Parallel NEURON idioms:

Information exchange during setup

Random numbers

Debugging

Michael Hines

CodeJam 2014

Information exchange during setup

Results must be independent of
Number of processors
Distribution of cells

A process is often interested in all the objects with
a particular property.

But it generally does not know where the objects are.

And the process that owns the object does not
know who is interested in it.

There is not enough memory in any one process to hold
a map of which ranks hold which objects.

Example: MPI_ISend/Recv spike exchange

Cells do not know which ranks are interested in its spikes.

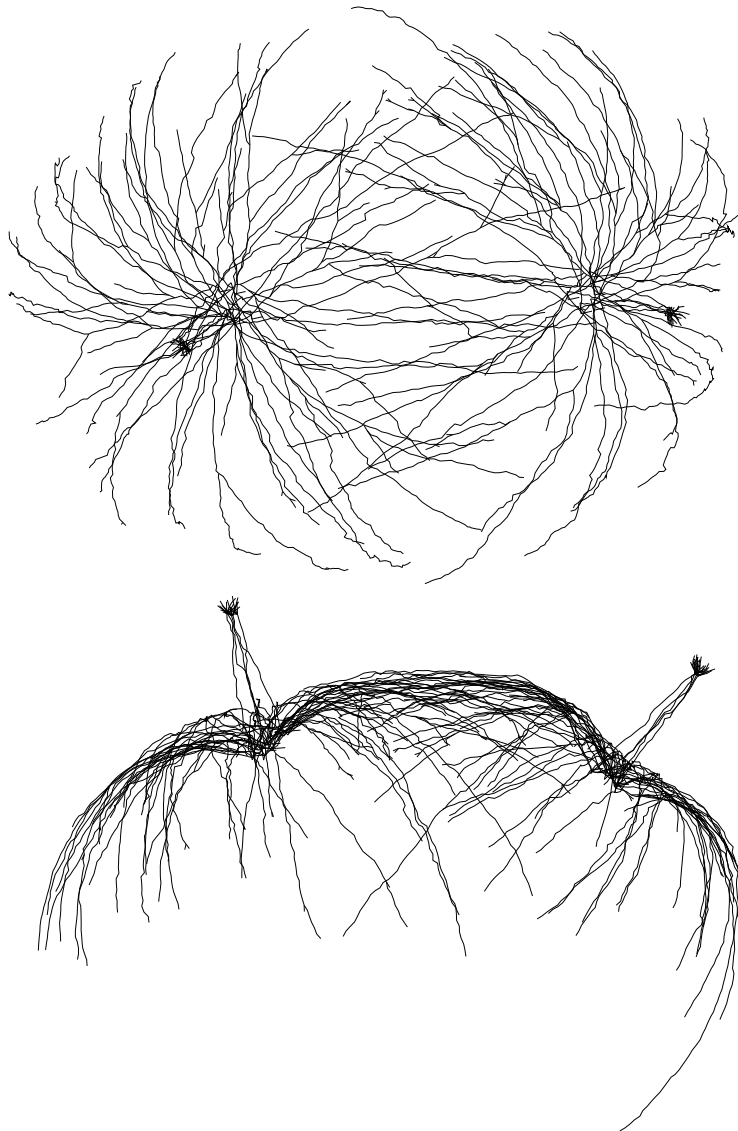## Example: Source/Target connectivity
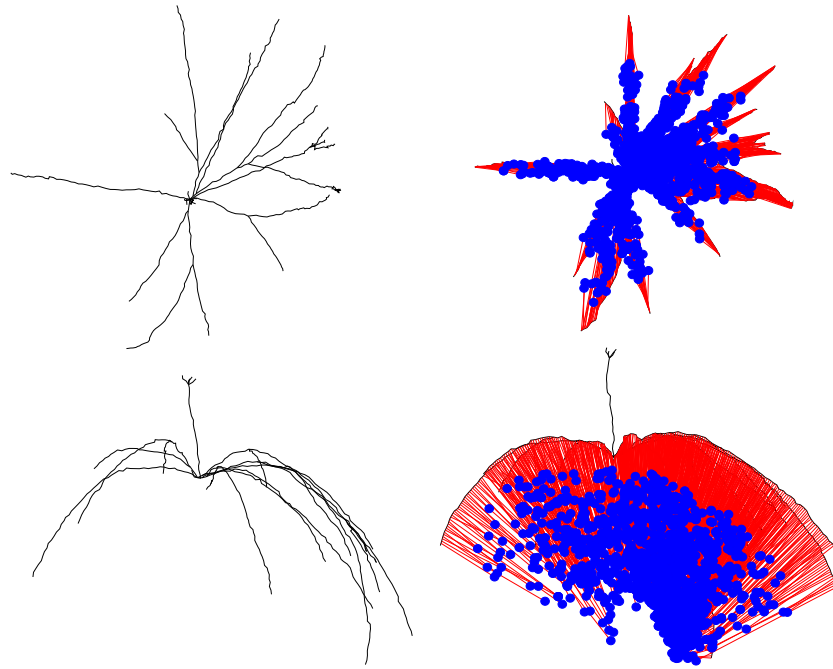
Reciprocal synapse connection description.
   (mitral_gid, mdend_index, xm, granule_gid, gdend_index, xg, ...)

Construct a mitral => all the tuples with that mitral_gid.

Granules don't know enough for construction of the tuples.

