SpiNNaker: What’s New

Andrew Rowley and Alan Stokes

HBP CodeJam Workshop #7

Jan 2016
Overview

- SpiNNaker
- Summary of tools to date
- PACMAN Work Flows
  - External Algorithm Support
- PyNN Front End New Features
  - Repeated Runs and Reset
  - Closed Loop Simulations
- The Graph Front End
- Coming Soon!
SpiNNaker
SpiNNaker
SpiNNaker
Tools Summary

● Previous releases:
  ■ Just Testing       April 2014
  ■ Little Rascal      April 2015
  ■ Arbitrary            September 2015

● Arbitrary functionality:
  ■ Basic PyNN 0.7 functionality
  ■ Live closed-loop functionality

● Hope to make next release soon after HBP codeJam #7
  ■ With merged achievements from the hackathon.
PACMAN Work Flows - 1 of 10

PACMAN Work Flows - 2 of 10

Algorithm Descriptions (XML) -> PACMAN -> Create Workflow
<algorithms>
  <algorithm name="BasicPlacer">
    <python_module>pacman.operations.placer_algorithms.basic_placer</python_module>
    <python_class>BasicPlacer</python_class>
    <required_inputs>
      <parameter>
        <param_name>partitioned_graph</param_name>
        <param_type>MemoryPartitionedGraph</param_type>
      </parameter>
      <parameter>
        <param_name>machine</param_name>
        <param_type>MemoryExtendedMachine</param_type>
      </parameter>
    </required_inputs>
    <produces_outputs>
      <parameter>
        <param_name>placements</param_name>
        <param_type>MemoryPlacements</param_type>
      </parameter>
    </produces_outputs>
  </algorithm>
</algorithms>
<algorithms>
  <algorithm name="ExternalRouter">
    <command_line_args>
      <arg>run_router.py</arg>
      <arg>--graph={graph}</arg>
      <arg>--machine={machine}</arg>
      <arg>--placements={placements}</arg>
      <arg>--algorithm=ner</arg>
    </command_line_args>
    <required_inputs>
      <parameter>
        <param_name>graph</param_name>
        <param_type>FilePartitionedGraph</param_type>
      </parameter>
      <parameter>
        <param_name>machine</param_name>
        <param_type>FileMachine</param_type>
      </parameter>
      <parameter>
        <param_name>placements</param_name>
        <param_type>FilePlacements</param_type>
      </parameter>
    </required_inputs>
    <produces_outputs>
      <parameter>
        <param_name>FileRoutingPathsFilePath</param_name>
        <param_type>FileRoutingPaths</param_type>
      </parameter>
    </produces_outputs>
  </algorithm>
</algorithms>
PACMAN Work Flows - 5 of 10

- Algorithm Descriptions (XML)
- Algorithms To Run
- Outputs to Generate
- Create Workflow
- PACMAN

- Inputs
PACMAN Work Flow

Inputs: MemoryPartitionableGraph and HostName

MemoryExtendedMachine, MemoryPartitionedGraph

Basic Placer
MemoryPlacements

FilePartitionedGraph, FilePlacements, FileMachine

External Router
FileRoutingPaths

MemoryMachine, MemoryPartitionableGraph

Basic Partitioner
MemoryPartitionedGraph

HostName

Basic Partitioner
MemoryPartitionedGraph

MemoryMachine
MemoryMachine

Machine Finder

Outputs: MemoryRoutingPaths
PACMAN Work Flows - 7 of 10
PACMAN Work Flow

1. Machine Finder
2. Basic Partitioner
3. Basic Placer
4. PartitionedGraphFileWriter
5. PlacementsFileWriter
6. MachineFileWriter
7. External Router
8. RoutingPathsFileReader
PACMAN Work Flows - 9 of 10

JSON Files: machine.json

```json
{
  "width": 2,
  "height": 2,
  "chip_resources": {
    "cores": 17,
    "sdram": 119275520
  },
  "dead_chips": [],
  "dead_links": [
    [0, 0, "west"], [1, 1, "east"],
    [0, 0, "south_west"], [1, 1, "north_east"],
    [1, 0, "north_east"], [0, 1, "south_west"],
    [0, 1, "west"], [1, 0, "east"]
  ],
  "chip_resource_exceptions": [
    [0, 0, {"tags": 8}]
  ]
}
```
[Mapping]

# Name of extra algorithms to execute in workflow
algorithms = MyPlacer, MyRouter

# Path to extra algorithm description XML file(s)
extra_xml_paths = /path/to/myxml.xml
import pyNN.spiNNaker as p
p.setup(timestep=1.0, min_delay=1.0, max_delay=144.0)
populations = list()

# Create pops and projections for a synfire chain
...

populations[0].record()
p.run(5000)
spikes = populations[0].getSpikes()

# Plot spikes 1
p.run(2000) <= 5000
spikes = populations[0].getSpikes()

# Plot spikes 2
p.end()
import pyNN.spiNNaker as p
p.setup(timestep=1.0, min_delay=1.0, max_delay=144.0)
populations = list()

# Create pops and projections for a synfire chain
...

populations[0].record()
p.run(3000)
p.run(2000)
spikes = populations[0].getSpikes()

# Plot spikes 1

p.reset()
p.run(3000)  # == 3000
spikes = populations[0].getSpikes()

# Plot spikes 2

p.end()
PyNN FrontEnd new functionality
Delayed closed loop simulations 1 of 2

Virtual Neurorobotics Environment

- p.run(1000)
- p.get_spikes()
- Spikes
- Update
- SpikeSourceArray

sPyNNaker
PyNN FrontEnd new functionality
Delayed closed loop simulations 2 of 2

Thanks to Felix Schneider from SP 10 for helping with this
The Graph Front End

- Heat Demo
- Key-Value Pair Database
- ...

- sPyNNaker

- SpiNNFrontEndCommon

- DataSpecification
- PACMAN
- SpiNNMan
Coming Soon!!!

1. Data expansion on chip to reduce load times.
2. Automatic pause and resume of simulation to allow recording of long running and/or large simulations on smaller machines.
3. PyNN 0.8 support.
4. Refactoring of delay representation to improve memory usage.