Parallelizing Network Simulations

Robert A. McDougal

Yale School of Medicine

11 November 2016

Many slides adapted from slides by Michael L. Hines.

Why use parallel computation?

Two reasons:

- "Faster" run-time simulations.
- Support for large models that do not fit on one machine.

What are the downsides?

Parallel models introduce:

- Greater programming complexity.
- New kinds of bugs.

You have to decide if the time spent parallelizing your model can be recovered.

Note that the 16384 core EPFL IBM BlueGene/P can theoretically do as many calculations in 1 hour at 850 MHz as a 3 GHz desktop computer can do in 6 months.

Three main classes of parallel problems

Parameter sweeps

Running the same (typically fast) simulation 1000s of times with different parameters is an example of an *embarrassingly parallel* problem. NEURON supports this natively with bulletin boards; Calin-Jageman and Katz (2006) developed a screen saver solution.

Distributing networks across processors

Cells can communicate by

- logical spike events with significant axonal, synaptic delay.
- postsynaptic conductance depending continuously on presynaptic voltage.
- gap junctions.

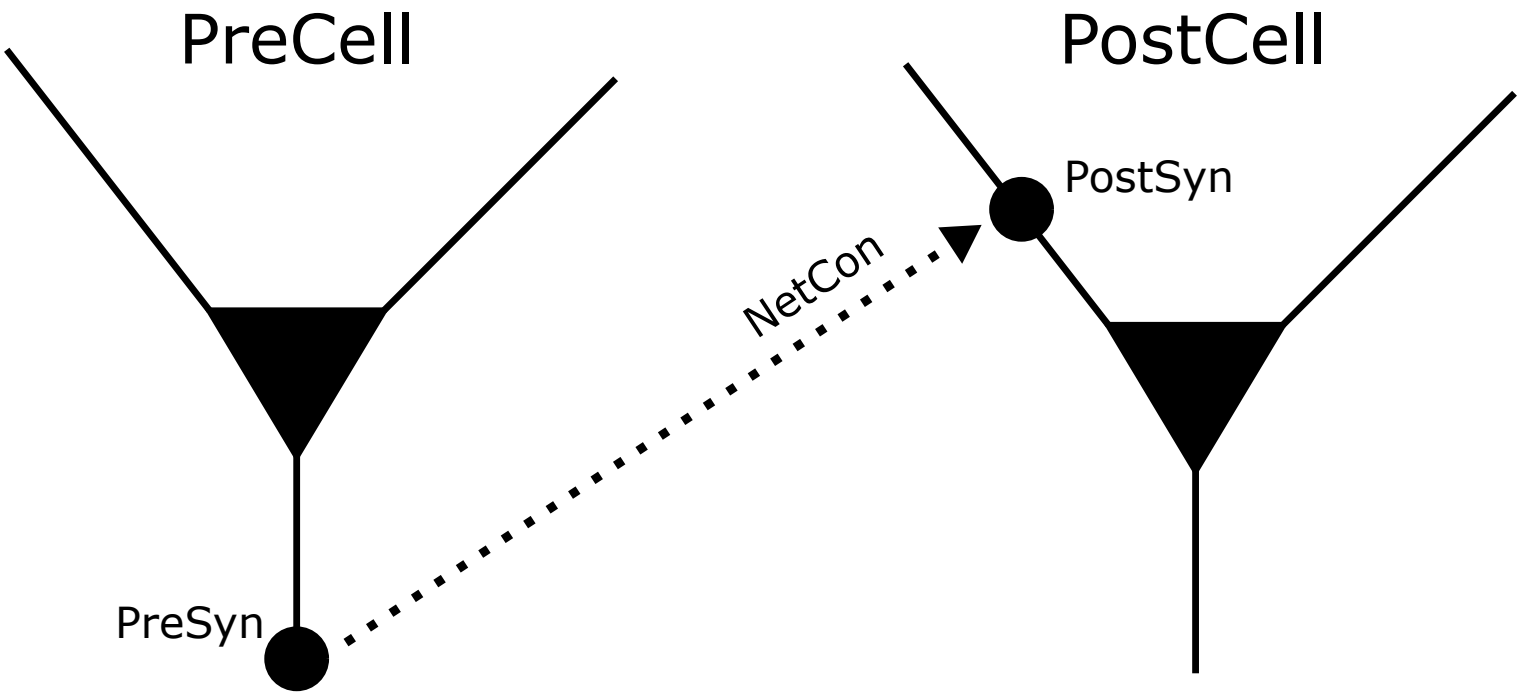
Distributing single cells across processors

The *multisplit* method distributes portions of the tree cable equation across different machines.

A parallel model can fall in 1, 2, or 3 of these classes.

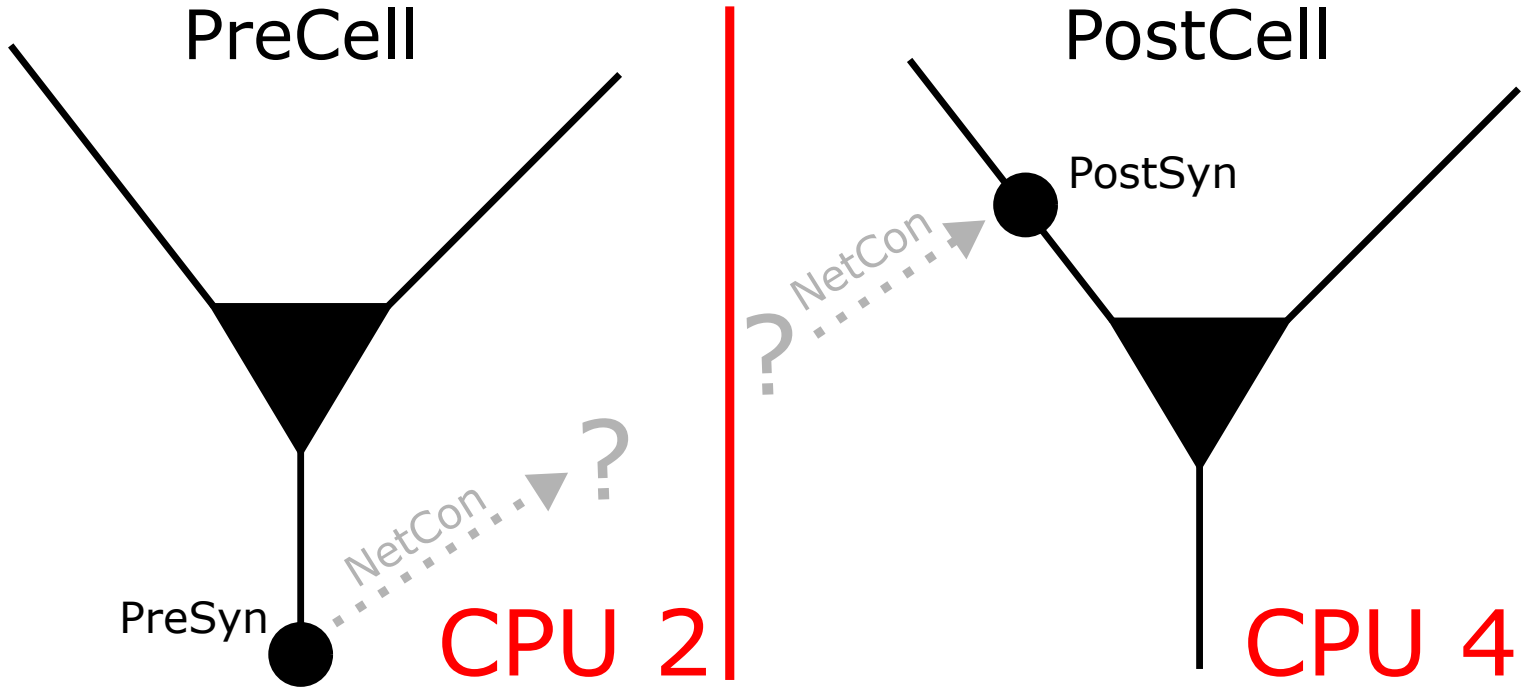
Chemical Synapses

Synaptic connections with one processor




```
nc = h.NetCon(PreSyn, PostSyn)
```

