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?"*

Why the tadpole?

"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."[1]

[1] Parker, 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 glyinergic synaptic connections (growth model)

Equations and units

$$\underbrace{i_{ca}}_{pA} = \underbrace{A}_{\mu m^2} \underbrace{P_{Ca}}_{cm/s} 2\nu \underbrace{F}_{C/mol} \underbrace{\frac{\left[Ca^{2+}\right]_i - \left[Ca^{2+}\right]_o e^{-\nu}}{1 - e^{-\nu}}}_{mmol/liter} m^2$$

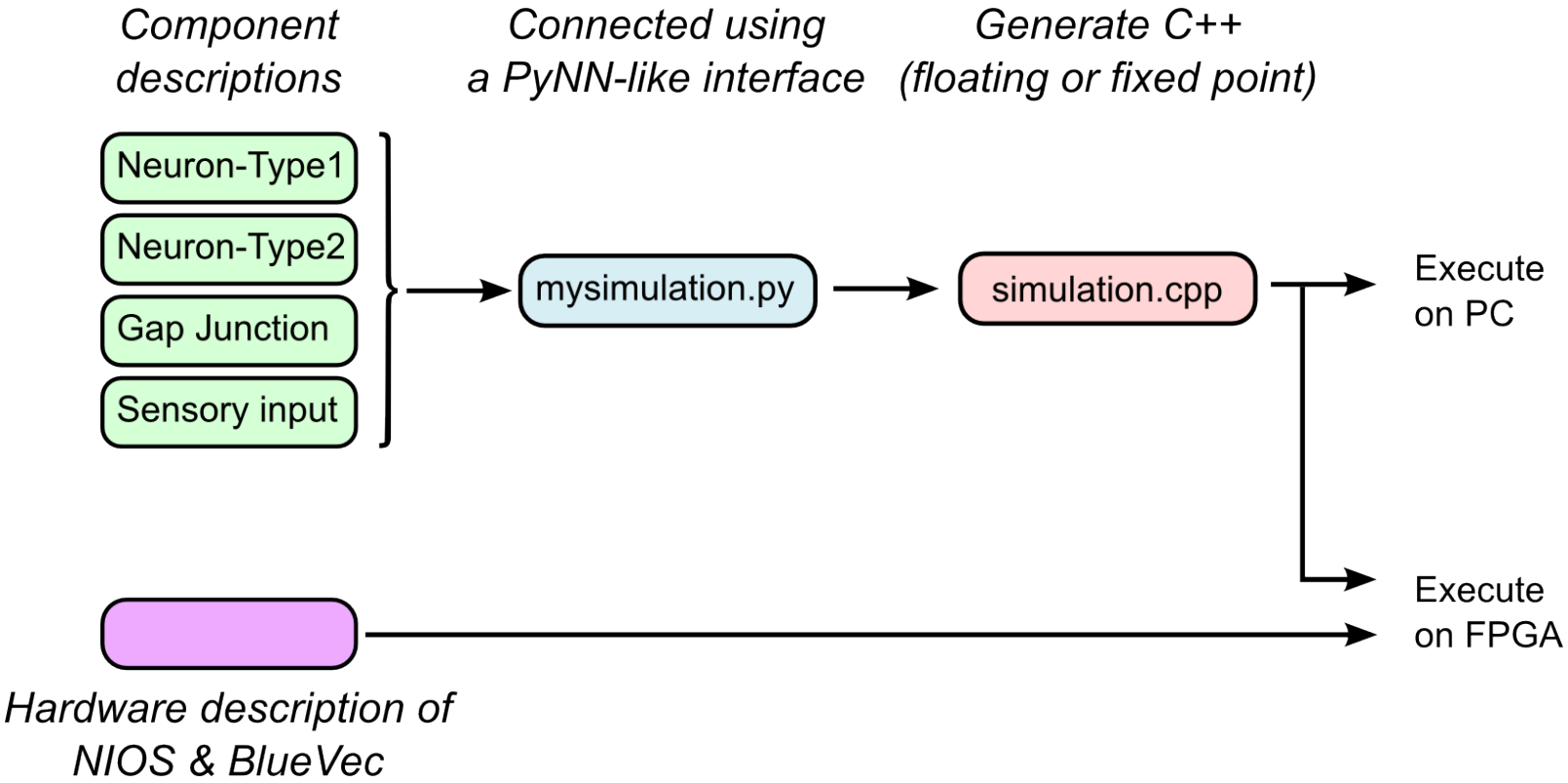
Mapping to Integers

- Existing C-model - improve performance by running on an FPGA?
- Sequential + floating point (software emulation) $\rightarrow$ 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)

