Neurokernel Documentation

Release 0.1.0

Lev Givon

Aug 18, 2017

Contents

Project Website | GitHub Repository | Mailing List | Forum

CHAPTER 1

Contents

Introduction

Neurokernel is an open software platform written in Python for emulation of the brain of the fruit fly (*Drosophila melanogaster*) on multiple Graphics Processing Units (GPUs). It provides a programming model based upon the organization of the fly's brain into fewer than 50 modular subdivisions called *local processing units* (LPUs) that are each characterized by unique populations of local neurons[1]. Using Neurokernel's API, researchers can develop models of individual LPUs and combine them with other independently developed LPU models to collaboratively construct models of entire subsystems of the fly brain. Neurokernel's support for LPU model integration also enables exploration of brain functions that cannot be exclusively attributed to individual LPUs or brain subsystems.

Examples of Neurokernel's use are available on the project website.

Installation

Prerequisites

Neurokernel requires

- Linux (other operating systems may work, but have not been tested);
- Python 2.7 (Python 3.0 is not guaranteed to work);
- at least one NVIDIA GPU with Fermi architecture or later;
- NVIDIA's GPU drivers;
- CUDA 5.0 or later;
- OpenMPI 1.8.4 or later compiled with CUDA support.

To check what GPUs are in your system, you can use the inxi command available on most Linux distributions:

[1] Chiang, A.-S., Lin, C.-Y., Chuang, C.-C., Chang, H.-M., Hsieh, C.-H., Yeh, C.-W., et al. (2011), Three-dimensional reconstruction of brain-wide wiring networks in Drosophila at single-cell resolution, Current Biology, 21, 1, 1–11, doi:10.1016/j.cub.2010.11.056

```
inxi -G
```

You can verify that the drivers are loaded as follows:

```
lsmod | grep nvidia
```

If no drivers are present, you may have to manually load them by running something like:

```
modprobe nvidia
```

as root.

Although some Linux distributions do include CUDA in their stock package repositories, you are encouraged to use those distributed by NVIDIA because they often are more up-to-date and include more recent releases of the GPU drivers. See this page for download information.

If you install Neurokernel in a virtualenv environment, you will need to install OpenMPI. See this page for OpenMPI installation information. *Note that OpenMPI 1.8 cannot run on Windows*.

Some of Neurokernel's demos require either ffmpeg or libav installed to generate visualizations (see *Examples*).

Installation

Download the latest Neurokernel code as follows:

```
git clone https://github.com/neurokernel/neurokernel.git
```

Since Neurokernel requires a fair number of additional Python packages to run, it is recommended that it either be installed in a virtualenv or conda environment. Follow the relevant instructions below.

Virtualenv

See this page for virtualenv installation information.

Create a new virtualenv environment and install several required dependencies:

```
cd ~/
virtualenv NK
~/NK/bin/pip install numpy cython numexpr pycuda
```

If installation of PyCUDA fails because some of the CUDA development files or libraries are not found, you may need to specify where they are explicitly. For example, if CUDA is installed in `/usr/local/cuda/`, try installing PyCUDA as follows:

```
CUDA_ROOT=/usr/local/cuda/ CFLAGS=-I${CUDA_ROOT}/include \
LDFLAGS=-L${CUDA_ROOT}/lib64 ~/NK/bin/pip install pycuda
```

Replace `${CUDA_ROOT}/lib` with `${CUDA_ROOT}/lib64` if your system is running 64-bit Linux. If you continue to encounter installation problems, see the PyCUDA Wiki for more information.

Run the following to install the remaining Python package dependencies listed in *setup.py*:

```
cd ~/neurokernel
~/NK/bin/python setup.py develop
```

Conda

Note that conda packages are currently only available for 64-bit Ubuntu Linux 14.04. If you would like packages for another distribution, please submit a request to the Neurokernel developers.

First, install the following Ubuntu packages:

- `libibverbs1`
- `libnuma1`
- `libpmi0`
- `libslurm26`
- `libtorque2`

Tthese are required by the conda OpenMPI packages prepared for Neurokernel. Ensure that the stock Ubuntu OpenMPI packages are not installed because they may interfere with the ones that will be installed by conda. You also need to ensure that CUDA has been installed in `/usr/local/cuda`.

Install conda by either installing Anaconda or Miniconda. Make sure that the following lines appear in your *~/.condarc* file so that conda can find the packages required by Neurokernel:

```
channels:
- https://conda.binstar.org/neurokernel/channel/ubuntu1404
- defaults
```

Create a new conda environment containing the packages required by Neurokernel by running the following command:

```
conda create -n NK neurokernel_deps
```

PyCUDA packages compiled against several versions of CUDA are available. If you need one compiled against a specific version that differs from the one automatically installed by the above command, you will need to manually install it afterwards as follows (replace `cuda75` with the appropriate version):

```
source activate NK
conda install pycuda=2015.1.3=np110py27_cuda75_0
source deactivate
```

Activate the new environment and install Neurokernel in it as follows:

```
source activate NK
cd ~/neurokernel
python setup.py develop
```

Examples

Introductory examples of how to use Neurokernel to build and integrate models of different parts of the fly brain are available in the Neurodriver package. To install it run the following:

```
git clone https://github.com/neurokernel/neurodriver
cd ~/neurodriver
python setup.py develop
```

Other models built using Neurokernel are available on GitHub.

Building the Documentation

To build Neurokernel's HTML documentation locally, you will need to install

- mock 1.0 or later.
- sphinx 1.3 or later.
- sphinx_rtd_theme 0.1.6 or later.

Once these are installed, run the following:

```
cd ~/neurokernel/docs
make html
```

Reference

Model Development API

Local Processing Units

neurokernel.core.Module	Processing module.
neurokernel.core_gpu.Module	Processing module.

neurokernel.core.Module

class neurokernel.core.**Module**(*sel, sel_in, sel_out, sel_gpot, sel_spike, data_gpot, data_spike, columns=['interface', 'io', 'type'], ctrl_tag=1, gpot_tag=2, spike_tag=3, id=None, device=None, routing_table=None, rank_to_id=None, debug=False, time_sync=False*)

Processing module.

This class repeatedly executes a work method until it receives a quit message via its control network port.

Parameters

- **sel** (str, *unicode, or sequence*) – Path-like selector describing the module's interface of exposed ports.
- **sel_out, sel_gpot, sel_spike** (*sel_in,*) – Selectors respectively describing all input, output, graded potential, and spiking ports in the module's interface.
- **data_spike** (*data_gpot,*) – Data arrays associated with the graded potential and spiking ports in the . Array length must equal the number of ports in a module's interface.
- **columns** (*list of str*) – Interface port attributes. Network port for controlling the module instance.
- **gpot_tag, spike_tag** (ctrl_tag,) – MPI tags that respectively identify messages containing control data, graded potential port values, and spiking port values transmitted to worker nodes.
- **id** (str) – Module identifier. If no identifier is specified, a unique identifier is automatically generated.
- **device** (*int*) – GPU device to use. May be set to None if the module does not perform GPU processing.

- **routing_table** (`neurokernel.routing_table.RoutingTable`) – Routing table describing data connections between modules. If no routing table is specified, the module will be executed in isolation.
- **rank_to_id** (*bidict.bidict*) – Mapping between MPI ranks and module object IDs.
- **debug** (*bool*) – Debug flag. When True, exceptions raised during the work method are not be suppressed.
- **time_sync** (*bool*) – Time synchronization flag. When True, debug messages are not emitted during module synchronization and the time taken to receive all incoming data is computed.

interface
Interface – Object containing information about a module's ports.

pm
dict – *pm['gpot']* and *pm['spike']* are instances of neurokernel.pm.PortMapper that map a module's ports to the contents of the values in *data*.

data
dict – *data['gpot']* and *data['spike']* are arrays of data associated with a module's graded potential and spiking ports.