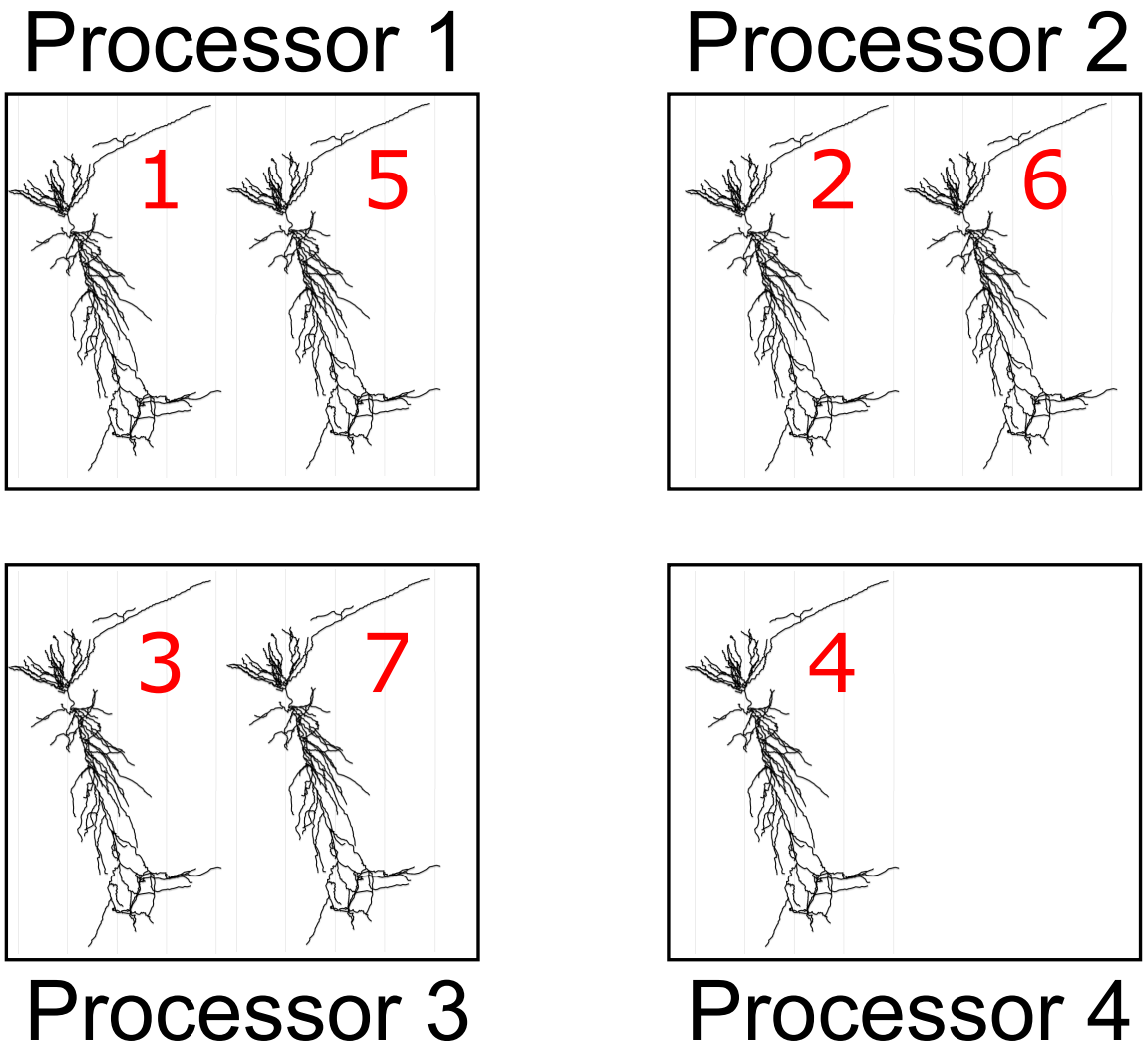
If cells in different processes, a different approach is needed



The `ParallelContext` object facilitates building parallel models.

```
pc = h.ParallelContext()
```

Every spike source **must** have a GID.



Note: to ensure the model produces identical results regardless of the number of processors, also use GIDs to selecting random streams (e.g. Random123).

Assign GIDs to cells

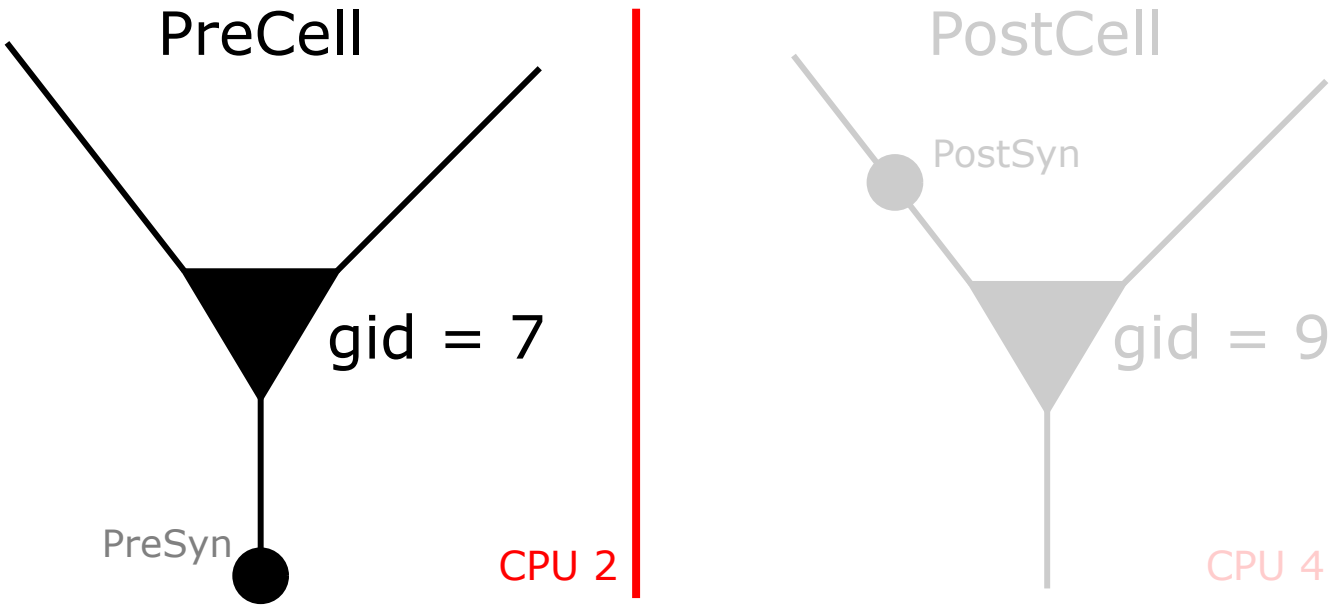
CPU 0		CPU 3		CPU 4	
pc.id	0	pc.id	3	pc.id	4
pc.nhost	5	pc.nhost	5	pc.nhost	5
ncell	14	ncell	14	ncell	14
gid		gid		gid	
0		3		4	
5		8		9	
10		13			

An efficient way to distribute, especially if all cells similar:

```
for gid in range(pc.id, ncell, pc.nhost):
    pc.set_gid2node(gid, pc.id)
    ...
```

(Note: the body is executed at most $\lceil$`ncell/nhost`$\rceil$ times, not `ncell`.)

Configuring the presynaptic connection site



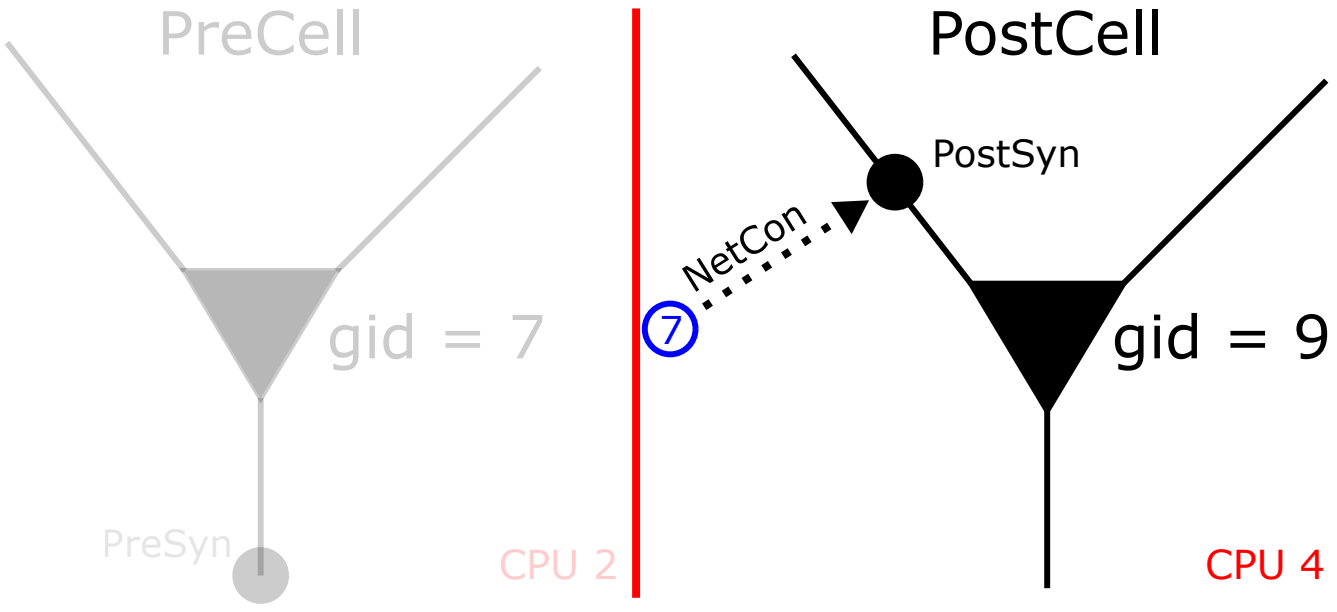
Create cell only where the gid exists:

```
if pc.gid_exists(7):
    PreCell = Cell()
```

Associate gid with spike source:

```
nc = h.NetCon(PreSyn, None)
pc.cell(7, nc)
```

