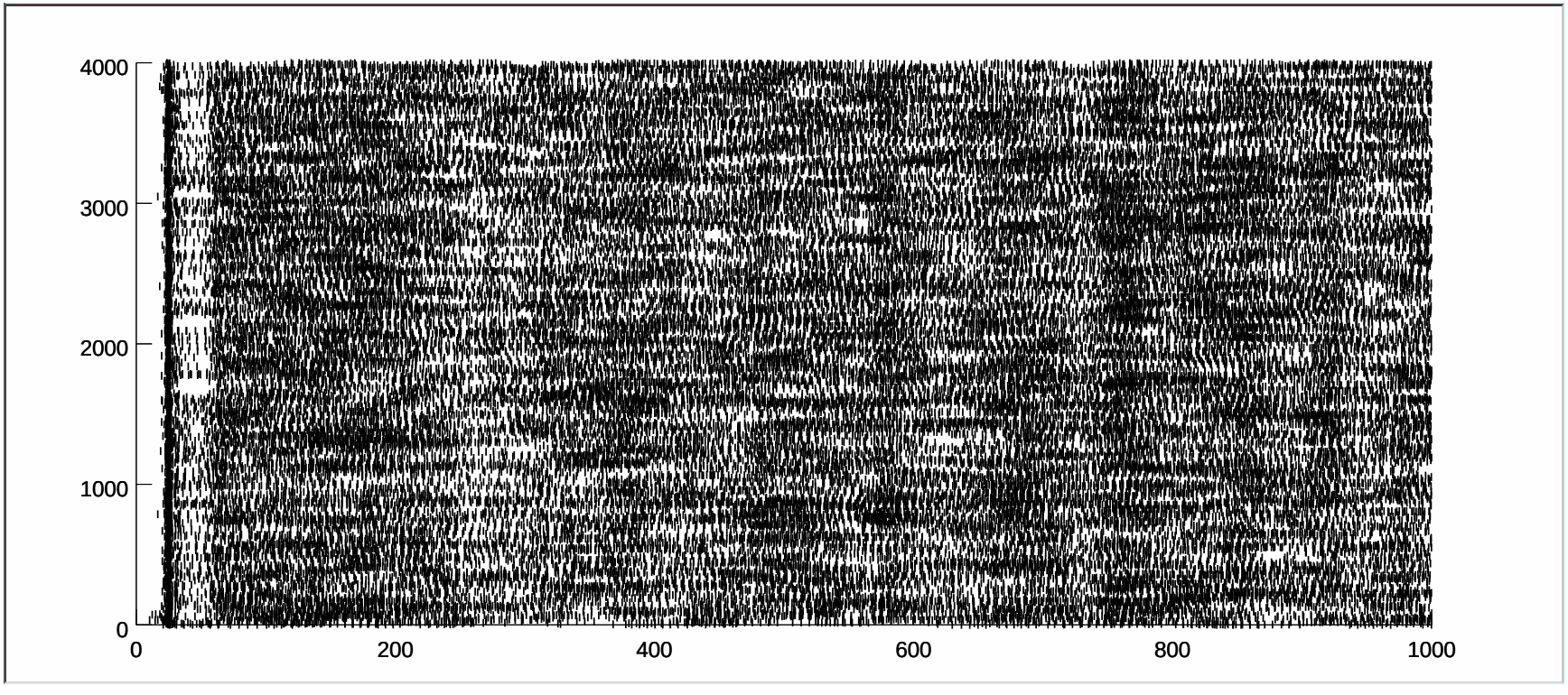
Tutorial: Parallel Network Simulations with NEURON

