768 partPopulations need to be placed; only 768 partProjections, instead of 50 million synapses, need to be routed.

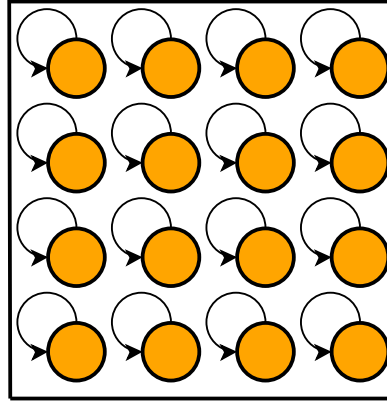


Figure 5.15: Example of a single chip containing independent populations, as the model used to test and profile the 48-node boards.

Table 5.3: Profiling results for 48-chip boards

Neurons per core	100	150	256
Synapses per core	10,000	22,500	65,536
Input (mV)	20	7	1
Population Firing Rate (Hz)	168	100	25
Synaptic events/chip	26,880,000	36,000,000	25,677,824
Neurons (48 chips)	76,800	115,200	196,608
Synapses (48 chips)	7,680,000	17,280,000	50,331,648
Synaptic events (48 chips)	1,290,240,000	1,728,000,000	1,232,535,552
SDRAM bandwidth (MByte/s/chip)	107	144	102
Performance %	16.8	22.5	16

### 5.2.7 Opening SpiNNaker to PyNN models

The continuous introduction of new neuron, network, connectivity and plasticity models, while giving a wide range of research possibilities, poses challenges on how to share and standardize work between different groups and areas. A universal hardware simulator must be able to accommodate such developments. PyNN is emerging as a standard neural network modelling language which can encourage sharing of models with the possibility to run them on hardware.

A PyNN interface to load models on the SpiNNaker system has been developed, and it has been shown how it can be used to handle network translation into specific hardware data files. Likewise, it can manage software and data loading and distribution on a massively parallel machine, even for non-standard models such as

Izhikevich's. Finally it has been used to validate the neural implementation on SpiNNaker. Future work includes implementation and integration of synaptic dynamics with PyNN. One goal is to implement standard neural models and verify them against standard simulators. Another is to be able to create new models which can easily be configured with PyNN. In contrast to previous systems [Brüderle et al., 2009], the SpiNNaker/PyNN combination lets the user configure software at both ends. As new computational units are modelled inside the system, PyNN can be used to organize their relationships and dynamically configure them on board. Users will have the ability to work in a standard environment, producing files that can easily be interpreted and exchanged within the community.

The advantages of using and extending a standard neural modelling tool such as PyNN to configure a universal neural network hardware system such as SpiNNaker have been demonstrated. SpiNNaker, with its innate parallelism and asynchronous packet-based communication system, offers an efficient platform for large neural network modelling. It is a universal platform in the sense that it imposes no particular neural model or neural network topology, but is fully configurable by the end user. The advantage of using general-purpose hardware, coupled with PyNN, is that as new models emerge, both the configuration interface and the simulation code run on the hardware can be extended to accommodate them, increasing the longevity and re-usability of the hardware system and exploiting the extensibility of an open-source standard modelling language.

### 5.3 Neural Engineering Framework on SpiNNaker

Construction of large-scale spiking neural models is possible thanks to the emergence of unified approaches that are able to scale up seamlessly. In this section the method to map the principles of the Neural Engineering Framework (NEF) [Eliasmith and Anderson, 2003] to the SpiNNaker System is described, exploiting the massively parallelism of the platform and its programmable architecture oriented to the simulation of large scale models of spiking neural networks. The NEF is a unified approach for implementing complex neuro-dynamical systems and mapping control-theoretic algorithms with the neural connections between a highly heterogeneous population of spiking neurons.

This work describes the approach taken to encode and decode values directly

on-chip, taking advantage of the programmability of the SpiNNaker system and exploiting the fast on-chip spike-based interconnect for communication between neural populations.

A variety of networks that can be built using encoding/decoding methods are shown. The approach presents the basis for testing large-scale neural models built with the NEF integrating SpiNNaker as the computational back-end in the existing framework and tools.

### 5.3.1 Encoding/Decoding approach

The NEF allows representation and computation of values and functions entirely in the neural space, once the values are encoded/decoded using the framework. Hence it is possible to build a system that communicates only with spikes, by inputting and collecting them on a host machine which is responsible for the encoding and decoding process. This approach however, while being very efficient for spike/AER based systems [Lazzaro et al., 1993], does not scale up seamlessly as the size or firing rates (and consequently the number of spikes needed to be sent from/to the system) of the encoding and decoding populations increase. Moreover the computational cost increases with the number of neurons: for example 3,000,000 neurons running on two Quad Core Intel Xeon E5540 processors with Hyper-Threading at 2.53 GHz take 3 hours to produce 1 second's worth of data; simulating it within a GPU environment leads to a 20x speed-up (6-9 minutes per simulated second).

To address the computational time costs, the programmability of the ARM968 cores constituting the computational heart of the SpiNNaker system is exploited by implementing the encoding and decoding process directly on the SpiNNaker chip, through the use of two *ad-hoc* populations based on the leaky integrate-and-fire (LIF) neuron: the *NEF/LIF-encoder* and the *NEF-decoder* populations. The former translate values into spike trains for neurons in the population according to principle 1 (see Section 2.3.3), while the latter collects spikes and estimates the value using the decoders for the source population. In other words, the *NEF/LIF-encoder* population is implementing neurons that obey Equation (2.12) where the neural dynamics obey the standard LIF equation  $dV/dt = I/C - V/RC$ , while the NEF-decoder population is decoding values using Equation (2.13). This kind of neural population is only used when the value needs to be explicitly represented, otherwise decoders are implicit in the connection weights, since communication between all other populations on the chip is done using spikes and standard LIF neurons. Those are weighted using

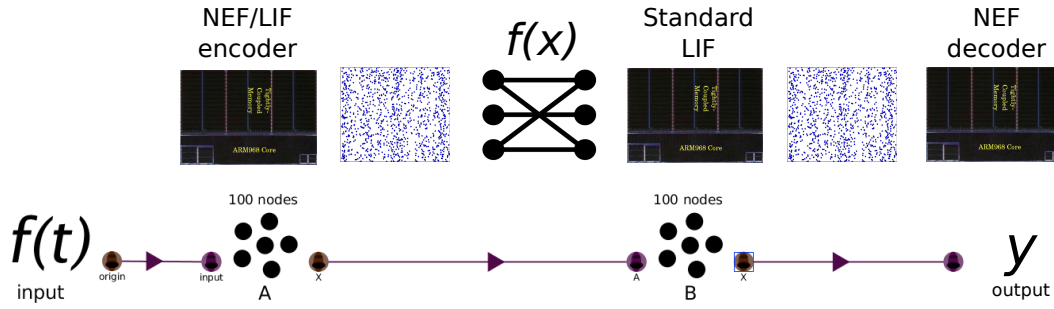


Figure 5.16: Approach — Encoding and decoding processes happen directly on the SpiNNaker chip, through the use of two *ad-hoc* populations: the *LIF/NEF-encoder* and the *NEF-decoder* populations. The former translates values into spike trains for neurons in the population accordingly to Equation (2.12), while the latter collects spikes and estimates the output value using the decoders for the population connected to it, following Equation (2.13). On the rest of the chip, spikes travel in the neural space (consisting of standard LIF neurons) where the weights implement a function  $f(x)$  and are calculated starting from Equation (2.15)

NEF connection weights computed as described in Equation (2.15). The approach is summarised in Figure 5.16. Compared to a standard LIF model it adds a bias current proportional to the encoder and stimulus values.

Precision in encoders and decoders (and therefore in interconnection weights) is crucial to avoid information corruption. Digital systems have the advantage of being programmable with a finite precision that can be evaluated. In this work we use a 32 bit fixed point implementation for neural state variables and parameters, including encoders and decoders, and 20 bit precision for the weights.

Spikes are therefore produced and collected only on-board, by taking advantage of the fast custom interconnect characterizing the SpiNNaker machine. This reduces the bandwidth and the load on a host, by sending and receiving only values to/from SpiNNaker. This approach tends to avoid difficulties and bottlenecks in translating and sending/receiving spikes directly from a host machine or from an FPGA by porting the encoding/decoding process onto SpiNNaker. Moreover it offers a "closed" spike system, where the interface communication consists in sending and receiving values for example from Nengo [Stewart et al., 2009] (the software<sup>1</sup> that implements the NEF principles), or from a sensor or to a robotic arm while neural based computation is carried out on board.

<sup>1</sup>available at <http://nengo.ca>

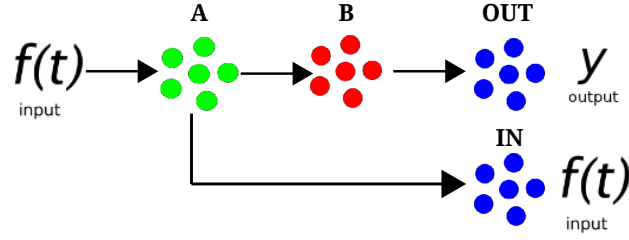


Figure 5.17: Structure of the communication channel – the input value is encoded by population A (green, encoding population), which translates it into a population spike train. A is connected to B (standard LIF population) with an all to all connection. Weights between A and B are set to compute  $y = x$  in the communication channel experiment and  $y = x^2$  in the transformation experiment.

### 5.3.2 Results

#### Representation: Communication Channel

In order to test the representation of values in spike trains, we have implemented a communication channel with the structure illustrated in Figure 5.17. The communication channel experiment shows how information can be represented using the NEF, as so to be able to encode/decode information directly within the SpiNNaker System. This is done by a population which encodes a scalar value into neural activity and then decodes it with another population, equipped with *representational decoders* able to extract the original value. Such encoders/decoders are used to compute the function  $f(x)=x$  in the connection weights between the two populations (see Figure 5.16).

A portion of the Nengo script describing the communication channel is shown below while the full script can be found in Appendix C.2.2:

```
input=net.make_input('input',[0])

A=net.make('A', neurons=128, dimensions=1, max_rate=(100,150),
           radius=1, intercept=(-.9,.9))
B=net.make('B', neurons=128, dimensions=1, max_rate=(100,150),
           radius=1, intercept=(-.9,.9))

net.connect(input,A)
net.connect(A,B)
```

Population A comprises 150 *LIF-encoder* neurons. Population B is a standard LIF population of 150 neurons. *IN* and *OUT* are populations of 150 NEF-Decoder neurons each. Therefore the whole communication channel is composed of 450 neurons

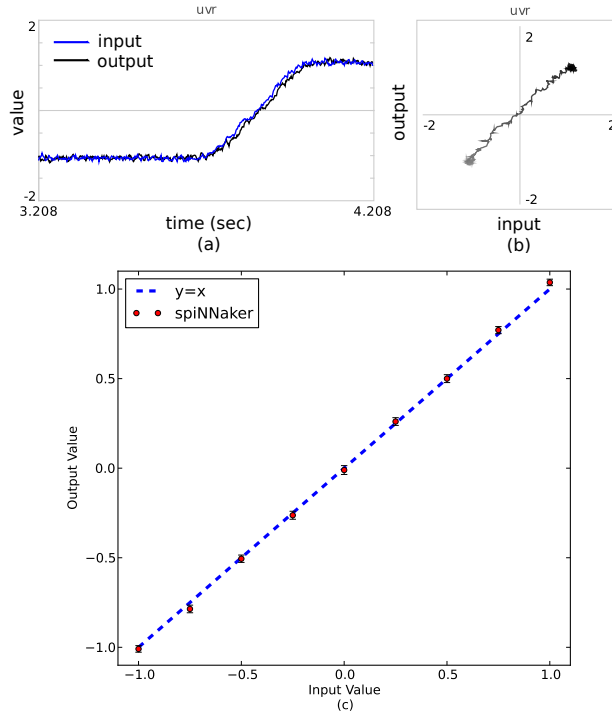


Figure 5.18: Representation Principle — Communication channel: weights between A and B are set so that B represents the same value as A. The value is then decoded by population OUT (decoding population) and compared against IN to verify correct encoding.

firing in the 40-100 Hz range. Each Population occupies a dedicated core within the same chip. A value  $X$  is encoded by population A, the NEF-LIF encoder population. This population encodes values in spike trains according to the tuning curves of its neurons.

Spike trains travel to population B through an all to all connection with weights implementing the communication channel function  $y=x$ . Spike trains are then passed to the OUT population that converts them back into a value  $Y$  and outputs it to the external world. In particular it is possible to integrate SpiNNaker as a back-end simulator in Nengo by communicating with ethernet attached chips. Results are shown in Figure 5.18 where the precision of the encoding is evaluated and integration with Nengo is shown. All the experiments presented run in real time.

### Transformation: Computing the Square

In order to show computation we implemented the function  $y = x^2$  in the NEF, using the same network structure as that for the communication channel. Only the

decoders of (and subsequently the connection weights to) population B are changed, implementing the function to be computed. Such decoders are called *transformational decoders* in the sense that they compute a (feed-forward) transformation, as opposed to the *representational decoders* used for the communication channel experiment. The Nengo script describing the square has been reported in Section 3.2.5 and can be found in Appendix C.2.1.

Results are shown in Figure 5.19: the blue line represents the direct decoding from the input sent to SpiNNaker. The black line is the decoded result of the operation implemented in the weights from A to B. When the input is 0 (leftmost plot) the output from SpiNNaker is 0 as well (black line). When the input is shifted to 1 both input (blue) and output go to 1. When the input is shifted to -1 the result of squaring stays at 1. The quadratic relation is particularly evident when the input is plotted against the output, as done in Figure 5.19(b) and 5.19(c). The network consists of 450 neurons as in the Communication Channel example; it is the same network with the weights between A and B changed so as to compute the square, by estimating transformational decoders for B.

### Dynamics: Integrator

In order to show an implementation of neural dynamics within the NEF we implemented a neural integrator. Such a mechanism has been proposed as the neuronal basis of oculomotor control [Fukushima et al., 1992] where it is used as a velocity to position integrator: the input of the system represents the eye movement velocity which is integrated to represent the final eye position. More generally the integrator can be considered to be a line attractor in the higher dimensional neural state space, letting the network maintain a value (representing for instance the eye position) through self-sustained activity in an abstract space over a period of time. The integrator has also been proposed to be the basis of working memory in neurons [Singh and Eliasmith, 2006].

The third principle can be used, as described in Section 2.3.3, to translate the dynamics of an integrator defined by standard control theory input and dynamics matrices [Eliasmith and Anderson, 2003]. Using Equations 2.14, by setting  $\mathbf{A} = 0$  and  $\mathbf{B} = 1$  (as in standard control theory  $\mathbf{A} = 0$  and  $\mathbf{B} = 1$  correspond to a linear attractor) we obtain  $\mathbf{A}' = 1$  and  $\mathbf{B}' = \tau$ , where  $\tau$  is the synaptic time constant. We can then compute the neural connection weights using eq. 2.15. The integrator structure is shown in Figure 5.20: an input is fed into population A, and it travels

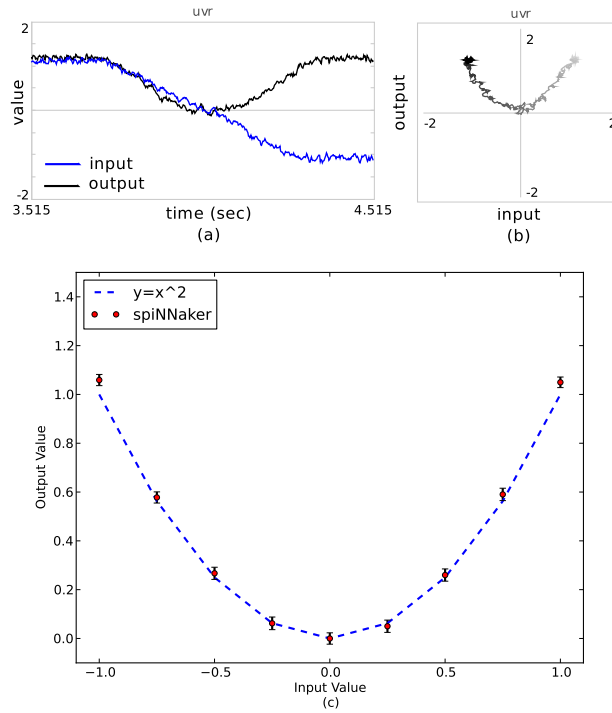


Figure 5.19: Transformation Principle — Computing the Square: (a) The blue line represents the direct decoding from the input sent to SpiNNaker and represents the ability of SpiNNaker to encode and decode input. The black line is the decoded result of the square operation implemented in the weights from A to B. (b) The quadratic relation can also be observed when the input is plotted against the output within Nengo. (c) Precision evaluation of the square computation in the range of interest. Each value is sampled for 1 second and activity is averaged and standard deviation is shown.

through a communication channel to population B. Population B then computes the integral of the input received by A by means of recurrent connections. The Input population comprises 150 NEF-encoder neurons firing at 80-100 Hz. The integrator is composed of 200 neurons fully recurrently connected (40000 connections) firing at 80-150 Hz. Weights for these connections are computed using the encoders and decoders as described above. For simplicity, the weights are imported directly from Nengo and loaded on board. Population A is an encoding population as described in the section above.

A portion of the Nengo script describing the communication channel is shown below while the full script can be found in Appendix C.2.2:

```
input=net.make_input('input',[0])

tau_syn = .05
A=net.make('A', neurons=100, dimensions=1,
           max_rate=(120,150), intercept=(-.90,.90))
B=net.make('B', neurons=100, dimensions=1,
           max_rate=(120,150), intercept=(-.90,.90))
C=net.make('C', neurons=100, dimensions=1,
           max_rate=(120,150), intercept=(-.90,.90))

net.connect(input,A,pstc=.001)
net.connect(A,B,pstc=tau_syn, weight=tau_syn)
net.connect(B,B,pstc=tau_syn)
net.connect(B,C)
```

Results from a simulation run, sending non-encoded input values to SpiNNaker and getting decoded output back, is displayed in Figure 5.20: population B integrates the positive pulse represented by input population A and holds the integrated value. Then a negative pulse input is received and integrated. Between the two pulses the integrator is able to hold the value with little drifting. The difference in the response of the neural integrator to the ideal one is due to the fact that the neural integrator has a PSC filter applied, and therefore the response passes through a first-order filter whose time constant is equivalent to the synaptic time constant.

### Cyclic attractors: Oscillator

All models presented so far use a scalar representation for encoders, decoders and transformation. It is possible to extend the representational methods to vectors in an n-dimensional space by choosing neurons with preferred direction vectors in the space, and hence employ n-dimensional encoders and decoders and transformational

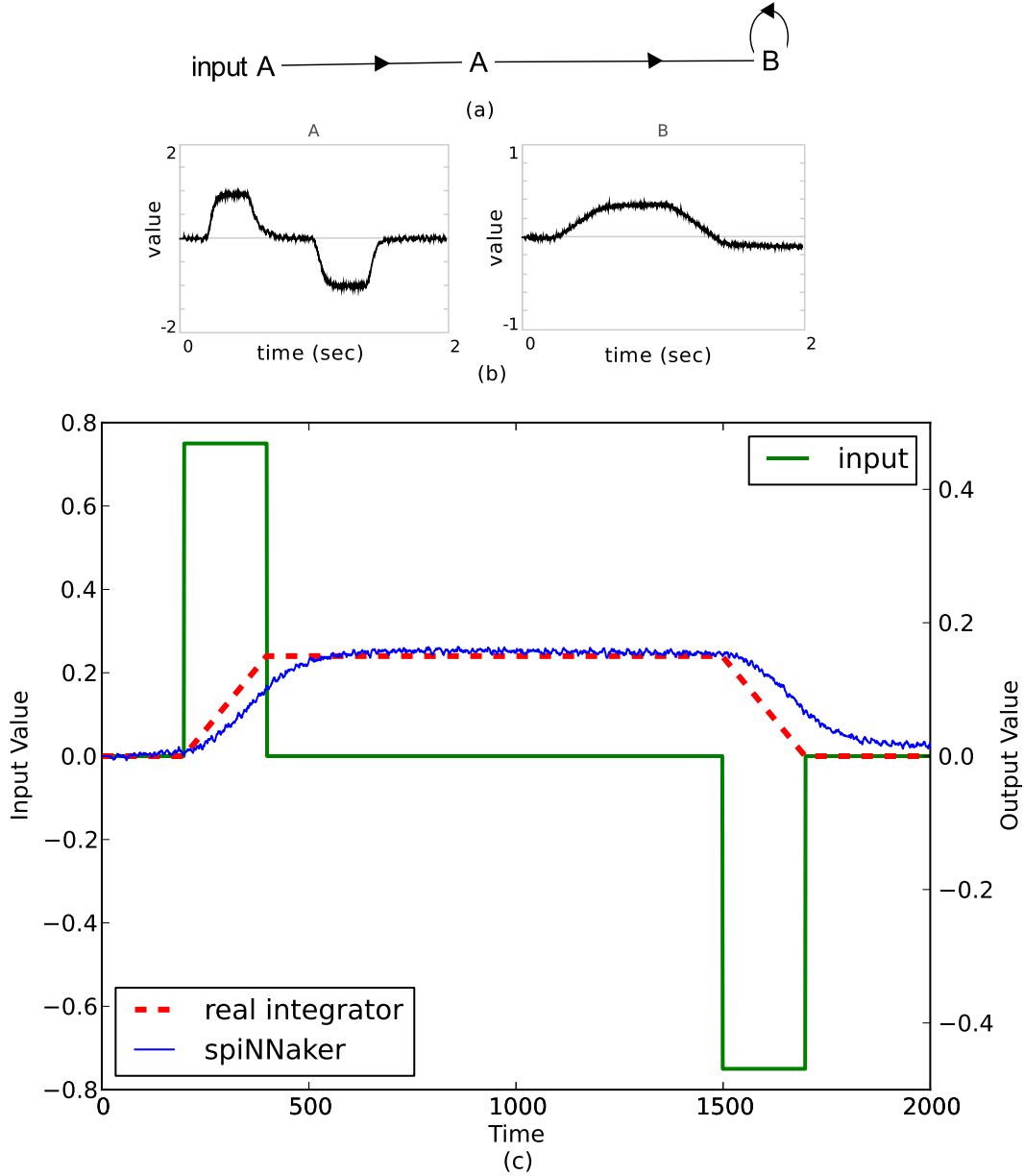


Figure 5.20: Dynamics — Neural Integrator: (a) Structure of the integrator network: An input is fed into population A, which travels through a communication channel to population B. Population B then computes the integral of the input received by A by means of recurring connection. The Input population is composed by 150 NEF-encoder neurons firing at 80-100 Hz. The integrator is composed by 200 neurons fully recurrently connected (40000 connections) firing at 80-150 Hz. (b) Integration in Nengo. (c) Integration of the input value compared to an ideal integrator.

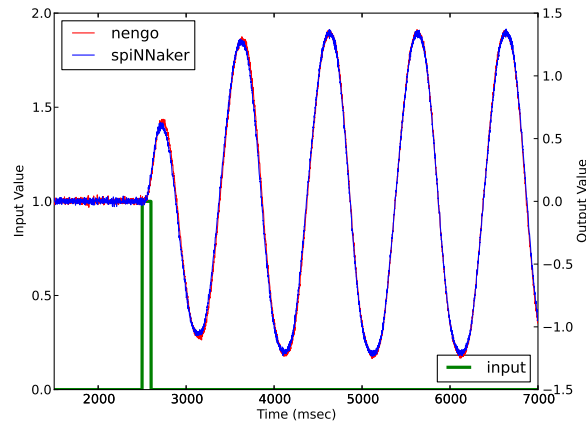


Figure 5.21: Cyclic Attractor — Oscillator: response of a neural oscillator to a perturbation; after the system is perturbed it starts oscillating at its characteristic frequency in Nengo and SpiNNaker. Only one dimension of the oscillating population is represented.

matrices for dynamical systems. It is then possible to evaluate neurobiological evidence to choose the appropriate kind of representation. By doing this it is possible to build another class of attractor: cyclic attractors. Rather than stabilising on a fixed point, line or plane, cyclic attractors settle on a periodic pattern of activity that is dynamically stable. Such attractors can be used to explain repetitive behaviours like walking, flying, chewing or swimming; in particular a (more complex) cyclic attractor built accordingly to the NEF principles has been used as the basis to model swimming behaviour in the lamprey eel [Eliasmith and Anderson, 2003]. A cyclic attractor can be defined using the input and dynamics matrices introduced in Section 2.3.3 with the control equation  $\dot{x} = Ax + Bu$  where

$$A = \begin{bmatrix} 0 & \omega \\ -\omega & 0 \end{bmatrix}$$

implementing a cyclic attractor with a harmonic oscillator. The results are shown in Figure 5.21: after the system is 'shocked' it starts oscillating at its characteristic frequency  $\omega$ .

Using the framework it is possible to manipulate the parameters that control attractor properties by modifying the transformational matrix  $M^{\alpha\beta}$  (see Section 2.3.3) using a control signal represented by another population: if the signal is a function of time the system described is a linear time-varying system; if A is an input then

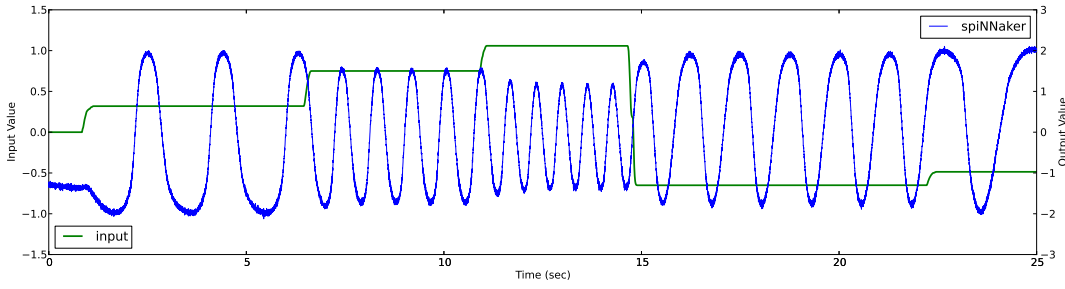


Figure 5.22: Non-linear system: Frequency Controlled Oscillator. The oscillator has its orbit period (frequency of the oscillation) controlled by another input neural population. Only one dimension of the oscillating population is represented.

the system becomes non linear. In this example we use a population to control the frequency of the oscillator, therefore controlling the speed of the cyclic attractor (e.g. controlling the swimming speed in the zebra fish [Kuo and Eliasmith, 2005]) by increasing the dimensionality of the space encoded by the population to accommodate a frequency control input. Results are shown in Figure 5.22. In both experiments inputs and oscillating populations comprise 150 neurons each firing in the 80-120 Hz range.

