KTH Computer Science
and Communication

Large-scale Simulation of Neuronal Systems

MIKAEL DJURFELDT

Doctoral Thesis
Stockholm, Sweden 2009

TRITA-CSC-A 2009:06
ISSN-1653-5723
ISRN-KTH/CSC/A--09/06--SE
ISBN 978-91-7415-323-1

KTH School of Computer Science and Communication
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framläggs till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi tisdagen den 9 juni 2009 klockan 10.00 i sal F2, Kungl Tekniska högskolan, Lindstedtsvägen 26, Stockholm.

Tryck: Universitetsservice US AB

To my parents

Abstract

This thesis provides conceptual, mathematical and software methods and tools to enable and facilitate the simulation of large-scale neuronal systems on supercomputers. A perspective on the role of large-scale models in neuroscience is given and a terminology for classifying models of neuronal networks is proposed. A novel formalism for the description of connectivity of neuronal network models is presented. This formalism, the connection-set algebra, can be used both to provide concise and unambiguous descriptions of connectivity in papers in computational neuroscience, and as a component of simulator scripting languages. Two different approaches to modularity when simulating systems of networks are provided in the See simulator and in the MUSIC API and library. A parallel simulation library for neuronal network models, SPLIT, is improved and extended to handle large-scale models. Using this library, a neuronal network model of layers II/III of the neocortex, based on biophysical model neurons is simulated. Several key phenomena seen in the living brain appear as emergent phenomena in the simulations. The memory capacity of two models of different network size is measured and compared to that of an artificial neural network. We conclude that the layer II/III model performs as a robust auto-associative memory. Furthermore the model is robust against perturbation of parameters, which is a hallmark of correct models of living systems.

Sammanfattning

Denna avhandling tillhandahåller konceptuella, matematiska och programvarumässiga metoder och verktyg för att möjliggöra och underlätta simulering av storskaliga neuronala system på superdatorer. Avhandlingen ger ett perspektiv på rollen av storskaliga modeller i neurovetenskapen och föreslår en terminologi för klassificering av modeller av neuronala nätverk. En ny formalism för beskrivning av konnektivitet i sådana modeller presenteras. Denna formalism, kopplingsmängdsalgebran, kan användas både för att tillhandahålla koncisa och entydiga beskrivningar av konnektivitet i vetenskapliga artiklar inom beräkningsneurobiologi och som en komponent i skriptspråk för simulatorer. Två olika angreppssätt för att åstadkomma modularitet vid simulering av system av nätverk presenteras, simulatorn See och API-standarden och programvaran MUSIC. En parallel simuleringsprogramvara för modeller av neuronala nätverk, SPLIT, förbättras och utökas så att storskaliga modeller kan hanteras. Denna programvara används för att simulera en neuronnätsmodell av lager II/III i neokortex, baserad på biofysikaliska neuronmodeller. Flera karaktäristiska fenomen som förekommer i den levande hjärnan uppkommer också i dessa simuleringar. Minneskapaciteten för två modeller av olika storlek uppmäts och jämförs med motsvarande mätningar för ett artificiellt neuronnät. Slutsatsen dras att modellen av lager II/III fungerar som ett robust autoassociativt minne. Vidare är modellen robust vid störning av dess parametrar. Detta är ett kännetecken för korrekta modeller av levande system.

Acknowledgements

Many thanks to my main supervisor Örjan Ekeberg, my second supervisor Anders Lansner and all others at the department for Computational Biology at KTH, Stockholm. Some of the work presented here was done in collaboration with Mikael Lundqvist, Johannes Hjorth, Martin Rehn, Christopher Johansson, Anders Sandberg, Moritz Helias, Jochen Eppler, Tobias Potjans, Markus Diesmann, Niraj Dudani, Upinder S. Bhalla and Jeanette Hellgren. I thank NADA/CSC, SANS/CB and Anders Lansner for providing a place for me, and resources to complete this thesis, and would like to direct special thanks to Erik Aurell and Ingrid Melinder for support in times of need.

The work on this thesis has been performed during an extended period where I've been involved in several projects: the development of the GNU Guile Scheme interpreter and the See simulator, studies of the basal ganglia with Ann Graybiel at MIT, tool development and demos in preparation for the acquisition of the Blue Gene/L supercomputer Hebb, and work in the MUSIC and FACETS projects, to name the most important ones. During this time, and before, I've met many people who have taught me, shaped my thinking, and/or been important in other ways. I would like to thank Tomas Hökfelt, Joacim Halén, Richard M. Stallman, Gerald Jay Sussman, Jim Blandy, Marius Vollmer, Ann Graybiel, Yasuo Kubota, Naotaka Fujii, Richard Courtemanche, Bill DeCoteau, Carl G. Tengwall, Andrew Davison, and many more whom I've arbitrarily omitted.

Örjan Ekeberg has not only given me unfailing support, shaped and strengthened my thinking, given the best advice and deepest insights on countless occasions, but has also been a very good problem solving partner in many, many instances, especially during our joint design of MUSIC, which was a joy.

I thank Gösta Fröléen for important discussions and influences of classical and empirical thinking, Per Larsson and Anders Holst for all great discussions on metaphysics and physics, Roland Orre for introducing me to the higher levels of hardware and software and for being a kind and good friend, Fredrik Ullén for all inspiration and for being a good friend, Erik Fransén for a sincere attitude towards science and for being a trustworthy friend with good humor, Johannes Hjorth for brightness and humor, Anders Sandberg for scolarly insanity and humor, and Peter Lönnerberg for being the master of humor and the best of friends. I could go on forever praising all of you good friends and colleagues—please take no offense those of you not mentioned.

This thesis could not have been made without the help of Alex and Alfred—special thanks for taking care of your sister on those many occasions. Thanks also for being such nice and friendly guys! Thanks to Sara for being so bright and lovely, and easy and fun to be with. Finally, warm thanks to Diana for patience and support, especially during the final weeks when this thesis was compiled.

This work was partly supported by KTH (doktorandtjänst), INCF (International Neuroinformatics Coordinating Facility), the Swedish Science Council (Vetenskapsrådet, VR-621-2004-3807), and by the European Union (FP6-2004-IST-FETPI-015879, FP7-HEALTH-2007-A-201716).

Contents

List of Figures

Chapter 1

Introduction

In our quest to understand the human mind, this very mind presents some obstacles. The mind arises through the processes of our brain, a physical entity, and is thus constrained by physical limitations. Just as our immune system, although constrained by the physics of our bodies, still is prepared to recognize *any* pathogen, our mind may well be prepared to tackle any scientific question. This comes at a cost, however. Just as our immune system achieves its remarkable universality by compensating its lack of physical resources with a sufficiently low specificity of its antibodies (Edelman, 1987), we do not perceive the world as it is, but clad in human categories, divided into the bits and pieces, and using the strategies, which were relevant for guiding action in the ecological niche where we evolved.

How is it possible for an entity which cannot hold more than seven objects at a time in its working memory (Miller, 1956) to understand a system as complex as the brain? The answer is *technology*—the technology of reason (see, e.g., Plato, 387–347 BCE; Aristotle, 350 BCE), the technology of our scientific methods and the technology of our tools. This thesis aims to provide conceptual, mathematical and software methods and tools to enable and facilitate the simulation of large-scale neuronal systems on supercomputers.

The field of computational neuroscience has its roots in seminal works of individuals like Hebb (1949), Hodgkin and Huxley (1952), Rall (1959), Hubel and Wiesel (1962) and Marr (1969, 1982) and in the field of neural networks. This field, in turn, shares its roots with the field of computer science in the 1940's and 50's with works such as McCulloch and Pitts (1943) and Papert (1960). Hodgkin and Huxley (1952) described the interaction of ion channels in a patch of neuronal membrane using a set of coupled non-linear ordinary differential equations and could explain and quantitatively reproduce the basic features of the action potential. Via the equivalent cylinder model of the dendrite (Rall, 1959), developments in computational neuroscience led to the first descriptions of the dynamics of neocortical neurons (see, e.g., Traub, 1979). The speed of computation of computers did not allow

simulation of detailed models of neuronal networks until during the 1980's during which the first neuronal network simulation programs were developed (e.g., Ekeberg et al., 1993; Hines, 1993; Bower and Beeman, 1998). Here, *detailed* means models based on multi-compartmental units with Hodgkin-Huxley dynamics. The SPLIT simulator (Hammarlund and Ekeberg, 1998) was the first parallelized software for simulation of detailed models of neuronal networks. SPLIT made it possible to run such simulations on supercomputers, such as vector machines and cluster computers, thereby enabling the simulation of larger network models. This paved the way for the first biologically detailed model of a cortical associative memory (Fransén and Lansner, 1998).

As the system under study, and its model, become more complex, we need techniques to handle this complexity. At the conceptual level, in mathematics as well as in software, *abstraction* and the strategy of *divide-and-conquer* are the main enabling tools. In this thesis, we will use these tools at all three levels to make the simulation of a large and complex model of cortical layers II/III (chapter 5) feasible.

1.1 Overview

The thesis begins with a discussion of mathematical modeling in neuroscience in chapter 2. Building on the works of Marr (1982) and Churchland and Sejnowski (1992), this chapter presents a terminology for certain properties which distinguishe models of neural systems, such as abstraction, degree of detail, realism and explicitness. It concludes with the presentation of a novel formalism for the description of connectivity in neuronal network models in section 2.11. Chapter 3 provides a review of issues related to simulation of neuronal network models and a discussion of simulation tools. This is followed by chapter 4 which begins with the presentation of two different kinds of modular frameworks, the See simulator and MUSIC, which can be used to simulate systems of networks, and concludes with a description of the SPLIT simulator for the simulation of very large neuronal networks. In chapter 5, this technology is applied in the simulation of a large-scale model of cortical layers II/III. The thesis is concluded with an outlook in chapter 6, discussing future challenges in the field and conclusions in chapter 7.

1.2 Contributions of this thesis

- I discuss the role of mathematical modeling in computational neuroscience. (Chapter 2)
- I present a perspective on the role of large-scale modeling in computational neuroscience. (Papers **I**, **VI**, Chapter 2.)
- I present a novel formalism for the representation of connectivity patterns in neuronal network models—the *connection set algebra*. (Paper **II**.)
- I present an analysis of software for simulation of large-scale neuronal network models. (Chapter 3)
- In the See simulator (Paper **III**) I provide an early example of the use of a general purpose scripting language for model description and simulator extension.
- In the See simulator (Paper **III**), its modular framework is an early example of a methodology which has later been absorbed by simulators such as Genesis3 and MOOSE.
- I design and implement software which allows parallel applications to communicate massive amounts of data efficienctly within a cluster computer. (Paper **IV**.)
- I design and implement a framework which enables software modularity and re-usability in the domain of parallel neuronal network simulators and associated tools. (Paper **IV**.)
- I demonstrate that very large scale biologically detailed models of brain networks are technologically feasible. (Papers **V**, **VI**.)
- I demonstrate that a large-scale detailed neuronal network model of cortical layers II/III model displays ground state in addition to the active state and that these network states are associated with population-level rythms in the α and γ regimes, respectively. (Paper **VI**.)
- I demonstrate that medium and large-scale detailed neuronal network models of cortical layers II/III model show a range of emergent dynamic phenomena which are also present *in vivo*. (Papers **VI**, **VII**.)

1.3 Papers

I. Mikael Djurfeldt, Örjan Ekeberg and Anders Lansner, "Large-scale modeling—a tool for conquering the complexity of the brain", *Front. Neuroinform.* 2(1) (2008), 1–4.

II. Mikael Djurfeldt, "The Connection-set Algebra—a novel formalism for the representation of connectivity structure in neuronal network models", *Neuroinformatics* (2009), submitted.

III. Mikael Djurfeldt, Anders Sandberg, Örjan Ekeberg and Anders Lansner, "See—a framework for simulation of biologically detailed and artificial neural networks and systems", *Neurocomputing* 26–27 (1999), 997–1003.

IV. Mikael Djurfeldt, Johannes Hjorth, Jochen Eppler, Niraj Dudani, Moritz Helias, Tobias Potjans, Upinder S. Bhalla, Markus Diesmann, Jeanette Hellgren and Örjan Ekeberg, "Run-time interoperability between neuronal simulators based on the MUSIC framework", *Neuroinformatics* (2009), in preparation.

V. Mikael Djurfeldt, Christopher Johansson, Örjan Ekeberg, Martin Rehn, Mikael Lundqvist and Anders Lansner, "Massively parallel simulation of brain-scale neuronal network models", KTH, School of Computer Science and Communication (2005), TRITA-NA-P0513.

VI. Mikael Djurfeldt, Mikael Lundqvist, Christopher Johansson, Martin Rehn, Örjan Ekeberg and Anders Lansner, "Brain-scale simulation of the neocortex on the Blue Gene/L supercomputer", *IBM J Research and Development* 52(1/2) (2008), 31–42

VII. Mikael Lundqvist, Martin Rehn, Mikael Djurfeldt and Anders Lansner, "Attractor dynamics in a modular network model of the neocortex", *Network: Computation in Neural Systems* 17 (2006), 253–278

My contributions per paper

I I performed the major part of the analysis of the role of large-scale models in computational neuroscience and wrote the paper.

II I conceived the connection-set algebra and wrote the paper.

III I designed and implemented the See simulator, took part in the development of the Guile scheme interpreter and implemented major parts of it. I wrote the paper.

IV I coordinated the work described in the paper. I conceived the original idea behind MUSIC, developed the design together with Örjan Ekeberg and implemented it. I took part in the development of the NEST-MUSIC interface and assisted the development of the MOOSE-MUSIC interface. I wrote part of the paper.

V I adapted and extended the SPLIT simulator for use with large-scale models. I implemented a simple but reasonably efficient complete pair-wise exchange algorithm and the connection-set algebra. I wrote the paper and made the figures, with exception for the chapter on the abstract model (chapter 5). I developed a scalable version of the cortical layer II/III model.

VI I designed the experiments. I made the simulation work and the data analysis. I wrote software for simulation and analysis and extended and adapted the SPLIT simulator for use with large-scale models. I developed an up-scaled version of the layer II/III cortex model. I made the graphs and wrote the major part of the paper.

VII I participated in model development and wrote part of the software used in the paper. I made figure 3A.

Chapter 2

Mathematical modeling in neuroscience

In this chapter we discuss the role of mathematical modeling in neuroscience, and the role of large-scale models in particular. The discussion concludes with section 2.11, which presents the connection-set algebra formalism. This chapter is, with the exception of section 2.11, based on a report from the 1st INCF Workshop on Large-scale Modeling of the Nervous System (Djurfeldt and Lansner, 2007).

2.1 Simulation versus emulation

Alan Turing used the term *simulation* in a very specialized sense. The term referred to the simulation by a digital computer of a subject discrete-state machine, defined by a set of state transitions, inputs and outputs.

In computational neuroscience, the term refers to the computation of the numerical approximation of a solution over time to the equations of a mathematical model. When performed on a digital computer, such computations are subject to the limitations of the computer. For example, some quantum mechanical processes can not be simulated on a digital computer. However, such limitations do not seem to be of any relevance to current computational models of the nervous system.

By an *emulation*, we refer to a model of the nervous system in the shape of a physical realization, for example in terms of an electronic circuit. The projects DAISY (Kennedy, 2005) and FACETS (Meier, 2005) both implement VLSI chips emulating neural circuits (see, e.g., Schemmel et al., 2008).

While most simulations are slower than real-time (time simulated is shorter than the wall-clock time required to compute the solution), it is possible to construct an artefact which can emulate a neural circuit in real-time. The important consequence is that the artefact can interact with the environment in real time and react to a continuous flow of events occurring asynchronously. In some cases this is also achievable with a simulation on a digital computer, as exemplified by the dynamic-

clamp technique (Sharp et al., 1993) or the goal of the Blue Brain project to simulate a cortical column on a super-computer in real time.

2.2 What do we mean by "model"?

Mathematical models are the language of science. According to *Wikipedia*, a *mathematical model* is an *abstract model* expressed in mathematical language. Further:

> An abstract model [...] is a theoretical construct that represents something, with a set of variables and a set of logical and quantitative relationships between them. Models in this sense are constructed to enable reasoning within an idealized logical framework [...] and are an important component of scientific theories. Idealized here means that the model may make explicit assumptions that are known to be false (or incomplete) in some detail. Such assumptions may be justified on the grounds that they simplify the model while, at the same time, allowing the production of acceptably accurate solutions [...].

It should be remembered that a mathematical model of reality should *always* be regarded as idealized in the sense above. At least this holds true for all types of model considered here.