__init__(*sel, sel_in, sel_out, sel_gpot, sel_spike, data_gpot, data_spike, columns=['interface', 'io', 'type'], ctrl_tag=1, gpot_tag=2, spike_tag=3, id=None, device=None, routing_table=None, rank_to_id=None, debug=False, time_sync=False*)

Methods

`__init__`(sel, sel_in, sel_out, sel_gpot, ...)	
`do_work`()	Work method.
`post_run`()	Code to run after main loop.
`pre_run`()	Code to run before main loop.
`recv_parent`([tag])	Receive data from parent process.
`recv_peer`([source, tag])	
`run`()	Body of process.
`run_step`()	Module work method.
`send_parent`(data[, tag])	Send data to parent process.
`send_peer`(data, dest[, tag])	Send data to peer process.

Attributes

`intercomm`	Intercommunicator to access parent process.
`intracomm`	Intracommunicator to access peer processes.
`log_on`	Logger switch.
`max_steps`	Maximum number of steps to execute.
`rank`	MPI process rank.
`size`	Number of peer processes.

neurokernel.core_gpu.Module

class `neurokernel.core_gpu.`**`Module`**(*sel, sel_in, sel_out, sel_gpot, sel_spike, data_gpot, data_spike, columns=['interface', 'io', 'type'], ctrl_tag=1, gpot_tag=2, spike_tag=3, id=None, device=None, routing_table=None, rank_to_id=None, debug=False, time_sync=False*)

Processing module.

This class repeatedly executes a work method until it receives a quit message via its control network port.

Parameters

- **sel** (*str, unicode, or sequence*) – Path-like selector describing the module's interface of exposed ports.
- **sel_out, sel_gpot, sel_spike** (*sel_in,*) – Selectors respectively describing all input, output, graded potential, and spiking ports in the module's interface.
- **data_spike** (*data_gpot,*) – Data arrays associated with the graded potential and spiking ports in the . Array length must equal the number of ports in a module's interface.
- **columns** (*list of str*) – Interface port attributes. Network port for controlling the module instance.
- **gpot_tag, spike_tag** (*ctrl_tag,*) – MPI tags that respectively identify messages containing control data, graded potential port values, and spiking port values transmitted to worker nodes.
- **id** (*str*) – Module identifier. If no identifier is specified, a unique identifier is automatically generated.
- **device** (*int*) – GPU device to use. May be set to None if the module does not perform GPU processing.
- **routing_table** (*neurokernel.routing_table.RoutingTable*) – Routing table describing data connections between modules. If no routing table is specified, the module will be executed in isolation.
- **rank_to_id** (*bidict.bidict*) – Mapping between MPI ranks and module object IDs.
- **debug** (*bool*) – Debug flag. When True, exceptions raised during the work method are not be suppressed.
- **time_sync** (*bool*) – Time synchronization flag. When True, debug messages are not emitted during module synchronization and the time taken to receive all incoming data is computed.

interface

Interface – Object containing information about a module's ports.

pm

dict – *pm['gpot']* and *pm['spike']* are instances of neurokernel.pm_gpu.PortMapper that map a module's ports to the contents of the values in *data*.

data

dict – *data['gpot']* and *data['spike']* are arrays of data associated with a module's graded potential and spiking ports.

`__init__`(*sel, sel_in, sel_out, sel_gpot, sel_spike, data_gpot, data_spike, columns=['interface', 'io', 'type'], ctrl_tag=1, gpot_tag=2, spike_tag=3, id=None, device=None, routing_table=None, rank_to_id=None, debug=False, time_sync=False*)

Methods

`__init__`(sel, sel_in, sel_out, sel_gpot, ...)	
`do_work`()	Work method.
`post_run`()	Code to run after main loop.
`pre_run`()	Code to run before main loop.
`recv_parent`([tag])	Receive data from parent process.
`recv_peer`([source, tag])	
`run`()	Body of process.
`run_step`()	Module work method.
`send_parent`(data[, tag])	Send data to parent process.
`send_peer`(data, dest[, tag])	Send data to peer process.

Attributes

`intercomm`	Intercommunicator to access parent process.
`intracomm`	Intracommunicator to access peer processes.
`log_on`	Logger switch.
`max_steps`	Maximum number of steps to execute.
`rank`	MPI process rank.
`size`	Number of peer processes.

Inter-LPU Connectivity

`neurokernel.pattern.Interface`	Container for set of interface comprising ports.
`neurokernel.pattern.Pattern`	Connectivity pattern linking sets of interface ports.

neurokernel.pattern.Interface

class `neurokernel.pattern.`**`Interface`**(*selector='', columns=['interface', 'io', 'type']*)

Container for set of interface comprising ports.

This class contains information about a set of interfaces comprising path-like identifiers and the attributes associated with them. By default, each port must have at least the following attributes; other attributes may be added:

- interface - indicates which interface a port is associated with.
- io - indicates whether the port receives input ('in') or emits output ('out').
- type - indicates whether the port emits/receives spikes or graded potentials.

All port identifiers in an interface must be unique. For two interfaces to be deemed compatible, they must contain the same port identifiers and their identifiers' 'io' attributes must be the inverse of each other (i.e., every 'in' port in one interface must be mirrored by an 'out' port in the other interface.

Examples

```
>>> i = Interface('/foo[0:4],/bar[0:3]')
>>> i['/foo[0:2]', 'interface', 'io', 'type'] = [0, 'in', 'spike']
>>> i['/foo[2:4]', 'interface', 'io', 'type'] = [1, 'out', 'spike']
```

data
pandas.DataFrame – Port attribute data.

index
pandas.MultiIndex – Index of port identifiers.

Parameters

- **selector** (*str, unicode, or sequence*) – Selector string (e.g., 'foo[0:2]') or sequence of token sequences (e.g., [['foo', (0, 2)]]) describing the port identifiers comprised by the interface.
- **columns** (*list, default = ['interface', 'io', 'type']*) – Data column names.

See also:

plsel.SelectorMethods

__init__(*selector='', columns=['interface', 'io', 'type']*)

Methods

`__init__`([selector, columns])	
`clear`()	Clear all ports in class instance.
`copy`()	Make a copy of this object.
`data_select`(f[, inplace])	Restrict Interface data with a selection function.
`equals`(other)	Check whether this interface is equivalent to another interface.
`from_csv`(file_name, **kwargs)	Create an Interface from a properly formatted CSV file.
`from_df`(df)	Create an Interface from a properly formatted DataFrame.
`from_dict`(d)	Create an Interface from a dictionary of selectors and data values.
`from_graph`(g)	Create an Interface from a NetworkX graph.
`from_selectors`(sel[, sel_in, sel_out, ...])	Create an Interface instance from selectors.
`get_common_ports`(a, i, b[, t])	Get port identifiers common to this and another Interface instance.
`gpot_ports`([i, tuples])	Restrict Interface ports to graded potential ports.
`in_ports`([i, tuples])	Restrict Interface ports to input ports.
`interface_ports`([i, tuples])	Restrict Interface ports to specific interface.
`is_compatible`(a, i, b[, allow_subsets])	Check whether two interfaces can be connected.
`is_in_interfaces`(s)	Check whether ports comprised by a selector are in the stored interfaces.
`out_ports`([i, tuples])	Restrict Interface ports to output ports.
`port_select`(f[, inplace])	Restrict Interface ports with a selection function.
`set_pm`(t, pm)	Set port mapper associated with a specific port type.
`spike_ports`([i, tuples])	Restrict Interface ports to spiking ports.