### 5.3.3 The Neural Engineering Framework on SpiNNaker

The work presented here constitutes the basis for building real-time, large scale neural systems using the Neural Engineering Framework on SpiNNaker. The programmability of the ARM cores makes the integration of SpiNNaker with the existing tools and software possible in a seamless way. It should be noted that the advantages of running large-scale models in real-time are strongly reduced if such models take a long time to compile and load on a computational back-end. In fact experiments have shown that the most computationally expensive task is to compute the weight connection matrices (described by eq. 2.15) and map them on a parallel system such as SpiNNaker. However, it is possible to exploit the possibility to program SpiNNaker cores to parallelize this job and run it on board. Once the neural populations are mapped to specific cores, connections can be generated by just sending them encoders, decoders and transformational matrices, having each core do the matrix multiplications, self-configuring and indexing its own local connections. It has been possible to map multidimensional encoders and decoders in the

weight space to represent transformations in the neural space; it is however possible to implement *n*-dimensional *NEF/LIF-encoder* and *NEF-decoder* populations. Such high dimensional inputs can be used to manipulate complex, symbol-like structures such as language [Stewart et al., 2010b, 2011]. Such high dimensional spaces can be mapped to large scale populations of neurons so as to represent a large variety of symbols, each neuron mapping a fraction of the high dimensional space. Running such models in real time makes it possible to test them and embed them in real world, interactive scenarios, such as decision making [Stewart and Eliasmith, 2010] or other cognitive functions such as working memory, recognition and fluid reasoning [Eliasmith, 2007].

This section has presented how to construct neural circuits using the Neural Engineering Framework on the SpiNNaker hardware. Encoding and decoding values using the NEF is implemented directly on board to perform feed forward and recurrent dynamic computations. This approach takes advantages of the programmability of the ARM968 cores inside a SpiNNaker chip, letting it encode and decode spikes on-board. This reduces the bandwidth and the computational load imposed on a host machine, by sending and receiving only values to/from SpiNNaker and using it as a fast, scalable, configurable and power efficient computational back-end. This approach offers advantages when the firing rates and the dimensions of the input and output neurons increase, letting the system scale up seamlessly and be integrated in interactive real-time systems. It also presents the basis for building and testing large-scale neural models built with the Neural Engineering Framework on the SpiNNaker architecture.

## 5.4 Summary

This chapter shows how the mapping approach, presented in the previous chapter, can be interfaced with neural languages already present in the computational neuroscience community, making the platform accessible to non-hardware experts.

Integration with PyNN enables users to write their models with a cross-simulator, standard language, shareable in the community. A variety of neural models have been integrated in the approach, exposing the flexibility of both SpiNNaker and the mapping process. Larger and increasingly complex PyNN models have been simulated on different generations of SpiNNaker platform; some results, validating the

implementation and testing the performances of the architecture, have been presented in this chapter.

Integration with Nengo enables the Neural Engineering Framework methods to be used on SpiNNaker to determine the topology and parameters for a network model. The implementation on SpiNNaker of the NEF principles enables a high-level, functional specification of a spiking neural network to be deployed on the platform. The method exploits SpiNNaker's reconfigurability to encode and decode the information from values into spike trains directly on board, making the system a *neural black box* which interacts with the external world (e.g. with Nengo or with a robot) through values rather than spike trains: all the neural computation is performed by SpiNNaker which also decodes output spike trains into values.

This chapter described the flexibility of the approach, which is able rapidly to incorporate new hardware resources, neural models and computational frameworks. The next chapter shows how it is possible to enable SpiNNaker to be embedded in the real world, through interfaces with AER sensors, already present in the community, and with a robotic platform. These resources are integrated in the mapping approach and can be configured seamlessly at a high level, through the PyNN or Nengo interface.

# Chapter 6

## Mapping AER sensors and robotic actuators

### 6.1 Introduction

The previous chapter described how PACMAN, a flexible translation method presented in Chapter 4, is used to interface existing neural-oriented languages with SpiNNaker, automating model placement and abstracting the platform to a final user. This chapter describes the expansion of PACMAN to integrate other elements present in the neuromorphic community: AER sensors and robots. Computation, all done in spiking neurons, takes advantage of the abstraction of action potentials into streams of stereotypical events which encode information in their timing. This feature has been used by neuromorphic engineers to reduce both power consumption and communication bottlenecks. As a consequence, a number of spiking custom mixed-signal address event representation (AER) chips have been developed in recent years, such as the ones described in Section 3.3.

In this work AER sensors and robots become resources available at model level, while the system is configured automatically using PACMAN. The approach is used to build complex networks in PyNN and Nengo; those presented in this chapter integrate SpiNNaker, a silicon retina and a holonomic robot. The integration of such systems provides non-hardware experts a standard interface to bring their models to the platform, ready for exploration of networked computation principles and applications at different levels of abstraction.

The first part of the chapter illustrates a visual attentional model and the integration with a DVS silicon retina. The retina is accessible through PyNN, which is also

used to describe the rest of the network model, performing different steps of visual processing and orienting its response to the location where a preferred stimulus is detected.

The rest of the chapter describes a model comprising different Neural Engineering models for self-localization and mapping (SLAM) and navigation integrated with a robotic platform. Increasingly complex models are proposed, and they culminate in a cortical, basal ganglia and hippocampal place cell model, based on the work of [Blair et al. \[2008\]](#), achieving autonomous navigation and mapping of different places.

## 6.2 Visual attentional model using a silicon retina

The neural processes which subserve attentional selection, and subsequently drive intelligent behaviour in animals, remain the subject of intense investigation within the field of computational neuroscience [[Knudsen, 2007](#)]. Precise spike timing is understood to play an important role in biological neural computation, for example, spike timing precision has been hypothesised to maximise the information transfer rate [[Van Rullen and Thorpe, 2001](#)]. Event-driven platforms provide a natural architecture on which to simulate spiking neural networks. Traditionally, the control flow of a program is time-driven, while the flow in event-driven systems is dependent on the occurrence of events and time is represented implicitly. Recently, sensors [[Lichtsteiner et al., 2008](#), [Lenero-Bardallo et al., 2011](#)], mixed signal VLSI chips [[Sonnleithner and Indiveri, 2011](#), [Indiveri et al., 2011](#), [Wijekoon and Dudek, 2012](#)] and parallel architectures [[Furber and Brown, 2009](#)] that are natively based on events have been developed. Such systems use Address-Event-Representation (AER), a lightweight protocol for asynchronous communication of spike events [[Serrano-Gotarredona et al., 2009](#)].

In this context a configurable event-driven platform, comprising an AER visual sensor [[Lenero-Bardallo et al., 2011](#)] and the SpiNNaker system is presented. A significant feature of this system is that a direct connection (via an FPGA) between the AER sensor and the event-driven SpiNNaker system allows timing information in the spike-train to be preserved: spikes (events) are processed as they arrive, eliminating delays caused by both buffering and by the use of a host machine as a protocol translator [[Davies et al., 2010](#)]. While the SpiNNaker system offers a flexible event-driven platform for the real-time exploration of neural networks, which natively conform with AER, SpiNNaker also allows neural network dynamics and topologies to

be rapidly reconfigured using PyNN as a high level language [Davison et al., 2008, Galluppi et al., 2012d].

An attentional selection system was implemented upon this neuromorphic platform. Visual cues consisted of oriented Gaussians, which were represented as locations on a retinotopic visual saliency map [Itti et al., 1998]; such features are the most basic representation in the hierarchical structure which subserves object recognition in the visual cortex [Riesenhuber and Poggio, 1999]. The mechanisms of selection were inspired by the biased competition hypothesis of Duncan and Humphries [Duncan and Humphreys, 1989], in which competition takes place between different representations of potential targets and goal-directed information from a particular working memory can be used to bias selection towards a particular target. Goal-directed signals were feature-based, whereby the activity of a working memory neuron represented the behavioural importance of attending to a particular visual feature. This work [Galluppi et al., 2012a] represents the first neuromorphic implementation of a feature-based selection system. Spatial bias has previously been implemented by injecting activity into a particular location on the visual saliency map [Sonnleithner and Indiveri, 2011].

The hardware setup is described in the next section, and experimental results are presented in the results section; discussion and conclusions are presented in the final section.

### 6.2.1 Hardware setup

An image of the event-driven platform hardware is shown in Figure 6.1.

**Silicon Retina:** the visual front-end is a Dynamic Video Sensor (DVS) silicon retina, an asynchronous sensor which provides spike events encoding the addresses of pixels undergoing a contrast change [Lenero-Bardallo et al., 2011]. This approach contrasts to the more traditional method of sending entire frames and can provide fast (3  $\mu$ s latency) data-driven contrast detection at a wide range of illuminations. The sensor is capable of transmitting from 1 Keps to 20 Meps (events per second).

**SpiNNaker System:** neural processing is carried by 4 SpiNNaker chips, offering an event-driven digital platform that can interpret incoming events as neural spikes and inject them into the neural system. Each SpiNNaker chip natively responds to events occurring in the network, and is therefore able to process information arriving from event-based sensors attached to its asynchronous links, provided the AER protocol is translated correctly.

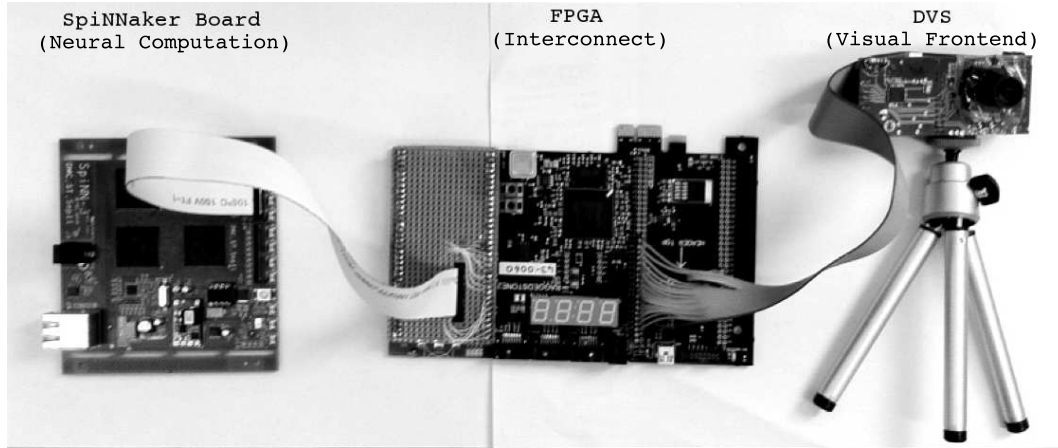


Figure 6.1: Overview of the hardware setup: 4 SpiNNaker chips board (left), an FPGA translating the AER protocol between the two asynchronous systems (middle) and a DVS Silicon Retina (right).

**Interconnection:** retinal events are translated into asynchronous spike trains in the neural network: as soon as an event is emitted from the silicon retina, it is translated into a SpiNNaker spike by a Xilinx SPARTAN-6 FPGA and injected into the system directly via the fast on-board interconnect, using one of the six asynchronous links available on each SpiNNaker chip.

**Configuration:** two levels of virtualisation allow a transparent mapping between the silicon retina and the SpiNNaker system. From a hardware point of view the DVS is mapped as a virtual SpiNNaker chip, injecting spike events into the network-on-chip. To the neural network, the silicon retina is represented as a *Spike-Source Population* by the mapping approach introduced in Section 4.2.1. As with the rest of the neural network, the SpiNNaker system allows configurable *Projections* to this neural population using a high-level language such as PyNN (see sec. 3.2.7).

## 6.2.2 Network Description

Figure 6.2 illustrates the implemented network, developed conjunctively with Kevin Brohan [Brohan et al., 2010], which was inspired by the primate visual system: shapes indicate preferred orientation (black neurons do not encode orientation information). Complete projections from only the palest neurons are shown for clarity. Neurons are represented by triangles, arrows represent excitatory synapses and

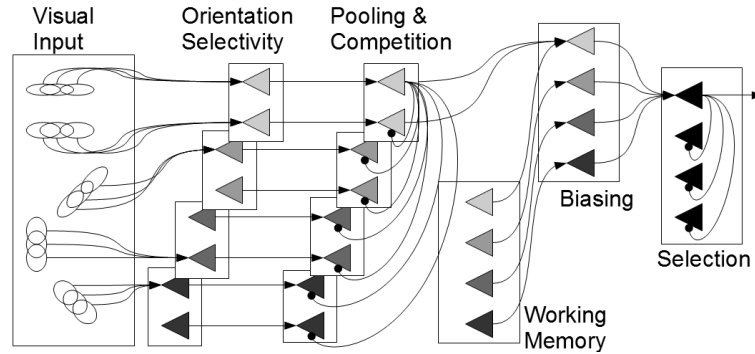


Figure 6.2: Overview of the neural network for the visual attention model. In the first layer, neuron responses are selective to stimulus orientation, in a similar manner to neuronal responses in cortical area V1. The *pooling & competition* layer subsampled the activity of neurons with the same preferred orientation (implementing a *localmax* function, in a manner similar to V2 complex cells). Four neuronal populations in the *working memory* layer encoded the goal of selecting a stimulus with a particular orientation.

circles represent inhibitory synapses. In the first layer, neuron responses were selective to stimulus orientation, in a similar manner to neuronal responses in cortical area V1 [Hubel, 1988]. The *pooling & competition* layer subsampled the activity of neurons with the same preferred orientation (implementing a *localmax* function, in a manner similar to V2 complex cells [Riesenhuber and Poggio, 1999]), while local competition between neurons with different preferred orientations sustained the activity of neurons whose preferred orientation matched the stimulus, and suppressed activity relating to non-matching neuronal responses. Four neuronal populations in the *working memory* layer encoded the goal of selecting a stimulus with a particular orientation. Activity in this layer was determined by a set of external biasing currents. In primates, visual search goals are believed to be encoded in the activity of the dorso-lateral prefrontal cortex neurons [Knudsen, 2007]. The *biasing* layer received combined activity from the *pooling & competition* layer and the *working memory* layer, such that activity was maximised for stimuli of the desired orientation, provided they are also present in the visual field. These neurons are analogous to the neurons of cortical area V4, which receives a large input from working memory via the frontal eye fields [Noudoost et al., 2010]. Activity was pooled across all orientation maps in the *selection* layer to form a retinotopic visual saliency map (corresponding to the lateral intraparietal cortex, LIP area [Bisley et al., 2011]), and

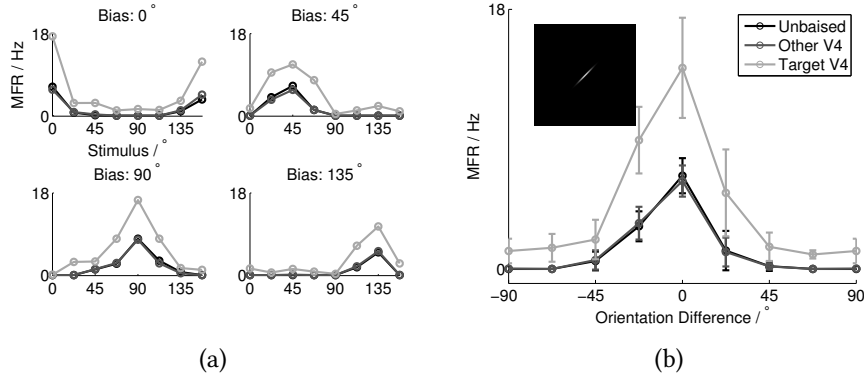


Figure 6.3: Experiment I: tuning curves.

competition between different retinotopic locations in this layer resulted in the selection of a single target. Activity at this location (i.e. the target of attention) was maintained, while activity at other locations was suppressed. Attentional effects themselves were not included in this model.

The network was implemented with 1221 leaky integrate-and-fire neurons and 20530 current-based exponential synapses on a single SpiNNaker chip, and 128x128x2 DVS neurons that were binned to 16x16x2 spatial locations on the FPGA, while preserving the number of events received by SpiNNaker. Neurons were arranged in a square topology (14x14 in *visual input*, 10x10x4 in *orientation selectivity*, 10x10x4 in *pooling & competition*, 5x5x4 in *working memory*, 5x5x4 in *biasing*, 5x5 in *selection*). Both excitatory and inhibitory projections were allowed from individual neurons, though this is not observed in biology.

An excerpt of the PyNN script configuring the network can be found below, while the rest is contained in Appendix C.1.4.

Two maps, describing the two different polarities modelled by the retina, are instantiated:

```
print "%g - Creating input population"
input_pol_1 = Population(input_size*input_size,          # size
                        SpikeSource,                     # Neuron Type
                        {},                               # Neuron Parameters
                        label="input_pol_1")              # Label
input_pol_1.set_mapping_constraint({'x':2, 'y':0, 'p':0})
input_pol_1.record()

input_pol_2 = Population(input_size*input_size,          # size
                        SpikeSource,                     # Neuron Type
```

```

        {}, # Neuron Parameters
        label="input_pol_1") # Label
input_pol_2.set_mapping_constraint({'x':2, 'y':0, 'p':8})
#input_pol_2.record()

```

Then separate populations mapping different orientations for the *Orientation Selectivity layer* (V1), the *Pooling & Competition layer* (V2), and the *Biasing layer* (V4) are created with different sizes:

```

for i in range(orientations): # Cycles orientations
    # creates a population for each connection
    v1_pop.append(Population(v1_pop_size*v1_pop_size, # size
        IF_curr_exp, # Neuron Type
        cell_params, # Neuron Parameters
        label="v1_%d" % i)) # Label)

```

The *Working memory* (PFC) and *Selection* (LIP) populations are created analogously. Then the two input retinal populations are connected to V1 through means of a convolutional network, using the kernels obtained with the *gaussianConnectorList* function and creating an explicit list of weights representing the convolution with the *Filter2DConnector* function.

```

projections = [] # Connection Handler
gaussian_filters = gaussianConnectorList(scales, orientations, sizeg1)

for i in range(orientations):
    # creates connections lists for different orientations, implementing a
    # convolutional network with different gaussian orientation filters (single scale)
    conn_list = Filter2DConnector(input_size, input_size,
        v1_pop_size, v1_pop_size,
        gaussian_filters[i],
        size_k1, size_k2,
        jump, delays,
        gain=gaussian_gain)
    projections.append(Projection(input_pol_1, v1_pop[i],
        FromListConnector(conn_list), label='input[p0]->v1_pop_%d' % (i)))
    projections.append(Projection(input_pol_2, v1_pop[i],
        FromListConnector(conn_list), label='input[p1]->v1_pop_%d' % (i)))

```

Populations in the *Pooling and Competition layer* (V2) receive lateral inhibition which enhances the contrast of the response; this is done implementing a *ProximityConnector* function to generate a list of weights implementing a local winner-take-all within and between orientations. V2 populations are then connected to the V4 layer, which represents the result of this operation at higher level in the visual hierarchy:

```

if(wta_v2 == True):
    print "%g - Creating Lateral inhibition for the v2 populations" % timer.
        elapsedTime()
    for i in range(orientations):          # Cycles orientations
        for j in range(orientations):      # Cycles orientations
            if (i!=j):                     # Avoid self connections
                # Creates lateral inhibition between the v2 populations
                print "%g - v2[%d]->v2[%d] lateral inhibition" % (timer.elapsedTime()
                    , i , j)
                wta_between_list = ProximityConnector(v1_pop_size , v1_pop_size ,
                    pooling_size ,
                                                    wta_between_v2_weight , 1,
                                                    allow_self_connections=
                                                        True)
                projections.append(Projection( v2_pop[i] ,
                    v2_pop[j] ,
                    FromListConnector(wta_between_list) ,
                    target='inhibitory'))

print "%g - Creating within inhibition pools" % timer.elapsedTime()
for i in range(orientations):              # Cycles orientations
    wta_within_list = ProximityConnector(v1_pop_size , v1_pop_size , pooling_size ,
        wta_within_v2_weight , 1,
        allow_self_connections=False)

    print "%g - v2[%d] within inhibition" % (timer.elapsedTime() , i)
    projections.append(Projection( v2_pop[i] ,
        v2_pop[i] ,
        FromListConnector(wta_within_list) ,
        target='inhibitory'))

for i in range(orientations):              # Cycles orientations
    v2_v4_conn_list = subSamplerConnector2D(v1_pop_size , v4_pop_size , weights_v2_v4 ,
        1)

    print "%g - v2->v4[%d] subsampling projection" % (timer.elapsedTime() , i)
    projections.append(Projection( v2_pop[i] ,
        v4_pop[i] ,
        FromListConnector(v2_v4_conn_list) ,
        target='excitatory'))

```

Finally the projections between the PFC layer and V4, and the V4 layer and the LIP layer (representing the result of the selection biased by the PFC layer) are instantiated:

```

print "%g - Creating v4->lip projections" % timer.elapsedTime()
for i in range(orientations):              # Cycles orientations
    projections.append(Projection( v4_pop[i] ,
        lip ,
        OneToOneConnector(weights=weights_v4_lip , delays
            =1) ,
        target='excitatory'))

print "%g - Creating LIP WTA" % timer.elapsedTime()

```

```

projections.append(Projection( lip ,
                               lip ,
                               OneToOneConnector(weights=wta_lip_weight , delays=1),
                               target='inhibitory'))

print "%g - Creating pfc->v4 projections" % timer.elapsedTime()

for i in range(orientations):          # Cycles orientations
    projections.append(Projection( pfc[i],
                                   v4_pop[i],
                                   AllToAllConnector(weights=pfc_v4_weights , delays
                                                       =1),
                                   target='excitatory'))

```

### 6.2.3 Results

Experiment I tested interaction between working memory activity and the biasing layer for stimuli in isolation; Experiment II tested the ability of the *working memory* activity to bias selection towards a particular target in the *selection* layer. The stimulus consisted of a video of blinking oriented Gaussians, which were presented on a neutral background, Figure 6.3(b) (insert) and Figure 6.4. The experiments are illustrated in the video *Demonstration of an event driven neural based visual system Silicon Retina on SpiNNaker.mp4* in Appendix C.3.1

#### Experiment I: Tuning Curves for Stimuli in Isolation

The *retina* was sequentially stimulated with 8 differently oriented Gaussian functions ( $\sigma = 4.0$  pixels,  $\gamma = 0.2$  pixels). Each stimulus was presented for 5.0 s, stimuli blinked with a frequency of 0.5 Hz and *working memory* neurons fired with a mean frequency of 9.9 Hz. Figure 6.3(a) contains four panels, each of which shows the mean firing rates (MFR) of the *biasing layer* neurons when a particular *working memory* neuron was active and Figure 6.3(b) shows the average tuning curve for all bias conditions. The MFR of the *biasing layer* neurons for the preferred orientation (palest) was compared with the MFR of the other *biasing layer* neurons (mid) and the MFR in the unbiased condition (dark). The MFR for the unbiased and non-preferred neurons was very similar, while the target neurons experienced an increased firing rate for all stimuli, indicating that the *working memory* successfully increased the gain for only the target neurons. Similar stimuli which were not explicitly represented by any orientation map also show increased activity.

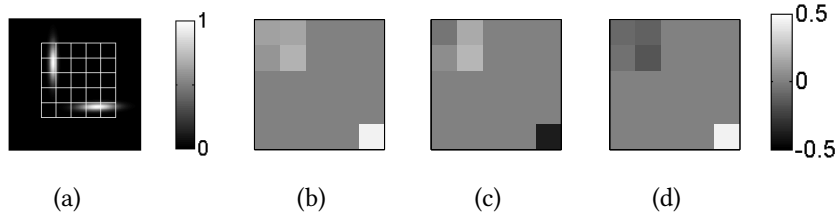


Figure 6.4: Experiment II - visual attention task.

### Experiment II: Visual Selection Task

Stimuli consisted of two oriented Gaussian functions ( $\sigma = 0.89$  pixels,  $\gamma = 0.07$  pixels), one in the top left corner of the stimulus image, and one in the bottom right corner of Figure 6.4(a). Two sets of orientations were tested at both positions:  $\{0^\circ, 90^\circ\}$  and  $\{45^\circ, 135^\circ\}$ . Stimuli blinked with a frequency of 0.5 Hz and activity was recorded for 40 s. Figure 6.4(b) shows the probability for a location becoming the attentional target in the unbiased condition (calculated as the ratio of the number of spikes at a location in *selection* to the total number of spikes in the *selection* layer). No selection bias was observed in this condition. Figs. 6.4(c) and 6.4(d) show the change in the selection probability when *working memory* was used to bias selection towards the feature in the top left corner and the bottom-right corner respectively. In both cases, selection was biased almost entirely towards the target feature. Figure 6.5 shows that the firing of the two active groups of *selection* neurons follows the activity of the *working memory* over time.

