Tadpole on FPGA
mapping floating-point equations
into integers using code-generation

Mike Hull

Computer Laboratory
University of Cambridge

19th Dec. 2013
The tadpole

- Ph.D in modelling with the experimental lab of Alan Roberts (Bristol)
- Brief touch -> sustained swimming
- “How does a tadpole wire up a neuronal network within 48 hours that can generate behaviour?”
“While not widely regarded as such, it has arguably become the best-understood spinal cord locomotor network in terms of network organization and functional properties.”

1Parker, J Physiol 2009
Why the tadpole?

- ‘Simple’, well-characterised nervous system:
  - direct *in situ* electrophysiological recordings
  - constrained anatomical layout
  - behavioural studies

- Computational model of swimming:
  - Hodgkin-Huxley type biophysical models ($\approx 1500$ neurons)
  - Sodium, two potassium and calcium voltage-gated channels
  - Electrical coupling via gap junctions
  - $\approx 100,000$ AMPA, NMDA and glycinergic synaptic connections (growth model)
Equations and units

\[ i_{ca} = \frac{A \cdot P_{Ca} \cdot 2 \nu \cdot F}{\left[ Ca^{2+}\right]_i - \left[ Ca^{2+}\right]_o \cdot e^{-\nu}} \]

- **Units:**
  - \( i_{ca} \): pA
  - \( A \): \( \mu m^2 \) cm/s
  - \( P_{Ca} \): mmol/liter
  - \( 2 \nu \cdot F \): m²
  - \( \left[ Ca^{2+}\right]_i \): mmol/liter
  - \( \left[ Ca^{2+}\right]_o \): mmol/liter
  - \( e^{-\nu} \): dimensionless
Mapping to Integers

- Existing C-model - improve performance by running on an FPGA?
- Sequential + floating point (software emulation) → poor performance
- Mapping to integers:
  - Floating-Point hardware expensive in chip-area, power, latency
  - Take advantage of BlueVec on FPGA
  - Other platforms without a floating point unit (SpinNaker)
Mapping to Integers

- Time consuming to manually convert complex equations
- Use a high-level description and map this to C++ simulation code
- [Bonus: take care of the units problem?]
- Used an experimental NineML-based library (‘neurounits’).
Overview of mapping

- **Component descriptions**
  - Neuron-Type1
  - Neuron-Type2
  - Gap Junction
  - Sensory input

- **Connected using a PyNN-like interface**
  - `mysimulation.py`

- **Generate C++ (floating or fixed point)**
  - `simulation.cpp`

- **Execute on PC**
- **Execute on FPGA**

*Hardware description of NIOS & BlueVec*
Example neurounits

\begin{align*}
define\ _{component} \ g\_juncti on \\
\{ \\
\quad \leftrightarrow \text{PARAMETER} \ g\_g j: (S) \\
\quad \leftrightarrow \text{INPUT} \ V1: (V), \ V2: (V) \\
\quad \quad i1 = g\_g j \ast (V2 - V1) \\
\quad \quad i2 = -i1 \\
\}
\end{align*}
Example neurounits

```plaintext
define_component gap_junction
{
  \( \Leftrightarrow \) PARAMETER \( g\_gj : (S) \)
  \( \Leftrightarrow \) INPUT \( V1 : (V) , V2 : (V) \)
  \( i1 = g\_gj \ast (V2 - V1) \)
  \( i2 = -i1 \)
}
```

```plaintext
define_component passive_cell
{
  \( \Leftrightarrow \) TIME \( t \)
  \( \Leftrightarrow \) PARAMETER \( g\_leak \)

  \( C = 10\text{pF} \)
  \( V' = \frac{(i\_leak + i\_inj + i\_syn)}{C} \)

  \# Leak Channel:
  \( i\_leak = \{2\text{nS}\} \ast \{-64\text{mV} - V\} \)

  \# Injected Current:
  \( i\_inj = [30\text{pA}] \) if \( [50\text{ms} < t < 100\text{ms}] \) else \([0\text{pA}]\)

  \# Synapse:
  \( i\_syn = \{300\text{pS}\} \ast \{0\text{mV} - V\} \ast (B-A) \)
  \( A' = -A / \{1.5\text{ms}\} \)
  \( B' = -B / \{5\text{ms}\} \)
  on ampa_input {
    \( A = A + 1 \)
    \( B = B + 1 \)
  }
}
```
Example AST

\[ \frac{dV}{dt} = \frac{(i_{\text{leak}} + i_{\text{inj}} + i_{\text{syn}})}{C} \]

\[ i_{\text{syn}} = 2nS \times (0mV - V) \times (B-A) \]

\[ \frac{dA}{dt} = \frac{-A}{1.5ms} \]

\[ \frac{dB}{dt} = \frac{-B}{1.5ms} \]