2.3 The role of the model in current neuroscience

Let us now briefly list the various roles that models currently have in neuroscience. As stated above, models are the language of science. We use models to

- formulate hypotheses regarding the function of the nervous system

The activity of formulating a hypothesis in terms of a model requires collection of experimental data. It is often discovered that crucial data are missing. In this respect a model could be considered

- a tool to identify what we don't know

Often, hidden contradictions and inconsistencies are revealed during the formulation process, and it happens that models don't yield expected results so that a further role of the model is

- validating self-consistency of the description of a phenomenon or function

If the confidence in the model is strong but the predictions differ from experiment the model can be used to

- falsify hypotheses

If phenomena in the model are unexpected or unobserved a model can

- suggest new experiments

Finally, a model can be used as a

- platform for integrating knowledge

unifying experimental data from many sources in a consistent manner.

2.4 Model complexity

The golden standard with regard to model building is well captured by words often attributed to a certain famous physicist: "Everything should be made as simple as possible, but not simpler." Model complexity can be measured in many ways. For this discussion, two measures are particularly important: 1. the number of model parameters, and 2. the number of state variables in the model. The latter measure will here be denoted as *model dimension*.

A model is always tailored to answer a specific set of questions. For a physicist, the ideal is to pick the model with the smallest number of parameters which will still be sufficient to answer the scientific questions posed. This has several good consequences:

1. A simple model can be analyzed and understood. More complex aspects of nature can be understood through the strategy of divide-and-conquer.

2. A simpler model expresses a simpler scientific hypothesis in the sense of Occam's razor. It cuts away elements that are irrelevant to the problem studied.

3. A model with fewer parameters is better constrained, making it easier to falsify. A model with many parameters can too easily be adapted to the results of any experiment.

Of the two measures of model complexity mentioned above, the first one is most important. A model which has few parameters is more likely to express a well defined piece of scientific knowledge in the form of a strong, i.e. falsifiable, hypothesis, while the lack of this kind of simplicity threatens all three of the benefits above. A large model dimension can make the model harder to analyze and understand but there are many techniques available for handling that kind of complexity.

In addition, it is important to make a distinction between *free* parameters—parameters which are tuned by the modeler or software to achieve a certain model behavior—and parameters constrained by experimental data. A model should have few free parameters in order not to lose benefit 3 above. Also, all model behavior which is used to tune parameters is transformed from an "output" of the model to an "input", i.e., it can no longer be claimed as a result, or prediction, of the model.

2.5 Abstraction and level of description

The tool used to achieve a simple model is abstraction. The system is described at a certain level and elements which are not believed to be important for answering the scientific questions asked are taken away.

Neuroscience spans many levels of description from molecules to behavior. But what does *level* mean? Churchland and Sejnowski (1992) discuss three categories of levels. The structure of the nervous system has many different spatial scales with substructures such as molecules, synapses, neurons, microcircuits, networks, regions and systems (list slightly modified from Churchland). We call these *levels of organization*. With some good will, *behavior* could be added at the top of this hierarchy.

Marr (1982) described levels along a different dimension in his *levels of analysis*: 1. the computational level of abstract problem analysis, 2. the algorithmic level, specifying a formal procedure to perform the task so that a given input will yield the correct output, 3. the level of physical implementation. Marr argued that a higher-level question was largely independent of the levels below and could be analyzed independently of the lower level. However, it should be noted that Marr used, to a large extent, neurobiological considerations to constrain and inspire his computational theories and algorithms.

Churchlands third category, *levels of processing*, will not be discussed here.

For a model of the primate primary visual cortex, V1, Marr's computational level would correspond to *what* computations are being performed in V1 and *why*. The algorithmic level would correspond to *how* the information being processed on the computational level is represented and how the computations are carried out, while the level of physical implementation describes the actual computational elements performing these computations.

A typical large-scale network model, with single- or multicompartment units, thus belongs to the level of physical implementation, since it deals with neurons and synapses. But it can still be inspired, and constrained, from the other two levels, and can embody principles from the "higher" levels.

So far, we have discussed levels of organization and levels of analysis. An *abstract model* leaves out aspects of the description of reality in order to achieve simplicity. Sometimes this means leaving out elements from a lower level of organization, but it can also mean describing something at a higher level of analysis. A model of visual processing in terms of filter banks and kernels is considered more abstract than a model of neuronal populations in V1. Thus, we also talk about level of abstraction.

A detailed model is often considered the opposite of an abstract model. Here, a *detailed model* is defined to mean a model which spans several levels of organization. A model which spans the levels from networks to behavior can thus simultaneously be more abstract and more detailed compared to a model restricted to a single but lower level of organization. A model of brain imaging data which incorporates networks of simple units can be more detailed than a statistical model of an ion

channel. Yet, because it leaves out so much of the detail beneath the level of networks, it can also be more abstract than the latter model.

2.6 Realism

In order to say something about reality, and in order for a corresponding hypothesis to be falsifiable, a model needs to be well rooted in empirical data, i.e. formulated in a way that is consistent with a large set of experimental data.

It should now be made clear that, just as a model can be formulated on all of the levels of organization discussed in section 2.5, empirical data can be obtained on different levels of organization. In addition, models can be formulated on higher levels of analysis (in the sense of section 2.5): A cognitive psychologist can retrieve behavioral data and construct models at the algorithmic level, without a direct reference to how these processes are physically implemented in the brain. Brain imaging retrieves data at the levels of networks, regions and systems. Electrophysiology collects data at many levels of organization and this data can be used to construct models at the level of physical implementation.

A model based on data from a lower level of organization, or formulated in terms of elements from multiple lower levels of organization, is not necessarily more *realistic* than data formulated and rooted at a higher level. Rather, a model is realistic if it is well rooted in empirical data at the given level, if its parameters are well constrained by these data, and if it *correctly predicts* data which has not been used to tune the model.

Not only the parameters of a model, but also the equations, represent assumptions about reality. Some models are based on *first principles*, that is, their equations are established laws of physics. Models based on first principles, or models with equations which can be derived from the laws of physics, can be trusted to a larger extent than other models, because they rely less than other models on assumptions in the form of peculiarities of the model equations or fitting of parameters.

2.7 Top-down and bottom-up approaches

The *top-down* approach to modeling most often means using hypotheses on the computational or algorithmic levels as a starting point when approaching the formulation of a model at the level of physical implementation. These functional hypotheses guide the formulation of the model at the implementational level in terms of what elements are included in the model and which experimental data are considered important. A functional hypothesis can also complement experimental data in the sense of giving additional constraints. For example, if we have limited experimental data on the functional connectivity between inhibitory and excitatory connectivity, an additional functional constraint would be that the activity in the network must not be allowed to grow in an uncontrolled manner.

A top-down approach can also start with an abstract model, neglecting detail below the level of organization at which the model is formulated. Succeeding models can then increase the amount of detail, enabling them to account for a larger set of experimental data. For example, a connectionist, rate-based model can be developed into a spiking network model, followed by a Hodgkin-Huxley style model. However, it should be noted that models are more typically developed outside of such sequences, at a specific level of abstraction and detail suitable to the scientific questions posed.

The *bottom-up* approach, in contrast, means using the level of physical implementation as a starting point with the hope of capturing function. For example, what does the anatomy of the cerebral cortex mean? If we can, from the physical level of synapses, dendrites, neurons and networks, identify computational primitives of the cortex such primitives can be abstracted and we can move up one level of analysis. This is one goal of projects like DAISY (Kennedy, 2005), FACETS (Meier, 2005) and Blue Brain (Markram and Peck, 2004).

As with the top-down approach, a bottom-up approach can be concerned with levels of organization. In this case, it means to take a lower level of organization as a starting point for understanding the higher level.

In practise, the approach of a modeller is usually neither purely top-down nor purely bottom-up, as was already evident in Marr's work. Also, in the bottom-up approach, the focus of experiments and the choice of elements to include in the model is largely guided by functional hypotheses.

2.8 Explicitness

We define *model explicitness* to mean the degree to which the model is isomorphic with reality, or, how directly state variables of the model can be mapped to empirical data.

The degree of detail in a multi-compartment, Hodgkin-Huxley, model of a neuron aids in making this type of model explicit in the sense above. During intracellular recording of a neuron, it is often possible to block a subset of the ion channels in order to directly measure the current of another channel subset, or, in order to study the effect on cell behavior. The explicitness of the Hodgkin-Huxley formalism then makes it easy to perform a corresponding manipulation in the model. It is also easy to identify and display individual currents in the model.

In comparison, an integrate-and-fire model is less explicit. While recent versions (e.g. Brette and Gerstner (2005)) of this type of model can faithfully reproduce a spike train, some state variables correspond to the phenomenological effect of the coordinated action of multiple channel types in the real neuron. In this case, it is easy to compare the spike trains, but it is not as easy to map the dynamics of the model to the individual currents of the real neuron. On the other hand, the simplicity of the integrate-and-fire model makes it easier to analyze and understand.

Another aspect of explicitness is that it supports the role of the model as a

platform for integrating knowledge. An explicit model is more likely to connect well to a wider range of experimental data, even data which were not targeted when constructing the model.

2.9 Large-scale models

As defined here, a *large-scale model* is a model with a high dimension, i.e. a model with a large number of state variables (on the order of hundreds of millions or more). Thus, a detailed model of one cortical column can be large-scale while an abstract model of a large network encompassing multiple columns is not necessarily large-scale.

Integrate-and-fire models

The classical integrate-and-fire model (MacGregor and Oliver, 1974; Tuckwell, 1988) has one state variable per neuron, representing the membrane potential. It is basically a linear leaky integrator with a voltage threshold and a reset mechanism. The main advantage of this type of model compared to Hodgkin-Huxley type models is its simplicity. For example, it has far fewer parameters, while it still captures essential features of the neurons. Because it has fewer parameters it is easier to adapt this type of model to experimental data. In this sense, it is easier to achieve a certain level of realism with an integrate-and-fire model than with a Hodgkin-Huxley type model, while, with more effort and more data, the latter model can reach an even higher degree of realism. Its simplicity also makes it possible to develop automated procedures for extracting parameters from data (Jolivet et al., 2004; Keat et al., 2001; Paninski et al., 2004). Because of the low dimension and mathematical tractability, it is easier to analyze and understand this type of model. Finally, the simulation of this type of models require fewer computational resources making it possible to simulate larger networks given the same hardware. For examples of large integrate-and-fire-based network models, see Mehring et al. (2003); Aviel et al. (2003); Tetzlaff et al. (2004); Kumar et al. (2007); Morrison et al. (2007a).

The integrate-and-fire paradigm has recently been developed in three directions (Brette and Gerstner, 2005):

- the addition of a quadratic or exponential term, yielding a smooth spike initiation zone (Latham et al., 2000; Fourcaud-Trocme et al., 2003);
- the addition of a second state variable, enabling modeling of subthreshold resonances or adaptation (Izhikevich, 2003; Richardson et al., 2003);
- using active conductances to model synaptic inputs (Destexhe et al., 2003).

Hodgkin-Huxley models

In the Hodgkin-Huxley formalism (Hodgkin and Huxley, 1952; Ekeberg et al., 1991), the cell membrane potential, $V(t)$, of a neural compartment is expressed as a differential equation

$$C_m \frac{dV}{dt} = I_{comp} + I_{cond} + I_{syn} \tag{2.1}$$

where C_m is the membrane capacitance, I_{comp} the sum of currents from adjacent compartments, I_{cond} the sum of ionic currents through channels in the cell membrane, and I_{syn} the sum of synaptic currents. The electrical behavior of the cell is determined by by the ionic currents which are described through *activation* and *inactivation* functions. For example, the *delayed rectifier* current, carried by potassium ions, is described by

$$I_{Kr} = (E_{Kr} - V(t))G_{Kr}n^{k_{Kr}} \tag{2.2}$$

Here n is an activation function described by

$$\frac{dn}{dt} = \alpha_n(1-n) - \beta_n n \tag{2.3}$$

where α_n and β_n depend nonlinearly on $V(t)$.

A network model with multi-compartmental Hodgkin-Huxley type units is more detailed than its integrate-and-fire counterpart. It is more complex, both by having more parameters and a larger model dimension. As has been discussed in section 2.9, this requires more labor to determine parameters from experimental data. Sometimes, not all data is available so that parameters need to be determined more indirectly. The disadvantage is that this turns an output of the model into an input (c.f., section 2.4). For example, if we tune the conductance of a K_{Ca} channel in order to obtain the correct time course of an AHP, instead of measuring this conductance, we can no longer claim that our model predicts a correct AHP. In this case, however, the kind of predictions which are of interest in a network model appear at another level of organization within the model.

In section 2.8, the explicitness of HH-type models was discussed. The higher level of detail allows this type of model to be connected to a wider range of experimental data. The presence of ionic currents allows for comparatively easy modeling of pharmacological manipulations. The 3D extent of a compartmental model allows for the synthesis of EEG and LFP signals (Einevoll et al., 2007).

The bottom-up approach to the cortical column

Regardless of whether we use integrate-and-fire or Hodgkin-Huxley type units in a network model, an important set of parameters that currently largely lacks an experimental basis is the set of connectivity parameters. Data from, for example, Thomson et al. (2002) and Binzegger et al. (2004) gives the statistics of connectivity between pairs of cell types. This type of data has led to the use of random Gaussian

(e.g. Brunel (2000)) or random uniform (e.g. Haeusler and Maass (2007)) connectivity in network models, consistent with such statistics. However, it is reasonable to assume that the microcircuitry of a column has more structure than that. Also, there is very limited data on the structure of long-range connectivity.

The Blue Brain project (Markram and Peck, 2004) aims to collect data on individual cells, for example acquisition of cell morphology through cell labeling and 2-photon microscopy, and then using database techniques and specialized software to reconstruct a virtual column. The superposition of reconstructed cells in 3D-space may give additional constraints needed to get a more complete picture of the microcircuitry. The aim is also to simulate a large-scale network model of a complete column with multicompartmental Hodgkin-Huxley-type units without reference to functional hypotheses about the network. Thus, the approach is essentially hard-core bottom-up.

It is generally very difficult to experimentally determine the existence of a synaptic contact between cortical neurons from morphological data since axonal processes can pass close to the dendritic processes of neurons without forming a synapse. In essence, the only way to determine anatomically if there is a contact is by looking at it with an electron microscope. It is also possible to determine the existence of a connection electrophysiologically by recording from pairs of neurons (e.g., Thomson et al., 2002).

One particularly interesting development with regard to the acquisition of connectivity data is the method called "serial block-face scanning electron microscopy" or SBFSEM (Denk and Horstmann, 2004). A microtome is placed in the chamber of a scanning electron microscope. The face of the tissue sample is scanned and 50–70 nm slices cut away, generating stacks of thousands of images from which a bulk 3D volume can be reconstructed. The acquired data has enough resolution to trace thin axons and identify synapses. This method holds the promise of geometrically reconstructing an entire neocortical minicolumn and extracting its circuitry.

The acquisition of data at multiple levels of organization leads to a new scale of projects—a development which parallels the study of the genome and studies in particle physics. There will be new kinds of challenges associated with industrial-scale data acquisition in projects of thousands of man-years.

Combining the top-down and bottom-up approaches

A survey of existing literature in computational neuroscience shows that the pure bottom-up approach to understanding network function is very rare. Part of the explanation is that missing empirical data need to be complemented with functional hypotheses in order to make progress possible. But a top-down approach may have importance in other ways than replacing lacking knowledge. One may consider the question whether a correctly implemented, detailed computer replica of the cortical column would by itself say very much about network function. Because a detailed model can be complex and hard to analyze and understand, modellers tend to see functional hypotheses and abstract models as a necessary complement

in dissecting cortical function. It should still be noted that a computer replica is much more accessible to experimentation than the living tissue in the sense that any set of state variables can be logged and an arbitrary set of variables can be simultaneously perturbed in a precise fashion.

An example of how a top-down approach can be combined with the bottom-up approach is given by the model of paper **VII** (see paper **VI** for a large-scale version of the model). The model is mainly designed to target the question: Is neocortical microarchitecture consistent with the hypothesis of attractor memory network function? Here, most parameters of the neuron models are determined from experimental data in a bottom-up manner. However, the connectivity parameters are determined by combining a long-range connectivity structure required for attractor memory network function with the currently existing empirical constraints on connectivity mentioned in section 2.9.

Another aspect of this model, and most or all other network models, is that the parameters of a neuron type are replicated over the population of model neurons, with or without random perturbation, in a crystal-like manner. This means that even if a large-scale model of this type has a large model dimension, it can still have a comparatively small number of parameters and, thus, be simple in the important sense (c.f. section 2.4).

Volume simulation

Until now, Hodgkin-Huxley type models have represented the most basic level of organization at which we simulate neurons and circuits, with the exception of hybrid models also including biochemical processes inside the cell. Data from the SBFSEM method mentioned in section 2.9 opens up the prospect of a full 3D volume simulation of a cortical column. Queisser et al. (2008) have presented initial attempts in this direction at the level of a single cell. In the Hodgkin-Huxley approach, the neuron is modelled as an electrical circuit. Here, instead, the 3D volume in which the neuron is embedded is described in its entirety by partial differential equations (PDEs) and simulated using the solver μG (Bastian et al., 1997; Wittum, 2007).

