CN510: Principles and Methods of Cognitive and Neural Modeling

KInNeSS Intrinsics. Descriptive Languages. GPUs for Neural Modeling

Lecture 26

Instructor: Anatoli Gorchetchnikov <anatoli@bu.edu>
Teaching Fellow: Rob Law <nosimpler@gmail.com>
The Structure

Standard user interface
- allows new users to learn the software package faster
- simplifies the creation and use of new environments and modeling frameworks

KDE Integrated NeuroSimulation Software (KInNeSS) is designed to implement this framework

CN 510 Lecture 26
Brief Introduction to KParts and KPlugins

KPart is an independent shared library that includes

- Some functionality
- Components of the user interface to control this functionality
- Mechanism to load and unload it on demand from the application

Only one KPart can be active at any given time per application

KPart::Plugin is similar but

- It is loaded and unloaded by KPart
- Multiple plugins can be active at the same time
- Each plugin can be utilized by multiple parts
KInNeSS Structure

Only one environment is necessary at any given time, so KPart is a perfect base for it.

Network editors and computational engines can be implemented as plugins, this way single editor can serve multiple engines and vice versa.

Additional plugins might include data analysis and visualization tools.
Network Editor Plugin

Each equation is combined from pieces according to the structure defined by the user.
The network editor plugin allows hierarchical access to all levels of details in the network.

Users can:

– Create populations
– Clone existing populations
– Fill:
  • Populations with cells
  • Cells with compartments
  • Compartments with channels
  • Channels with gating variables

It also provides the default sets of neurons and channels, each of which can be expanded by the user.
Examples of Old Network Editor Windows

Network is represented as a tree
There is no graphic representation
Channel Name: Na (original HH)  
Equilibrium Potential: 55.0 mV
Number of Gating Variables: 2  
Maximal Conductance: 120.0 mS/cm²

Dependency: voltage

Forward (alpha): 
\[
\frac{0.100 \times (25.0 - V_m)}{e^{\frac{-25.0 - V_m}{10.0}} - 1}
\]

Backward (beta): 
\[
e^{\frac{0.0 - V_m}{18.0}}
\]
Model Visual Editor Resource (MoVER) Plugin

Three levels:

- The bottom level is the same tree-like editing, but it is limited to cells only
- The middle level allows to set geometry of populations made of these cells
- The top level uses editable visual representation of the circuitry
Interactions Between Editing and Simulating

Network editor plugin stores network descriptions as a single XML (Extensible Markup Language) structure.

KInNeSS plugins support XML for both import (simulator and editor) and export (editor only) of neural network architectures.

Technically, a user can edit XML files directly, bypassing the editor plugin.
Why Do We Lack Standards?

None of the existing tools provides human comprehensible model descriptions that can be included in publications. As long as matching the experimental data is a sole purpose of modeling there is no need for standard.

We need a drive to compare models against each other and a way to automatically generate human-readable model description for publication.
NeuroML 1

First attempt to create XML based language to describe compartmental neurons, their morphology, and ionic channels. Later extended by NetworkML for network descriptions. Initially enthusiastically supported by many simulator developers and leading modelers. Gradual decline in interest:

– Complete development is a long process, people do not like to wait
– Some design decisions that cannot make everybody happy

<variable name="voltage" value="10mV"/>
<variable name="voltage">
  <value>
    10
    <unit>mV</unit>
  </value>
</variable>
NeuroML 1

Too much focus on detailed models of neurons, not enough on networks of simplified neurons

Usage:

- PyNN has NeuroML parser to convert models into NEURON and GENESIS 2
- NEURON and GENESIS 2 support NeuroML
- Lots of other tools listed on NeuroML web site: some are simulators, others are database tools, visualization tools and translation tools
- KInNeSS used a dialect of NeuroML natively

Main issue: NeuroML 1 does not lead to a deterministic model’s specification because only parameters are described, not the mathematical foundations
Why Do We Need NineML?

Erik de Shutter went to NeuroML workshop trying to push them towards the goals that he thought critical and failed.

So he enlisted INCF MSM initiative support for developing an alternative language.

NineML is intended to complement NeuroML with network support.
NineML Team

Not in the picture: Mike Hines (NEURON), Eilif Muller (Blue Brain/NEST), Sean Hill (Blue Brain), Mikael Djurfeldt (SPLIT/MUSIC)
Model representation:
- Integrate-and-fire neurons
- Synapses
- Populations
- Connectivity

NineML

Poisson

Isyn₁

\[ \int \theta \rightarrow \text{spike} \]

\[ \text{spike} \rightarrow \text{G} = \text{G} + \text{G} \rightarrow \text{Isyn} \]

NEURON

nest::simulated()
What Makes NineML Different?

**User Layer**: Biological concepts – such as leaky integrate-and-fire model, exponentially decaying synapse, neuronal populations and projections – and their parametrizations

**Abstraction Layer**: Each concept in the user layer has a standardized mathematical representation in the abstraction layer
Is That Enough?

No!

**Below Abstraction Layer** should be an **Implementation Layer**: details of the implementation like numerical integration methods, threshold crossings, etc.