Continued on next page

Table 1.7 – continued from previous page

to_selectors([i])	Retrieve Interface's port identifiers as list of path-like selectors.
to_tuples([i])	Retrieve Interface's port identifiers as list of tuples.
which_int(s)	Return the interface containing the identifiers comprised by a selector.

Attributes

idx_levels	Number of levels in Interface index.
index	Interface index.
interface_ids	Interface identifiers.
io_inv	Returns new Interface instance with inverse input-output attributes.

neurokernel.pattern.Pattern

class neurokernel.pattern.**Pattern**(**selectors*, ***kwargs*)

Connectivity pattern linking sets of interface ports.

This class represents connection mappings between interfaces comprising sets of ports. Ports are represented using path-like identifiers; the presence of a row linking the two identifiers in the class' internal index indicates the presence of a connection. A single data attribute ('conn') associated with defined connections is created by default. Specific attributes may be accessed by specifying their names after the port identifiers; if a nonexistent attribute is specified when a sequential value is assigned, a new column for that attribute is automatically created:

```
p['/x[0]', '/y[0]', 'conn', 'x'] = [1, 'foo']
```

The direction of connections between ports in a class instance determines whether they are input or output ports. Ports may not both receive input or emit output. Patterns may contain fan-out connections, i.e., one source port connected to multiple destination ports, but not fan-in connections, i.e., multiple source ports connected to a single destination port.

Examples

```
>>> p = Pattern('/x[0:3]','/y[0:4]')
>>> p['/x[0]', '/y[0:2]'] = 1
>>> p['/y[2]', '/x[1]'] = 1
>>> p['/y[3]', '/x[2]'] = 1
```

data

pandas.DataFrame – Connection attribute data.

index

pandas.MultiIndex – Index of connections.

interface

Interface – Interfaces containing port identifiers and attributes.

Parameters

- **sel1, ..** (*sel0,*) – Selectors defining the sets of ports potentially connected by the pattern. These selectors must be disjoint, i.e., no identifier comprised by one selector may be in any other selector.
- **columns** (*sequence of str*) – Data column names.

See also:

plsel.SelectorMethods

__init__(**selectors*, ***kwargs*)

Methods

__init__(*selectors, **kwargs)	
clear()	Clear all connections in class instance.
connected_port_pairs([as_str])	Return connections as pairs of port identifiers.
connected_ports([i, tuples])	Return ports that are connected by the pattern.
dest_idx(src_int, dest_int[, src_type, ...])	Retrieve destination ports connected to the specified source ports.
from_concat(*selectors, **kwargs)	Create pattern from the concatenation of identifers in two selectors.
from_csv(file_name, **kwargs)	Read connectivity data from CSV file.
from_df(df_int, df_pat)	Create a Pattern from properly formatted DataFrames.
from_graph(g)	Convert a NetworkX directed graph into a Pattern instance.
from_product(*selectors, **kwargs)	Create pattern from the product of identifiers comprised by two selectors.
gpot_ports([i, tuples])	Restrict Interface ports to graded potential ports.
in_ports([i, tuples])	Restrict Interface ports to input ports.
interface_ports([i, tuples])	Restrict Interface ports to specific interface.
is_connected(from_int, to_int)	Check whether the specified interfaces are connected.
is_in_interfaces(selector)	Check whether a selector is supported by any stored interface.
out_ports([i, tuples])	Restrict Interface ports to output ports.
spike_ports([i, tuples])	Restrict Interface ports to spiking ports.
split_multiindex(idx, a, b)	Split a single MultiIndex into two instances.
src_idx(src_int, dest_int[, src_type, ...])	Retrieve source ports connected to the specified destination ports.
to_graph()	Convert the pattern to a networkx directed graph.
which_int(s)	Return the interface containing the identifiers comprised by a selector.

Attributes

from_slice	Slice of pattern index row corresponding to source port(s).
index	Pattern index.
interface_ids	Interface identifiers.
to_slice	Slice of pattern index row corresponding to destination port(s).

Emulation Management

Construction and Execution

neurokernel.core.Manager	Module manager.
neurokernel.core_gpu.Manager	Module manager.

neurokernel.core.Manager

class neurokernel.core.**Manager**(*required_args=['sel', 'sel_in', 'sel_out', 'sel_gpot', 'sel_spike'], ctrl_tag=1*)

Module manager.

Instantiates, connects, starts, and stops modules comprised by an emulation. All modules and connections must be added to a module manager instance before they can be run.

ctrl_tag
int – MPI tag to identify control messages.

modules
dict – Module instances. Keyed by module object ID.

routing_table
routing_table.RoutingTable – Table of data transmission connections between modules.

rank_to_id
bidict.bidict – Mapping between MPI ranks and module object IDs.

__init__(*required_args=['sel', 'sel_in', 'sel_out', 'sel_gpot', 'sel_spike'], ctrl_tag=1*)

Methods

__init__([required_args, ctrl_tag])	
add(target, id, *args, **kwargs)	Add a module class to the emulation.
connect(id_0, id_1, pat[, int_0, int_1])	Specify connection between two module instances with a Pattern instance.
process_worker_msg(msg)	
quit()	Tell the workers to quit.
recv([tag])	Receive data from child process.
send(data, dest[, tag])	Send data to child process.
spawn()	Spawn MPI processes for and execute each of the managed targets.
start([steps])	Tell the workers to start processing data.
stop()	Tell the workers to stop processing data.
validate_args(target)	Check whether a class' constructor has specific arguments.
wait()	

Attributes

average_step_sync_time	Average step synchronization time.

Continued on next page

Table 1.13 – continued from previous page

average_throughput	Average received data throughput per step.
intercomm	Intercommunicator to spawned processes.
log_on	Logger switch.
total_throughput	Total received data throughput.

neurokernel.core_gpu.Manager

class neurokernel.core_gpu.**Manager**(*required_args=['sel', 'sel_in', 'sel_out', 'sel_gpot', 'sel_spike'], ctrl_tag=1*)

Module manager.

Instantiates, connects, starts, and stops modules comprised by an emulation. All modules and connections must be added to a module manager instance before they can be run.

ctrl_tag
int – MPI tag to identify control messages.

modules
dict – Module instances. Keyed by module object ID.

routing_table
routing_table.RoutingTable – Table of data transmission connections between modules.

rank_to_id
bidict.bidict – Mapping between MPI ranks and module object IDs.

__init__(*required_args=['sel', 'sel_in', 'sel_out', 'sel_gpot', 'sel_spike'], ctrl_tag=1*)

Methods

__init__([required_args, ctrl_tag])	
add(target, id, *args, **kwargs)	Add a module class to the emulation.
connect(id_0, id_1, pat[, int_0, int_1])	Specify connection between two module instances with a Pattern instance.
process_worker_msg(msg)	
quit()	Tell the workers to quit.
recv([tag])	Receive data from child process.
send(data, dest[, tag])	Send data to child process.
spawn()	Spawn MPI processes for and execute each of the managed targets.
start([steps])	Tell the workers to start processing data.
stop()	Tell the workers to stop processing data.
validate_args(target)	Check whether a class' constructor has specific arguments.
wait()	

Attributes

average_step_sync_time	Average step synchronization time.
average_throughput	Average received data throughput per step.

