

## Parallel Simulations with NEURON

Installation (pages 2-6)

Tutorial notes: conceptual (pages 7-12)

Vogels and Abbott COBA example (pages 13-19)

Debugging (pages 20-21)

## Parallel Examples

Parallel network simulations with NEURON (Migliore et al 2006)

<http://senselab.med.yale.edu/senselab/modeldb/ShowModel.asp?model=64229>

Translating network models to parallel hardware in NEURON (Hines, Carnevale 2008)

<http://senselab.med.yale.edu/modeldb/ShowModel.asp?model=96444>

see: <http://www.neuron.yale.edu/neuron/nrnpubs>

Simulation of networks of spiking neurons: A review of tools and strategies (Brette et al 2007).

<http://senselab.med.yale.edu/modeldb/ShowModel.asp?model=83319>

These are reminiscent of Vogels and Abbott (2005) J. Neurosci. 25, "Signal Propagation and Logic Gating in Networks of Integrate-and-Fire Neurons.

<http://senselab.med.yale.edu/modeldb/ShowModel.asp?model=83319>

FORTTRAN to NEURON translation of Traub et al (2005)

Single-column thalamocortical network model exhibiting gamma oscillations sleep spindles, and epileptogenic bursts. J Neurophysiol. 2005 Apr;93(4):1829-30.

<http://senselab.med.yale.edu/senselab/modeldb/ShowModel.asp?model=45539>

But see the most recent parallelized version in the mercurial repository:

<http://www.neuron.yale.edu/hg/z/models/nrntraub/>

A sequence of transformations of the Dentate Gyrus network model (Santhakumar et al 2005)

<http://senselab.med.yale.edu/modeldb/ShowModel.asp?model=51781>

into an MPI parallel (as well as threadsafe) version is in the the mercurial repository:

<http://www.neuron.yale.edu/hg/z/models/santhakumar2005>

## Further Info

[http://www.neuron.yale.edu/neuron/static/docs/help/quick\\_reference.html](http://www.neuron.yale.edu/neuron/static/docs/help/quick_reference.html)

ParallelContext ParallelNetManager

ParallelNetwork ParallelTransfer MPI

## Installation

### MSWindows

For XP install:

<http://www.neuron.yale.edu/ftp/neuron/versions/alpha/nrn-7.2.alpha-422-setup.exe>

For Vista or Windows7 install:

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

(The important difference is that to work around an mpiexec problem 422 uses mpich2-1.0.7 whereas 426 uses mpich2-1.2.1p1)

Test by opening an rxvt window and typing (you can leave out the #comments):

```
mpd& # start the multiprocessor daemon
```

```
mpdtrace # should indicate you have one machine
```

```
mpiexec -n 2 /cygdrive/c/nrn72/bin/echo 'hello' # should print help twice
```

```
mpdallexit # when you want to kill the daemon use this.
```

Test that NEURON works with mpi by creating the hoc program test0.hoc

```
objref pc
```

```
pc = new ParallelContext()
```

```
{ printf("I am %d of %d\n", pc.id, pc.nhost) }
```

```
{pc.runworker() }
```

```
{pc.done() }
```

```
quit()
```

and launching

```
mpiexec -n 4 /cygdrive/c/nrn72/bin/nrniv -mpi test0.hoc
```

(try leaving out the '-mpi' arg)

Beowulf cluster or multiprocessor personal machine (Linux or Mac OS X).

See MPI under ParallelContext

Can you login to any node without a password?

```
ssh `hostname`
```

If not...

```
ssh-keygen -t rsa
```

```
cd $HOME/.ssh; cat id_rsa.pub >> authorized_keys
```

Is MPI installed?

```
which mpicc
which mpic++ # or mpicxx, mpiCC
If not...
Go to http://www.mcs.anl.gov/research/projects/mpich2/
and from the downloads section get the MPICH2-1.2.1p1 sources.
cd $HOME
tar xzf mpich2-1.2.1p1.tar.gz
cd mpich2-1.2.1p1
./configure --prefix=$HOME/mpich2
make
make install
export PATH=$HOME/mpich2/bin:$PATH
```

Test it. (about the same as the mswindows case above)

```
mpdboot
mpdtrace
mpiexec -n 2 echo 'hello'
mpdallexit
```

Now build NEURON from the sources.

