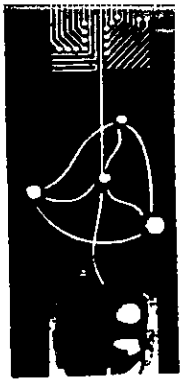

1° Congresso Brasileiro de Redes Neurais

Escola Federal de Engenharia de Itajuba
Itajuba, 24 a 27 de outubro de 1994

HENNS: Heterogeneous Environment Neural Network Simulator *

Wagner Meira Junior† André Luiz de Senna ‡ Márcio Luiz Bunte de Carvalho §

Departamento de Ciência da Computação – UFMG
Caixa Postal 702 – Belo Horizonte – MG – Brazil – CEP 30.161-970
Tel: +55.31.443-4088 Fax: +55.31.443-4352

Key-Words: Neural Networks, Simulation, Heterogeneous and parallel processing

Abstract

The simulation of artificial neural networks (ANN) is a hard task due to two reasons: the complexity of ANN specification and the computational cost associated. The first problem has been addressed by some simulators by allowing specification of ANNs. Parallel processing has been recognized as a solution for the second problem. This work presents a neural network simulation strategy that runs in a heterogeneous environment, that combines different types of parallel processors and servers. This strategy is implemented in the HENNS, an environment that allows the user to describe and simulate various neural network models in parallel and heterogeneous environments. The results show that the use of heterogeneity is better than the use of parallel architectures alone.

1 Introduction

Artificial neural networks are models based in the human brain, which works via the interaction of a great number of appropriately interconnected computationally simple nodes, which mimics the behavior that is believed of the neurons. A neural network stores its information on the weights associated to the connections between pairs of nodes and in the activation level of each node. Using simple algorithms, neural systems are reliable and capable of dealing with a great amount of information. However, their simulation shows to be a hard task mainly due to two reasons: the functional complexity of the whole and its high computational cost.

The implementation of the models has been facilitated by the development of various general-use neural network simulators. These tools allows the simulation of neural networks providing model's flexibility and functioning abstraction.

On the other hand, more efficient neural network simulators alternatives are found in the literature, for example, using hardware implementation and/or parallelism. Hardware implementations are the fastest, but has smaller flexibility; parallel implementations are less efficient but more flexible, a more suitable approach for scientific investigation.

This work presents HENNS, a software neural network simulator intended to run on an heterogeneous and parallel environments. This system provides tools that enables the user define the neural network model to be simulated and various parameters that specify the simulation itself. From this specification, the simulator generates an appropriate C source-code, compiles and links it building an executable code that performs the simulation.

This paper has six sections: this is the first, an introduction; the second reviews some previous work in sequential and parallel neural network simulation; section three describes the computing enviroment used; the fourth and fifth sections presents HENNS and some results achieved with its utilization; the conclusions and future work are presented in the last section.

*This work was sponsored by FAPEMIG (proc. TEC 1113/90) and CNPq (proc. 502353/91-0(NV)).

†meira@cs.rochester.edu
‡senna@dcc.uimg.br
§mlbc@dcc.uimg.br

2 Neural Network Simulators

2.1 Sequential Simulators

The sequential neural network simulators in the literature can be divided up into two classes. The first simulators are batch-oriented; a neural model is completely specified and then all the simulation is performed: in this case, the user can not interact with the simulation in any way. PABLO [Perkel, 1976] and BOSS [Wittie, 1978] are examples of these simulators.

The second class is the interactive simulators, which enables the user to modify (and/or visualize graphically) the simulation while it is being performed. In these simulators, the network to be simulated is specified via a description language, which is either compiled or interpreted to perform the simulation. In the case were graphical interface environment is used, primitives that allows changes and verification of the simulation events are commonly provided. In this class we can cite ISCON [Small et al., 1983], GENESIS, Asprin, MIRRORS/II [D'Autrechy et al., 1988], SFINX [Skrzypek and Mesrobian, 1992] and PLANET [Miyata, 1991]. Although they work similarly, these simulators differ from each other in the complexity and flexibility in specifying the neural network to be simulated; we find from analogical to particular model simulators. However, all of them have the common goal of joining abstraction with flexibility of constructing neural networks. A more complete reference on these simulators and their features can be found in [Meira Jr., 1993b].