**Above User Layer** should be a **Simulation Layer**: description of external inputs, simulation length, etc.

These were intentionally left out of scope of NineML in hope to include them later, when the core is developed.
Summary of the User Layer

Is intended to be primarily machine-readable and uses XML syntax

Is designed with a focus on ease of parsing, verification, and automatic model construction

Serves to instantiate objects defined in the abstraction layer and listed in the ontology

Provides the syntax for specifying the model and parameters used to instantiate the key elements of a spiking neuron network: cells, synapses, inputs, grouping, geometry, and connectivity
Summary of the Abstraction Layer

Is intended to be machine readable, currently there are 3 proposed implementations: XML, python, and custom script.

Is designed for automatic parsing and model construction, python and C code generators are in development for modelers that do not use simulation software.

Serves to unambiguously define mathematics behind each biological construct in the model and link these constructs together.
Why Did NineML Task Force Fail?

Oversight Committee failed because big shots are busy and don’t care about details: they never met after the inauguration meeting.

Task Force failed because we all have other jobs.

Essentially, Task Force took the job of the Oversight Committee to lay down the foundations of the language but there was nobody there to do the implementation.

Take home messages:
- A dozen of smart guys in the room does not guarantee success, there should be at least one guy that does the implementation.
- The implementation guy must be well paid.
Why NineML Was Not a Total Waste?

The idea that we need Abstraction Layer resulted in addition of LEMS (Low Entropy Model Specification) to NeuroML 2

LEMS also includes Simulation Layer

Several other concepts that we discussed in NineML meetings migrated into NeuroML 2 and got implemented (NeuroML guys are paid for their effort 😊)

Leading simulator developers got better understanding of the issues related to model specifications

Yann Le Franc: NineML Task Force definitely succeeded in making NeuroML better
Analysis Tools

Local field potentials and current source densities can be computed off-line after the simulation.

Introduction of spatial geometry for the populations will make these tools even more realistic.
Computational Engine: KBrain and INDRA

KBrain is based on KPart::Plugin and provides user interface for setting up and running simulations with INDRA.

INDRA is Iterative Nonspecific Distributed Runtime Algorithm based upon
- Polymorphism
- Use of functional objects
- Standard Template Library (STL) classes
- Multithreading

It is capable of numerical integration of event coupled large systems of non-homogeneous differential equations

\[ C_m \frac{dV}{dt} = \sum I \]
Branching

$V_B$ is the potential at the branching point, all three currents in the figure are calculated separately.

When two compartments are connected by a non-branching intersection ($V_3$ and $V_4$) the specific axial resistance is the same for both compartments and only one current needs to be calculated.
Set of Currents (Gates) in KInNeSS

Multiple currents implemented in KInNeSS, all fit into one format: conductance based gating variable times voltage based state variable.

<table>
<thead>
<tr>
<th>Gating variable</th>
<th>Specific use in KInNeSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Injection</td>
<td>Current injection</td>
</tr>
<tr>
<td>Input</td>
<td>Voltage input that drives compartment towards a certain voltage. In the case of infinite conductance it turns into a perfect voltage clamp.</td>
</tr>
<tr>
<td>Gap</td>
<td>Gap junctions</td>
</tr>
<tr>
<td>Voltage</td>
<td>Classical voltage gates: Exponential, Sigmoid, Linoid, Generalized version that incorporates all three of the above: Parameterized And two gates for ( g_{\infty} / \tau ) form of representation Simple Tau Thalamic Reticular Tau</td>
</tr>
<tr>
<td>Ligand</td>
<td>Synaptic Currents</td>
</tr>
<tr>
<td>Voltage block</td>
<td>Voltage dependent blocking of a channel (e.g. Mg(^{2+}) for NMDA)</td>
</tr>
<tr>
<td>Modulation</td>
<td>Neuromodulatory effect on a current</td>
</tr>
<tr>
<td>AHP/ADP</td>
<td>After-hyperpolarization or after-depolarization current</td>
</tr>
<tr>
<td>Reduced</td>
<td>Pseudo current with quadratic integrate-and-fire to replace a set of Hodgkin-Huxley currents</td>
</tr>
</tbody>
</table>
**STDP Rule**

\( x_{post} \) to accommodate the variable spike duration, and to put realistic bounds on \( \lim_{t \to \infty} w \) for each training episode.

Passive weight decay to baseline to prevent unbounded growth.

Four types of gating: presynaptic, postsynaptic, dual OR, and dual AND

\[
\dot{w} = \lambda \left( x_{pre} x_{post} (\bar{w} - \bar{w}) + w_0 - w \right) f_G \left( x_{pre}, x_{post} \right)
\]

Minimal weight is set to 0, maximal weight, baseline weight and the duration of \( 1/A \) is set in the interface.

CN 510 Lecture 26
Multithreading

Using this approach we can add as many computational threads as our hardware permits. Furthermore, these threads can utilize other than CPU computational hardware.

We utilize CUDA GPUs to provide extra computational power.
Single Instruction Multiple Data (SIMD)

Identical code instruction is executed in parallel on many data elements.