#### 6.2.4 An event-driven configurable platform

The system presented in this work forms a generic event-driven platform which comprises the SpiNNaker system with an existing AER sensor, exposing the advantage of a digital, programmable architecture for neural computation, as a generic event-driven system where neural networks may be rapidly configured and deployed. This processing platform benefits from the flexibility of SpiNNaker, in which neural network models can be described in a high-level programming language such as PyNN, making the hardware accessible to non-hardware experts. Large-scale, real-time models can be rapidly developed and configured before casting them into their more efficient, but also less accessible and more expensive, task-specific analog

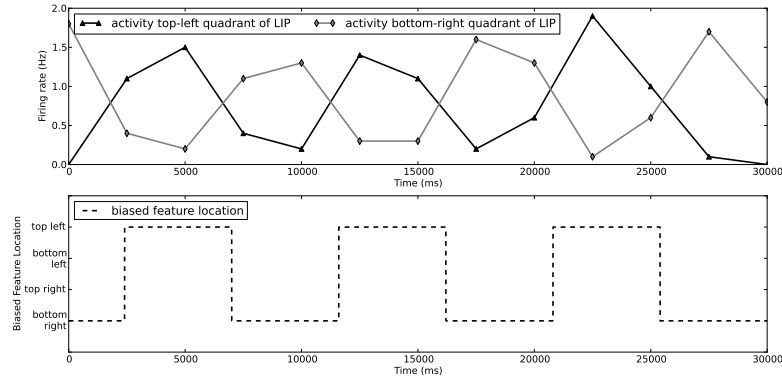


Figure 6.5: Activity in LIP follows the movements of the biased stimulus.

counterparts [Sonnleithner and Indiveri, 2011, Wijekoon and Dudek, 2012, Serrano-Gotarredona et al., 2009].

A neural network model of feature-based attentional selection was implemented upon this processing platform. In this network, a feature-based working memory was used to drive attentional selection towards a target visual feature. This work represents the first neuromorphic implementation of a feature-based attentional selection system. In future, this work will be extended to include a more faithful model of the neural circuit which subserves attentional selection through the inclusion of modulatory top-down bias, as opposed to the additive effect presented, and on attending complex stimuli learned from the composition of basic features discussed in this work.

### 6.3 Integrating SpiNNaker, NEF and the robot

While integration with AER sensors, as described in the previous section, makes use of the event-driven nature of the platform, interfacing the system with robots performing in a real environment makes use of its real-time capabilities. It is no surprise this has been attempted since the first SpiNNaker test chips have been produced; for example in the work presented in Davies et al. [2010] the neural network couples the sensors with actuators; it uses a Leaky Integrate-and-Fire (LIF) feed-forward network organised into 3 layers. The input layer comprises two 16x16 matrices combining the input for both polarities, and is connected to a 4x4 sub-sampler layer where units are in competition through lateral inhibition. Each neuron in the sub-sampler maps a 4x4

receptive field in both input matrices. The more frequently cells in the sub-sampling neuron's receptive field fire, the more likely that the sub-sampling neuron fires, inhibiting other sub-sampling neurons - a Winner Take All (WTA) mechanism. Finally, the output layer stimulates the actuators (Figure 6.6), and is designed to maintain the most salient stimuli in the central portion of the visual field. Outputs are also in competition, to achieve better stability. The model was first used to guide a robot during the Telluride 2010 workshop<sup>1</sup> on a test chip board, and then has consequently been re-implemented for the full SpiNNaker chip, with the addition of wireless communication and a new retina interface<sup>2</sup>.

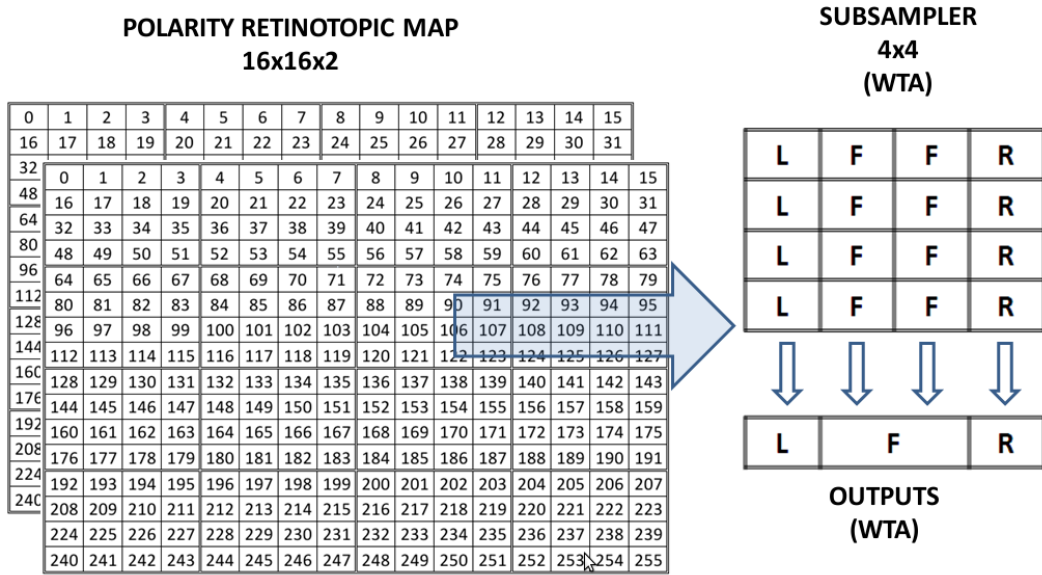


Figure 6.6: The Neural Network used. 512 input neurons from the retina feed into a hidden sub-sampler layer, and then to a competitive output layer to give the robot direction.

The neural network was generated with a PyNN script, and uploaded to the SpiNNaker chip through the software process described in the previous chapter.

However this approach uses a PC in the loop which buffers AER events, defeating the objectives of having an event based computing platform as the one described in the previous section. In the next section the integration of SpiNNaker 10<sup>2</sup> and 10<sup>3</sup> machines with the robot, done in collaboration with the Technische Universität München, will be presented.

The mobile robot used in this project is a custom developed omni-directional

<sup>1</sup><http://neuromorphs.net/nm/wiki/2010/rob10/SpiNNaker>

<sup>2</sup><http://www.youtube.com/watch?v=WcwEr5cMSoM>

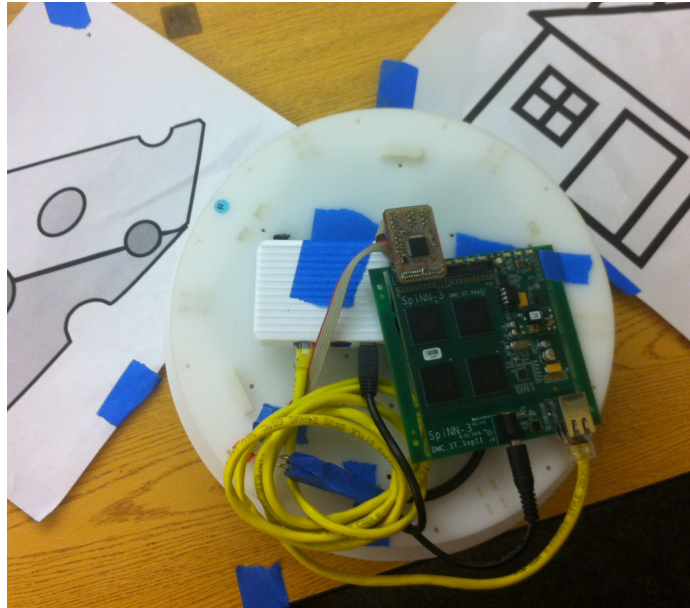


Figure 6.7: 4-chip board/OmniBot configuration for the system used in Telluride 2012.

mobile platform of 26cm diameter, with embedded low-level motor control and elementary sensory systems, provided by the Cognition for Technical System (CoTeSys) group of the Technische Universität München. An on-board ARM7 microcontroller for robot control receives desired motion commands in the  $x$  and  $y$  directions and rotation through a UART communication interface, and continuously adapts three motor control signals (PD-control) to achieve the desired velocities. The robot's integrated sensors include wheel encoders for position estimates, a 9 degrees of freedom inertial measurement unit (3 accelerometers, 3 gyroscopes, and 3 compasses) and a simple bump-sensor ring which triggers binary contact switches, located at  $60^\circ$  intervals on the robot circumference, upon contact with objects in the environment.

The robot can be used with both PyNN and Nengo, with PACMAN automating the mapping process and enabling messages to and from the robot. SpiNNaker communicates with the robot through a small customised interface board with an ARM Cortex micro-controller that translates SpiNNaker packets into robot commands and sensory commands. A configuration protocol, using MC packets, has been developed to configure the AER packets associated with sensor events or robot commands. The interface board is currently under improvement to allow higher data rates, so that event based sensory systems (such as silicon retinas or cochleae) can get interfaced directly to SpiNNaker. The overall system is a stand-alone, autonomous configurable platform with no PC in the loop (except for loading the model on board). An example

of the system used during the Telluride Neuromorphic Workshop 2012<sup>3</sup> is shown in Figure 6.7.

### 6.3.1 Simple model of path integration

The first network proposed uses a working memory to track the position of the agent in space by integrating its own velocity commands (a process known as path integration) and to switch between two competing behaviours: free exploration and returning home. As the agent moves freely in the environment its position with respect to the initial position (*home*) is tracked in the working memory, modelled with Nengo as a 2D integrator; when the return behaviour is activated, the position information is used to return to the starting point.

To accommodate for the OmniBot movement on surfaces the first step needed is to extend the NEF integration for SpiNNaker to cope with bi-dimensional representations for inputs and outputs, so to be able to elaborate driving commands in the *XY* plane.

This has been done by adding new 2D cell types for the NEF encoder and decoder neural types, presented in the previous chapter, and interfacing them with Nengo, both during the network creation phase (hence integrating it in PACMAN) and during the simulation phase by linking it to Nengo's simulation benchmark. This provides user control over the robot and visualisation of motor commands and internal states of the model for bi-dimensional inputs and outputs. The network represented in Figure 6.8 has been used to control the robot. The current state (behaviour) of the robot is represented in the *state* population, which acts as an integrator (working memory); this area is therefore able to maintain the current state over time with no additional input. The *position* 2D integrator population maintains the position of the agent by integrating the neural driving commands sent to the motors by the *drive* population. If the robot is in the explore state it can be driven using the commands in Nengo, which are then sent as 2D values to the explore population on the SpiNNaker board, which outputs the driving commands to the motors.

If the robot is in the return home state the driving commands are controlled by the return population, which tries to compensate for the value in the position population by sending opposite driving commands, hence returning to the origin point (*home*).

The script describing the network can be found in Appendix C.2.3. At first the *explore* and *state* populations are created and connected with their inputs. The *state*

<sup>3</sup><http://neuromorphs.net/nm/wiki/act12/results/OmniSpiNN>

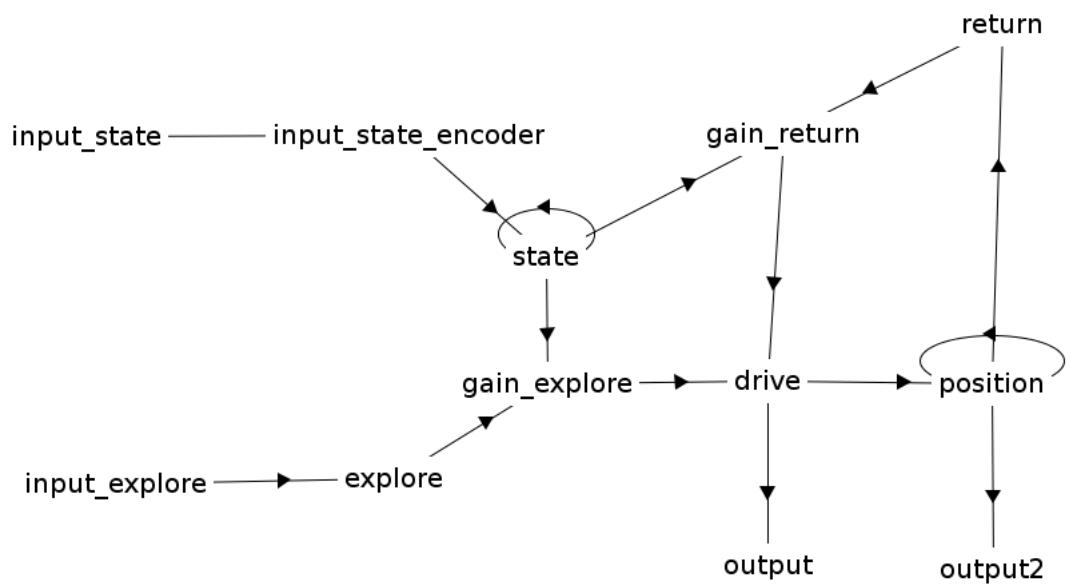


Figure 6.8: Nengo network for the return home robot. The current state (behaviour) of the robot is represented in the state population, which acts as an integrator (working memory) which is able to maintain the current state over time with no additional input. The position integrator population maintains the position of the agent by 2D integrating the neural driving commands sent to the motors. If the robot is in the explore state it can be driven using the commands in Nengo, which are then sent as 2D values to the explore population on the SpiNNaker board, which outputs the driving commands to the motors by passing through the laptop's wifi. If the robot is in the return home state the driving commands are controlled by the return population, which tries to compensate the value in the position population by sending negative drive commands.

population is self connected as well, as it serves as an integrator encoding input commands in a working memory module:

```
# Exploration behaviour
net.make_input('input_explore',[0,0])
net.make('explore',200,2,intercept=(-.9,.9), max_rate=(40,100))
net.connect('input_explore','explore')

# state integrator
net.make_input('input_state',[0,0])
net.make('input_state_encoder',200,2,intercept=(-.9,.9))
net.connect('input_state','input_state_encoder')

net.make('state',400,2,intercept=(-.9,.9), max_rate=(40,100), tau_ref=.002)
net.connect('input_state_encoder','state',pstc=tau)
#net.connect('state','state',transform=[[1,-0.1],[-0.1,1]],weight=tau,pstc=tau)
net.connect('state','state',transform=[[1.1,-0.1],[-0.1,1.1]],weight=tau,pstc=tau)
```

The *position* population, integrating the motor commands of the *drive* population in a 2D integrator is then created and connections are built.

```
# position integrator
net.make('position',1000,2,radius=1,intercept=(-.5,.5),max_rate=(40,150))
net.make('drive',400,2,intercept=(-.9,.9))
net.connect('drive','position',weight=tau,pstc=tau)
net.connect('position','position',pstc=tau)
```

Two gain populations are then created and connected, and are used to control the behaviour of the robot in the exploring and returning home state, and their effects on the drive population, which in turn controls the motors.

```
# gain explore
net.make('gain_explore',400,3,radius=1.4,intercept=(-.9,.9), tau_ref=.002, max_rate=(40,100))
net.connect('explore','gain_explore',index_post=[0,1])
net.connect('state','gain_explore',func=lambda x:[0,0,max(x[0],0)])
net.connect('gain_explore','drive',func=lambda x:(x[2]*x[0],x[2]*x[1]))

# return behaviour
net.make('return',200,2,intercept=(-.9,.9))
net.connect('position','return',weight=-1)

net.make('gain_return',200,3,radius=1.4,intercept=(-.9,.9))
net.connect('return','gain_return',index_post=[0,1])
def gain_return(x):
    return [0,0,max(x[1],0)]

net.connect('state','gain_return',func=gain_return)
net.connect('gain_return','drive',func=lambda x:(x[2]*x[0],x[2]*x[1]))
```

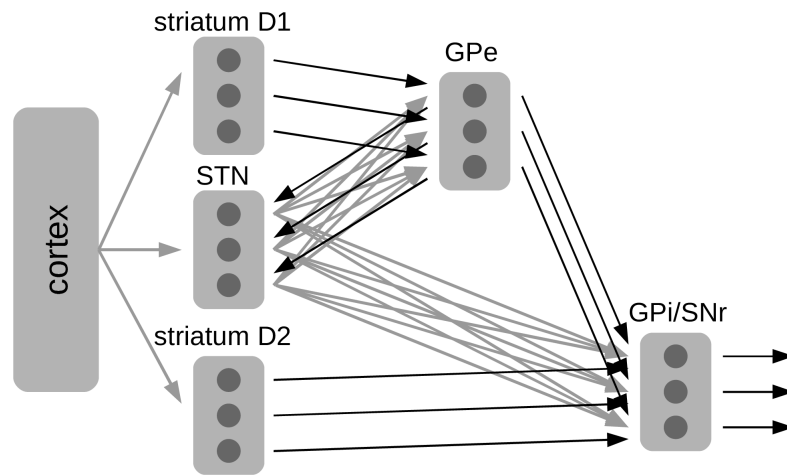


Figure 6.9: Basal ganglia model from Stewart et al. [2010b], based on Gurney et al. [2001].

The system was demonstrated live at the Telluride Neuromorphic workshop 2012<sup>4</sup>. The video shows the robot performing the task: at first the robot is controlled manually by inputting  $x$  and  $y$  coordinates to the *explore* population, responsible for encoding moving directions into neural activity; as the *state* population (also controlled externally by a bi-dimensional population controlling the gain of each of the two behaviours) is switched from controlled exploration to autonomous return home behaviours, the gain of the former behaviour decreases as the (autonomous) return behaviour, mediated by the *return* population which tries to compensate for the value represented in the *position* population, controls the *drive* population back to the free exploration starting point. A demonstration of the system can be found the video *NEF Return Home Nengo Model running on SpiNNaker.mp4* in Appendix C.3.2.

### Adding Basal Ganglia for state selection

The model above is coupled with a biologically plausible NEF model of basal ganglia presented in Stewart et al. [2010a] for selecting the state, where the basal ganglia selects the plan accordingly to the state represented in the working memory. Introduction of basal ganglia provides the possibility of manipulating inferences through IF-THEN clauses, hence implementing a neural based production system [Stewart et al., 2010b], as the basal ganglia is is generally considered the structure responsible for action selection of different actions with different utility values [Redgrave et al., 1999].

<sup>4</sup>[http://www.youtube.com/watch?feature=player\\_embedded&v=GVRVRB\\_deA](http://www.youtube.com/watch?feature=player_embedded&v=GVRVRB_deA)

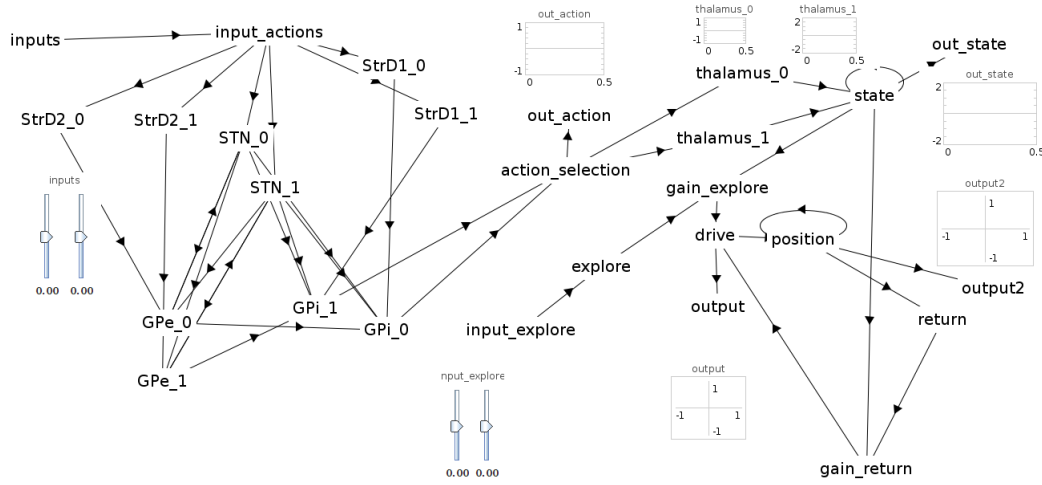


Figure 6.10: Nengo network for the return home robot with basal ganglia replacing the state integrator for action selection in the model described in Section 6.3.1.

The model of basal ganglia, originally based on Gurney et al. [2001], is presented in Figure 6.9, where light lines represent excitatory connections while dark lines represent inhibitory ones. Input from the cortex (as from a working memory, coding for the utility of a specific action such as explore) produces excitation in the striatum and in the sub-thalamic nuclei (STN): such excitatory inputs from the cortex are mediated by the *striatum D1* cells, which inhibit corresponding areas in the *globus pallidus internal* (GPi) and in the *substantia nigra pars reticulata* (SNr), while cells from the *striatum D2* and *globus pallidus external* (GPe) areas form a modulatory control structure. GPi and SNr areas can be considered the output of the structure; as output from the basal ganglia is inhibitory, the selected output's activity is turned off to prevent inhibition. The model parameters are biologically constrained, and latencies and response timings comparable to humans, emerge by the selection of realistic neural time constants.

In this context the state input can be thought of as representing the behavioural value of selecting a particular action (explore or return home); the basal ganglia selects the appropriate actions by controlling the gains into the *drive* population, associating different behaviours to different states.

The model comprises 5000 neurons and 2.95M synapses on 49 cores of a SpiNNaker  $10^2$  board. Figure 6.10 shows the position integrator already discussed, in conjunction with the basal ganglia module. In this model the SpiNNaker board is for

the first time mounted directly on top of the robot and controls its motor directly<sup>5</sup>.

Communication from Nengo to SpiNNaker (driving command values to input populations, visualizing neural responses) is done through wifi, while SpiNNaker sends and receives inputs/commands directly from/to the robot.

### 6.3.2 Spiking ratSLAM

Previous sections have proposed naive models of path integration, which share little or no resemblance to their biological counterparts. The hippocampus is notoriously able to hold invariant memory representations, in particular of high level concepts such as objects (as described earlier by the experiments of [Quijan Quiroga et al. \[2005\]](#)) or places; some hippocampal cells in rats are able to selectively encode location in space in their activity, and have therefore been called *place cells*: whenever the animal returns to a familiar place the corresponding place cells show a dramatic increase in activity [[O’Keefe and Dostrovsky, 1971](#)]. Analogously to receptive fields, the area (location) where a place cell responds can be called a *place field*. As the animal moves in space, firings from place cells can be used to build a *cognitive map* of the environment, a concept that has been researched since the early days of cognitive psychology [[Tolman, 1948](#)]. Such a cognitive map makes navigation through complex mazes and remembering the location of interesting stimuli possible.

In this section it is shown how place fields can be constructed utilizing the model presented by [Blair et al. \[2008\]](#), representing it with Nengo on SpiNNaker and running it on a robot, introducing a model for self localisation and mapping (SLAM) on a robotic rat (ratSLAM). The model is first shown in the mono-dimensional case, and then is extended to the bi-dimensional case, while also adding a basal ganglia structure to navigate autonomously between different places conditionally to previously visited places. The integration with the robot constitutes a step towards a neuro-physiologically plausible ratSLAM model.

#### Model of theta, grid and place cells

To navigate complex mazes and remember the location of important places as home and food, rats take advantage of cells in the hippocampus that respond when it is at specific locations, and when it has covered specific distances. This system, known as path integration, enables the animal to integrate the velocity of its own movements

---

<sup>5</sup>[http://www.youtube.com/watch?feature=player\\_embedded&v=MrumEW2oxbc](http://www.youtube.com/watch?feature=player_embedded&v=MrumEW2oxbc)

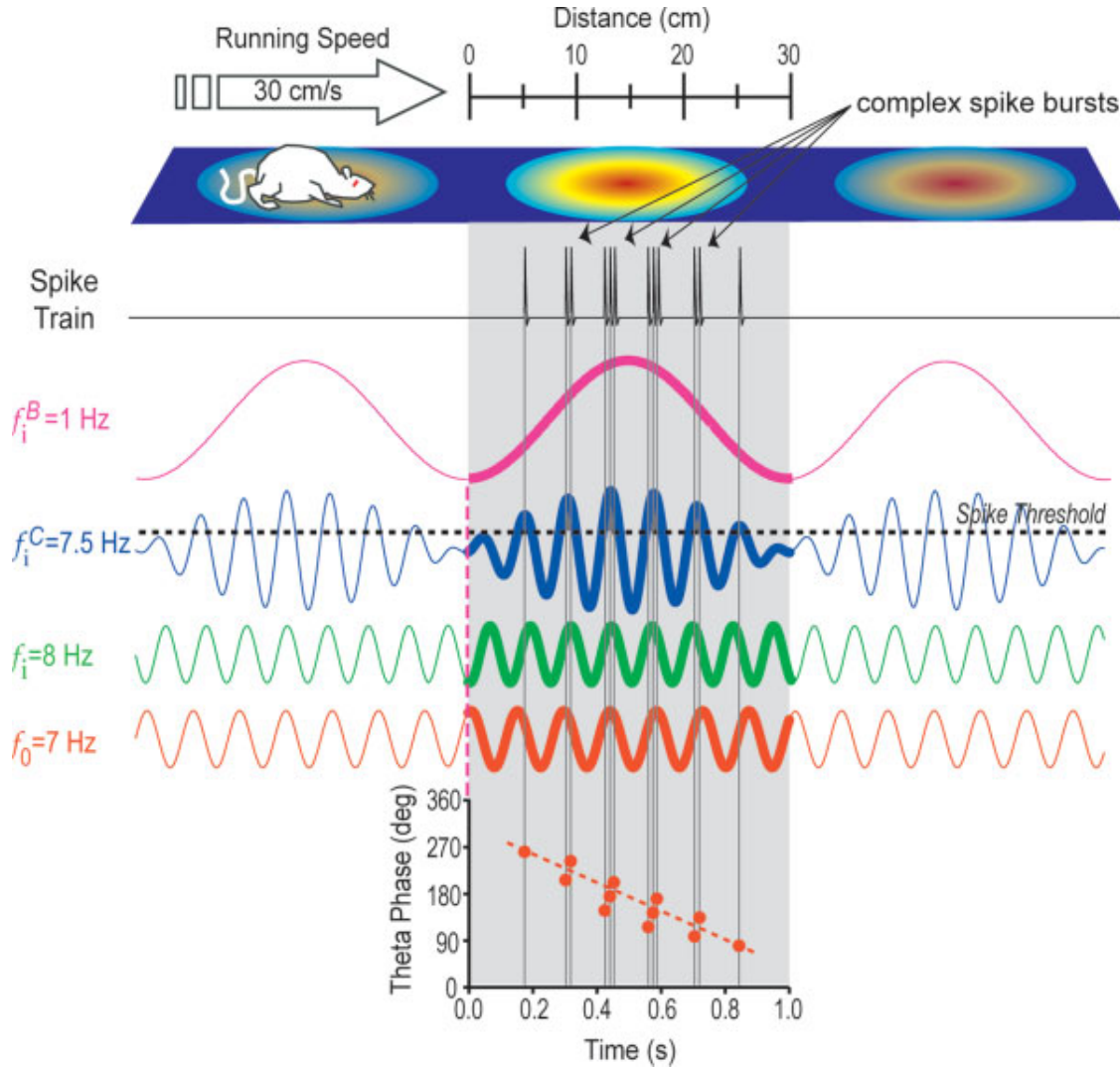


Figure 6.11: The theta oscillator model for place cells from Blair et al. [2008]. As a rat moves at a constant speed, a theta oscillator (red), with a preferred velocity and orientation modulating its oscillating frequency, interferes with a reference oscillator, representing the underlying theta rhythm (green); such interference produces an "envelope" waveform, a new beat, low-frequency, oscillation (magenta) riding on a high-frequency carrier (blue). Complex bursts of spikes are produced at the peaks of the carrier frequency which is in the constructive interference phase with the beat oscillation, and have a specific phase relationship (*phase precession*) with the first (red) oscillator, as shown at the bottom of the picture.

over time to update its position in a cognitive map of the environment. The position of the rat is encoded with a phase code, associating a pattern of phase angles between velocity-modulated theta oscillators, oscillating at 8-12Hz, to a specific place. Place cells receive their inputs from grid cells, which conversely show periodic activity when the animal is at the vertices which form a geometrically-arranged, hexagonal lattice pattern, or *grid field*, which extends over the environment perceived by the animal.