2.2 Parallel Simulations

The inherent parallelism of neural networks has motivated their simulation on parallel machines. MIMD and SIMD machines are the most popular target architectures. In each of these architectures we find different problems to be addressed, as it will be shown below.

Parallel implementations explore the processing independence of the neural networks nodes, i.e. all nodes that has all its necessary inputs may start its processing phase. The major problem in these simulations is how establish efficient communication among the nodes.

In a MIMD architecture this problem is harder: to achieve efficiency, we must also guarantee load balance amongst the processes and the correct arrival sequence of the messages exchanged between the asynchronous processors. We find several proposals in the literature, which have been implemented in various architectures such Intel iPSC Hypercube, BBN Butterfly, SBC and mainly Transputers. We can conclude that the most important factor in a MIMD implementation is the communication costs, i.e. the interconnection pattern of the network to be simulated is more significative than its local processing.

The use of SIMD architectures in neural network simulation presents new variables. An important one is the architectural restrictions of these machines. Examining related works, where implementations on machines such Zephyr, DAP, CM-2 and Hughes SCAP are described, we can conclude that SIMD implementations are extremely machine dependent, in spite of some general mapping strategy proposed [Salles et al., 1992].

A interesting proposal is given by Nordstron [Nordström and Svensson, 1992], who affirms that the MIMSIMD architectures are the most suitable for neural network simulation. His proposal and the problems experienced during the simulation of neural networks on conventional parallel architectures has motivated us to investigate the use of heterogeneous architectures in the simulating neural networks.

3 The Computing Environment

In order to verify the suitability of the neural networks simulation in distributed and heterogeneous processing environments, we used two groups of resources available at DCC-UFMG: (i) a Sun workstation network with the PVM package; (ii) the Zephyr Wavetracer SIMD machine; they are described below:

PVM: PVM (Parallel Virtual Machine)[Geist et al., 1993] is a software package that allows an heterogeneous network of parallel and serial computers to appear as a single concurrent computational resource. PVM consists of two parts: a daemon process that any user can install on a machine, and a user library that contains

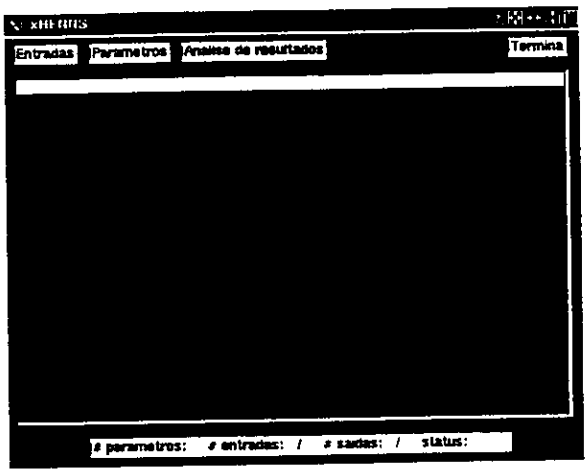

Figure 1: xHENNS initial screen

routines for initiating processes on other machines, for communicating between processes, and changing the configuration of machines.

Zephyr-Wavetracer: the Zephyr Wavetracer (WT) is a 8,192 1-bit processors SIMD machine. Each processor have two kinds of memory: (i) an internal cache memory of 256 bytes; (ii) an external one of 32 kbytes. Another interesting feature is that the number of active processors is not limited to the quantity of physical processors, because it is allowed to create virtual processors. This machine works as a Sun co-processor and there is an ANSI C language extension, the multiC, to access and use it. So, the executable code is executed sequentially in the host (a Sun workstation) and, when a multiC instruction is found, it is performed in the Wavetracer.

