HAS:
A BROADCAST HIERARCHY SIMULATOR FOR THE INTEL iPSC/1

Kevin Neil Jagla
B.A. 1972, Colby College
M.B.A. 1978, Boston College

A thesis submitted to the faculty
of the Oregon Graduate Center
in partial fulfillment of the
requirements for the degree
Master of Science
in
Computer Science and Engineering

June, 1989

The thesis "HAS: A Broadcast Hierarchy Simulator for the Intel iPSC/1" by Kevin N. Jagla has been examined and approved by the following Examination Committee:

Daniel Hammerstrom
Associate Professor
Thesis Research Advisor

Michael Wolfe
Associate Professor

William Bain Jr.
Adjunct Assistant Professor

Acknowledgment

I wish to express my thanks to my wife Pamela for the support and encouragement that she gave me during the development of this thesis. I also would like to thank the people at Oregon Graduate Center especially Dan Hammerstrom, Bart Butell, and Harry Porter, all of whom stand out as excellent examples of people who have both the ability to do and to teach. A final thanks to Margaret Day for her work in ironing out the myriad of administrative details that hinder the progress of learning.

Table of Contents

i. Title Page

ii. Approval Page

iii. Acknowledgment

iv. Table of Contents

v. List of Figures

vi. Abstract

1. INTRODUCTION.................................... 1

2. DESIGN CONSIDERATIONS FOR A DISTRIBUTED SIMULATOR 2

   2.1 Research Environment....................... 2

   2.2 Constructing A Connectionist Application.... 2

   2.3 Flexibility to Simulate Many Different Neural Models.................................. 3

      2.3.1 Handling of Functions Used by Connection Node Types............................ 4

      2.3.2 Connectivity Specification: How do you Encode a Network into a File........... 6

   2.4 Control over Timing Constraints............. 6

   2.5 Operates on the Intel Hypercube............ 7

   2.6 Use of Input Files Written in BIF.......... 8

3. HARDWARE....................................... 9

4. NATURE OF THE PROBLEM.......................... 10

5. FUNCTIONAL ORGANIZATION OF THE SIMULATOR......... 14

   5.1 Cube Manager: cmgr1........................ 14

   5.2 Node Process: pn0.......................... 16

   5.3 User Function: user_fx..................... 17

   5.4 Communication Process: tbh................. 18

5.5 Process Structure.............................. 19

6. MAPPING BROADCAST HIERARCHY..................... 21

7. TIMING ALGORITHMS FOR PARALLEL SIMULATORS....... 24

7.1 Misra Algorithm............................ 24

7.2 Time Warp.................................. 27

7.3 Step Syncronization........................ 30

8. USING SEPARATE PROCESSES TO EMULATE BROADCAST HIERARCHY............................ 32

9. IMPLEMENTATION OF HAS (HIERARCHICAL ARCHITECTURE SIMULATOR)....................................... 34

9.1 Load Process............................... 34

9.2 Implementation Details of the Timing Process 35

9.3 A Complete Cycle on the Simulator.......... 36

10. THE EXPERIMENT................................. 38

11. PERFORMANCE.................................... 41

11.1 Evaluating Performance.................... 42

11.2 Effects of Parallelism on the Original Problem.......................... 43

11.3 Solving the Parallel Problem.............. 46

11.4 Managing the Scarce Interprocess Communication Resource.................................. 47

11.5 A New Mapping of the Application onto the Hypercube................................. 49

12. CONCLUSIONS.................................... 51

12.1 Usefulness of Parallelism in Solving Neural Network Problems.......................... 51

12.2 Sizing Resource Needs..................... 51

12.3 Effects of Hierarchies on Performance...... 52

12.4 Time Stepped Synchronization.............. 52

13. REFERENCES..................................... 54

LIST OF FIGURES

2.3.1 Connection Node................................ 5

4.0 Block Diagram of Processor...................... 11

4.0 Eight Node Hierarchy............................ 11

4.0 32 Processor Simulation System.................. 13

6.0 Mapping the Broadcast Hierarchy on the Hypercube 23

7.1 The Misra Algorithm............................. 25

7.2 The Time Warp Algorithm......................... 28

10.0 3x4 Network Cycles to Convergence.............. 39

11.2 Graph of Internal Node Processing.............. 44

11.2 Graph of All Processing Except Receive Message. 44

11.2 Graph of All Processing........................ 45

11.3 Detail of Communication Time................... 48

11.3 Graph of Total Time Spent in Receive Message... 48

Abstract

A Broadcast Hierarchy Simulator
for the Intel iPSC

Kevin N. Jagla, M.S.
Oregon Graduate Center, 1989

Supervising Professor: Dan Hammerstrom

This paper discusses the features of a software program developed to simulate a parallel architecture on the Intel Hypercube. The design has parallel processors with no shared memory. The parallelism available in our Intel iPSC, which has 32 processors, enables the software to simulate the proposed architecture, including the interconnect bus. The simulations uncover some important resource needs.

The major task of the simulator was to aid in hardware design. This task required the program to be constructed to represent portions of the final hardware design. A major feature of the design was a novel hierarchical "come from" addressing scheme. This addressing scheme fit well with the problem of updating many thousands of connection node weights based on one node's output.

Performance figures were taken to assess the time consumed by each function performed by the simulator. These figures were gathered by simulating different sized neural networks. The results show how the architecture reacts to certain bottlenecks. The simulations also show the general applicability of a parallel approach.

1. INTRODUCTION

This paper explains how to construct a simulator to aid research in connectionist theory. The paper also explores performance issues related to parallelism.

The primary goal of the simulation is to test a parallel architecture to be used to model neural networks. Eventually the modeled architecture will be used in a silicon chip, and a complete neural modeling system will be designed from VLSI circuits.

The neural modeling system is designed to be flexible and can be "loaded" with any given type of neural network architecture. These architectures may have different node connection schemes, different input/output functions, and different activation functions. This flexibility makes it ideal for studying connectionist theory, and for prototyping applications based on networks. The final VLSI system will be a hardware implementation of the architecture simulated on the hypercube by this software package.

2. DESIGN CONSIDERATIONS FOR A DISTRIBUTED SIMULATOR

2.1. Research Environment

To understand the design of the simulator, it is important to understand that the simulator is not a stand-alone tool. Instead it is constructed to fit within a larger research environment, which supports research in connectionist theory. The environment has been developed with a layered design approach similar to a VLSI design environment. Tools in the top layer of the environment, such as Network Design Language (NDL) support the design of a network, which is similar to high level VLSI design tools. Lower layer tools, such as MAPPER [Bai88], specifies the most efficient mapping of the network, which are similar to VLSI auto-routing tools. Simulation tools are used to test performance of the neural network being designed. These neural simulation tools ANNE (Another Neural Network Simulator) and HAS (Hierarchical Architecture Simulator) are similar to SPICE and other circuit simulation tools found in commercial design packages. Together these network design tools allow one to take a neural network design from conception to final hardware implementation.

2.2. Constructing a connectionist application

To avoid confusion between nodes in a user's network, and nodes of the iPSC and nodes in the modeled architecture, connection nodes are henceforth called CNs, hypercube nodes are referred to as HNs and processor nodes in the modeled architecture are called PNs. To construct

a connectionist application, the following general steps (and corresponding tools) must be used. NDL lets you use high-level directives to specify a network. NDL uses inputs from the researcher to create a file containing a description of the desired network. Mapper gives you a way to map specified connectionist nodes onto the target simulation platform. Mapper takes information about the layout of the network from the BIF (Beaverton Intermediate Format) file and merges it with information about the simulation platform to create a new file specifying how the network will be mapped onto the simulation platform. The last two elements of the environment are the simulators built to run on the Intel Hypercube.

ANNE [Bah88] is a general purpose simulator for debugging networks and verifying they are operating correctly. The second simulator HAS (Hierarchical Architecture Simulator) tests how the application will run on the proposed VLSI hardware. The model can be tested on HAS, and its efficiency on the final VLSI system can be estimated.

In the research environment HAS models debugged networks. HAS measures how efficiently a particular network will be on the hardware under development. It provides information on how much system resources will be necessary to model certain networks.

2.3. Flexibility to simulate many different types of neural models.

One of the design goals of the simulator was to make it flexible enough to allow many different types of network models. Supporting flexibility

led to a general view of what would be needed to allow many different models to work: a set of neural functions, changeable between network models, different types of data necessary to run certain models, and the interconnected layout (or topography) of the network. All these had to be conveyed to the simulator.

2.3.1. Handling of functions used by CN types.

Different types of networks contain different nodal algorithms for processing the data each CN receives. So much flexibility is required for user functions that a general approach was taken. Figure 1 depicts a connection node. It has several input links connected to the connection node at a particular site. The site is a conceptual portion of the connection node where a function is used to modify the incoming link values. This may be any function, but in the figure the site function consists of summing together the product of the link weight and the input value. The connection node then uses some type of activation function to determine if the node will fire or not. In the figure, the activation function is a sigmoid function using the value of the input site function. Finally based on the results of the activation function there is output produced. The simulator was written such that there is an external user procedure, provided by the user. The user codes the CN's algorithms within the user procedure. The user procedure is written to conform to a simple circular state diagram for all CN's:

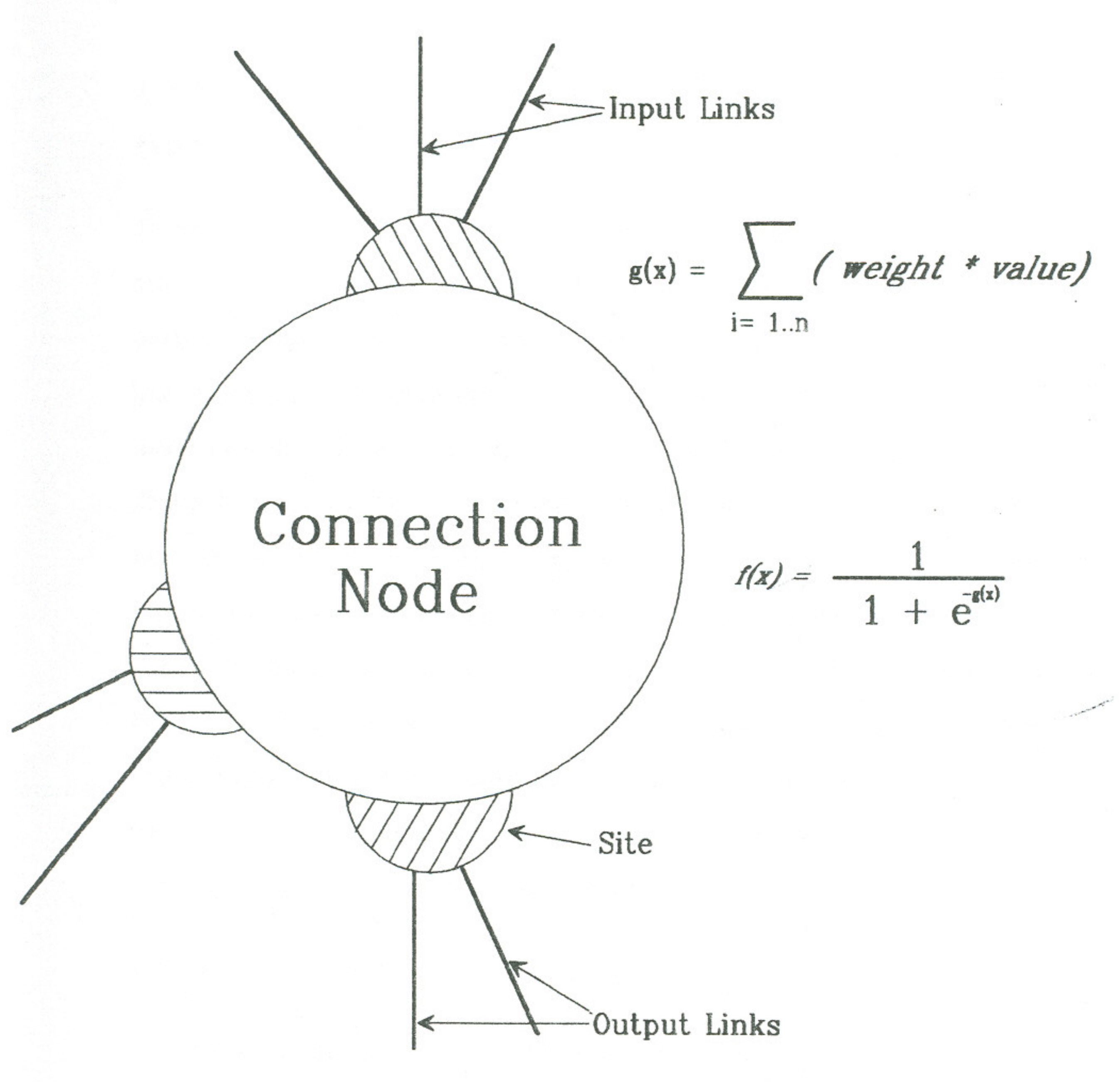
STATE ONE: Receive messages based on results of last cycle.