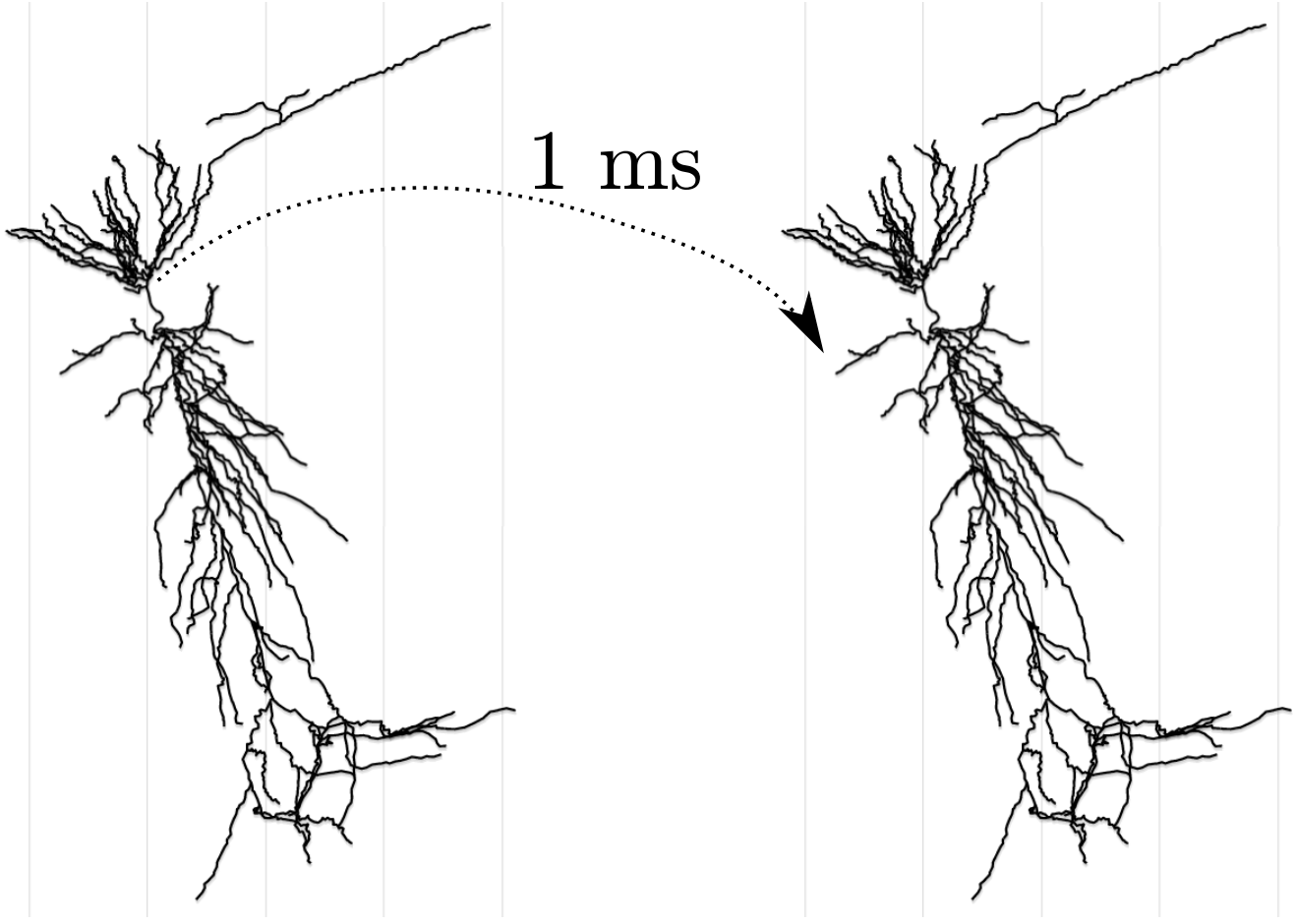
Configuring the postsynaptic connection site



Create NetCon on node where target exists:

```
nc = pc.gid_connect(7, PostSyn)
```

Exploit transmission delays



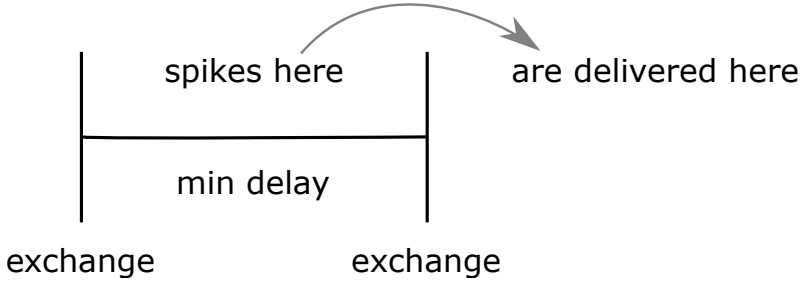
Use `pc.set_maxstep` to set the data transfer frequency. Time for signal propagation down an axon and across the synaptic cleft is many times greater than time step.

Exploit transmission delays: using `pc.set_maxstep`

Run using the idiom:

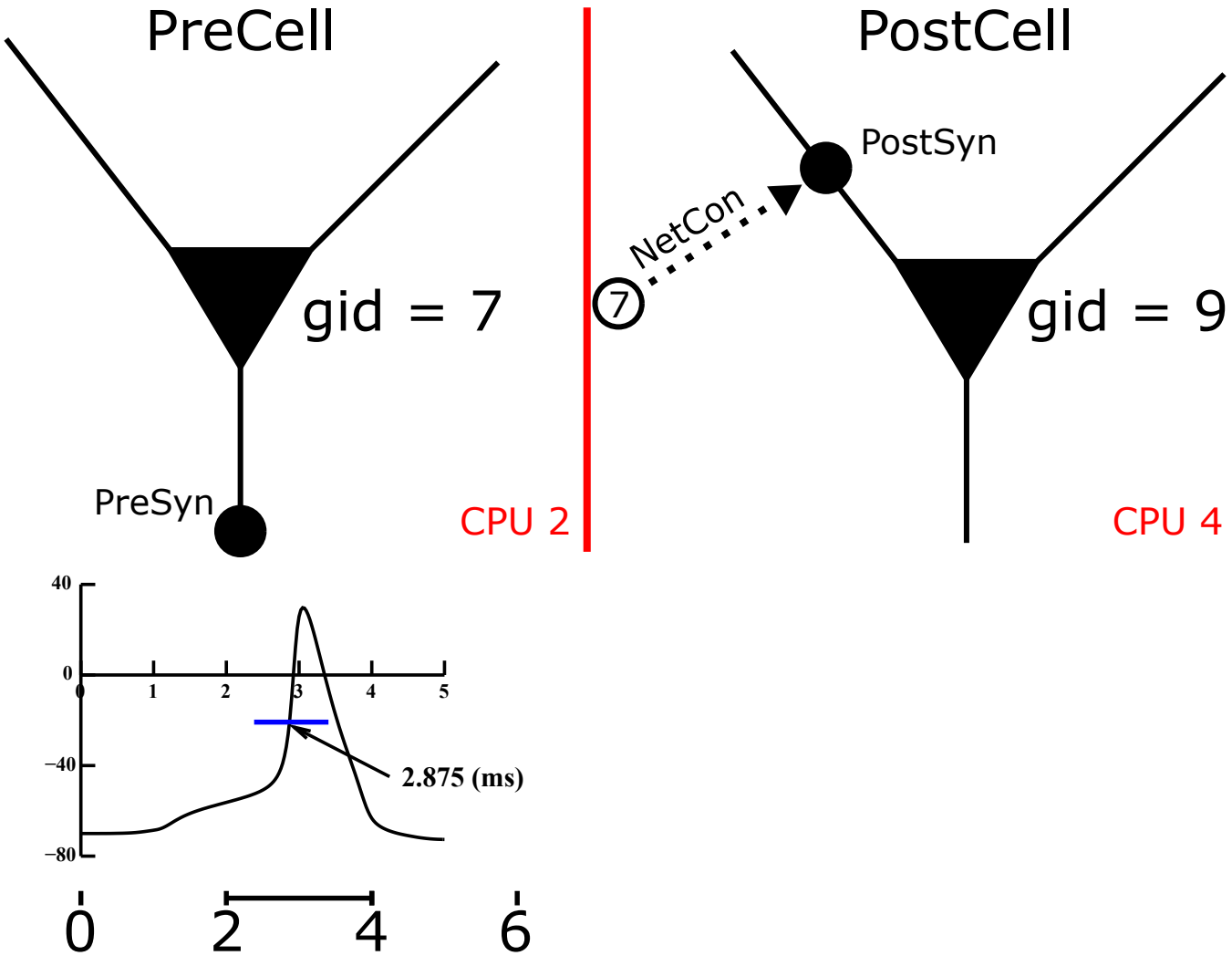
```
pc.set_maxstep(10)
h.stdinit()
pc.psolve(tstop)
```

NEURON will pick an event exchange interval equal to the smaller of all the `NetCon` delays and of the argument to `pc.set_maxstep`. In general, larger intervals are better because they reduce communication overhead.