4 HENNS

HENNS [Meira Jr., 1993b. Meira Jr., 1993a], developed at DCC-UFMG [1], is a neural network simulator in which the user can define topologies and neural network models arbitrarily. HENNS provide also means of specifying simulation parameters, such as the computing environment: sequential, distributed and vectorized. In the sequential form, all of the network parts are combined in a single executable program. In the distributed and vectorized executions, the network and each of its components that must be simulated by itself corresponds to a PVM process. The vectorized programs uses the Zephyr-Wavetracer primitives. In the distributed manner if operation, each of its sequential processes are implemented as PVM processes.

The development of neural applications using HENNS can be divided into two phases: application and neural network. The application works as a front-end of the network and can be freely modified by the user. The network can be seen as a "black–box" which receives inputs and parameters and returns outputs and parameters. The protocol between application and network is simple and implemented via PVM messages.

Included in the HENNS package there are two examples of application software, the first one, called CHENNS, is a ascii based interface to HENNS generated neural networks; the second, xHENNS has the same functionality of the former with a xWindows interface. The initial screen of the later can be seen in the Figure 1.

The neural network implementation using HENNS, can be divided in three steps: (i) definitions of the neural architecture and its simulation using the description language of HENNS; (ii) run HENNS to read the definitions and generate a C code that performs the simulation; (iii) compile and execute this code.

Following, each of these steps will be described in details:

4.1 Neural Architecture Definition

In this step, the user specifies his simulation using HENNS' description language. This specification is made in two levels:

Structural: in the HENNS, a neural network is defined as an hierarchical structure of three levels: (i) **Network**: represents the neural network as a processing unit for the external tasks and serves as basis of the various cluster's functioning; (ii) **Cluster**: it is an abstraction that join similar nodes, which have the same input and output connections and work similarly. In the back-propagation [Rumelhart et al., 1986] neural network model, for example, each level of the network may be seen as a cluster; (iii) **Node**:

[1] HENNS is available via *anonymous* ftp at dcc.ufmg.br

it's the smallest functional unit of a neural network, the neuron.

To completely define the structure, one needs also to define the connections between the network parts. In HENNS, these connections can be between the network and some cluster or between clusters. It is not permitted intra-clusters connections due to efficiency and parallelization considerations. Because cluster is a abstraction proposed by HENNS, this should not cause any problem.

Functional: the definition of the simulation behavior is made in a procedural way. The neural network (and the other hierarchy parts) may work based on various operation modes, each defined as a list of commands drawn from a set of primitives that deal with various aspects, some of them are:

- Protocol: they implement easy exchange of information between the network hierarchical parts, independently of the architecture and machine under use;
- Control Flow: these are commands that control the execution flow: while e if, that makes the procedural language more powerful and flexible;
- User-defined functions: any function related to the specific internal behavior of each network part in the various operation modes are defined by the user. This definition is made using a target machine language, to access HENNS hierarchical data structures and library functions.

4.2 Source-code Generation

In this step, based on the specification files, various information are generated:

Definition: constants and other network specification definitions;

Initialization: functions that performs network components data and protocol initializations;

Operation: functions that implement the network work;

Compilation: specifications and compilation dependencies of the network and its components, described using the syntax of a Unix program called make;

Simulation: a Unix *shell script* with the steps necessary for the execution: *daemons* instantiation, compilation and the execution of the simulation.

4.3 Source-code Compilation and Execution

In order to compile the generated application, the Unix program make is invoked and everything is performed automatically: the initialization and operation source-codes generated are compiled and linked to the HENNS library. If the simulation is to be performed sequentially, only one executable code is generated. If a network with three clusters should be performed distributed, four executable codes are generated: one for the network and one for each cluster. If there are vectorized processing, the multiC compiler is used and an executable is generated as it is done in the distributed way.