Blair et al. [2008] propose that path integration is performed through the generation of sub-cortical theta rhythms, which serve as a central pattern generator producing velocity-modulated theta rhythms. Such rhythms can be modelled by ring attractor networks. The current state (behaviour) of the robot is represented in the state population, which acts as an integrator (working memory) which is able to maintain the current state over time with no additional input. The position integrator population maintains the position of the agent by 2D integrating the neural driving commands sent to the motors.

If the robot is in the explore state it can be driven using the commands in Nengo, which are then sent as 2D values to the explore population on the SpiNNaker board, which outputs the driving commands to the motors by passing through the laptop's wifi.

If the robot is in the return home state the driving commands are controlled by the return population, which tries to compensate the value in the position population by sending negative drive commands. In the model, velocity controls the frequency of the oscillation. As a rat enters a place field, the corresponding place cell starts emitting bursts of spikes within a single theta cycle, with a stereotypical phase relationship with theta oscillations recorded locally with EEG. This observation can be explained by the model illustrated in Figure 6.11: as a rat moves at a constant speed, a theta oscillator (red), with a preferred velocity and orientation modulating its oscillating frequency, interferes with a reference oscillator, representing the underlying theta rhythm (green); such interference produces an "envelope" waveform, a new beat, low-frequency, oscillation (magenta) riding on a high-frequency carrier (blue). Complex bursts of spikes are produced at the peaks of the carrier frequency which is in the constructive interference phase with the beat oscillation, and have a specific phase relationship (*phase precession*, O'Keefe and Recce [1993]) with the first (red) oscillator, as shown at the bottom of the picture. In this model the place

and grid cells are constructed by many theta cells with different velocity tunings interfering. The frequency of oscillation of the theta cells increases (decreases) when the rat's velocity is matched (unmatched) to the preferred velocity of the individual theta oscillator. Hence, the frequency of oscillation tracks the velocity of the rat, while the phase of the oscillation, relative to the unperturbed oscillator, tracks the distance covered by the animal.

In the final part of this chapter it is demonstrated how this approach can be used to endow a mobile robot with real-time place and grid cells, and how they can be used in navigation to perform the role of a Simultaneous Localization and Mapping (SLAM) system for the robot, and has been developed in conjunction with Terry Stewart and Ralph Etienne-Cummings (model description in Nengo) and Jorg Conradt (robot interface).

### **Mono-dimensional case**

In the mono-dimensional case, when the animal is running in a linear track, grid cells can be constructed using the interference model, presented in the previous section, to create cells responding to vertex bumps located periodically at evenly spaced intervals along the track, by having their oscillation frequency as a function of the movement velocity. Two theta oscillators will then interfere, creating a waveform whose beat oscillation is proportional to the spatial period  $\lambda$ . A grid cell will then respond in every location  $x + k\lambda$ , where  $k$  is an arbitrary integer. Grid cells can then convert a phase signal, such as the difference in phase of multiple theta oscillators, into a bump of activity, or a periodic (firing) rate signal.

From these considerations [Blair et al.](#) propose that the neural substrate for position to phase coding is the same as the one used to generate velocity-tuned theta oscillations, with grid cells receiving inputs from sub-cortical ring attractor networks located in the midbrain and mammillary cortex, tuned to the velocity of the movement of the rat. These structures serve as a central pattern generator for velocity-tuned theta rhythms, encoding velocity in the frequency of oscillation. Grid fields can interfere to create *moiré grids* [[Blair et al., 2007](#)] that replicate a scale-invariant hexagonal lattice over the rat's environment. Such grid fields are then connected to place cells, performing a weighted sum of this input, constructing a Gaussian place field as a linear sum of grid fields of different periods, where the different spatial frequencies represented by the grid fields are the basis used to reconstruct a position in a Gaussian place field.

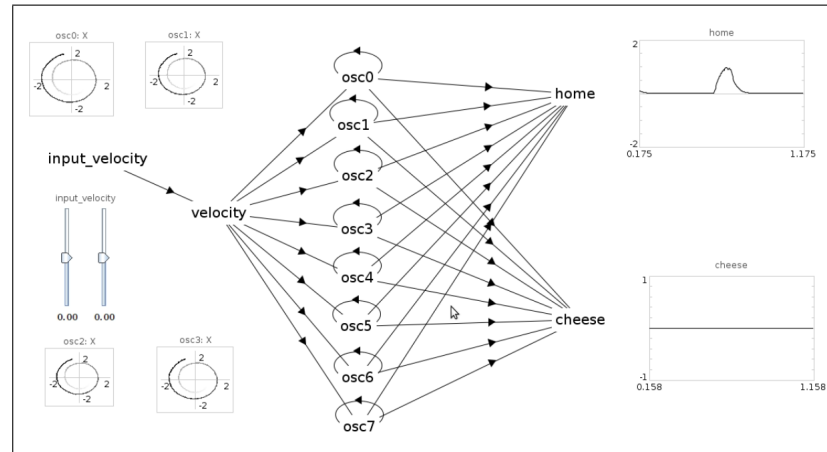


Figure 6.12: Nengo network model for path integration, mono-dimensional case.

The network used for this work<sup>6</sup> is illustrated in Figure 6.12: 8 oscillators have been modelled as velocity-controlled ring attractors, like the one described in Section 5.3.2, which receive, as input, the translational velocity of the robot from the same population used to control it (externally from Nengo). The oscillators (grid fields) are then connected to the populations representing the two place fields (*cheese* and *home*), where weights are determined by Nengo to perform phase-based extraction of the location, as the place is encoded in the phase differences of the velocity controlled and reference oscillators.

The model, written in Nengo and simulated on a SpiNNaker 4-chip board directly interconnected on the robot, comprises 2,700 neurons and 840,400 synapses, running on 36 ARM cores (3 SpiNNaker chips out of 4 in the board); results<sup>7</sup> show the ability of the model to perform path integration, by showing preferred activity of two place cells (*home* and *cheese*) for two distinct locations on a linear track. Figure 6.13 shows the robotic platform (top left), controlled with Nengo, navigating to the cheese landmark, and the *cheese* place cell (rightmost red box on the bottom) responding with a bump of activity while the *home* place cell (leftmost red box on the bottom) does not show activity. A demonstration of the system can be found in Appendix C.3.2 (*Embedded SpiNNaker robotic platform running a Nengo model of hippocampal place cells 1D.mp4*), while the full script modelling the network can be found in Appendix C.2.3.

<sup>6</sup><http://neuromorphs.net/nm/wiki/act12/results/Combined>

<sup>7</sup>[http://www.youtube.com/watch?feature=player\\_embedded&v=5bFc9csp3-s](http://www.youtube.com/watch?feature=player_embedded&v=5bFc9csp3-s)

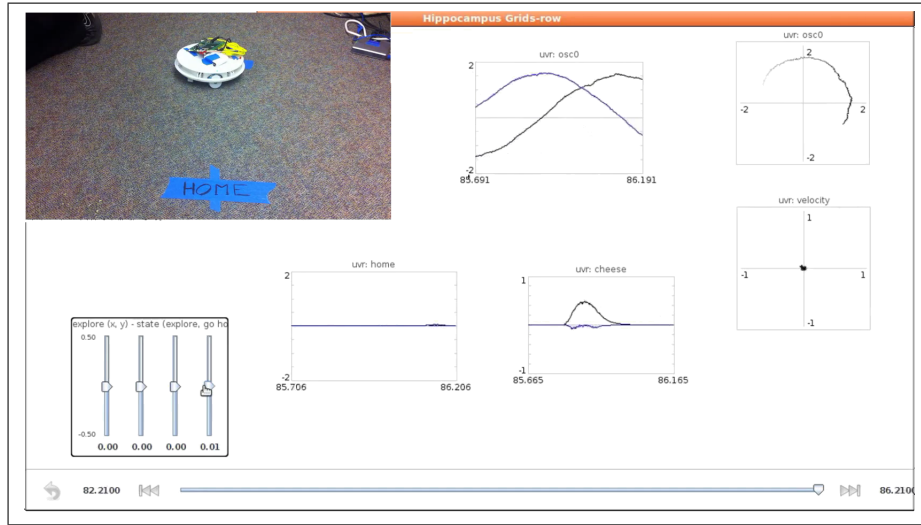


Figure 6.13: The robot/SpiNNaker platform navigating to the *cheese* place field in a video from the Telluride 2012 neuromorphic workshop. While the robot is navigating to the cheese landmark, the *cheese* place cell (rightmost red box on the bottom) is responding with a bump of activity while the *home* place cell (leftmost red box on the bottom) does not show any activity.

### Bi-dimensional case

The mechanism used to construct models of grid and place cells from theta oscillators can be extended to the bi-dimensional case as demonstrated in [Blair et al. \[2007\]](#), where scale-invariant grid fields are organised in a hexagonal mesh tessellating the animal environment. The extension in two dimensions is achieved by postulating the existence of theta oscillators modulated by both a velocity and a direction signal; Position of the animal is therefore encoded in oriented velocity signals. This is equivalent to modelling independent oscillators that encode for the same spatial frequency  $\lambda$ , each responding to a preferred speed and direction encoded in a velocity vector. Grid fields, constructed in such a way, constitute a basis for constructing a radial Gaussian field analogous to the mono-dimensional case.

To obtain autonomous navigation the model has been extended with a working memory module as integrator, representing the state of the system, and a basal ganglia module which acts as a production system: based on the cortical representation of the last place visited, the applicability of 6 rules is computed, and this information is passed into the striatum and STN areas of basal ganglia, which then connect onto the GPe and GPi areas, producing a thalamic output that indicates which of the six actions to perform.

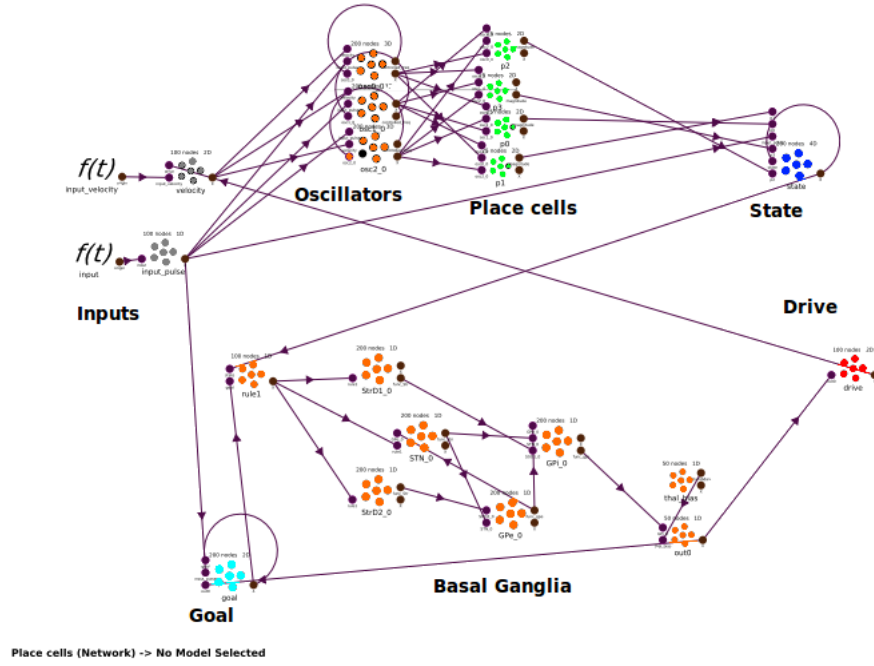


Figure 6.14: Simplified Nengo network for the bi-dimensional ratSLAM model.

The Nengo network used for this experiment is presented in Figure 6.14 in a simplified form: inputs (grey) are not used, as the basal ganglia effectively controls the navigation, while a synchronization input is given at the beginning of the simulation to synchronize the phase of the different oscillators; grid fields are modelled with 24 oscillators representing  $8 \text{ speeds} \times 3 \text{ directions}$ ; grid fields are connected to 4 place cells (green); place cells update the state module, a cortical representation (working memory) representing the last place visited (blue) and the direction (cyan); this information is used by the basal ganglia (orange) to plan the next action by controlling the drive population (red), which feeds back its signal to the velocity/direction modulated oscillators. Grid and place cells are connected by means of proxy populations used to decrease the fan-in of place cells, and are not shown in the figure for simplicity.

Overall the model comprises 13,910 neurons in 230 populations firing in the 40-100 Hz range, and 4,806,500 synapses in 1601 projections; a total of 670 ARM cores are used, corresponding to 42 SpiNNaker chips. A 48-node SpiNNaker board which is mounted directly on the robot and which communicates with it through a micro-controller based interface, is responsible for translating MC packets into commands for the robot. The overall power consumption of the board running the model is 23W. Results of the robot detecting 3 different place cells while being manually controlled

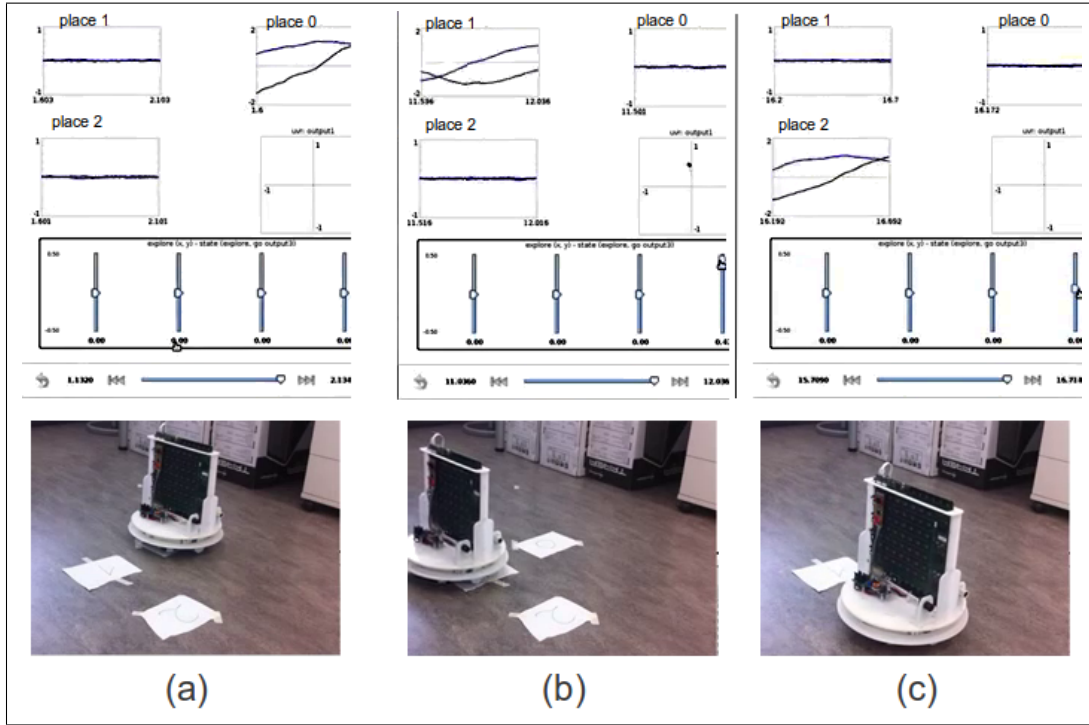


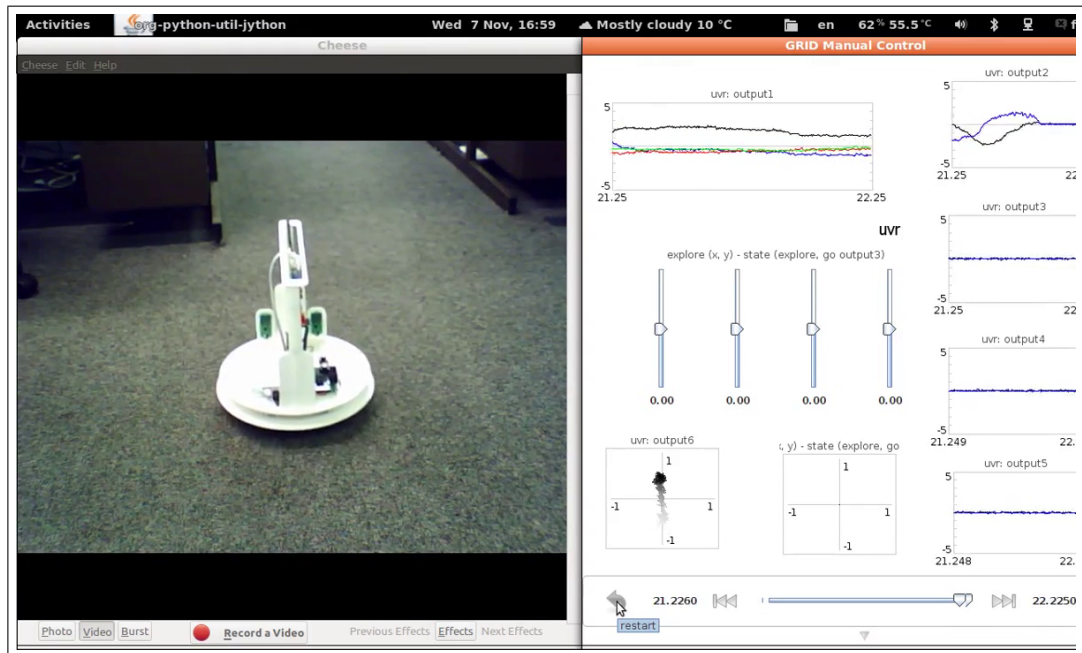
Figure 6.15: The robot/SpiNNaker platform detecting 3 places: (a) the robot is in *place 0*, the corresponding top right place cell is active (b) the robot moves to *place 1* (top left place cell) (c) the robot moves to the *place 2* cell.

are shown in Figure 6.15 (see Appendix C.3.2 - *Hippocampal place cells model using an holonomic robot and SpiNNaker.mp4*).

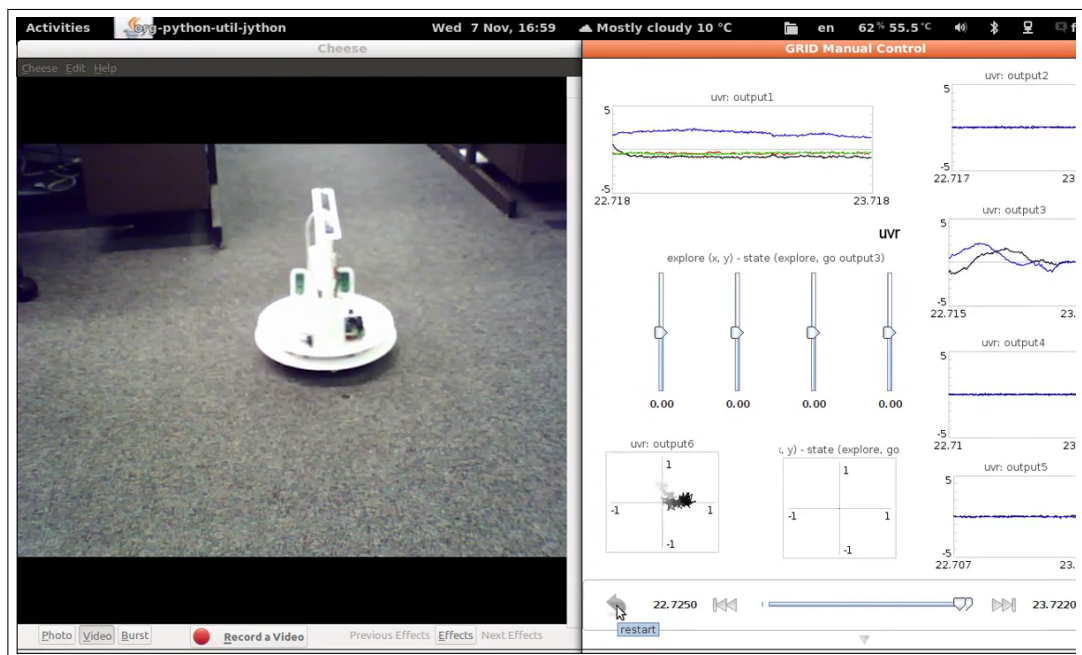
The system is able to navigate autonomously between four different places, detecting its position and changing trajectory accordingly to the last place visited. An example of a running experiment is shown in Figure 6.16 and 6.17: the robotic platform (top left) leaves the first place cell and its activity goes to zero (output2), and therefore the action plan in the basal ganglia (output1) changes as well. The video demonstrating the behaviour of the system can be found in Appendix C.3.2 (*Autonomous navigation in a cortical-hippocampal-place cells-basal ganglia neuromorphic model*), while the Nengo script modelling the network can be found in Appendix C.2.3.

## 6.4 Summary

This chapter has proposed a flexible, reconfigurable, event-based platform that can be used for building complex network models. The experiments presented run in

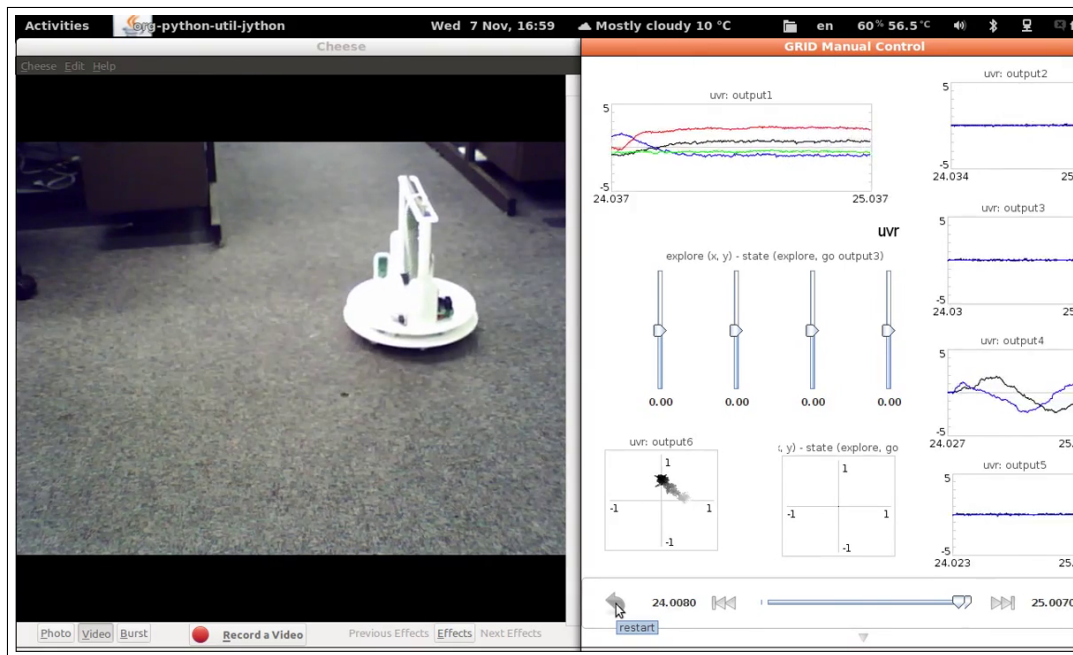


(a) The RatSLAM 2D model as the robot leaves the first place cell.

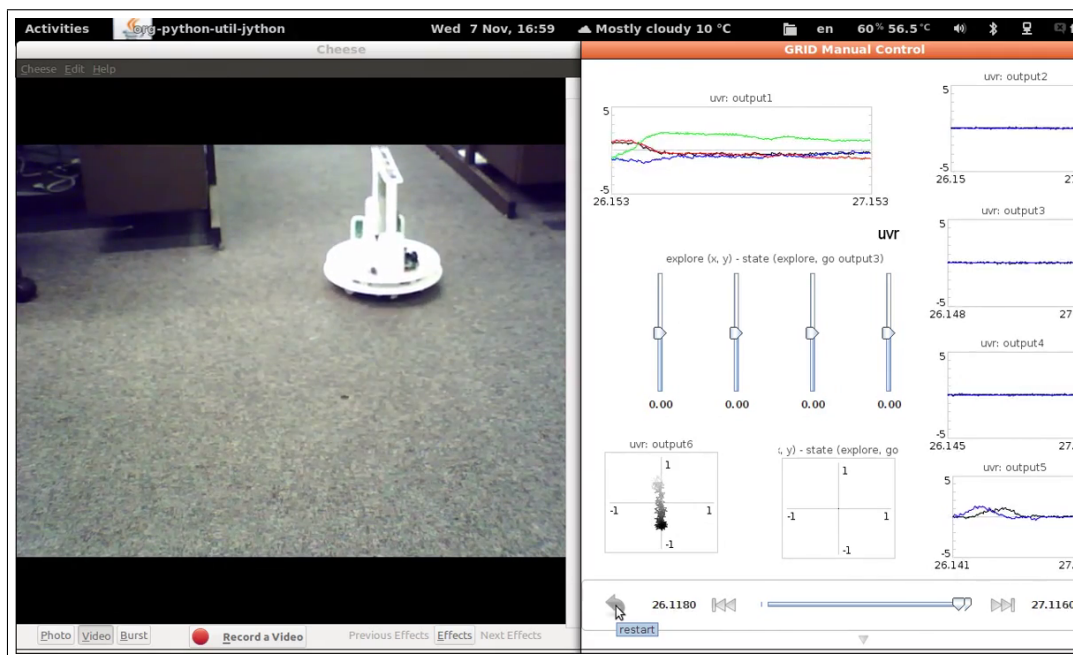


(b) The RatSLAM 2D model as the robot leaves the second place cell.