A model of the 3D volume can be based on *first principles* in the sense discussed in section 2.6. In this case, the model is based on Maxwell's equations which describe the dynamics of the electromagnetic field.

Simulation of growth processes

In order to fully understand the cortical architecture, it is necessary to understand the development and growth processes from which it results. Within the DAISY project (Kennedy, 2005), initial steps are currently taken to simulate the migration of neuroblasts. This adds a requirement on the simulator which is not yet fulfilled by standard neuron simulators such as Neuron and Genesis: there is a need to quantize space. Simulation tools are being developed within DAISY to meet this demand. The solver μG is based on a communication layer, DDD (Dynamic Distributed

Data), which allows computational loads to migrate within a parallel computer during simulation. This layer could also be a suitable substrate for the simulation of growth processes.

2.10 Upscaling

A model of the bulk 3D volume of neural tissue using PDEs (section 2.9), if well-rooted in empirical data, can be considered more realistic than the Hodgkin-Huxley model. It is based on first principles (sections 2.9, 2.6) while the Hodgkin-Huxley model is partly based on simplifying assumptions and curve fitting. This means that we can use the 3D volume model to *validate* the multicompartmental Hodgkin-Huxley model.

At the 1st INCF Workshop on Large-scale Modeling of the Nervous System, Wittum (2006) reported on simulations of a single cell where ephaptic interactions could be observed between two of the dendritic processes of the cell itself. Such a phenomenon would not arise in the Hodgkin-Huxley model. It is also clear that the surrounding interstitial fluid, neuropil, and dendritic processes have substantial effects on the electric behavior of the cell membrane, diverging from the Hodgkin-Huxley model. Clearly, however, the 3D volume simulation is not a good replacement for the Hodgkin-Huxley model, because it is computationally heavier. The question then arises what alternatives we have to the HH model.

An important answer is given by the 3D model itself: There exist mathematical techniques for deriving a model at a coarser scale from a model at a finer scale. This methodology is called *upscaling* (Eberhard et al., 2004). Through upscaling techniques it might be possible to derive a candidate model which might serve as a better replacement for the Hodgkin-Huxley model.

2.11 The connection-set algebra

When building models of neuronal networks or systems, regardless if it is connectionist rate-based, integrate-and-fire or Hodgkin-Huxley models, we are confronted with the problem of describing how neurons connect to each other. As model complexity increases, especially if the model contains multiple populations of neurons, network connectivity can become demanding to handle. In this section, we present a mathematical tool to describe the structure of connections between two neuronal populations in a concise manner which captures the ideas of the model constructor without introducing unnecessary complexity.

Introduction

The connection-set algebra is a novel formalism for the description of connectivity in neuronal network models. It is described in detail in paper **II**. Some model scripting languages contain higher level constructs which abstract aspects of the model

description. For example, PyNN (Davison et al., 2009), the NEST Topology Module (Plesser and Austvoll, 2009), and, EpsiloNN (Strey, 1997) contain constructs for concisely treating sets of neurons collectively as *populations*, and standard *connection patterns* such as "all-to-all", "one-to-one" and random connectivity. All languages mentioned allows the user to resort to a lower-level description where neurons and connections are dealt with individually.

Here, we suggest a new way to describe connectivity in neuronal network models, a *connection-set algebra*, which is declarative, compact and expressive. The scope is more specialized compared to the model description languages referred to above in that it *only* provides a representation of connectivity structure and does not deal with neuronal populations, model composition, parameters or dynamics.

The connection-set algebra may be used to describe *types* of connectivity, i.e. connection patterns, both to humans and to neuronal network simulators. It is not restricted to a basic set of connection patterns. Rather, new ones can be constructed using the operators of the algebra. The description is independent of sizes of pre- and postsynaptic populations and of synapse and neuron models. It decouples the description of connectivity structure from details of parallelization, such as the partitioning of the problem and mapping of neurons and connections to processors. A pilot implementation of the algebra has been used in a large-scale neuronal network simulation (paper **VI**).

Connection-sets

In the connection-set algebra, a *connection-set* represents a connection pattern. It can be regarded as the infinite connection matrix between two infinitely large neuronal populations. When setting up connections between the populations the neurons are first enumerated. For the standard case that the populations are enumerated using contiguous intervals of integers starting at 0, a rectangle, with one corner in origo, is cut out of the infinite matrix. This rectangular matrix then becomes the connection matrix between the two populations (see Figure 2.1). When describing a connection pattern, we are both interested in which connections *exist* and in assigning *values*, such as synapse efficacy and axonal delay, to connections. In the connection-set algebra, the first purpose is served by *masks*, the second by *value sets*.

A connection-set is a tuple of a *mask*, $\overline{M}$ and zero or more *value sets*, $V_0, V_1, \ldots$:

$$\langle \overline{M}, V_0, V_1, \ldots \rangle \tag{2.4}$$

A mask describes which of all possible pairs of neurons are connected. A mask can be viewed as a boolean matrix or as an indicator function

$$I_S : \mathbb{N}_0 \times \mathbb{N}_0 \to \{\mathcal{F}, \mathcal{T}\} \tag{2.5}$$

Here, the domain $\mathbb{N}_0 \times \mathbb{N}_0$ is the set of all possible pairs of pre- and postsynaptic neuron indices, i.e. all pairs of non-negative integers.

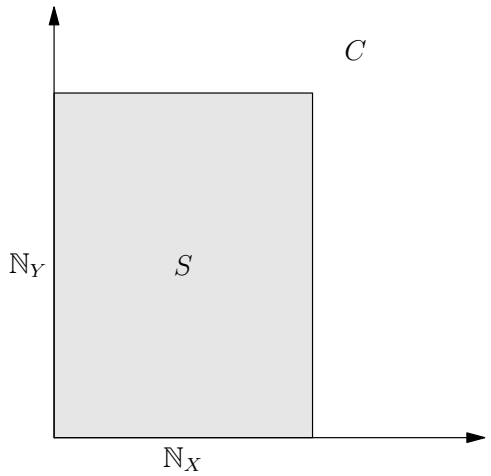

Figure 2.1: Enumerations of neuronal populations X and Y give integer indices into the finite connection matrix S cut out of the infinite connection-set C.

In analogy to masks, a value set can be viewed as a matrix or as a function

$$V : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{R}^N \tag{2.6}$$

where the domain $\mathbb{N}_0 \times \mathbb{N}_0$ is the set of all possible pairs of pre- and postsynaptic neuron indices and $N = 1$ in the common case.

Computing with connection-sets

The connection-set algebra includes a set of elementary (pre-existing) connection-sets. One of these is the mask $\bar{\delta}$ which is used to specify one-to-one connectivity. It is defined:

$$\forall (i,j) \in \mathbb{N}_0 \times \mathbb{N}_0 : \bar{\delta}(i,j) = \begin{cases} \mathcal{T} & \text{if } i = j, \\ \mathcal{F} & \text{otherwise.} \end{cases} \tag{2.7}$$

One of the operators of the algebra is the *block operator*, **B**. It expands elements of a connection-set into blocks of a block structured connection-set. For example, the connection matrix of a network consisting of independent, unconnected, groups of four cells with all-to-all connectivity within each group is a block diagonal matrix

$$\mathbf{B}(4)\bar{\delta} \tag{2.8}$$

This matrix is illustrated in Figure 2.2. With two arguments, **B** generates rectangular blocks. Paper **II** contains a more thorough explanation and definition of the algebra.

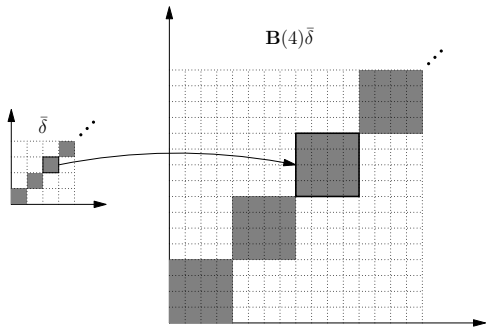

Figure 2.2: The block operator $\mathbf{B}(n)$ expands a connection-set into a block-matrix where every element of the original connection-set becomes an $n \times n$ block in the new matrix. $\mathbf{B}(4)$ is here applied to $\bar{\delta}$ to yield the connectivity of independent groups of 4 fully connected neurons each.

A more complex example

The following is, with the exception of constants and the minicolumn-level connectivity matrix P, the full description of the connectivity structure of the model in chapter 5 and papers **VI-VII**.

$$C_{\mathrm{bp}} = \bar{\rho}(0.7) \cap \mathbf{B}(h_{\mathrm{b}}, h_{\mathrm{p}})\bar{\delta} \tag{2.9}$$
$$C_{\mathrm{rp}} = \bar{\rho}(0.7) \cap \mathbf{B}(m_{\mathrm{r}}, m_{\mathrm{p}})\bar{\delta} \tag{2.10}$$
$$C^{\mathrm{l}}_{\mathrm{pp}} = \bar{\rho}(0.25) \cap \mathbf{B}(m_{\mathrm{p}})\bar{\delta} - \bar{\delta} \tag{2.11}$$
$$C^{\mathrm{g}}_{\mathrm{pp}} = \bar{\rho}\mathbf{B}(m_{\mathrm{p}})\theta(a_e P) - \mathbf{B}(m_{\mathrm{p}})\bar{\delta} \tag{2.12}$$
$$C_{\mathrm{pr}} = \bar{\rho}\mathbf{B}(m_{\mathrm{p}}, m_{\mathrm{r}})\theta(-a_i P)) - \mathbf{B}(m_{\mathrm{p}}, m_{\mathrm{r}})\bar{\delta} \tag{2.13}$$
$$C_{\mathrm{pb}} = \bar{\rho}(0.7) \cap \mathbf{B}(m_{\mathrm{p}}, m_{\mathrm{b}})\mathrm{n\_closest\_pre}(g_m, h_m, 8) \tag{2.14}$$

Here, equations 2.9-2.14 each describe the mask of one of the projections in the model. (Cell types are p for pyramidal cells, b for basket cells and r for regular spiking non-pyramidal interneurons; this example is more thoroughly covered in paper **II**.) The connection-set algebra here gives a concise and unambiguous description of a comparatively complex connectivity between three populations of neurons. Also note how the algebra makes no reference to sizes of the participating populations making the model description scalable.

Chapter 3

Review of simulation software concepts

In the previous chapter, we have presented conceptual and mathematical tools to handle the complexity of neural systems. In this chapter we will discuss software tools and issues related to simulation of neuronal network models. The chapter is based on the report from the 1st INCF Workshop on Large-scale Modeling of the Nervous System (Djurfeldt and Lansner, 2007).

3.1 Important properties of simulator software

When selecting software for large-scale simulation, there are many criteria that need to be examined, some of which will be mentioned in this section. Brette et al. (2007) reviews existing tools for simulation of networks of spiking neurons.

- **Model types supported** What neuronal/synaptic/plasticity models can be simulated?
- **Accuracy** Does the simulator give correct results? Recent work (Brette, 2006; Rudolph and Destexhe, 2006; Morrison et al., 2007b) has presented methods for determining spike times in a simulator more precisely, and has shown that this can have effects on discharge statistics and temporal precision in resolving synaptic inputs. This is discussed further in section 3.4 below.
- **Scalability** In general, it is difficult to write efficient parallel implementations. How does *speedup* (simulation time divided by simulation time on one processor) scale with the number of processors used? Ideally it should grow linearly. How does simulation time scale as a function of the size of the model?
- **Documentation** How good is the documentation?

- **Support** How quickly will the developers respond to bug-reports or feature requests?
- **License and availability of source code** In a research environment, it is an advantage to have the source code for the simulator available, and to have permission to modify it. This is guaranteed for all software covered by the GPL license from the Free Software Foundation and some related licenses.
- **Adaptability** How easy is it to adapt the simulator to your purpose? How easy is to to add new mechanisms?
- **Portability** Does it run on my preferred platform?
- **Interoperability** How easy is it to collaborate with others using a different simulator? This is discussed further in section 3.3 below.
- **Is there a graphical interface?**
- **What analysis/post-processing tools are available?**

3.2 Diversity of simulators

One might ask if it is sensible for the computational neuroscience community to split efforts into the large and growing set of neuron simulators available today rather than focussing on one or a few tools. Here, it is important to note that there is currently a strong ongoing development of simulation technology, and that simulators tend to have unique strong points not shared by others.[1]

The reproducibility of models, that is the possibility for another research group to reproduce the original research results, is a major problem (see section 3.10). The diversity of simulators might allow for simulating the model on a different platform from that on which it was originally developed, thereby verifying the reproducibility of results. Thus, in the first major finding of the report on the 1st INCF Workshop on Large-scale Modeling of the Nervous System (Djurfeldt and Lansner, 2007), workshop participants agreed that the current diversity of simulators creates vigor in the field and has benefits for the validation of models.

3.3 Software interoperability

Given the diversity of simulators, it is important to find ways to share the gains produced by the efforts put into individual simulators by both developers and users. There exists some approaches to simulator-independent modeling environments which allow a model to be simulated using more than one simulation engine.

[1] Michael Hines, tha author of Neuron, has once noted that "The reason why we keep reinventing the wheel is that we haven't got it quite round yet."

This is important for verification of simulator accuracy, for the reproduction, testing and extension of published models, and for collaboration between modellers using different simulation tools. Examples are graphical environments, declarative model specifications using XML, and procedural model specifications using an API implemented in the Python programming language. One such simulator-independent environment ('meta-simulator') which has emerged strongly is PyNN (Davison et al., 2009). In the second major finding of the the 1st INCF Workshop on Large-scale Modeling of the Nervous System (Djurfeldt and Lansner, 2007) the workshop participants agreed upon the importance of facilitating software interoperability and re-use of simulation software components.

Modularity

The practise of dividing software into *modules* with well-defined roles has many advantages. It eases development and increases maintainability of the code. If such modules have well-defined *interfaces*, modules can be re-used in other circumstances. Such an interface can be in the form of

1. an *application programming interface* (API), enabling a module in the form of a compilation unit or a library to be linked into an application. This includes the definition of *data structures* required to pass information through the interface.

2. a *communication interface*, enabling modules in the form of processes to communicate while running simultaneously on the same or on different machines

3. a *file format*, allowing the output of a module in the form of an application to be read as input to another application. Here, "input" can be a model specification. In this case, the interface takes the shape of a model specification language.

As an example of the first form of modularity, a simulator can be divided into a *simulator kernel*, responsible for the distribution and allocation of data structures over a cluster, for building the model on the nodes, and for performing the computations during a simulation, and other modules required for module specification, input and output. The simulator kernel can be further divided into a *solver*, with the sole responsibility of performing computations, and modules required for allocation, distribution etc.

Another possibility is to enable on-line interaction between simulators during a simulation (see below). This would entail modularity of the second type.

An example of the third type of modularity is neuroConstruct which is a software application for creating 3D models of networks of biologically realistic neurons through a graphical user interface (GUI) (Gleeson et al., 2007). neuroConstruct can import morphology files in Genesis, Neuron, Neurolucida, SWC and MorphML formats for inclusion in network models and can generate model specification files

for Genesis and Neuron. Efforts put into developing neuroConstruct further will thus benefit both the Genesis and Neuron communities. Note, though, that the choice of two output formats is forced by the current lack of a standard format for model description. This will be discussed further below.

The next generation of Genesis, Genesis 3 (Cornelis et al., 2008), aims to foster collaborative modeling through a rich set of interfaces.

We cannot gain fully from a modularization of simulation software until we have developed *standard interfaces* between software components.

On-line interaction between simulators

As was discussed in section 3.2, different simulators have different strengths. Current development is moving in the direction of simulation of large systems of networks. The situation could arise that one simulation framework is most suitable for one part of the system while another framework is needed for another part. For example, a retina model could provide input for a model of LGN/PGN/V1, or, a medium-sized network of detailed structurally realistic neurons could interact with a very large network of simple integrate-and-fire point neurons. In the former example, one solution would be to perform the simulation in *batch mode*, letting the retina model generate spikes and save the spike times to file. Such spike files can then be read by the second simulator. This would be an example of the third type of modularity discussed above. In the latter example, this is not possible due to the need for bi-directional interaction. This situation is resolved if the simulators can transfer spikes *on-line* through a communication interface.

Even in cases where batch mode simulation is possible, it may be desirable to let such a simulation interact with the environment in real time. Another argument for preferring on-line communication between simulators over batch mode simulation is if the amount of data generated is of such a large magnitude that it is undesirable or impossible to store intermediate data in files.

