

```

1  """
2
3  A minimum working example of a NEURON gap junction over MPI
4
5  Author: Tom Close
6  Date: 8/1/2013
7  Email: tclose@oist.jp
8  """
9
10 import os
11 import argparse
12 import numpy as np
13 # This is a hack I use on our cluster, to get MPI initialised=True. There is pr
obably something
14 # wrong with our setup but I can't be bothered trying to work out what it is at
this point. All
15 # suggestions welcome :)
16 try:
17     from mpi4py import MPI #@UnresolvedImport @UnusedImport
18 except:
19     print "mpi4py was not found, MPI will remain disabled if MPI initialized==F
alse on startup"
20     from neuron import h, load_mechanisms
21     # Not sure this is necessary, or whether I can just use h.finitialize instead o
f h.stdinit
22     h.load_file('stdrun.hoc')
23
24     # The GID used for the gap junction connection. NB: this number is completely i
ndependent from the
25     # GID's used for NEURON sections.
26     GID_FOR_VAR = 0
27
28     # Arguments to the script
29     parser = argparse.ArgumentParser(description=__doc__)
30     parser.add_argument('--plot', action='store_true', help="Plot the data instead
of saving it")
31     parser.add_argument('--output_dir', type=str, default=os.getcwd(),
32                         help="The directory to save the output files into")
33     parser.add_argument('--gap_mechanism_dir', type=str, default=os.getcwd(),
34                         help="The directory to load the gap mechanism from")
35     args = parser.parse_args()
36
37     # Load gap mechanism from another directory if required
38     if args.gap_mechanism_dir is not os.getcwd():
39         load_mechanisms(args.gap_mechanism_dir)
40     # Get the parallel context and related parameters
41     pc = h.ParallelContext()
42     num_processes = int(pc.nhost())
43     mpi_rank = int(pc.id())
44     print "On process {} of {}".format(mpi_rank+1, num_processes)
45
46     print "Creating test network..."
47     # The pre-synaptic cell is created on the first node and the post-synaptic cell
on the last node
48     # (NB: which will obviously be the same if there is only one node)
49     if mpi_rank == 0:
50         print "Creating pre-synaptic cell on process {}".format(mpi_rank)
51         # Create the pre-synaptic cell
52         pre_cell = h.Section()
53         pre_cell.insert('pas')
54         # Connect the voltage of the pre-synaptic cell to the gap junction on the p
ost-synaptic cell
55         pc.source_var(pre_cell(0.5)._ref_v, GID_FOR_VAR)
56         # Stimulate the first cell to make it obvious whether gap junction is worki
ng
57         stim = h.IClamp(pre_cell(0.5))
58         stim.delay = 50
59         stim.amp = 10
60         stim.dur = 100
61         # Record Voltage of pre-synaptic cell
62         pre_v = h.Vector()
63         pre_v.record(pre_cell(0.5)._ref_v)
64         if mpi_rank == (num_processes - 1):
65             print "Creating post-synaptic cell on process {}".format(mpi_rank)
66             # Create the post-synaptic cell
67             post_cell = h.Section()
68             post_cell.insert('pas')
69             # Insert gap junction
70             gap_junction = h.gap(0.5, sec=post_cell)
71             gap_junction.g = 1.0
72             # Connect gap junction to pre-synaptic cell
73             pc.target_var(gap_junction._ref_vgap, GID_FOR_VAR)
74             # Record Voltage of post-synaptic cell
75             post_v = h.Vector()
76             post_v.record(post_cell(0.5)._ref_v)
77             # Finalise construction of parallel context
78             pc.setup_transfer()
79             # Record time
80             rec_t = h.Vector()
81             rec_t.record(h._ref_t)
82             print "Finished network construction on process {}".format(mpi_rank)
83
84             # Run simulation
85             print "Setting maxstep on process {}".format(mpi_rank)
86             pc.set_maxstep(10)
87             print "Finitiaialise on process {}".format(mpi_rank)
88             #h.finitiaialize(-60)
89             h.stdinit()
90             print "Solving on process {}".format(mpi_rank)
91             pc.psolve(100)
92             print "Running worker on process {}".format(mpi_rank)
93             pc.runworker()
94             print "Completing parallel context on process {}".format(mpi_rank)
95             pc.done()
96             print "Finished run on process {}".format(mpi_rank)
97
98             # Convert recorded data into Numpy arrays
99             t_array = np.array(rec_t)
100             if mpi_rank == 0:
101                 pre_v_array = np.array(pre_v)
102             if mpi_rank == (num_processes - 1):
103                 post_v_array = np.array(post_v)
104
105             # Either plot the recorded values
106             if args.plot and num_processes == 1:
107                 print "Plotting..."
108                 import matplotlib.pyplot as plt
109                 if mpi_rank == 0:
110                     pre_fig = plt.figure()
111                     plt.plot(t_array, pre_v_array)
112                     plt.title("Pre-synaptic cell voltage")
113                     plt.xlabel("Time (ms)")
114                     plt.ylabel("Voltage (mV)")
115                 if mpi_rank == (num_processes - 1):
116                     pre_fig = plt.figure()
117                     plt.plot(t_array, post_v_array)
118                     plt.title("Post-synaptic cell voltage")
119                     plt.xlabel("Time (ms)")
120                     plt.ylabel("Voltage (mV)")
121                 plt.show()
122             else:
123                 # Save data
124                 print "Saving data..."
125                 if mpi_rank == 0:
126                     np.savetxt(os.path.join(args.output_dir, "pre_v.dat"),
127                                np.transpose(np.vstack((t_array, pre_v_array))))
128                 if mpi_rank == (num_processes - 1):
129                     np.savetxt(os.path.join(args.output_dir, "post_v.dat"),
130                                np.transpose(np.vstack((t_array, post_v_array))))
131             print "Done."
132         end

```