Figure 6.16: The RatSLAM 2D model autonomously navigating through 4 place cells.



(a) The RatSLAM 2D model as the robot leaves the third place cell.



(b) The RatSLAM 2D model as the robot leaves the fourth place cell.

Figure 6.17: The RatSLAM 2D model autonomously navigating through place cells 3 and 4.

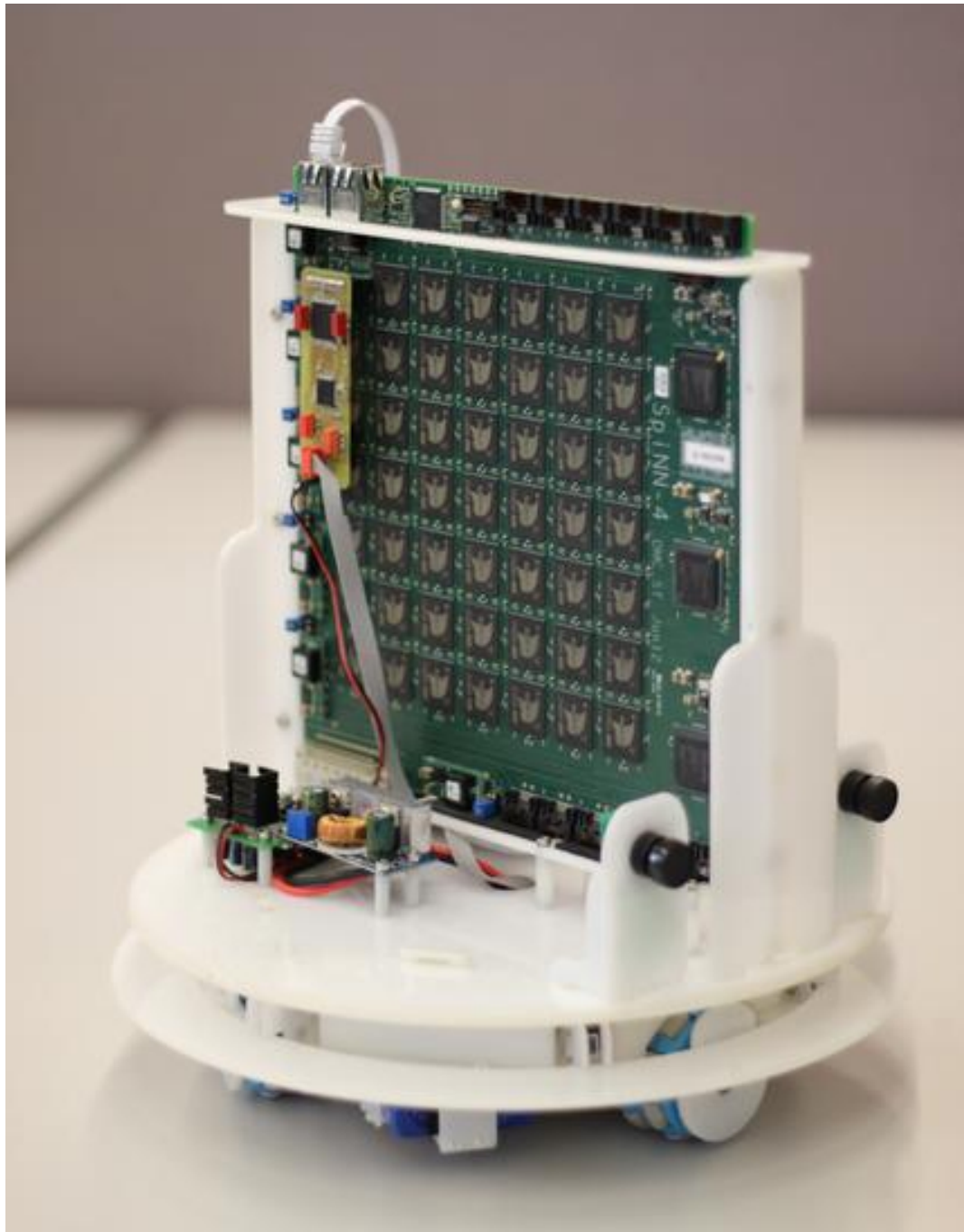


Figure 6.18: The SpiNNaker/robot platform.

real time on SpiNNaker. Interaction with the environment via AER-based-sensors and robotic platforms is possible through custom AER interfaces; the first example proposed is the FPGA used to connect the retina in the visual attention model; the second is the micro-controller board used to control the robot in the ratSLAM experiment. A realization of the platform, done in collaboration with the Technische Universität München, is presented in Figure 6.18: a 48-node SpiNNaker board is connected to the robot and to its sensors (including two silicon retinas) by means of the microcontroller board, handling translation of multicast AER packets from and to SpiNNaker using the bottom left asynchronous link of chip 0,0.

Sensors and robots become resources accessible at the model level: for example by creating a population representing the retina in PyNN, or a motor output population that encodes firing rates into motor commands for the robot. The mapping process manages them as custom entities, and provides the infrastructure seamlessly to place them on the system, for example by modifying routing table entries to route the appropriate motor-command MC packets to the link connected to the robot.

# Chapter 7

## Conclusions

Understanding how the brain works is a very ambitious project, which is shared by scientists from diverse disciplines studying at many different levels of abstraction, *from molecules to behaviour* as mentioned by Churchland and Sejnowski [1992] (Figure 7.1). Researchers are producing increasingly accurate data on morphology and dynamics, how they interconnect forming local and long-range circuits, and how activity in different brain areas correlates with behaviour.

Data from multiple laboratories is being consolidated into mathematical models, which are continuously refined within the computational neuroscience community. Computer simulations of such models are therefore essential tools to formalize, test, and verify hypotheses. Simulation of large models is extremely challenging, due to the computational and communication requests in the simulation of many elements and propagating activity between them; this limits simulation on standard computers, and simulation on supercomputers very expensive in terms of power.

The need for more powerful machines is a shared interest of engineers and computer scientists who seek inspiration from biology for new computational methods to fill the gap between human and machine intelligence. This interest is not surprising, as the brain is a very power-efficient and fast information elaboration system, and reverse-engineering the brain is a grand challenge of the National Academy of Engineering<sup>1</sup>. Networks of biologically inspired neurons are massively parallel systems, making them prime candidates for exploring novel distributed algorithms.

Configuring and distributing algorithms, such as networks of neurons on parallel architectures, imposes hardware- and model-specific challenges: placing the model on different nodes for cluster computing needs to balance the computational load

---

<sup>1</sup>[www.engineeringchallenges.org/?ID=9605](http://www.engineeringchallenges.org/?ID=9605)

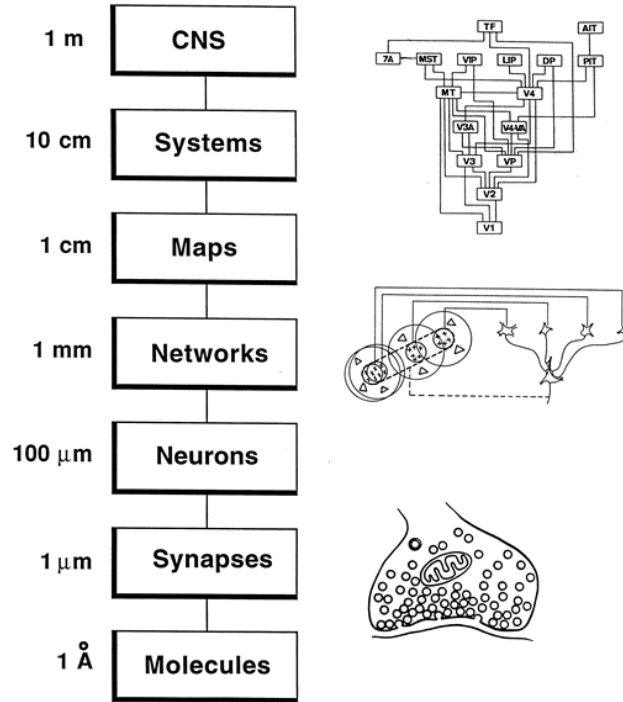


Figure 7.1: Levels of investigation in computational neuroscience, from [Churchland and Sejnowski \[1992\]](#).

equally to effectively exploiting parallelism [[Morrison et al., 2005](#)]; bias generations for VLSI systems need to be configured at runtime to compensate mismatch [[Indiveri and Chicca, 2011](#)]; FPGAs use synthesis tools to implement neurons and synapses in terms of circuitry efficiently; tools for configuring GPUs need to be used to generate code solving neural equations [[Goodman, 2010](#)], and to maximize performance in memory access by organizing synaptic data in coalesced structures [[Brette and Goodman, 2012](#)].

SpiNNaker enters the computational neuroscience community with its own specific set of performance and hardware features. A reconfigurable, efficient communication system, connecting many power-frugal, programmable digital cores has flexibility as one of its key features. Such flexibility is exposed with an approach that maps the model to the system, by abstracting the hardware from the language used to describe the model. Using the methods described in this thesis it has been possible to show real-time performance in simulating thousands of neurons and millions of synapses with different models.

Whilst flexibility is a strongly motivated feature, *separation of concerns* is the principle adopted when designing the mapping system. In particular, different concerns

are:

- The **user interface**, which makes the system accessible through a standard language such as PyNN or Nengo, without prior hardware knowledge.
- The **algorithms** performing the mapping process (PACMAN).
- The **data** accessed and transformed by such algorithms, stored in a database, along with all the data describing the hardware and modelling resources.

Separation of concerns is also used to organize data in three different representations: one concerning the neural model; a second intermediate PACMAN representation; and a final description of the model mapped onto the machine, and compiled to run. Being modular, the mapping process can easily be extended without a radical rewrite of the whole software stack, avoiding the *monolithic application* pitfall [Cornelis et al., 2012]. This enables, for example, the possibility to provide user-level control over the mapping of network elements to hardware resources, making the approach very flexible. Modularization contributes also in isolating and optimizing the underperforming modules in the system, as required to scale up the implementation of the mapping approach to bigger SpiNNaker systems. In particular the binary data structure compilation has been parallelised, with different processes generating structures for each chip. This starts from an abstracted representation of the model, which represents how Population and Projections are mapped onto a physical instantiation of a SpiNNaker system. Parallelization of this last, and very compute-intensive, step hints to the possibility of exploiting the parallel nature of SpiNNaker to self-configure and compile such data structures directly on the system, a step necessary to rapidly configure SpiNNaker systems much larger than the ones considered in this thesis.

The mapping approach can be rapidly expanded to accommodate new modelling or hardware resources in a consumer-producer model, where modelling resources are consumers of hardware resources such as computing cores, memory and communication bandwidth and routing entries. Rather than being specialised for a single neural or network model, or a to a single sensor or modality, the platform is a flexible and efficient exploration tool, whose aim is to let students and researchers experiment with an event-driven, biologically inspired system. Integration with different languages makes modelling possible at three levels of abstraction:

- **Unit Level:** by using C with the SpiNNaker API proposed in Sharp et al. [2011] to design custom neural and learning algorithms, or more generally to program

single units of a parallel, event-driven, real-time system, and map them using the approach proposed in this thesis.

- **Network Level:** by using PyNN to configure network topologies and neural parameters.
- **Functional level:** encoding functions in neurons by using the Neural Engineering Framework, a formal method for mapping control-theoretic algorithms onto the neural connections between populations of spiking neurons.

The mapping approach presented in this thesis was demonstrated by simulating heterogeneous models (even concurrently) on the system, opening new possibilities for mixed event-driven approaches, real-time systems interacting with the environment via AER sensors and robots.

Novel event-based computation paradigms inspired by the asynchronous, parallel and real-time nature of AER sensors, are being introduced, using time as the main information coding mechanism. Driven by visual sensors, event-based ASICs and novel models are proposing alternatives to frame-based visual processing. These assumptions have, for example, been used by [Serrano-Gotarredona et al. \[2006\]](#) to propose an asynchronous, event-based neuromorphic chip capable of performing convolutions with a programmable kernel on the silicon retina optical flow, or by [Benosman et al. \[2011\]](#) proposing an event-based derivation of epipolar geometry.

Event-driven computation finds in SpiNNaker a natural hardware substrate, as demonstrated by the work done in this thesis. Network models, integrating AER sensors and robots, can use SpiNNaker as an event-based, neural computational platform, configurable with standard languages, and capable of rapidly integrating a variety of different models through PACMAN.

Rather than being mutually exclusive these approaches can run concurrently on a configurable, parallel event-driven SpiNNaker system, mimicking the functional segregation and specialization of brain areas [[Tononi et al., 1998](#)], integrating them in a process running on the same platform. The approach described in this thesis is able to map such functionally-specialised structures (programs running on a cores) exchanging AER signals, encoding information in the type, source and timing the of an event (MC packet). While some portions of a model might need high biological fidelity, others might be modelled with simpler neurons, or more generally by transponders responding to events [[Izhikevich and Hoppensteadt, 2009](#)], or with units performing functions as in the convolutional network case.

Such flexibility has a key role in presenting SpiNNaker as a new effective technology in the neuromorphic field, enabling researchers and students from international institutions to use SpiNNaker hardware: there are active collaborators using the methods described in this thesis at the Ecole Polytechnique Federal de Lausanne, the Technische Universität München, the Highly-parallel VLSI Systems and Brain Microelectronics group of Dresden, the Institute for Neuroscience and Medicine in the Technische Universität Jülich and the Department of Informatics in the University of Sussex.

Some collaborations were fundamental in the research reported in this thesis: the collaboration with the University of Waterloo on porting the Neural Engineering Framework to SpiNNaker (described in Section 5.3, [Galluppi et al. \[2012c\]](#)); the collaboration with the University of Seville in interfacing the silicon retina (described in Chapter 6, [Galluppi et al. \[2012a\]](#)); the collaboration with the University of Munich (TUM) in the robotic platform presented in Chapter 6 [[Galluppi et al., 2012b](#)].

Interfacing AER sensors with SpiNNaker has already proven to be a fruitful research area, and different groups are working on prototype interfaces: the Institut de la Vision in Paris is working on interconnecting SpiNNaker with the ATIS (Asynchronous Time-based Image Sensor, [[Posch et al., 2010](#)]); the Biology group at the University of Osaka have successfully interconnected a neuromorphic visual sensor, inspired by the sustained and transient responses of the retina [[Kameda and Yagi, 2003](#)], to SpiNNaker; the Institute of Neuroinformatics in Zurich is working on interfaces with a neuromorphic silicon cochlea [[van Schaik and Liu, 2005](#)].

All the contributions described in the thesis converge in the SpiNNaker-robot platform, a flexible system which exploits the synergy of different elements present in the research field: fast, power-efficient neuromorphic AER sensors and SpiNNaker as an event-driven, efficient computational system that can be configured with standard tools already present in the computational neuroscience milieu.

# Appendix A

## Rank order codes and polychronization combined

This appendix describes in details the work presented in *Using Polychronization to decode rank order codes in a network of spiking neurons* [Galluppi and Furber, 2011]. Rank order coding can be used to encode information in the spike timings of an input population and polychronization can be used by a post-synaptic neuron to decode a temporal pattern emitted by a source population by compensating the latencies with the axonal delays.

Polychronization then seems the natural candidate to decode rank order coded inputs. If we look again at Figure 2.15 we can think of the firing patterns in neurons 1, 2 and 3 as two distinct rank order codes, and the two output neurons A and B as responders to a particular rank order code.

In theory a network exploiting connections delays value can be devised so to compute the number of neurons requested and their combination. Instead in this work a polychronous layer is connected to the input source with random delays. This layer acts as a spiking neural implementation of a sparse distributed memory [Furber et al., 2007] in spiking neural networks which is able to store and recall rank order codes.

### Proposed Model Architecture

In order to test the hypothesis that rank order coding and polychronization can be coupled to effectively represent information in a neural network a 3 layers model is proposed. All neurons are modeled as Leaky Integrate-and-Fire (LIF).

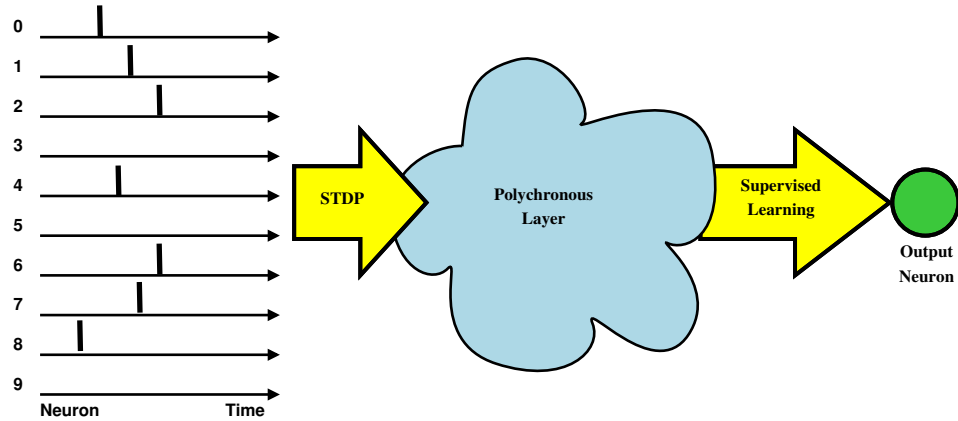


Figure A.1: Model architecture.

The input code is proposed at the input layer after a training phase with STDP and 200 random input codes. Neurons in the polychronous layer respond to different time locked properties of the output (parts of the input code with convergent delays will make a polychronous neuron fire). The combination of the polychronous neurons active are then learned by the output neuron through a supervised learning process.

The first layer is a population composed by 10 neurons. The coding strategies will determine the firings in the input layer. We have tested the system with a 7-of-10 rank order (like the one shown in 2.12(c) with  $N=7$  and  $M=10$ ) code and then with a rank order code coding for intensity. In this case all the input neurons in the population layer fired accordingly to the intensity of the stimulation (example b in Figure 2.12).

The first layer connects through a Spike-Timing dependent plasticity (STDP) [Song et al., 2000] [Bi and Poo, 1998] projection to the polychronous layer, which is formed by 1000 LIF neurons, intrinsically connected by inhibitory connections in a Winner-Take-All [Maass, 1999] pattern in order to avoid that two neurons learned the same pattern. The connection is made with 30% probability, and the delays are uniformly distributed in the range of the length of the code, letting the post-synaptic neuron potentiate subportions of the codes with convergent delays. In this way different neurons in the polychronous layer will detect different parts of the input code, accordingly on how delays converge on them.

Each code will then be represented by a different set of neurons in the polychronous layer responding to a precise time sequence of the inputs. In this sense the polychronous layer acts as a sparse distributed memory [Kanerva, 1988]: the input is

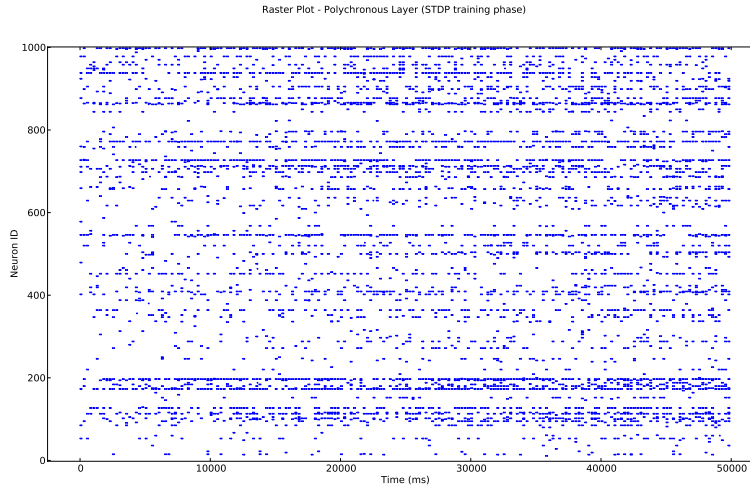


Figure A.2: STDP training phase in the polychronous layer: the combination of neuron activated in the polychronous layer is different for every code. 200 codes used in the simulation, every code lasts 250 ms.

represented in a higher dimensional space where the elements respond to parts of the input code, accordingly to how much their delays match part of the input pattern. These connections are trained with STDP in a preliminary tuning phase where 200 randomly picked symbols are submitted to the network, in order to make the neurons adaptively respond to different convergent delays. After this phase STDP is turned off, and the polychronous layer is capable of representing and discriminating different input codes.

To prove our hypothesis we implemented a supervised learning strategy whose only purpose is to verify that an output neuron can be associated to an arbitrary code and discriminated against other codes and noise. This supervised learning phase is used to map the output neuron to a combination of the activity in the polychronous layer so that the output (detector) neuron responds only to a particular combination of afferents. The learning is based on the change of the weight accordingly to an external supervised signal. Therefore it does not exploit the timing code of the system (is independent from time) and it doesn't affect the timing behaviour of the model (all delays are set to 1 ms). This shows that all the information is actually stored in the polychronous sparse distributed memory layer.

## Simulations and Results

**Tuning the polychronous layer with STDP:** Simulations were carried as follows: at the beginning the network is trained with 200 random symbols coded as described in the previous section in order to let STDP shape the connections; although some input codes share some neurons, the combination of neuron activated in the polychronous layer is different for every code. As a net effect STDP potentiated 30% of the connections. After this phase STDP is turned off, and the only learning that occurs is the one in the next supervised learning association phase.

**Associating a code to a detector neuron with supervised learning:** In this phase a code is selected to be learned and is presented at the input layer repeatedly, while a supervised learning algorithm is used to map the desired combination of input to the output neuron[Ponulak]. The weights is changed upon the receipt of an external control signal so to train it to respond to the combination of neurons firing in the polychronous layer (the weights in the polychronous layer don't change since STDP is turned off). All the neurons from the polychronous layer project to the output neuron with an initial weight set to 0 in order to let the training potentiate only the ones that contribute to the right code. The supervised learning algorithm used is structured as follows:

- the code is presented at the input layer and activates a subset of neurons in the polychronous layer
- if the output neuron fires learning is completed
- else the weight is increased and the process is repeated

Such process ensures that the output neuron will fire only when the combination of neurons in the polychronous layer is detected, hence mapping the combination of input to the output. It also does not alter the timing properties of the system since all the delays are set to 1 ms. An example of the supervised learning phase is presented in Figure A.3.

## Testing *N-of-M* rank order codes

In this test we trained the output neuron to respond to a rank order code of 7 out of 10 neurons firing each with a different position encoded in the firing time (see fig 2.12(c) and fig A.6). Accordingly to our prevision, after the output neuron is trained it only

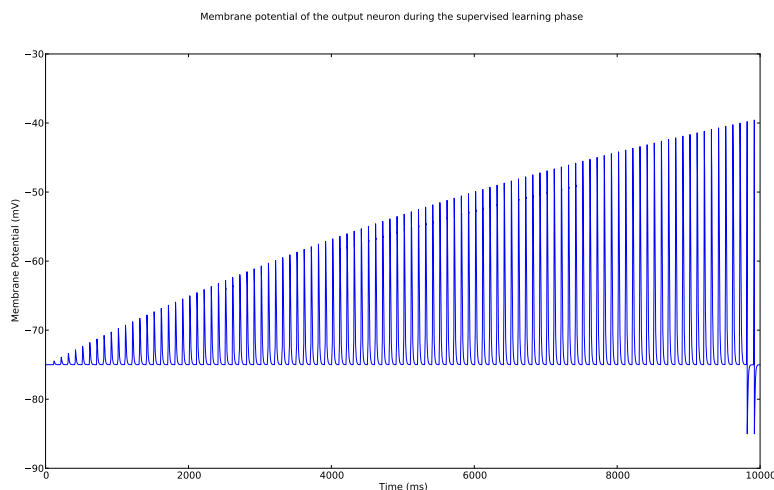


Figure A.3: The output neuron is trained to respond only when the combination of firings in the polychronous layer appears. This is done by increase the weight until a spike is produced (in this case at sec 10 of the simulation).

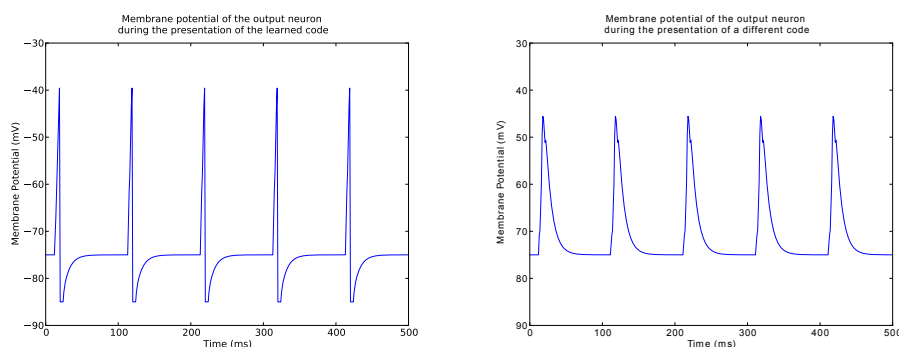


Figure A.4: Membrane potential of the output neuron during the presentation of two codes.

responds to the combination of firing neurons in the polychronous layer associated with the input code, otherwise it exhibits sub-threshold oscillations but stays silent. An example of the output neuron membrane potential in case of the presentation of the matching code and of a code with two positions inverted is presented in Figure A.4(a) and A.4(b) respectively.