If both simulators are run on the same parallel computer, some further coordination may be required with regard to allocation of nodes, initialization of MPI, etc. This could be achieved if the communication interface has the form of a communications library, providing services like initialization of MPI and the option of external communication. There is also a need for a naming or addressing mechanism to allow for flexible connectivity between modules through multiple communication links.

The development of such an interface may or may not result in a software component that will be generally adopted, but has value in itself as an exploration of the concept. The 1st INCF Workshop on Large-scale Modeling of the Nervous System (Djurfeldt and Lansner, 2007) gave as a recommendation to implement an experimental framework for connecting software components and that a feasibility study should be performed regarding the possibility of on-line communication between different software modules, for example two parallel simulators. This recommendation led to the development of MUSIC, an API allowing large scale neuron

simulators using MPI internally to exchange data during runtime (see section 4.2 and paper **IV**).

Common specification language

Different simulator environments have different ways of specifying a model. For example, Neuron and Genesis have distinct specification languages. The translation of a model description from one environment to another can entail substantial effort. This is especially true in the case that environments use different formalisms. If the differential equation for the activation factor in the Hodgkin-Huxley model of a channel has different forms in the two environments, the translation of the model will even involve finding a new set of parameters. This situation creates barriers between laboratories using different simulators, makes it harder for one laboratory to freely choose the tool suitable for the problem at hand, and threatens the reproducibility of scientific results. A common, standard specification language, supported by all simulators, would alleviate such problems and increase the utility of model repositories such as ModelDB (Hines et al., 2004) and databases such as NeuronDB (Mirsky et al., 1998).

NeuroML (Goddard et al., 2001) is one viable candidate for a model description standard. NeuroML is a collection of projects with the aim of developing standards for specification of neuroscience models in XML. It is organized into four *levels of scale*, where each succeeding level extends the features of the language. The first level covers the neuronal level of organization while the last level (level 4) covers all levels of organization from biochemical networks to systems. It includes the XML schemas MorphML, ChannelML and NetworkML. NetworkML (Crook et al., 2007) is currently the least developed schema and needs input from simulator developers.

PyNN, discussed in the following section, is another candidate as a model description standard.

Common scripting language

A *scripting language* is an interpreted language which is used to control an application. It is often *embedded* in the sense that much of the functionality of the application is available as function or procedure calls in the language—a *language binding*. If it is possible to add new functionality to the application in terms of code written in the scripting language, it is also called an *extension language*. The use of a full-fledged general purpose programming language as scripting/extension language has emerged as a powerful concept. Scripting languages such as Emacs Lisp, TCL, Matlab and Python have each given rise to prolific user communities and rich sets of software tools and libraries, and can provide a backbone in a framework with multiple modules.

Within the FACETS project (Meier, 2005), Andrew Davison has proposed the PyNN framework (Davison et al., 2009) as a standard scripting language binding for neuronal network simulators. This abstracts differences between simulators

and provides a common way to specify models and run simulations. The goal is that simulation scripts in PyNN for simulator A will run on simulator B without modification.

PyNN is based on the Python programming language and includes the development of an API and the binding to individual simulation engines. The API has two parts, a low-level, procedural API, and a high-level, object-oriented API. The low-level API can be used for small networks, and gives more flexibility than the high-level API. The high-level API hides details and book-keeping, and is intended to have a one-to-one mapping with NeuroML, i.e. a population element in NeuroML will correspond to a Population object in PyNN, etc. Another requirement for a common scripting language is standard cell models. PyNN translates standard cell-model names and parameter names into simulator-specific names.

The use of Python in PyNN results in a free, Matlab-like environment with tools for data analysis, plotting, mathematical libraries, etc., leveraging the efforts of the Python user community.

3.4 Accuracy of simulation

Very large networks of spiking neurons can be simulated efficiently in parallel under the constraint that spike times are bound to an equidistant time grid. Within this scheme, the subthreshold dynamics of a wide class of integrate-and-fire type neuron models can be integrated exactly from one grid point to the next. However, the loss in accuracy caused by restricting spike times to the grid can have undesirable consequences, which has led to interest in interpolating spike times between the grid points to retrieve an adequate representation of network dynamics. The exact integration scheme can be combined naturally with off-grid spike events found by interpolation (Morrison et al., 2007b). By exploiting the existence of a minimal synaptic propagation delay, the need for a central event queue is removed, so that the precision of event-driven simulation on the level of single neurons is combined with the efficiency of time-driven global scheduling. Further, for neuron models with linear subthreshold dynamics, even local event queuing can be avoided, resulting in much greater efficiency on the single neuron level. A measure of the efficiency of network simulations in terms of their integration error shows that, for a wide range of input spike rates, these novel techniques are both more accurate and faster than standard techniques.

3.5 Specification of large-scale network models

In section 2.9 we discussed the bottom-up approach to large-scale modeling of the cortical column using data from the SBFSEM method. For a 3D volume simulation (section 2.9), a corresponding 3D volume of dielectric properties must be generated as part of the model specification. For a more traditional simulation

based on compartment model neurons, the data must be analyzed with respect to cell morphology and a list of network connections be generated.

In section 2.9 we discussed a combined top-down/bottom-up approach where the connectivity is based on functional hypotheses regarding cortical function. The Brunel (2000) and Haeusler and Maass (2007) models also belong to this type. In such models, there is a large contrast between the seemingly complex connectivity of the simulated model and the simple ideas upon which the connectivity is based. Usually the connectivity is expressed by a serial algorithm that sets up connections one by one. On a parallel machine, several simulators use the approach of running the serial algorithm in each process independently and simply ignoring attempts to set up connections not belonging to the local process. The advantage is that the complexity of parallelism is hidden from the user. However, this approach does not scale well. Papers **II** and **V** present an approach which scales well at the same time as expressing the connectivity in a form that preserves the underlying ideas explicitly in the program code. Projections between neuronal populations are described by iterators representing infinite connection matrices. Only the relevant finite piece of a matrix is used to connect the populations during model setup. A basic set of parameterized matrices can be combined through a *connection-set algebra* to form new connectivity structures. This algebra is presented in section 2.11 and paper **II**.

A third way to specify networks is to generate them based on empirical structure parameters. Regarding automatic generation of realistic, large-scale neural networks, there is a need to distinguish between generation and growing of cells. Growing of cells might depend on local or global parameters in the network. Further, an open question is how to specify and generate spatial connectivity. The easiest way of generating connectivity is using experimental data about the localization of different neuron types in the brain. This approach is commonly used in the available software tools.

3.6 Declarative versus procedural model description

What are the relative merits of declarative versus procedural model specifications? Model definitions based on NeuroML are basically declarative, although it is in principle possible to specify algorithmic elements in XML. In contrast, a model description in a Python-based scripting language is often procedural.

A declarative description is often easier to read and understand than a procedural description. It also usually gives a software tool more leeway for internal optimizations.

However, in a certain respect, a declarative descriptions throws away structural knowledge about the problem already available to the researcher. This knowledge is expressed in algorithms which can be close to the mathematical formulation of the model. We should look for a good high-level description that allows a human reader to see from the code what the point is.

3.7 Postprocessing and visualization

Large-scale have, by definition, a large number of state variables. When the number of variables is in the order of hundreds of millions or more, the problem how to visualize network activity becomes important. DAVIS (Robbins et al., 2004) is one example of an existing tool. However, there seems to be a great need of further tool development in this area. For example, for sufficiently large data volumes, the visualization tool needs to be parallelized.

A large-scale model spans many levels of organization, and, thus, needs to connect to experimental data on many levels. There is, therefore, a need for tools which can transform logged data from model state variables, such as cell membrane voltages, into synthetic versions of common brain imaging methods such as fMRI, VSD and EEG.

3.8 Verification of simulator function

In science, an individual experiment carries little weight, but when the result is reproduced in other laboratories, by different people, in slightly different ways, this gives strong validation. In parallel to this, there is a need for validation of simulator function.

It is an established software engineering fact that any sufficiently complex program does contain errors, regardless of the quality of the developer team. For a simulator, there can be errors at the level of a model built in implicitly in a simulator—e.g. the Hodgkin-Huxley neuron model, at the level of the numerical method used, and at the level of its implementation. Furthermore, different simulators can have different accuracy and vary in their efficiency.

Section 3.2 discusses the diverse set of existing neuron simulators. One of the ways in which to draw advantage from this diversity is to use the fact that different simulators are programmed differently to cross-validate simulators: If simulation results can be reproduced using a differently programmed simulator, this gives a relatively strong verification of the correctness of both simulators.

For such cross-validation to work well, there is a need to agree on a common set of simulator "benchmarks". Such a framework could also make possible comparisons of accuracy and efficiency. This could work as an inspiration and driving force for simulator developers, and help in focussing development. Without quantitative comparison data it is difficult for developers to know what is practically feasible.

A standard benchmark suite would be of great community value. It is important that such benchmarks be with respect to published models not contrived by the developers. There is otherwise a risk to neglect good methods because they had poor performance in artificial benchmarks. This experience parallels that in software engineering, where the trend during the previous two decades has been from artificial contrived benchmarks towards more application-like. It is also important that the data on the web pages be updateable so that old information can

be superseded by current best practices in each simulator.

Examples of possible benchmarks are the DeSchutter Purkinje cell model (Schutter, 1998) and the model of Vogels and Abbott (2005). ModelDB (Hines et al., 2004) contains 280+ such published models from which a selection could be drawn.

3.9 Model verification

In section 3.8, the verification of the simulation tools was discussed. Here we will discuss the verification of *models* and the reproducibility of modeling results.

A crucial point for a simulation is: How do you know that what you're simulating is correct? As was discussed in section 2.6, the model needs to be well rooted in empirical data. This can be achieved if test problems are chosen with care. Such problems should deal with basic traits of the domain under study and be chosen so that many aspects of the problem are easily accessible through experimental measurement. The test problems can validate the simulation technique so that simulations can then be extrapolated to other problems with some degree of confidence. Some areas in the brain are easier to work with, with regard to model validation, than others. It is especially important to look at a region with well-defined inputs and outputs, such as, for example, the barrel cortex.

Section 2.6 also discussed how models based on first principles, such as a 3D volume model of the electromagnetic field in neural tissue, increase confidence in the model; section 2.10 discussed how this gives the possibility of validating the Hodgkin-Huxley model and how upscaling techniques could be used to derive, from the 3D volume model, a new model at the same level of organization as the Hodgkin-Huxley model.

From the top-down perspective, there is doubt whether a 3D volume model of an entire network, such as those discussed in section 2.9, would really say anything, and, in particular, that such very detailed network models lack measures of verifiability. An approach to the latter problem could be to verify the simulation tool on simpler problems, such as the test problems discussed above.

In order to develop good test problems, there is a need for experiments tailored towards simulation. There is a sociological, or structural problem in the field such that experimentalists will not go very far in answering such needs. On the other hand, there are now some experimental laboratories which use modeling as a fundamental research tool.

3.10 Model reproducibility

Published simulation results are, in general, very hard to reproduce. In the third major finding of the 1st INCF Workshop on Large-scale Modeling of the Nervous System (Djurfeldt and Lansner, 2007) workshop participants reported on difficulties in reproducing simulations from published articles.

There seem to be several common reasons for this:

- The model description is incomplete. It is often necessary to contact the authors to get additional information required to run the simulation.
- The model description differs from the model used to produce the results. It occurs that last-minute modifications are made to the model, while the same modifications do not find their way into the methods section.
- The model description uses a formalism not supported on the simulator used to reproduce the results. One example of this is given in section 3.3.

It may also happen that the simulations described in the article have been carried out on peculiar hardware and/or with custom code, entailing substantial re-implementation efforts when running the simulation with standard software on standard hardware.

One may ask whether the concept of reproducibility should include exact quantitative reproducibility, i.e. reproduction of not only the behavior of the model but also the exact numerical values of simulation output. Such a requirement is harsh, because it means not only that identical algorithms must be used, but also that numerical operations need to be performed in an identical order so that cancellation effects, rounding errors, etc. do not cause noticeable differences.

An intermediate requirement would be that every simulation of the model with the same simulation software on the same machine gives identical results. This kind of reproducibility is important, for example during development and debugging. Even this is far from trivial on a cluster, since this requires a mechanism for seeding the pseudo random number generator in each process. It may even be impossible to make the result independent of the number of processes, because the uses of random numbers occur in a different order depending on how the model is distributed over processes.

Chapter 4

Modeling tools

This chapter begins with the presentation of two different kinds of modular frameworks, the See simulator and MUSIC, which can be used to simulate systems of networks, and concludes with a description of a series of improvements to the SPLIT simulator for the simulation of very large neuronal networks. This chapter builds on material from papers **III** and **VI**.

4.1 See

"See" is a software framework for simulation of biologically detailed and artificial neural networks and systems. See represents an approach to software modularity (see section 3.3) where the simulator framework provides API:s for services such as scheduling of events and communication. In this respect, it is an early example of a methodology which has later been absorbed by simulators such as Genesis3 (Cornelis et al., 2008) and MOOSE. It is also an early example of using a general purpose language as simulation scripting language. The scripting language is based on Scheme, while the basic framework is written in C++. Models can be built on the Scheme level from "simulation objects", each representing a population of neurons, a projection, etc. The simulator provides a flexible and efficient protocol for data transfer between such objects. See contains a user interface to the parallelized, platform independent, library SPLIT intended for biologically detailed modeling of large-scale networks and is easy to extend with new user code, both on the C++ and Scheme levels. The See simulator has been used in a simulation of the primary visual pathway of the cat (Djurfeldt, 1997).

Modularity

Similar to MatLab's Simulink and Genesis, See is built around the idea of interacting simulation objects which exchange data through links (figure 4.1). For example,

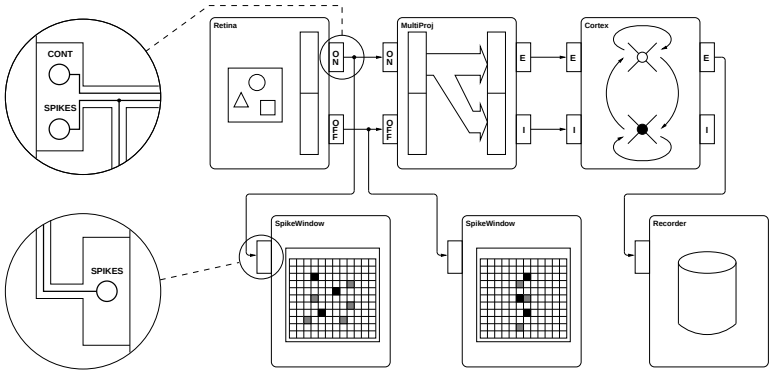

Figure 4.1: Structure of a See model of the early feline visual pathway Djurfeldt (1997).

a simulation object can be a population of neurons, a complex projection, sensor/-effector arrays, a graphical display or data sources/recorders.

Links connect the *interfaces* of two objects, not unlike cables and connectors on computers. Several objects can connect to the same output interface. Links can contain separate channels of data, not unlike the pins of a connector. For example, it might be desirable to send both spike and continuous data through the same link. The recipient objects can then select which channels to use.

The user can easily write a new See module with his own, dedicated, C++ code. The complexity involved in moving data between components of a model is hidden by the interfaces. The programmer of a piece of dedicated simulation code only needs to declare an interface in order to create a source or sink for data. By building on previously defined classes, using inheritance and predefined object components, the user can concentrate on the essentials of his model. The C++ library, which can be loaded into the simulator without recompilation, provides a simulation object which can be created and manipulated from the scripting language. It is possible to prototype new simulation objects as scripts, and later implement them in a more efficient manner.

The modularity in See is coarse: In Genesis, a "module" can be a compartment or a channel object with scalar values communicated between them. In See, the module is instead typically a population of cells, and the link between objects typically convey chunks of information or vector data. This gives flexibility (the user is almost as free as if he wrote his own simulator) and high performance. It makes it easy to implement two forms of parallelism. On workstation clusters, different objects can be placed on different machines. On vector computers, objects are vectorized through the use of parallel versions of standard numerical libraries.

General Purpose Scripting Language

The scripting language in See is a superset of Scheme, a dialect of Lisp, provided by the Guile scheme interpreter. Since the scripting language is based on a *general purpose* language, the user has maximum flexibility. It makes it possible to use the same language for model specification and control of simulation experiments. It can further be used to write tools for data analysis and extensions to the simulator. In fact, similar to the *Emacs* editor Stallman (1981), many parts of See are *implemented* in the extension language. In the case of Emacs this type of architecture has proven very successful. See also the discussion in Cannon et al. (2007).

Simulation protocols and tools, e. g. for parameter search, can be written in the scripting language and saved for the future. This enables testing different models with the same protocol, and setting up procedures to do extended experiments overnight or longer. The choice to base the scripting language on Scheme was due to its expressiveness.

The See simulator framework combines the ease of use of a general purpose simulation package with the flexibility of a general purpose programming language by providing a set of libraries of C++ classes which are utilized from an embedded Scheme interpreter.

