brains, minds media
ISSN: 1861-1680

Using P-GENESIS for Parallel Simulation of GENESIS Models

A Brief Overview

Greg Hood

Pittsburgh Supercomputing Center, Carnegie Mellon University, United States, ghood@psc.edu

urn:nbn:de:0009-3-2271

Abstract. P-GENESIS is an extension to the GENESIS neural simulator that allows users to take advantage of parallel machines to speed up the simulation of their network models or concurrently simulate multiple models. P-GENESIS adds several commands to the GENESIS script language that let a script running on one processor execute remote procedure calls on other processors, and that let a script synchronize its execution with the scripts running on other processors. We present here some brief comments on the mechanisms underlying parallel script execution. We also offer advice on parallelizing parameter searches, partitioning network models, and selecting suitable parallel hardware on which to run P-GENESIS.

Keywords: GENESIS, P-GENESIS, parallel simulation, parameter searching

Citation: Hood G (2005). Using P-GENESIS for Parallel Simulation of GENESIS Models – a Brief Overview. Brains, Minds and Media, Vol. 1, bmm227 (urn:nbn:de:0009-3-2271)

Supplementary Material

Article Resources

GENESIS Resources

Datasheet

Who Can Benefit from P-GENESIS?

As one scales up a neural model to become more realistic, it is not unusual to run into practical limits on the size of that model due to the simulation time required to adequately evaluate it. However, the processors on commodity desktop computers are nearly as fast (in terms of clock speed) as any general-purpose processor currently available in the marketplace, so the only way to substantially increase performance is to apply parallel simulation techniques. P-GENESIS allows one to take advantage of parallelism when performing GENESIS (Bower & Beeman 1998) simulations. There are two main areas in which P-GENESIS is useful: simulating large network models (Goddard & Hood 1998, Goddard & Hood 1997, Howell et al. 2000, Vanier 2005a) and parameter searching (Vanier and Bower 1999, Vanier 2005b). In contrast, large single-cell models are currently best handled by using optimizations such as the GENESIS `hsolve` object, and these are not appropriate for partitioning with P-GENESIS.

Once one decides to go the parallel route, there are several options for hardware on which to run the simulations. At the low end is to simply use a local network to couple available desktop machines together into a single computing system. Such an arrangement is usually adequate for doing parallel parameter searches, but the limited TCP/IP network bandwidth and high communication latencies often limit the degree of useful parallelism when doing large network simulations. For these situations, one may obtain a higher degree of parallelism through the use of computational clusters which are more tightly coupled through the use of special interconnect hardware. These can be found as departmental or institutional computing resources at many universities and research institutions. One may also have the option of applying for time at regional or national supercomputing facilities, and thereby gain access to very large scale systems for the largest simulations.

Mechanisms

One uses P-GENESIS by modifying one's GENESIS simulation scripts so that portions of them may execute in parallel. We will refer to each instance of a running P-GENESIS process as a "node". Generally, for best efficiency the number of nodes will be the same as the number of physical processors in a system, although, for testing, the two may be different. Each node in a P-GENESIS configuration runs the same set of scripts, but the flow of control can vary from node to node so that each may be performing different operations. For example, to obtain a master/worker form of parallel execution, a script can test whether it is running on node 0, in which case it behaves as a master node, or, if not, it behaves as a worker node.

P-GENESIS adds a number of new commands to the standard GENESIS script language. The *paron/paroff* commands are required in any parallel script, and enable and gracefully disable parallel script execution. Synchronization between scripts can be accomplished using *barrier* commands. A simple "Hello, world" script (the `hello.g` example in the Supplementary Material) illustrates this, and is useful in making sure P-GENESIS was configured and installed correctly before trying more complex scripts.

One of the most important concepts in P-GENESIS is the notion of a remote procedure call. A script running on one node can remotely execute a GENESIS command on another node by appending an "@" symbol followed by a specification of the node(s) on which to run the command. For example, a "create@2 compartment /cell/soma" command causes the command `"create compartment /cell/soma"` to be executed on node 2. Note that each node has a distinct namespace so that if an element is created with a certain name on one node, that element is not directly accessible by that name on other nodes.

The basic command for linking elements (*e.g.*, creating a synapse) on different nodes is called *raddmsg*. It is analogous to *addmsg*, with the enhancement that the destination element can be specified to be on a different node by using the "@" syntax. So, for example, "`raddmsg /cellA/soma/spike /cellB/dend[5]/Ex_channel@2 SPIKE`" connects the neuron of cellA (which is located on the same node as where this command is executed) to an excitatory channel of the fifth dendritic compartment on cellB (which is located on node 2). While P-GENESIS attempts to reproduce the behavior of GENESIS very closely, an extra one-timestep delay is introduced on messages requiring internode communication. This is a side effect of the way P-GENESIS implements internode communication using "postmaster" objects. Normally, in networks with biologically-realistic synaptic delays, this additional one timestep delay is insignificant, and does not make an appreciable difference in the simulation results.

Parameter Searches

Parameter searches for neural models are especially amenable to parallelization. Genetic algorithms (GA) and simulated annealing (SA) are two commonly used methods for performing parameter searches (Vanier and Bower 1999, Vanier 2005b). Each can be parallelized to achieve significant speedups in comparison with doing the search sequentially on a single processor (*i.e.*, doing simulation/evaluation of particular parameter sets one-by-one).

One way of parallelizing a genetic search is to use a population-based approach wherein a fixed number of parameter sets are kept "alive", and new candidates (mutations from the existing population) are evaluated in parallel. If a new candidate's evaluation is better than the worst of the existing population, the new candidate replaces it.