Since the process can be repeated with as many output neuron and input codes as desired, is possible to train an arbitrary number of output neurons to respond to different input codes. To test the robustness of the learning we tested the network with 1000 different inputs, inserting the learned input code amongst them. Results

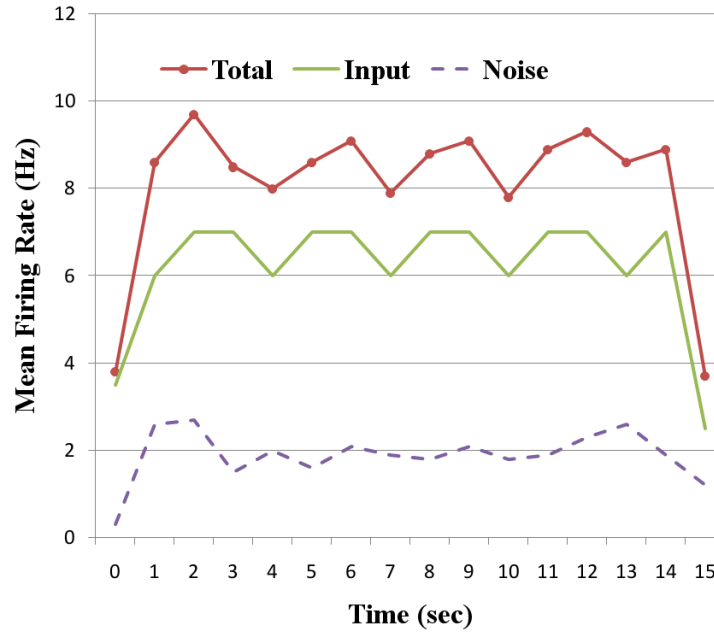


Figure A.5: Contribution in terms of population mean fire rate of noise to the normal activity of the input population

of the test are reported in Figure A.6. It can be seen that only when the learned code (on the right) is presented the output neurons fire; all the other codes are not capable of let the output neuron cross its firing threshold, set at -40 mV.

## Testing intensity rank order codes in a noisy regime

We repeated the procedure for a network coding intensity of the stimulus at the input layer by using rank order coding [Van Rullen et al., 1998] as shown fig2.12(b) and fig A.5. The learned pattern can be observed in Figure A.7. After the learning phase we observed that the output neuron was responding only to the selected code, as in the previous experiments, so we added a Poissonian noise source at the input layer with mean firing rate of 2Hz (equal to 25% of the input mean fire rate, as shown in fig A.5). Despite of that the output neuron is still able to respond only to the learned code, as shown in Figure A.7. Increasing the noise above the 30% of the original signal led to false detection (results not shown - 2 false detections on 100 symbols tested).

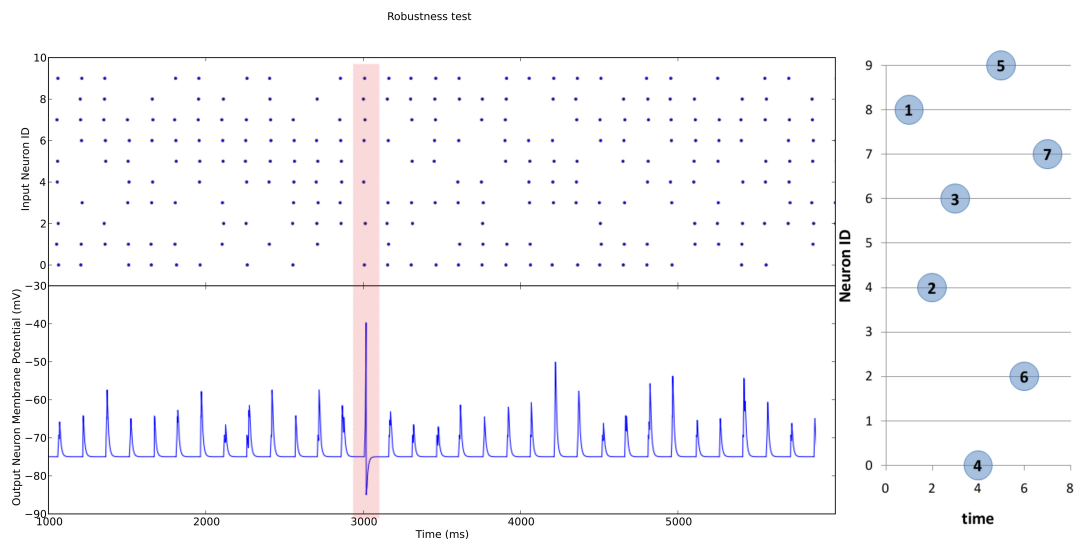


Figure A.6: The output neuron spikes only when the learned code is presented

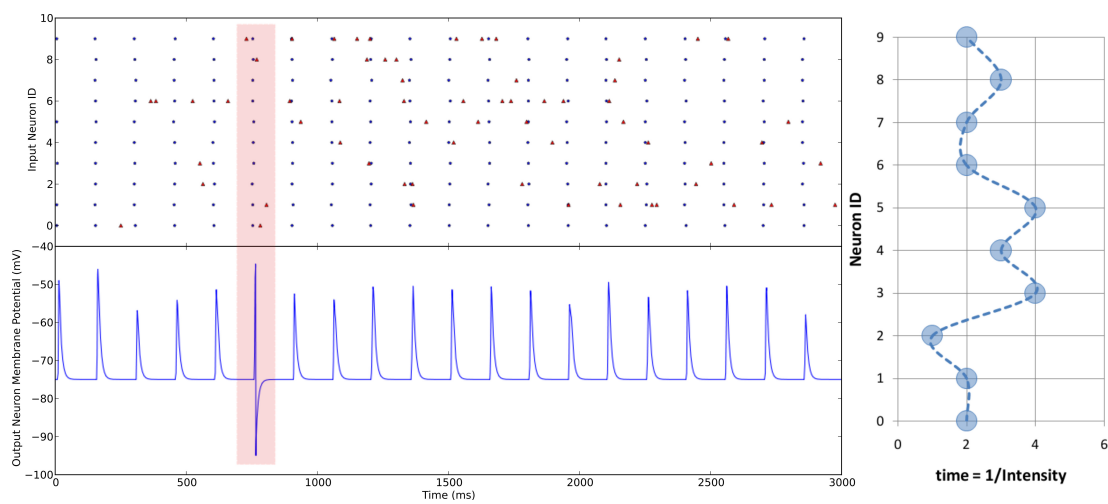


Figure A.7: The output neuron fires only when the learned code is presented.

## Discussion

The results of the tests show that rank order coding and polychronization can be used together as an efficient code representation strategy. The supervised learning algorithm proposed in the model shows that the information is encoded only in the polychronous layer. It is different from previous proposed models [Thorpe et al., 2001] [Delorme and Thorpe, 2001] because it uses a polychronous layer as a spiking neural implementation of a sparse distributed memory [Furber et al., 2007] to represent the code and a supervised learning strategy to learn it, rather than using shunt inhibition [Chance and Abbott, 2000]. Even in this basic form the learning has confirmed that the polychronous layer is capable of representing rank order codes in a way that can be disambiguated by a detector neuron.

More realistic methods of association can be implemented, for instance by using an STDP regulated by a signal as dopamine [Izhikevich, 2007]. Such control signal can be externally generated in supervised learning algorithm or can be generated by another portion of the network such as a value system [Sporns et al., 2000] in an unsupervised adaptive learning. The polychronous layer exploits the capabilities of polychronization by a minimum extent, in the sense that there's no plastic internal connection that can lead to the formation of polychronous groups inside the layer itself [Izhikevich et al., 2004]. Further tests in this direction are needed to verify the possibility of associating different code sequences together represented by polychronous groups within the polychronous layer.

If the spikes are considered to be produced by the same source the polychronous layer can code for different patterns where fine recognition is given by the integration where time codes significance. But the same input can be considered to be generated by different source areas each coding a different feature in parallel in functionally specialised maps [Tononi et al., 1994]. The polychronous layer could be a converging neural association area as the prefrontal cortex [Miller and Cohen, 2001]. Integration from different areas can then be done by coding the origin of the contribution into the delay to the associative area. The time order code could be embedded in oscillatory brain rhythms in order to have a common time reference (eg. gamma rhythms, which have been proposed to underlie the binding of different features into a single perceptual entity [Tononi et al., 1992, Singer and Gray, 1995]). Such oscillatory behaviour could be obtained with re-entrant connections [Sporns et al., 1989] and the rank order code could be synchronised with it and be coded on the crest of each oscillation.

The probability connection has proven to be a very important parameter. Lowering the connection number between the input and the polychronous layer has proven to lead to less discrimination power in the system. All the time parameters of the system (time constants, delays) as proven to lead to different behaviour. A more formal definition of the system can be researched, in order to optimize the parameters of the model.

# Appendix B

## Results from simulations with the LIF neuron

### B.1 Simulation results with a LIF neuron

Results from [Rast, Galluppi, Jin, and Furber \[2010a\]](#).

#### B.1.1 Single Neuron Dynamics

We tested single neuron dynamics by injecting short pulses of currents into a neuron defined by the following parameters:  $V_0 = V_s = -75mV$ ,  $V_r = -66mV$ ,  $f_n = 1/4ms^{-1}$ ,  $R = 8$ ,  $V_t = -56mV$ . In order to test the accuracy of our implementation we confronted it with the same neuron implemented with Brian [Goodman \[2008\]](#) and drove it with the same input current. Results are shown in Figure B.1: the membrane potential of the simulation run on SpiNNaker (continuous line) is confronted with the same neuron implemented in Brian (dashed line). The difference in the spiking region is due to the fact we artificially set  $V = 30mV$  when a neuron crosses the threshold in order to have a self-evident spike.

#### B.1.2 Spikes Propagation

In order to test the time precision of spike generation and propagation processes we implemented a network capable of detecting inter-pulse interval between spikes generated by two neurons, in order to reproduce the results presented in [Izhikevich and Hoppensteadt \[2009\]](#) (Section 2). Two input neurons (transponders) connect to

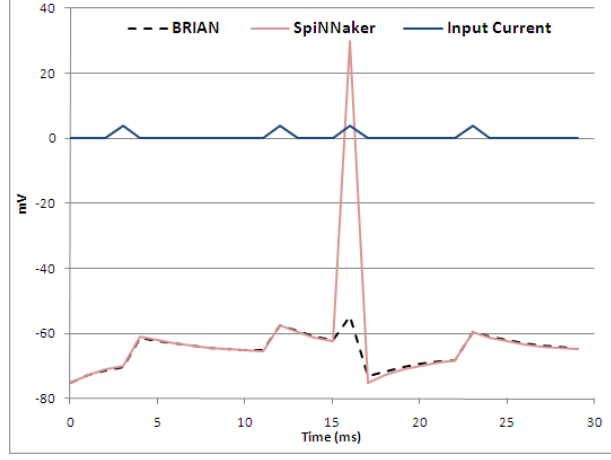


Figure B.1: Single neuron dynamics. The neuron is injected with 4 pulses of current.

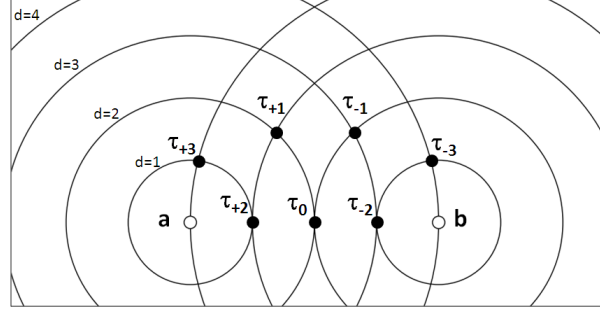


Figure B.2: Detection of interpulse interval between spikes generated from neurons  $a$  and  $b$ . Synaptic delay between transponders and detectors is marked as  $d$ .

7 output neurons (detectors) with a delay proportional to the distance, as shown in Figure B.2.

Weights are set so that a detector will fire only if hit by two coincident spikes with millisecond precision: detector neuron  $\tau_i$  only fires when inter-pulse interval  $t_a - t_b = i$ , where  $t_a$  and  $t_b$  are the absolute firing times of neuron  $a$  and  $b$  respectively (eg. neuron  $\tau_{-3}$  fires when neuron  $b$  fires 3 ms *after* neuron  $a$ , neuron  $\tau_{+2}$  fires when neuron  $b$  fires 2 ms *before* neuron  $a$  etc.) The network structure is represented in Figure B.3, where delays are specified. Simulation results are presented in Figure B.4, which presents (a) the raster plot, where only the neuron detecting the correct interpulse interval fires (b) the membrane potential of neurons  $a$  (blue) and  $b$  (red) (c) the membrane potential of detector neuron  $\tau_{+2}$ . The neuron only fires when spikes from neurons  $a$  and  $b$  converge on it accordingly to synaptic delays. Detector neuron  $\tau_i$  fires when inter-pulse interval  $t_a - t_b = i$ . The network is able to discriminate the inter-pulse interval between the firings of neuron  $a$  and neuron  $b$  with millisecond

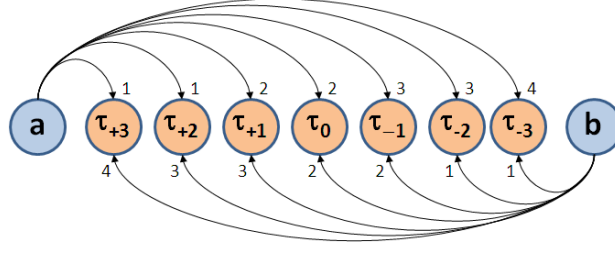


Figure B.3: Network Structure. Arrows represent connections between neurons. Values at the end of each arrow represent synaptic delays, which are set accordingly to detectors placement (cfr. Figure B.2)

precision by producing a spike from the correspondent detector neuron. This result confirms the millisecond precision of the LIF module implementation, and places the basis for frequency analysis using polychronous wavefronts computation on the SpiNNaker machine.

### B.1.3 Oscillatory Network Activity

We tested network dynamics by simulating the network described in Figure B.5: the network is formed by 100 neurons ( $V_0 = V_s = -75mV$ ,  $V_r = -65mV$ ,  $f_n = 1/16ms^{-1}$ ,  $R = 8$ ,  $V_t = -56mV$ ) divided in 80 excitatory neurons and 20 inhibitory neurons. Excitatory and inhibitory groups are connected so that when the activity of the excitatory group gets high the inhibitory group shuts the activity of the whole network. Every neuron in the excitatory group connects to 56 (70%) excitatory neurons and 2 (10%) inhibitory neurons with a random delay set between 1 and 8 milliseconds. Every inhibitory neuron is connected to every excitatory neuron (full connection - 100%) with a delay of 1 or 2 milliseconds. 8 neurons from the excitatory group are selected to be input neurons and are injected with a constant current of 3 mV, which makes them fire approximately every 10 milliseconds. Excitatory weights are set in order to slowly build up the background activity of the network; once the activity propagated is sufficient, the whole excitatory group starts firing, thus making the inhibitory neurons fire some millisecond later (due to less sparse connectivity). Inhibitory weights are set to quickly shut down the activity of the network.

Results of the simulation are presented in Figure B.6: input neurons (ID 1, 11, 22, 33, 44, 55, 66, 77) feed excitatory neurons (ID 0-79), slowly building up the activity until excitatory neurons start to fire with high frequencies. Inhibitory neurons (ID 80-99) are then activated, shutting down the activity of the network

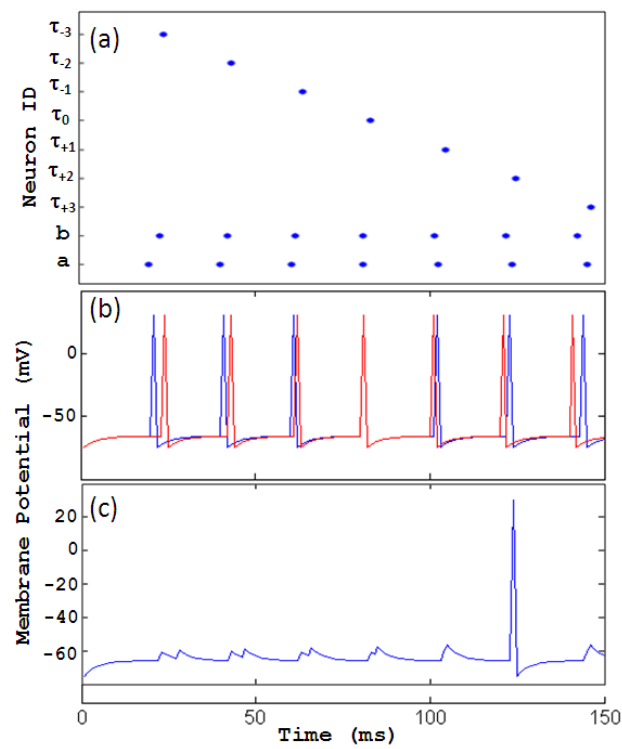


Figure B.4: Spikes Propagation.

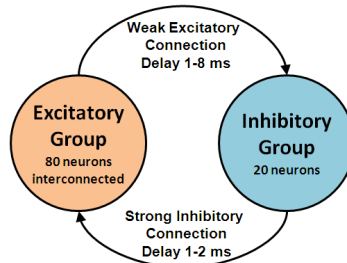


Figure B.5: Oscillatory Network Structure.

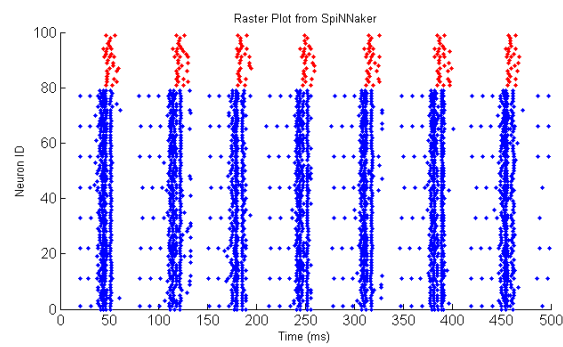


Figure B.6: Oscillatory Network Raster Plot.

# Appendix C

## Supplementary Material

This appendix contains a list of the supplementary material attached with the thesis.

### C.1 PyNN Files

#### C.1.1 Chapter 3

- `pynn/IF_cond_exp.py`

#### C.1.2 Chapter 4

- `pynn/va_benchmark.py`
- `pynn/random_connection.py`

#### C.1.3 Chapter 5

- `pynn/single-LIF.py`
- `pynn/single-IZK.py`
- `pynn/multimodel_1d.py`
- `pynn/multimodel_model2D_lat_inh_topdown.py`

### C.1.4 Chapter 6

- retina\_experiment\_chapter\_6.py

## C.2 Nengo Files

### C.2.1 Chapter 3

- nengo/square.py
- nengo/basalganglia.py

### C.2.2 Chapter 5

- nengo/communication\_channel.py
- nengo/integrator.py

### C.2.3 Chapter 6

- nengo/return\_home\_robot\_chapter6.py
- nengo/ratSLAM\_1D.py
- nengo/ratSLAM\_2D.py

## C.3 Videos

### C.3.1 AER Sensors

- Demonstration of an event driven neural based visual system Silicon Retina on SpiNNaker.mp4

### C.3.2 Robot

- NEF Return Home Nengo Model running on SpiNNaker.mp4
- Embedded SpiNNaker robotic platform running a Nengo model of hippocampal place cells 1D.mp4

- Hippocampal place cells model using an holonomic robot and SpiNNaker.mp4
- Autonomous navigation in a cortical-hippcampal-place cells-basal ganglia neuromorphic model.mp4

## C.4 SpiNNaker Package Documentation

- docs/spinnaker\_package.pdf

# Bibliography

- L F Abbott. Lapicque's introduction of the integrate-and-fire model neuron (1907). *Brain Research Bulletin*, 50(5-6), 1999.
- M Abeles. Local cortical circuits: An electrophysiological study. 6:102, 1982. URL <http://www.getcited.org/pub/102143568>.
- S Achard and E Bullmore. Efficiency and cost of economical functional brain networks. *PLoS Computational Biology*, 3:E17, 2007.
- I Aleksander. Neural systems engineering: towards a unified design discipline? *Computing Control Engineering Journal*, 1(6):259–265, November 1990.
- P Alivisatos, M Chun, GM Church, RJ Greenspan, ML Roukes, and R Yuste. The Brain Activity Map Project and the Challenge of Functional Connectomics. *Neuron*, 74(6):970–974, 2012. ISSN 08966273. doi: 10.1016/j.neuron.2012.06.006. URL <http://linkinghub.elsevier.com/retrieve/pii/S0896627312005181>.
- DJ Amit. *Modeling brain function: The world of attractor neural networks*. Cambridge University Press, 1992.
- R Ananthanarayanan, SK Esser, HD Simon, and D Modha. The cat is out of the bag: cortical simulations with  $10^9$  neurons,  $10^{13}$  synapses. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC '09*, pages 63:1–63:12, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-744-8. doi: <http://doi.acm.org/10.1145/1654059.1654124>. URL <http://doi.acm.org/10.1145/1654059.1654124>.
- L Badel, S Lefort, R Brette, CH Petersen, W Gerstner, and MJE Richardson. Dynamic I-V curves are reliable predictors of naturalistic pyramidal-neuron voltage traces. *Journal of Neurophysiology*, 99(2):656–66, 2008. URL <http://dx.doi.org/10.1152/jn.01107.2007>.

- R Benosman, P Rogister, and C Posch. Asynchronous event-based Hebbian epipolar geometry. *Neural Networks, IEEE Transactions on*, 22(11):1723–1734, 2011. ISSN 10459227. URL <http://www.ncbi.nlm.nih.gov/pubmed/21954205>[http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=6026950](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6026950).
- MA Bhuiyan, VK Pallipuram, and MC Smith. Acceleration of spiking neural networks in emerging multi-core and GPU architectures. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE, 2010. URL [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=5470899](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5470899).
- GQ Bi and MM Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience*, 18(24):10464–10472, December 1998.
- T Binzegger, RJ Douglas, and KAC Martin. A quantitative map of the circuit of cat primary visual cortex. *Journal of Neuroscience*, 24(39):8441–8453, September 2004. doi: 10.1523/JNEUROSCI.1400-04.2004. URL <http://dx.doi.org/10.1523/JNEUROSCI.1400-04.2004>.
- T Binzegger, RJ Douglas, and KAC Martin. Topology and dynamics of the canonical circuit of cat V1. *Neural Networks*, 22:1071–1078, 2009.
- JW Bisley, K Mirpour, F Arcizet, and WS Ong. The role of the lateral intraparietal area in orienting attention and its implications for visual search. *European journal of neuroscience*, 33:1982–1990, 2011.
- H Blair, A Weldon, and K Zhang. Scale-invariant memory representations emerge from moiré interference between grid fields that produce theta oscillations: a computational model. *Journal of Neuroscience*, 27(12):3211–29, 2007. ISSN 15292401. doi: 10.1523/JNEUROSCI.4724-06.2007. URL <http://www.ncbi.nlm.nih.gov/pubmed/17376982>.
- H Blair, K Gupta, and K Zhang. Conversion of a phase- to a rate-coded position signal by a three-stage model of theta cells, grid cells, and place cells. *Hippocampus*, 18(12):1239–55, January 2008. ISSN 1098-1063. doi: 10.1002/hipo.20509. URL <http://www.ncbi.nlm.nih.gov/pubmed/19021259>.

- K Boahen. Neurogrid: Emulating a Million Neurons in the Cortex. In *Engineering in Medicine and Biology Society, 2006. EMBS '06. 28th Annual International Conference of the IEEE*, volume Supplement, page 6702, 2006. doi: 10.1109/IEMBS.2006.260925.
- JM Bower and D Beeman. GENESIS (simulation environment). *Scholarpedia*, 2(3): 1383, 2007.
- JM Bower, D Beeman, and AM Wylde. *The book of GENESIS: exploring realistic neural models with the GEneral NEural Simulation System*. Springer-Verlag New York, Inc., New York, NY, USA, 1998. ISBN 0-387-94938-0. URL <http://portal.acm.org/citation.cfm?id=275590>.
- R Brette and W Gerstner. Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity. *Journal of neurophysiology*, 94(5): 3637–3642, November 2005. ISSN 0022-3077. doi: 10.1152/jn.00686.2005. URL <http://dx.doi.org/10.1152/jn.00686.2005>.
- R Brette and DFM Goodman. Simulating spiking neural networks on GPU. *Network: Computation in Neural Systems*, 23(4):167–182, 2012. URL <http://informahealthcare.com/doi/abs/10.3109/0954898X.2012.730170>.
- R Brette, M Rudolph, T Carnevale, M Hines, D Beeman, JM Bower, M Diesmann, A Morrison, PH Goodman, FC Harris, M Zirpe, T Natschlger, D Pecevski, B Ermentrout, M Djurfeldt, A Lansner, O Rochel, T Vieville, E Muller, AP Davison, S Boustani, and A Destexhe. Simulation of networks of spiking neurons: a review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398, December 2007. doi: 10.1007/s10827-007-0038-6. URL <http://dx.doi.org/10.1007/s10827-007-0038-6>.
- K Brohan, K Gurney, and P Dudek. Using reinforcement learning to guide the development of self-organising feature maps for visual orienting. In L S Illiadis, K Diamantaras, and W Duch, editors, *International Conference on Artificial Neural Networks*, pages 180–189, 2010.
- D Brüderle, E Müller, A Davison, E Muller, J Schemmel, and K Meier. Establishing a Novel Modeling Tool: A Python-based Interface for a Neuromorphic Hardware System. *Frontiers in Neuroinformatics*, 3(17):1–10, 2009. ISSN 1662-5196. doi: 10.3389/neuro.11.017.2009.