STATE TWO: Store received value in a field on the link record.

STATE THREE: Process results based on inputs and functions. In a

Figure 1

A Connection Node



common case this step would multiply the received value by a weight, and sum the result to a value, and use a sigmoid function to determine its output.

STATE FOUR: Send messages that were generated by the calculation step.

2.3.2. Connectivity Specification: How to you encode the network into the file?

To recreate the network on the simulator, you must encode the connections into the network file. In HAS, the encoding has another twist: each CN's output is directed to a single process emulating a serial bus. Since this bus broadcasts to many PNs, output goes to many CNs at once. The MAPPER has information on the layout of HAS and its broadcast regions. BIF [Bah88] contains the connection information in its link records. These records show to whom each CN is connected. HAS contains Broadcast Hierarchy links rather than CN links for output connections that are read into the simulator. Records are constructed to download to the different PNs. Each PN receives the CN records, containing all of their links at a time. These records are then loaded into areas set aside in memory. The CN and its records are then available at only one PN. No maps exist in the simulator of exactly where each CN is.

2.4. Control over timing constraints.

In the most general case, the simulator must be able to perform the simulation by driving all CNs through their four states in lock step. This

is desirable in testing. In some network configurations, lock step is the only way the simulator will arrive at correct results. Some of the timing issues in these networks have not been explored. So the ability to control the timing mechanism was built into the simulator [BHJ88]. The basic option allows the simulator to drive each CN through all four of its states, then synchronize before continuing the simulation. Other options include allowing a message within a certain time range to be accepted as if it were on the correct cycle. The simulator can also allow any message to be accepted, regardless of time stamp. There is also the possibility of allowing the simulation to proceed until a specified step before actually having to synchronize. Separate HNs would be able to "average" their work load over ten simulation steps prior to stopping and synchronizing. Networks may be found which will be relatively insensitive to the consequences of late messages, and will be simulated with no synchronizing step or a long time between steps.

2.5. Operates on the Intel iPSC/1.

The simulator was written to run on the Intel iPSC/1. The simulator was designed to maximize the size of the network that could be held in the simulator. Network maps, as used by the Rochester [FFGL88] simulator, were not used by HAS. The simulator was to locate its connections through either a point-to-point message to its output, or a message over one or more broadcast regions. Memory space was only allocated for holding the CN data base and its current state.

2.6. Use of input files written in BIF.

To develop a useful research environment it is necessary to have the ability to transfer networks from one development tool to the next. To accomplish this, a descriptive file was developed that contains all the fields necessary to specify many different types of networks. (The file specification is described in the Appendix of the User's Manual portion of [Bah88]) The first file created, using NDL, is called BIF (Beaverton Intermediate Format). NDL, MAPPER, ANNE, and HAS all use the BIF format to read in networks.

A main feature of BIF is its ability to specify connections and their weights. There is a major difference between a BIF file developed for the other tools and the BIF file developed for HAS. To take advantage of the broadcast hierarchies developed in HAS, the BIF file specifies the output of each CN in terms of which hierarchy it is connected to, rather than what particular CNs receive the output. The use of hierarchies as output destinations leads to an optimization within BIF and the simulator. In examples of networks in which a great many of the connections are contained within a hierarchy, only one link record is necessary to represent all of these connections. In the simulator one message is transmitted and affects all CNs within that hierarchy. Often a highly interconnected network in normal BIF will become a much smaller file when readied for HAS.

3. THE HARDWARE

The Intel iPSC parallel system is a family of commercial machines that contain processors linked in a hypercube network. The first of these machines was based on the 80286 processor. The architecture of the iPSC favors the complete separation of the problem into self contained PNs, with little or no global information. The lack of global memory actually fits well with the architecture being tested. The question is how well will the architecture being simulated map onto the hypercube network? The architecture simulated has eight processors in each hierarchy serviced by a single broadcast bus. The eight PN per hierarchy grouping was mapped onto an eight node segment of the iPSC. Every eight HN grouping, representing a hierarchy, will have some messages that must travel three node-hops and some that will only have to go one node-hop. To understand more of the simulation, a look at the architecture is necessary.

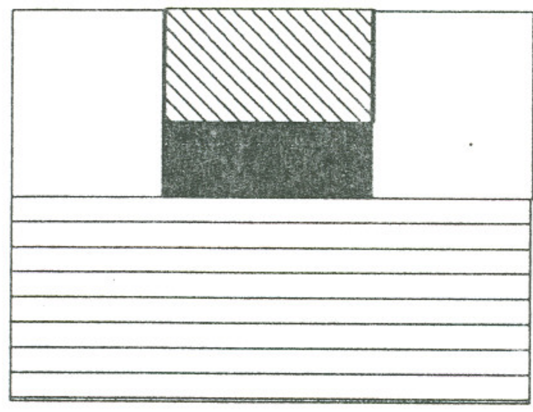
4. THE NATURE OF THE PROBLEM

The architecture to be simulated consists of large numbers of relatively simple processors [RuH88]. A conceptual look at the sections of the PN can be seen in Figure 2. There is a section of memory used to download the user's simulation functions. In the diagram ROM (Read Only Memory) is shown for storing user functions (not modified during the run). Two areas of associative memory are shown. One of the areas is fed the CN number from each message and the memory decodes it to the CN numbers that are affected by the message. The other associative memory area is used to decode output messages, sending them to the proper hierarchies. The output associative memory may not need to be large at all. A third portion is devoted to a central processor. It applies the downloaded functions to the different stored values of a CN. Finally there is the RAM storage for the CN's. The size of RAM storage will decide the number of CN's that can be simulated.

The PNs communicate with each other using a hierarchical bus structure, which allows communication with many processors in one bus-cycle. Figure 3 depicts the structure. In the figure, an eight PN hierarchy is shown. All eight PNs are linked to one broadcast bus. The broadcast bus receives messages from other hierarchies via the connection labeled "hierarchy bus". The hierarchy bus connection intercepts messages coming from other hierarchies to the broadcast bus and in turn captures outgoing messages from the broadcast bus and directs them to their proper hierarchy. To complete the communication link, each processor uses its bank of associative memory to identify CNs that are receiving input from an input

Figure 2

Block Diagram of PN



ROM memory for Downloaded Functions

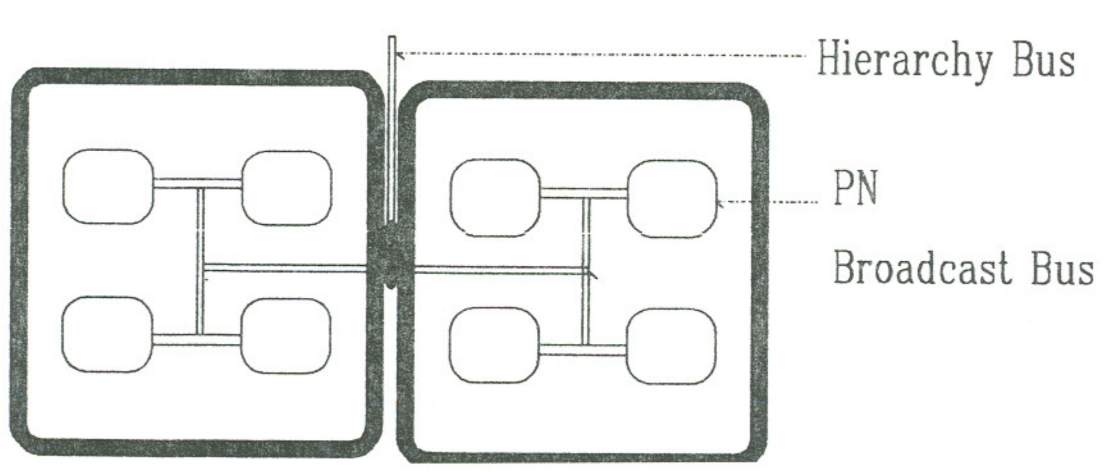
Associative Memory used to store Input Connection Node Numbers