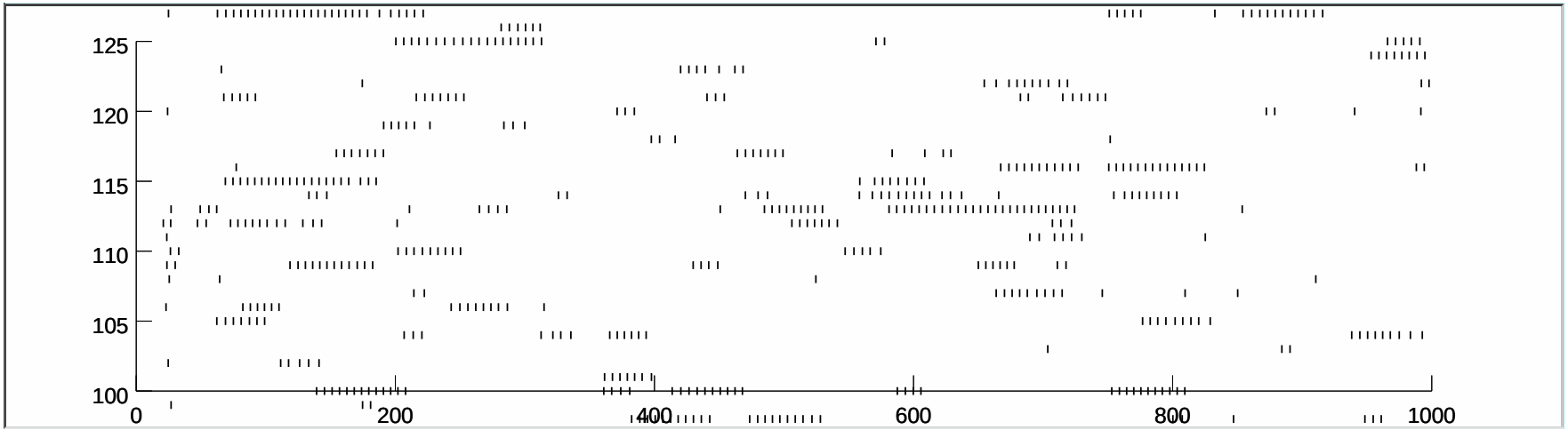
- Installation
  - MPI
  - NEURON
  - Launch
- Network specification
  - GID distribution
  - Create cells
  - Connections
  - Random
    - cell parameters, connections, weight, delay
    - spike input stimulus
- Running
  - performance measurement
  - efficiency
- An example
  - Vogels and Abbott (2005)

- Installation
  - MPI
  - NEURON
  - Launch
- Network specification
  - GID distribution
  - Create cells
  - Connections
  - Random
    - cell parameters, connections, weight, delay
    - spike input stimulus
- Running
  - performance measurement
  - efficiency
- An example
  - Vogels and Abbott (2005)

Same result, no matter
how many machines
how cells distributed

reproducible
statistically
independent


4000
3000
2000
1000
0
0
200
400
600
800
1000


125
120
115
110
105
100
0
200
400
600
800
1000

```
$ mpirun ... −np 10  ...
$ cat out.dat
18.1    2410
20.1    530
21.2    3880
21.4    2670
...
999.8   550
999.9   1190
999.9   1810
18.2    75
19.6    1915
19.6    3805
...

$ cat ~/bin/sortspike
#!/bin/sh
sort −k 1n,1n −k 2n,2n $1 > $2

$ sortspike out.dat out.10
$ mpirun ... −np 5  ...
$ sortspike out.dat out.5
$ diff out.5 out.10
```

Beowulf cluster or multiprocessor personal machine. (Linux)
See MPI under ParallelContext

Can you login to any node without a password?

```
ssh ‘hostname‘
```

Is MPI installed?

```
which mpicc
which mpic++ # or mpicxx, mpiCC, ...
```

Beowulf cluster or multiprocessor personal machine. (Linux)
See MPI under ParallelContext

Can you login to any node without a password? If not...

```
ssh-keygen -t rsa
cd $HOME/.ssh; cat id_rsa.pub >> authorized_keys
```