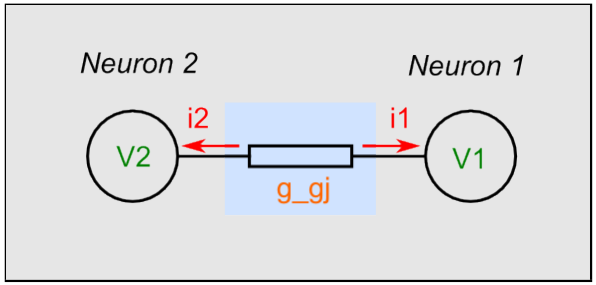
Mapping to Integers

- 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

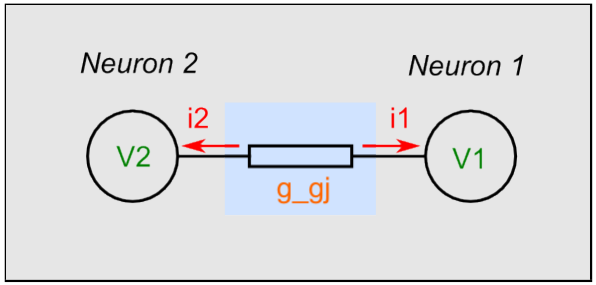


Example neurounits

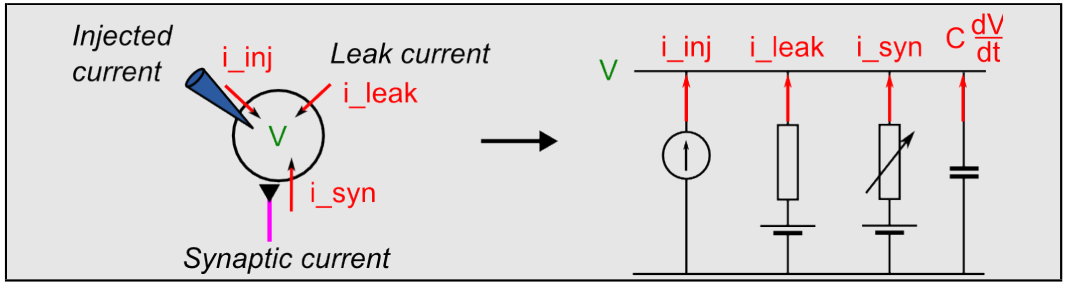



```
define_component gap_junction
{
    <=> PARAMETER g_gj:(S)
    <=> INPUT V1:(V), V2:(V)
    i1 = g_gj * (V2-V1)
    i2 = -i1
}
```

Example neurounits




```
define_component gap_junction
{
    <=> PARAMETER g_gj:(S)
    <=> INPUT V1:(V), V2:(V)
    i1 = g_gj * (V2-V1)
    i2 = -i1
}
```