Microprocessor to Perform Functions and Sequence

RAM for Storing Connection Nodes

Figure 3

Eight Node Hierarchy

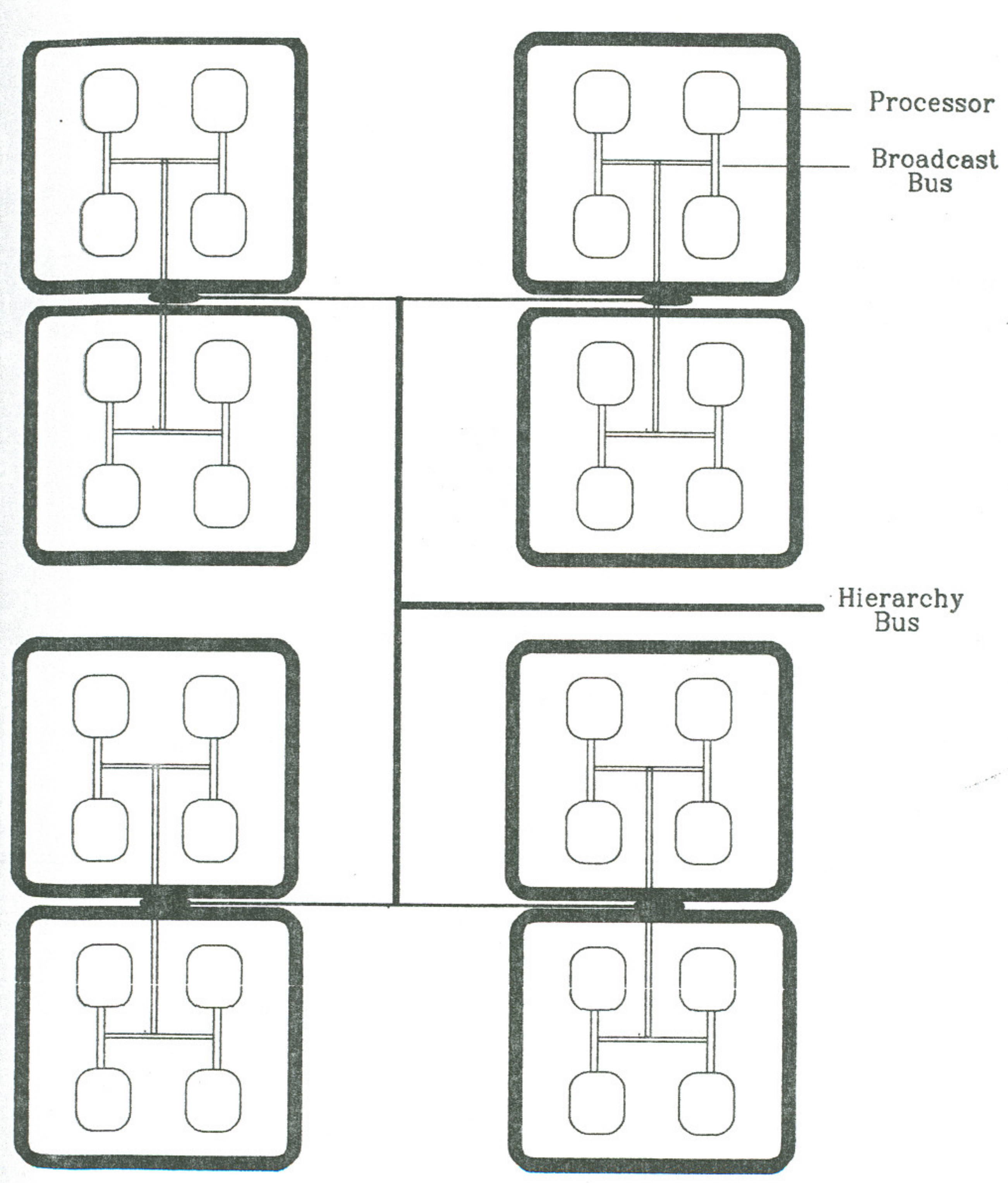


CN. Each processor performs a series of calculations using the input value sent to it from an input CN and weights associated with the connection link between the CNs. The user function is used by the simulator to calculate a value using the input values and a function provided by a user. The resulting value is transmitted as an output message.

The hierarchy structure shown in Figure 3 is easy to connect into larger and larger systems. Figure 4 depicts a possible layout for a 32-processor system.

Figure 4

32 Processor Simulation System



5. FUNCTIONAL ORGANIZATION OF THE SIMULATOR

The simulator needs a certain organization to function on the Intel iPSC and emulate the proposed architecture. To meet these needs, three basic types of processes were developed. The first is cmgr1, the Cube Manager, the second is pn0, the Physical Node, and the third is tbh, The Broadcast Hierarchy.

The simulator also needs a general method for debugging the cube processes. A debugging system was implemented between each HN and the cube manager. A general purpose call was made that creates a message and forwards it to the cube. The cube manager functions as an output for all HNs. A benefit was the debugging method worked equally well on the Vax running bsim, a cube simulator, or the iPSC, so conditional compile statements were not necessary.

5.1. The Cube Manager: Cmgr1

There is only one processor in the iPSC with the ability to access and use input/output. The processor called the cube manager is used to interact with the model under test and perform other tasks requested by the user. It has little to do with the simulation of the neural model, or the simulation of the hardware.

The first job of the cmgr1 is to load the BIF file. The user specifies which BIF file is going to be simulated. In the next step, the file is opened and each line is read into the simulator. Since no global information is stored in the cube manager, no records are kept

in the cube manager, they are all shipped directly to the HNs. The first few fields contain type information common to all CNs. It is sent to every PN in the system. Next, the CN information is processed. To do this efficiently, a record of CN information is prepared in the cube manager to download into the HNs. Each record created goes to only one HN, the one assigned a particular CN. The record is variable, since it is impossible to tell how many sites or links a particular CN contains. When the last link of the last site is finally read into the simulator, the cmgr1 determines which HN will receive the CN and it sends it. The loading continues until all of the CN records have been forwarded to their respective PNs on the HNs.

In handling user requests the cmgr1 is equipped with a user interface. The interface accepts commands from the user that will control the simulator. Some important commands include the use of setup commands for specifying simulation time step size, specifying the BIF file to be loaded, and specifying single step mode. There is also a single step command that causes the simulator to single-step through the simulation. All these operations aid the user in observing the simulation, and in modifying the simulation parameters.

When the simulation is started, output CNs begin to produce output messages. These need to be accessible to the user. Every output of an output CN is directed to cmgr1. Here the outputs can be seen by the user if he wishes, and can be checked against a "convergence" array containing expected final (or target) values for output CNs.

5.2. The Node Process: pn0

The node process performs most of the neural network simulation. Its first task is to set up communication with the cube manager. The cube manager counts the number of received messages and compares the number with the expected number of PNs. When the correct number of return messages is received from the PNs, the cube manager begins sending packet messages containing numbers of how many of each type of record will be received. The four database records are Type records, CN records, Site records, and Link records. Each PN allocates a block of memory for the expected number of records. A record with the number for each type of database record is taken from a configuration file developed prior to the simulation. After sending the allocation message to each of the HNs, the cube manager begins reading in the BIF file and sending to the HNs connection information. The first records are the Type records. Due to their size they are sent to all of the PNs. There is a chance that a type record may be stored in a PN and never be used, but that is unlikely in most cases. Then the CN information packets are sent. Each packet contains one CN record then two or more site records with their respective link records. After these have been sent by the cube manager, it sends down a message to "sort" using a routine internal to the PN. The only records that are actually sorted are the link records. These are sorted to aid in searching for the address of a message. After the sort the input file is read into the simulator and the input messages are transmitted by the cube manager. The input messages are directed to primary hierarchy processes. The hierarchy processes distribute the input messages within their group of PNs. The input

messages constitute the beginning of the first cycle of the simulator. After they have been received, the CNs will determine if they will output a record.

Another important function of the PN is the production of information for eventual display on the terminal. Debugging information can be displayed by the cube manager. Each debug statement is formatted into a buffer. The buffer is used to create a message to be sent to the cube manager. Each message arrives with an input value, the id of the CN that sent the message, and a time stamp. A flag in the iPSC message packet is set to differentiate packets for the cube manager. Incorporating this mechanism removed the need for many conditional compilation statements, which would have been necessary to allow debug statements for each type of machine.

5.3. The user function: user_fx

A part of each PN is a separately compiled portion called user_fx. Here the user codes the functions that will be used by the simulator. Each CN is expected to receive messages, sum the messages received, or perform some type of processing involving the received input. Finally the CN must output some type of message depending on computed output values. The user function is divided into four areas called user_modes to correspond to the four steps outlined above. Within each user_mode area the user enters the code specific to the desired simulation. The user function portion of the simulator is compiled separately, then during the link step it is connected to the other portion of the PN. The final image is downloaded onto the HNs.

When mode equals 1, the CNs are receiving messages from the broadcast hierarchy, which is passing them on from other CNs. There is a field in the link message that is used to store the value of the message received on that particular input link. After the messages have been received there are input values on all of the appropriate links.

When Mode equals 2, the values from the link fields are summed to the site level and stored in a field called siteval.

When Mode equals 3, the CN's calculate the activation function which was coded in the user function.

$$\text{output} = \frac{1}{1 + e^{-siteval}}$$

The output result is stored in the CN field called value. Every CN has an output value derived from its inputs. Even inputs of zero will create an output value of 0.5.

When Mode equals 4 all of the CN's transmit their output messages. The cycle begins again.

5.4. The Communication Process: TBH

The simulation of the broadcast bus and its queue is done by the broadcast hierarchy process. The broadcast hierarchy receives each message sent from one CN to another. The broadcast hierarchy sequentially passes the messages, and logs how many messages actually ocurred during a cycle. Accumulated statistical data is sent to the cube manager at the end of each cycle. The user can measure the number of messages sent during a cycle.

5.5. Process Structure

All of the work within the system is initiated via a message of a given type. The message type is used to encode what the message is to perform. There are two types of messages. The first type are those messages initiated by the system to perform the simulation. The second type are those messages that support the functioning of the simulator. Such a message might be a request to load a new BIF description, a command message to initiate cycle control, or other messages initiated by the user interface.

Along with the transaction nature of the simulator is the use of processes to emulate bus or other hardware units. The design leads to a group of processes each with a specific job to perform emulating the hardware system. The cube manager is the command process and only services the simulator during operation. The cube manager's most important job during the simulation is to provide the message initiating the next simulation step.

In the HNs, PNs are constructed to perform certain operations. First the PNs read in downloaded neural data and store the CNs they will control. Second, PNs receive and send information based on results of internal calculation.

The process found in one HN per hierarchy is the broadcast hierarchy process or tbh. Tbh is an integral part of the communication link between PNs. PNs are required to send their output messages through the broadcast hierarchy. The process acts as the bus for a group of eight PNs. The

broadcast hierarchy sends received messages on to the eight PNs and keeps track of message traffic.

The design creates a software simulator that performs like the envisioned hardware. The use of simple processes to emulate portions of the system help point out some of the areas where performance bottlenecks will occur. Also measurement information can be transmitted from the PNs to the cube manager for display.

6. MAPPING THE BROADCAST HIERARCHY

One bus structure being simulated is called the Broadcast Hierarchy. A hierarchy consists of four PNs in a shared bus network. These four PN units are then connected to four more identical units, creating an eight PN building block for the next level of integration. See Figure 3 for an example of the 8 PN building block used to create a 32 PN hierarchy cell. As scaling continues a network of processors is created. The architecture is useful for simulating connectionist networks. First, if a connectionist network can be laid out among the processors such that a majority of its communication occurs in localized areas, then the simulation will benefit from the large amount of local bandwith that is available in the architecture.

Second, is the availability of a bus that can broadcast to many CNs at once. In high fanout networks when one CN outputs a value, the value is sent to several thousand other CNs. The broadcast bus allows a CN to transmit to many CNs in one message cycle, the architecture anticipates the high fanout characteristic of some networks. The user maps the network to insure CNs expecting output from an input CN can be reached in one hierarchical region. If a good mapping of network graph to broadcast hierarchy was accomplished, each CN sends one message and services all of its receivers. An efficient mapping cuts down on message traffic as the architecture scales [HoB86].

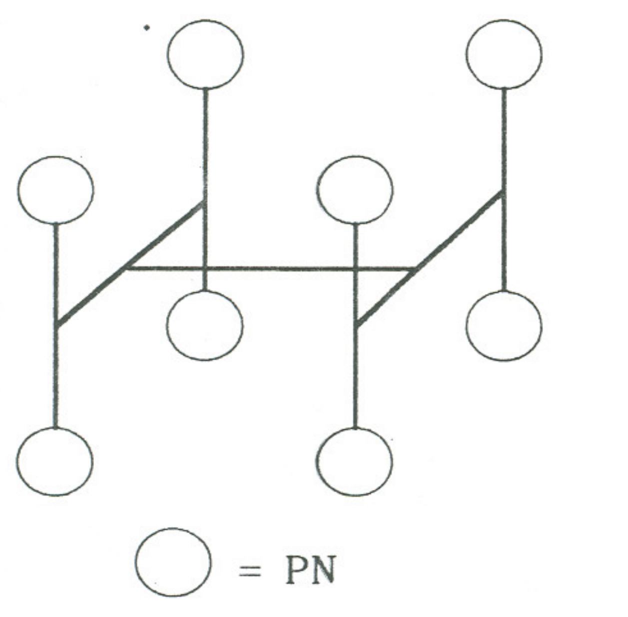
To simulate the broadcast bus on the Intel iPSC/1, the simulator maps the hierarchical bus structure onto the cube. Figure 5 shows the approach.

