A VIRTUAL MACHINE MODEL FOR ARTIFICIAL NEURAL NETWORK PROGRAMMING

Pierre Bessière[1], Ali Chams[2] & Traian Muntean[3]

ABSTRACT

This paper introduces the model of a virtual machine for A.N.N. (Artificial Neural Networks).

The context of this work is a collaborative project to study new V.L.S.I. implementations and new architectures for neuronal machines[4]. The work consists in the specification and a prototype implementation of a description language for A.N.N., of the associated virtual machine, of the compiler between them and of the compilers mapping the virtual machine on different highly parallel computers.

In this short paper we present the virtual machine model which combines the features of various parallel programming paradigms. Our model allows, in particular, to have the same A.N.N. program running on both synchronous or asynchronous type of machines. In this framework a parallel architecture (S.M.A.R.T.) and a dynamically reconfigurable parallel machine of Transputers® (SuperNode) are considered as target machines.

---

[1]L.G.I./I.M.A.G. : Laboratoire de Génie Informatique / institut d'Informatique et de Mathématique Appliquée de Grenoble. Adress : B.P. 53X, F-38041 Grenoble, FRANCE. Phone: 33-76.51.45.72. E-mail: bessiere@imag.imag.fr.

[2]L.T.I.R.F. : Laboratoire de Traitement d'Image et de Reconnaissance des Formes / institut d'Informatique et de Mathématique Appliquée de Grenoble. Adress : 46 ave Felix Viallet, F-38031 Grenoble cedex, FRANCE. Phone: 33-76.57.45.50.

[3]L.G.I./I.M.A.G. : Laboratoire de Génie Informatique / institut d'Informatique et de Mathématique Appliquée de Grenoble. Adress : B.P. 53X, F-38041 Grenoble, FRANCE. Phone: 33-76.51.48.64 E-mail: muntean@imag.imag.fr.

[4]The work described in this paper is part of an ESPRIT-B.R.A. (Basic Research Action) project named NERVES (NEurocomputing Research and Vlsi for Enhanced Systems).

I. INTRODUCTION

We will enhance mainly the concepts of a virtual machine which combine various programming paradigms encountered in ANN.

The introduction of a virtual machine has several advantages:

- the definition of the virtual machine instructions obliges to clarify the fundamental concepts and their semantic properties;
- the programming language for A.N.N. is independent of the actual target machines;
- only the compiler between the virtual machine and the target machine should be written when a new target machine is considered.

In a first phase two target machines have been selected for experiments: S.M.A.R.T. (Sparse Matrix Adaptative Recursive Transforms) and SuperNodes of Transputers®.

SMART is a parallel machine dedicated to A.N.N., which is under development at L.T.I.R.F. (Laboratoire de Traitement de l'Image et de Reconnaissance des Formes). It has been designed to be especially efficient for sparse matrix operations which are some of the basic computations needed for A.N.N. For further descriptions see: [Jutten90] & [Lawson90].

SuperNode is a general purpose, dynamically reconfigurable, parallel machine based on Transputer®. The underlying programming paradigm is C.S.P. [Hoare85]. SuperNode machines have been developed in the ESPRIT project 1085 between 1986 and 1988. For detail descriptions see: [Harp86] & [Muntean89].

II. MOTIVATIONS AND MAIN DIFFICULTIES

II.1. Motivations

A.N.N. are intrinsically highly parallel. However, most existing implementations are still sequential. Numerous works are under way to build parallel machines dedicated to A.N.N..

Such parallel computers need a programming environment. They need especially, user friendly interfaces and powerful programming languages.

The existence of such a programming environment is essential for a practical use of parallel computers to emulate A.N.N.

II.2. Main difficulties

Two main difficulties have been encountered:

- The design of a programming language should be considered independently of the features of the different computers. Classical sequential architectures, even if they lead to different computers, they share the same underlying programming paradigm. In parallel computing the situation is quite different, there are several very different parallel programming models due to various possible control structures of concurrency; synchrony or asynchrony of process behavior is to be considered, nondeterminism due to multiple execution paths can arise. A hard problem for a parallel programming language for A.N.N. is to make a synthesis between those different parallel programming paradigms.

- Usually, A.N.N. have simple treatment processes and complex interconnection schemes between them. Consequently, a programming language for A.N.N. has to propose powerful features to describe this aspect. The second difficulty has been precisely to find a right and efficient way to specify the network architecture and connectivity since it is not a common feature of usual programming languages.

This second difficulty will not be addressed in such a short paper.

III. APPROACH

III.1. Basic concepts of the virtual machine

The implementation principle is a classical virtual machine approach (see figure 1 below).