Example *param* in the Supplementary Material illustrates the use of this method with a simple search for parameters that optimize a mathematical formula. A more realistic example (from a neural modeling perspective) of a parameter search is contained in example *param2*, where conductances within a single cell model form a parameter set that is optimized with respect to making the model reproduce an experimentally-obtained spiking pattern.

Simulated annealing (SA) algorithms may also be parallelized by trying multiple moves in parallel. We refer the reader to the literature (Azencott 1992) for details on these methods. In general, they scale less well than genetic algorithms because the search space is more focused and there is a greater probability of visiting (or revisiting) points in the search space that would not be evaluated in a sequential SA search.

One could, of course, conduct a parameter search by simply using serial GENESIS as a "black box", and invoking it repeatedly via a shell script or other control program. The advantage that P-GENESIS has is that one can run multiple simulations without repeatedly incurring the overhead of starting the GENESIS executable and building the model for each evaluation of a point in the search space.

Network Models

The other area in which P-GENESIS is extremely useful is in the parallel simulation of large network models, models that would be prohibitively time-consuming on a single processor. In order to run a network in parallel, P-GENESIS requires the script writer to explicitly partition the model across the available set of computational nodes. Modifying the scripts for parallel execution can be simple or difficult, depending on various factors, and thus the decision on when it makes sense to parallelize a model must take these into account. The following factors make a model a better candidate for parallelization with P-GENESIS:

- populations of similar neurons that may be partitioned across nodes
- the use of computationally-intensive (*i.e.* multicompartment) models for at least some of the neurons (in contrast, if all neurons are integrate-and-fire, then it is likely that the communication costs will outweigh the benefits of running in parallel)
- simulation runs that take several hours or more on serial machines (it is harder to justify the human time to modify the scripts for shorter run times)
- models that will be altered and rerun multiple times (again, it is harder to justify the cost of modifying the scripts for a one-shot run)
- very large models that do not fit in the memory of a single computer (and thus must be parallelized to run at all)

When parallelizing a network, one typically creates the network elements (*i.e.*, neurons), then makes all the interconnections (synapses) among these. Thus, a typical script sequence is the following:

```
paron
// set up network elements using create commands
barrier
// make connections between elements using raddmsg commands
barrier
// run simulation
barrier
paroff
```

Example *orient1* illustrates a parallel decomposition of the `orient_tut` model found in the `Scripts/orient_tut` directory within the GENESIS software distribution. This parallelization is based on putting all elements of one type on one node, all elements of another type on another node, etc. This example is included for its simplicity. However, it does not scale well, in that there is no way to take advantage of more than 3 nodes in the machine configuration. It is also subject to load imbalance problems, since it is quite likely that the population of V1 cells will take more computational effort than the retinal cells.

One can partition this model up in a much more scalable way by dividing each population of neurons across the set of available nodes. In general, one should try to distribute the workload involved in simulating a network as evenly as possible across the nodes. It is important to not have a single node that is overburdened and becomes a bottleneck, which would decrease the overall processor utilization. A common way to accomplish such a division is to parameterize the scripts in terms of *n*, where *n* is a user-specified node count, or is obtained at runtime from the environment with the *nnodes* function. An example of such a scalable parallelization is given in *orient2*.

It is recommended that one first try their parallelization on a smaller-scale simulation model, rather than on the ultimate model one aims to construct. The model can be scaled down in terms of time interval to be simulated, number of elements in the network, number of nodes in the computing system, or some combination thereof. This makes debugging more tractable, and allows one to check correspondence between the results of the serial and parallel scripts. These results should agree fairly closely, but they will not be exactly the same because of the extra timestep delay on internode communication (as discussed above). Finally, obtaining preliminary performance measurements on smaller-scale models allows one to obtain estimates of scaling behavior for the target computing system, and these can then be used to predict the computational requirements (memory, run time) of the full-size model as a function of the number of nodes employed. This enables one to choose a suitable number of nodes for the simulation, a number that is large enough to fit the model into memory and get a substantial speedup, while not so large that efficiency is sacrificed.

References

Azencott, R. (Ed.) (1992). Simulated Annealing: Parallelization Techniques, John Wiley & Sons, Inc.

Bower, J.M. and Beeman, D., (Eds.) (1998), The Book of GENESIS, 2nd ed., Springer-Verlag.

GENESIS Home Page (1994-2005). http://www.genesis-sim.org/GENESIS/.

Goddard, N.H. and Hood, G. (1997). Parallel Genesis for Large Scale Modeling, in Computational Neuroscience: Trends in Research 1997, Plenum Publishing, NY, pp 911-917.

Goddard, N.H. and Hood, G. (1998) Large-Scale Simulation using Parallel GENESIS, in The Book of GENESIS, 2nd ed., Bower, J.M. and Beeman, D. (Eds), Springer-Verlag.

Howell, F.W., Dyhrfjeld-Johnsen, J., Maex, R., Goddard, N.H., De Schutter (2000), E. A large-scale model of the cerebellar cortex using PGENESIS. Neurocomputing 32-33, pp 1041-1046.

P-GENESIS Home Page (1997-2005). http://www.psc.edu/general/software/packages/pgenesis.

Vanier, MC and Bower, JM (1999). A Comparative Survey of Automated Parameter-Search Methods for Compartmental Neural Models. J. Comput. Neurosci. 7: 149-171.

Vanier, MC (2005a). Constructing Large-scale Network Models, Advanced Tutorial held at Wam-Bamm'05; http://www.wam-bamm.org/WB05/Tutorials/

Vanier, MC (2005b). Parameter Searching in Neural Models, Advanced Tutorial held at Wam-Bamm'05; http://www.wam-bamm.org/WB05/Tutorials/