Eight PNs are mapped to eight HNs in the architecture model. An independent software process is used to simulate the broadcast hierarchy. A broadcast hierarchy is assigned for every eight PNs. The broadcast hierarchy passes messages to the CNs contained in its eight HN sector of the Intel hypercube. A separate process is used in order to insure that a simple first in, first out queue is used passing messages within the eight HN hierarchy. Each message to the hierarchy must queue up and pass through the hierarchy bus process. To pass messages to eight PNs on the iPSC/1 requires several message steps. First the broadcast hierarchy contacts the PNs it can reach. Let each of the eight HNs occupy a vertex on a cube. If the broadcast hierarchy is on vertex 0, then it can contact PN 0 (the PN process sharing HN 0 with the bus process), and PN 1 (on HN 1), PN 2 and PN 4. These are the only PNs the broadcast hierarchy can communicate with directly. The four arrows in figure four emanating from the broadcast hierarchy represent the messages described. These PNs are one hop from the broadcast hierarchy. In order to communicate with the other four PNs the messages need to be forwarded. Code is in the PN to forward a message from PN 1 to PN 3, from PN 2 to PN 6, and from PN 4 to PN 5, as seen in figure 5. Finally the last PN 7 is contacted by PN 3. All of this code can be used in other eight HN groups of the cube.

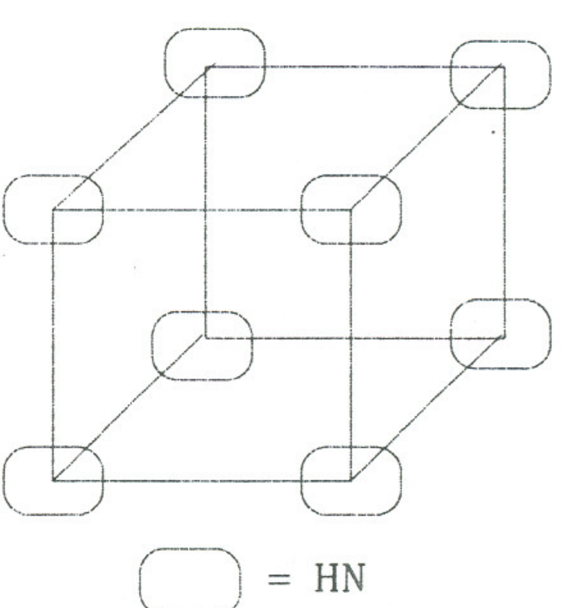
Figure 5

Mapping the Broadcast Hierarchy onto the Hypercube

The Broadcast Hierarchy.



Hypercube Bus Structure



Architecture Mapped to Cube

Broadcast Bus Process

PN1

PN0

PN3

PN2

PN5

PN4

7. TIMING ALGORITHMS FOR PARALLEL SIMULATORS

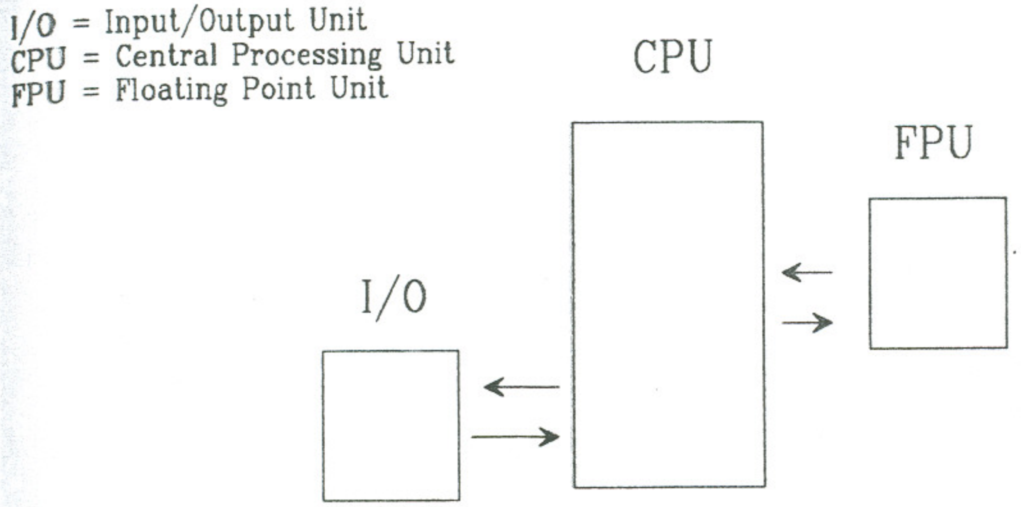
One of the most difficult decisions to make in designing HAS was how to preserve chronological time in the simulation. In order to simulate a bus structure it is important the simulator realistically model the characteristics of the bus structure. Otherwise two different portions of the simulation may be counting on control of the broadcast bus at the same time. Properly modeling the serial nature of the bus is also critical in assessing bus bandwidth requirements. Assessing bus bandwidth was accomplished by using two separate approaches. First, one process functions as the broadcast bus and the simulation is insured that a single queue is used to model the bus resource. The bus exhibits the serial nature necessary for a realistic simulation. Second, control over how the simulator is synchronized was desired. The simulator was to have the ability to allow each processor to continue to the next cycle, allowing the HNs the ability to run at their own speed. Three different timing mechanisms were explored to insure that all processors in the cube were working on the same time step.

7.1. Misra Algorithm

In looking for an efficient technique to control timing on a parallel simulation, the work of K. M. Chandy and J. Misra was considered [ChMis79]. His study of timing for simulations concentrated on event driven simulations and their problems when implemented on a parallel machine. The major drawback up until

Figure 6

Chandy,Misra Algorithm



Implemented Using the Null Message

	I/O	CPU	FPU
CYCLE 1	MSG->CPU 2 Numbers	MSG->I/O NULL MSG->FPU NULL	MSG->CPU NULL
CYCLE 2	MSG->CPU NULL	MSG->I/O NULL MSG->FPU 2 Numbers	MSG->CPU NULL
CYCLE 3	MSG->CPU NULL	MSG->I/O NULL MSG->FPU NULL	MSG->CPU 1 Number
CYCLE 4	MSG->CPU NULL	MSG->I/O 1 Number MSG->FPU NULL	MSG->CPU NULL

Informational Messages 4

Null Messages 12

Algorithm requires a three fold increase in messges

their work was based on the fact that event driven simulations worked from an event queue which was global to the problem. Thus, event driven simulations were sequential problems, which do not benefit much from parallelism. To overcome running the simulation sequentially, Misra focuses on the timing problem in a new way. First he assumes that between two processes there is a link, such that if one process sends an ordered set of messages the order will be preserved on the link. Then he can say that if one process receives a message from another process, the process knows it has received all of the messages prior to that message. The simulation introduces time-stamped messages from each process sent to all of the processors expecting input. The order of the messages insures the processes know that no other messages are coming from a given process.

One synchronization approach uses null messages to ensure a process is still operational when no output was developed. After processing each input message, a process generates a message, only sometimes it is a null message which is nothing but the time stamp. The use of null messages allows all portions of the system that receive a message to continue, knowing the receipt of a null message signals the process is not going to have output this cycle.

The problem with implementing the Misra null message algorithm rests in the extensive use of message passing to insure that chronological time is preserved. The simulator has communication as its biggest bottleneck with the connectionist messages. The use of scarce bus resources to implement this algorithm is not feasible, and the extra messages used to implement

the algorithm would overwhelm the simulation. In figure 6, a simple system is modeled using Misra's algorithm. In some circumstances many of the messages are overhead to support the algorithm. So it was decided to not use the Misra's algorithm, since it is a poor design fit to the problem being addressed.

7.2. Time Warp

Another timing algorithm considered for the simulator was Time Warp, which was developed at the Rand Corporation [Jef84]. The algorithm was originally implemented in Lisp at Rand, and was adopted by a group at the Jet Propulsion Laboratory. They implemented the Time Warp on a 32 node Mark II Hypercube. In time warp, messages are processed and their time stamps are noted. As long as the time stamps are in order the simulation proceeds normally. If a message is received that has a time stamp earlier than the current time, then the algorithm turns back the time clock, unsending messages that had been received. This leads to more sections of the simulation having to perform the same process. Finally the message can be incorporated at the proper time and the simulation proceeds. This process requires a series of history files to be kept to allow the time to be backed up. Figure 7 depicts the series of history files that need to be stored. Since the nodes on the ipsc/1 do not have much space available for storage, the algorithm would not fit well with the available resources. The message needs of a neural network simulation and the needs of storage would overwhelm the iPSC/1. Also the job implementing the Time Warp on the Intel Cube would be a long project in itself, since much of

the operating system had to be rewritten to allow for the Time Warp algorithm. In the interest of limiting the scope of the project, the Time Warp was abandoned as a viable choice for controlling timing.

Figure 7

Time Warp

Past States

Current State

0

t - 2

t - 1

t

States are stored to be retrieved and updated with any late arriving messages.

Requires storage for past states

Required a re-write of the operating system.

7.3. Time Stepped Synchronization

The actual synchronization method used by the simulator is step synchronization. In the step synchronization model, the simulator is stepped through time and all the events that must occur in a cycle are done. There is a move to the next cycle and the cycle repeats. The time step algorithm allows parallelism because each process moves through the steps in a cycle at its own pace. The algorithm does restrict each process to moving at the pace of the slowest HN.

Syncronization is maintained by using a relaxation algorithm. The simulation waits a time interval after the last message received. If no messages are received during the time t, then the PN assumes it has received the last message. At this time it contacts the cube manager, who waits to hear from all active processors. When all have sent a message, the cube manager advances the time stamp in each processor and the cycle repeats. If a message is received prior to the time t elapsing, then the timer is reset.

The time t is determined empirically. The simulator is run with a small t. If messages are still in transit and the PN signals for another step early, the out of order messages are reported back to the cube manager. The time t is then increased, and when no messages are being reported back, the simulator is running with sufficient delay to catch slower or in transit messages.

Load balancing helps optimize the speed of the simulation. The load can be balanced by properly assigning CNs evenly across the simulator. Mapper can do this. Since processor load is directly related to messages received, the relative load can be estimated by the number of messages the processor will need to process per cycle.

8. USING SEPARATE PROCESSES TO EMULATE BROADCAST HIERARCHY

To control the Broadcast Hierarchy's bus and provide simulation parameters of bandwidth and cycle times, a process was used for each region of the hierarchy. In the simulator there are eight PNs sharing a bus. Each time a CN fires, it creates a message intended for one or more CNs. The message is directed by output links to specific hierarchies, but not PNs or CNs. When a hierarchy receives a message, all eight PNs read the message, which consists of the CN number, the value being passed as an output value, and the cycle that the message was created. All eight PNs look to see if the message received was intended for them. The PNs have a table of valid input CN numbers. If the input CN number of the message has a match in the input link table, then there is one or more CNs that need to be updated based on the value received. In the simulator the associative memory lookup is emulated with a list of input links sorted into input CN order. The process looks through the input CN list, finds the input CN, and updates the CNs with an value from the input CN. The HN then applies the message value to all of the links that have that input CN index.