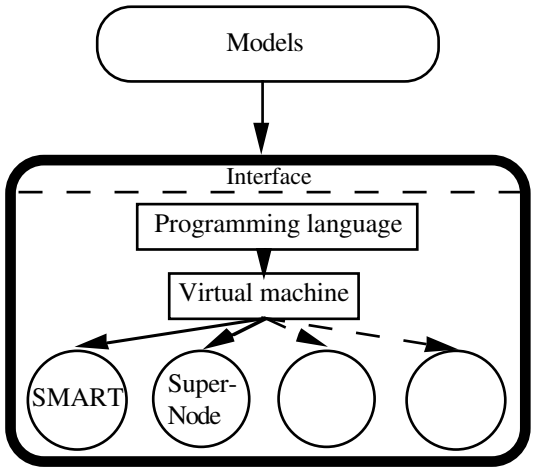

figure 1

This approach has some difficulties, especially, the clarification of the fundamental concepts when defining the virtual machine. This is one of the advanced research topics.

The synthesis between the different parallel programming paradigms is based on the following concepts introduced for the description of any A.N.N. at the virtual machine level:

- cell;
- aggregate;
- connector.

A network could therefore be represented graphically (figure 2) by using circles for cells, sets of cells for aggregates, and lozenges for connectors.

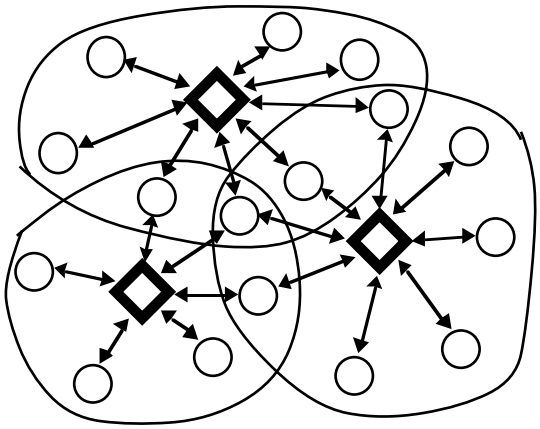

figure 2

III.1.1. Cell

A cell is a process executing a program as soon as its inputs are available. Consequently, there is no way to control explicitly the execution of cells' programs since

they are triggered by the availability of their inputs from other cells. As cells behavior is only function of their interaction and their description program, control structures of the language should reflect the kind of synchronizations attached to any interaction. This function will be devoted to connectors.

III.1.2. Aggregate and connectors

An aggregate is a set of potentially fully connected cells, whose activities are temporarily correlated.

A cell can be a member of more than one aggregate. Any cell in an aggregate can exchange information with any other cell in the same aggregate. This exchange of information can be synchronous or asynchronous. In both cases, this correlation is expressed by a protocol. The connectors are communication objects defined to implement those protocols.

The principal information, stored in a connector, is a connection matrix ***W***. If an aggregate connects N cells, the matrix size is N^2. In most cases the connection matrix is sparse, and it reflects the topological structure of the aggregate. In fact, the connection matrix represents more than communication capabilities, it contains entries (real or integer) usually called synaptic weights. These weights are used as parameters for a composition function ***C*** and an integration function ***I*** which are used to transform information exchanged by the cells; the informations should go through the same function $\boldsymbol{I} \circ \boldsymbol{C}$ before being broadcasted to cells of an aggregate. Typically, ***C*** is a multiplication function and ***I*** is a summation one. A learning function ***A*** is associated to each aggregate. For any given connection matrix and some parameters reflecting the past activity of the network, ***A*** produces a new connection matrix.

Examples:

- For "game of life" [Gardner70] a possible modelization is made of one aggregate for the all population of cells.***W*** is a very sparse matrix which encodes the 8-nearest-neighbors neighborhood. All its non zero entries are fixed to one. ***C*** is multiplication and ***I*** is summation. All the cells have the same program consisting in the usual test on the number of alive neighbors. ***A*** is obviously identity because no learning occurs in this model.
- For "competitive learning" models [Rumelhart86], two kinds of aggregates exist. One kind groups the cells of two connected levels, the other kind corresponds to the set of cells of a cluster. For the first kind, ***W*** encodes the full forward connection from level n to level n+1, weights are initialize at random values, ***C*** is multiplication and ***I*** is summation, ***A*** is the standard competitive learning function. For the second kind of aggregate, ***W*** encodes full connection inside a cluster, weights are all set to -1 but the entries of the diagonal of ***W*** which are set to n-1 if there are n cells in the cluster, ***C*** is multiplication and ***I*** is summation and each cell returns the value 1 if the result of $\boldsymbol{I} \circ \boldsymbol{C}$ is positive and 0 otherwise.
- Some realistic asynchronous simulations of parts of brain could be encoded using one aggregate for each sets of neurons possibly interesting from a biological point of view.