Continued on next page

Table 1.15 – continued from previous page

`intercomm`	Intercommunicator to spawned processes.
`log_on`	Logger switch.
`total_throughput`	Total received data throughput.

Support Classes

neurokernel.routing_table.RoutingTable	Routing table class.
neurokernel.mpi.Worker	MPI worker class.
neurokernel.mpi.WorkerManager	Self-launching MPI worker manager.

neurokernel.routing_table.RoutingTable

class `neurokernel.routing_table.`**`RoutingTable`**(*g=None*)

Routing table class.

Simple class that stores pairs of strings that can signify one-hop routes between entities in a graph. Assigning a value to a pair that isn't in the class instance will result in the pair and value being added to the class instance. All data associated with a specific connection is stored as a dict; if a non-dict is assigned to a connection, it is stored in a dict with key 'data'. Specific values in the dict can be retrieved by passing the desired key directly to the [] operator.

Examples

```
>>> r = RoutingTable()
>>> r['a', 'b'] = 1
>>> r['a', 'b']
1
>>> r['a', 'c'] = [1, 2]
>>> r['a', 'c']
[1, 2]
>>> r['a', 'c'] = {'x': 1}
>>> r['a', 'c', 'x']
1
>>> r['a', 'c']['x']
1
```

Parameters **g** (*networkx.DiGraph*) – Directed graph that describes the routes between entities.

`connections`

list – List of directed connections between identifiers.

`ids`

list – Identifiers currently in routing table.

`copy`()

Return a copy of the routing table.

`dest_ids`(*src_id*)

Destination identifiers connected to the specified source identifier.

`has_node`(*n*)

Check whether the routing table contains the specified identifier.

ids()
IDs currently in routing

src_ids(*dest_id*)
Source identifiers connected to the specified destination identifier.

subtable(*ids*)
Return subtable containing only those connections between specified identifiers.

to_df()
Return a pandas DataFrame listing all of the connections.

__init__(*g=None*)

Methods

`__init__`([g])	
`copy`()	
`dest_ids`(src_id)	Destination identifiers connected to the specified source identifier.
`has_node`(n)	Check whether the routing table contains the specified identifier.
`src_ids`(dest_id)	Source identifiers connected to the specified destination identifier.
`subtable`(ids)	Return subtable containing only those connections between specified identifiers.
`to_df`()	Return a pandas DataFrame listing all of the connections.

Attributes

`connections`	List of directed connections between identifiers.
`ids`	Identifiers currently in routing table.

neurokernel.mpi.Worker

class neurokernel.mpi.**Worker**(*ctrl_tag=1*, **args*, ***kwargs*)
MPI worker class.

This class repeatedly executes a work method.

Parameters **ctrl_tag** (*int*) – MPI tag to identify control messages transmitted to worker nodes.

__init__(*ctrl_tag=1*, **args*, ***kwargs*)

Methods

`__init__`([ctrl_tag])	
`do_work`()	Work method.
`post_run`()	Code to run after main loop.

Continued on next page

Table 1.19 – continued from previous page

pre_run()	Code to run before main loop.
recv_parent([tag])	Receive data from parent process.
recv_peer([source, tag])	
run()	Main body of worker process.
send_parent(data[, tag])	Send data to parent process.
send_peer(data, dest[, tag])	Send data to peer process.

Attributes

intercomm	Intercommunicator to access parent process.
intracomm	Intracommunicator to access peer processes.
log_on	Logger switch.
max_steps	Maximum number of steps to execute.
rank	MPI process rank.
size	Number of peer processes.

neurokernel.mpi.WorkerManager

class neurokernel.mpi.**WorkerManager**(*ctrl_tag=1*)

Self-launching MPI worker manager.

This class may be used to construct an MPI application consisting of

- a manager process that spawns MPI processes that execute the run() methods of several subclasses of the Worker class;
- worker processes that perform some processing task; and

The application should NOT be started via mpiexec.

Parameters **ctrl_tag** (*int*) – MPI tag to identify control messages transmitted to worker nodes. May not be equal to mpi4py.MPI.ANY_TAG

Notes

This class requires MPI-2 dynamic processing management.

See also:

Worker

__init__(*ctrl_tag=1*)

Methods

__init__([ctrl_tag])	
add(target, *args, **kwargs)	Add a worker to an MPI application.
process_worker_msg(msg)	Process the specified deserialized message from a worker.
quit()	Tell the workers to quit.

Continued on next page

Table 1.21 – continued from previous page

recv([tag])	Receive data from child process.
send(data, dest[, tag])	Send data to child process.
spawn()	Spawn MPI processes for and execute each of the managed targets.
start([steps])	Tell the workers to start processing data.
stop()	Tell the workers to stop processing data.
wait()	Wait for execution to complete.

Attributes

intercomm	Intercommunicator to spawned processes.
log_on	Logger switch.

Support Classes and Functions

Path-Like Port Identifier Handling

Selector	Validated and expanded port selector.
SelectorMethods	Class for manipulating and using path-like selectors.
SelectorParser	This class implements a parser for path-like selectors that can be associated with elements in a sequential data structure such as a Pandas DataFrame; in the latter case, each level of the selector corresponds to a level of a Pandas MultiIndex.

neurokernel.plsel.Selector

class neurokernel.plsel.**Selector**(*s*)

Validated and expanded port selector.

Parameters **s** (*Selector, str, or unicode*) – Existing Selector class instance or string representation. The selector may not be ambiguous. If an existing Selector instance is specified, the new instance is a copy of the existing instance.

str

str – String representation of selector.

expanded

tuple of tuples – Expanded selector.

max_levels

int – Maximum number of levels in selector.

__init__(*s*)

Methods

__init__(s)	

Continued on next page

Table 1.24 – continued from previous page

add(*sels)	Combine the identifiers in multiple selectors into a single selector.
add_str(*s)	Combine the identifiers in multiple selector strings into a single selector.
concat(*sels)	Concatenate the identifiers in multiple selectors elementwise.
prod(*sels)	Compute the product of identifiers in multiple selectors.
union(*sels)	Compute the union of the identifiers in multiple selectors.

Attributes

expanded	Expanded selector.
identifiers	List of individual identifiers in selector.
max_levels	Maximum number of levels in selector.
nonempty	True if the selector contains identifiers.
str	String representation of selector.

neurokernel.plsel.SelectorMethods

class `neurokernel.plsel.`**`SelectorMethods`**

Class for manipulating and using path-like selectors.

Contains class methods for expanding selectors, selecting rows from a Pandas DataFrame using a selector, etc.

The class can also be used to create new MultiIndex instances from selectors that can be fully expanded into an explicit set of identifiers (and therefore contain no ambiguous symbols such as '*' or '[:]').

`__init__`()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

are_consecutive(int_list)	Check whether a list of integers is consecutive.
are_disjoint(*selectors)	Check whether several selectors are disjoint.
collapse(selector)	Collapse a selector into a single string.
count_ports(selector)	Count number of distinct port identifiers in unambigious selector.
expand(selector[, pad_len])	Expand an unambiguous selector into a list of identifiers.
get_index(df, selector[, start, stop, names])	Return index corresponding to rows selected by specified selector.
get_tuples(df, selector[, start, stop])	Return tuples containing index labels selected by specified selector.
index_to_selector(idx)	Convert an index into an expanded port selector.
is_ambiguous(selector)	Check whether a selector cannot be expanded into an explicit list of identifiers.
is_expandable(selector)	Check whether a selector can be expanded into multiple identifiers.

