SIRNEM: A Parallel Neural Network Simulation System

Wagner Meira Jr * Marcio L B Carvalho †

March 10, 1992

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal 702
30.161 - Belo Horizonte - Minas Gerais
Tel: +55-31-443-4088
Fax: +55-31-443-4352

Abstract

In this article, we present SIRNEM, a system to simulate the behavior of neural networks using a message-passing multicomputer. The problem of designing an efficient inter-neuron communication algorithm is addressed, this algorithm should also enable a reliable exchange of information, avoiding unpredictable situations as deadlock and starvation. A brief description of the main modules of SIRNEM are presented.

1 Introduction

In the context of computer science, the expression neural networks has been used to describe systems that have a great number of simple processors. These processors communicate among themselves by sending their activation values to every processor it is connected to; these values are also called the state of processor. A weight is associated to each connection, showing how significant the transmitted information is. The behavior of these systems are a function of the interaction between the processors through their connections. A motivation to investigate these systems comes from the analogy between them and the human brain.

Although there is available some theoretical development about neural networks, they are not sufficient to the completely describe the behavior of these systems, so the technique most frequently used to understand the behavior of neural networks is simulation. As in other computational models, it is important that researchers can implement and test their ideas. There is a great number of models of neural networks, each of them developed for different specific applications. when using neural networks to a new application it is necessary to simulate the models. There are not, until nowadays, general use neural computer so parallel simulation of neural networks is a "realistic" good and cheap way to study these systems.

An important feature of the neural networks is that they perform a great number of parallel computations [FH83], memory is distributed, processing is asynchronous and systems have a natural fault tolerant capacity [HMT86]. Parallel neural network simulation is much faster and realistic than the sequential one. But a distributed simulation comes with problems inherent to network models particularities.

Writing simulation programs for these models is a hard and error-prone work mainly because of the difference between usual programming languages primitives and the basic operations of neural networks. These simulations have great computational cost and a way to minimize it is to explore the parallelism of the neural network systems in a parallel simulation.

*E-mail:MEIRA@DCC.UFMG.BR
†E-mail:MLBC@DCC.UFMG.BR

In this article, we present an environment for simulation of different models of neural networks in a multiprocessing architecture. In the next section, the question of the inter-neuron communications reliability is studied. In the third and fourth sections we describe the simulation of neural networks; in the third we present how we can specify a neural network for simulation and in the fourth how we perform a simulation with SIRNEM.

2 Inter-neuron communications

Neural networks are discrete dynamical systems whose behavior is completely specified in terms of a local relation; time advances in discrete steps and the laws for these cells are expressed by a single recipe through which at each step each neuron computes its new state from that of its neighbors.

We identify an important property of neural network: The cells of the network work synchronously, at each step all the neurons must obtain the state of their neighbors to determine their next state. To correct simulate a neural network, the inter-neuron communications must be performed in such a way that guaranties the maintenance of the above mentioned property. We will now discuss some alternatives to solve this problem:

Establishing a synchronizing process may be the simplest solution to this problem; this process waits until all the neurons are ready and only then permits that they change their states [FFGL88]. A disadvantage of this policy is that it creates a bottleneck, limiting the parallelism of the system.

Another solution is to implement the synchronization in a distributed way [BL90], by the neurons; it is called implicit synchronism. This option is better than the previous because it allows more parallelism and neurons that do not have dependencies, may operate in parallel. A problem that arises is to determine what neurons may do so. To solve it we need to study the communications in a neural network and what conditions are necessary to satisfy the property described above.

We now present a solution that deals the synchronization of simulating asynchronous operation of neural networks, which can be found in the literature [GB81, CM84, BG89]. We will give an overview of this solution, but, before, we need to define some terminology:

The neural network may be viewed as a directed graph where the vertices are associated to the neurons and the edges to the connections between the neurons. The direction of each edge is the same as the information flow. So, there is an edge from vertex i to vertex j, if the state of neuron i influences neuron j.