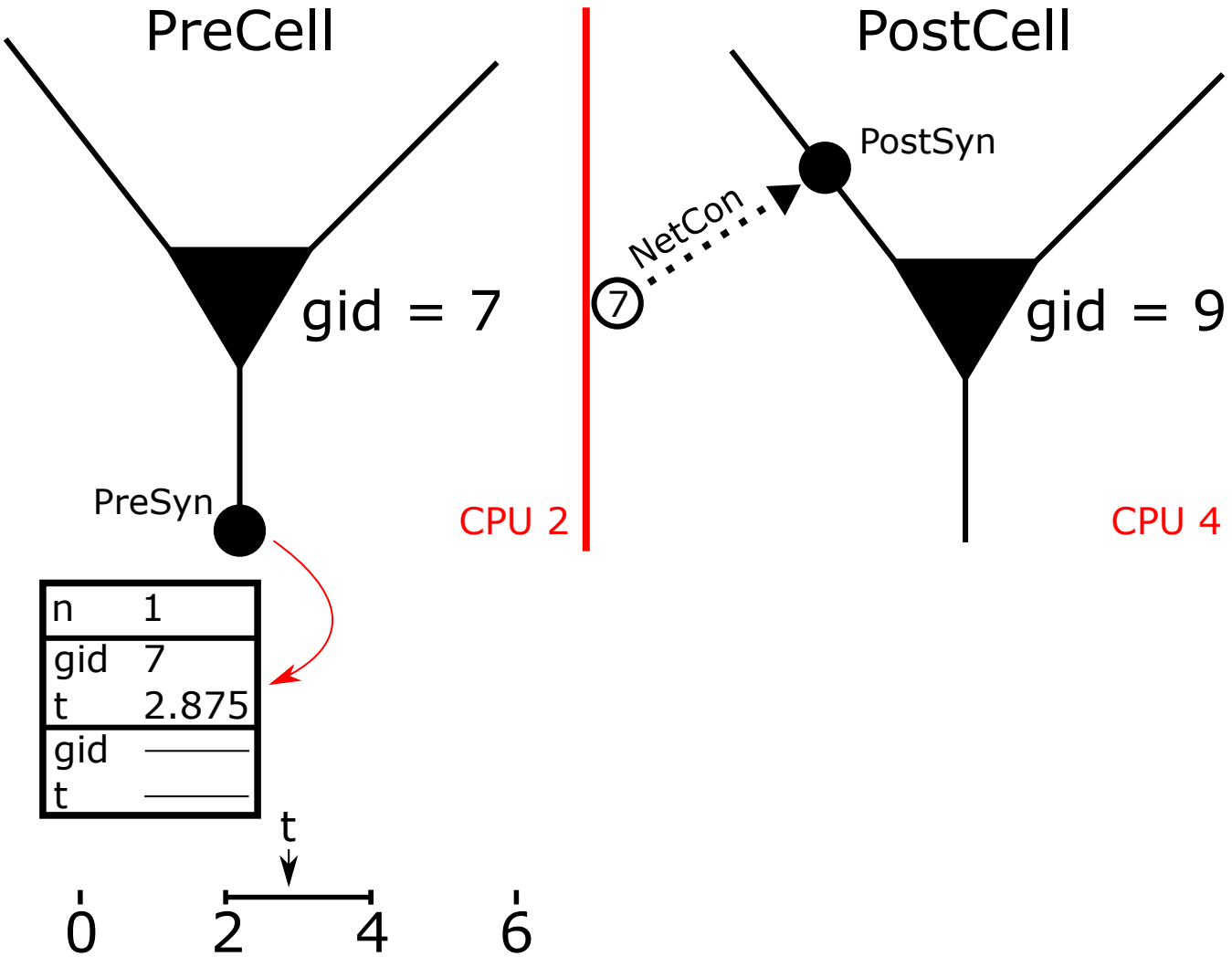


`pc.set_maxstep` must be called on each node; it uses `MPI_Allreduce` to determine the minimum delay.