4.2 MUSIC

MUSIC is a standard for run-time exchange of data between parallel applications in a cluster environment. The standard is designed specifically for interconnecting large scale neuronal network simulators, either with each-other or with other tools (Ekeberg and Djurfeldt, 2009).

A typical usage example is illustrated in figure 4.2, where three applications (A, B, and C) are executing in parallel while exchanging data via MUSIC. We will refer to this as a *multi-simulation*, since the participating applications typically are neuronal simulators, or tools to support such simulators. In this example, application A produces runtime data which is then used by B and C. In addition, B and C mutually send data to each other. The data sent between applications can be either event based, such as neuronal spikes, or graded continuous values, for example membrane voltages.

The primary objective of MUSIC is to support multi-simulations where each participating application itself is a parallel simulator with the capacity to produce and/or consume massive amounts of data. This promotes *inter-operability* by allowing models written for different simulators to be simulated together in a larger system. It also enables *re-usability* of models or tools by providing a standard interface. The fact that data is spread out over a number of processors makes it non-trivial to coordinate the transfer of data so that it reaches the right destination at the right time. The task for MUSIC is to relieve the applications from handling this complexity.

Paper **IV** describes the addition of a MUSIC interface to the simulators NEST

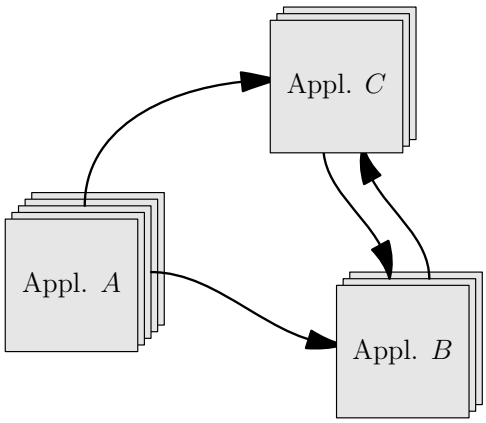

Figure 4.2: Illustration of a typical multi-simulation using MUSIC. Three applications, A, B, and C, are exchanging data during runtime.

(Diesmann and Gewaltig, 2002) and MOOSE and a multi-simulation involving a cortex model using integrate-and-fire units and a striatum model using multi-compartmental Hodgkin-Huxley units.

Design Goals

Portability

The MUSIC library and utilities have been designed to run smoothly on state-of-the-art high-performance hardware. For maximal portability, the software is written in C++, which is the de facto standard for current high-end hardware. MUSIC also provides a C-interface, making it possible for applications written in C or FORTRAN to participate in a MUSIC multi-simulation.

Most, if not all, current efforts in large scale neuronal simulations are based on the MPI standard. MUSIC is built on top of MPI, and uses it to run the different simulators. MUSIC provides means to allow each simulator to use MPI internally without interfering with the others.

MUSIC has been developed using two reference platforms: Intel-based multi-core workstations and the IBM BlueGene/L supercomputer. These platforms can be considered as two extremes, where the multi-core machine represents a small parallel environment while the BlueGene/L represents a large scale massively parallel supercomputer with special requirements. In particular, the compute nodes on the BlueGene/L do not support multiple threads or processes.

Simplicity

For MUSIC to be useful, it must be possible to adapt existing simulators so that they can participate in a multi-simulation without too much effort. We rely on the simulator developers to make these adaptations. An important design goal has therefore been to adapt the design to the typical structure of current simulators. It should be possible to add MUSIC library support without invasive restructuring of the existing code.

The primary requirements on an application using MUSIC is that it declares what data should be exported and imported and that it repeatedly calls a function at regular intervals during the simulation to allow MUSIC to make the actual data transfer.

Independence

The MUSIC interface ensures that each individual application does not need special adaptation to specific properties of other applications. The application only needs to adhere to the specification of the MUSIC interface in order to communicate with other applications performing complementary tasks. This makes the development of MUSIC-aware software independent of what other applications it will communicate with.

We hope that this will facilitate the development of general purpose tools. For example, a researcher can develop a tool for calculating synthetic EEG from simulation data. Via MUSIC, this tool should then be useful for anybody using any neuronal simulator which supports the common MUSIC interface.

Performance

The MUSIC API has been designed to allow for data transport of high bandwidth and low latency within the cluster. One means of ensuring the best use of the hardware while maintaining portability is to use the facilities of MPI for communication. MPI encapsulates software optimizations for specific hardware. By basing the interface on MPI we can benefit from such optimizations.

Extensibility

Where possible, MUSIC allows for extensions by the application programmer. Some classes in the MUSIC API (such as the index and data maps) can be subclassed in order to provide facilities not available directly in the API.

Spatial Distribution of Data

Communication between applications is handled by ports. Ports are named sources (output ports) or sinks (input ports) of data flows. The data to be communicated may be differently organized in process memory on the receiver side compared to

the sender side. The applications may run on different numbers of processes, and, the data may be differently distributed among the sender processes and the receiver processes, as is shown in Figure 4.3. How does MUSIC know which data to send where?

In MUSIC, there are two views of the data to be communicated over a connection. Data elements are enumerated differently according to these views. MUSIC uses *shared global indices* to enumerate the entire set of data to be sent over the connection while *local indices* enumerate the subset of data which is stored in the memory of a particular MPI process. Data does not need to be ordered in the same way according to the two views. For example, data stored in an array may be associated with an arbitrary subset of global indices in an arbitrary order.

The MUSIC library is informed about the relationship between global and local indices and how data is stored in memory during the setup phase. Two abstractions are used to carry this information:

The IndexMap maps local indices to global indices. That is, the IndexMap tells which parts of a distributed data array are handled by the local process and how the data elements are locally ordered.

The DataMap encapsulates how a port accesses its data. The DataMap contains an IndexMap. While an index map is a mapping between two kinds of indices, the data map also contains information about where in memory data resides, how it is structured, and, the type of the data elements. The type is used for marshalling when running on a heterogeneous cluster.

During setup every process of the application individually provides the port with a DataMap (or an IndexMap in the case of event ports).

Timing Considerations

Different applications may use different time steps and it is the responsibility of MUSIC to ensure that data is delivered at the appropriate time. In order to minimize handshaking, both parts of a connection pair locally calculate when the actual data transfer over MPI takes place. To ensure that these calculations produce predictable results, simulation time is internally represented using integers with a global micro-timestep common for all applications.

Simulation time is local for each application and MUSIC does not enforce unnecessary synchronization between these local clocks. Thus, an application producing data may be running ahead of another application which consumes the same data. MUSIC internally builds a schedule which ensures that data arrives at the appropriate local time in the receiving application. Scheduling becomes more complex when data is not only transferred in a feed-forward manner, i.e. when the connection graph contains loops. In this case MUSIC has to rely on the existence of sufficient delays in the simulated model, typically corresponding to axonal delays.

Figures 4.4 and 4.5 illustrate how MUSIC handles time when transferring continuous data over a connection. In figure 4.4, the sender application uses a shorter tick interval than the receiver. The sender side uses values sampled at the tick

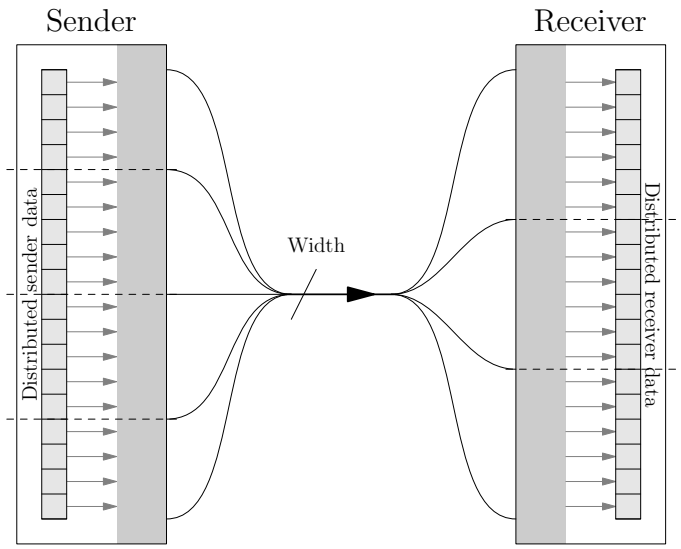

Figure 4.3: Data transfer over a connection from an application running in four processes to an application running in three processes. The light gray areas in the sender and receiver represents the MUSIC port. Dashed lines divide the application into distinct processes.

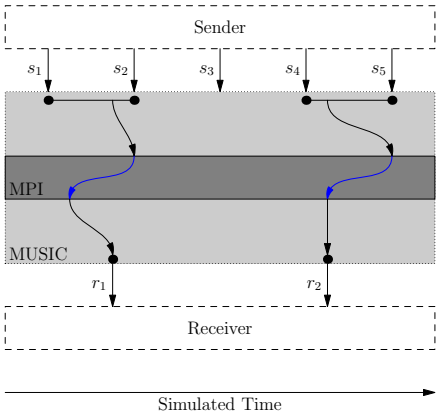

Figure 4.4: Transfer of data when sender has a shorter tick interval than the receiver.

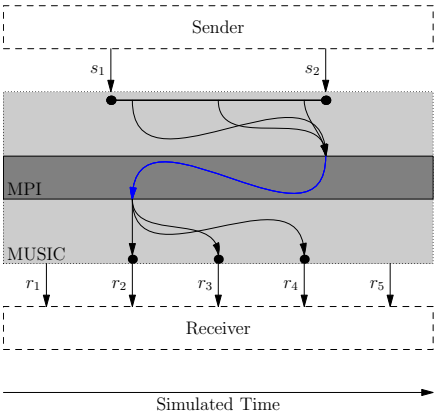

Figure 4.5: Transfer of data when sender has a longer tick interval than the receiver.

points to interpolate a value corresponding to the point in time when the receiver makes its tick call.

The dark middle area (labelled "MPI") is where the actual data transfer takes place. MUSIC makes use of the fact that the receiving application can run with its simulation clock set independently of the sender. The arrows going "backwards in time" in this area reflect the fact that the receivers clock is lagging. This makes it possible for data to arrive in time despite the fact that it was available later (e.g. at tick s_2) than when it was arriving (at r_1), when talking about simulated time.

Figure 4.5 illustrates what happens when the receiver of continuous data is calling tick faster then the sender. The sender will then buffer up values from the preceding and current ticks and transfer this at a suitable tick call. The receiver will portion these values out by interpolating at the appropriate ticks.

The strategy of having the receiver application running with a delayed local clock only works when the connection graph forms a directed acyclic graph (DAG). When loops occur it is necessary to allow for data arriving late, at least somewhere along each loop. MUSIC handles this via *acceptable latency* which is a property of event input ports. The receiving application declares how late, according to simulation time, data may arrive, thus giving MUSIC room for resolving the scheduling problem. In the case of continuous data, the application specifies a *delay* which fulfills the same purpose.

In figures 4.4 and 4.5, the sending application must be running ahead of the receiver in order to maintain the illusion that communication is instantaneous. Figure 4.6 illustrates the timing relation between sender and receiver along a real time axis (wallclock time) when the receiver accepts a delay of incoming data. This allows the receiver (B) to run ahead of the sender (A), thus creating the slack necessary to make schedules for communication loops.

4.3 The SPLIT simulator

SPLIT is a parallel neuronal network simulator for models using multi-compartmental Hodgkin-Huxley units. Paper **V** describes a series of improvements made to the simulator in order to support large-scale simulations. These improvements included the addition of the connection-set algebra described in paper **II** and resulted in the scaling performance presented below. The improved version of SPLIT was used in the simulations in papers **VI** and **VII**.

The development of parallel simulation in computational neuroscience has been relatively slow. Today there are a few publicly available parallel simulators. These are far from as general, flexible, and well documented as commonly used serial simulators, such as Neuron (Hines and Carnevale, 1997) and Genesis (Bower and Beeman, 1998). There is some development going on, however. For Genesis there is PGenesis and the development of a parallel version of Neuron has started. In addition there exists simulators like NCS (Frye, 2005), NEST (Morrison et al., 2005), and our own parallelizing simulator SPLIT (Hammarlund and Ekeberg,

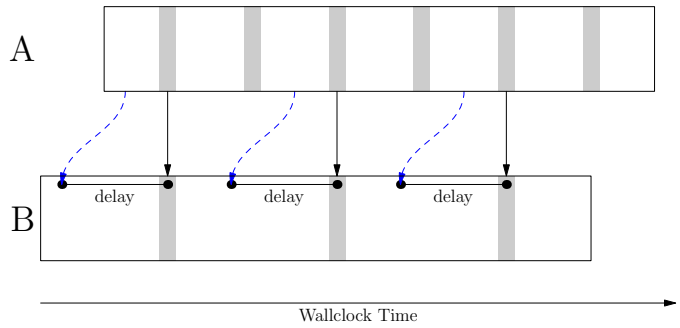

Figure 4.6: This figure illustrates how MUSIC can allow one application (B) to execute ahead of another (A) when transferring continuous data. The receiver (B) has specified a delay on the input port which means that the value to be delivered at each tick (gray areas) corresponds to a simulated time in A (blue arrows) which has already happened. Note that the tick times when MUSIC actually transfers data will be aligned on the real time axis, since blocking communication is used. In practice, one of the applications will have to wait for the other to reach the same point in its execution.

1998). However, they are in many ways still in the experimental and developmental stage.

The SPLIT simulator (Hammarlund and Ekeberg, 1998) was developed in the mid 90's with the aim of exploring how to efficiently use the resources of various parallel computer architectures for large-scale biophysically detailed neuronal network simulations. Since different computer architectures benefit from different kinds of optimizations, the code uses programming techniques which encapsulate such optimizations and hardware specific features from the rest of the program, and in particular from the neural network model specification. SPLIT has also served as a platform for experiments with communication algorithms.

SPLIT takes the form of a C++ library which is linked into the user program. The SPLIT API is provided by an object of the class `split` which is the only means of communicating with the library. The user program specifies the model using method calls on the `split` object. The user program is serial, and can be linked with a serial or parallel version of the library. Parallelism is thus completely hidden from the user. In the parallel case, the serial user program runs in a master process which communicates, through mechanisms internal to the SPLIT library, with a set of slave processes. On clusters, SPLIT uses MPI.

The library exploits data locality for better cache-based performance. To benefit from vector architectures, state variables are stored in sequence. It uses adjacency lists for compact representation of the neural projections and AER (Address Event Representation) for spike events (Bailey and Hammerstrom, 1988).

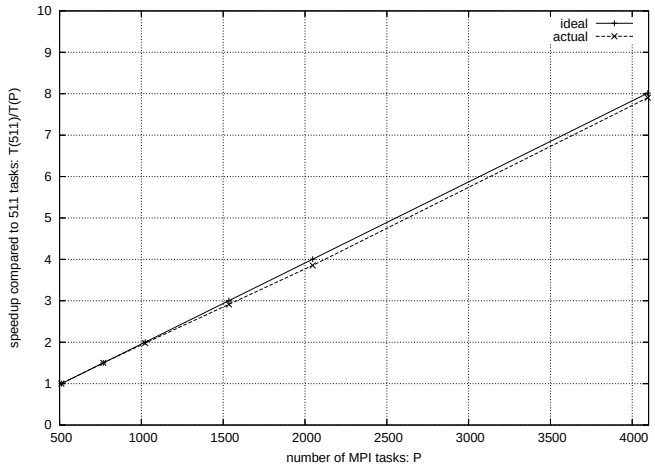

Figure 4.7: Speedup for model with 4 million cells and 2 billion synapses on BG/L simulated with SPLIT. Data points up to 2048 processors were collected on the Rochester BG/L while the last data point is obtained on the Watson Research BG/L.

The neurons in the model can be distributed arbitrarily over the set of slaves. This gives great freedom in optimizing communication so that densely connected neurons reside on the same CPU and so that axonal delays between neurons simulated on different slaves are maximized.

SPLIT also makes use of a novel abstraction, the connection set algebra, which implements an efficient domain decomposition of the connectivity meta data. With the connection set algebra, connectivity structure can be described in a modular way by using operators to make combinations of a set of basic connectivity types (paper **II**).

Figure 4.7 shows scaling results from simulations on the Blue Gene/L installations at Rochester, MN, and at Watson Research center in Yorktown Heights, NY. The figure shows the speedup for a model with 4 million cells and 2 billion synapses on BG/L up to 4096 processors. Data points up to 2048 processors were collected on the Rochester BG/L while the last data point is obtained on the Watson Research BG/L.

Chapter 5

A large-scale model of the cerebral cortex

In this chapter, the technology developed in previous chapters is applied in the simulation of a large-scale model of cortical layers II/III. This chapter partly builds on material in paper **VI**.

