MUSIC

Mikael Djurfeldt, PDC/KTH
HBP Neuromorphic SP
Outline

● Interfaces in computational neuroscience software
● What is MUSIC?
● Two problems solved by MUSIC:
  ● spatial aliasing problem
  ● temporal aliasing problem
● How to use MUSIC from C++, Python and PyNN
● Use cases
● Where to get software and documentation
Simulation environments in computational neuroscience, such as NEURON, NEST or Brian, each provide many tools needed by the user to carry out high-quality simulation studies.

Models described differently, environments have specific features => hard to move models

Difficult to build larger simulations which re-use existing models
Interfaces in computational neuroscience

- As systems grow and encompass more subsystems, they rapidly become unwieldy to develop
- In general, software in computational neuroscience tends to have a monolithic structure
- *Software interfaces* (APIs) allow for use of different implementations of software components
- MUSIC is an API, and implementation in the form of a C++ library, supporting flowing of data between tools during simulation (INCF initiative, originally developed by Ö. Ekeberg and M. Djurfeldt)
MUSIC co-simulations

A co-simulation with multiple parallel applications (A, B, C) exchanging runtime data (such as neuronal events)

Shipping data around between applications during simulation useful e.g. for:

• Building larger models by combining models as components
• Modeling multiple scales and/or combining different formalisms simultaneously
• Pre/postprocessing and visualization
• Interfacing to external hardware
Network simulation

![Diagram of network simulation with labels: Simulator, Neuronal populations, Projection ("axons")]}
Using MUSIC to expose data
Co-simulation

Second simulator

MUSIC input port

Reflected data source
Loop
Spatial aliasing problem

Processes
Spatial aliasing
Scheduling of communication

Handling of time in MUSIC

• An application calls MUSIC `tick()` at points regularly spaced in simulated time
• This is where data may be sent and/or received
• Different applications are allowed to call `tick()` at different rates
• MUSIC may allow applications to run out-of-sync (each with its own offset between simulation time and wallclock time)
• MUSIC allows complex topology of port connectivity

Scheduling problem

• How to deliver data in time while avoiding deadlocks
• How to interpolate continuous data given different tick rates
Interfaces to MUSIC

PyNN

Python

Application

MUSIC

MPI

UDP

ZMQ

MUSIC API

Python API

PyNN API
C++ app: eventsource

```cpp
#include <music.hh>

int main (int argc, char *argv[])
{
    // Get real argc and argv
    MUSIC::Setup* setup = new MUSIC::Setup (argc, argv);
    ...
    // Publish an output port
    MUSIC::EventOutputPort* out = setup->publishEventOutput (portName);
    ...
    // Associate the port with neurons ("map" the port)
    MUSIC::LinearIndex indices (firstIndex, nLocalUnits);
    out->map (&indices, MUSIC::Index::GLOBAL);
    ...
    // Prepare for simulation
    MUSIC::Runtime* runtime = new MUSIC::Runtime (setup, timestep);
    ...
    // Simulation loop
    ...
    // End simulation
    runtime->finalize ();
    ...
}


C++ app: eventsource

```cpp
#include <music.hh>

int main (int argc, char *argv[]) {

    ... 

    // Simulation loop
    spikeFile >> t >> id;
    double time = runtime->time ();
    while (time < stoptime) {
        double nextTime = time + timestep;
        while (!spikeFile.eof () && t < nextTime) {
            out->insertEvent (t, MUSIC::GlobalIndex (id));
            spikeFile >> t >> id;
        }