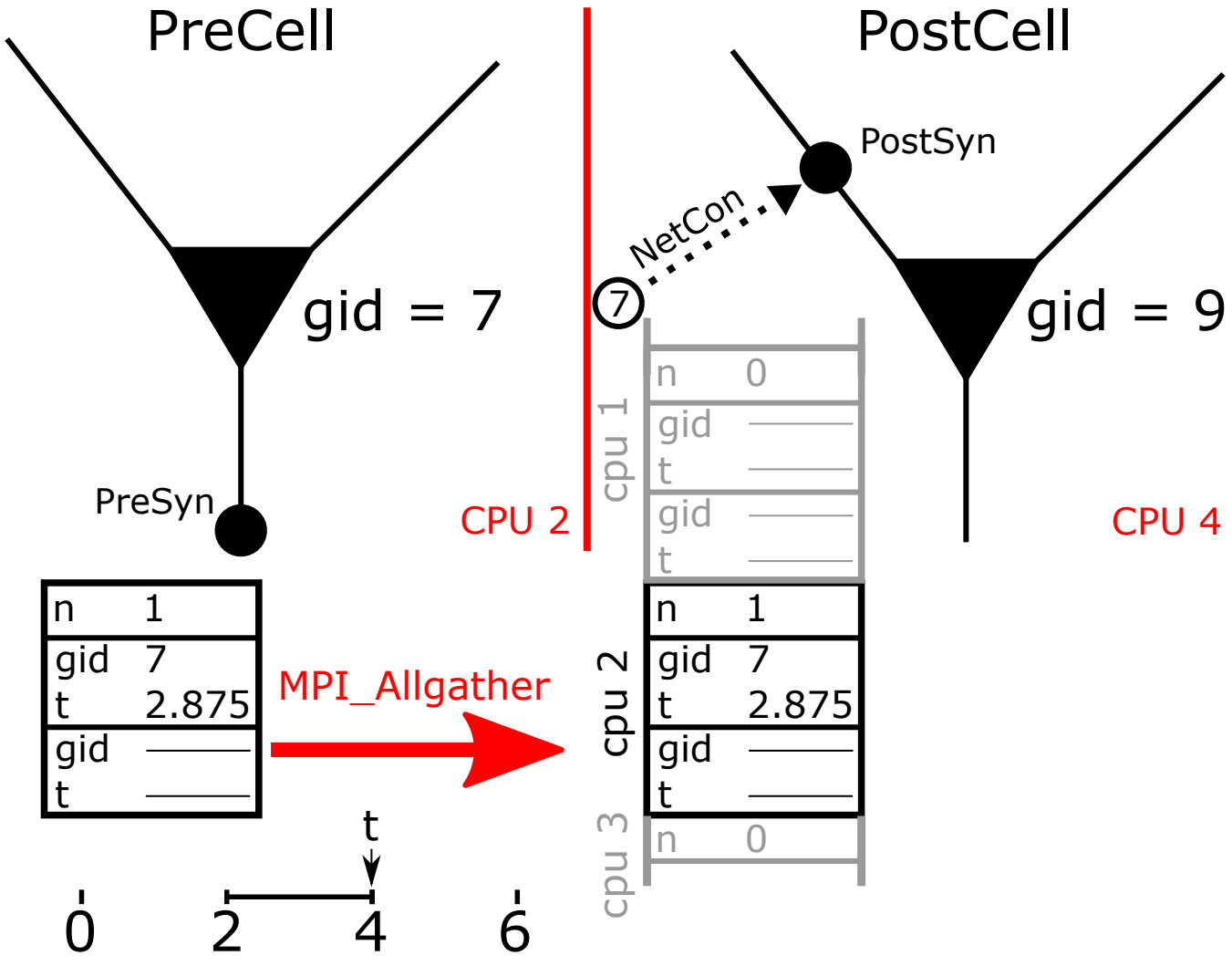
Spike exchange method



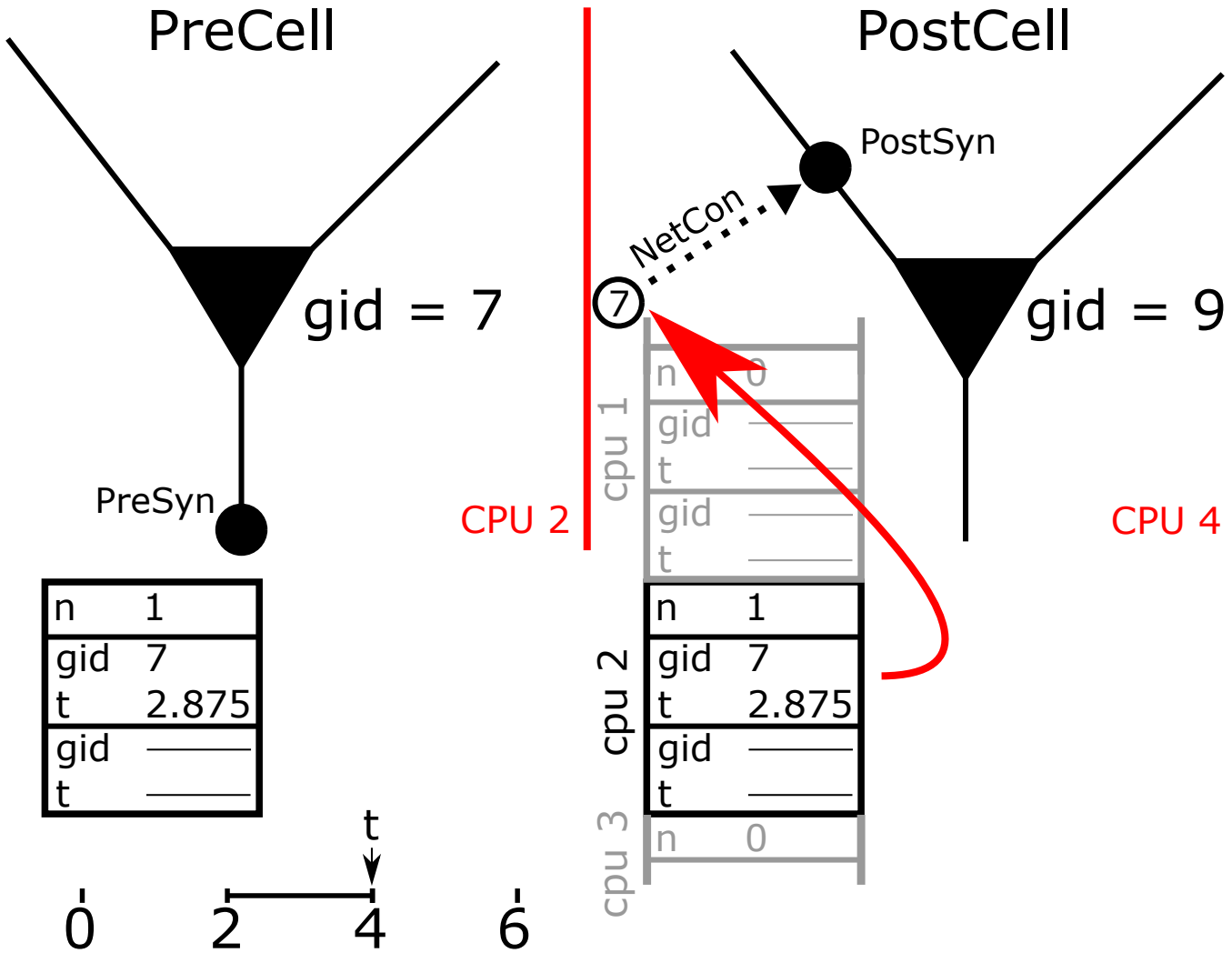
Spike exchange method



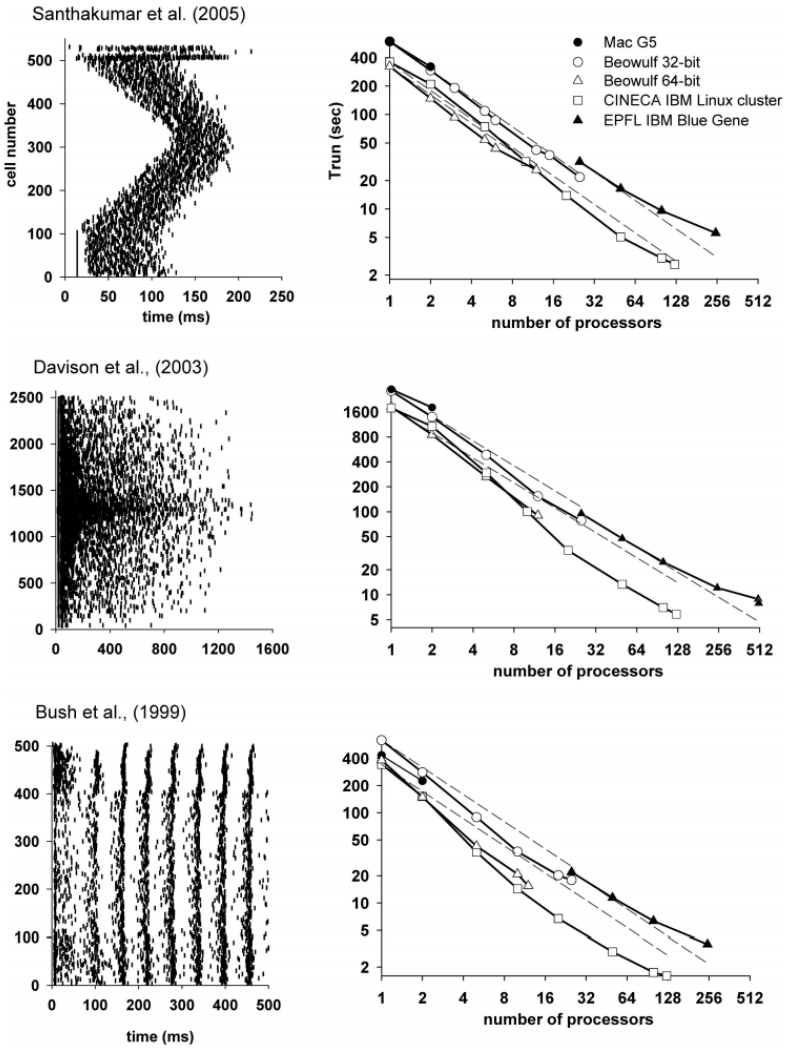
Spike exchange method

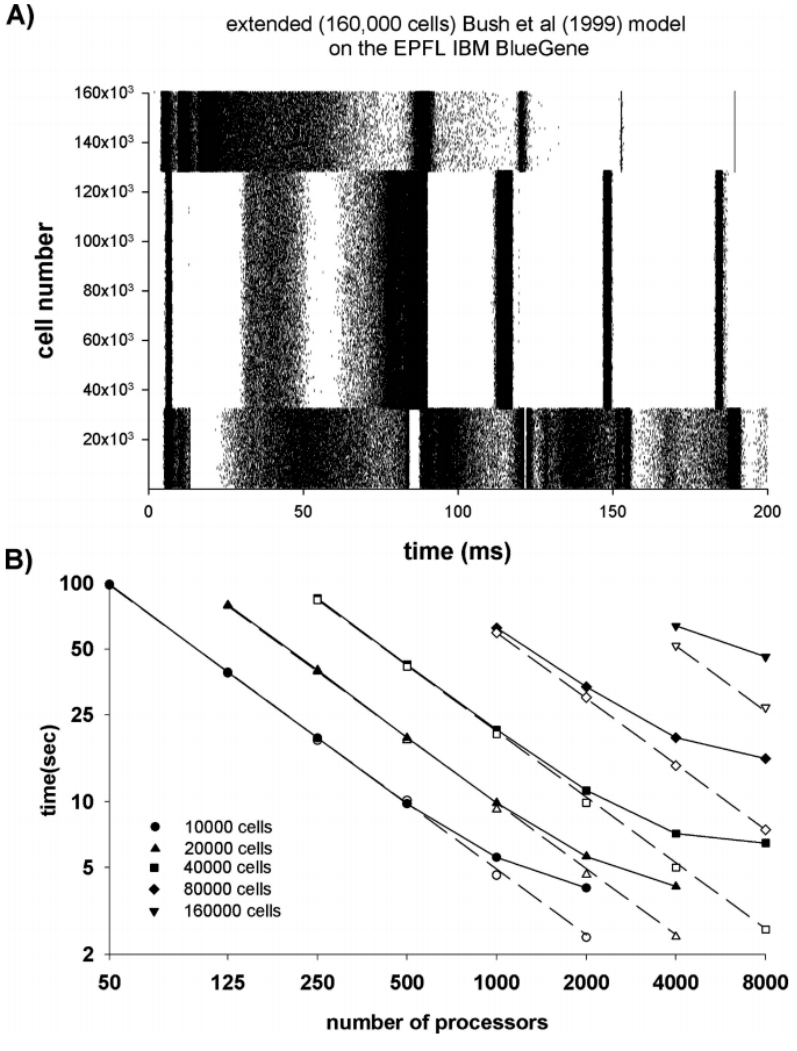


Spike exchange method

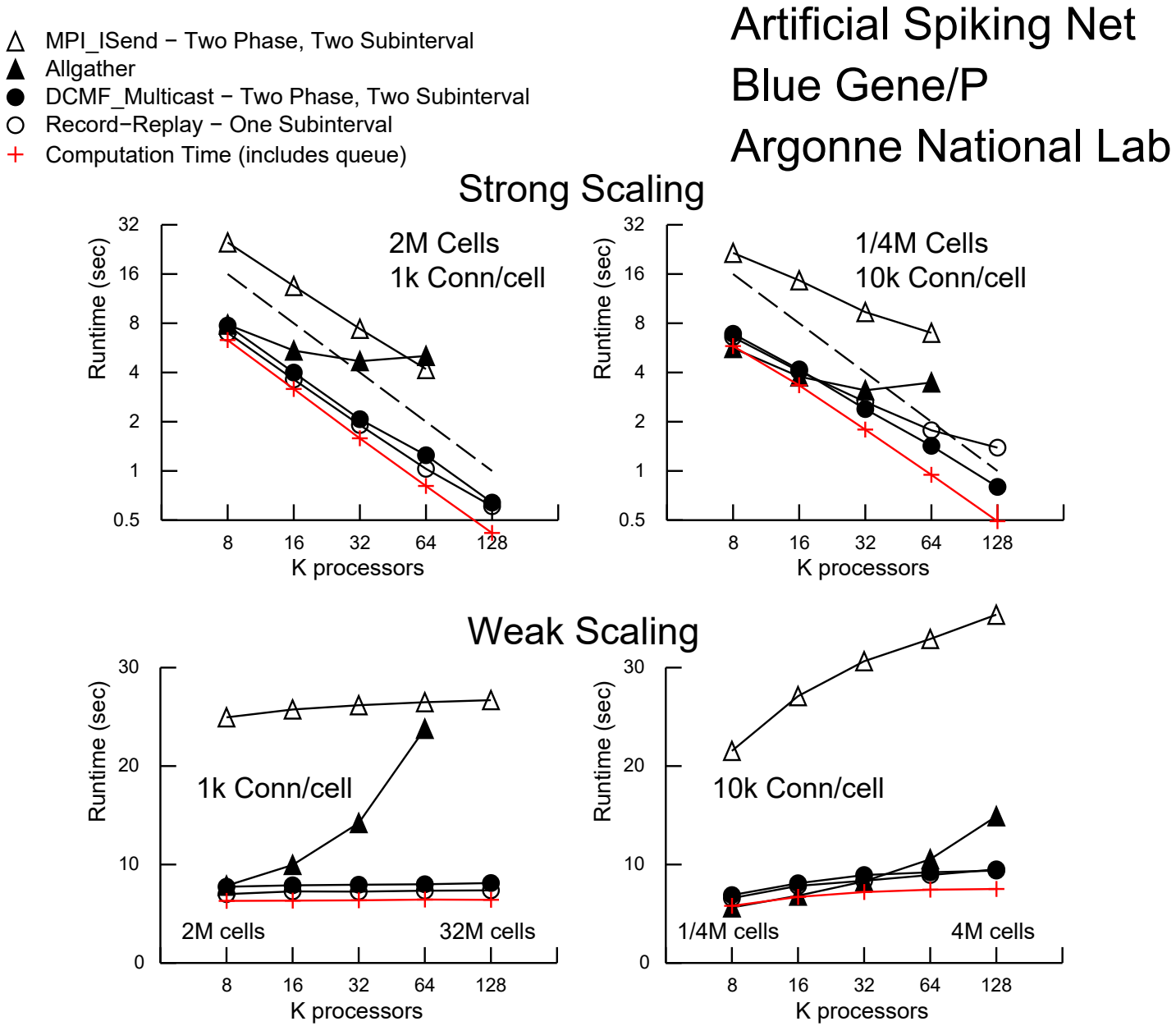


Performance: MPI scaling

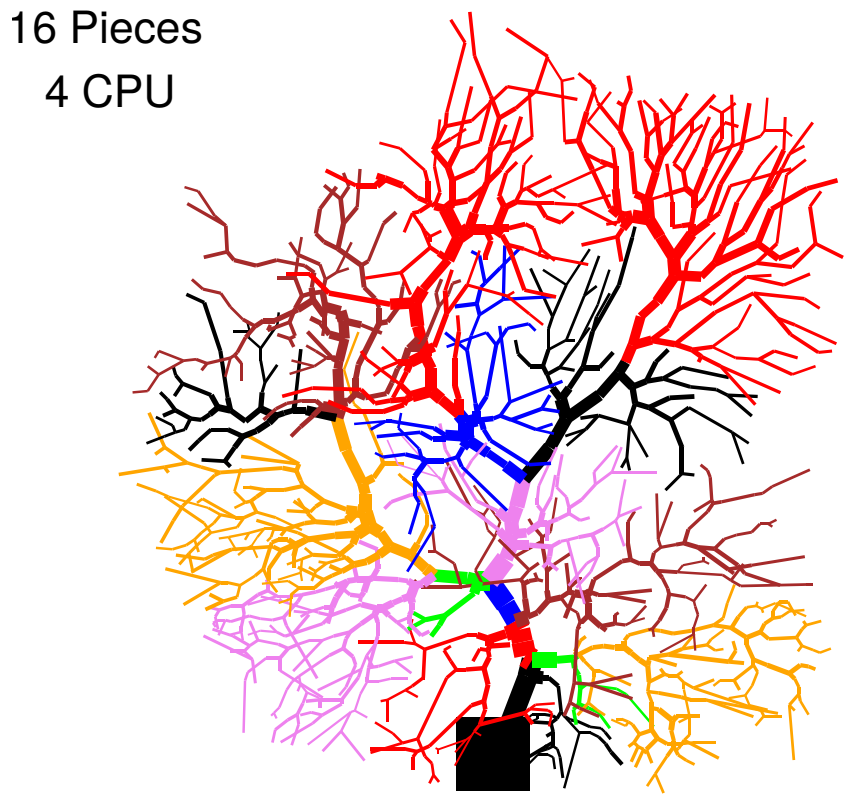


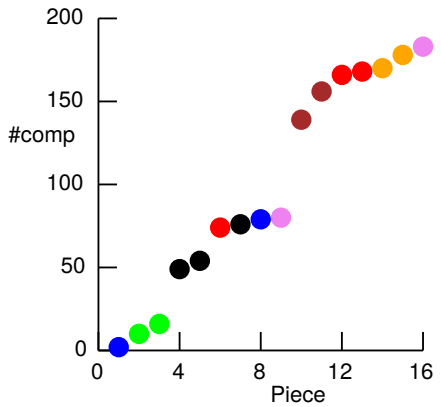