Is MPI installed? If not...

```
Get the latest stable version from:
http://www.mcs.anl.gov/research/projects/mpich2/
tar xzf mpich2-1.2.1p1.tar.gz
cd mpich2-1.2.1p1
./configure --prefix=$HOME/mpich2
make
make install
export PATH=$HOME/mpich2/bin:$PATH
cd $HOME;echo 'secretword=nrnmpi'>.mpd.conf;chmod 600 .mpd.conf
```

Test it.

```
mpd&
mpiexec -n 2 echo 'hello' # should print twice
mpdallexit
```

Beowulf cluster or multiprocessor personal machine. (Linux)
Installing NEURON (without InterViews)

Assume sources are in $HOME/neuron/nrn

```
cd $HOME/neuron
mkdir nrnmpi ; cd nrnmpi


../nrn/configure --prefix=‘pwd‘ --with-paranrn --with-nrnpython \
    --without-x
make install
```

Supercomputers need special configuration...

SDSC IBM DataStar
http://www.sdsc.edu/user_services/datastar/
272 (8−way) P655+ and 7 (32−way) P690 compute nodes.
could not get autoconf−5.9 installed so extracted a nrn−x.tar.gz file over the cvs sources. /usr/bin/oslevel does not work so, in nrn, run
grerep /usr/bin/oslevel /usr/local/bin/oslevel config.guess configure

```
sh ../nrn/configure  −−prefix=‘pwd‘ \
  ’−−srcdir=../nrn’ −−with−nmodl−only −−without−x CC=gcc CXX=g++
make; make install

sh ../nrn/configure  −−prefix=‘pwd‘  ’−−srcdir=../nrn’ \
  ’−−without−x’ ’−−without−memacs’ ’−−without−readline’ \
  ’−−disable−shared’ ’−−with−paranrn’ ’−−without−nmodl’ \
  CC=mpcc_r CXX=mpCC_r MPICC=mpcc_r MPICXX=mpCC_r \
  CFLAGS=−q64 CXXFLAGS=−q64 linux_nrnmech=no AR="ar −X64" \
  always_call_mpi_init=yes java_dlopen=no \
  −−host=powerpc−ibm−aix5.2.0.0
make; make install
```

Install: MSWindows

Vista or Windows 7 install

http://www.neuron.yale.edu/ftp/neuron/versions/alpha/nrn−7.2.alpha−426−setup.exe

Test by opening an rxvt window and typing
mpd&
mpiexec −n 2 /cygdrive/c/nrn72/bin/echo ’hello’
mpdallexit
Test that NEURON works with mpi by creating a test0.hoc program:

```
objref pc
pc = new ParallelContext()
{printf("I am %d of %d\n", pc.id, pc.nhost)}
{pc.runworker() pc.done() quit() }
```

and launch with
mpiexec −n 4 /cygdrive/c/nrn72/bin/nrniv −mpi test0.hoc
(try leaving out the −mpi arg)

Launch (Linux)

cd neuron/nrn/src/parallel
mpiexec −n 3 nrniv −mpi test0.hoc

numprocs=3
NEURON −− Version 7.2 (428:986821b56b98) 2010−03−07
Duke, Yale, and the BlueBrain Project −− Copyright 1984−2008
See http://www.neuron.yale.edu/credits.html

I am 0 of 3
I am 1 of 3
I am 2 of 3

nrn/src/parallel/test0.hoc

```
objref pc
pc = new ParallelContext()
{printf("I am %d of %d\n", pc.id, pc.nhost)}
{pc.runworker()}
{pc.done()}
quit()
```

common/net.hoc

```
proc create_cells() { local i, gid  localobj cell, nc
    for pcitr(&i, &gid) {
        cell = newcell(gid)
        pnm.register_cell(gid, cell)
    }
}
```