It is possible that two neurons mutually influence each other. In this case there is an edge from i to j and another one in the reversed direction ($j - i$). An example of this case is the Hopfield's neural networks [HT86], where each neuron is connected with every other in the network. We observe that for each pair of connected neurons, only one of then may operate in the same step; if both were allowed to operate, there could be unpredictable situations as deadlock, starvation or an inconsistent network.

We can classify the communication between any two neurons (i and j) of the network in two types:

1. Bidirectional: there is an edge from i to j and another from j to i;
2. Unidirectional: there is an edge from i to j and there is not an edge from j to i;

The algorithm now being described makes the following assuption about the neural network to be simulated:

- The neural network to be simulated has only bidirectional connections.
- Each discrete step is composed by one or more substeps, and in each of them a group of independent neurons obtain activation values from neighbors. Because information flow through each connection only in one direction at a given substep, the direction of each

neuron to neuron communication in a substep k can be represented by a directed edge and the graph formed by these edges is the subgraph G_k, where there are only one edge between each connected neurons.

If G_k is acyclic, there can not be any unpredictable situations in the information exchanges of the network. It is shown in [GB81, CM84, BG89].

The main aspect of this solution is that it guaranties that each neuron communicate to its neighbors in a step. This can be achieved by maintaining G_k acyclic during the evolution of the network. If this condition is satisfied, there is at least a sink [1] node in G_k that has received all the information it needs, so it can calculate its next state.

After the sink has calculated its next activation value, this value can now be sent to its neighbors so they can calculate their new states too. As each sink computes its new state, it does not need any more information about the current step, so it reverses its edges and turns into a source.

At each substep, at least a neuron turns into a source and after all the neurons have turned into a source a step has been completed.

There are some neural networks with neurons that have bidirectional and unidirectional connections (we call them hybrid neurons) [BN89]. If this class of neural network uses the algorithm described above, it can cause unpredictable situations because the direction of some of its connections must be reversed and others can not to complete a step.

We propose an extension to the previously presented algorithm to simulate hybrid networks as well. In our solution, a neuron that have unidirectional connection will only change its state when it has turned into a sink and has received all the information from its unidirectional connections. Neurons with only bidirectional connections are treated in the same way as the original algorithm.

As we have some fixed direction edges, the algorithm is not able to guarantee that every instantaneous subgraph G_k is acyclic. So we must verify, from the definition of the network's topology, whether there is the possibility of cycles that include a fixed direction edge. A way to detect this kind of cycle in a graph G is verifying if the graph has a strongly connected component [2] which contains a fixed direction edge, which is the only situation that may have cycles.

3 Neural network simulation

We can identify two main activities in a neural network simulation: Initially we must define the model to be simulated and using it, we perform simulations, applying different inputs to the network and verifying its outputs.

When we are simulating these systems, a difficulty that arises is the great functional and topological differences among the models. It justifies the necessity of a flexible way to describe and simulate neural networks.

Another important feature to this kind of simulation is its homogeneity. Usually, the neurons of a network are functionally identical. We specify each neural network in three levels of hierarchy:

Neuron: a type of a neuron is based on a functional specification of a defined cell, its definition is completed with the exactly number of inputs, outputs and parameters.

Cell: in this level, we define how a cell of the network will work. This is done by some C code. Each cell must receive as parameters its number of inputs, outputs and some initial parameters.

Network: a neural network is composed by some interconnected neurons. In this level, must be defined the type of each neuron, its connections for input and output and any parameters it needs to work.

[1] **Sink:** a vertex were all edges are directed inward.

[2] **Components:** set of vertices of a graph with all the edges that connect any two vertices of the set. **Strongly connected components:** component where there are two paths between any two vertices. [Szw84]

From one neural network definition we may need to execute a large number of various simulations. Each of them may contain its own features, so it would be usefully a flexible way to specify the simulations is available and interact with the network to obtain data to perform the simulation.