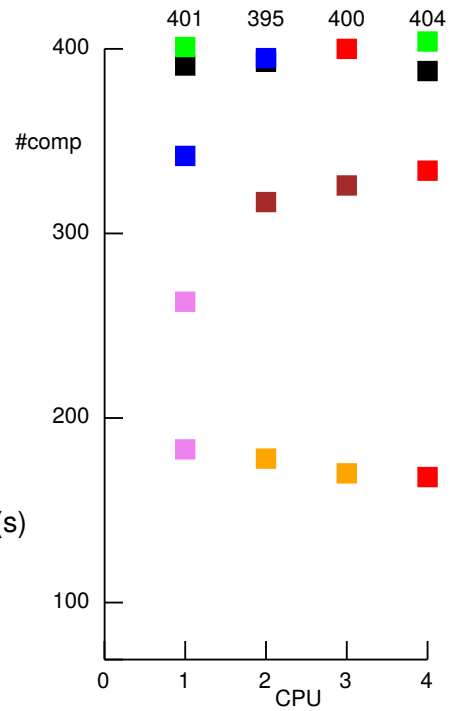
Performance: MPI scaling



Improve load balancing with multisplit







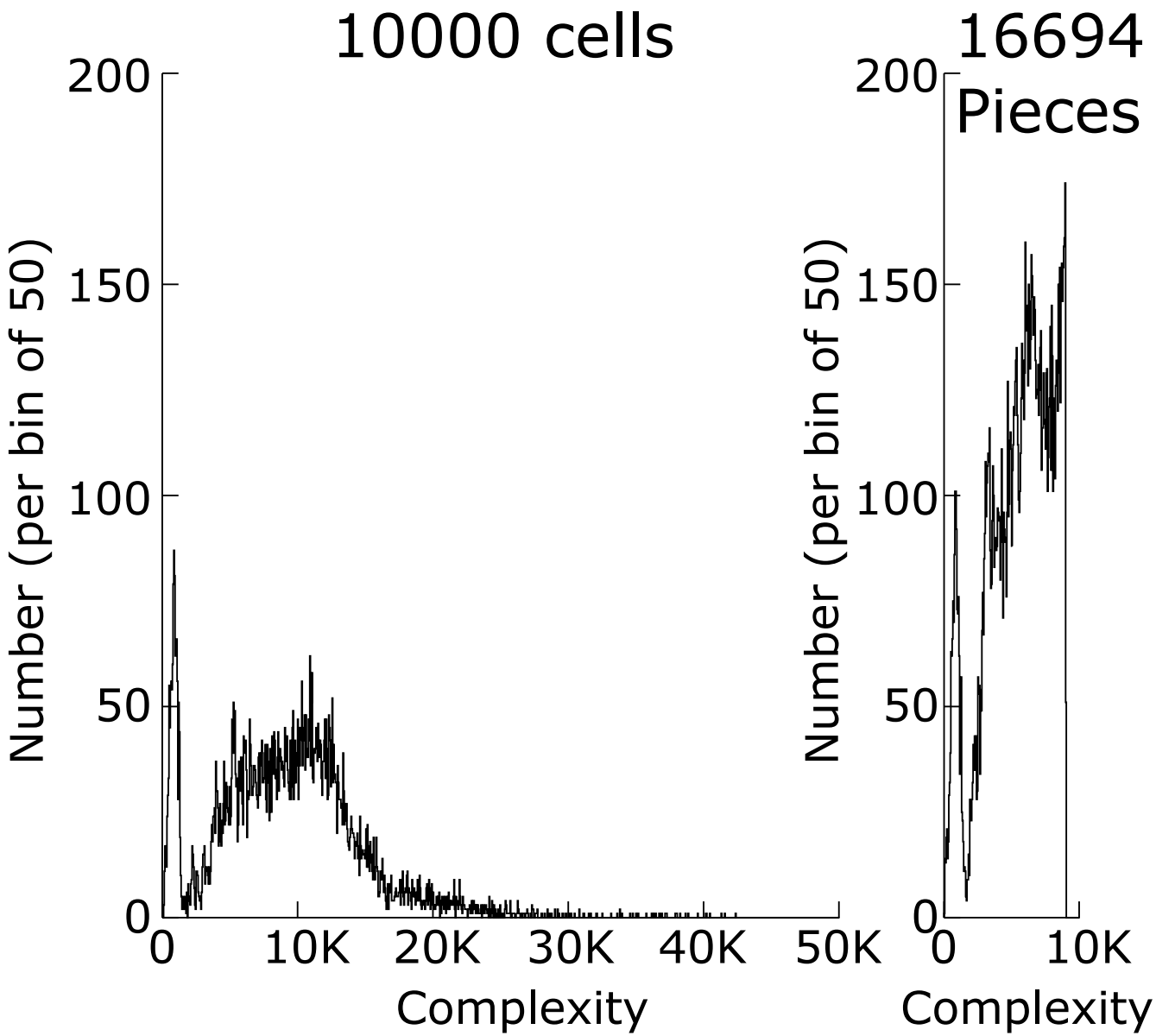
Time (s)

CPU	Computation	Exchange
0	13.82	0.56
1	13.35	1.03
2	13.47	0.90
3	13.56	0.82

	Runtime(s)
16 pieces, 1 cpu	55.0
wholecell, 1 cpu	56.2
16 pieces, 4 cpu	14.4

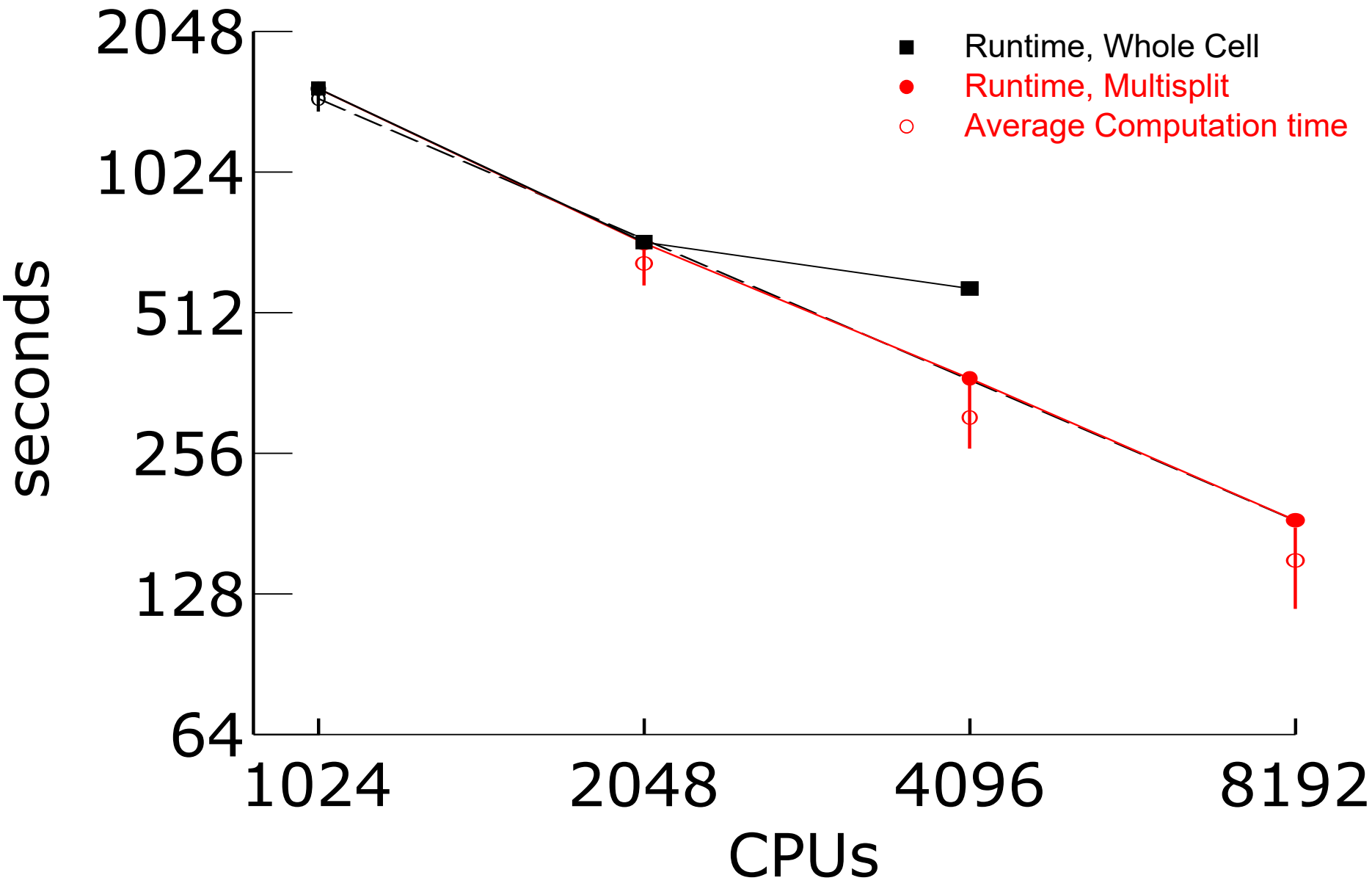
Multisplit algorithm described in Hines et al 2008. DOI: 10.1007/s10827-008-0087-5