Create

coba/init.hoc

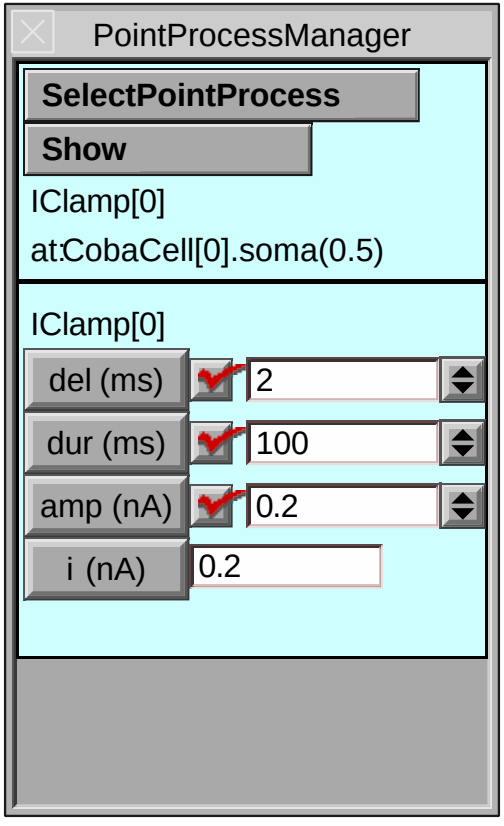
```
obfunc newcell() {
    return new CobaCell()
}
```

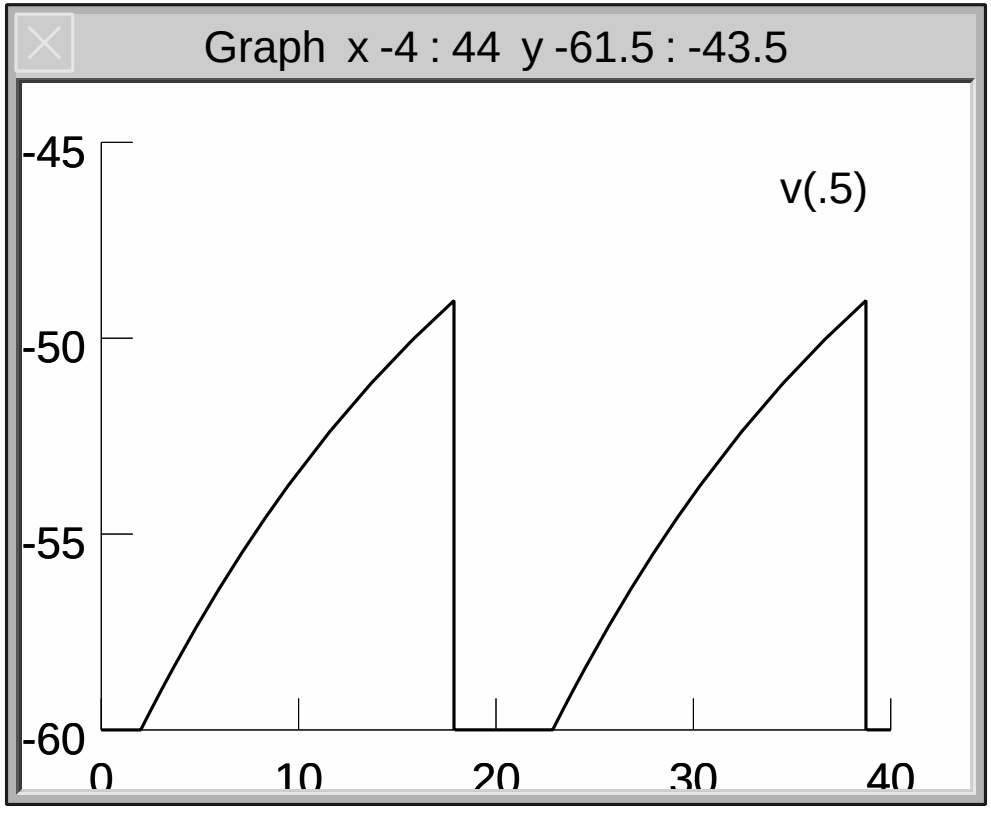
coba/cobacell.hoc

```
begintemplate CobaCell
public ...connect2target...
proc init() {
    soma spkout = new SpikeOut(.5)
}
proc connect2target() { //$o1 target point process, $o2 returned NetCon
    $o2 = new NetCon(spkout, $o1)
}
endtemplate CobaCell
```

cuba/cubacell.hoc

```
obfunc newcell() {localobj cell
    cell =  new IF3()
    cell.taum = 20
    cell.taue = 10
    cell.taui = 20
    return cell
}
```


