SE 390 - User Manual
Brain Simulator Parallelization

Prepared by
Team Spike
Greta Cutulenco - gcutulen
Robert Elder - relder
James Hudon - jbhudon
Artem Pasyechnyk - apasyech

Presented to
Dr. Paul Ward

University of Waterloo
Software Engineering
December 14, 2012

Glossary

Amdahl's law: A mathematical relationship describing the maximum realizable speedup due to parallelization of a computation.

Horizontally Scaling: To scale horizontally means to add more nodes to a system as opposed to scaling vertically, which means to add resources to a single node in a system.

Nengo: A graphical and scripting based software package for simulating large-scale neural systems.

NEF: Nengo uses the Neural Engineering Framework (NEF) to solve for the appropriate synaptic connection weights to achieve this desired computation.

NumPy: NumPy is the fundamental package for scientific computing with Python.

Spaun: A model that can see, remember, think and write using a mechanical arm.

Theano: A Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently.

Introduction

This document will describe the function and operation of a piece of software designed to simulate the workings of a human brain. This software will be developed in collaboration with Terry Stewart from the Center for Theoretical Neuroscience at the University of Waterloo. A large portion of this software has already been written and is actively being used for neuroscience research. Our team's focus is concerned with the task of optimizing the software to increase the performance and, ideally, allow the system to operate faster with a larger number of neurons by scaling horizontally instead of vertically. Since this piece of software is used as a research project, the end user is expected to have a considerable amount of knowledge about software systems. This also means that many of the high-level requirements for this project have unusually specific and technical descriptions that would otherwise not be found in the requirements analysis for an ordinary consumer product.

Use Cases

Customer Use Cases

The most important use case for this software is for use in brain simulations by Terry Stewart's team. This software will be used in a variety of ad hoc simulations that give empirical evidence to hypotheses in neuroscience. Some examples of specific tasks that have been mentioned by the customer include navigating a robot, simulations of machine vision, and optical character recognition.

These ad hoc tasks will be requested by the user in the form of a network topology description, which describes the layers of neurons that exist in the model, and how they are connected to each other.

Community Use Cases

This project is licensed under the Mozilla Public License 1.1, which seeks to balance the concerns of proprietary and open source developers. As a result, it is likely that some people

will want to use this product for commercial applications. This introduces a number of use cases which are difficult to predict.

In order to promote the use of this software by others in the open source community, the most important use cases are successful software installation and easy adaptation to a particular application of brain simulation. This could be any of the traditional applications of machine learning that are used today, such as optical character recognition. Finally, it is also meaningful to consider the ease with which any member of the community can contribute to, or fork this software project to improve it.

Description of Use Cases

Integration With Graphical Interface

An important use case is the ability to interface with the low level code which drives the simulations of neurons from a high level interface such as Nengo. Creating a brain simulation usually consists of a large number of similar operations, such as creating large arrays of neurons, and forming connections between them in a way that is often repeated. The ability to visualize this information at a high level is very important in this research area because the alternative would require inspecting large files with thousands of numbers representing neuron output.

Distributed Deployment

The goal of our team's contribution will be to significantly increase the speed of the simulations offered by this software. We anticipate that the most meaningful use case will be to simulate many millions of neurons using a distributed computation. In order to be practical, this use case must be executed using a minimum amount of interaction from the user. In practice, distributing a computation will require obtaining the network addresses for a large number of nodes and then deciding how to break up the work, where to send it, and how to recombine it. This task can be non-trivial and tedious if manual intervention is required, so this use case precludes the user from needing to specify all this information.

Operating Specifications

Inter-node Latency

Any acceptable solution which implements a distributed model must be capable of inter-node communication not exceeding a time on the order of 1 millisecond. The reason for this is related to the current model of processing used to simulate the brain. The part of computation which can be parallelized is the computation of one 'tick' or time quantum of data, which is used to calculate the results of the next time quantum. Current benchmarking shows that for a small network size, the time taken for one tick is about 3 milliseconds. This means any network latency which is on the order of 1 millisecond will significantly reduce the opportunity for speedup due to parallelization of the computation. This is mathematically described by Amdahl's law.[1]

Other configurations have also been tested where more latency could be tolerated. For example, a model of three million neurons runs at about 1000 milliseconds per 'tick'. Thus, if the latency of the distributed system is 100 milliseconds, we would be getting a factor of ten performance increase in the best case. Further research is needed to determine if there are other bottlenecks or caveats for this extreme use case.

Programming language

An ideal solution will be able to interface with the Java brain simulator which already has significant support on this project. There is also another team which is focusing on improving the graphical user interface which further marries us to use Java for at least some part of the project. With respect to the low-level computational part of the model, there are currently two python based models which are faster than the full Java implementation. One is fully written in Python, and the other uses Python with theano. Theano is used to pre-compile Python code to run as C code for a significant performance gain. Preliminary research suggests that it may be practical to re-implement some of the theano based model as pure C code when implementing it as a horizontally scaling distributed model. This decision will need to take into consideration the current familiarity that members of the research team have with the Python based model, and the potential loss of speedup that will be incurred from having a slightly slower serial portion of the program. In summary, this project must be written at least partially in Java and in theano-based Python or pure C for the neuron computation.

Aside from these adaptors, there will likely be a system that acts as an administrator. Its task will be to coordinate the distribution of the work involved in modelling the brain. One potential framework we can use to enable communication between nodes delegated by the administrator is ZeroMQ. ZeroMQ is a high-performance messaging library that has bindings in many languages, including the ones previously listed. Furthermore, since the delegation work is likely to occur only at the start of the simulation, we would want to pick a language that is optimal for development time and not for execution time.

[1] Amdahl, Gene (1967). "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities" (PDF). *AFIPS Conference Proceedings* (30): 483–485.