Continued on next page

Table 1.26 – continued from previous page

`is_identifier`(s)	Check whether a selector or token sequence can identify a single port.
`is_in`(s, t)	Check whether all of the identifiers in one selector are comprised by another.
`is_selector`(s)	Check whether a string or sequence is a valid selector.
`is_selector_empty`(selector)	Check whether a string or sequence is an empty selector.
`is_selector_seq`(s)	Check whether a sequence is a valid selector.
`is_selector_str`(s)	Check whether a string is a valid selector.
`make_index`(selector[, names])	Create an index from the specified selector.
`make_index_two_concat`(sel_0, sel_1[, names])	Create an index from two selectors concatenated elementwise.
`make_index_two_prod`(sel_0, sel_1[, names])	Create an index from the product of two selectors.
`max_levels`(selector)	Return maximum number of token levels in selector.
`p_error`(p)	
`p_level`(p)	level : ASTERISK
`p_selector_comma_selector`(p)	selector : selector COMMA selector
`p_selector_dotplus_selector`(p)	selector : selector DOTPLUS selector
`p_selector_level`(p)	selector : level
`p_selector_paren_selector`(p)	selector : LPAREN selector RPAREN
`p_selector_plus_selector`(p)	selector : selector PLUS selector
`p_selector_selector_level`(p)	selector : selector level
`p_selector_selector_plus_level`(p)	selector : selector PLUS level
`pad_parsed`(selector[, pad_len, inplace])	Pad token lists in a parsed selector to some maximum length.
`pad_selector`(selector[, pad_len])	Expand and pad a selector with blank tokens.
`pad_tuple_list`(tuple_list, pad_len)	Pad a list of tuples with blank strings.
`parse`(selector[, pad_len])	Parse a selector string into tokens.
`select`(df, selector[, start, stop])	Select rows from DataFrame using a path-like selector.
`t_ASTERISK`(t)	/*
`t_COMMA`(t)	,
`t_DOTPLUS`(t)	.+
`t_INTEGER`(t)	/?d+
`t_INTEGER_SET`(t)	/?[(?:d+,?)+]
`t_INTERVAL`(t)	/?[d*:d*]
`t_LPAREN`(t)	(
`t_PLUS`(t)	+
`t_RPAREN`(t)	)
`t_STRING`(t)	/[^*/[]():,.d][^+*/[]():,.]*
`t_STRING_SET`(t)	/?[(?:[^+*/[]():,.d][^+*/[]():,.]*,?)+]
`t_error`(t)	
`to_identifier`(s)	Convert an expanded selector/token sequence into a single port identifier string.
`tokenize`(selector)	Tokenize a selector string.
`tokens_to_str`(tokens)	Convert expanded/parsed token sequence into a single selector string.

Attributes

`lexer`

Continued on next page

Table 1.27 – continued from previous page

parser
tokens

neurokernel.plsel.SelectorParser

class neurokernel.plsel.**SelectorParser**

This class implements a parser for path-like selectors that can be associated with elements in a sequential data structure such as a Pandas DataFrame; in the latter case, each level of the selector corresponds to a level of a Pandas MultiIndex. An index level may either be a denoted by a string label (e.g., 'foo') or a numerical index (e.g., 0, 1, 2); a selector level may additionally be a list of strings (e.g., '[foo,bar]') or integers (e.g., '[0,2,4]') or continuous intervals (e.g., '[0:5]'). The '*' symbol matches any value in a level, while a range with an open upper bound (e.g., '[5:]') will match all integers greater than or equal to the lower bound.

Examples of valid selectors include

Selector	Comments
/foo/bar	
/foo+/bar	equivalent to /foo/bar
/foo/[qux,bar]	
/foo/bar[0]	
/foo/bar/[0]	equivalent to /foo/bar[0]
/foo/bar/0	equivalent to /foo/bar[0]
/foo/bar[0,1]	
/foo/bar[0:5]	
/foo/*/baz	
/foo/*/baz[5]	
/foo/bar,/baz/qux	
(/foo,/bar)+/baz	equivalent to /foo/baz,/bar/baz
/[foo,bar].+/[0:2]	equivalent to /foo[0],/bar[1]

Notes

An empty string is deemed to be a valid selector.

Since there is no need to maintain multiple instances of the lexer/parser used to process path-like selectors, they are associated with the class rather than class instances; likewise, all of the class' methods are classmethods.

Numerical indices in selectors are assumed to be zero-based. Intervals do not include the end element (i.e., like numpy, not like Pandas).

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

Methods

p_error(p)	
p_level(p)	level : ASTERISK
p_selector_comma_selector(p)	selector : selector COMMA selector
p_selector_dotplus_selector(p)	selector : selector DOTPLUS selector
p_selector_level(p)	selector : level

Continued on next page

Table 1.28 – continued from previous page

`p_selector_paren_selector`(p)	selector : LPAREN selector RPAREN
`p_selector_plus_selector`(p)	selector : selector PLUS selector
`p_selector_selector_level`(p)	selector : selector level
`p_selector_selector_plus_level`(p)	selector : selector PLUS level
`pad_parsed`(selector[, pad_len, inplace])	Pad token lists in a parsed selector to some maximum length.
`parse`(selector[, pad_len])	Parse a selector string into tokens.
`t_ASTERISK`(t)	/*
`t_COMMA`(t)	,
`t_DOTPLUS`(t)	.+
`t_INTEGER`(t)	/?d+
`t_INTEGER_SET`(t)	/?[(?:d+,?)+]
`t_INTERVAL`(t)	/?[d*:d*]
`t_LPAREN`(t)	(
`t_PLUS`(t)	+
`t_RPAREN`(t)	)
`t_STRING`(t)	/[^*/[]():,.d][^+*/[]():,.]*
`t_STRING_SET`(t)	/?[(?:[^+*/[]():,.d][^+*/[]():,.]*,?)+]
`t_error`(t)	
`tokenize`(selector)	Tokenize a selector string.

Attributes

`lexer`
`parser`
`tokens`

GPU Port Mappers

`GPUPortMapper`	Maps a PyCUDA GPUArray to/from path-like port identifiers.

neurokernel.pm_gpu.GPUPortMapper

class `neurokernel.pm_gpu.`**`GPUPortMapper`**(*selector*, *data=None*, *portmap=None*, *make_copy=True*)

Maps a PyCUDA GPUArray to/from path-like port identifiers.

`__init__`(*selector*, *data=None*, *portmap=None*, *make_copy=True*)

Methods

`__init__`(selector[, data, portmap, make_copy])	
`copy`()	Return copy of this port mapper.
`equals`(other)	Check whether this mapper is equivalent to another mapper.

Continued on next page

Table 1.31 – continued from previous page

from_index(idx, data[, portmap])	
from_pm(pm)	Create a new port mapper instance given an existing instance.
get(selector)	
get_by_inds(inds)	Retrieve mapped data specified by integer index.
get_inds_nonzero()	
get_map(selector)	Retrieve integer indices associated with selector.
get_ports(f)	Select ports using a data selection function.
get_ports_as_inds(f)	Select integer indices corresponding to ports in map.
get_ports_nonzero()	
inds_to_ports(inds)	Convert list of integer indices to port identifiers.
ports_to_inds(selector)	Convert port selector to list of integer indices.
set(selector, data)	
set_by_inds(inds, data)	Set mapped data by integer indices.
set_by_inds_array(inds, data)	Set mapped data with array by integer indices.
set_by_inds_scalar(inds, data)	Set mapped data with scalar by integer indices.
set_map(selector, portmap)	Set mapped integer index associated with selector.