The broadcast bus in the simulated architecture has the benefit of trading off computer processing power for bus bandwidth. With HNs containing many CNs, the number of iPSC/1 messages that are sent is decreased. The structure lends itself to scaling, which is more difficult to achieve in highly interconnected schemes. Each message causes eight PNs to do a table lookup to determine if they are being addressed. Forcing eight PNs through

a look-up table is efficient because it is implemented in associative memory. The worst case is when only one CN in a hierarchy is being addressed. When the networks are more interconnected many CNs share the same input CN, and the efficiency is greater [BaH86].

9. IMPLEMENTATION OF HAS

The implementation of the simulator was begun using a VAX 780 computer running Unix 4.2 Bsd and equipped with the Intel cube simulator, bsim.

The initial job was to partition the simulator into two groups of functions. First was a set of functions performed at the cube manager level. One of these functions was the loading and unloading of files. Another cube manager function was the communication of information to and from the user. The cube manager is the only input/output capability for the HNs. The cube manager is also used to display messages and errors. In addition, timing controls are implemented in the cube manager. Each HN shares a communication channel directly to the cube manager, which served as the communication channel for timing control messages.

On the HNs, there are two distinct types of processes. The first receives the CN records from BIF and actually performs the updates to the CN records. The other is a process developed to emulate the broadcast Bus. The broadcast process is busy receiving messages, gathering statistical data and sending messages to PNs during the simulation.

9.1. Load process

The simulator was made more flexible by adding the ability to load BIF files. During the loading sequence, the input BIF file is scanned and a single record is created for each CN. The record created contains the top level CN information followed by information about the CN's sites and the links to each site. When the scanner detects the beginning of another CN,

the created record is directed to the HN that will store it, as determined by MAPPER. At the PN the CN record is divided into three parts. The first is the connection node record. This record is added to a memory space already allocated. The second portion of the record is the site information. The sites of the CN are added to the list of site records. And, finally, all of the input and output links for the CN are added to the link records. When the HNs have all received their information, the link records are sorted according to whom they are connected. The sorted link records are used to determine the receivers of input messages.

9.2. Implementation details of the timing process.

The use of the time driven simulation technique preserves chronological time at a reasonable cost. The timing algorithm steps the simulation in time by using a global timer. In the simulator the cube manager contains the mechanism for the global timer. The simulator inputs the first input vector upon initialization, and then the simulator begins a cycle. As the messages are received a routine is invoked that checks the number of messages that have been received and compares it to a previous count of the messages received. The synchronizing routine is started with the receipt of the first message. The amount of time the simulator waits between checks is programmable and will be based on actual system results. When the timing routine discovers that no new messages have been received since the last check, it increments a counter. When the counter reaches a maximum value, which is programmable, the PN assumes that no further messages will be received. The timing routine sends a message to the cube

manager requesting a step to the next time cycle. When the cube manager has received a similar request from all of the HNs, it sends a message to continue with the next time stamp contained in it. When the individual PNs receive this message, they send out all of the messages for the next cycle, and the cycle repeats.

9.3. Explanation of a complete cycle on the simulator

The following description explains the steps used in the simulator to complete a cycle.

1). The first step of a cycle involves the receipt of a message from the cube manager used to flag the beginning of a cycle. The timing message is sent to every specified HN of the simulator.

2). The next portion of the cycle has the PN scan its input links in a step that sums the values of the links into the sites. The sum of the link values is stored in the site record.

3). The next portion of the cycle uses the summed value in a calculation that determines the value used as output. It is here where the activation function is applied to the sum of the inputs. The value of the calculation is stored at the CN node record in the output field of the CN record.

4). Finally the CN records are scanned and those with values to output have messages formatted and sent.

5). The messages are received by another CN from its broadcast process. The broadcast process notes the message in its metrics and then passes the

message on to the PNs located in HNs 0,1,2,and 4 or the modulus 8 of numbers 0,1,2 and 4 in other hierarchies.

HN 1 passes a message on to HN 3; HN 2 passes a message on to HN 6; HN 4 passes a message on to HN 5. Then in a final communication, HN 3 sends a message to HN 7 completing the massaging necessary to simulate an eight PN hierarchy on the hypercube.

6). The last step begins when the synchronization routine determines a PN has not received any messages within the default wait. The synchronization routine sets a flag and the PN contacts the cube manager with a message signifying it is ready to move to the next simulation step.

10. The Experiment

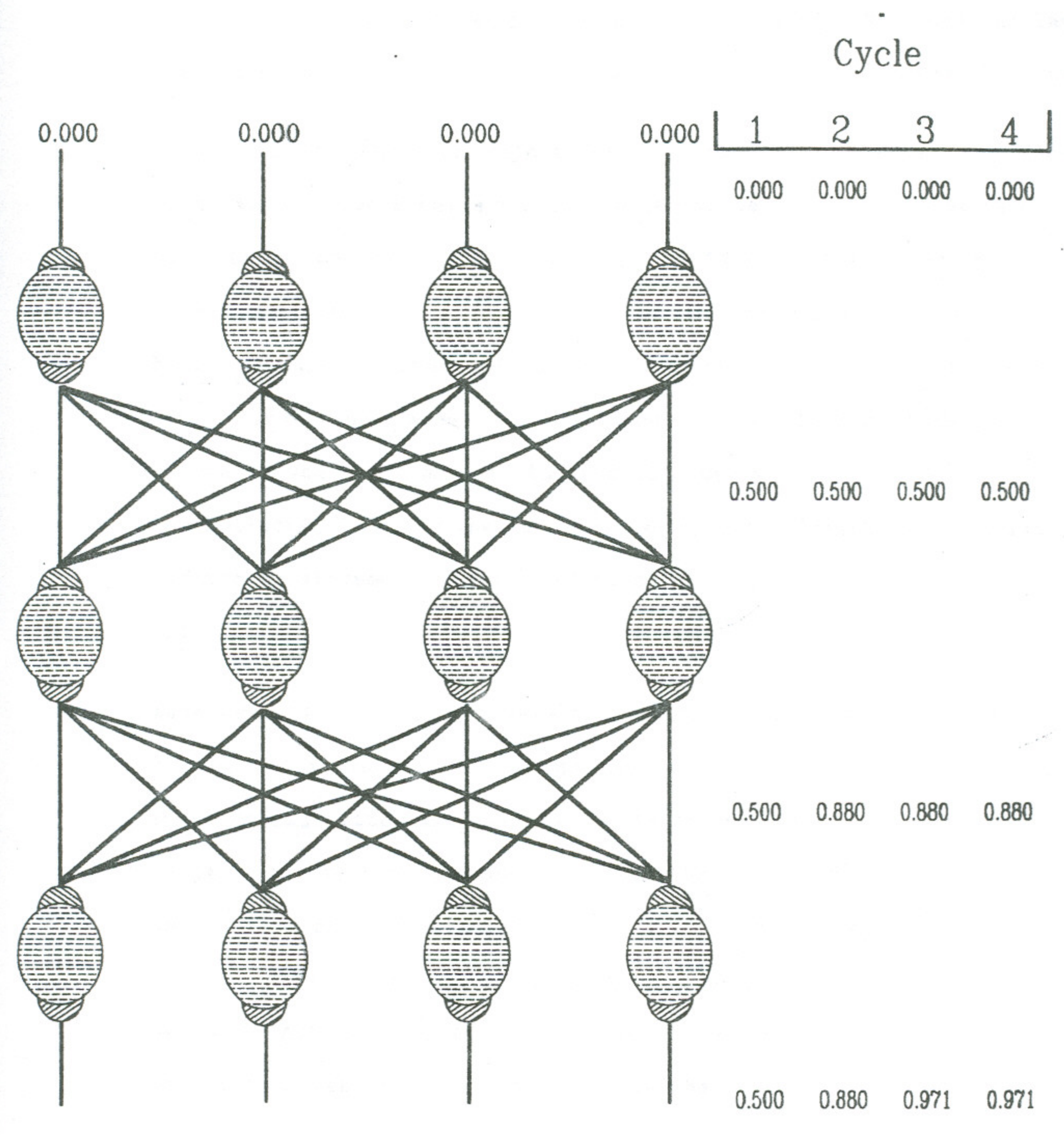
The simulations were run in a batch mode on the hypercube. The results were captured on the Vax. Using recorded output allowed time to evaluate results. An added benefit of batch mode was that the output files serve as a record of the experiments.

The first set of tests were used to verify that the simulator was capable of modeling neural networks. Tests were to document the ability of the simulator to drive a known network to known output values. Both the output values and the number of cycles were checked with expected results. To accomplish this task the simulator was set to require the nodes synchronize after every step. In a simple 4x3 network calculations show what each output state will be for each of the four steps necessary for the system to converge. In figure 8, there is a 4x3 network with an input vector of zeros.

In the first cycle the messages passing from layer 1 to layer 2 and layer 2 to layer 3 will have a values of 0.5. This is because the sum of the inputs will be zero, and then the sigmoid function of zero equals 0.5. In the second cycle the input remains zeros, messages are stable from layer 1 to layer 2 with a value of 0.500. The messages from layer 2 to layer 3 will reflect the value of summing 4 inputs of 0.5 which is 2 and taking the sigmoid of 2 which is 0.880 rounded. This is also the value that layer 3 will produce. In the third cycle of the network the input remains zeros, the first layer remains 0.500, the second layer remains

Figure 8

3x4 Network Cycles to Convergence



0.880 and finally the output layer begins producing 0.971. At this point the network has reached convergence.

The simulator was expected to go through the same cycles with similar results with three different types of networks. One was a three layer network with eight CNs per layer. The next was a three layer network with 15 CNs per layer and finally a three layer network with 32 CNs per layer. The expected message values were calculated prior to running the tests.

In the second phase of experiments the simulator was used to develop performance numbers while simulating neural networks. Performance numbers were taken on particular functions performed. The relation of each function to the whole, shows what size resources need to be in a system based on this architecture. In order to gather statistics, the PNs had timing statements added to their code. These timing statements where placed around functions. At the end of a cycle, the PN would communicate the collected data via the message passing channel with the cube manager. Additional statements were added to obtain timing on areas of particular interest.

Each network was mapped several different ways. These were permutations of two variables, number of processors and number of hierarchies. The first configuration used is all of the network in one PN. In all but the largest network this was possible. Next the mapping would involve more PNs until all eight PNs of one hierarchy were used. PNs were added in powers of two. So the first set of performance runs involved 1, 2, 4, and 8 PNs. Next the network was distributed over two hierarchies. Then the same permutation was used all the way to 16 PNs.

11. PERFORMANCE

The purpose of the simulator is to study the performance of the simulated hardware architecture. The final hardware will be much faster than the simulator. In order for the hardware to be effective, certain characteristics must be present in the simulation. The accurate simulation to uncover these characteristics is more important than the raw speed of the simulator. In order to benefit from a parallel simulation, the simulated problem must be inherently parallel. This allows the problem to be partitioned among multiple processors to solve the problem faster than would be possible on a single processor. A second and equally important characteristic is scalability, which allows the addition of processors without incurring a non-proportional amount of overhead.

