ABSTRACT

Simulation of two neural network models are used to illustrate the benefits of using C* as a programming language to create applications which utilize massive, fine-grained parallel processing architectures such as the connection machine. An overview of the C* language is made and general data structures to be used are discussed. Two neural network models are presented and contrasted: a relaxation energy model (Hopfield), and a forward propagation model (Rumelhart and McClelland). A discussion is made of using C* for simulation purposes and advantages of simulating neural networks on large parallel processors.

INTRODUCTION

The simulation of large neural networks is a memory and computationally intensive task. Fully connected networks require $n^2$ memory locations to store connection weights. Evaluation typically includes a dot product, and some non-linearity yielding calculations which also grows linearly.

Fortunately, neural network simulation can be structured to take advantage of data parallelism provided by system such as The Connection Machine (©Thinking Machines, Inc.). Using parallel processing techniques, much of the computational load can be distributed increasing system throughput.

The Connection Machine (CM) was initially configured around a Symbolics Lisp Machine as the front end and was programmed in *Lisp, a parallel version of Lisp which utilizes the capabilities of fine-grained parallelism in a SIMD architecture. Initial CM applications were developed in *Lisp or in Lisp calling PARIS (PARallel Instruction Set).

Last year (1987), a new language was introduced by Thinking Machines which capitalizes on the broad knowledge of the C programming language in the Software Engineering profession. This language, C* [1], is a set of extensions to C which supports parallel programming on massive, fine-grained machines. Extensions made are also compatible with the C++ object oriented programming language.

The use of C* to create applications for the Connection Machines is examined here using Neural Network simulation as an example. Neural networks are well-suited to implementation on the CM, and have the advantage of existing on other architectures for comparison [2].

AN OVERVIEW OF C*

The C* programming language was developed as a set of extensions to the C programming language developed by Bell Laboratories. C has been called an intermediate programming language since it is close the hardware platform while retaining some of the structure of higher-level languages.

Several extensions to C have been developed over the years to augment the basic language. One of these, C++, was used as a model for C*. C++ is an object oriented programming language which provided support for defining generic objects and manipulating instances of them. C* has been designed to be compatible with C++ and borrows several language concepts from it.

The underlying model for the language is an array of processors with data memory each executing the same instruction at a given time. Data organization is the same in each processor's memory. The only thing that differs is the values contained in these data structures. This model allows developers familiar with C to easily picture the parallel operations being performed, since array operations are common in C. This visualization ease is a strong feature of the language.

Four basic extensions were made to support parallelism: poly data types, the domain concept, processor selection, and a few new operations. An effort to stay close to C was made by retaining the existing syntax as much as possible while interpreting them in a parallel domain. For example, assignment works just like it does in C, but also allows
parallel assignments to be made. One design goal of C* is to allow compilation of pure C code in C* (this would run on the front end only).

RELAXATION MODEL

The first neural network model simulated on the Connection Machine was a relaxation model based on the work of J. Hopfield [3]. As shown in figure 1, each of the net nodes are fully connected to all others in the system, although self-connections are suppressed. The output of the jth element is:

\[ X_j = f \left\{ \sum_{i,j} (W_{ij} \cdot X_i) - T_j \right\} \]  

Where \( W_{ij} \) is the connection weight from element i to j, and \( X_i \) is the output of element i. \( T_j \) is a threshold associated with the element, and f is a non-linear function such as Sign or Sigmoid.

Because this type of network is fully connected, a connection weight matrix must be stored with each network element. The size of this matrix increases as the square of the network size. This weight matrix dominates all other data structures and makes large networks difficult, since the amount of memory per processor element in the CM-2 is limited to 64K bits.

One way around this problem, which was chosen for this simulation, is to represent a single network node as an array of processors in the Connection Machine. Each processor in this array computes a part of the net node's output. Inner products are summed over the array of processors to produce a new \( X_j \) in the zeroth element. C* provides all the numerical manipulations needed to calculate these quantities in an efficient manner.

This technique was used to create a relaxation network of 256 nodes which were fully connected. A variety of graphic pattern recognition problems were run, including compensation for positional, and rotational shifts in images to be recalled.

FORWARD PROPAGATION MODEL

The Forward Propagation model differs from the Relaxation model by having distinct layers which propagate information forward to a single output representing the recognized pattern. This type of network has been used extensively for Neural Network experimentation by researchers such as Rumelhart and McClelland [4], Hinton [5], Sejnowski [6], etc.

This model has an advantage over the relaxation model in that substantially fewer connections are required. In general, connections grow linearly with the size of the network. A connection weight matrix must be saved in each processor element, but the size of matrix is not as large. Furthermore, excess processors can be used as additional layers, since the forward propagation model does not require a symmetrical network.

In a fully configured CM-2 with 64K processors, 65536 network nodes could be simulated arranged as 64 layers of up to 1024 nodes each assuming 16 bit weight values, as shown in figure 2. If learning is added to the system using a rule such as back propagation, the number of nodes drops to 32768, since two copies of the network are needed to implement the rule.

Figure 3 shows how information flows through the network in a Connection Machine implementation of the Forward Propagation model. At each sample iteration, an input vector, \( I_i \) is initialized in the left most column of a two dimensional array of processors. This array can be up to 1024 by 64 in size. Each value is multiplied by a connection weight.
associated with a processor in the next layer, and summed. The sum is then thresholded and passed through a non-linearity as in the Relaxation model. This value is then deposited in the appropriate processor the next layer.

In general C* provides a powerful, clean syntax and eases the visualization of parallel calculations. The domain concept allows data to be organized and manipulated in a parallel fashion. By extrapolating existing C syntax to parallel behavior, transition from C to C* is a much less painful process than learning the structure of a new language.

### CURRENT AND FUTURE WORK

The Relaxation model described has been implemented and tested on a series of 8 by 8 images. The system is able to recall simple patterns encoded in the network. It has been tested up to 25% noise levels. Networks that correct for translational and rotational shifts have also been tested. The Forward Propagation model exists as a crude prototype. Lack of good test cases has made validation difficult.

The Connection Machine provides a powerful execution environment for neural network simulations. Future work with it will include development of learning rules, such as the Generalized Delta Rule and other neural network models, such as Kosko’s Bidirectional Associative Memory and Hecht-Nielsen’s Spatio-temporal Pattern Recognition model.

### REFERENCES