Introduction to SNNAP, OO design and Java

SNNAP (Simulator for Neural Networks and Action Potentials; http://SNNAPuth.tmc.edu) is a versatile and user-friendly tool for rapidly developing and simulating realistic models of neurons and neural networks. SNNAP is written in the Java programming language, and is portable to almost any computer. SNNAP is being redesigned to take advantage of OO features of Java (see accompanying abstract by Cai et al.). An OO design provides many benefits, such as a multithreaded architecture.

A multithreaded architecture provides an execution framework for parallel processing. Simulations often incorporate multiple ‘computational elements’, such as individual neurons in a neural network or individual compartments in a model neuron. In an OO design, each computational element is an object, and as such, may be executed on a separate thread. On a parallel computer, multiple threads can be processed in parallel. An example of classic dual thread architecture is shown below with the graphical user interface (GUI) running on one thread, and the model solver running on a second thread.

Parallel computer architecture

There are four major types of computer architectures that are used for parallel computation: i) parallel vector processors, ii) shared-memory, symmetrical multiprocessors (SMP), iii) distributed-memory, massively-parallel processors and iv) distributed-memory, cluster computers (often referred to as a Beowulf cluster). Of these four architectures, in terms of cost, the two most popular parallel-processing paradigms are the SMP and cluster computers.

Parallel SNNAP: from object-oriented design to multithreading

Our work focuses on developing a tool for simulating complex models of neurons. We began by analyzing different parallel computer architectures.

There are four major types of computer architectures that are used for parallel computation:

1. Parallel vector processors: These processors are designed to perform operations on many data points simultaneously. This architecture is well-suited for applications where data can be processed in a sequential manner, such as in numerical simulations.
2. Shared-memory, symmetrical multiprocessors (SMP): These systems consist of multiple processors that share a common memory. Each processor can access any part of the memory, allowing for efficient communication between processes.
3. Distributed-memory, massively-parallel processors: These systems consist of many processors that communicate over a network, sharing data only through explicit communication.
4. Distributed-memory, cluster computers: These systems are made up of separate computers that can communicate with each other over a network. Each computer has its own memory space, and the system as a whole can be viewed as a single computer with many processing units.

We developed a prototype model to examine the cost/benefits of parallel processing. The prototype executed a variable number of coupled Hodgkin-Huxley models in parallel. The computational cost of parallel processing was due mainly to communication between threads. Tests identified two issues that determined the benefits from parallel processing: load balancing and model granularity. Load balancing relates to how many threads run efficiently on a single processor. Model granularity relates to how many ODEs are solved per thread. Compared to a nonparallel program (blue curve, upper figure above), a multithreaded program which ran on a dual Xeon processor computer had execution times reduced by 50% for up to 8 threads with 25 ODEs per thread (red curve, lower figure above). The intersection of the red and green curves in the figure above corresponds to the case when the computing time of 4 threads executing 25 ODEs (with communication) was equal to 25 threads executing 4 ODEs (without communication), i.e., 100 ODEs computed in total. The difference between the blue and green curves in the upper figure displays the gain due to parallel processing. With communication, the overall gain (difference between blue and red curves above) increased for larger numbers of ODEs, due to the increase in the ratio between computing time versus communication time increases.

Summary

We developed a prototype model to examine the cost/benefits of parallel processing. The prototype executed a variable number of coupled Hodgkin-Huxley models in parallel. The computational cost of parallel processing was due mainly to communication between threads. Tests identified two issues that determined the benefits from parallel processing: load balancing and model granularity. Load balancing relates to how many threads run efficiently on a single processor. Model granularity relates to how many ODEs are solved per thread.

Comparison of computation times for solving an equivalent number of ODEs, for multiple threads of uncoupled HH models (4 ODEs) with communication, and multiple threads of coupled HH models (4 ODEs) without no communication among threads, and 4 threads with multiple (4-50) coupled ODEs (red). The green curve represents batch processing corresponding to the green curve in the upper figure. The difference between the blue and green curves represents batch processing corresponding to the green curve in the upper figure. The difference between the blue and green curves in the upper figure displays the gain due to parallel processing. With communication, the overall gain (difference between blue and red curves) increased for larger numbers of ODEs, due to the increase in the ratio between computing time versus communication time increases.

Gain from batch processing on a dual Xeon (hyperthread) computer. The number of compartments was systematically increased. With sequential processing (i.e., all compartments simulated on a single thread, blue), the time required to complete a simulation increased linearly with a slope of 15 sec/compartment. However, with proper granularity (red) the benefits of parallel processing are realized, i.e., the ratio of computing time versus communication time increases.

Supported by NIH grants R01 RR11626 and P01 NS38310.