Build NEURON using the --with-paranrn configure argument  
Use the latest alpha version nrn-xxx.tar.gz file from  
<http://www.neuron.yale.edu/ftp/neuron/versions/alpha/>  
...better yet, clone the mercurial repository... See:  
<http://www.neuron.yale.edu/neuron/download/getdevel>

```
cd $HOME
mkdir neuron
cd neuron
curl -O http://www.neuron.yale.edu/ftp/neuron/versions/alpha/iv-17.tar.gz
curl -O http://www.neuron.yale.edu/ftp/neuron/versions/alpha/nrn-7.2.alpha-428.tar.gz
tar xzf iv*.tar.gz
tar xzf nrn*.gz
mv iv-17 iv
mv nrn-72 nrn
```

Here is where Linux and Mac OS X potentially diverge.

## LINUX

```
cd iv
```

```
./configure --prefix=`pwd`
```

```
make
```

```
make install
```

```
cd ..
```

```
mkdir nrnmpi
```

```
cd nrnmpi
```

```
../nrn/configure --prefix=`pwd` --with-paranrn --with-nrnpython
```

```
make -j 2 install
```

## Mac OS X

Same as Linux if using X11 (required for an x86\_64 Snow Leopard for a 64 bit version; there is no carbon for 64 bit). If wish to use carbon then:

```
cd iv
```

```
./configure --prefix=`pwd` --enable-carbon CFLAGS='-arch i386' CXXFLAGS='-arch i386'
```

```
make
```

```
make install
```

```
cd ..
```

```
mkdir nrnmpi
```

```
cd nrnmpi
```

```
../nrn/configure --prefix=`pwd` --enable-carbon --with-paranrn --with-nrnpython \  
CFLAGS='-arch i386' CXXFLAGS='-arch i386'
```

(note that MPI would have to be build using these CFLAGS as well)

Supercomputers are usually special and often require cross-compiling.

## EPFL IBM BlueGene/P

16384 cores 750MHz powerpc each with 512MB

```
../nrn/configure --prefix=`pwd` --without-x --with-nmodl-only
```

```
make; make install
```

```
../nrn/configure --prefix=`pwd` --enable-bluegeneP --with-paranrn --with-nrnpython \  
--host=powerpc64-suse-linux
```

```
make; make install; make after_install
```

## Launch

mpdboot

cd nrn/src/parallel

mpiexec -n 4 \$HOME/neuron/nrnmpi/x86\_64/bin/nrniv -mpi test0.hoc

```
$ mpiexec -n 4 $HOME/neuron/nrnmpi/x86_64/bin/nrniv -mpi  
test0.hoc
```

```
numprocs=4
```

```
NEURON -- VERSION 7.2 (426:7b4f020b29e8) 2010-03-12
```

```
Duke, Yale, and the BlueBrain Project -- Copyright 1984-2008
```

```
See http://www.neuron.yale.edu/credits.html
```

```
I am 1 of 4
```

```
I am 0 of 4
```

```
I am 2 of 4
```

```
I am 3 of 4
```

```
[hines@hines490 parallel]$
```

On a Beowulf cluster...

```
[hines@hines490 ~]$ ssh hines@colossus.cs.yale.edu
Last login: Wed Mar 17 10:07:51 2010 from
ip98-179-188-162.ri.ri.cox.net
psh compute uptime
[hines@master hines]$ ssh node12
psh compute uptime
[hines@node12 hines]$ cat mpd.hosts
node12
node13
node14
node15
node16
node17
[hines@node12 hines]$ mpdboot -n 6
[hines@node12 hines]$ mpdtrace
node12
node13
node15
node16
node14
node17
[hines@node12 hines]$ cd neuron/nrn/src/parallel
[hines@node12 parallel]$ mpiexec -n 12 `which nrniv` -mpi
test0.hoc
numprocs=12
NEURON -- VERSION 7.2 (426+:7b4f020b29e8+) 2010-03-12
Duke, Yale, and the BlueBrain Project -- Copyright 1984-2008
See http://www.neuron.yale.edu/credits.html

I am 0 of 12
I am 6 of 12
I am 2 of 12
I am 8 of 12
I am 4 of 12
I am 3 of 12
I am 10 of 12
I am 9 of 12
I am 1 of 12
I am 7 of 12
I am 5 of 12
I am 11 of 12
[hines@node12 parallel]$
```

