HiveMind:
Providing an efficient programming model for extreme-scale real-time neural network simulation

Steven Marsh
Supervised by: Dr. Simon Moore
The human brain

- 100 billion neurons
- 100 trillion synapses
- Powerful yet power-efficient
- Only just scratching the surface
Current simulators

- Scalability issues beyond a few thousand spiking neurons
  - $O(n^2)$
- Slow performance
  - Nowhere near real-time
  - Doesn’t make full use of hardware
- Different algorithm implementations
- Simulator-specific models
Network definition: NeuroML and LEMS

- Simulator agnostic descriptions
- Provides a standard way to express models
- XML based language
- Compatible with popular simulators
- Tool-chain support
- Active developers
- LEMS – behaviour
- NeuroML – network
Spiking neuron models:

- Neurons can be modeled as differential equations
- Simulation time-step: 1 millisecond

Biological neural network characteristics:

- Firing rate of 10-100Hz
- ~1,000 synaptic inputs per Neuron
- 10% maximum spike activity
- 10,000 Neurons produce 100,000,000 spike messages per second
The Bluehive – a scalable architecture

- Communication-centric
  - Huge bandwidth
  - Very-low latency
- 16 FPGAs
- 1 million Izhikevich neurons
- 1 billion synapses
- Flexible architecture
  - Currently: Custom neuron hardware pipeline
Finding an efficient programming model

• Asking non-computer scientists to try and program our esoteric computer systems won’t work
  ▪ Really bad user-experience and steep learning curve
• Currently we have a lack of real-world and reproducible benchmarks
• Take existing models and massively speed them up
  ▪ Fits seamlessly into existing tools and workflow
• Needs to be cross-platform!
HiveMind – a NeuroML Compiler

- **Objective:** Take existing NeuroML neural network models and compile parallelized C-code
- Components defined in NeuroML have behavioral definition in LEMS.
- Take LEMS descriptions and generate C-Code equivalent
- Convert NeuroML network description and component instantiation into our flat-file format
- Dynamically create neuron-specific yet generic simulators
- Cross-platform, optimised for whatever parallel hardware you have
  - Multi-core, Many-core, Million-core, GPUs, Vector hardware…
Generation of simulation code

• Use LEMS **Dynamics** description to generate neuron-specific C-simulation code

```xml
<ComponentType name="izhikevichCell"
  extends="baseCellMembPot">
  <Parameter name="v0" dimension="voltage"/>
  <Parameter name="a" dimension="none"/>
  <Parameter name="b" dimension="none"/>
  <Parameter name="c" dimension="none"/>
  <Parameter name="d" dimension="none"/>
  <Parameter name="thresh" dimension="voltage"/>
  <Constant name="MSEC" dimension="time" value="1ms"/>
  <Constant name="MVOLT" dimension="voltage" value="1mV"/>
  <Attachments name="synapses" type="basePointCurrentDL"/>
  <Exposure name="U" dimension="none"/>
  <Dynamics>
    <StateVariable name="v" dimension="voltage" exposure="v"/>
    <StateVariable name="U" dimension="none" exposure="U"/>
    <DerivedVariable name="ISyn" dimension="none" select="synapses[*]/I" reduce="add"/>
    <TimeDerivative variable="v" value="(0.04 * v^2 / MVOLT + 5 * v + (140.0 - U + ISyn) * MVOLT)/MSEC"/>
    <TimeDerivative variable="U" value="a * (b * v / MVOLT - U) / MSEC"/>
    <OnStart>
      <StateAssignment variable="v" value="v0"/>
      <StateAssignment variable="U" value="v0 * b / MVOLT"/>
    </OnStart>
    <OnCondition test="v .gt. thresh">
      <StateAssignment variable="v" value="c * MVOLT"/>
      <StateAssignment variable="U" value="U + d"/>
      <EventOut port="spike"/>
    </OnCondition>
  </Dynamics>
</ComponentType>
```

```c
void izhikevichCellSimulation(izhikevichCell* neuron){
  float Isyn = reduceSynapticInput(neuron);
  float v = neuron->v;
  float U = neuron->U;
  float a = neuron->a;
  float b = neuron->b;
  float I = neuron->I;
  float thresh = neuron->thresh;
  float state_v;
  float state_U;
  float state_I
  if (OnStart) { ... }
  if (v > thresh) {
    neuron->v = c * MVOLT;
    neuron->U = U + d;
    emitspike(neuron);
  }
  if (t > Idel && t < Idel + Idur) {
    state_I = Iamp;
  }
  state_v = (0.04 * v*v*2 / MVOLT + 5 * v + (140.0 - U + I + ISyn) * MVOLT)/MSEC;
  state_u = a * (b * v / MVOLT - U) / MSEC;
  neuron->v = state_v;
  neuron->u = state_u;
  neuron->I = state_I;
}
```
Going massively parallel

• Neural simulation is embarrassingly parallel

• Communication is the main problem
  • Conventional systems – including PC Clusters, GPUs, BlueGene etc. suffer from this

• Once single threaded version is available use OpenMP and MPI to implement multi-threaded C-version
  • Runs on standard multi-processor architectures
Future work and roadmap

- Single threaded Izhikevich neural simulator
- Multi-threaded Izhikevich with OpenMP/MPI
- Create Bluehive compatible version
  - Identify required hardware accelerator blocks
  - Potential vector-processing architecture
- Addition neural model support: IaF, LIF, Fitz-Hugh-Nagumo etc.
- Open Bluehive/HiveMind to neuroscience community for running remote jobs
Speculation: Software Vs. Custom Hardware

- Communication is major source of bottlenecks in standard systems
- Bluehive takes a communication centric-approach
- Common von Neumann bottleneck (memory)
- Processing time of neuron small compared to summation of synaptic inputs – (FPGA Custom hardware accelerator can help here)
- Near-optimum performance if we saturate the memory link despite software algorithm
- Aim is to get HiveMind in same ball park as custom written Izhikevich bluespec (HDL) whilst also solving for the general case