5.1 The cortex as a recurrent attractor network

The view of the cortex as an attractor network has its origin more than fifty years back in Hebb´s cell assembly theory (see, e.g., Fuster (1995) for a review). Hebb suggested that the functional unit of the cortex is a subset of neurons which are repeatedly active together, and that such a *cell assembly* is the basis of mental representation. The thought of an apple would invoke one cell assembly, the thought of an orange another. The theory has since been mathematically instantiated in the form of the Willshaw-Palm (Willshaw and Longuet-Higgins, 1970; Palm, 1982) and Little-Hopfield models (Hopfield, 1982) and has subsequently been elaborated on and analyzed in great detail (Amit, 1989; Hertz et al., 1991). This has resulted in the view of the persistent firing of cell assemblies as attractors in a dynamic system.

The olfactory cortex (Haberly and Bower, 1989) as well as the hippocampal CA3 field (Treves and Rolls, 1994) have previously been perceived and modeled as prototypical neuronal auto-associative attractor memory networks. The connections in the network form a landscape of attractors, each representing a memory. When the state of the system slips into the attractor, the memory has been recalled. More recently, sustained activity in an attractor memory of a similar kind has been proposed to underlie prefrontal working memory, although in this case the attractor state itself and not the connectivity is assumed to hold the memory (Compte et al., 2000).

Minicolumns in the cerebral cortex

If cell assemblies, or attractor states, are the basis of cortical function, how do they relate to cortical anatomy? In a classic work on the visual system, Hubel and Wiesel (Hubel and Wiesel, 1977) penetrated the primary visual cortex with a recording electrode. They found that cells responded most strongly to a specific orientation of an oblong bar in the visual field, and that the preferred angle seemed to shift discretely as the electrode moved tangentially to the cortical surface, while cells along a line normal to the surface tended to have similar response properties. They deduced that the basic unit of cortical organization must be what they called a *functional column* and suggested that the cortex is a lattice of such columns. They also suggested that the sets of such columns are grouped into larger entities, *hypercolumns*, that together form a complete representation of all possible attribute values within each region of retinotopic space.

Later, Mountcastle (1978) suggested the concept of anatomical minicolumns. These were described in detail by Peters and Sethares (1991). Could such a minicolumn, consisting of some hundred neurons, be the basic functional unit of the cortex? If so, a Hebbian cell assembly may consist of a set of such columns and the activation of these columns would correspond to entering a dynamic attractor of the cortical network.

The organization of visual cortex into minicolumns and hypercolumns has inspired our view of cortical associative memory, which has been expressed in the form of an abstract neural network model (Lansner et al., 2003; Sandberg et al., 2002, 2003), and in biophysically detailed models (Fransén and Lansner, 1998; paper **VII**).

5.2 Methods

The simulations in paper **VI** are based upon a model of layers II/III of the association cortex of the rat. It is an upscaled version of the model presented in paper **VII**.

The overall architecture of our model is illustrated in Figure 5.1. Figure 5.1A illustrates the geometric layout of a subset of 100 hypercolumns in the plane of the cortical sheet, each marked with a distinct color. Each hypercolumn consists of 100 minicolumns. Each minicolumn contains 30 *pyramidal cells* which excite each other through short-range axons and excite pyramidal neurons of other minicolumns through long-range axons (Figure 5.1B). This long-range projection constitutes the memory matrix of the attractor memory and forms the cell assemblies: Only minicolumns belonging to the same memory pattern, or cell assembly, excite each other.

Each hypercolumn also contains a population of 100 inhibitory *basket cells* which are excited by the minicolumns of the local hypercolumn. They, in turn, inhibit the pyramidal cells of the local hypercolumn, thereby normalizing activity. This enables the hypercolumn to operate like a winner-take-all module, where different patterns can compete. Each minicolumn also contains 2 inhibitory *RSNP cells* (regular

spiking non-pyramidal) which contact the local pyramidal cells. The abstract neural network model, upon which the long-range connectivity of the present model is based, suggests an additional way in which cell assemblies can compete. This means of competition has been realized in the present model through long-range axons from pyramidal cells to RSNP cells of minicolumns belonging to other cell assemblies, which indirectly suppress their activity. Such connections have not yet been identified anatomically. Connectivity is compatible with experimental data to the extent that such data is available. However, for simplicity, we have made borders of mini- and hypercolumns sharp, in contrast to the local approximately Gaussian structure observed experimentally (see, e.g., Buz'as et al., 2006). For details, see paper **VII**. A separate set of cells model cortical layer IV which provides input to the pyramidal cells described above. External input to the attractor memory is provided as simulated synaptic events in these cells.

Cells are modeled using the Hodgkin-Huxley formalism described in section 2.9. A pyramidal cell in our model consists of six compartments, one state variable each for the potentials, carrying up to five ionic currents, one to two state variables per current. Some compartments have a flow of calcium into an intracellular store described by an additional state variable. Furthermore, some synapses carry a separate flow of calcium with yet another associated state variable. Synapses are generally governed by three state variables, one for the degree of opening, and two for short term changes in synapse strength (facilitation and depression).

For the simulations in papers **VI**–**VII** and here, an *orthogonal* set of non-overlapping memory patterns was formed. One minicolumn was selected from each hypercolumn to form one pattern. A long-range connection between two distant minicolumns was formed probabilistically if the two minicolumns belonged to the same pattern. Each pyramidal cell, thus, received long-range excitation only from a subset of the cells in the pattern. Similarly, each RSNP cell received excitation from a subset of pyramidal cells belonging to minicolumns of foreign patterns.

5.3 Simulation results

Figure 5.2 shows the spiking activity of a simulation of 49 hypercolumns (100 minicolumns each). Each row of the raster plot shows the spikes of one neuron. The lower portion (the first 9800 cells) of the raster shows activity in all RSNP cells, the mid portion (147000 cells) shows the pyramidal cells, and the upper portion (4900 cells) shows the basket cells. The long-range pyramidal-pyramidal and pyramidal-RSNP synapses store orthogonal memories. As a consequence of stimulation of the cells in layer IV between t=0.5–0.64 s and t=1.5–1.64 s, the network state can be seen switching from a *ground state* to an active *memory state*. Only cells in layer IV representing a part of one of the memory patterns stored in the interhypercolumnar memory matrix are stimulated. This partial pattern is quickly completed to the full memory pattern. This behavior was robust for all memories stored and shows that, although each pyramidal cell only connects to a random subset of cells in the

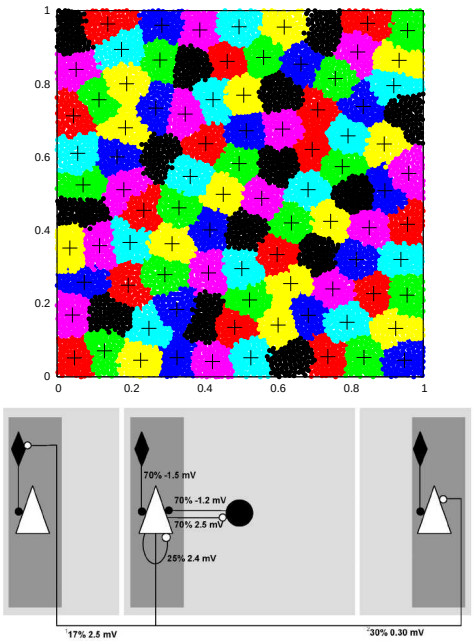

Figure 5.1: A. Geometric layout of 100 hypercolumns consisting of 100 minicolumns each. B. Schematic connectivity of the model. White triangles, pyramidal cells, project locally as well as to pyramidal cells in other minicolumns belonging to the same cell assembly and RSNP cells in minicolumns belonging to other assemblies. Black circle, basket cell, normalizes activity in the local hypercolumn. Black rhombs, RSNP cells provides local inhibition of pyramidal cells. Percentages and voltages show connection probability and size of EPSP (excitatory postsynaptic potential) respectively.

pattern, on a population level the cells has formed a cell assembly corresponding to the pattern. Apart from pattern completion, the model is capable of all the functionality usually ascribed to attractor networks, such as noise reduction and resolution of ambiguity.

In the present version of the model, the ground state is characterized by oscillations at a frequency of approximately 15 Hz (Figure 5.4D), where the oscillations of individual minicolumns are phase-locked to other minicolumns in the containing hypercolumn. The coexistence of a stable ground state with active memory states was first shown in a model of delay period activity in the prefrontal cortex (Amit and Brunel, 1997). In the ground state of our model, pyramidal cells fire at around 0.1 Hz. The 15 Hz rhythm thus emerges as a collective network level phenomenon. The fact that it only appears when there is no input to the network and the net-

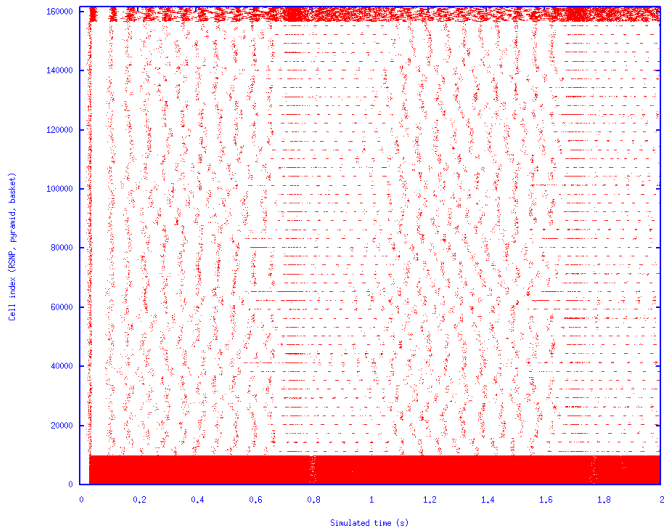

Figure 5.2: Raster plot for a simulation of 49 hypercolumns. 100 minicolumns are shown per hypercolumn. The lower portion (first 9800 cells) of the raster shows activity in all RSNP cells, mid portion (147000) shows pyramidal cells, and upper portion (4900) basket cells. The long-range pyramidal-pyramidal and pyramidal-RSNP synapses store orthogonal memories. As a consequence of stimulation (simulated incoming action potentials) of another part of the network at t=0.5 s and t=1.5 s, the network state can be seen switching from a ground state to an active memory state.

work is not in one of its active memory states is suggestive the class of α rythms, which has been proposed to reflect cortical idling (Pfurtscheller et al., 1996). The frequency lies close to the α band and is consistent with the cat μ rythm (Hughes and Crunelli, 2005).

In the active state, only the pyramidal cells of a single minicolumn are active in each hypercolumn. In this state, pyramidal cells fire at 10–15 Hz, basket cells at 50 Hz and RSNP cells at 25–35 Hz. One particularly interesting phenomenon, which consistently arises in our simulations for a broad range of parameters and all model sizes, is a rhythmic modulation of pyramidal cells during active states with a frequency of 25–40 Hz (Figure 5.4D). This is reminiscent of the γ band activity observed in working memory tasks (Pesaran et al., 2002).

One of the experimental techniques used to study activity in populations of neurons is to record changes in color of a voltage sensitive dye (VSD) (Grinvald et al., 1994). Figure 5.3 shows a synthesized VSD-signal for a network with 49 hypercolumns during three phases of activity. The signal was computed as the low-

pass filtered sum of the membrane potentials of all cells in each minicolumn. Figure 5.3A shows the ground state condition. Figure 5.3B shows the VSD-signal just after stimulation of a partial pattern. In Figure 5.3C, the network has completed the shift to the active memory state.

Our model exhibits some emergent phenomena that have also been observed in the brain. When the network attains an active memory state, pyramidal cells participating in the active cell assembly get bombarded with synaptic events. This elevates their membrane potential, so that the global shift of dynamic state is reflected in a shift of their membrane potential from a hyperpolarized state to a more depolarized state (Figure 5.4A). These states are very similar, respectively, to the DOWN and UP states that have been observed physiologically (Steriade et al., 1996; Cossart et al., 2003). These shifts cause many cells to have a bimodal distribution of membrane potentials (Figure 5.4B), also consistent with physiology (Anderson et al., 2000). Despite the regularities seen on a network level, the firing of individual pyramidal cells is Poisson distributed in the active state (Figure 5.4B) (see, e.g., Bedard et al., 2006). One particularly attractive feature of the model is that it is robust to perturbation of parameters (paper **VII**), which is to be expected from a biological system.

The largest simulation was performed on 8192 nodes of a Blue Gene/L supercomputer. The simulation occupied 336 MB of memory at each node, giving a total of 2.8 TB. The model used consisted of 22 million neurons and 11 billon synapses which corresponds to a cortical surface area of 16 cm^2, comparable to the cortex of a small mammal. While real pyramidal cells have 10000 synapses, the average number of synapses per neuron is only 500 in our model due to the orthogonality of the memory matrix and due to the lack of connections to other cortical areas. One question was if such a large patch of cortex could maintain a stable attractor state despite the significant propagation delays caused by axonal conduction time. Our results show that this is indeed the case.

5.4 Memory capacity

The memory capacity of attractor networks has been explored in highly abstract artificial neural networks (ANNs) (see, e.g., Johansson et al., 2001). In this section, we study to what extent network models closer to biology compare to ANNs. In previous sections of this chapter, a biologically detailed model of cortical layers II/III, using spiking, multicompartmental neurons and Hodgkin-Huxley formalism for ion channels, was presented. To what extent can this type of model match artificial recurrent networks with regard to memory capacity?

We ran simulations of the detailed model and a Bayesian Confidence Propagating Neural Network (BCPNN) (Johansson and Lansner, 2006a), counting correctly recalled memory patterns. Long-range connections in the detailed model were determined using a weight matrix w_{ij} obtained by training the ANN. After training, connections in both networks were set up probabilistically with uniform connection

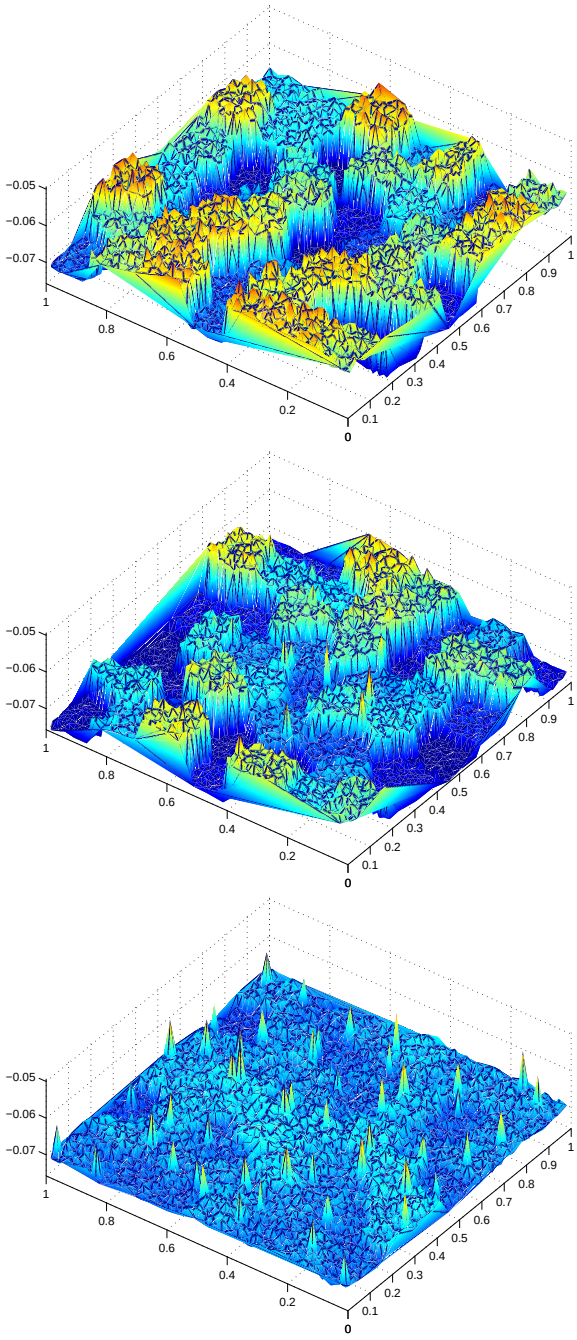

Figure 5.3: Simulated VSD-signal of 4900 minicolumns in a simulated cortical patch of size 3x3mm. A. The ground state with waves of hypercolumnar activity. B. A part of a memory pattern is stimulated through layer IV. C. The network has attained an attractor memory state.