A basic operation of the virtual machine is the activation of an aggregate. Roughly speaking, this activation consists in broadcasting the informations delivered by every cell of the aggregate to the others, through the transformation $\boldsymbol{I} \circ \boldsymbol{C}$. This process means the automatic execution of the programs associated to all the cells of an aggregate, as soon as their inputs are available. In fact, the differences between synchronous and asynchronous models appear in this operation.

III.2. Activation of synchronous aggregate

If we suppose that each cell delivers an output, so each synchronous aggregate has a vector ***V*** representing the output values for all the cells; then v_j (*j*th component of ***V***) gives the output of the *j*th cell in the aggregate.

The composition function ***C*** can be applied for each couple (w_{ij}, v_j) to obtain c_{ij}, then the integration function ***I*** produces for all N-tuples of c_{ij} the result vector ***R***. In the case where ***C*** is multiplication and ***I*** is addition, the whole operation can be written using a matrix operation $\boldsymbol{R}=\boldsymbol{W}*\boldsymbol{V}$.

Each r_i can be an input for the *j*th cell, whenever the inputs are available, the *j*th cell re-execute automatically its own program to produce the new v_j, for another iteration.

Examples:

- The synchronous activation of aggregate for game of life is the usual scheme. If ***V*** is generation n, then ***R*** is generation n+1.
- For competitive learning we have first synchronous activation of the aggregate corresponding to the transition from layer 1 to layer 2, the activation of the aggregates corresponding to the clusters of layer 2, then activation of the aggregate for the transition from layer 2 to layer 3, and so on until output layer is reached.

III.3 Activation of asynchronous aggregate

The asynchronous model has "time" as an extra information to be exchanged between cells. Three values are delivered by an asynchronous cell after execution of its program:

- the outputs;
- the time when the values are delivered;
- the expected time for the next exchange of information between the connector and the cell.

The temporal informations are controlled by the connector which will wake up the cells at the right time. Asynchronous models use the data structures described previously for synchronous models, with the additional items:

- a vector ***H*** containing the arrival time of the last messages coming from each cell (h_i for the *i*th cell);
- a vector ***U*** containing the next activation time for each cell;
- a counter ***m*** representing the last activation time for a cell in the same aggregate.

After the initialization phase of ***W***, ***V***, ***H***, ***U***, and ***m*** (***m***=0), the activation of an aggregate is described by the following algorithm:

- search the next cell(s) to wake up and update m;
- send to all such cells the vector ***V*** after application of the composition function ***C***;
- send to these cells the vector ***H***;
- all cells execute the function ***I*** and their own programs as soon as ***V*** and ***H*** are available;
- each cell i sends to the connector its output values, the present time (h_i), and its next activation time (u_i).

Note that the synchronous model can be emulated in the asynchronous mode by waking up the cells at each clock step.

Examples:

- As function ***I*** is executed within each cell, its expression may be much more complex. In particular, ***I*** can take into account information on the state of the cell and on the delays (***H***), and may be used to simulate behaviors like attenuation of action potentials, refractory periods or hysteresis, etc.... This is, of course, of great importance for simulation of realistic biological model.

IV. CONCLUSION

Compared to the other works having similar purposes, our approach has two specificities:

- the choice of the virtual machine methodology to construct a programming environment for A.N.N.;
- the choice of massively parallel architectures as hardware support for this environment.

The basic concepts of this virtual machine for A.N.N. have been validated writing programs for several classical examples of neural networks models. Unfortunately, there is no room in this paper to describe those examples.

A first specification of the programming language has been produced. Implementation of the three needed compilers has not been done yet. This work will take place during the coming last two years of the NE.R.V.E.S. project.

We expect through this approach to come out with a tool usable on a wide range of machines and offering full programs portability.

We also hope that work about the virtual machine will have significantly contributed to clarify the A.N.N. paradigm.

V. BIBLIOGRAPHY

[Gardner70] Martin Gardner
The fantastic combinations of John Conway's new solitaire game of life
Scientific American, N° 223, 1970

[Harp86] J.G. Harp, C.R. Jesshope, T. Muntean & C. Whitby-Strevens
SuperNode project P1085: Developpement & application of a low cost high performance multiprocessor machine
ESPRIT 86, Bruxelles, 1986

[Hoare85] C.A.R. Hoare
Communicating Sequential Processes
Prentice Hall International, 1985

[Jutten90] C. Jutten, A. Guerin & J. Herault
Simulation machine and integrated implementation of Neural Networks
to appear Proceedings of EURASIP, Portugal, 1990

[Lawson90] J.C. Lawson, A. Chams & J. Herault
SMART: how to simulate huge networks
Submited to this conference

[Muntean89] T. Muntean
SuperNode: achitecture parallele et dynamiquement reconfigurable de transputers
in Proceedings of "Onzième journées francophones sur l'informatique"
EC2 éditeur, 1989

[Rumelhart86]D.E. Rumelhart & D. Zisper
Feature discovery by competitive learning
in *Parallel Distributed Processing*
M.I.T. Press, 1986