The need for these characteristics within the simulation model demonstrates that the use of parallel architecture introduces complexity and potential overhead to the simulation. These obstacles can in fact obscure the measured performance of the simulated hardware architecture. In effect, when a problem is translated to a parallel architecture the problem becomes a part of a larger problem. The larger problem contains the original algorithm and also the algorithms and overhead that come with supporting the problem in a parallel environment.

Data is collected in several ways to analyze how major portions of simulator cycle time is used. By reviewing timing data, areas of performance are identified that are caused by the parallel platform on

which the algorithm is run. These have little to do with the analysis of the simulated architecture directly. They do tend to be part of the larger problem of overhead and as such point to areas where all parallel architectures have problems. The analysis also allows one to review some of the potential overheads that can be expected in such an architecture.

11.1 Evaluating performance.

To analyze the performance of the simulation, each simulation cycle was divided into four parts. Each part represented a major function the processor would have to perform. Output statements were added that timed the part using the internal clock of the HN. Each HN timed how long it took to perform each step of the simulation. The performance results were captured using script files. The final numbers were entered into a Lotus 123 speadsheet and averages were taken for each step and for the total. Also the number of links updated per second was calculated. Links updated per second shows the performance changes in the simulation as the neural model is moved over larger numbers of processors. Running the neural model on different numbers of processors also reveals the ability of the architecture to scale effectively. To get additional information there was a standard number of runs done per network/processor simulation. First the network was executed on one PN only. The same network was then mapped to two PNs, then four PNs and then eight PNs. All of the mapping was done within one eight PN hierarchy. Next the same network was remapped using two PNs, each in its own hierarchy. The network was then mapped to four PN's, two in each hierarchy. Finally 16 PNs in two hierarchies were

simulated. The simulation cycle contained four major parts. The first part contains the time it takes to multiply the input by the weight of the link and sum the results into the site value. The site value is input for the sigmoid function. The next part that is timed is the actual calculation of the sigmoid function and the storing of the result in the CN field called "output". The third part timed involves taking the CN outputs, building a message and transmitting it. Finally, the last part timed involves the receipt of messages, their passing on to other HNs, and the time taken to locate the affected CNs and place the results on the respective links.

11.2 Effects of parallelism on the original problem

To aid in demonstrating the effects of parallelism on the problem, please refer to figures 9 through 11. As the numbers of PNs are increased, it can be seen in Figure 9 that the original problem experiences a dramatic increase of speed. Therefore, the original problem scales onto parallel processors well. Once a process has received its input values, it can proceed without waiting for additional data. Some of the parallelism is lost when the processors synchronize at the end of each cycle, but it does not take a great deal of extra time when the numbers of links that require update are spread evenly across the processors. Also the time scheduled for synchronization is a fixed quantity, so it scales with HNs, becoming a smaller and smaller portion of the overall time.

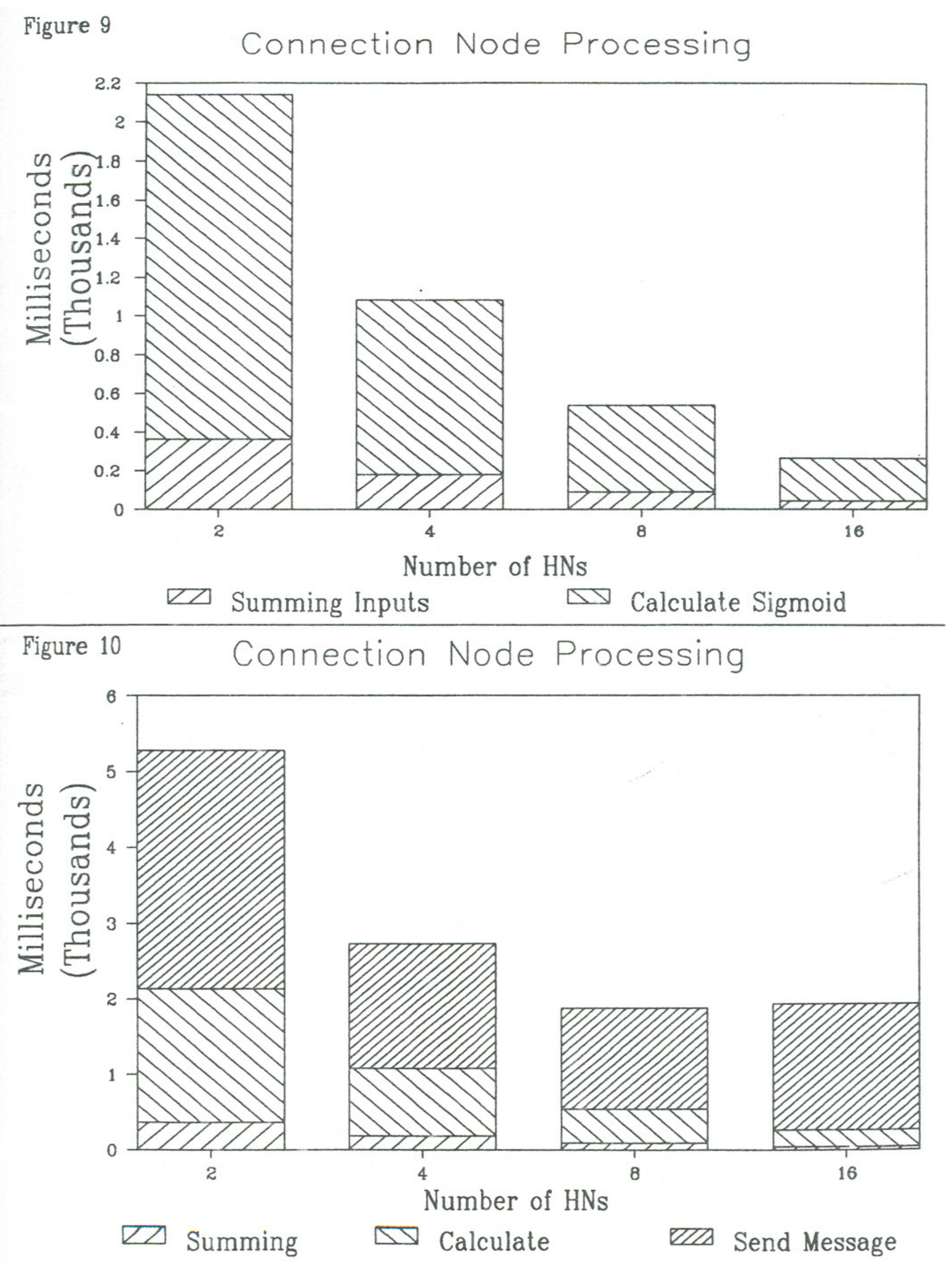
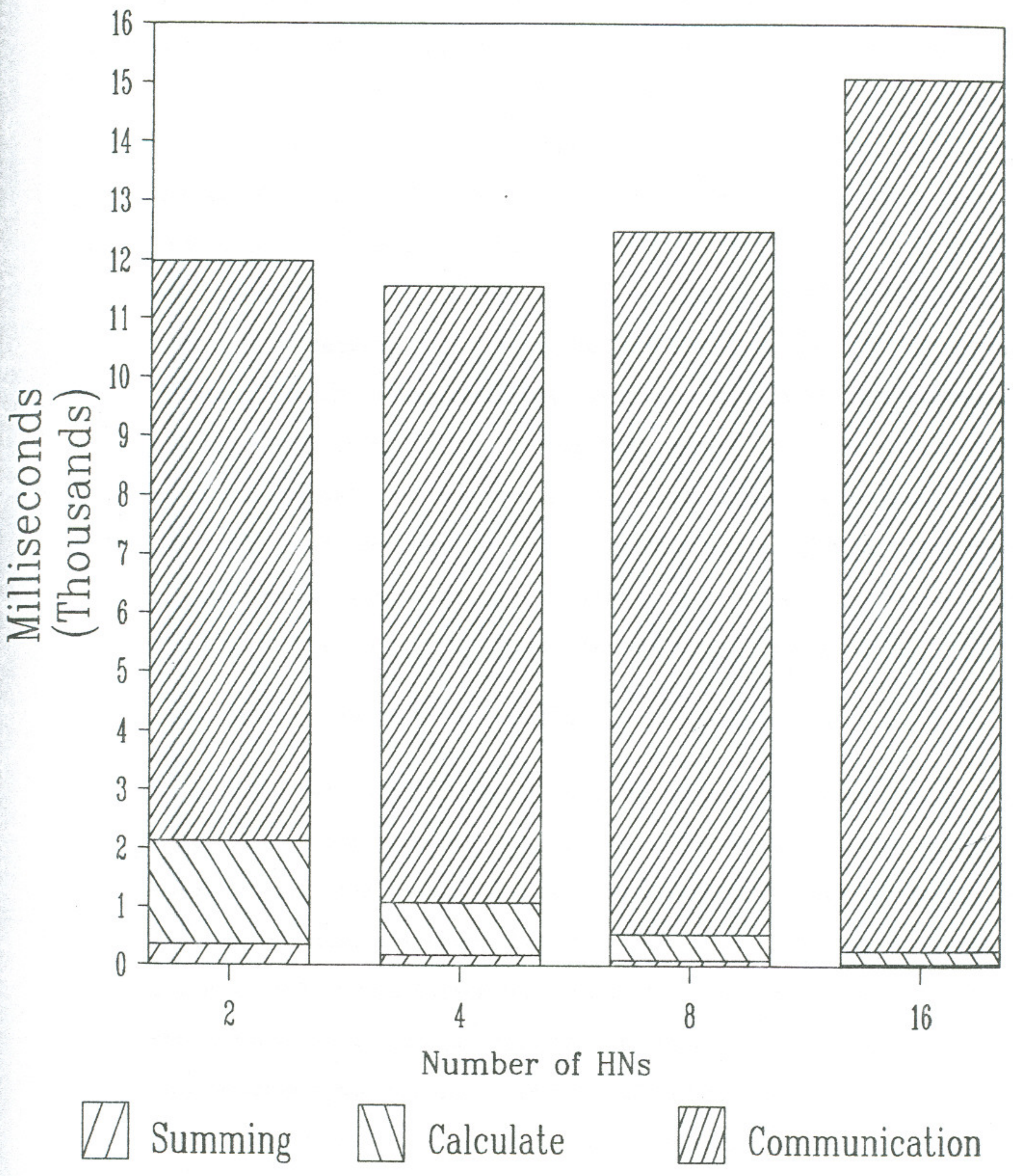


Figure 9
Connection Node Processing
Milliseconds (Thousands)
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
2
4
8
16
Number of HNs
Summing Inputs
Calculate Sigmoid
Figure 10
Connection Node Processing
Milliseconds (Thousands)
6
5
4
3
2
1
0
2
4
8
16
Number of HNs
Summing
Calculate
Send Message

Figure 11



The performance changes a little when the additional step of putting the results onto the hypercube interconnect is added. The effect can be seen in Figure 10. Here we still see the positive results of additional processors, but this has a limit that appears to be reached when eight processors are involved. Beyond that, the speedup begins to level off. Also in the larger networks there is a spillover effect that may be accounted for by collisions beginning to occur on the ethernet bus. The bus contention causes the communication times to increase by an order of magnitude from what is experienced with light message traffic. Time is consumed contending for the scarce bus resource. Finally the results become different altogether when the time taken to receive messages is added as shown in Figure 11. The parallel speedup witnessed in the solving of the original problem becomes overwhelmed by the message passing overhead. We are solving the original problem faster with the introduction of parallelism, but we are paying for the parallelism with overheads that are an order of magnitude larger than the speedup returns!