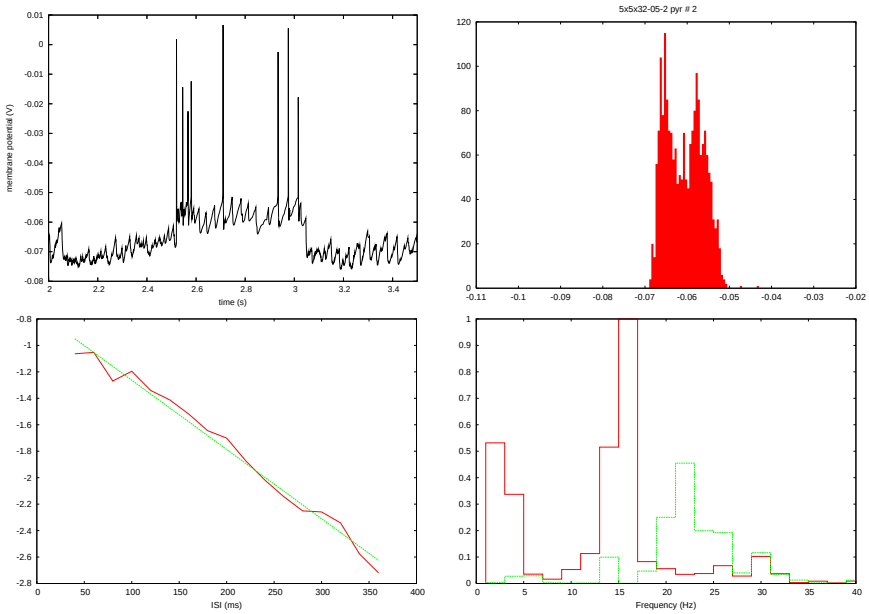

Figure 5.4: A. A model pyramidal neuron switching from a hyperpolarized state (similar to DOWN state) to a more depolarized state (similar to an UP state) and back. The mean membrane potential becomes elevated at $t = 2.5$ s due to network activity when the neuron starts participating in an active attractor state. The attractor state was activated through simulated electrical stimulation of other member neurons. B. Membrane potential histogram. Many cells show a bimodal distribution of membran potentials. C. Exponentially distributed activity of pyramidal cells in the depolarized state in a network with 25 hypercolumns. ISI:s (9443 spikes) were collected from all pyramidal cells during a total of 2 s of simulated time when the network was in a globally active state, i.e. one in which a subset of cells shows UP-state activity. The logarithm of the distribution of ISI was plotted as a function of ISI length. An exponential distribution was fitted to the data and is shown as a straight line. r^2 was 0.98 for the exponential fit and 0.86 for a power-law distribution (not shown). D. Normalized power spectrum for VSD signal from one minicolumn. Red: ground state. Green: active state.

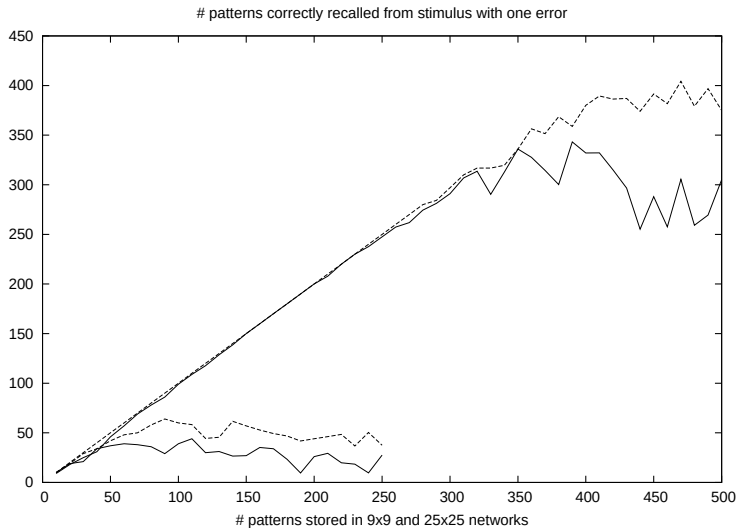

Figure 5.5: Memory capacity assessment for two network sizes (9x9 and 25x25 minicolumns). The x-axis shows the number of random memory patterns stored in the long-range connectivity matrix. The y-axis shows the number of correctly recalled memory patterns. Dashed lines shows performance of layer II/III model, solid lines shows ANN performance.

strengths: For each pair of units ((i, j); elementary units in the ANN, minicolums in cortex model), the absolute value $abs(w_{ij})$ of the weight matrix value was scaled to a connection probability P_{ij}. In the cortex model, P_{ij} was used to determine the probability in a Bernoulli trial for each possible pair of cells (c_i, c_j) where c_i was a cell of the presynaptic minicolumn and c_j a cell of the postsynpatic minicolumn. A positive w_{ij} implied long-range excitatory pyramidal-pyramidal connections (with no pyramidal-RSNP connections), while a negative value implied long-range pyramidal-RSNP connections (with no pyramidal-pyramidal connections; see section 5.2).

Figure 5.5 shows an assessment of the memory capacity of the ANN and cortex models, for two different network sizes. For each recall cycle, the models were stimulated with a partial pattern, 2/3 of one of the stored patterns $\xi^{(k)}$. The stored pattern was considered correctly recalled if the activity sampled between 200–300 ms after stimulation corresponded to the stored pattern $\xi^{(k)}$ with at most one mismatch. The lower curves show performance for a network with 9 hypercolumns of 9 minicolumns each (9x9). The upper curves show the performance of a 25x25 network. Memory performance varies in a similar way for the two classes of model (ANN vs detailed cortex model) when increasing the number of patterns stored in

the network. More specifically, the detailed model and the ANN display a similar maximum memory capacity, demonstrating that a spiking network with detailed neuronal and synaptic dynamics can perform the function of a robust attractor memory. While a full understanding of cortical memory must cover more than static memory patterns, we regard the present approach as an essential step in bridging the gap between attractor theories of memory function and cortical anatomy and electrophysiology.

Chapter 6

Outlook

In the simulation results of papers **VI** and **VII**, when the network attains one of the attractor states associated with a stored memory, network spiking activity exhibits a γ-like rythm. But this oscillation frequency is *higher* than the firing frequency of any of the excitatory neurons participating in this activity! That is, neurons of a minicolumn "take turns" in contributing to the γ rythm. The rythm of the attractor state, which is clearly visible in the raster plot in Figure 5.2, is not as visible at the level of individual cells—we are seeing a network-level phenomenon. In fact, the attractors of an attractor network are also network-level phenomena. While traces of attractor states are reflected in single-cell activity, the attractor state is not always discernible at the single-cell level. Attractor states in the networks of papers **VI** and **VII** are also not causally dependent on the activity of any single cell in the sense that we can remove any cell without loosing the attractor state. The γ-like rythm and the attractors are *emergent* properties at the network level which occur due to the interaction of the cells in the network, similarly to how air waves can arise due to the interaction of air molecules.

The model was designed using the combined top-down and bottom-up approaches described in section 2.9. The emergent attractor dynamics was not a surprise, but rather *imposed* from a pre-existing, more abstract, description, even though it is an interesting result that units of detailed minicolumns built from multiple cells with Hodgkin-Huxley dynamics collectively perform as a robust attractor network. But, when moving from an abstract model of an attractor network to a biologically detailed model, new, unpredicted, phenomena arise, among them the network-wide γ-like oscillation when recalling a memory. One role of such cortical rythms could be to increase efficiency of communication, since multiple action potentials arriving within a small time window have more effect on the postsynaptic cell than if they arrived asynchronously. The rythmicity also means that cells are more likely to fire in response to synaptic input during one part of the cycle than in other parts. If we want to study phenomena where this phase dependency matters, we could move back again to an abstract description where minicolumns are treated

as units, as in the abstract attractor network, but where we also include the *phase* of the oscillation as a state varible. Such models have been studied previously (e.g. Rao et al., 2008). The conclusion we can draw from this example is that it is worthwhile to move back and forth between models at different levels of abstraction. We might start from a simple abstract model, use this as a functional hypothesis for a biologically detailed model, and move back to a different abstract model.

The model of papers **VII** and **VI** can be seen as an initial attempt to understand information processing in cortical layers II/III. The model is unique in that it simultaneously incorporates a relatively high degree of biological detail at the same time as it performs the information processing functions of an attractor network. However, the current model only incorporates point attractors (or limit cycles depending on viewpoint) while the real cortex needs to deal with dynamic patterns. If we look ahead towards the challenges of the future, we need to explore further how the cortex handles spatiotemporal patterns. Another near-term goal is to understand the interplay with other cell groups spread through the cortical layers. The apparent existence of a canonical microcircuit for the cerebral cortex urges us to explain its function. While many feel this holy grail of cortical physiology is now within reach, we can only conclude that it has not yet been found.

The understanding of the canonical microcircuit is connected to the question of how brain areas work together as systems of networks. This implies challenges for hardware, software as well as modeling. In paper **VI** we have seen a detailed model of a cortical region which still only corresponds to a part of a brain area of a higher animal. In addition, note that this work was based only on a handful of simulation runs, while, in order to probe properties of the network, such as memory capacity (see section 5.4), thousands of runs are required. In this respect, the large network of paper **VI** is still out of reach also on the impressive BG/L supercomputer at Watson Research. The road ahead thus means to initially use more abstract models for the exploration of systems of networks until the advent of still more powerful supercomputers. An exciting possibility for the near future is to use the wafer-scale integrated neural hardware developed within the FACETS project (Schemmel et al., 2008) which enables emulations of large networks 10000 times *faster* than real-time (see section 2.1). Simulating systems of networks becomes easier if taking a modular approach (see section 3.3), for example through using a simulator providing a modular framework (section 4.1) or using a framework such as MUSIC (section 4.2) to connect multiple simulators together. When modeling systems of networks, the question of how to connect networks together is a relatively unexplored territory (see, e.g., Johansson and Lansner, 2006b; Dileep and Hawkins, 2005). Also, while this thesis discusses techniques for simulating cortical networks, a full understanding of such networks will never come about without understanding the complex network of diverse cortical areas as components in a larger system, together with the thalamus, the basal ganglia, the cerebellum, the additional components of the mediotemporal lobe, and, modulatory nuclei of the basal forebrain and brain stem, with associated circuitry.

If we look far ahead, the ultimate challenge for a thorough understanding of

the brain is how consciousness arises as a result of brain activity. According to the global workspace theory of consciousness (Baars, 2005, 1989), consciousness provides access to information between brain functions that are otherwise separate. Using Baars theory as a starting point, Gaillard et al. (2009) recorded intracranial electroencephalogram (iEEG) from ten patients during non-conscious as well as conscious processing of words. Conscious processing, in contrast to non-conscious, was charaterized by sustained voltage changes, large increases in spectral power in the γ band, increases in long-distance phase synchrony, and increases in long-range Granger causality (Granger, 1969). Could consciousness be explained as a brain-wide attractor state, binding together participating networks, shifting in shape and memberships over time, creating a flexible, immaterial, information processing structure reconfigurable at the 100 ms timescale?

Returning to the discussion of emergence, we should remember that sound is better described with the wave equation than as a particle system. Might there be completely new abstractions to discover and use as tools on our long journey of modeling starting with networks and moving to larger and larger systems? Might the ultimate object of our study, the human mind, be best described with very different tools than those imaginable today?

In this theses we have focused on the development and use of tools to conquer the complexity of neural systems. However, as a final word, we must never forget that there is more to technology than its tools:

> *Socrates to Glaucon:* [Τῶν] δὲ ἄλλων ὀργάνων οὐδὲν οὐδένα δημιουργὸν οὐδὲ ἀθλητὴν ληφθὲν ποιήσει, οὐδ᾽ ἔσται χρήσιμον τῷ μήτε τὴν ἐπιστήμην ἑκάστου λαβόντι μήτε τὴν μελέτην ἱκανὴν παρασχομένῳ (Plato, 380 BCE).

> [No] tool will make a man be an artist or an athlete by his taking it in hand, nor will it be of any service to those who have neither acquired the science of it nor sufficiently practised themselves in its use

Chapter 7

Conclusions

This thesis has presented a set of contributions to the technology involved in large-scale simulations of neuronal networks. Chapter 2 and paper **I** gave a perspective on modeling in neuroscience in general and on the role of large-scale models in particular. Chapter 2 also provided a terminology which is useful when discussing models in neuroscience. For example, it was pointed out that the questions whether a model is *abstract*, *detailed* or *realistic* are to some degree mutually independent. The concept of model *explicitness* was introduced. The role of the model in neuroscience was discussed as well as the strategy of a combined top-down and bottom-up approach during model development.

In section 2.11 and paper **II** we presented a novel formalism for the description of connectivity in neuronal network models. This formalism, the connection-set algebra, can be used both to provide concise and unambiguous descriptions of connectivity in papers in computational neuroscience, and as a component of simulator scripting languages. Chapter 3 provided a review of issues related to simulation of neuronal network models and a discussion of simulation tools, while chapter 4 reviewed three contributions to simulator technology. An early example of the use of a general purpose, interpreted, scripting language for model description and simulator extension was given in the See simulator (section 4.1, paper **III**). Two different approaches to modularity when simulating systems of networks were provided in the See simulator and in the MUSIC API and library (section 4.2). The See simulator has been used in a simulation of the primary visual pathway of the cat (Djurfeldt, 1997) while MUSIC was used in the memory capacity measurements of section 5.4. Paper **V** shows a set of improvements to the SPLIT simulator (section 4.3), enabling the simulation, in chapter 5, of a model with 11 billion synapses and 22 million neurons (paper **VI**).

Chapter 5 reviewed a neuronal network model of layers II/III of the neocortex built with biophysical model neurons. Several key phenomena seen in the living brain appeared as emergent phenomena in the simulations. The memory capacity of two models of different network size was measured and compared to that of an artificial neural network. We conclude that the layer II/III model performs as a robust auto-associative memory. Furthermore, paper **VII** demonstrated that the model is robust against perturbation of parameters, which is a hallmark of correct models of living systems.

Bibliography

Amit, D. (1989). *Modeling Brain Function: The World of Attractor Neural Networks*. Cambridge University Press, New York.

Amit, D. J. and Brunel, N. (1997). Model of global spontaneous activity and local structured activity during delay periods in cerebral cortex. *Cereb. Cortex*, 7:237–252.

Anderson, J., Lampl, I., Reichova, I., Carandini, M., and Ferster, D. (2000). Stimulus dependence of two-state fluctuations of membrane potential in cat visual cortex. *Nat. Neurosci.*, 3(6):617–621.

Aristotle (350 BCE). *I–III*. Loeb Classical Library. Harvard University Press, Cambridge, MA, USA.

Aviel, Y., Mehring, C., Abeles, M., and Horn, D. (2003). On embedding synfire chains in a balanced network. *Neural Computation*, 15(6):1321–1340.

Baars, B. J. (1989). *A cognitive theory of consciousness*. Cambridge University Press, Cambridge, UK.

Baars, B. J. (2005). Global workspace theory of consciousness: toward a cognitive neuroscience of human experience. In Laureys, S., editor, *Prog. Brain Res.*, volume 150, chapter 4, pages 45–53. Elsevier.

Bailey, J. and Hammerstrom, D. (1988). Why VLSI implementations of associative VLCNs require connection multiplexing. In *International Conference on Neural Networks*, San Diego, U.S.A.

Bastian, P., Birken, K., Johannsen, K., Lang, S., Neuss, N., Rentz-Reichert, H., and Wieners, C. (1997). Ug - a flexible software toolbox for solving partial differential equations. *Computing and Visualization in Science*, 1:27–40.

Bedard, C., Kroger, H., and Destexhe, A. (2006). Does the 1/f frequency scaling of brain signals reflect self-organized critical states? *Phys Rev Lett*, 97(11):118102.

Binzegger, T., Douglas, R. J., and Martin, K. A. C. (2004). A quantitative map of the circuit of cat primary visual cortex. *J. Neurosci.*, 39:8441–8453.

Bower, J. M. and Beeman, D. (1998). *The book of GENESIS: Exploring realistic neural models with the GEneral NEural SImulation System.* Springer-Verlag, New York, 2 edition. ISBN 0-387-94938-0.

Brette, R. (2006). Exact simulation of integrate-and-fire models with synaptic conductances. *Neural Computation*, 18(8):2004–2027.

Brette, R. and Gerstner, W. (2005). Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J. Neurophysiol.*, 94:3637–3642.

Brette, R., Rudolph, M., Carnevale, T., Hines, M., Beeman, D., Bower, J. M., Diesmann, M., Morrison, A., Goodman, P. H., Harris, F. C., Jr., Zirpe, M., Natschläger, T., Pecevski, D., Ermentrout, B., Djurfeldt, M., Lansner, A., Rochel, O., Vieville, T., Muller, E., Davison, A. P., Boustani, S. E., and Destexhe, A. (2007). Simulation of networks of spiking neurons: a review of tools and strategies. *J. Comp. Neurosci.* Online.

Brunel, N. (2000). Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. *J. Comp. Neurosci.*, 8:183–208.

Buz'as, P., Kov'acs, K., Ferecsk'o, A. S., Budd, J. M. L., Eysel, U. T., and Kisv'arday, Z. F. (2006). Model-based analysis of excitatory lateral connections in the visual cortex. *J. Comp. Neurol.*, 499:861–881.