## Tutorial notes: conceptual (pages 7-11)

ParallelContext ParallelNetwork ParallelNetManager

same result no matter how many machines or how cells distributed

even with random connections

even with random stimulation

Distribute cells on machines.

Target centric.

Looking to use

`pnm.nc_append(srcgid, targetgid, synid, w, d)`

on machine where targetgid exists.

### GID distribution (conceptual)

Round robin

```
iterator pcitr() {local i, gid
    for (i=0 gid=0; gid < ncell; i += 1 gid += pc.nhost) {
        $&1 = i $&2 = gid
        iterator_statement
    }
}
```

General gidvec (maybe distribution generated by load balance program)

```
iterator pcitr() { local i, gid
    for i=0, gidvec.size-1 {
        $&1 = i $&2 = gidvec.x[i]
        iterator_statement
    }
}
```

GID distribution (instantiate)

```
for pcitr(&i, &gid) {
    pc.setgid2node(gid, pc.id)
}
```

## Cell creation

```
for pcitr(&i, &gid) {  
    // somehow figure out what class and parameters  
    // to use based on (i and pc.id) or gid.  
    cell = ...  
    pnm.register_cell(gid, cell)  
}
```

real cells should be in templates and have connect2target method  
artificial (spiking) cells can be bare.

if you know a gid you can:

- 1) know if the cell is on this machine  
    if (pc.gid\_exists(gid)) { ... }
- 2) get a reference to the cell  
    cell = pc.gid2cell(gid)

It has not so far been essential that a cell contain a field that specifies the gid. But the iterator has proven to be very useful.

```
for pcitr(&i, &gid) {  
    cell = pnm.cells.object(i)  
    rs = ranlist.object(i)  
    ... other things in parallel lists indexed by i ...  
}
```

even more useful than pc.gid2cell because the cell is often just a lightweight ARTIFICIAL\_CELL.

```
for gid=0, ncell-1 if (pc.gid_exists(gid)) {  
    cell = pc.gid2cell(gid)  
    ... now what? Is there useful administrative  
        info in the cell? ...  
}
```



## Network Connections

Real cells need a synlist in their template. For ARTIFICIAL\_CELL, synid = -1

Best setup time when connectivity computed using

```
for pcitr(&i, &targetgid) {  
    for ... srcgid ... {  
        // srcgid, synid, w, d may be random  
        pnm.nc_append(srcgid, targetgid, synid, w, d)  
    }  
}
```

## running

```
proc prun() {  
    pc.set_maxstep(10)  
    runtime=startsw()  
    stdinit()  
    pc.psolve(tstop)  
    runtime = startsw() - runtime  
}
```

serialize output with

```
proc foo() {local i  
    pc.barrier()  
    if (pc.id == 0) { print "header info" }  
    for i=0, pc.nhost-1 {  
        if (i == pc.id) {  
            print "stuff on rank i"  
        }  
        pc.barrier()  
    }  
}
```

A pattern for repeatable random simulations

A separate random stream for each cell using MCellRan4

```
random_stream_offset_ = 1000
```

```
begintemplate RandomStream
```

```
public r, repick, start, stream
```

```
external random_stream_offset_
```

```
objref r
```

```
proc init() {
```

```
    stream = $1
```

```
    r = new Random()
```

```
    start()
```

```
}
```

```
func start() {
```

```
    return r.MCellRan4(stream*random_stream_offset_ + 1)
```

```
}
```

```
func repick() {
```

```
    return r.repick()
```

```
}
```

```
endtemplate RandomStream
```

## efficiency

Setup:

use pcitr as the outer loop.