PointProcessManager
SelectPointProcess
Show
IClamp[0]
at:CobaCell[0].soma(0.5)
IClamp[0]
del (ms)
2
dur (ms)
100
amp (nA)
0.2
i (nA)
0.2


Graph x -4 : 44 y -61.5 : -43.5
-45
-50
-55
-60
v(.5)
0
10
20
30
40

nrn/lib/hoc/netparmpi.hoc

```
begintemplate ParallelNetManager
...
proc register_cell() { localobj nc
        if (!pc.gid_exists($1)) { pc.set_gid2node($1, pc.id) }
        // all existing cells must have an associated gid which
        // is stored in the cell's PreSyn. The nc below will be
        // unreffed but the PreSyn will continue
        // in existence and from the gid we will quickly be able
        // to find the PreSyn and from that the Cell
        // we force the cell to be an outputcell due to the danger of
        // user error
        cells.append($o2)
        if (hoc_sf_.is_artificial($o2)) {
                nc = new NetCon($o2, nil)
        }else{
                $o2.connect2target(nil, nc)
        }
        pc.cell($1, nc, 1)
}
```

```
proc connect_cells() { local i, j, gid, r, d   localobj cell, u, rs
    d = DELAY // in the paper it is 0
    mcell_ran4_init(connect_random_low_start_)
    u = new Vector(ncell) // for sampling without replacement
    for pcitr(&i, &gid) { // target cell
        u.fill(0)
        cell = pnm.cells.object(i)
        rs = ranlist.object(i)   rs.start()
        rs.r.discunif(0, N_E−1)
        j=0 while(j < C_E) { // Excitatory sources
            r = rs.repick()
            if (r != gid) if (u.x[r] == 0) {
                pnm.nc_append(r, gid, AMPA_INDEX, AMPA_GMAX, d)
                u.x[r] = 1     j += 1
        }   }
        rs.r.discunif(N_E, ncell−1)
        j=0 while(j < C_I) { // Inhibitory sources
            r = rs.repick()
            if (r != gid) if (u.x[r] == 0) {
                pnm.nc_append(r, gid, GABA_INDEX, GABA_GMAX, d)
                u.x[r] = 1      j += 1
}   }   }   }
```

pick exactly C_E and C_I unique non−self random connections

for each target. Assume many more sources than target connections.

Connect

Stimulate

```
proc create_stim() {local i, gid  localobj stim, cell, nc, rs
    mcell_ran4_init($1)
    stimlist = new List()
    ncstimlist = new List()
    // first N_STIM (excitatory) cells stimulated
    for pcitr(&i, &gid) {
        if (gid >= N_STIM) { break }
        cell = pc.gid2cell(gid)
        rs = ranlist.object(i)
        stim = new NetStim()
        stim.interval = STIM_INTERVAL
        stim.number = 1000 // but will shut off after STOPSTIM
        stim.noise = 1        stim.start = 0
        // use the gid specific random generator so random streams
        // independent of where and how many stims there are
        stim.noiseFromRandom(rs.r)
        rs.r.negexp(1)          rs.start()
        nc = new NetCon(stim, cell.synlist.object(0))
        nc.delay = 1    nc.weight = $2
        ncstimlist.append(nc)
        stimlist.append(stim)
    }
```

```
    stim = new NetStim() // will turn off all the others
    stim.number = 1
    stim.start = STOPSTIM
    for i=0, stimlist.count−1 {
        nc = new NetCon(stim, stimlist.object(i))
        nc.delay = 1    nc.weight = −1
        ncstimlist.append(nc)
    }
    stimlist.append(stim)
}
```