Example: Blue Brain Model with 10,000 cells



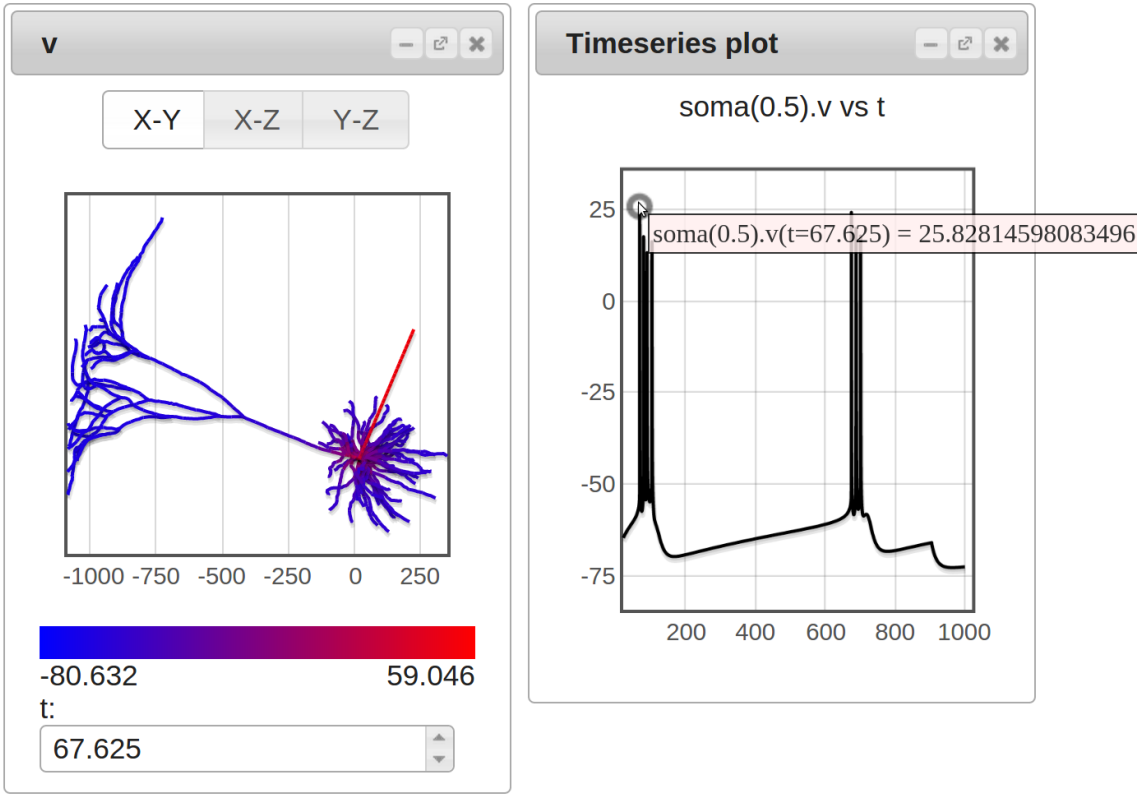
Note: The most complex cell in this model is 7x more complex than the average cell.

Example: Blue Brain Model with 10,000 cells



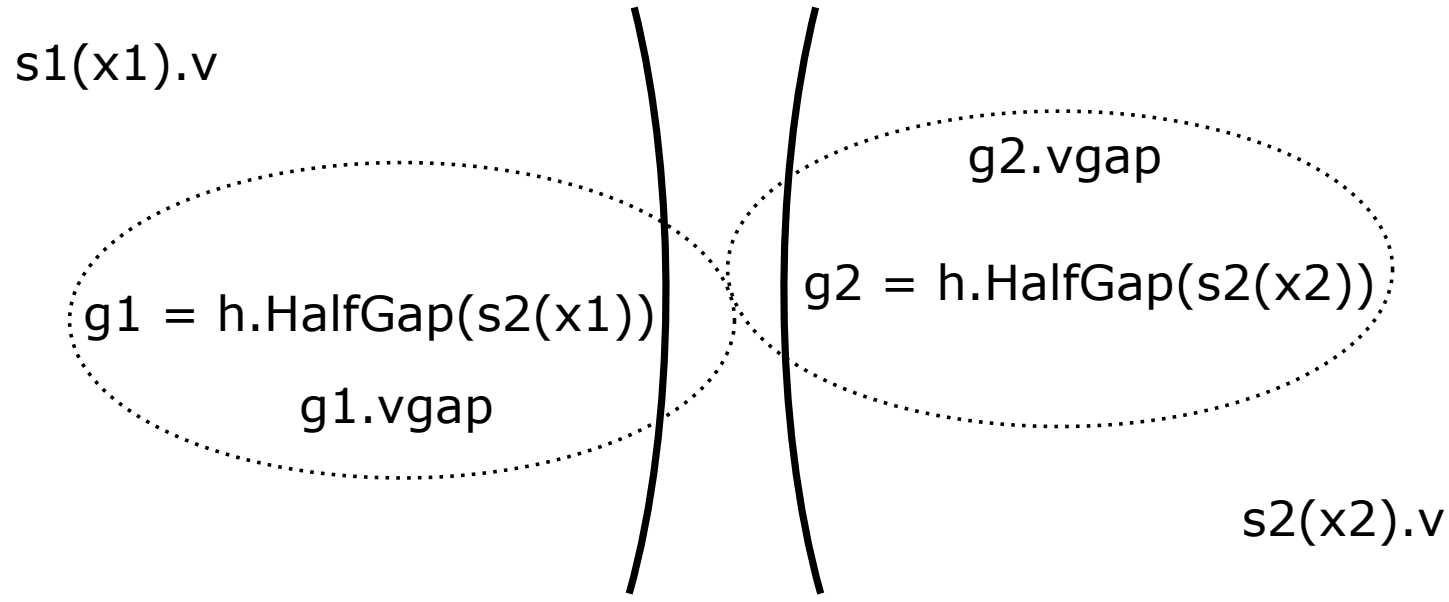
Tip: Store synaptic events; recreate single cells as needed

initial conditions
+
synaptic events
→ neuron dynamics



Gap Junctions

Continuous voltage exchange

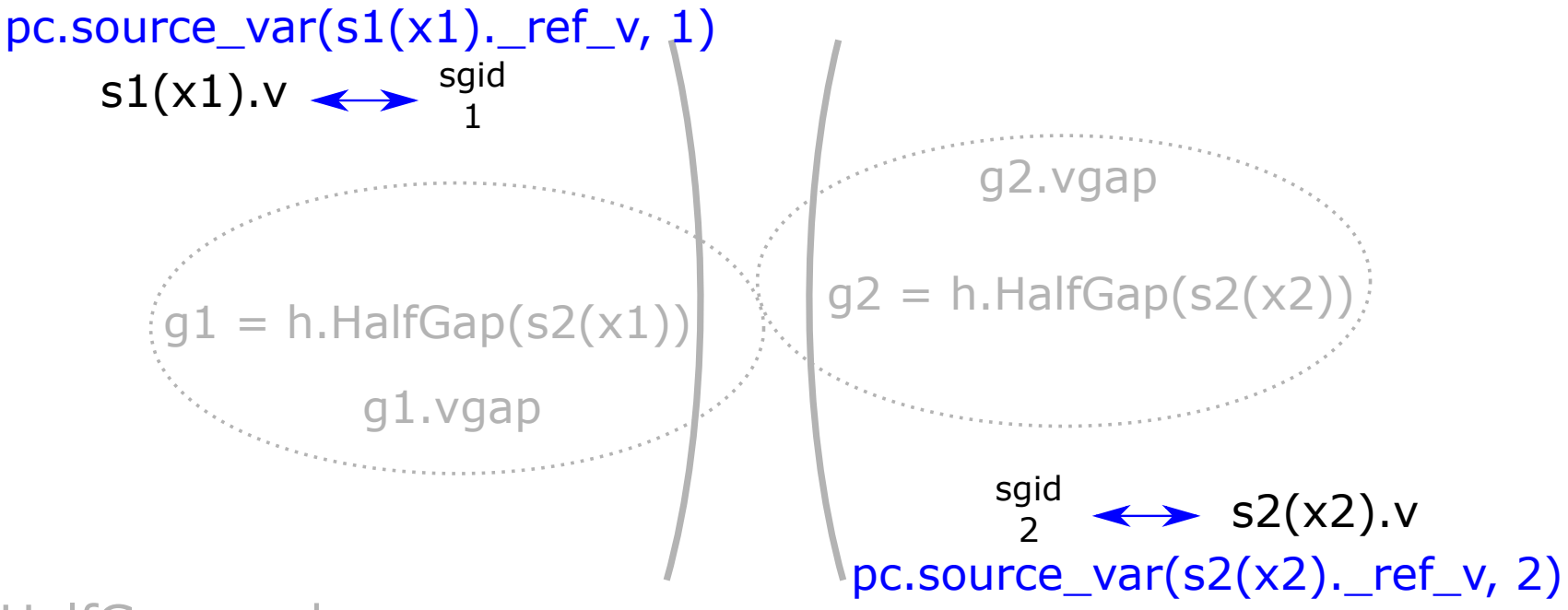


HalfGap.mod

```
NEURON  {
  POINT_PROCESS HalfGap
  ELECTRODE_CURRENT i
  RANGE r, i, vgap
}
PARAMETER { r = 1e9 (megohm) }
```

```
ASSIGNED {
  v (millivolt)
  vgap (millivolt)
  i (nanoamp)
}
CURRENT { i = (vgap - v) / r }
```