- D Brüderle, MA Petrovici, B Vogginger, M Ehrlich, T Pfeil, S Millner, A Grübl, K Wendt, E Müller, MO Schwartz, D Husmann, S Jeltsch, J Fieres, M Schilling, P Müller, O Breitwieser, V Petkov, L Muller, AP Davison, P Krishnamurthy, J Kremkow, M Lundqvist, E Muller, J Partzsch, S Scholze, L Zühl, C Mayr, A Destexhe, M Diesmann, TC Potjans, A Lansner, R Schüffny, Johannes Schemmel, Karlheinz Meier, and Others. A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. *Biological cybernetics*, 104(4-5):263–96, May 2011. ISSN 1432-0770. doi: 10.1007/s00422-011-0435-9. URL <http://www.springerlink.com/index/xlh783325w537622.pdf>.
- RC Cannon, MO Gewaltig, P Gleeson, US Bhalla, H Cornelis, ML Hines, FW Howell, E Muller, JR Stiles, and S Wils. Interoperability of neuroscience modeling software: current status and future directions. *Neuroinformatics*, 5(2):127–138, 2007. URL <http://www.springerlink.com/index/X650742T27460207.pdf>.
- T Carnevale. Neuron simulation environment. *Scholarpedia*, 2(6):1378, 2007.
- A Cassidy, AG Andreou, and J Georgiou. Design of a one million neuron single FPGA neuromorphic system for real-time multimodal scene analysis. In *Information Sciences and Systems (CISS), 2011 45th Annual Conference on*, pages 1–6. IEEE, 2011. URL [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=5766099](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5766099).
- S Celebrini, S Thorpe, Y Trotter, and M Imbert. Dynamics of orientation coding in area V1 of the awake primate. *Visual Neuroscience*, 10(5):811–825, 1993.
- DJ Chalmers. The puzzle of conscious experience. *Scientific American*, 273(6):80–86, December 1995. ISSN 0036-8733. URL <http://view.ncbi.nlm.nih.gov/pubmed/8525350>.
- FS Chance and LF Abbott. Divisive Inhibition in Recurrent Networks. *Network: Computation in Neural Systems*, 11:119–129, 2000.
- EY Chang, KF Morris, R Shannon, and BG Lindsey. Repeated sequences of interspike intervals in baroresponsive respiratory related neuronal assemblies of the cat brain stem. *Journal of Neurophysiology*, 84(3):1136–1148, September 2000.
- TYW. Choi, BE Shi, and K Boahen. An ON-OFF Orientation Selective Address Event Representation Image Transceiver Chip. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 51(2):342–353, February 2004. ISSN

- 1057-7122. doi: 10.1109/TCSI.2003.822551. URL [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=1266835](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1266835).
- PS Churchland and TJ Sejnowski. *The Computational Brain*, volume 63 of *Computational neuroscience*. MIT Press, 1992. ISBN 0262031884. URL <http://psycnet.apa.org/psycinfo/1992-97967-000>.
- H Cornelis, AD Coop, and JM Bower. A Federated Design for a Neurobiological Simulation Engine: The CBI Federated Software Architecture. *PLoS ONE*, 7(1): e28956, 2012. ISSN 19326203. doi: 10.1371/journal.pone.0028956. URL <http://dx.plos.org/10.1371/journal.pone.0028956>.
- Sharon M Crook, James A Bednar, Sandra Berger, Robert Cannon, Andrew P Davison, Mikael Djurfeldt, Jochen Eppler, Birgit Kriener, Steve Furber, Bruce Graham, Hans E Plesser, Lars Schwabe, Leslie Smith, Volker Steuber, and Sacha Van Albada. Creating, documenting and sharing network models. *Network Computation in Neural Systems*, 2012. URL <http://informahealthcare.com/doi/abs/10.3109/0954898X.2012.722743>.
- S Davies, C Patterson, F Galluppi, A D Rast, D Lester, and S B Furber. Interfacing Real-Time Spiking I/O with the SpiNNaker neuromimetic architecture. *Proceedings 17th International Conference, ICONIP 2010.*, pages 7–11, August 2010.
- S Davies, AD Rast, F. Galluppi, and SB Furber. Maintaining real-time synchrony on SpiNNaker. *Proceedings of the 8th ACM Conference on Computing Frontiers*, page 15, 2011.
- S Davies, F Galluppi, AD Rast, and SB Furber. A forecast-based STDP rule suitable for neuromorphic implementation. *Neural Networks*, 32:3–14, 2012a. URL <http://www.sciencedirect.com/science/article/pii/S0893608012000470>.
- S Davies, J Navaridas, F Galluppi, and S Furber. Population-based routing in the SpiNNaker neuromorphic architecture. *The 2012 International Joint Conference on Neural Networks IJCNN*, pages 1–8, 2012b. doi: 10.1109/IJCNN.2012.6252635.
- AP Davison, D Brüderle, JM Eppler, J Kremkow, E Muller, D Pecevski, L Perrinet, and P Yger. PyNN: A Common Interface for Neuronal Network Simulators. *Frontiers in Neuroinformatics*, 2(0):11, 2008. ISSN 1662-5196. doi: 10.3389/neuro.11.011.2008. URL <http://dx.doi.org/10.3389/neuro.11.011.2008><http://www.frontiersin.org>

[org/Journal/Abstract.aspx?s=752&name=neuroinformatics&ART\\_D0I=10.3389/neuro.11.011.2008](http://www.sciencedirect.com/science/article/pii/S1053811908001011).

P Dayan and L F Abbott. *Theoretical Neuroscience*. MIT Press, Cambridge, 2001a.

P Dayan and LF Abbott. *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 1st edition, December 2001b. ISBN 0262041995. URL <http://www.worldcat.org/isbn/0262041995>.

E De Schutter. *Computational Modeling Methods for Neuroscientists*. The MIT Press, 1st edition, November 2009. ISBN 0262013274, 9780262013277. URL <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262013274>.

S Dehaene, M Kerszberg, and JP Changeux. A neuronal model of a global workspace in effortful cognitive tasks. *Proceedings of the National Academy of Sciences of the United States of America*, 95(24):14529–14534, 1998. URL [http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list\\_uids=9826734](http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=9826734).

S Dehaene, C Sergent, and JP Changeux. A neuronal network model linking subjective reports and objective physiological data during conscious perception. *Proceedings of the National Academy of Sciences of the United States of America*, 100(14):8520–8525, July 2003. ISSN 0027-8424. doi: 10.1073/pnas.1332574100. URL <http://dx.doi.org/10.1073/pnas.1332574100>.

A Delorme. Early cortical orientation selectivity: how fast inhibition decodes the order of spike latencies. *Journal of computational Neuroscience*, 15(3):357–365, 2003. ISSN 0929-5313. URL <http://dx.doi.org/10.1023/A:1027420012134>.

A Delorme and SJ Thorpe. Face identification using one spike per neuron: resistance to image degradations. In *Neural Networks*, volume 14, pages 795–803. Elsevier, 2001. URL <http://linkinghub.elsevier.com/retrieve/pii/S0893608001000491>.

DC Dennet. *Consciousness explained*. ePenguin, 1993.

A Destexhe, ZF Mainen, and TJ Sejnowski. An efficient method for computing synaptic conductances based on a kinetic model of receptor binding. *Neural Computation*, 6(1):14–18, January 1994. ISSN 0899-7667. doi: 10.1162/neco.1994.6.1.14. URL <http://dx.doi.org/10.1162/neco.1994.6.1.14>.

- J Duncan and GW Humphreys. Visual search and stimulus similarity. *Psychological Review*, 96(3):433–458, 1989.
- A Dupuy, J Schwartz, Y Yemini, and DF Bacon. NEST: A network simulation and prototyping testbed. *Communications of the ACM*, 33(1-):63–74, 1990. URL <http://dl.acm.org/citation.cfm?id=84549>.
- D Durstewitz. Implications of synaptic biophysics for recurrent network dynamics and active memory. *Neural Networks*, 22(8):1189–1200, October 2009. doi: 10.1016/j.neunet.2009.07.016. URL <http://dx.doi.org/10.1016/j.neunet.2009.07.016>.
- GM Edelman. *Neural Darwinism: The Theory of Neuronal Group Selection*. Basic Books, 1987. ISBN 0465049346. URL <http://www.loc.gov/catdir/enhancements/fy0831/87047744-b.html>.
- GM Edelman. *Bright air, brilliant fire: On the matter of the mind*. Basic books, 1993.
- C Eliasmith. A unified approach to building and controlling spiking attractor networks. *Neural computation*, 7(6):1276–1314, 2005. URL <http://compneuro.uwaterloo.ca/cnrglab/f/eliasmith.2005.controlling.attractors.neurcomp.pdf><http://www.ncbi.nlm.nih.gov/pubmed/15901399>.
- C Eliasmith. How to build a brain: from function to implementation. *Synthese*, 2007. URL <http://www.springerlink.com/index/t316076556447j11.pdf>.
- C Eliasmith and CH Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT Press, Cambridge, MA, 2003.
- C Eliasmith, MB Westover, and CH Anderson. A general framework for neurobiological modeling: an application to the vestibular system. *Neurocomputing*, pages 1071–1076, 2002.
- C Eliasmith, TC Stewart, X Choo, T Bekolay, T DeWolf, Yichuan Tang, and Daniel Rasmussen. A Large-Scale Model of the Functioning Brain. *Science*, 338(6111):1202–1205, 2012. URL <http://www.sciencemag.org/content/338/6111/1202.abstract>.
- J Eppler, H Plesser, A Morrison, M Diesmann, and MO Gewaltig. Multithreaded and distributed simulation of large biological neuronal networks. *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 391–392, 2007.

- J Eppler, M Helias, E Muller, M Diesmann, and MO Gewaltig. PyNEST: A Convenient Interface to the NEST Simulator. *Frontiers in Neuroinformatics*, 2:12, 2008. doi: 10.3389/neuro.11.012.2008. URL <http://dx.doi.org/10.3389/neuro.11.012.2008>.
- AK Fidjeland and M Shanahan. Accelerated simulation of spiking neural networks using GPUs. In *Neural Networks, International Joint Conference on*, pages 1–8, 2010.
- AK Fidjeland, EB. Roesch, M Shanahan, and W Luk. NeMo: A Platform for Neural Modelling of Spiking Neurons Using GPUs. In *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, pages 137–144. IEEE, July 2009. ISBN 978-0-7695-3732-0. doi: 10.1109/ASAP.2009.24. URL <http://dl.acm.org/citation.cfm?id=1602239.1602629>.
- AK Fidjeland, D Gamez, MP Shanahan, and E Lazdins. Three Tools for the Real-Time Simulation of Embodied Spiking Neural Networks Using GPUs. *Neuroinformatics*, pages 1–24, 2012.
- MD Fox, AZ Snyder, JL Vincent, M Corbetta, D Van Essen, and ME Raichle. The human brain is intrinsically organized into dynamic, anticorrelated functional networks. *Proceedings of the National Academy of Sciences of the United States of America*, 102(27):9673–9678, July 2005. doi: 10.1073/pnas.0504136102. URL <http://dx.doi.org/10.1073/pnas.0504136102>.
- K Fukushima, CR Kaneko, and AF Fuchs. The neuronal substrate of integration in the oculomotor system. *Progress in neurobiology*, 39(6):609, 1992. URL <http://ukpmc.ac.uk/abstract/MED/1410443>.
- SB Furber and A Brown. Biologically-Inspired Massively-Parallel Architectures - Computing Beyond a Million Processors. In *Proceedings of the 2009 Ninth International Conference on Application of Concurrency to System Design*, ACSD '09, pages 3–12, Washington, DC, USA, July 2009. IEEE Computer Society. ISBN 978-0-7695-3697-2. doi: <http://dx.doi.org/10.1109/ACSD.2009.17>. URL <http://dx.doi.org/10.1109/ACSD.2009.17>.
- SB Furber and S Temple. Neural Systems Engineering. *Computational Intelligence: A Compendium*, 4(13):763–796, April 2008. doi: 10.1098/rsif.2006.0177. URL <http://dx.doi.org/10.1098/rsif.2006.0177>.

- SB Furber, S Temple, and AD Brown. High-Performance Computing for Systems of Spiking Neurons. *The AISB06 workshop on GC5: Architecture of Brain and Mind*, 2006a.
- SB Furber, S Temple, and AD Brown. On-chip and Inter-Chip Networks for Modelling Large-Scale Neural Systems. In *ISCAS*, pages 4–pp. IEEE, 2006b. URL [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=1692992](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1692992).
- SB Furber, G Brown, J Bose, JM Cumpstey, P Marshall, and JL Shapiro. Sparse Distributed Memory Using Rank-Order Neural Codes. *IEEE Transactions on Neural Networks*, 18(3):648–659, 2007.
- F Galluppi and SB Furber. Representing and decoding rank order codes using polychronization in a network of spiking neurons. *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 943–950, July 2011. doi: 10.1109/IJCNN.2011.6033324. URL [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=6033324](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6033324).
- F Galluppi, AD Rast, S Davies, and S Furber. A general-purpose model translation system for a universal neural chip. In *Neural Information Processing. Theory and Algorithms*, pages 58–65. Springer-Verlag, 2010. URL <http://www.springerlink.com/index/V7J81RQ677782U73.pdf>.
- F Galluppi, K Brohan, S Davidson, T Serrano-Gotarredona, JA Carrasco, B Linares-Barranco, and Steve B Furber. A real-time, event-driven neuromorphic system for goal-directed attentional selection. In *Neural Information Processing*, pages 226–233. Springer, 2012a.
- F Galluppi, J Conradt, T Stewart, C Eliasmith, T Horiuchi, J Tapson, B Tripp, S Furber, and R Etienne-Cummings. Live Demo: Spiking ratSLAM: Rat hippocampus cells in spiking neural hardware. In *Biomedical Circuits and Systems Conference (BioCAS), 2012 IEEE*, page 91. IEEE, 2012b.
- F Galluppi, S Davies, S Furber, T Stewart, and C Eliasmith. Real time on-chip implementation of dynamical systems with spiking neurons. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, June 2012c. ISBN 978-1-4673-1490-9. doi: 10.1109/IJCNN.2012.6252706. URL <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6252706>.

- F Galluppi, S Davies, AD Rast, T Sharp, LA Plana, and S.B. Furber. A Hierarchical Configuration System for a Massively Parallel Neural Hardware Platform. In ACM, editor, *CF '12 Proceedings of the 9th conference on Computing Frontiers*, pages 183–192, New York, 2012d.
- P Gao, BV Benjamin, and K Boahen. Dynamical System Guided Mapping of Quantitative Neuronal Models Onto Neuromorphic Hardware. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 59(10):2383–2394, 2012. ISSN 1549-8328. doi: 10.1109/TCSI.2012.2188956.
- A Gara and J E Moreira. {IBM Blue Gene} Supercomputer. *IBM Research Report*, 2011.
- W Gerstner and R Brette. Adaptive exponential integrate-and-fire model. *Scholarpedia*, 4(6):8427, June 2009. ISSN 1941-6016. doi: 10.4249/scholarpedia.8427. URL [http://www.scholarpedia.org/article/Adaptive\\_exponential\\_integrate-and-fire\\_model](http://www.scholarpedia.org/article/Adaptive_exponential_integrate-and-fire_model).
- W Gerstner, R Raphael, U Fuentes, and JL van Hemmen. A biologically motivated and analytically soluble model of collective oscillations in the cortex. *Biological Cybernetics*, 71:349–358, 1994.
- MO Gewaltig and M Diesmann. NEST (NEural Simulation Tool). *Scholarpedia*, 2(4):1430, 2007.
- NH Goddard, M Hucka, F Howell, H Cornelis, K Shankar, and D Beeman. Towards NeuroML: model description methods for collaborative modelling in neuroscience. *Philosophical Transactions of the Royal Society B: Biological Science*, 356(1412):1209–1228, August 2001. doi: 10.1098/rstb.2001.0910. URL <http://dx.doi.org/10.1098/rstb.2001.0910>.
- D Goodman. Brian: a simulator for spiking neural networks in Python. *Frontiers in Neuroinformatics*, 2, 2008. ISSN 16625196. doi: 10.3389/neuro.11.005.2008. URL <http://dx.doi.org/10.3389/neuro.11.005.2008>.
- D Goodman. Code generation: a strategy for neural network simulators. *Neuroinformatics*, 8(3):183–96, October 2010. ISSN 1559-0089. doi: 10.1007/s12021-010-9082-x. URL <http://www.ncbi.nlm.nih.gov/pubmed/20857234>.

- K Gurney, T Prescott, and P Redgrave. A computational model of action selection in the basal ganglia. i. A new functional anatomy. *Biological Cybernetics*, 85(6): 401–410, 2001.
- ML Hines and NT Carnevale. The NEURON Simulation Environment. *Neural Computation*, 9(6):1179–1209, August 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.6.1179. URL <http://dx.doi.org/10.1162/neco.1997.9.6.1179>.
- ML Hines and NT Carnevale. Expanding NEURON’s Repertoire of Mechanisms with NMODL. *Neural computation*, 12(5):995–1007, 2000. ISSN 0899-7667. doi: <http://dx.doi.org/10.1162/089976600300015475>.
- ML Hines and NT Carnevale. NEURON: a tool for neuroscientists. *Neuroscientist*, 7(2):123–135, April 2001.
- AL Hodgkin and AF Huxley. Action Potentials record from inside a nerve fiber. *Nature (Lond)*, 144:710–711, October 1939.
- AL Hodgkin and AF Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, August 1952. ISSN 0022-3751. URL <http://view.ncbi.nlm.nih.gov/pubmed/12991237>.
- D H Hubel and T N Wiesel. Receptive Fields, Binocular Interaction and Functional Architecture of the Cat’s Cortex. *Journal of Physiology*, 160:106–154, 1962.
- DH Hubel. *Eye, Brain, and Vision*. Scientific American Library (HPHLP), New York, 1988.
- DH Hubel and TN Wiesel. Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat. *Journal of Neurophysiology*, 28(2):229–289, 1965.
- MD Humphries, R Wood, and K Gurney. Dopamine-modulated dynamic cell assemblies generated by the GABAergic striatal microcircuit. *Neural Networks*, 22(8): 1174–1188, 2009. URL <http://dx.doi.org/10.1016/j.neunet.2009.07.018>.
- KM Hynna and K Boahen. Neuronal ion-channel dynamics in silicon. In *Proc. 2006 Int’l Symp. Circuits and Systems (ISCAS 2006)*, pages 3614–3617, 2006.

- G Indiveri and E Chicca. A VLSI neuromorphic device for implementing spike-based neural networks. In *Neural Nets WIRN11-Proceedings of the 21st Italian Workshop on Neural Nets*, pages 305–316, 2011.
- G Indiveri, E Chicca, and R Douglas. A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity. *IEEE Transactions on Neural Networks*, 17:211–221, 2006.
- G Indiveri, B Linares-Barranco, T Hamilton, A Van Schaik, R Etienne-Cummings, T Delbruck, SC Liu, P Dudek, P Häfliger, S Renaud, J Schemmel, G Cauwenberghs, J Arthur, K Hynna, F Folowosele, S Saighi, T Serrano-Gotarredona, J Wijekoon, Y Wang, and Boahen. Neuromorphic Silicon Neuron Circuits. *Frontiers in neuroscience*, 5(May):23, 2011. URL <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3130465&tool=pmcentrez&rendertype=abstract>.
- L Itti, C Koch, and E Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE transactions on pattern analysis and machine intelligence*, 20(11):1254–1259, 1998.
- EM Izhikevich. Simple Model of Spiking Neurons. *IEEE Trans. Neural Networks*, 14: 1569–1572, November 2003.
- EM Izhikevich. Which model to use for cortical spiking neurons? *IEEE Trans. Neural Networks*, 15(5):1063–1070, September 2004. ISSN 1045-9227. doi: 10.1109/TNN.2004.832719. URL <http://www.ncbi.nlm.nih.gov/pubmed/15484883>.
- EM Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. The MIT Press, 1 edition, November 2006a. ISBN 978-0-262-09043-8. URL <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262090430>.
- EM Izhikevich. Polychronization: Computation with Spikes. *Neural Computation*, 18(2):245–282, February 2006b. ISSN 0899-7667. doi: <http://dx.doi.org/10.1162/089976606775093882>. URL <http://www.ncbi.nlm.nih.gov/pubmed/16378515>.
- EM Izhikevich. Solving the distal reward problem through linkage of STDP and dopamine signaling. *Cereb Cortex*, 17(10):2443–2452, October 2007. ISSN 1047-3211. doi: 10.1093/cercor/bhl152. URL <http://dx.doi.org/10.1093/cercor/bhl152><http://www.ncbi.nlm.nih.gov/pubmed/17220510>.

- EM Izhikevich and GM Edelman. Large-scale model of mammalian thalamocortical systems. *Proceedings of the National Academy of Sciences of the United States of America*, 105(9):3593–3598, March 2008. doi: 10.1073/pnas.0712231105. URL <http://dx.doi.org/10.1073/pnas.0712231105>.
- EM Izhikevich and FC Hoppensteadt. Polychronous Wavefront Computations. *International Journal of Bifurcation and Chaos*, 19:1733–1739, 2009.
- EM Izhikevich, JA Gally, and GM Edelman. Spike-timing dynamics of neuronal groups. *Cerebral Cortex*, 14(8):933–944, August 2004. doi: 10.1093/cercor/bhh053. URL <http://dx.doi.org/10.1093/cercor/bhh053>.
- Dethier J, Gilja V, Nuyujukian P, Elassaad S, Shenoy KV, and Boahen K. Spiking neural network decoder for brain-machine interfaces. In *Proc. of the 5th International IEEE EMBS Conference on Neural Engineering, Cancun, Mexico*, pages 396–399, 2011.
- CE Jahr and CF Stevens. A quantitative description of NMDA receptor-channel kinetic behavior. *Journal of Neuroscience*, 10(6):1830–1837, June 1990.
- X Jin, SB Furber, and JV Woods. Efficient modelling of spiking neural networks on a scalable chip multiprocessor. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN) 2008*, pages 2812–2819, June 2008. doi: 10.1109/IJCNN.2008.4634194. URL <http://dx.doi.org/10.1109/IJCNN.2008.4634194>.
- X Jin, A Rast, F Galluppi, M Khan, and SB Furber. Implementing Learning on the SpiNNaker Universal Neural Chip Multiprocessor. In Chi S Leung, Minho Lee, and Jonathan H Chan, editors, *Neural Information Processing*, volume 5863, chapter 48, pages 425–432. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-10676-7. doi: 10.1007/978-3-642-10677-4\_48. URL [http://dx.doi.org/10.1007/978-3-642-10677-4\\_48](http://dx.doi.org/10.1007/978-3-642-10677-4_48).
- X Jin, F Galluppi, C Patterson, A Rast, S Davies, S Temple, and SB Furber. Algorithm and Software for Simulation of Spiking Neural Networks on the Multi-Chip SpiNNaker System. In *Neural Networks, 2010. IJCNN 2010. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, September 2010. doi: 10.1109/IJCNN.2008.4634194. URL <http://dx.doi.org/10.1109/IJCNN.2008.4634194>.

- S Kameda and T Yagi. An analog VLSI chip emulating sustained and transient response channels of the vertebrate retina. *Neural Networks, IEEE Transactions on*, 14 (5):1405–1412, 2003.
- ER Kandel, JH Schwartz, and TM Jessell. *Principles of Neural Science*. Appleton and Lange, Norwalk, CT, 1991.
- P Kanerva. *Sparse Distributed Memory*. MIT Press, Cambridge, MA, USA, 1988. ISBN 0262111322.
- MM Khan, AD Rast, J Navaridas, X Jin, LA Plana, M Luján, S Temple, C Patterson, D Richards, J V Woods, J Miguel-Alonso, and SB Furber. Event-Driven Configuration of a Neural Network CMP System over an Homogeneous Interconnect Fabric. *Parallel Computing*, 30(7):435–444, November 2010.
- EI Knudsen. Fundamental components of attention. *Annu. Rev. Neurosci.*, 30: 57–78, 2007. URL <http://www.annualreviews.org/doi/pdf/10.1146/annurev.neuro.30.051606.094256>.
- J Kremkow, A Aertsen, and A Kumar. Gating of Signal Propagation in Spiking Neural Networks by Balanced and Correlated Excitation and Inhibition. *Journal of Neuroscience*, 30(47):15760–15768, 2010. ISSN 02706474. doi: 10.1523/JNEUROSCI.3874-10.2010. URL <http://www.jneurosci.org/cgi/doi/10.1523/JNEUROSCI.3874-10.2010>.
- PD Kuo and C Eliasmith. Integrating behavioral and neural data in a model of zebrafish network interaction. *Biological Cybernetics*, 93(3):178–187, 2005. URL <http://www.ncbi.nlm.nih.gov/pubmed/16136350>.
- J Lazzaro, J Wawrzynek, M Mahowald, M Silviotti, and D Gillespie. Silicon Auditory Processors as Computer Peripherals. *IEEE Transactions on Neural Networks*, 4(3): 523–528, May 1993.
- Y LeCun and Y Bengio. Convolutional Networks for Images, Speech, and Time-Series. In M A Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.
- VW Lee, C Kim, J Chhugani, M Deisher, D Kim, Anthony D Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. Debunking the 100X GPU vs. CPU myth: an evaluation

- of throughput computing on CPU and GPU. In *Computer Architecture, International Symposium on*, pages 451–460, 2010.
- JA Lenero-Bardallo, T Serrano-Gotarredona, and B Linares-Barranco. A 3.6us Latency Asynchronous Frame-Free Event-Driven Dynamic-Vision-Sensor, 2011. ISSN 00189200. URL [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=5746543](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5746543).
- FL Lewis and EW Kamen. Applied optimal control and estimation. *IEEE Transactions on Automatic Control*, 39(8):1773–1773, 1994. URL <http://arri.uta.edu/acs/ee4343ee5320/lectures99/OPfeedback.pdf>.
- P Lichtsteiner, C Posch, and T Delbruck. A 128x128 120dB 15us Latency Asynchronous Temporal Contrast Vision Sensor. *IEEE Journal of Solid State Circuits*, 43:566–576, 2008.
- J Lin, P Merolla, J Arthur, and K Boahen. Programmable Connections in Neuromorphic Grids. In *49th IEEE Midwest Symposium on Circuits and Systems*, pages 80–84, 2006.
- B G Lindsey, K F Morris, R Shannon, and G L Gerstein. Repeated patterns of distributed synchrony in neuronal assemblies. *Journal of Neurophysiology*, 78(3):1714–1719, September 1997.
- M London, A Roth, L Beeren, M Häusser, and PE Latham. Sensitivity to perturbations in vivo implies high noise and suggests rate coding in cortex. *Nature*, 466(7302):123–127, 2010. URL <http://www.nature.com/doifinder/10.1038/nature09086>.
- W Maass. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural Networks*, 10(9):1659–1671, December 1996. ISSN 08936080. doi: 10.1016/S0893-6080(97)00011-7. URL [http://dx.doi.org/10.1016/S0893-6080\(97\)00011-7](http://dx.doi.org/10.1016/S0893-6080(97)00011-7).
- W Maass. Noisy spiking neurons with temporal coding have more computational power than sigmoidal neurons. *Advances in Neural Information Processing Systems*, 9(211-217), 1997.
- W Maass. Neural computation with winner-take-all as the only nonlinear operation. *Advances in neural information processing systems*, 12:293–299,

1999. URL <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.33.2941&rep=rep1&type=pdf>.
- W Maass and H Markram. On the computational power of circuits of spiking neurons. *Journal of computer and system sciences*, 69(4):593–616, 2004. URL <http://www.sciencedirect.com/science/article/pii/S0022000004000406>.
- W Maass, G Schnitger, and E Sontag. On the computational power of sigmoid versus boolean threshold circuits. In *32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 767–776, 1991.
- W Maass, T Natschläger, and H Markram. Computational models for generic cortical microcircuits. *Computational neuroscience A comprehensive approach*, 18:575–605, 2003. URL <http://www.igi.tugraz.at/maass/psfiles/149-v05.pdf>.
- LP Maguire, TM McGinnity, BP Glackin, A Ghani, A Belatreche, and Jim Harkin. Challenges for large-scale implementations of spiking neural networks on FPGAs. *Neurocomputing*, 71(1-3):13–29, December 2007.
- M Mahowald. *VLSI analogs of neuronal visual processing: a synthesis of form and function*. PhD thesis, California Inst. Tech., Pasadena, CA, 1992.
- BQ Mao, F Hamzei-Sichani, D Aronov, RC Froemke, and R Yuste. Dynamics of spontaneous activity in neocortical slices. *Neuron*, 32(5):883–898, December 2001.
- H Markram. The blue brain project. *Nature Reviews Neuroscience*, 7:153–160, 2006.
- W Mcculloch and W Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133, December 1943. doi: 10.1007/BF02478259. URL <http://dx.doi.org/10.1007/BF02478259>.
- C Mead. *Analog VLSI and neural systems*. Addison-Wesley Longman Publishing Co., Inc., 1989.
- C Mead. Neuromorphic electronic systems, 1990. ISSN 00189219. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=58356>.
- P Merolla, J Arthur, F Akopyan, N Imam, R Manohar, and Dharmendra S Modha. A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. *Custom Integrated Circuits Conference (CICC), 2011 IEEE*, pages 1–4, 2011. ISSN 0886-5930. doi: 10.1109/CICC.2011.6055294.