We have large populations of neurons, each described by an equation(s), the only difference for neurons in the population is in activity and input levels.

How much better can it get?

GPU is a modern SIMD device.
Why GPUs Are Good for Neural Modeling?

GPU is especially well-suited for data-parallel computations
– the same program is executed on many data elements in parallel
– with high ratio of arithmetic operations to memory operations

Numerical integration of spiking neural models is computationally intensive
Communication between populations is based on spikes, the amount of data transfer is relatively small
Some texture filtering operations when implemented in hardware can speed-up calculations of input in many-to-many projections

Why CUDA?
CUDA Runtime Structure

One neuron – one thread

Threads are grouped into blocks

Blocks are grouped into grids

One population – one grid

Code is written for a thread

Threads have unique IDs, you can know which neuron you are in

You can also access your neighborhood (beware of the cost)
CUDA Runtime Structure: SIMT

Maximum number of threads in a block depends on the compute capability (1024 on Fermi)

GPU multiprocessor creates, manages, schedules and executes threads in warps of 32

Warp executes one common instruction at a time
  - Threads are allowed to branch, but each branch is executed serially
CUDA Runtime Structure: Number of Threads

Number of simultaneous threads depends on the device:

- For Fermi it is 512

For maximum throughput many more threads are needed

- Many threads will be waiting for memory queries or register values
- Thread scheduler can hide latencies efficiently only if there are enough threads waiting for execution
Choosing Your Sizes

How big of a population is good for a GPU?
Number of blocks shall be more than a number of multiprocessors (512 on Fermi)
   – All MPs have a block to execute
Ideally, number of blocks shall be at least double of the number of MPs
Threads per block should be multiple of warp size (32)
   – No under populated warps
   – Coalesced memory access
So populations of ~30k neurons at least…
But… Even with smaller populations beats the CPU…
Experiment!
Coalesced Memory Access

Memory access is optimized for coalesced access where threads read from / write to successive memory locations.

Exact alignment rules and performance issues depend on the computing capability. Serious performance penalties if violated.
Why GPUs Are Bad for Neural Modeling?

Because the same program is executed for each data element, there is a limitation on sophisticated flow control

– Conditionals become a waste of processing power, because each branch is executed on a subset of threads while the remaining threads wait, the block finishes only after both branches are done

– Conditionals in integrate-and-fire dynamics and in spike event production – not too bad, rarely more often then 100 times per simulated second per neuron

– Conditionals in synaptic inputs on every arriving spike – worse, can have thousand of interruptions per compartment

– Both are usually short: bump a state variable and continue
Why GPUs Are Bad for Neural Modeling?

There are no variable size data types on the CUDA level

- To register a spike event in the brute force solution we need to allocate memory for all possible events, while the use of this memory will be very sparse and almost never coalesced

Neuronal communication almost always go outside a block, and often goes off-chip

- The problem becomes memory-access-bound rather than computation-bound
- No efficient and scalable spike delivery algorithms, especially on the GPU side
Master node manages interactions with the user, slaves’ synchronization, model placement across network nodes, and data collection for visualization in off-line mode.

All nodes manage their own thread structure and peer-to-peer network communication.
Main Computational Cycle

- **Input Queue**: input from environment
- **Output Queue**: host to device
- **Device Computation**: device to host
- **CPU Computation**: insert events in the output queue
- **Select and Deliver**: proper events to receivers
- **Push to Local**: events
- **Push to Network**: computed values of all needed variables
- **Event Output**: outgoing events
- **Data Display**: Data Output
- **Fill from Network**: insert events
- **Input Update**: computed values of all needed variables
- **Post Barrier**: Non-CPU computational devices
- **Compute Semicycle**: Compute Barrier
- **Post Semicycle**
Communication Model: Nodecentric View
Summary of KInNeSS Design

Core top-level standard KDE interface allows loading various project environments, which can be written separately from KInNeSS by third parties.

The project environment contains everything that simulates the environmental and behavioral component of the simulation.

The modeling interface is implemented as a set of plugins, so that the same modeling approach can be used in different project environments and different approaches can be used with the same environment.

Platform and interface independent computational engine to do calculations (INDRA).

Note that other modeling plugins can use their own engines.
Summary of the Course

There is no perfect model that answers all questions.

There is no ground truth in either modeling or experimental papers, always be skeptical and look for small font.

There is no mature underlying theory of brain function.

There is plenty of small well-studied circuits that have useful properties and can be brought in your work whenever necessary.

There is plenty of unsolved or partially solved problems for you to tackle.
Summary of the Course

Computational neuroscience tasks can be of different levels:

– What behavior my model has to produce and what experimental data to match? (*psychology*, *physiology*)

– What function do I need to achieve for a specific network? (*physiology*, *AI?*)

– Which equation I need to use for a specific neuron? (*mathematics*, *physics*)

– How do I need to solve the equation of this type? (*computer science*, *numerical analysis*)

– How do I implement this solver in an efficient way? (*computer science*, *computer engineering*)

– How do I present my model and results in a reproducible way? (*informatics*, *neuroinformatics*)
Always Doubt What You See!