The simulation is the execution of the network code. When the simulation is distributed, the network program initiates the others, performing all the initialization and termination control of the "child" processes.

5 Results

The results presented next are part of an investigation of the most suitable architecture for neural network simulation in the DCC-UFMG hardware environment. In order to perform the measures, a back-propagation neural network was specified and his topological characteristics changed during the various tests.

5.1 Test Environment

The tests presented below were performed in the Sun network of the DCC: the SIMD machine Zephyr-Wavetracer was attached to a Sun Sparc Station 2 also connected to the DCC network. To perform the tests presented below, we used a back-propagation neural network with three levels as described in [Hertz et al., 1991]. Concerning to

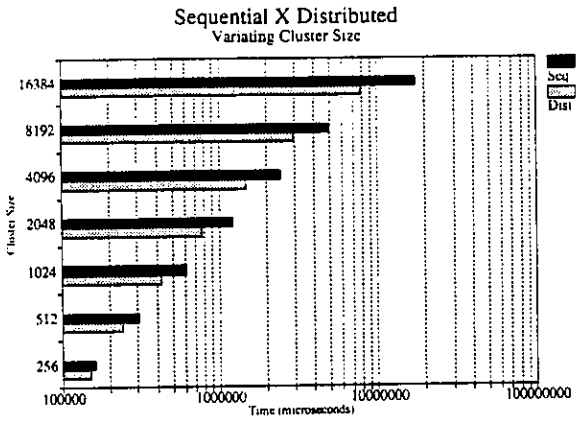

Figure 2: Sequential × Distributed

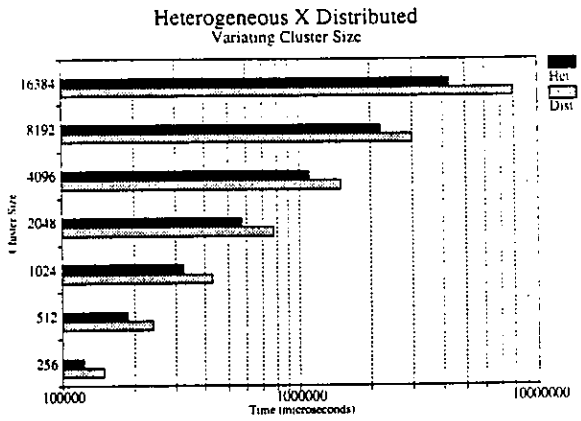

Figure 3: Heterogeneous × Distributed

machine configurations we have used the following:

Sequential: the simulation was performed in only one workstation. It was used two models of SUNs: Sparc Station 2 and Sparc Station SLC;

Distributed: The clusters or parts of clusters are executed in SLC machines, one per machine:

Vectorized: It was performed by two processes in the WT host machine: (i) the vectorized cluster to be simulated: (ii) an executable that performs the remaining parts of the network sequentially. This differentiation was necessary due to the fact that the programing language used in vectorized implementation (multiC) is different from the one used by the sequential;

Heterogeneous(distributed/vectorized): in these simulations, it was used three kinds of executable code-machine pair: (i) the network program executed in a Sparc 2; (ii) the vectorized cluster performed in the WT host; (iii) the other clusters performed in SLC machines. one per machine.

Below, we will present a comparison among the various machine configurations, varying some of the topological parameters of the network simulated. A more complete set of tests can be seen in [Meira Jr., 1993b]. Although, we won't present any result using a pure vectorized configuration. it is described for completeness purpose.

5.2 Distributed × Sequential

In this test we verify the suitability of distributed processing to simulate clusters with various number of nodes. So we create neural networks where the input and output levels have always the same number and variate the length of the hidden level. Both the simulations, sequential and distributed, are performed in Suns SLC.