Run:

load balance (time to compute each subnet for maxstep time on each machine)

if rate limited by spike exchange

and using fixed step method so spikes on fixed boundaries then

pc.spike\_compress(nspike, gid\_compress)

(nspike>0 and gid\_compress=1 means 2 bytes per spike instead of at least 12)

"optimal" choice of nspike aided by perusal of spike exchange histogram statistics.

if rate limited by the event queue

due to heavy use of self events, set second arg to 1 of

cnode.queue\_mode(use\_fixed\_step\_binqueue, use\_self\_queue)

and if you don't mind events being on fixed dt boundaries

set that first arg to 1

performance statistics

Always at least measure:

// first thing in init.hoc

```
setuptime = startsw()
```

//just before calling prun()

```
setuptime = startsw()
```

```
proc prun() {
```

```
    runtime=startsw()
```

```
    waittime = pc.wait_time
```

```
    stdinit()
```

```
    pc.psolve(tstop)
```

```
    waittime = pc.wait_time - waittime
```

```
    runtime = startsw() - runtime
```

```
}
```

And often useful to get more detailed statistics

perfrun.hoc from most projects prints to stdout: eg. traub model

nrnmpi\_init(): numprocs=250 myid=0

...

SetupTime: 28.83

RunTime: 52.04

Maximum integration interval: 0.05

histogram of #spikes vs #exchanges

0	40
1	708
2	1006
3	220
4	26
5	1

end of histogram

...

setup	run	avgspkx	avgcomp	avgx2q	avgvxfr
28.83	52.04	1.367	47.77	0.0682	2.808

	id	spkx	id	comp	id	x2q	id	vxfr
min	2	0.8482	210	45.47	104	0.06623	14	1.269
max	212	1.92	40	49.72	0	0.07246	108	4.882

A good idiom for getting summary information about performance that avoids the bulletin board system is to use `pc.allreduce`. E.g. To find the average, max, and min computation time and communication time and print from host 0.

```
w_avg = pc.allreduce(1, waittime)/pc.nhost
w_max = pc.allreduce(2, waittime)
w_min = pc.allreduce(3, waittime)
comp_avg = pc.allreduce(1, pc.step_time)/pc.nhost
...
if (pc.id == 0) { printf("computation average=%g max=%g min=%g\n", \
    comp_avg, comp_max, comp_min )
```

If `comp_max` is close to `comp_avg`, then load balance is good.  
If not, can use `loadbal.hoc` to decide how to distribute cells on processors.

## Vogels & Abbott COBA example

in alain2006 directory:

networks

coba

cobacell.hoc	14
init.hoc	13
spikeout.mod	13

cobahh

hhcell.hoc
hhchan.ses
init.hoc

common

init.hoc	15
net.hoc	16
netstim.hoc	17
perfrun.hoc	18
ranstream.hoc	18
spkplt.hoc	19

cuba

cubacell.hoc
if3.mod
init.hoc

cubadv

cubadvcell.hoc
init.hoc
intfirecur.mod

## coba/init.hoc

```
{load_file("nrngui.hoc")}
{load_file("cobacell.hoc")}
{load_file("../common/init.hoc")}

obfunc newcell() {
    return new CobaCell()
}

create_net()
create_stim(run_random_low_start_, AMPA_GMAX)
finish_setup()

// parallel run to tstop
prun()
if (pc.id == 0) {print "RunTime: ", runtime}
{pc.runworker()}
collect_results()
{pc.done()}
output_results()
quit()
```

## coba/spikeout.mod

```
NEURON {
    POINT_PROCESS SpikeOut
    GLOBAL thresh, refrac, vrefrac, grefrac
    NONSPECIFIC_CURRENT i
}
PARAMETER {
    thresh = 1 (millivolt)
    refrac = 5 (ms)
    vrefrac = 0 (millivolt)
    grefrac = 100 (microsiemens) :clamp to vrefrac
}
ASSIGNED { i (nanoamp)    v (millivolt)    g (microsiemens) }
INITIAL {
    net_send(0, 3)
    g = 0
}
BREAKPOINT { i = g*(v - vrefrac) }
NET_RECEIVE(w) {
    if (flag == 1) {
        net_event(t)
        net_send(refrac, 2)
        v = vrefrac
        g = grefrac
    } else if (flag == 2) {
        g = 0
    } else if (flag == 3) {
        WATCH (v > thresh) 1
    }
}
```