Cannon, R. C., Gewaltig, M.-O., Gleeson, P., Bhalla, U. S., Cornelis, H., Hines, M. L., Howell, F. W., Müller, E., Stiles, J. R., Wils, S., and De Schutter, E. (2007). Interoperability of neuroscience modeling software: Current status and future directions. *Neuroinform.*, 5:127–138.

Churchland, P. S. and Sejnowski, T. J. (1992). *The Computational Brain.* The MIT Press, Cambridge, Massachusetts.

Compte, A., Brunel, N., Goldman-Rakic, P. S., and Wang, X. J. (2000). Synaptic mechanisms and network dynamics underlying spatial working memory in a cortical network model. *Cereb Cortex*, 10(9):910–23.

Cornelis, H., Edwards, M., Coop, A. D., and Bower, J. M. (2008). The cbi architecture for computational simulation of realistic neurons and circuits in the genesis 3 software federation. *BMC Neuroscience*, 9(Suppl 1):P88.

Cossart, R., Aronov, D., and Yuste, R. (2003). Attractor dynamics of network up states in the neocortex. *Nature*, 423(6937):283–8.

Crook, S. M., Gleeson, P., and Silver, R. A. (2007). NetworkML: Level 3 of the neuroml standards for multiscale model specification and exchange. In *Soc. Neurosci. Abstr.*

Davison, A. P., Brüderle, D., Eppler, J., Kremkow, J., Muller, E., Pecevski, D., Perrinet, L., and Yger, P. (2009). PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2:1–10.

Denk, W. and Horstmann, H. (2004). Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure. *PLoS Biology*, 2(11):1900–1909.

Destexhe, A., Rudolph, M., and Pare, D. (2003). The high-conductance state of neocortical neurons in vivo. *Nat. Rev. Neurosci.*, 4:739–751.

Diesmann, M. and Gewaltig, M.-O. (2002). NEST: An environment for neural systems simulations. In Plesser, T. and Macho, V., editors, *Forschung und wisschenschaftliches Rechnen Beitrage zum Heinz-Billing-Preis 2001*, volume 58, pages 43–70. Göttingen: Ges. für Wiss. Datenverarbeitung.

Dileep, G. and Hawkins, J. (2005). A hierarchical bayesian model of invariant pattern recognition in the visual cortex. In *IJCNN 2005*.

Djurfeldt, M. (1997). Modeling the primary visual pathway of the cat, using spiking Hodgkin-Huxley style units. Tech. Rep. TRITA-NA-E9742, Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden.

Djurfeldt, M. and Lansner, A. (2007). Large-scale modeling of the nervous system. Tech. rep., International Neuroinformatics Coordinating Facility (INCF). Workshop report.

Eberhard, J., Attinger, S., and Wittum, G. (2004). Coarse graining for upscaling of flow in heterogeneous porous media. *Multiscale Model. Simul.*, 2(2):269–301.

Edelman, G. M. (1987). *Neural Darwinism*, chapter 3. Basic Books, New York, USA.

Einevoll, G. T., Pettersen, K. H., Devor, A., Ulbert, I., Halgren, E., and Dale, A. M. (2007). Laminar population analysis: Estimating firing rates and evoked synaptic activity from multielectrode recordings in rat barrel cortex. *J. Neurophysiol.* In press.

Ekeberg, Ö. and Djurfeldt, M. (2009). *MUSIC — Multi-Simulation Coordinator, Users Manual*. INCF, Karolinska Institutet, Nobels väg 15 A, SE-171 77 Stockholm, Sweden, 1st edition. `http://software.incf.org/software/music`.

Ekeberg, Ö., Hammarlund, P., Levin, B., and Lansner, A. (1993). SWIM — A simulation environment for realistic neural network modeling. In Skrzypek, J., editor, *Neural Network Simulation Environments*, pages 47–71. Kluwer, Hingham, MA, USA.

Ekeberg, Ö., Wallén, P., Lansner, A., Tråvén, H., Brodin, L., and Grillner, S. (1991). A computer based model for realistic simulations of neural networks. i: The single neuron and synaptic interaction. *Biol. Cybern.*, 65(2):81–90.

Fourcaud-Trocme, N., Hansel, D., van Vreeswijk, C., and Brunel, N. (2003). How spike generation mechanisms determine the neuronal response to fluctuating inputs. *J. Neurosci.*, 23:11628–11640.

Fransén, E. and Lansner, A. (1998). A model of cortical associative memory based on a horizontal network of connected columns. *Network: Computation in Neural Systems*, 9:235–264.

Frye, J. (2005). http://brain.cse.unr.edu/ncsdocs/.

Fuster, J. M. (1995). *Memory in the Cerebral Cortex*. The MIT Press, Cambridge, Massachusetts.

Gaillard, R., Dehaene, S., Adam, C., Cl'emenceau, S., Hasboun, D., Baulac, M., Cohen, L., and Naccache, L. (2009). Converging intracranial markers of conscious access. *PLOS Biology*, 7(3):1–21.

Gleeson, P., Steuber, V., and Silver, R. A. (2007). neuroConstruct: A tool for modeling networks of neurons in 3D space. *Neuron*, 54(2):219–235.

Goddard, N. H., Hucka, M., Howell, F., Cornelis, H., Shankar, K., and Beeman, D. (2001). Towards neuroml: model description methods for collaborative modelling in neuroscience. *Philos. Trans. R. Soc. Lond. B Biol. Sci.*, 356(1412):1209–28.

Granger, C. W. J. (1969). Investigating causal relations by econometric models and cross-spectral methods. *Econometrica*, 37(3):424–438.

Grinvald, A., Lieke, E. E., Frostig, R. D., and Hildesheim, R. (1994). Cortical point-spread function and long-range lateral interactions revealed by real-time optical imaging of macaque monkey primary visual cortex. *J Neurosci*, 14(5 Pt 1):2545–68.

Haberly, L. B. and Bower, J. M. (1989). Olfactory cortex: model circuit for study of associative memory? *Trends Neurosci*, 12(7):258–64.

Haeusler, S. and Maass, W. (2007). A statistical analysis of information-processing properties of lamina-specific cortical microcircuit models. *Cerebral Cortex*, 17:149–162.

Hammarlund, P. and Ekeberg, O. (1998). Large neural network simulations on multiple hardware platforms. *J Comput Neurosci*, 5(4):443–59.

Hebb, D. O. (1949). *The Organization of Behavior*. John Wiley, New York.

Hertz, J., Krogh, A., and Palmer, R. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood, CA.

Hines, M. (1993). NEURON—a program for simulation of nerve equations. In Eeckman, F., editor, *Neural Systems: Analysis and Modeling*, pages 127–136. Kluwer, Norwell, MA, USA.

Hines, M. and Carnevale, N. T. (1997). The neuron simulation environment. *Neural Comput.*, 9:1179–1209.

Hines, M. L., Morse, T., Migliore, M., Carnevale, N. T., and Shepherd, G. M. (2004). Modeldb: A database to support computational neuroscience. *J Comput Neurosci*, 17(1):7–11.

Hodgkin, A. L. and Huxley, A. F. (1952). A quantitative description of membrane current and its application to conduction and excitation in nerve. *J. Physiol.*, 117:500–544.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences, USA*, 79:2554–2558.

Hubel, D. H. and Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *J. Physiol.*, 160:106–154.

Hubel, D. H. and Wiesel, T. N. (1977). Ferrier lecture. functional architecture of macaque monkey visual cortex. *Proc R Soc Lond B Biol Sci*, 198(1130):1–59.

Hughes, S. W. and Crunelli, V. (2005). Thalamic mechanisms of eeg alpha rythms and their pathological implications. *Neuroscientist*, 11(4):357–372.

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

Johansson, C. and Lansner, A. (2006a). Attractor memory with self-organizing input. In *LNCS*, volume 3853, pages 265–280. Springer Verlag.

Johansson, C. and Lansner, A. (2006b). A hierarchical brain-inspired computing system. In *International Symposium on Nonlinear Theory and its Applications (NOLTA '06)*, Bologna, Italy.

Johansson, C., Sandberg, A., and Lansner, A. (2001). A capacity study of a bayesian neural network with hypercolumns. Tech. Rep. TRITA-NA-P0120, Dept. of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden.

Jolivet, R., Lewis, T. J., and Gerstner, W. (2004). Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accuracy. *J. Neurophysiol.*, 92:959–976.

Keat, J., Reinagel, P., Reid, R. C., and Meister, M. (2001). Predicting every spike: a model for the responses of visual neurons. *Neuron*, 30:803–817.

Kennedy, H. (2005). The DAISY Project. `http://daisy.ini.unizh.ch`.

Kumar, A., Schrader, S., Aertsen, A., and Rotter, S. (2007). The high-conductance state of cortical networks. *Neural Computation*. In press.

Lansner, A., Fransén, E., and Sandberg, A. (2003). Cell assembly dynamics in detailed and abstract attractor models of cortical associative memory. *Theory Biosci*, 122:19–36.

Latham, P. E., Richmond, B. J., Nelson, O. G., and Nirenberg, S. (2000). Intrinsic dynamics in neuronal networks. i. theory. *J. Neuophysiol.*, 83:808–827.

MacGregor, R. J. and Oliver, R. M. (1974). A model for repetitive firing in neurons. *Kybernetik*, 16:53–64.

Markram, H. and Peck, C. (2004). The Blue Brain Project. `http://bluebrainproject.epfl.ch`.

Marr, D. (1969). A theory of cerebellar cortex. *J. Physiol.*, 202:437–470.

Marr, D. (1982). *Vision*. Freeman, New York, USA.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bull. Maths. and Biophysics*, 5:115.

Mehring, C., Hehl, U., Kubo, M., Diesmann, M., and Aertsen, A. (2003). Activity dynamics and propagation of synchronous spiking in locally connected random networks. *Biol. Cybern.*, 88(5):395–408.

Meier, K. (2005). The FACETS Project. `http://facets.kip.uni-heidelberg.de`.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97.

Mirsky, J. S., Nadkarni, P. M., Healy, M. D., Miller, P. L., and Shepherd, G. M. (1998). Database tools for integrating and searching membrane property data correlated with neuronal morphology. *J. Neurosci. Methods*, 82(1):105–121.

Morrison, A., Aertsen, A., and Diesmann, M. (2007a). Spike-timing dependent plasticity in balanced random networks. *Neural Computation*. In press.

Morrison, A., Mehring, C., Geisel, T., Aertsen, A., and Diesmann, M. (2005). Advancing the boundaries of high-connectivity network simulation with distributed computing. *Neural Computation*, 17:1776–1801.

Morrison, A., Straube, S., Plesser, H. E., and Diesmann, M. (2007b). Exact subthreshold integration with continuous spike times in discrete-time neural network simulations. *Neural Computation*, 19(1):47–79.

Mountcastle, V. B. (1978). An organizing principle for cerebral function: The unit module and the distributed system. In Edelman, G. M. and Mountcastle, V. B., editors, *The mindful brain*. The MIT Press, Cambridge, Massachusetts.

Palm, G. (1982). *Neural assemblies. An alternative approach to artificial intelligence*. Springer.

Paninski, L., Pillow, J. W., and Simoncelli, E. P. (2004). Maximum likelihood estimation of a stochastic integrate-and-fire neural encoding model. *Neural Computation*, 16:2533–2561.

Papert, S. (1960). Redundancy and linear logical nets. In *Bionics symposium. Dayton, OH*, Washington D.C., USA. Office of Technical Services, U. S. Department of Commerce.

Pesaran, B., Pezaris, J. S., Sahani, M., Mitra, P. P., and Andersen, R. A. (2002). Temporal structure in neuronal activity during working memory in macaque parietal cortex. *Nat. Neurosci.*, 5(8):805–11.

Peters, A. and Sethares, C. (1991). Organization of pyramidal neurons in area 17 of monkey visual cortex. *J Comp Neurol*, 306(1):1–23.

Pfurtscheller, G., Stancak, Jr, A., and Neuper, C. (1996). Event-related synchronization (ers) in the alpha band–an electrophysiological correlate of cortical idling: a review. *Int J Psychophysiol*, 24(1-2):39–46.

Plato (380 BCE). Book II: Socrates – Glaucon. In *V: Republic I*, Loeb Classical Library, page 167. Harvard University Press, Cambridge, MA, USA.

Plato (387–347 BCE). *I–XII*. Loeb Classical Library. Harvard University Press, Cambridge, MA, USA.

Plesser, H. E. and Austvoll, K. (2009). Specification and generation of structured neuronal network models with the NEST topology module. In preparation.

Queisser, G., Xylouris, K., Kolozis, E., Otto, C., Draguhn, A., Bading, H., and Wittum, G. (2008). Detailed 3d-models of cells and their functional units. In *Frontiers in Computational Neuroscience. Conference Abstract: Bernstein Symposium 2008.*

Rall, W. (1959). Branching dendritic trees and motoneuron membrane resistivity. *Exp. Neurol.*, 1:491–527.

Rao, A. R., Cecchi, G. A., Peck, C. C., and Kozloski, J. R. (2008). Unsupervised segmentation with dynamical units. *IEEE Trans. Neural Networks*, 19(1):168–182.

Richardson, M. J., Brunel, N., and Hakim, V. (2003). From subthreshold to firing-rate resonance. *J. Neurophysiol.*, 89:2538–2554.

Robbins, K. A., Grinshpan, I., Allen, K., and Senseman, D. M. (2004). Synchronized views for exploring populations of neurons. In Erbacher, R. F., Chen, P. C., Roberts, J. C., Gröhn, M. T., and Börner, K., editors, *Papers selected from Visualization and Data Analysis*, volume 5295 of *Proc. SPIE*, pages 235–245.

Rudolph, M. and Destexhe, A. (2006). Analytical integrate-and-fire neuron models with conductance-based dynamics for event-driven simulation strategies. *Neural computation*, 18(9):2146–2210.

Sandberg, A., Lansner, A., Petersson, K. M., and Ekeberg, Ö. (2002). Bayesian attractor networks with incremental learning. *Network: Computation in neural systems*, 13:179–194.

Sandberg, A., Tegner, J., and Lansner, A. (2003). A working memory model based on fast hebbian learning. *Network*, 14(4):789–802.

Schemmel, J., Fieres, J., and Meier, K. (2008). Wafer-scale integration of analog neural networks. In *Neural Networks, 2008. IJCNN 2008*, pages 431–438.

Schutter, E. D. (1998). Dendritic voltage and calcium-gated channels amplify the variability of postsynaptic responses in a purkinje cell model. *Journal of Neurophysiology*, 80:504–519.

Sharp, A. A., O'Neil, M. B., Abbott, L. F., and Marder, E. (1993). Dynamic clamp: Computer-generated conductances in real neurons. *J. Neurophysiol.*, 69(3):992–995.

Stallman, R. M. (1981). Emacs the extensible, customizable, self-documenting display editor. AI Memo 519a, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Steriade, M., Amzica, F., and Contreras, D. (1996). Synchronization of fast (30-40 hz) spontaneous cortical rhythms during brain activation. *J Neurosci*, 16(1):392–417.

Strey, A. (1997). EpsiloNN - a specification language for the efficient parallel simulation of neural networks. In *IWANN '97: Proceedings of the International Work-Conference on Artificial and Natural Neural Networks*, pages 714–722, London, UK. Springer-Verlag.

Tetzlaff, T., Morrison, A., Geisel, T., and Diesmann, M. (2004). Consequences of realistic network size on the stability of embedded synfire chains. *Neurocomputing*, 58–60:117–121.

Thomson, A. M., West, D. C., Wang, Y., and Bannister, A. P. (2002). Synaptic connections and small circuits involving excitatory and inhibitory neurons in layer 2–5 of adult rat and cat neocortex: Triple intracellular recordings and biocytin labelling in vitro. *Cerebral Cortex*, 12:939–953.

Traub, R. D. (1979). Neocortical pyramidal cells: A model with dendritic calcium conductance reproduces repetitive firing and epileptic behavior. *Brain Res.*, 173:243–257.

Treves, A. and Rolls, E. T. (1994). Computational analysis of the role of the hippocampus in memory. *Hippocampus*, 4(3):374–91.

Tuckwell, H. C. (1988). *Linear Cable Theory and Dendritic Structure*, volume 1 of *Introduction to Theoretical Neurobiology*. Cambridge University Press, Cambridge.

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

Willshaw, D. and Longuet-Higgins, H. (1970). Associative memory models. In Meltzer, B. and Michie, O., editors, *Machine Learning*, volume 5. Edinburgh University Press, Edinburgh, Scotland.

Wittum, G. (2006). Personal communication.

Wittum, G. (2007). μG. `http://sit.iwr.uni-heidelberg.de/~ug/`.