We then establish that the simulator must be also defined in the network specification phase, with all of its inputs and outputs defined and connected. The dealing of information exchanged between the simulator and any neuron must be done in a separate way from the inter neuron. These connections must be dealt in a explicit way in the neuron definition and declared with the type of the neuron.

4 SIRNEM

SIRNEM is an environment for simulation and study of neural networks. The user must specify how his network works (as outlined above) to generate a simulation. The environment deals with aspects like synchronization, allocation of processes into processors and the application generation.

To simulate a network, the user perform the following steps:

1. Definitions of the cells;
2. Definitions of neurons;
3. Definition of network;
4. The C code for the application is generated assuming one process per neuron and one for simulator. All the inter-process connections are specified. Code generation and the following operation are performed simultaneously:
   - It is verified whether the network has potential cycles, as described in section 2.
   - The neuron processes are allocated to the available processors, to obtain an even load balance. As all processes have basically the same computational cost, this embedding problem can be modeled as a graph partition problem as presented in [LFT92]. In a system of k processors, G must be partitioned in k subgraphs with a minimum number of connections among subgraphs. This problem is NP-complete and our solution uses Hopfield's neural networks as described in [LFT92].
5. All the code generated is compiled and the resultant application is loaded and executed in the target machine.

SIRNEM is implemented using facilities provided by another environment [MJSV+92] that solves some communications restrictions and allows all processes to use I/O resources.

By the time we are concluding this article, SIRNEM is already implemented and is being tested. We will include simulation results in the full version of the papers.

5 Conclusion

It is presented an algorithm to avoid unpredictable situations like deadlocks and starvations in a parallel neural network simulation. This algorithm can easily be extended to simulate not only neural networks but also any kind of cellular automata.

A neural network simulation environment is also presented. To perform the simulations, it needs only the description information about the network to generate a complete application program. This application has included the described protocol and other features as a balanced allocation of processes on processors and tools to monitor all the work of the neural network.

References

[BG89] V.C. Barbosa and E.M. Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Transactions Programming Languages and Systems*, (11):562–584, 1989.

[BL90] V.C. Barbosa and P.M.V. Lima. On the distributed parallel simulation of hopfield's neural networks. *Software Practice and Experience*, C-20(10):967–983, October 1990.

[BN89] R. Balaniuk and P.O. Navaux. Reconhecimento de padrões codificados em seqüências utilizando redes neurais. In *Anais do Sexto Simpósio Brasileiro de Inteligência Artificial*, pages 359–372, Novembro 1989.

[CM84] K.M. Chandy and J. Misra. The drinking philosofers problem. *ACM Transactions Programming Languages and Systems*, (6):632–646, 1984.

[FFGL88] J.A. Feldman, M.A. Fanty, N.H. Goddard, and K.J. Lynne. Computing with structured neural networks. *Communications of the ACM,*, C-31(2):170–187, February 1988.

[FH83] S.E. Fahlman and G.E. Hinton. Massively parallel architectures for ai: Netl, thistle and boltzman machines. In *Proc. National Conference on Artificial Inteligence*, pages 109–113, 1983.

[GB81] E.M. Gafni and D.P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, (29):11–18, 1981.

[HMT86] D. Hammestrom, D. Maier, and S. Thakkar. The cognitive architecture project. *Computer Architecture News*, (14):9–21, 1986.

[HT86] J.J. Hopfield and D.W. Tank. Computing with neural circuits: a model. *Science*, 233:625–633, 1986.

[LFT92] K.C. Lee, N. Funabiki, and Y. Takefuji. A parallel improvement algorithm for the bipartite subgraph problem. *IEEE Transactions on Neural Networks*, C-3(1):139–145, January 1992.

[MJSV+92] W. Meira Jr, J.P. Salles, M.A.G. Vieira, A.I. Tavares, M.L.B. Carvalho, and O.S.F. Carvalho. Programação paralela em arquiteturas transputer. Technical Report 002/92, Departamento de Ciência da Computação - UFMG, 1992.

[Szw84] J. L. Szwarcfiter. *Grafos e algoritmos computacionais*. Editora Campus Ltda., 1984.