portions of cobra/cobacell.hoc (modified from NetGUI generated hoc)

```
...
AMPA_INDEX = 0 /* synlist synapse indices
GABA_INDEX = 1 /*

thresh_SpikeOut = -50    // (mV)
refrac_SpikeOut = 5      // (ms)
vrefrac_SpikeOut = -60   // (mV) reset potential
grefrac_SpikeOut = 100   // (uS) clamped at reset

begintemplate CobraCell
public is_art
public init, topol, basic_shape, subsets, geom, biophys, geom_nseg,...
public synlist, x, y, z, position, connect2target, spkout
public soma
public all
objref synlist, spkout
create soma

proc init() {
    topol()
    subsets()
    geom()
    biophys()
    geom_nseg()
    synlist = new List()
    synapses()
    soma spkout = new SpikeOut(.5)
    x = y = z = 0 // only change via position
}
...
proc biophys() {
    forsec all {
        cm = 1
        insert pas
        g_pas = 5e-05
        e_pas = -60
    }
}
...
proc connect2target() { //$o1 target point process, $o2 returned NetCon
    $o2 = new NetCon(spkout, $o1)
}
objref syn_
proc synapses() {
    /* E0 */ soma syn_ = new ExpSyn(0.5) synlist.append(syn_)
    syn_.tau = 10
    /* I1 */ soma syn_ = new ExpSyn(0.5) synlist.append(syn_)
    syn_.tau = 20
    syn_.e = -80
}
func is_art() { return 0 }

endtemplate CobraCell
```

## common/init.hoc

```
setuptime = startsw()
{load_file("nrngui.hoc")}
{load_file("netparmpi.hoc")}
{load_file("ranstream.hoc")}
objref pnm, pc, ranlist

ncell = 4000
ranlist = new List()
random_stream_offset_ = 500 // adjacent streams will be correlated by this
offset
connect_random_low_start_ = 1
run_random_low_start_ = 2
pnm = new ParallelNetManager(ncell)
pc = pnm.pc
iterator pcitr() {local i1, i2
    // round robin style distribution of cells on cpus
    i1 = 0
    for (i2=pc.id; i2 < ncell; i2 += pc.nhost) {
        $&1 = i1
        $&2 = i2
        iterator_statement
        i1 += 1
    }
}
{load_file("net.hoc")} // create the model
{load_file("netstim.hoc")} // stimulate

// control parameters
tstop = 1000 //5000 // (ms)
dt = 0.1 // (ms) time step
steps_per_ms = 1/dt
v_init = -60
celsius = 36

// performance and statistics measurement to stdout
{load_file("perfrun.hoc")}
proc finish_setup() {
    want_all_spikes()
    mkhist(100)

    setuptime = startsw() - setuptime
    if (pc.id == 0) {print "SetupTime: ", setuptime}
}
proc collect_results() {
    pnm.prstat(1)
    getstat()
    pnm.gatherspikes()
    prhist()
    print_spike_stat_info()
}
proc output_results() {
    perf2file()
    spike2file()
}
```



## common/net.hoc

```

proc netparam() {
N_I = int(ncell/5.0)          // Number of inhibitory cells
N_E = ncell - N_I           // Number of excitatory cells
CONNECTIVITY = 0.02          // Connection probability
C_I = int(N_I*CONNECTIVITY)   // nb inh synapses per neuron
C_E = int(N_E*CONNECTIVITY)   // nb exc synapses per neuron
AMPA_GMAX      = 0.006        // (uS)
GABA_GMAX      = 0.067        // (uS)
DELAY          = 0            // (ms)
}
netparam()
proc create_cells() { local i, gid localobj cell, nc
    for pcitr(&i, &gid) {
        cell = newcell(gid)
        pnm.register_cell(gid, cell)
        ranlist.append(new RandomStream(gid)) // notice it is ith random
    }
}
proc connect_cells() { local i, j, gid, r, d localobj cell, u, rs
    // pick exactly C_E and C_I unique non-self random connections
    // for each target. Assume many more sources than target connections.
    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
            }
        }
    }
}
proc create_net() {
    create_cells()
    if (pc.id == 0) printf("%d...%d\n", pnm.ncell, pnm.cells.count)
    connect_cells()
    if (pc.id == 0) printf("%d connections...\n", pnm.nclist.count)
}