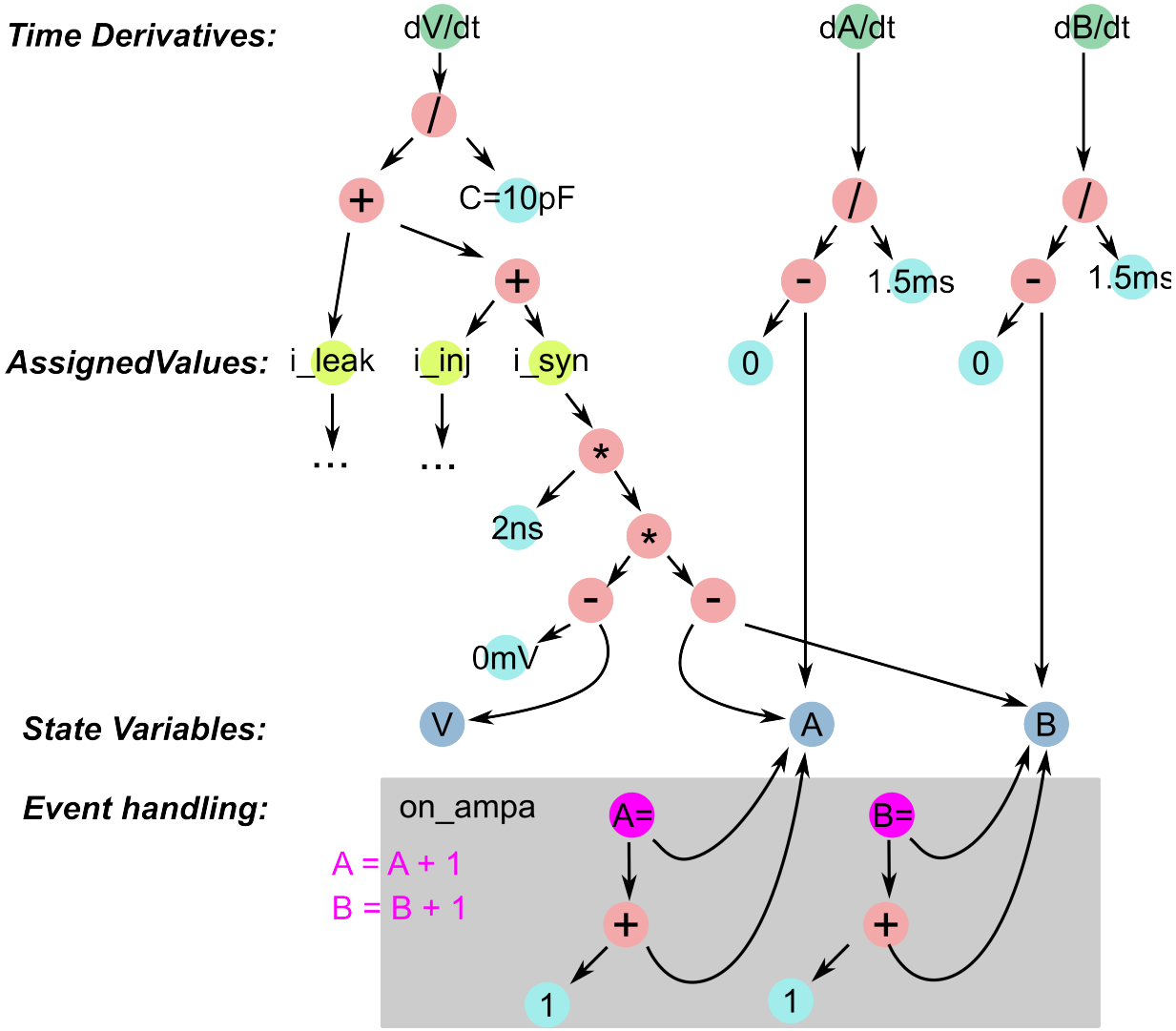

```
define_component passive_cell
{
    <=> TIME t
    <=> PARAMETER g_leak

    C = 10pF
    V' = (i_leak + i_inj + i_syn) / C

    # Leak Channel:
    i_leak = {2nS} * ({-64mV} - V)

    # Injected Current:
    i_inj = [30pA] if [50ms < t < 100ms] else [0pA]

    # Synapse:
    i_syn = {300pS} * ({0mV} - V) * (B-A)
    A' = -A / {1.5ms}
    B' = -B / {5ms}
    on ampa_input {
        A = A + 1
        B = B + 1
    }
}
```

Example AST

dV/dt = (i_leak + i_inj + i_syn) / C

i_syn = 2nS * (0mV - V) * (B-A)

dA/dt = -A / 1.5ms dB/dt = -B /



Encoding the fixed-point format of the nodes

- In N bits, we can store integers from :

$$-2^{(N-1)} < \texttt{value\_int} < 2^{(N-1)} - 1$$

- Based on the range of each node in the AST, choose a suitable upscale factor, U:

$$\texttt{value\_float} = \frac{\texttt{value\_int}}{2^{N-1}} \times 2^{U}$$

U	2^U	Min-Value	Max-Value
-8	3.9e-3	3.9e-3	-3.9e-3
...			
-4	0.0625	-0.0625	0.0625
-3	0.125	-0.125	0.125
-2	0.25	-0.25	0.25
-1	0.5	-0.5	0.5
0	**1**	**-1.0**	**1.0**
1	2	-2.0	2.0
2	4	-4.0	4.0
...			
8	256	-256	256