The plot of Figure 2 shows the results for clusters that varies from 256 to 16384 nodes. The distributed executions were clearly better than the sequential ones. The speedup varies from 10% (for the smallest cluster) to 70% (for the biggest cluster). this can be explained by the overhead of the communication between the processes.

5.3 Distributed × Heterogeneous

After we verified that the distributed and vectorized simulations are better than the sequential ones. we investigates the gain due to the using of both solutions simultaneously. The WT host machine is used as a node in a distributed simulation. configuring an heterogeneous environment.

In the graph of Figure 3 we can see the performance of the heterogeneous and distributed simulation. The heterogeneous simulations was always better with the speedup varying from 20 to 80% with the size of the hidden cluster.

6 Conclusion and Future Work

This works has presented a strategy for simulating neural networks in heterogeneous environments

and an ANN simulator that adopted this strategy, HENNS.

Using the Sun network of DCC-UFMG and the SIMD machine Zephyr-Wavetracer, we verified the suitability of distributed and vectorized simulations in that computing environment, followed by the analysis of our strategy. In this analysis, we noted that the proposed strategy allows performance enhancements proportional to the size of the network simulated. This can be explained by communication costs associated to these implementations. Quantitatively, the distributed simulation time was 80% greater than the heterogeneous time in the best case. If we consider sequential and heterogeneous time, this difference increases up to an order of magnitude.

Future work includes simulation of other ANNs models and the extension of HENNS for different computing environments. The enhancement of the application tools is also a goal, allowing the user not only simulates his neural network, but also visualize and tune the executions on the fly.

References

[D'Autrechy et al., 1988] D'Autrechy, C. L., Reggia, J. A., Sutton, G. C., and Goodall, S. M. (1988). A general-purpose simulation environment for developing connectionist models. *Simulation*, 51(1).

[Geist et al., 1993] Geist, A., Benguelim, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1993). *PVM 3.0 User's Guide and Reference Manual.* ORNL/TM-12187 Oak Ridge National Laboratory.

[Hertz et al., 1991] Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*, volume 1 of *Computation and neural systems series*. Allan M. Wylde.

[Meira Jr., 1993a] Meira Jr., W. (1993a). Guia do usuário e manual de referência do HENNS. Technical Report RT013/93, DCC-UFMG.

[Meira Jr., 1993b] Meira Jr., W. (1993b). Implementação de redes neuronais em ambientes paralelos. Master's thesis, Departamento de Ciência da Computação - UFMG, Belo Horizonte, MG.

[Miyata, 1991] Miyata, Y. (1991). *A User's Guide to PlaNet Version 5.6.* University of Colorado at Boulder.

[Nordström and Svensson, 1992] Nordström, T. and Svensson, B. (1992). Using and designing massively parallel computers for artificial neural networks. *Journal of Parallel and Distributed Computing*, (14):260 – 285.

[Perkel, 1976] Perkel, D. H. (1976). A computer program for simulating a network of interacting neurons. *Computers and Biomedical Research*, (9):31 – 43.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. MIT Press, Cambridge, MA.

[Salles et al., 1992] Salles, J., Vieira, M., Meira Jr., W., and Carvalho, M. L. B. (1992). Implementação de redes neuronais em máquinas simd. In *IV Simpósio Brasileiro de Arquiteturas de Computadores e Processamento de Alto Desempenho*, pages 347 – 361.

[Skrzypek and Mesrobian, 1992] Skrzypek, J. and Mesrobian, E. (1992). *UCLA-SFINX - Simulating Structure and Function in Neural Connections.* Machine Perception Lab., UCLA.

[Small et al., 1983] Small, S. L., Shastri, L., Brucks, M. L., Kaufman, S., Cottrell, G. W., and Addanki, S. (1983). Iscon: A network construction aid and simulator for connectionist models. Technical Report TR 109, University of Rochester.

[Wittie, 1978] Wittie, L. D. (1978). Large-scale simulation of brain corticies. *Simulation*, 3(31).