- PA Merolla, JV Arthur, BE Shi, and KA Boahen. Expandable Networks for Neuro-morphic Chips. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(2): 301–311, February 2007. ISSN 1057-7122. doi: 10.1109/TCSI.2006.887474. URL [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=4089120](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4089120).
- M Migliore, T M Morse, A P Davison, L Marenco, G M Shepherd, and M L Hines. ModelDB: making models publicly accessible to support computational neuroscience. *Neuroinformatics*, pages 1(1):135–9, 2003.
- E K Miller and J D Cohen. An integrative theory of prefrontal cortex function. *Annual Review of Neuroscience*, 24:167–202, 2001. doi: 10.1146/annurev.neuro.24.1.167. URL <http://dx.doi.org/10.1146/annurev.neuro.24.1.167>.
- S Millner, A Grubl, K Meier, J Schemmel, and MO Schwartz. A VLSI Implementation of the Adaptive Exponential Integrate-and-Fire Neuron Model. *Advances in Neural Information Processing Systems*, 23:1642–1650, 2010.
- SW Moore, PJ Fox, SJT Marsh, AT Markettos, and A Mujumdar. Bluehive - A Field-Programmable Custom Computing Machine for Extreme-Scale Real-Time Neural Network Simulation. *2012 IEEE 20th International Symposium on FieldProgrammable Custom Computing Machines*, pages 133–140, 2012. doi: 10.1109/FCCM.2012.32. URL [http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6239804&contentType=Conference+Publications&searchField=Search\\_All&queryText=bluehive](http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6239804&contentType=Conference+Publications&searchField=Search_All&queryText=bluehive).
- RJ Moran, M Symmonds, KE Stephan, KJ Friston, and RJ Dolan. An In Vivo Assay of Synaptic Function Mediating Human Cognition. *Current Biology*, 21(15):1320–1325, 2011. URL <http://discovery.ucl.ac.uk/1318958/>.
- A Morrison, C Mehring, T Geisel, AD Aertsen, and M Diesmann. Advancing the boundaries of high-connectivity network simulation with distributed computing. *Neural computation*, 17(8):1776–801, August 2005. ISSN 0899-7667. doi: 10.1162/0899766054026648. URL <http://www.mitpressjournals.org/doi/abs/10.1162/0899766054026648>.
- JM Nageswaran, N Dutt, JL Krichmar, and A Nicolau. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks*, 22(5–6), 2007. URL [http://www.sciencedirect.com/science/article/pii/S0893-6080\(09\)00137-3](http://www.sciencedirect.com/science/article/pii/S0893-6080(09)00137-3).

- E Nordlie, MO Gewaltig, and HE Plesser. Towards reproducible descriptions of neuronal network models. *PLoS computational biology*, 2009. URL <http://dx.plos.org/10.1371/journal.pcbi.1000456>.
- B Noudoost, M H Chang, N A Steinmetz, and T Moore. Top-down control of visual attention. *Current opinion in neurobiology*, 20(2):183–190, 2010.
- J O’Keefe and J Dostrovsky. The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34(1):171–175, 1971. URL <http://discovery.ucl.ac.uk/95537/>.
- J O’Keefe and ML Recce. Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus*, 3(3):317–330, 1993.
- M Oster, Yingxue Wang Yingxue Wang, R Douglas, and Shih-Chii Liu Shih-Chii Liu. Quantification of a Spike-Based Winner-Take-All VLSI Network, 2008. ISSN 15498328. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4490297>.
- E Painkras, LA Plana, J Garside, S Temple, F Gallupi, C Patterson, DR Lester, AD Brown, and SB Furber. SpiNNaker : A 1W 18-core System-on-Chip for Massively-Parallel Neural Net Simulation. *The IEEE Journal of Solid State Circuits*, VV:1–13, 2012.
- DL Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972. ISSN 00010782. doi: 10.1145/361598.361623. URL <http://portal.acm.org/citation.cfm?doid=361598.361623>.
- C Patterson, F Gallupi, A Rast, and SB Furber. Visualising large-scale neural network models in real-time. *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8, 2012a. URL [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=6252490](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6252490).
- C Patterson, J Garside, and E. Painkras. Scalable communications for a million-core neural processing architecture. *Journal of Parallel Distributed Computing*, 72(11):1507–1520, 2012b. URL <http://www.sciencedirect.com/science/article/pii/S0743731512000287>.

- T Pfeil, A Grübl, S Jeltsch, E Müller, P Müller, MA Petrovici, M Schmuker, D Brüderle, J Schemmel, and M Karlheinz. Six networks on a universal neuromorphic computing substrate. *Frontiers in neuroscience*, 7(11), 2013. URL <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3575075/>.
- L Plana, S Furber, S Temple, M Khan, Y Shi, J Wu, and S Yang. A GALS Infrastructure for a Massively Parallel Multiprocessor. *IEEE Design & Test of Computers*, 24(5): 454–463, 2007.
- T A Plate. Holographic reduced representations. *IEEE Transactions on Neural Networks*, 6(3):623–641, 1995. ISSN 10459227. doi: 10.1109/72.377968. URL <http://www.ncbi.nlm.nih.gov/pubmed/18263348>.
- HE Plesser, JM Eppler, A Morrison, M Diesmann, and MO Gewaltig. Efficient parallel simulation of large-scale neuronal networks on clusters of multiprocessor computers. In *Proc. 13th Int’l Euro-Par Conf. on Parallel Processing (Euro-Par 2007)*, pages 672–681. Springer, 2007. URL <http://www.springerlink.com/index/RW27162113P7L685.pdf>.
- F Ponulak. Supervised learning in spiking neural networks. *The Neuromorphic Engineer*. URL <http://www.ine-news.org/pdf/0045/0045.pdf>.
- C Posch, D Matolin, and R Wohlgenannt. A QVGA 143dB dynamic range asynchronous address-event PWM dynamic image sensor with lossless pixel-level video compression. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 400–401. IEEE, 2010.
- W Potjans, M Diesmann, and A Morrison. An Imperfect Dopaminergic Error Signal Can Drive Temporal-Difference Learning. *PLoS Computational Biology*, 7(5): 20, 2011. URL <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3093351&tool=pmcentrez&rendertype=abstract>.
- R Quiñan Quiroga, L Reddy, G Kreiman, C Koch, and I Fried. Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045):1102–1107, 2005. URL <http://dx.doi.org/10.1038/nature03687>.
- I Raikov, R Cannon, and R Clewley. NineML: the network interchange for neuroscience modeling language. *BMC Neuroscience*, 2(Suppl-1):P330, 2011. URL <http://www.springerlink.com/index/M82T314U31RQX185.pdf>.

- AD Rast, F Galluppi, X Jin, and SB Furber. The Leaky Integrate-and-Fire neuron: A platform for synaptic model exploration on the SpiNNaker chip. In *Neural Networks IJCNN The 2010 International Joint Conference on*, pages 18–23. IEEE, 2010a. ISBN 9781424469161. doi: 10.1109/IJCNN.2010.5596364. URL [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=5596364](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5596364).
- AD Rast, X Jin, F Galluppi, LA Plana, C Patterson, and SB Furber. Scalable event-driven native parallel processing: The spinnaker neuromimetic system. In *Proceedings of the 7th ACM international conference on Computing frontiers*, pages 21–30. ACM, 2010b. ISBN 9781450300445. URL <http://portal.acm.org/citation.cfm?id=1787279>.
- AD Rast, F Galluppi, and S Davies. An event-driven model for the SpiNNaker virtual synaptic channel. *The 2011 International Joint Conference on Neural Networks (IJCNN)*, 2011a. URL [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=6033466](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6033466).
- AD Rast, F Galluppi, S Davies, LA Plana, C Patterson, T Sharp, DR Lester, and SB Furber. Concurrent heterogeneous neural model simulation on real-time neuromimetic hardware. *Neural Networks*, 24:961–978, 2011b. ISSN 0893-6080. doi: 10.1016/j.neunet.2011.06.014. URL <http://www.sciencedirect.com/science/article/pii/S0893608011001742>.
- AD Rast, J Navaridas, X Jin, F Galluppi, LA Plana, J Miguel-Alonso, C Patterson, M Lujan, and S Furber. Managing Burstiness and Scalability in Event-Driven Models on the SpiNNaker Neuromimetic System. *International Journal of Parallel Programming*, 2011c. URL <http://personalpages.manchester.ac.uk/staff/javier.navaridas/pubs/ijpp11.pdf><http://www.springerlink.com/index/6490Q26M7034P473.pdf>.
- AD Rast, J Partzsch, C Meyr, S Hartmann, LA Plana, S Temple, and DR Lester. A Location-Independent Direct Link Neuromorphic Interface. In *Submitted to the International Joint Conference on Neural Networks - IJCNN 213*, 2013.
- JE Raymond, KL Shapiro, and KM Arnell. Temporary suppression of visual processing in an RSVP task: an attentional blink? *Journal of Experimental Psychology: Human Perception and Performance*, 18(3):849–860, 1992. URL <http://www.ncbi.nlm.nih.gov/pubmed/1500880>.

- P Redgrave, T J Prescott, and K Gurney. The basal ganglia: a vertebrate solution to the selection problem? *Neuroscience*, 89(4):1009–1023, 1999.
- LM Reyneri. Implementation issues of neuro-fuzzy hardware: going toward HW/SW codesign. *IEEE Transactions on Neural Networks*, 14(1):176–194, 2003. ISSN 10459227. doi: 10.1109/TNN.2002.806955. URL <http://www.ncbi.nlm.nih.gov/pubmed/18238000>.
- M Riesenhuber and T Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.
- Crook S, Beeman D, and Gleeson Fred Howell. XML for Model Specification in Neuroscience. *Brains, Minds and Media*, 1(2), 2005.
- J Schemmel, D Brüderle, A Grübl, M Hock, K Meier, and S Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of the 2010 International Symposium on Circuit and Systems (ISCAS2010)*, pages 1947–1950, 2010.
- TJ Sejnowski. The computational self. *Annals Of The New York Academy Of Sciences*, 1001(1):262–271, 2003. URL <http://www.ncbi.nlm.nih.gov/pubmed/14625366>.
- TJ Sejnowski, C Koch, and PS Churchland. Computational neuroscience. *Science*, 241(4871):1299–1306, September 1988. doi: 10.1126/science.3045969. URL <http://dx.doi.org/10.1126/science.3045969>.
- B Sen and SB Furber. Information recovery from rank-order encoded images. In *University of Surrey*, 2006.
- R Serrano-Gotarredona, T Serrano-Gotarredona, A Acosta-Jimenez, and B Linares-Barranco. A Neuromorphic Cortical-Layer Microchip for Spike-Based Event Processing Vision Systems, 2006. ISSN 15498328. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4026698>.
- R Serrano-Gotarredona, M Oster, P Lichtsteiner, A Linares-Barranco, R Paz-Vicente, F Gomez-Rodriguez, L Camunas-Mesa, R Berner, M Rivas-Perez, T Delbruck, SC Liu, RJ Douglas, P Hafliger, G Jimenez-Moreno, A Civit Ballcells, T Serrano-Gotarredona, AJ Acosta-Jimenez, and B Linares-Barranco. CAVIAR: a 45k neuron, 5M synapse, 12G connects/s AER hardware sensory-processing- learning-actuating system for high-speed visual object recognition and tracking. *IEEE*

- Transactions on Neural Networks*, 20(9):1417–1438, 2009. URL <http://www.ncbi.nlm.nih.gov/pubmed/19635693>.
- MN Shadlen and JA Movshon. Synchrony unbound: a critical evaluation of the temporal binding hypothesis. *Neuron*, 24(1):67–77,111–125, September 1999.
- T Sharp, LA Plana, F Galluppi, and SB Furber. Event-Driven Simulation of Arbitrary Spiking Neural Networks on SpiNNaker. In *The International Conference on Neural Information Processing (ICONIP)*, volume 2011, pages 424–430. Springer, 2011. ISBN 3642249647. URL <http://books.google.com/books?id=cU4skcyGRFUC&pgis=1>.
- T Sharp, F Galluppi, AD Rast, and SB Furber. Power-efficient simulation of detailed cortical microcircuits on SpiNNaker. *The Journal of Neuroscience Methods*, March 2012. ISSN 1872-678X. doi: 10.1016/j.jneumeth.2012.03.001. URL <http://www.ncbi.nlm.nih.gov/pubmed/22465805>.
- W Singer and C M Gray. Visual Feature Integration and the Temporal Correlation Hypothesis. *Annual Review of Neuroscience*, 18:555–586, 1995. ISSN 0147-006X.
- R Singh and C Eliasmith. Higher-dimensional neurons explain the tuning and dynamics of working memory cells. *Journal of Neuroscience*, 26(14):3667–3678, 2006. URL <http://www.ncbi.nlm.nih.gov/pubmed/16597721>.
- S Song, K D Miller, and L F Abbott. Competitive Hebbian learning through spike-timing-dependent synaptic plasticity. *Nature Neuroscience*, 3(9):919–926, September 2000. doi: 10.1038/78829. URL <http://dx.doi.org/10.1038/78829>.
- D Sonnleithner and G Indiveri. A Neuromorphic Saliency-Map based Active Vision System. In *Neuroinformatics*, pages 1–6. IEEE, 2011. ISBN 9781424498482. doi: 10.1109/CISS.2011.5766145. URL [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=5766145](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5766145).
- O Sporns, JA Gally, GN Reeke, and GM Edelman. Reentrant signaling among simulated neuronal groups leads to coherency in their oscillatory activity. *Proceedings of the National Academy of Sciences of the United States of America*, 86(18):7265–7269, September 1989.
- O Sporns, N Almássy, and GM Edelman. Plasticity in value systems and its role in

- adaptive behaviour. *Adaptive behaviour*, 8(2):129–148, March 2000. ISSN 1059-7123. doi: 10.1177/105971230000800203. URL <http://adb.sagepub.com/cgi/doi/10.1177/105971230000800203>.
- D Steinkraus, I Buck, and P Y Simard. Using GPUs for machine learning algorithms. In *Proceedings of the 8th International Conference on Document Analysis and Recognition*, pages 1115–1120, 2005.
- TC Stewart and C Eliasmith. Neural Symbolic Decision Making: A Scalable and Realistic Foundation for Cognitive Architectures. In *First International Conference on Biologically Inspired Cognitive Architectures*, pages 147–152, 2010.
- TC Stewart and C Eliasmith. Neural cognitive modelling: a biologically constrained spiking neuron model of the Tower of Hanoi task. *Proceeding of the 33rd Annual Meeting of the Cognitive Science Society*, pages 656–661, 2011. URL <http://csjarchive.cogsci.rpi.edu/proceedings/2011/papers/0125/paper0125.pdf>  
<http://mindmodeling.org/cogsci2011/papers/0125/paper0125.pdf>.
- TC Stewart, B Tripp, and C Eliasmith. Python scripting in the Nengo simulator. *Frontiers in Neuroinformatics*, 3(0), 2009. ISSN 1662-5196. doi: 10.3389/neuro.11.007.2009. URL [http://www.frontiersin.org/Journal/Abstract.aspx?s=752&name=neuroinformatics&ART\\_DOI=10.3389/neuro.11.007.2009](http://www.frontiersin.org/Journal/Abstract.aspx?s=752&name=neuroinformatics&ART_DOI=10.3389/neuro.11.007.2009).
- TC Stewart, X Choo, and C Eliasmith. Dynamic behaviour of a spiking model of action selection in the basal ganglia. In *Proceedings of the 10th international conference on cognitive modeling*, pages 235–240, 2010a.
- TC Stewart, X Choo, and C Eliasmith. Symbolic reasoning in spiking neurons: A model of the cortex/basal ganglia/thalamus loop. In *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*, pages 1100–1105, 2010b.
- TC Stewart, T Bekolay, and C Eliasmith. Neural representations of compositional structures: representing and manipulating vector spaces with spiking neurons. *Connection Science*, 23(2):145–153, 2011. URL <http://www.tandfonline.com/doi/abs/10.1080/09540091.2011.571761>.
- Terrence C Stewart, Feng-Xuan Choo, and Chris Eliasmith. Spaun: A Perception-Cognition-Action Model Using Spiking Neurons. In Naomi Miyake, David Peebles,

- and Richard P Cooper, editors, *Proceedings of the 34th Annual Meeting of the Cognitive Science Society CogSci 2012*, pages 1018–1023. Cognitive Science Society, 2012. URL <http://palm.mindmodeling.org/cogsci2012/papers/0184/paper0184.pdf>.
- E Stromatias, F Galluppi, C Patterson, and SB Furber. Power analysis of large-scale, real-time neural networks on SpiNNaker. In *Submitted to the International Joint Conference on Neural Networks - IJCNN 213*, 2013.
- IV Tetko and AE Villa. A pattern grouping algorithm for analysis of spatiotemporal patterns in neuronal spike trains. 1. Detection of repeated patterns. *Journal of Neuroscience Methods*, 105(1):1–14, 2001. ISSN 01650270. URL <http://www.ncbi.nlm.nih.gov/pubmed/11166361>.
- RF Thompson. *Foundations of physiological psychology*. Harper’s physiological psychology series. Harper & Row, 1967. URL <http://books.google.co.uk/books?id=Z3sfAAAAMAAJ>.
- AM Thomson and C Lamy. Functional maps of neocortical local circuitry. *Frontiers in neuroscience*, 1(1):19–42, 2007. URL <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2518047/>.
- S Thorpe and J Gautrais. Rank order coding. In *CNS ’97: Proceedings of the sixth annual conference on Computational neuroscience : trends in research, 1998*, pages 113–118, New York, NY, USA, 1998. Plenum Press. ISBN 0-306-45919-1. URL <http://portal.acm.org/citation.cfm?id=298181>.
- S Thorpe, D Fize, and C Marlot. Speed of processing in the human visual system. *nature*, 381:520–522, 1996. URL <http://fias.uni-frankfurt.de/~triesch/courses/260object/papers/SpeedOfProcessing.pdf>.
- S Thorpe, A Delorme, and R van Rullen. Spike-based strategies for rapid processing. *Neural Networks*, 14(6-7):715–725, 2001.
- S Thorpe, R Guyonneau, N Guilbaud, J Allegraud, and R Vanrullen. SpikeNet: real-time visual processing with one spike per neuron. *Neurocomputing*, 58-60(5690): 857–864, 2004. ISSN 09252312. doi: 10.1016/j.neucom.2004.01.138. URL <http://linkinghub.elsevier.com/retrieve/pii/S0925231204001432>.
- EC Tolman. Cognitive maps in rats and men. *Psychological Review*, 55(4):189–208, 1948. URL <http://www.ncbi.nlm.nih.gov/pubmed/9230435>.

- G Tononi, O Sporns, and G M Edelman. Reentry and the problem of integrating multiple cortical areas: simulation of dynamic integration in the visual system. *Cerebral Cortex*, 2(4):310–335, 1992.
- G Tononi, O Sporns, and G M Edelman. A measure for brain complexity: relating functional segregation and integration in the nervous system. *Proceedings of the National Academy of Sciences*, 91(11):5033–5037, 1994.
- G Tononi, G M Edelman, and O Sporns. Complexity and coherency: integrating information in the brain. *Trends in cognitive sciences*, 2(12):474–484, 1998. URL [http://dx.doi.org/10.1016/S1364-6613\(98\)01259-5](http://dx.doi.org/10.1016/S1364-6613(98)01259-5).
- AHT Tse, DB Thomas, and W Luk. Accelerating Quadrature Methods for Option Valuation. *2009 17th IEEE Symposium on Field Programmable Custom Computing Machines*, 29:29–36, 2009. doi: 10.1109/FCCM.2009.36. URL <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5290956>.
- D Vainbrand and R Ginosar. Scalable network-on-chip architecture for configurable neural networks. *Microprocessors and Microsystems*, 35(2):152–166, March 2011. ISSN 01419331. doi: 10.1016/j.micpro.2010.08.005. URL <http://linkinghub.elsevier.com/retrieve/pii/S0141933110000517>.
- J Van Leeuwen. Interval Routing. *The Computer Journal*, 30(4):298–307, April 1987. ISSN 0010-4620. doi: 10.1093/comjnl/30.4.298. URL <http://comjnl.oxfordjournals.org/cgi/content/abstract/30/4/298>.
- R Van Rullen and S J Thorpe. Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex. *Neural Computation*, 13(6):1255–1283, 2001. URL <http://www.ncbi.nlm.nih.gov/pubmed/11387046>.
- R Van Rullen, J Gautrais, A Delorme, and S Thorpe. Face processing using one spike per neurone. In *Biosystems*, volume 48, pages 229–239, November 1998. doi: 10.1016/S0303-2647(98)00070-7. URL [http://dx.doi.org/10.1016/S0303-2647\(98\)00070-7](http://dx.doi.org/10.1016/S0303-2647(98)00070-7).
- A van Schaik and SC Liu. AER EAR: A matched silicon cochlea pair with address event representation interface. In *IEEE International Symposium on Circuits and Systems ISCAS 2005*, pages 4213–4216, 2005. URL <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.63.5782>.

- TP Vogels and LF Abbott. Signal propagation and logic gating in networks of integrate-and-fire neurons. *J Neurosci*, 25(46):10786–10795, November 2005. doi: 10.1523/JNEUROSCI.3508-05.2005. URL <http://dx.doi.org/10.1523/JNEUROSCI.3508-05.2005>.
- RJ Vogelstein, U Mallik, E Culurciello, G Cauwenberghs, and R Etienne-Cummings. A multichip neuromorphic system for spike-based visual information processing. *Neural computation*, 19(9):2281–300, September 2007. ISSN 0899-7667. doi: 10.1162/neco.2007.19.9.2281. URL <http://www.mitpressjournals.org/doi/abs/10.1162/neco.2007.19.9.2281>.
- C Vreeswijk, LF Abbott, and G Bard Ermentrout. When inhibition not excitation synchronizes neural firing. *Journal of Computational Neuroscience*, 1(4):313–321, 1994. ISSN 0929-5313. URL <http://dx.doi.org/10.1007/BF00961879>.
- JHB Wijekoon and P Dudek. Integrated Circuit Implementation of a Cortical Neuron. In *Proceedings of the 2008 International Symposium on Circuits and Systems (ISCAS 2008)*, pages 1784–1787, 2008.
- JHB Wijekoon and P Dudek. VLSI circuits implementing computational models of neocortical circuits. *Journal of neuroscience methods*, February 2012. ISSN 1872-678X. doi: 10.1016/j.jneumeth.2012.01.019. URL <http://dx.doi.org/10.1016/j.jneumeth.2012.01.019>.
- TJ Wills, C Lever, F Cacucci, N Burgess, and J O’Keefe. Attractor dynamics in the hippocampal representation of the local environment. *Science*, 308(5723):873, 2005. URL <http://www.sciencemag.org/content/308/5723/873.short>.
- P Yger, D Bruderle, J Eppler, J Kremkow, D Pecevski, LU Perrinet, M Schmuker, E Muller, and AP Davison. NeuralEnsemble: Towards a meta-environment for network modeling and data analysis. *Eighth Göttingen Meeting of the German Neuroscience Society*, (T26-4C):T26–4C, 2009.