```

## common/netstim.hoc

```
// random external input for first 50 ms
// do not consider part of net so keep out of the pnm data structures

// Stimulation parameters

N_STIM      = ncell/50      // number of neurons stimulated
STOPSTIM    = 50            // duration of stimulation (ms)
NSYN_STIM   = 20            // nb of stim (exc) synapses per neuron
STIM_INTERVAL = 70          // mean interval between stims (ms)

objref svec, stvec
svec = new Vector()
stvec = new Vector()
objref stimlist, ncstimlist
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()
        if (hoc_sf_.is_artificial(cell)) {
            nc = new NetCon(stim, cell)
        }else{
            nc = new NetCon(stim, cell.synlist.object(0))
        }
        nc.delay = 1    nc.weight = $2
        nc.record(stvec, svec, ncstimlist.count)
        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)
}
```

## common/ranstream.hoc

```
random_stream_offset_ = 1000

begintemplate RandomStream
public r, repick, start, stream
external random_stream_offset_
objref r
proc init() {
    stream = $1
    r = new Random()
    start()
}
func start() {
    return r.MCellRan4(stream*random_stream_offset_ + 1)
}
func repick() {
    return r.repick()
}
endtemplate RandomStream
```

## portions of common/perfrun.hoc

```
proc want_all_spikes() { local i, gid
    for pcitr(&i, &gid) {
        pnm.spike_record(gid)
    }
}
...
objref tdat_
tdat_ = new Vector(3)
proc prun() {
    pnm.pc.set_maxstep(10)
    runtime=startsw()
    tdat_.x[0] = pnm.pc.wait_time
    stdinit()
    pnm.psolve(tstop)
    tdat_.x[0] = pnm.pc.wait_time - tdat_.x[0]
    runtime = startsw() - runtime
    tdat_.x[1] = pnm.pc.step_time
    tdat_.x[2] = pnm.pc.send_time
}
```

## common/spkplt.hoc

```
load_file("nrngui.hoc")

objref g
g = new Graph()
proc spkplt() {localobj x, y
    clipboard_retrieve($s1)
    printf("read %d spikes\n", hoc_obj_[1].size)
    x = hoc_obj_[1]
    y = hoc_obj_[0]
    g.size(x.min, x.max, y.min, y.max)
    y.mark(g, x, "|", 5, $2, 1)
}
spkplt("out.dat", 1)
```

SAVE the network spike pattern into a (spiketime gid) file, out.dat.

## Debugging

What to do when

```
sortspike out.dat out.np
```

```
diff out.1 out.np
```

says the files are not identical

What is the spiketime and gid of the first difference. Focus on that gid.

Is the problem in the spike input to the cell or the cell parameters/initialization.

Can run on a serial machine that computes exactly the same thing as though it were a particular rank on an nhost machine.

```
pnm.myid = rank  
pnm.nhost = nhost
```

Use PatternStim

```
objref pattern_, tvec_, idvec_  
pattern_ = new PatternStim()  
proc pattern() {  
    clipboard_retrieve("out.spk")  
    tvec_ = hoc_obj_[1].c  
    idvec_ = hoc_obj_[0].c  
    pattern_.play(tvec_, idvec_)  
}
```

Print the input spike information  
for the gid (see manuscript).

Print all parameters, states, etc for that gid.  
<http://www.neuron.yale.edu/ftp/hines/tucson/prcellstate.hoc>

```
prun(time_just_before_first_difference)
load_file("prcellstate.hoc")
proc pcs() {local gid
    gid = $1
    prcellstate(i, gid) // outfile: cs.i.id.nhost
    pc.runworker() pc.done() quit()
}
//pcs()
```

do for earlier times til the files are the same  
and see what variable first goes bad.