**Time Derivatives:**

\[ \frac{dV}{dt} \]

\[ C = 10\text{pF} \]

\[ \frac{dA}{dt} \]

\[ \frac{dB}{dt} \]

**Assigned Values:**

\[ i_{\text{leak}} \]

\[ i_{\text{inj}} \]

\[ i_{\text{syn}} \]

\[ \text{...} \]

\[ \star \]

\[ 
\]

\[ 
\]

\[ 2\text{ns} \]

\[ 
\]

\[ 
\]

\[ 0 \text{mV} \]

\[ 
\]

\[ 
\]

\[ 0 \]

\[ 0 \]

**State Variables:**

\[ V \]

\[ A \]

\[ B \]

**Event handling:**

\[ \text{on}_\text{ampa} \]

\[ A = A + 1 \]

\[ B = B + 1 \]
Encoding the fixed-point format of the nodes

- In N bits, we can store integers from:
  \[-2^{(N-1)} < \text{value_int} < 2^{(N-1)} - 1\]

- Based on the range of each node in the AST, choose a suitable upscale factor, \( U \):
  \[
  \text{value_float} = \frac{\text{value_int}}{2^{N-1}} \times 2^U
  \]

<table>
<thead>
<tr>
<th>U</th>
<th>(2^U)</th>
<th>Min-Value</th>
<th>Max-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>3.9e-3</td>
<td>3.9e-3</td>
<td>-3.9e-3</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-4</td>
<td>0.0625</td>
<td>-0.0625</td>
<td>0.0625</td>
</tr>
<tr>
<td>-3</td>
<td>0.125</td>
<td>-0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>-2</td>
<td>0.25</td>
<td>-0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>-1</td>
<td>0.5</td>
<td>-0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>-4.0</td>
<td>4.0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>-256</td>
<td>256</td>
</tr>
</tbody>
</table>
Encoding the fixed-point format of the nodes

- In N bits, we can store integers from:
  
  $$-2^{(N-1)} < \text{value}_\text{int} < 2^{(N-1)} - 1$$

- Based on the range of each node in the AST, choose a suitable upscale factor, $U$:
  
  $$\text{value}_\text{float} = \frac{\text{value}_\text{int}}{2^{N-1}} \times 2^U$$

<table>
<thead>
<tr>
<th>$U$</th>
<th>$2^U$</th>
<th>Min-Value</th>
<th>Max-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8</td>
<td>3.9e-3</td>
<td>3.9e-3</td>
<td>-3.9e-3</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-4</td>
<td>0.0625</td>
<td>-0.0625</td>
<td>0.0625</td>
</tr>
<tr>
<td>-3</td>
<td>0.125</td>
<td>-0.125</td>
<td>0.125</td>
</tr>
<tr>
<td>-2</td>
<td>0.25</td>
<td>-0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>-1</td>
<td>0.5</td>
<td>-0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>-2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>-4.0</td>
<td>4.0</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>256</td>
<td>-256</td>
<td>256</td>
</tr>
</tbody>
</table>

- For example:
  
  $$-100 \text{mV} < V < 50 \text{mV}$$

  $$-0.1 < V < 0.05$$

  $\rightarrow U : -3$
Example output C++ code

\[
i_{\text{leak}} = 2.5\text{nS} \times \begin{pmatrix} -50\text{mV} & V \\ U:-4 & U:-3 \end{pmatrix}
\]
Example output C++ code

\[ i_{\text{leak}} = 2.5nS \times \left( \frac{-50\text{mV}}{U:-28} - \frac{V}{U:-4} \right) \]
\[ \frac{V}{U:-3} \]
\[ U:-31 \]

```cpp
for (int i = 0; i < NrnPopData::size; i++) {
    // ...
    d.i_leak[i] = (ScalarOp<-31>::mul(
        ScalarType<-28>(5629500), // [Constant 2.5e-9]
        ScalarOp<-3>::sub(
            ScalarType<-4>(-6710886), // [Constant -0.05]
            d.V[i]
        )
    ) ;
    // ...
}
```
Exponentials

- Exponentials are common in biophysical models.
- Implemented as linear-interpolated lookup tables.