Attributes

data	Data associated with ports.
data_ctype	C type corresponding to type of data array.
dtype	Port mapper data type.
index	Port mapper index.

Python Port Mappers

BasePortMapper	Maps integer sequence to/from path-like port identifiers.
PortMapper	Maps a numpy array to/from path-like port identifiers.

neurokernel.pm.BasePortMapper

class neurokernel.pm.**BasePortMapper**(*selector*, *portmap=None*)

Maps integer sequence to/from path-like port identifiers.

Examples

```
>>> pm = BasePortMapper('/[a,b][0:2]')
>>> print pm.ports_to_inds('/b[0:2]')
array([2, 3])
>>> print pm.inds_to_ports([0, 1])
[('a', 0), ('a', 1)]
```

Parameters

- **selector** (*str, unicode, or sequence*) – Selector string (e.g., '/foo[0:2]') or sequence of token sequences (e.g., [['foo', (0, 2)]]) to map to *data*.

- **`portmap`** (*`sequence of int`*) – Integer indices to map to port identifiers. If no map is specified, it is assumed to be an array of consecutive integers from 0 through one less than the number of ports.

`index`
pandas.MultiIndex – Index of port identifiers.

`portmap`
pandas.Series – Map of port identifiers to integer indices.

Notes

The selectors may not contain any '*' or '[:]' characters. A single port identifier may be mapped to multiple integer indices, but not vice-versa.

`__init__`(*selector*, *portmap=None*)

Methods

`__init__`(selector[, portmap])	
`copy`()	Return copy of this port mapper.
`equals`(pm)	Check whether this mapper is equivalent to another mapper.
`from_index`(idx[, portmap])	Create port mapper from a Pandas index and a sequence of integer indices.
`from_pm`(pm)	Create a new port mapper instance given an existing instance.
`get_map`(selector)	Retrieve integer indices associated with selector.
`inds_to_ports`(inds)	Convert list of integer indices to port identifiers.
`ports_to_inds`(selector)	Convert port selector to list of integer indices.
`set_map`(selector, portmap)	Set mapped integer index associated with selector.

Attributes

`index`	Port mapper index.

neurokernel.pm.PortMapper

class `neurokernel.pm.`**`PortMapper`**(*selector*, *data=None*, *portmap=None*, *make_copy=True*)
Maps a numpy array to/from path-like port identifiers.

Examples

```
>>> data = np.array([1, 0, 3, 2, 5, 2])
>>> pm = PortMapper('/d[0:5]', data)
>>> print pm['/d[1]']
array([0])
>>> print pm['/d[2:4]']
array([3, 2])
```

Parameters

- **selector** (*str, unicode, or sequence*) – Selector string (e.g., '/foo[0:2]') or sequence of token sequences (e.g., [['foo', (0, 2)]]) to map to *data*.
- **data** (*numpy.ndarray*) – 1D data array to map to ports. If no data array is specified, port identifiers will still be mapped to their sequential indices but __getitem__() and __setitem__() will raise exceptions if invoked.
- **portmap** (*sequence of int*) – Integer indices to map to port identifiers. If no map is specified, it is assumed to be an array of consecutive integers from 0 through one less than the number of ports.
- **make_copy** (*bool*) – If True, map a copy of the specified data array to the specified port identifiers.

data
numpy.ndarray – Data that has been mapped to ports.

dtype
numpy.dtype – Type of mapped data.

index
pandas.MultiIndex – Index of port identifiers.

portmap
pandas.Series – Map of port identifiers to integer indices into *data*.

Notes

The selectors may not contain any '*' or '[:]' characters.

__init__(*selector, data=None, portmap=None, make_copy=True*)

Methods

`__init__`(selector[, data, portmap, make_copy])	
`copy`()	Return copy of this port mapper.
`equals`(other)	Check whether this mapper is equivalent to another mapper.
`from_index`(idx, data[, portmap])	
`from_pm`(pm)	Create a new port mapper instance given an existing instance.
`get`(selector)	Retrieve mapped data specified by given selector.
`get_by_inds`(inds)	Retrieve mapped data specified by integer index.
`get_inds_nonzero`()	Select indices of ports with nonzero data.
`get_map`(selector)	Retrieve integer indices associated with selector.
`get_ports`(f)	Select ports using a data selection function.
`get_ports_as_inds`(f)	Select integer indices corresponding to ports in map.
`get_ports_nonzero`()	Select ports with nonzero data.
`inds_to_ports`(inds)	Convert list of integer indices to port identifiers.
`ports_to_inds`(selector)	Convert port selector to list of integer indices.

Continued on next page

Table 1.36 – continued from previous page

set(selector, data)	Set mapped data specified by given selector.
set_by_inds(inds, data)	Set mapped data by integer indices.
set_map(selector, portmap)	Set mapped integer index associated with selector.

Attributes

data	Data associated with ports.
dtype	Port mapper data type.
index	Port mapper index.

XML Tools

graph_to_nml_module	Convert a module expressed as NetworkX graphs into Neurokernel NeuroML.
graph_to_nml_pattern	Convert a pattern expressed as a NetworkX graph into Neurokernel NeuroML.
load	Load a Neurokernel NeuroML document.
nml_pattern_to_graph	Convert a pattern expressed in Neurokernel NeuroML into a NetworkX graph.
nml_module_to_graph	Convert a module expressed in Neurokernel NeuroML into NetworkX graphs.
write	Write a Neurokernel NeuroML document to an XML file.

neurokernel.neuroml.utils.graph_to_nml_module

neurokernel.neuroml.utils.**graph_to_nml_module**(*g*, *i*, *id*)

Convert a module expressed as NetworkX graphs into Neurokernel NeuroML.

Parameters

- **g** (*networkx.DiGraph*) – Directed graph containing a module's neurons and synapses
- **i** (*networkx.Graph*) – Undirected graph containing a module's interface ports.
- **id** (str) – Module identifier.

Returns **module** – Module instance.

Return type neurokernel.neuroml.Module

neurokernel.neuroml.utils.graph_to_nml_pattern

neurokernel.neuroml.utils.**graph_to_nml_pattern**(*g*, *id*)

Convert a pattern expressed as a NetworkX graph into Neurokernel NeuroML.

Parameters

- **g** (*networkx.DiGraph*) – Directed graph containing the pattern's ports (nodes) and connections (edges).
- **id** (str) – Pattern identifier.

Returns **pattern** – pattern instance.

Return type neurokernel.neuroml.Pattern

neurokernel.neuroml.utils.load

`neurokernel.neuroml.utils.`**`load`**`(`*file*`)`

Load a Neurokernel NeuroML document.

Parameters **`file`** (`str or file`) – Input file name or handle.

Returns **nk_doc** – Neurokernel NeuroML document root.

Return type neurokernel.neuroml.NeurokernelDoc

neurokernel.neuroml.utils.nml_pattern_to_graph

`neurokernel.neuroml.utils.`**`nml_pattern_to_graph`**`(`*pattern*`)`

Convert a pattern expressed in Neurokernel NeuroML into a NetworkX graph.

Parameters **`pattern`** (*`neurokernel.neuroml.Pattern`*) – Pattern instance.

Returns **g** – Directed graph containing the pattern's ports (nodes) and connections (edges).