11.3. Solving the parallel problem

We have traded in our original problem of updating links faster for a larger less tractable problem involving interprocessor communication. We have decreased the processing time of our original problem correctly with a viable parallel algorithm, but throughput results are poor overall. In order to look at this larger problem, further timing was done to partition the message receiving portion of the problem. Some time is used by interprocess communication, some time is used doing such things as looking up CNs in tables. For further study, timing steps were introduced to

divide the receipt of messages into several pieces. First the time involved in actually receiving the message. Receiving a message includes the opening of the channel, the receipt of message and the closing of the channel. The second portion of the message cycle involves forwarding the message to other HNs within the hierarchy. The final contribution to the cycle is the table lookup operation to determine if a CN is affected and the updating of the affected CN's links. The results are shown in Figures 12 and 13. The portions performed within the PNs are once again measured in hundreds of milliseconds, the message receipt step remains several thousands of milliseconds long.

The analysis of the simulator cycle has identified that the majority of time is spent passing messages. Message passing is handled by the operating system and the hardware, and lies outside the programmer's control. As the simulator is currently designed, no amount of work on the internal algorithms will significantly affect throughput. We have hit a resource limit of the Intel Hypercube. The only hope for increasing speed is to begin to manage the scarce interprocess communication resource. Even this can only help to a point.

11.4. Managing the scarce interprocess communication resource

Several steps could be taken to minimize the amount of message passing the simulator needs to do. All of these will involve some compromise of the original design. One of the most straightforward changes would be to send messages in packets instead of individually. As a series of messages are processed, the resulting output would be placed in a message buffer. The

Figure 12 Parallel Portion of Receive Call

Milliseconds

200
190
180
170
160
150
140
130
120
110
100
90
80
70
60
50
40
30
20
10
0

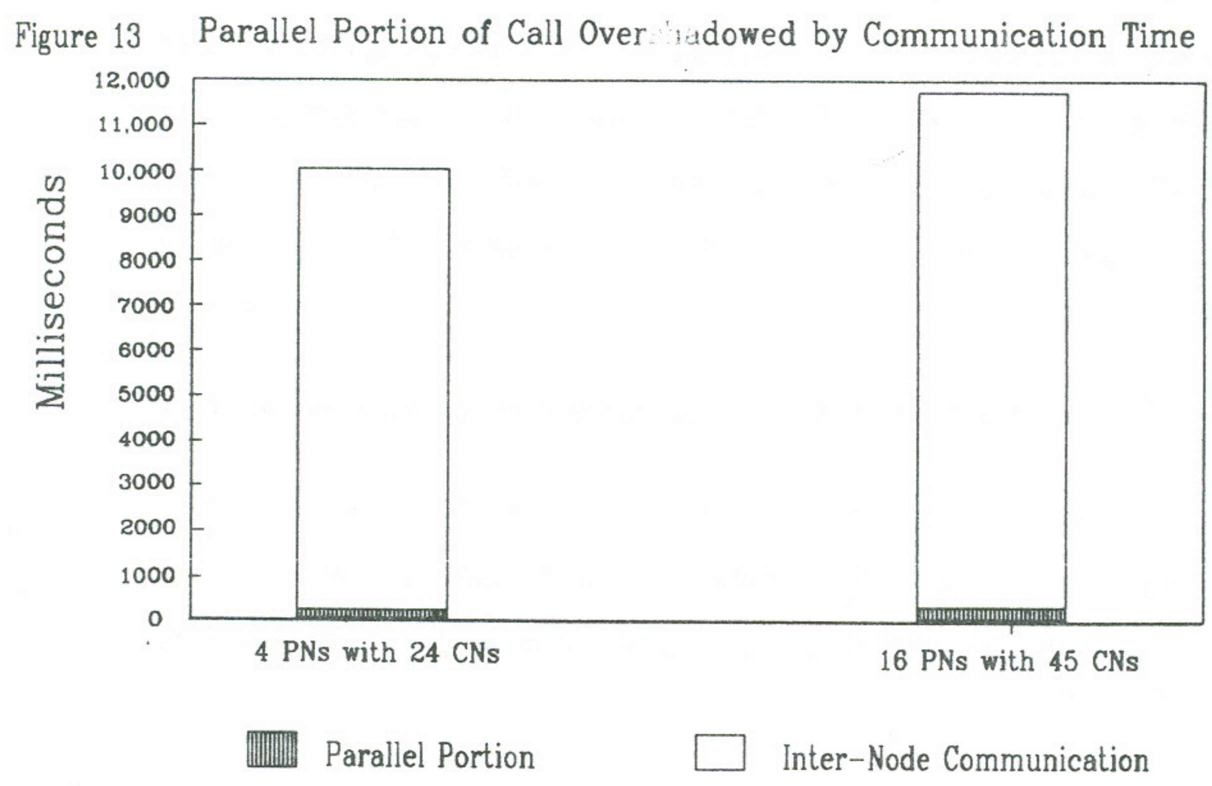
4 PNs with 24 CNs

16 PNs with 45 CNs

Relay Msg

Update Link

Figure 13 Parallel Portion of Call Overshadowed by Communication Time



buffer would not be sent until a specified number of messages were ready to be sent, or at the end of a message cycle if the buffer limit was not reached. Similarly, in the step in which a process communicates inputs to other members of the hierarchy, the messages could be sent in packets. All of these changes would create a new relationship between the simulator's emulation of the hierarchical bus and the hardware bus being modeled. The simulation would not relate to the final bus. There would have to be some way to account for the fact that messages from a particular PN would be arriving all at the same time. The update of CNs in all of the PNs would occur in different order. All of a particular HN's messages would arrive at the same time, be processed, and potentially a packet of results could end up being sent out. Finally there is the possibility that certain HNs messages would always be last. These effects would need to be studied and allowances made in the analysis to extrapolate useful performance data for the architecture under study. A smaller point is that it will take some time to manage the creation of packets. Since it is done in parallel, it will be positively affected by additional HNs, but it is still some additional time. In many ways the drawback of decreased performance is preferable to the complexity one would introduce in solving the resource problem.

11.5. A New Mapping of the Application onto the Hypercube.

Another method that may prove useful in upgrading the performance of the simulator on the hypercube goes back to the original mapping of the problem onto the cube. Since the hierarchies were set up to include eight

PNs, the final message passing algorithm ended up including messages that in the worst case needed to be passed from HN to HN three times to reach their destination. With the message passing identified as the primary bottleneck, it might be useful to limit the hierarchies to four PNs each, which is the number of HNs that can be communicated to by the bus process without having two or three hops (message passing between HNs to get to another HN) involved in the passing of information. The use of a new mapping would not compromise the serial nature of message passing, although it would change the architecture. Since the architecture is built to be scalable, simplification is a less troublesome compromise than message management. A new mapping involving four PNs would allow more hierarchies to be simulated on the 32 HN hypercube so it would also allow more tests involving problems that use a greater number of hierarchies. The message passing traffic would be reduced from ten messages required per output message, to five messages required. The reduction of messages would affect performance positively. The new mapping probably is not enough to actually solve the message passing overhead problem by itself. The new mapping will reduce the number of messages by one half. The resulting speedup should be a factor of 2. But the message passing overhead is an order of magnitude larger, not simply a factor of two larger. The problem must be tackled by providing an interconnect an order of magnitude faster then the existing interconnect available on the iPSC/1.

12. CONCLUSIONS

12.1. Usefulness of Parallelism in Solving Neural Network Problems.

A general conclusion is that parallelism will help forward the research of neural networks by providing a method to solve the large amount of processing in the neural model. The usefulness of parallelism will be limited by the ability to share data between processors.

12.2. Sizing Resource Needs.

Another important conclusion about the architecture under consideration involves the sizing of the storage for use in holding CNs in the simulator. Since "come from" addressing causes the number of messages sent to expand at a much slower rate than the complexity of the network overall, by allowing one message to be received by many CNs simultaneously there are economies of scale. These economies are not realized with small networks, since the broadcast bus will carry the same overhead essentially for small networks that it has for large networks. The larger the network one can simulate, the greater the efficiency. So an important resource for the final silicon implementation will be storage at the processor level. The greater the storage, the larger the network, and the more efficient the overall system. I would recommend storage at the processor level be considered at the several megabytes level. Several megabytes per processor will allow the kind of networks that are needed to benefit from the overall architecture.

12.3. Effects of Hierarchies on Performance

A observation on how mapping networks to different numbers of hierarchies affects its performance. Here results are somewhat less obvious, because the message passing so overwhelms the other portions of the test that differences between performing the test using one hierarchy and two are not very different. There is a trend that shows the fastest simulator times were reached when the problem was spread between more than one hierarchy. The serial nature of the broadcast bus is one factor helped by using two hierarchies. In effect if two hierarchies handle the message passing, the message passing begins to display some concurrency too, which results in a faster message passing stage and the simulator runs more quickly. Larger networks would exhibit this characteristic more so than small networks. The best results were obtained using four processors per hierarchy. The actual number of processors per hierarchy is probably dependent on the hypercube as much as the design of the broadcast bus, so it is difficult to generalize to what would be the optimum number of processors on a bus. Number of processors used is an implementation consideration that should be made modular. Let the need for storage override any desire to include more processors on the bus. Let the need for a faster overall bus override the need for many processors on a broadcast bus.

12.4 Time Stepped Synchronization

The use of a time stepped synchronization scheme was found to be effective in handling the simulator's need for a method to keep all portions of the

simulator on the same cycle. It was easy to adapt to a problem and was easy to implement. These characteristics are offset by the loss of some fraction of the parallelism available. This loss was not noticeable compared to other performance issues.

Bibliography

[Bah88] BAHR, C., "ANNE: Another Neural Network Simulator", Tech. Report CS/E-88-028, Dept. of Computer Science/ Engineering, Oregon Graduate Center, Beaverton, OR, August 1988.

[BHJ88] BAHR, C., HAMMERSTROM, D. and JAGLA, K., Concurrent Neural Network Simulation: Two Examples Within A Single, Integrated Neural Network Hardware Development Environment, IASTED Applied Simulation and Modelling Conference, Galveston, Texas, May 1988.

[Bai88] BAILEY, J., "A VLSI Interconnect Structure for Neural Networks," Ph.D. Dissertation, Dept. of Computer Science & Engineering, OGC, 1988.

[BaH86] J. Bailey and D. Hammerstrom, "How to Make a Billion Connections", Tech. Report CS/E-86-007, Dept. of Computer Science/Engineering, Oregon Graduate Center, Beaverton, Oregon, July 1986.

[ChMis79] Chandy, K. M., and Misra, J., Distributed simulation: A case study in design and verification of distributed programs., IEEE Trans. Softw. Eng. SE-5,5.

[Fan86] FANTY,M. 1986. A Connectionist Simulator for the BBN Butterfly Multiprocessor, Department of Computer Science, University of Rochester, Rochester, NY 14627

[FFGL88] FELDMAN,J., FANTY,M.,GODDARD,N.,LYNNE,K. 1988. Computing with Structured Connectionist Networks. Communications of the ACM, February 1988, Volume 31 Number 2.