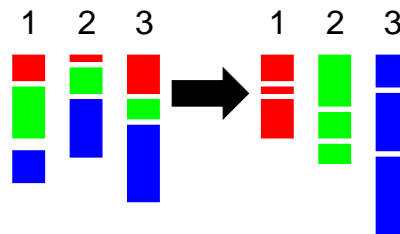Construct a granule => gather all the tuples with that granule_gid.

Basic exchange:

dest = ParallelContext.py_alltoall(src)

src and dest are a list of nhost pickleable objects.

src[j] on the ith rank will be copied to dest[i] on the jth rank.

Likely identical to mpi4py.MPI comm.alltoall(src, dest).

Basic exchange:

dest = ParallelContext.py_alltoall(src)

src and dest are a list of nhost pickleable objects.

src[j] on the ith rank will be copied to dest[i] on the jth rank.

Likely identical to mpi4py.MPI comm.alltoall(src, dest).

Essentially a wrapper for:

```
MPI_Alltoallv(s, scnt, sdispl, MPI_CHAR,
                 r, rcnt, rdispl, MPI_CHAR, comm);
```

along with a preliminary
MPI_all2all(scnt, 1, MPI_INT, rcnt, 1, MPI_INT, comm);
in order to calculate rcnt and rdispl.

But:

No one knows who holds what.

No room for anyone to have a global map.

Solution: A rendezvous rank function:

rank = rendezvous(property)

usually

rank = gid % nhost

1) Everyone sends the keys they own to the rendezvous rank.

2) Everyone sends the keys they want to the rendezvous rank.

3) The rendezvous rank sends back to the owners,
   which ranks want which keys.

4) The owners send the objects to the ranks that want them.

Usually simplification is possible:

If the objects are small.

1) Everyone sends the keys   and objects
   they own to the rendezvous rank.

2) Everyone sends the keys they want to the rendezvous rank.

3) The rendezvous rank sends the objects to the ranks that
   want them.

Usually simplification is possible:

If rendezvous(property) is known to be the source rank for
   all the keys (a–priori or by verifying with an all_reduce).

1) Everyone sends the keys they want to the owner ranks.

2) The owners send the objects to the ranks that want them.

Usually simplification is possible:

If rendezvous(property) is known to be the destination rank
   for all the keys (a–priori or by verifying with an all_reduce).

1) The owners send the objects to the ranks that want them.

What about RANDOM?

Results must be independent of
- Number of processors
- Distribution of cells

Associate a random stream with a cell.
- Reproducible
- Independent
- Restartable

Use cryptographic transformation of several integers.
- run number
- stream number (cell gid)
- stream pick index

Use cryptographic transformation of several integers.
- run number
- stream number (cell gid)
- stream pick index

Had been using MCellRan4

but only two integers to define x(n1, n2)

Thanks! to Eilif Muller for suggesting:

Parallel Random Numbers: As Easy as 1, 2, 3
Salmon et al. SC11 (2011)
D. E. Shaw Research, New York, NY 10036, USA

We introduce several counter–based PRNGs: some based on
cryptographic standards (AES, Threefish) and some completely
new (Philox). All our PRNGs pass rigorous statistical tests
(including TestU01's BigCrush) and produce at least 2^64
unique parallel streams of random numbers, each with period
2^128 or more.

http://www.deshawresearch.com/resources_random123.html

Use cryptographic transformation of several integers.

    run number

    stream number (cell gid)

    stream pick index

Random123:

    Eight integers define $x(n1, ..., n8)$

        But we use 5 (three for the stream number).

        Philox variant

    Good performance

	10 million picks	
ACG	0.329s	
MLCG	0.681s	
MCellRan4	0.150s	
numpy.random.rand(n)	0.233s	(Mersenne Twister)
Random123	0.201s	

```
from neuron import h
r = h.Random()
r.Random123(1,2,3)
r.negexp(1)

n=100000000
dx = .01
y = h.Vector(n).setrand(r)
y = y.histogram(0,5,dx).rotate(−1,0)
x = y.c().indgen(dx/2,dx)
g = h.Graph()
y.line(g, x)
```
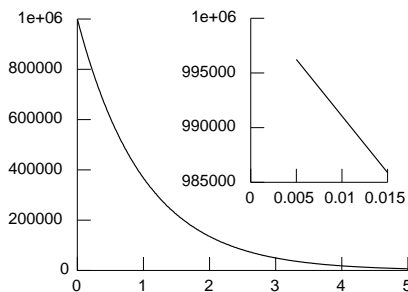
nrnran123.h (abridged)

all generator instances share the global index
extern void nrnran123_set_globalindex(uint32_t gix);

extern nrnran123_State*
    nrnran123_newstream(uint32_t id1, uint32_t id2);

extern uint32_t nrnran123_ipick(nrnran123_State*);
  uniform 0 to 2^32−1

extern double nrnran123_dblpick(nrnran123_State*);
  uniform open interval (0,1)
  minimum value is 2.3283064e−10
  max value is 1−min

```
extern double nrnran123_negexp(nrnran123_State*);
  mean 1.0
  min value is 2.3283064e−10
  max is 22.18071



            −log(1/2^32)  22.18071
            −log(2/2^32)  21.487563

            −log(10/2^32) 19.878125
            −log(11/2^32) 19.782815

            exp(−5)*2^32  28939262
      −log(28939262/2^32)  5.0000000001
      −log(28939263/2^32)  4.99999996




extern double nrnran123_negexp(nrnran123_State*);
  mean 1.0
  min value is 2.3283064e−10
  max is 22.18071


stateless (though the global index is still used)
extern nrnran123_array4x32
   nrnran123_iran(uint32_t seq, uint32_t id1, uint32_t id2);
```