Encoding the fixed-point format of the nodes

- In N bits, we can store integers from :

$$-2^{(N-1)} < \texttt{value\_int} < 2^{(N-1)} - 1$$

- Based on the range of each node in the AST, choose a suitable upscale factor, U:

$$\texttt{value\_float} = \frac{\texttt{value\_int}}{2^{N-1}} \times 2^U$$

U	2^U	Min-Value	Max-Value
-8	3.9e-3	3.9e-3	-3.9e-3
...			
-4	0.0625	-0.0625	0.0625
-3	0.125	-0.125	0.125
-2	0.25	-0.25	0.25
-1	0.5	-0.5	0.5
0	**1**	**-1.0**	**1.0**
1	2	-2.0	2.0
2	4	-4.0	4.0
...			
8	256	-256	256

- For example:

$$-100mV < V < 50mV$$
$$-0.1 < V < 0.05$$
$$\rightarrow U : -3$$

Example output C++ code

$$i\_leak = \underbrace{\underbrace{2.5nS}_{U:-28} \times \underbrace{\left(\underbrace{-50mV}_{U:-4} - \underbrace{V}_{U:-3} \right)}_{U:-3}}_{U:-31}$$

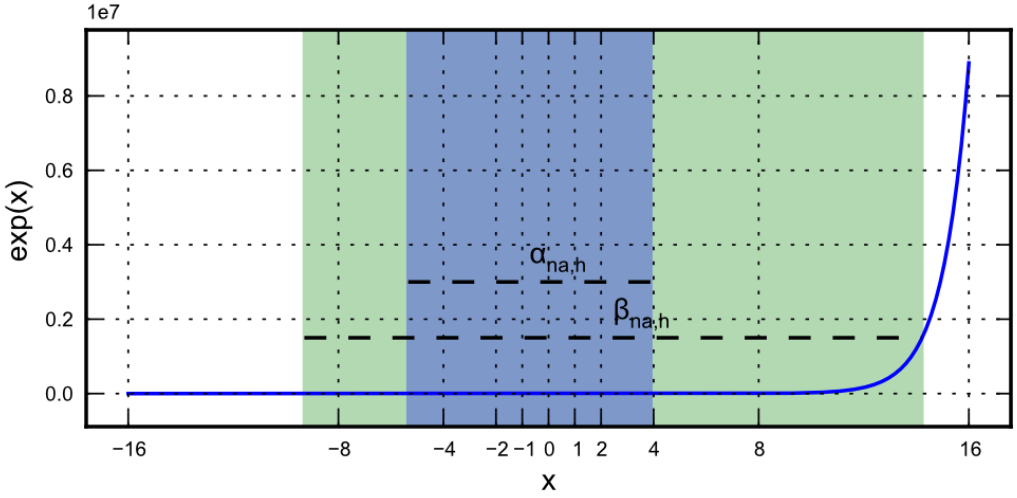
Example output C++ code

$$i\_leak = \underbrace{\underbrace{2.5nS}_{U:-28} \times \underbrace{\left(\underbrace{-50mV}_{U:-4} - \underbrace{V}_{U:-3} \right)}_{U:-3}}_{U:-31}$$

```
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]
                        )
                    )
                ) ;
    // ...
}
```

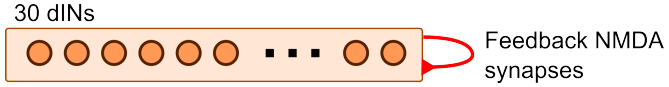
Exponentials

- 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 $exp(x)$.

Example simulation construction




```
network = Network()
dIN_comp = neurounits.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':'20pA'})

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

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

Loop-phase	Time without BlueVec	
Electrical coupling	4.5ms	(0.1%)
Solving state equations	**4486.2ms**	(99.4%)
Spike delivery	24.3ms	(0.5%)

Results - Tadpole on FPGA

Loop-phase	Time without BlueVec		Time with BlueVec	
Electrical coupling	4.5ms	(0.1%)	4.7ms	(15.0%)
Solving state equations	**4486.2ms**	(99.4%)	**4.2ms**	(13.3%)
Spike delivery	24.3ms	(0.5%)	22.4ms	(71.6%)

Results

- [Tadpole swimming video]

Issues encountered

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

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

Issues encountered

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

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

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

Issues encountered

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

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

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

- Introduce `IfThenElse` nodes and consider short-cicuiting 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?