Return type networkx.DiGraph

neurokernel.neuroml.utils.nml_module_to_graph

`neurokernel.neuroml.utils.`**`nml_module_to_graph`**`(`*module*`)`

Convert a module expressed in Neurokernel NeuroML into NetworkX graphs.

Parameters **`module`** (*`neurokernel.neuroml.Module`*) – Module instance.

Returns

- **g** (*networkx.DiGraph*) – Directed graph containing a module's neurons and synapses
- **i** (*networkx.Graph*) – Undirected graph containing a module's interface ports.

neurokernel.neuroml.utils.write

`neurokernel.neuroml.utils.`**`write`**`(`*nk_doc*, *file*, *root_name='nk'*`)`

Write a Neurokernel NeuroML document to an XML file.

Parameters

- **`nk_doc`** (*`neurokernel.neuroml.NeurokernelDoc`*) – Neurokernel NeuroML document root object.
- **`file`** (`str or file`) – Output file name or handle.
- **`root_name`** (`str`) – Document root name.

Context Managers

ExceptionOnSignal	Raise a specific exception when the specified signal is detected.
IgnoreKeyboardInterrupt	Ignore keyboard interrupts.
IgnoreSignal	Ignore the specified signal.
OnKeyboardInterrupt	Respond to keyboard interrupt with specified handler.
TryExceptionOnSignal	Check for exception raised in response to specific signal.

neurokernel.ctx_managers.ExceptionOnSignal

`neurokernel.ctx_managers.`**`ExceptionOnSignal`**`(`**args*, ***kwds*`)`

Raise a specific exception when the specified signal is detected.

neurokernel.ctx_managers.IgnoreKeyboardInterrupt

`neurokernel.ctx_managers.`**`IgnoreKeyboardInterrupt`**`(`**args*, ***kwds*`)`

Ignore keyboard interrupts.

neurokernel.ctx_managers.IgnoreSignal

`neurokernel.ctx_managers.`**`IgnoreSignal`**`(`**args*, ***kwds*`)`

Ignore the specified signal.

neurokernel.ctx_managers.OnKeyboardInterrupt

`neurokernel.ctx_managers.`**`OnKeyboardInterrupt`**`(`**args*, ***kwds*`)`

Respond to keyboard interrupt with specified handler.

neurokernel.ctx_managers.TryExceptionOnSignal

`neurokernel.ctx_managers.`**`TryExceptionOnSignal`**`(`**args*, ***kwds*`)`

Check for exception raised in response to specific signal.

GPU Tools

bufint	Return buffer interface to GPU or numpy array.
set_by_inds	Set values in a GPUArray by index.
set_by_inds_from_inds	Set values in a GPUArray by index from indexed values in another GPUArray.
set_realloc	Transfer data into a GPUArray instance.

neurokernel.tools.gpu.bufint

`neurokernel.tools.gpu.`**`bufint`**`(`*a*`)`

Return buffer interface to GPU or numpy array.

Parameters **a** (*pycuda.gpuarray.GPUArray or numpy.ndarray*) – GPU or numpy array.

Returns **b** – Buffer interface to array. Returns None if *a* has a length of 0.

Return type buffer

neurokernel.tools.gpu.set_by_inds

neurokernel.tools.gpu.**set_by_inds**(*dest_gpu*, *ind*, *src_gpu*, *ind_which='dest'*)

Set values in a GPUArray by index.

Parameters

- **dest_gpu** (*pycuda.gpuarray.GPUArray*) – GPUArray instance to modify.
- **ind** (*pycuda.gpuarray.GPUArray or numpy.ndarray*) – 1D array of element indices to set. Must have an integer dtype.
- **src_gpu** (*pycuda.gpuarray.GPUArray*) – GPUArray instance from which to set values.
- **ind_which** (*str*) – If set to 'dest', set the elements in *dest_gpu* with indices *ind* to the successive values in *src_gpu*; the lengths of *ind* and *src_gpu* must be equal. If set to 'src', set the successive values in *dest_gpu* to the values in *src_gpu* with indices *ind*; the lengths of *ind* and *dest_gpu* must be equal.

Examples

```
>>> import pycuda.gpuarray as gpuarray
>>> import pycuda.autoinit
>>> import numpy as np
>>> from nk.tools.gpu import set_by_inds
>>> dest_gpu = gpuarray.to_gpu(np.arange(5, dtype=np.float32))
>>> ind = gpuarray.to_gpu(np.array([0, 2, 4]))
>>> src_gpu = gpuarray.to_gpu(np.array([1, 1, 1], dtype=np.float32))
>>> set_by_inds(dest_gpu, ind, src_gpu, 'dest')
>>> np.allclose(dest_gpu.get(), np.array([1, 1, 1, 3, 1], dtype=np.float32))
True
>>> dest_gpu = gpuarray.to_gpu(np.zeros(3, dtype=np.float32))
>>> ind = gpuarray.to_gpu(np.array([0, 2, 4]))
>>> src_gpu = gpuarray.to_gpu(np.arange(5, dtype=np.float32))
>>> set_by_inds(dest_gpu, ind, src_gpu, 'src')
>>> np.allclose(dest_gpu.get(), np.array([0, 2, 4], dtype=np.float32))
True
```

Notes

Only supports 1D index arrays.

May not be efficient for certain index patterns because of lack of inability to coalesce memory operations.

neurokernel.tools.gpu.set_by_inds_from_inds

neurokernel.tools.gpu.**set_by_inds_from_inds**(*dest_gpu*, *ind_dest*, *src_gpu*, *ind_src*)

Set values in a GPUArray by index from indexed values in another GPUArray.

Parameters

- **dest_gpu** (*pycuda.gpuarray.GPUArray*) – GPUArray instance to modify.
- **ind_dest** (*pycuda.gpuarray.GPUArray or numpy.ndarray*) – 1D array of element indices in *dest_gpu* to set. Must have an integer dtype.
- **src_gpu** (*pycuda.gpuarray.GPUArray*) – GPUArray instance from which to set values.
- **ind_src** (*pycuda.gpuarray.GPUArray or numpy.ndarray*) – 1D array of element indices in *src_gpu* to copy. Must have an integer dtype and be the same length as *ind_dest*.

Examples

```
>>> import pycuda.gpuarray as gpuarray
>>> import pycuda.autoinit
>>> import numpy as np
>>> from nk.tools.gpu import set_by_inds_from_inds
>>> dest_gpu = gpuarray.to_gpu(np.zeros(5, dtype=np.float32))
>>> ind_dest = gpuarray.to_gpu(np.array([0, 2, 4]))
>>> src_gpu =  gpuarray.to_gpu(np.arange(5, 10, dtype=np.float32))
>>> ind_src =  gpuarray.to_gpu(np.array([2, 3, 4]))
>>> gpu.set_by_inds_from_inds(dest_gpu, ind_dest, src_gpu, ind_src)
>>> assert np.allclose(dest_gpu.get(), np.array([7, 0, 8, 0, 9], dtype=np.
↪float32))
True
```

neurokernel.tools.gpu.set_realloc

`neurokernel.tools.gpu.`**`set_realloc`**(*x_gpu*, *data*)

Transfer data into a GPUArray instance.

Copies the contents of a numpy array into a GPUArray instance. If the array has a different type or dimensions than the instance, the GPU memory used by the instance is reallocated and the instance updated appropriately.

Parameters

- **x_gpu** (*pycuda.gpuarray.GPUArray*) – GPUArray instance to modify.
- **data** (*numpy.ndarray*) – Array of data to transfer to the GPU.