pc.source_var to declare source sgid

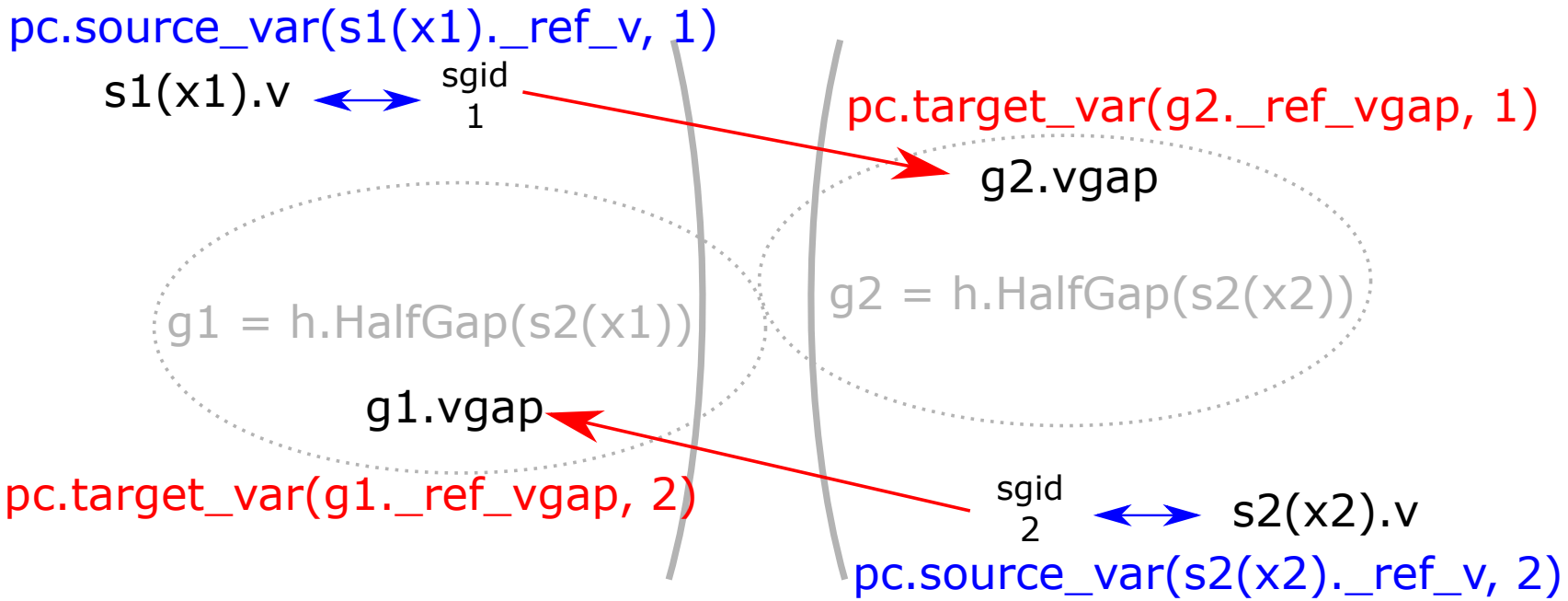


HalfGap.mod

```
NEURON  {
  POINT_PROCESS HalfGap
  ELECTRODE_CURRENT i
  RANGE r, i, vgap
}
PARAMETER { r = 1e9 (megohm) }
```

```
ASSIGNED {
  v (millivolt)
  vgap (millivolt)
  i (nanoamp)
}
CURRENT { i = (vgap - v) / r }
```

pc.target_var to declare target connection



HalfGap.mod

```
NEURON  {
  POINT_PROCESS HalfGap
  ELECTRODE_CURRENT i
  RANGE r, i, vgap
}
PARAMETER { r = 1e9 (megohm) }
```

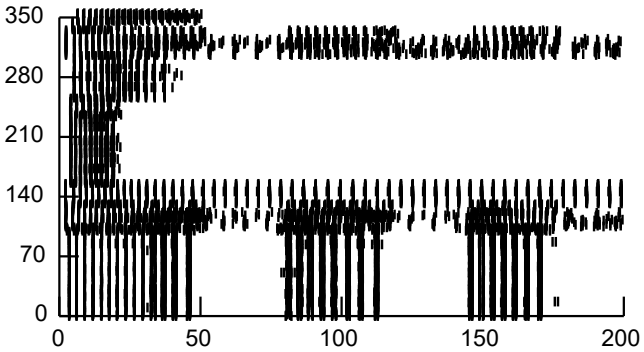
```
ASSIGNED {
  v (millivolt)
  vgap (millivolt)
  i (nanoamp)
}
CURRENT { i = (vgap - v) / r }
```

Performance: Traub model

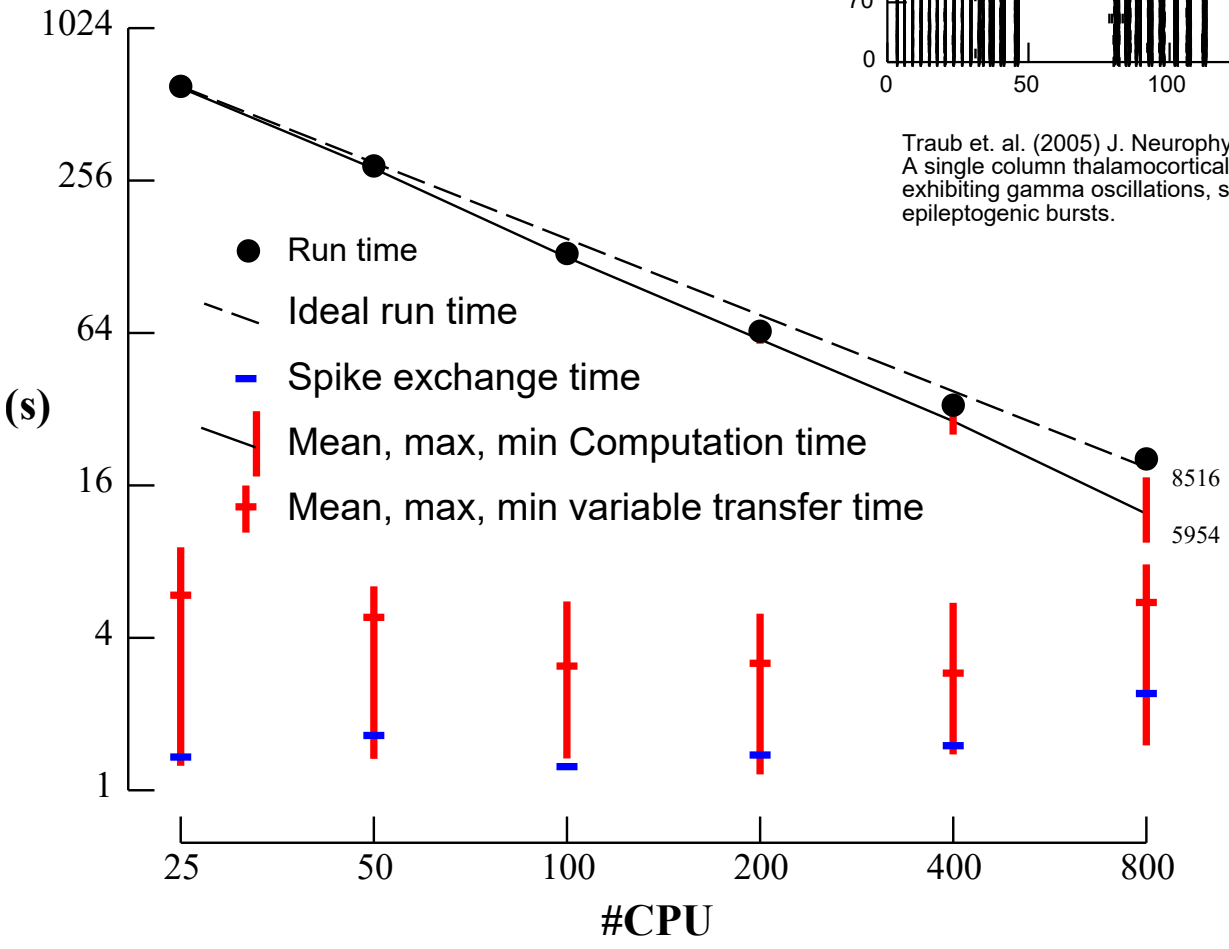
Pittsburgh Supercomputing Center

Bigben Cray XT3

2068 2.4 GHz Opteron Processors



Traub et. al. (2005) J. Neurophysiol 93: 2194
A single column thalamocortical network model exhibiting gamma oscillations, sleep spindles and epileptogenic bursts.



3560 cells 14 types
3500 gap junctions
5,596,810 equations
73,465 spikes
1,122,520 connections
19,844,187 delivered

Performance: Traub model with multisplit