        // Make data available for other programs
        runtime->tick ();

        time = runtime->time ();
    }

    ... 
}
```
C++ app: eventlogger

#include <music.hh>

class MyEventHandlerGlobal : public MUSIC::EventHandlerGlobalIndex {
  public:
    void operator () (double t, MUSIC::GlobalIndex id)
    {
      // Print incoming event
      std::cout << t << "\t" << id << std::endl;
    }
  
};

int main (int argc, char *argv[])
{
  ...
  double time = runtime->time ();
  while (time < toptime)
    {
      // Retrieve data from other program
      runtime->tick ();

      time = runtime->time ();
    }
  ...
}
MUSIC configuration file

np = 1
stoptime = 1.0

[A]
binary = ./eventsourcing
args = 10 spikes

[B]
binary = ./eventlogger
args = 10

A.out->B.in [10]
Interfaces to MUSIC

PyNN

Python

Application

MUSIC

MPI

UDP

ZMQ
import music

# Get setup handle
setup = music.Setup()

# Publish an output port
out = setup.publishEventOutput("out")

# Associate the port with neurons ("map" the port)
out.map(music.Index.GLOBAL, base=firstId, size=local)

# Prepare for simulation
runtime = setup.runtime(timestep)
...

# Simulation loop
try:
    t, id = next(spikes)
    while time < stoptime:
        nextTime = time + timestep
        while t < nextTime:
            out.insertEvent(t, id, music.Index.GLOBAL)
            runtime.tick()
            time = runtime.time()
except StopIteration:
    pass
Interfaces to MUSIC

Diagram:
- PyNN
  - Python
    - Application
      - MUSIC
        - MUSIC API
        - Python API
        - PyNN API
      - MPI
      - UDP
      - ZMQ
from pyNN import music

sim1, sim2 = music.setup(music.Config("nest", 1), music.Config("neuron", 1))

[...]

sim1.setuptimestep=0.1, min_delay=0.2, max_delay=1.0)
sim2.setuptimestep=0.1, min_delay=0.2, max_delay=1.0)

input_population = sim1.Population(1,
        sim1.SpikeSourceArray,
        {'spike_times': spike_times},
        label="input")

output_population = sim2.Population(2,
        sim2.IF_curr_alpha,
        cell_params,
        label="output")

# The connector is used on the receiving side (sim2)
projection = music.Projection(input_population, output_population,
        sim2.AllToAllConnector())

music.run(tstop)

output_population.printSpikes("Results/simpleNetwork_output.ras")
music.end()
Usage scenarios
Integrated simulation of the whole-body musculo-skeletal-nervous system for clarification of motor dysfunctions due to neurological diseases

Jun Igarashi¹, Jan Moren¹, Osamu Shouno³, Kazuya Shimizu², Naoto Yamamura², Junichiro Yoshimoto¹
Shu Takagi² & Kenji Doya¹

1: Okiniwa Institute of Science and Technology (OIST)
2: Tokyo University
3: Honda research Institute Japan
• Full size, neuron count of rat BG and motor cortex. (~3.2 million neurons)
• Conductance-based IaF or Izhikevich-type neuron models, static or STDP synapses.
• PyNEST and SLI models, connected with MUSIC
• dDIMS: Volumetric FEM-based fluid-mechanical muscle model, full skeletal physics model
MUSIC organization

Sample patches/columns on each surface:

- **L5BCS → Striatum**: 50×50 patches
- **L5BPT → Spinal cord**: 20x20 patches
- **Gpi/EP → Thalamus TC/HT**: 1 neuron per channel (3100 total)
- **Spinal cord → Muscle**: 1 motor neuron per channel (~750/biceps, 1500/triceps)
preliminary results

- **Up:** GPe, STN and GPi neurons oscillating at about 14.7Hz.
- **Left:** power spectrum of GPi, Thalamic CT neurons and L5B PT neurons.
Other highlighted use cases

- Bluebrain Monsteer
  Library for interactive visualization

- MUSIC-ROS toolchain
  Philipp Weidel Thursday 10:10
Where to get MUSIC

- Github INCF/MUSIC
- MUSIC manual in the distribution
Thanks

- Ekaterina Brocke, SciLife lab, KI – communication algorithms
- Alexander Peyser, Simlab neurosci, FZJ – Python interface
- Andrew Davison, Jochen Eppler and Eilif Muller – PyNN interface
- Rajalekshmi Deepu, Simlab neurosci, FZJ – Travis integration
- Jan Morén, OIST – MUSIC application example
- INCF
- HBP
- Simlab neuroscience
- INM6, FZ Juelich