To maintain accuracy, the value of $U$ varies to encode different values of $\exp(x)$. 

![Graph showing exponential function]

- $\alpha_{na,h}$
- $\beta_{na,h}$
Example simulation construction

```python
network = Network()
dIN_comp = neurotrans-userunits.ComponentLibrary.instantiate_component('dIN')

dINs = network.create_population(
    name='dINs',
    component=dIN_comp,
    size=30,
    parameters={'nmda_multiplier': 1.0, 'ampa_multiplier': 1.0, 'inj_current': '20 pA'})

network.create_eventportconnector(
    name='dIN_dIN_NMDA',
    src_population=dINs,
    dst_population=dINs,
    src_port_name='spike',
    dst_port_name='recv_nmda_spike',
    delay='1 ms',
    connector=AllToAllConnector(0.2),
    parameter_map={'weight': '150 pS'})

network.record_traces(dINs, 'V')
results = CBasedEqnWriterFixedNetwork(
    network,
    output_c_filename='simulation1.cpp',
    CPPFLAGS='-DON_NIOS=false -DPC_DEBUG=false -DUSE_BLUEVEC=false',
    step_size=0.1e-3,
    run_until=1.0,
    as_float=False).results
```
Results

- On PC:
  - Comparing traces of float/fixed point simulations for single Hodgkin-Huxley neuron
  - Small 30 neuron single sided-model and full tadpole model give behaviourally similar outputs
- Mapped full tadpole to take advantage of the BlueVec unit on the FPGA
## Results - Tadpole on FPGA

<table>
<thead>
<tr>
<th>Loop-phase</th>
<th>Time without BlueVec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electrical coupling</td>
<td>4.5ms (0.1%)</td>
</tr>
<tr>
<td>Solving state equations</td>
<td>4486.2ms (99.4%)</td>
</tr>
<tr>
<td>Spike delivery</td>
<td>24.3ms (0.5%)</td>
</tr>
<tr>
<td>Loop-phase</td>
<td>Time without BlueVec</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Electrical coupling</td>
<td>4.5ms (0.1%)</td>
</tr>
<tr>
<td>Solving state equations</td>
<td><strong>4486.2ms</strong> (99.4%)</td>
</tr>
<tr>
<td>Spike delivery</td>
<td>24.3ms (0.5%)</td>
</tr>
</tbody>
</table>
Results

- [Tadpole swimming video]
Issues encountered

- Use a bounded optimiser to find range of each node
- Division by zero & vector processing:

\[ \alpha_m(V) = \frac{25 - V}{10 \left(e^{(25-V)/10} - 1\right)} \]
Use a bounded optimiser to find range of each node

Division by zero & vector processing:

\[
\alpha_m(V) = \frac{25 - V}{10 \left( e^{(25-V)/10} - 1 \right)}
\]

\[
= \begin{cases} 
100 e^{(25-V)/10} & \text{if } \left| V - 25 \right| < 1 \times 10^{-5} \\
\frac{25 - V}{10 \left( e^{(25-V)/10} - 1 \right)} & \text{otherwise}
\end{cases}
\]
Issues encountered

- Use a bounded optimiser to find range of each node
- Division by zero & vector processing:

\[ \alpha_m(V) = \frac{25 - V}{10 \left( e^{(25-V)/10} - 1 \right)} \]

\[ = \begin{cases} 
100 e^{(25-V)/10} & \text{if } |\text{fabs}(V - 25)| < 1e-5 \\
\frac{25 - V}{10 \left( e^{(25-V)/10} - 1 \right)} & \text{otherwise}
\end{cases} \]

- Introduce IfThenElse nodes and consider short-circuiting semantics.
Conclusions

- It is possible to simulate a model neuronal network, including complexities such as electrical coupling, calcium channels, NMDA-voltage dependance, in fixed point.
- A minimal, human-readable language can be used to unambiguously specify complex dynamics of components, including inline units.
- Code-generation can produce efficient simulation code.
- Vector processing dramatically improved performance for this model.
Acknowledgements

- Bob Merrison
- Robert Cannon
- Matt Naylor
- Simon Moore & the rest of the group

Thanks for listening - any questions?