Examples

```
>>> import pycuda.gpuarray as gpuarray
>>> import pycuda.autoinit
>>> import numpy as np
>>> import misc
>>> x = np.asarray(np.random.rand(5), np.float32)
>>> x_gpu = gpuarray.to_gpu(x)
>>> x = np.asarray(np.random.rand(10, 1), np.float64)
>>> set_realloc(x_gpu, x)
>>> np.allclose(x, x_gpu.get())
True
```

MPI Tools

MPIOutput	

neurokernel.tools.mpi.MPIOutput

`neurokernel.tools.mpi.`**`MPIOutput`** **= <Mock spec='str' id='139755941507728'>**

ZeroMQ Tools

get_random_port	Return available random ZeroMQ port.
is_poll_in	Check for incoming data on a socket using a poller.
ZMQOutput	

neurokernel.tools.zmq.get_random_port

`neurokernel.tools.zmq.`**`get_random_port`**(*min_port=49152*, *max_port=65536*, *max_tries=100*)

Return available random ZeroMQ port.

neurokernel.tools.zmq.is_poll_in

`neurokernel.tools.zmq.`**`is_poll_in`**(*sock*, *poller*, *timeout=100*)

Check for incoming data on a socket using a poller.

neurokernel.tools.zmq.ZMQOutput

`neurokernel.tools.zmq.`**`ZMQOutput`** **= <Mock spec='str' id='139755941469520'>**

Visualization Tools

imdisp	Display the specified image file using matplotlib.
show_pydot	Display a networkx graph using pydot.
show_pygraphviz	Display a networkx graph using pygraphviz.

neurokernel.tools.plot.imdisp

`neurokernel.tools.plot.`**`imdisp`**(*f*)

Display the specified image file using matplotlib.

neurokernel.tools.plot.show_pydot

`neurokernel.tools.plot.`**`show_pydot`**(*g*)

Display a networkx graph using pydot.

neurokernel.tools.plot.show_pygraphviz

neurokernel.tools.plot.**show_pygraphviz**(*g*, *prog='dot'*, *graph_attr={}*, *node_attr={}*, *edge_attr={}*)

Display a networkx graph using pygraphviz.

Parameters

- **prog** (str) – Executable for generating the image.
- **graph_attr** (*dict*) – Global graph display attributes.
- **node_attr** (*dict*) – Global node display attributes.
- **edge_attr** (*dict*) – Global edge display attributes.

Logging Tools

log_exception	Log the specified exception data using twiggy.
set_excepthook	Set the exception hook to use the specified logger.
setup_logger	Setup a twiggy logger.

neurokernel.tools.logging.log_exception

neurokernel.tools.logging.**log_exception**(*type*, *value*, *tb*, *logger=<Mock name='mock.log' id='139755941507920'>*, *multiline=False*)

Log the specified exception data using twiggy.

Parameters

- **tb** (*value,*) – Parameters expected by traceback.print_exception.
- **logger** (*twiggy.logger.Logger*) – Logger to use. twiggy.log is assumed by default.
- **multiline** (*bool*) – If True, print exception using multiple log lines.

neurokernel.tools.logging.set_excepthook

neurokernel.tools.logging.**set_excepthook**(*logger*, *multiline=False*)

Set the exception hook to use the specified logger.

Parameters

- **logger** (*twiggy.logger.Logger*) – Configured logger.
- **multiline** (*bool*) – If True, log exception messages on multiple lines.

neurokernel.tools.logging.setup_logger

`neurokernel.tools.logging.`**`setup_logger`**(*name='', level=<Mock name='mock.levels.DEBUG' id='139755941508240'>, fmt=<Mock name='mock.formats.line_format' id='139755941508496'>, fmt_name=<built-in method format of str object>, screen=None, file_name=None, mpi_comm=None, zmq_addr=None, log_exceptions=True, multiline=False*)

Setup a twiggy logger.

Parameters

- **`name`** (`str`) – Logger name.
- **`level`** (*`twiggy.levels.LogLevel`*) – Logging level.
- **`fmt`** (*`twiggy.formats.LineFormat`*) – Logging formatter class instance.
- **`fmt_name`** (*`function`*) – Function with one parameter that formats the message name.
- **`screen`** (*`bool`*) – Create output stream handler to the screen if True.
- **`file_name`** (`str`) – Create output handler to specified file.
- **`mpi_comm`** (*`mpi4py.MPI.Intracomm`*) – If not None, use MPI I/O with the specified communicator for output file handler. Ignored if the *file_name* parameter is not specified.
- **`zmq_addr`** (`str`) – ZeroMQ socket address.
- **`log_exceptions`** (*`bool`*) – If True, exception messages are written to the logger.
- **`multiline`** (*`bool`*) – If True, log exception messages on multiple lines.

Returns

- **logger** (*twiggy.logger.Logger*) – Configured logger.
- *Bug*
- —
- *To use the ZeroMQ output class with multiprocessing, it must be added*
- *as an emitter within each process.*

Other

`LoggerMixin`(name[, log_on])	Mixin that provides a per-instance logger that can be turned off.

`catch_exception`	Catch and report exceptions when executing a function.
`rand_bin_matrix`	Generate a rectangular binary matrix with randomly distributed nonzero entries.

neurokernel.tools.misc.catch_exception

neurokernel.tools.misc.**catch_exception**(*func*, *disp*, **args*, ***kwargs*)

Catch and report exceptions when executing a function.

If an exception occurs while executing the specified function, the exception's message and the line number where it occurred (in the innermost traceback frame) are displayed.

Examples

```
>>> import sys
>>> def f(x): x/0
>>> catch_exception(f, sys.stdout.write, 1)
f: integer division or modulo by zero (...:1)
```

Parameters

- **func** (*function*) – Function to execute.
- **disp** (*function*) – Function to use to display exception message.
- **args** (*list*) – Function arguments.
- **kwargs** (*dict*) – Named function arguments.

neurokernel.tools.misc.rand_bin_matrix

neurokernel.tools.misc.**rand_bin_matrix**(*sh*, *N*, *dtype=<Mock name='mock.double' id='139755941509712'>*)

Generate a rectangular binary matrix with randomly distributed nonzero entries.

Examples

```
>>> m = rand_bin_matrix((2, 3), 3)
>>> set(m.flatten()) == set([0, 1])
True
```

Parameters

- **sh** (*tuple*) – Shape of generated matrix.
- **N** (*int*) – Number of entries to set to 1.
- **dtype** (dtype) – Generated matrix data type.

Authors & Acknowledgements

The Neurokernel Project was begun in July 2011 by Aurel A. Lazar at Columbia University's Department of Electrical Engineering after extensive discussions held during a research seminar on Massively Parallel Neural Computation. The Neurokernel Development Team currently comprises the following Bionet Group researchers:

- Lev Givon

- Konstantinos Psychas
- Nikul H. Ukani
- Chung-Heng Yeh
- Yiyin Zhou

Past contributors who have participated in the project include

- Daniel S. Chevitarese

The Neurokernel logo is based upon the logo of the FlyJunkies web site, used with permission for non-profit use by the site maintainers Gavin Davis and Fraser Perry.

Licenses

Neurokernel

Copyright (c) 2012-2015, Neurokernel Development Team All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of the copyright holders nor the names of any contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

libNeuroML

Copyright (c) 2012, libNeuroML authors and contributors All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the names of the copyright holders nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

CHAPTER 2

Index

- genindex

Index

Symbols

B

C

D

E

G

H

I

L

M

N

O

P

R

S

T

W

Z