[HoB86] HORVATH,J.,BOWER,J. 1986. Neural Network Modeling on the Hypercube Concurrent Processor Jet Propulsion Laboratory, Pasadena, California

[Jef84] JEFFERSON,D. 1984. Implementation of Time Warp on the Caltech Hypercube Jet Propulsion Laboratory, Pasadena, California

[Joh88] JOHNSON, M. A., "NDL User's Manual," CSE Technical Report, Oregon Graduate Center, Department of Computer Science/Engineering, Beaverton, OR, July 1988. In preperation

[Mis86] MISRA,J. 1986. Distributed Discrete-Event Simulation Ph.D dissertation, Computer Science Dept., Univ. Texas at Austin, Austin Tex.

[RuH88] RUDNICK,M.,HAMMERSTROM,D. 1988. "Physical Broadcast Structure", Technical Report No. CS/E-88-OXX, Dept. of Computer Science/Engineering, Oregon Graduate Center, Beaverton Oregon.

HAS USER MANUAL

Kevin N. Jagla
Oregon Graduate Center
Dept. of Computer Science & Engineering
Beaverton, Oregon 97006-1999
(503)690-1151

HAS
USER MANUAL

Version 1.0
May 1,1989

User Manual for
HAS - Hierarchical Architecture Simulator
A program to simulate a proposed architecture

1. General Description of Usage

HAS is a simulator used to asses the performance of different neural networks mapped onto the Broadcast Hierarchy simulation system [Bai88]. It is built to operate within a neural network development environment developed at Oregon Graduate Center by the CAP (Cognitive Architecture Project). In this environment the neural network itself is built, mapped to the target simulation system, and debugged, using other tools. When the network arrives at this simulator it is operational. This simulator measures its performance on the proposed simulation system. A good explanation of the steps used to build a neural network is contained in the "User's Manual for ANNE" by Casey Bahr [Bah88].

To proceed the following pieces of the puzzle must be at hand:

A. A BIF file of the network to be simulated. [Bah88]

B. A user function procedure written in c.

C. A set of files used by the HAS simulator. One is an input file containing the initial inputs to the network, another is a configuration file called "setupfile".

The next sections will deal with how to construct these pieces.

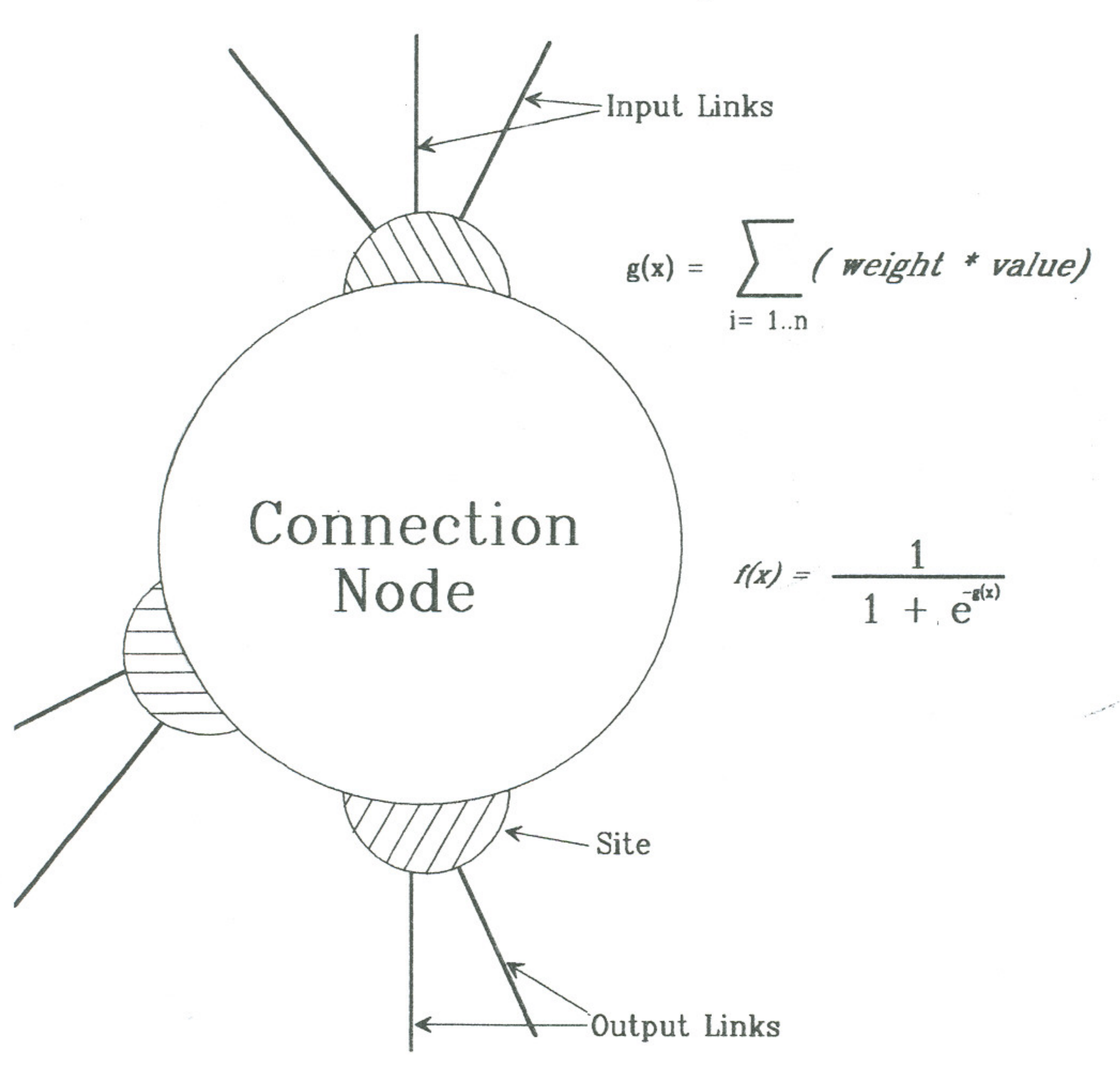
2. BIF Network File

A BIF file contains information about the neural network being simulated [Bah88] and also some information on the mapping of the neural network onto the target architecture. A model of a connection node for the simulator can be seen in Figure 1. In this model there are four main areas. The first is the input links, these are used to describe the connections between the connection nodes. The second is the site areas. These allow different functions to be initiated based on the site. For instance one site could be for negative inputs and a site function could be developed to reflect this. The third area is the activation function. This can be any function desired and can include a threshold value that determines if output will be initiated. Last is the output link. The output link directs the output to the proper hierarchy.

The original network is developed using the NDL design tool. For a complete explanation of NDL see reference [Jon88a]. A file that has been developed using NDL should then be mapped to the intel hypercube using the MAPPER [Bai88]. The most important difference between a BIF file mapped to the HAS simulator and preceding BIF files is the transformation that occurs when the initial BIF file is mapped to the Broadcast Hierarchy. Most importantly the number of output links in the file drop dramatically, and most connection nodes have only one output link, or at most four. The four possible output links have in the cn field a number between 0-4.

Figure 1

A Connection Node



Once the BIF file is ready for use it is included in the directory where the simulation will be running, and is input by the cube manager process and read down into the cube processors.

3. User Code

In the final design the set of functions that will be used in the processors will be downloaded as a separate file into a memory area. These downloaded functions will be written by the designers of the neural network. HAS node images el.o need to be linked with a C procedure called user_fx.o to create a complete node process. User_fx.o is the section of code developed to emulate the connection node processing steps. To do this, use the makefile supplied with the code.

3.1 Example of a user_fx.

An example of the necessary structure for the user function is included in Appendix A. The example controls a feed forward neural network with no learning step included. This would be emulating a neural network part that had been trained on ANNE or another network emulator, then downloaded onto the Broadcast Hierarchy. The most important point is to notice that several modes are used to specify which type of function is to be called. The user_fx is called four times during the simulator's cycle. During each pass, the function's parameter list contains the start addresses for the appropriate CN record and its Site being addressed and the Link being used. The values of these records are used along with some temporary variables. The results are evaluated and the changes stored back in the database.

3.2 Constructing the executable image

The system has its own makefile available. The user develops a user_fx similar to the one above, then using make creates and object file user_fx.o. Issue the command make user_fx and the compiler will be invoked with the proper switches creating the object image. Or Issue command make el0, and the makefile will create the executable image and automatically compile the new user_fx.c file.

4. HAS simulator support files

The first file that needs to be built is the file containing the input vector. This is done by feeding standard output messages into the system from the cube manager. The file is constructed using an editor. For each input node a record is created. A record appears as such: "-1 500 0". The -1 refers to the connection node address in an input link. The 500 is the value actually multiplied by the weight of the link. The final field is the time stamp. Since this is the input vector it is arbitrarily assigned a time stamp of 0.

The second file constructed is the configuration file or "setupfile". The configuration file contains two different types of records. The first record is designated with an "X". It carries information concerning the size BIF file that is to be read into the simulator:

4.1 Memory Allocation Record

Record Layout

Fields	1	2	3	4	5	6	7	8
	0	X	4	24	48	1200	0	0

Field 1 Value shown 0

This field is the HN destination. In this case this record will go the processor node 0.

Field 2 Value shown X

This is used in the node to designate this is a record for allocating memory for the network database.

Field 3 Value shown 4

This is how many Types of Connection nodes there will be. Each node may store only those types it needs. Currently since there have been only four or so types of nodes, each node was sent all of the potential types.

Field 4 Value shown 24

The number of connection nodes to be stored.

Field 5 Value shown 48

The number of sites to be stored.

Field 6 Value shown 1200

The number of links to store.

Fields 7 and 8

Padding in this record.

4.2 Parameter Setting Record

The second record sets parameters within the simulator. It has the following layout:

Record Layout

Fields	1	2	3	4	5	6	7	8
	0	Y	16	4000	4	0	2	2

Field 1 Value shown 0

The HN destination for this record. In this case the record will be shipped to processor 0.

Field 2 Value shown Y

This field is used to designate that this record will be setting parameters for the simulator.

Field 3 Value shown 16

This field is used to specify how many HN's will be used in the simulation. In this case a 16 processor hypercube will be used for processing.

Field 4 Value shown 4000

This field is used to specify how many loops the timing procedure will do before setting a flag "no new messages". As part of the timing mechanizm for determining if it is ready for the next cycle, the process will loop in a read procedure probing for messages 4000 times then go increment a variable "no new message". The routine was originally expected to use a timing interrupt for this portion, but no timing signal was available on the node processor.

Field 5 Value shown 4

This field is used to specify how many times the "no new messages" flag will be set before the HN signals to the cube manager it is ready for the next step. Along with the preceding timing loop the limit for "no new message" is 4. Using the last two variables the simulator will wait in a timing loop 4000 cycles long 4 times before deciding that no new message are going to arrive during the current simulator cycle.

Field 6 Value shown 0

This field set the time stamp of the node processors. Most would be set to 0.

Field 7 Value shown 2

This field sets how late a time stamp is acceptable for processing. So if you are on step 11 and a message comes in with step 8, it is not processed. If a message with the time stamp 9 comes in it is acceptable.

Field 8 Value shown 2

This field determines how early a time stamp is acceptable for
processing. So if you are on step 11 and a message comes in with time stamp 14, it is not processed. If a message with the time stamp 13 comes in it is acceptable.

5. Starting the Simulator

A script file has been constructed to aid in the starting of the simulator. The name of the file is starthas. The user needs to have the executable image for the nodes, pn0 created by the make file. There needs to be a copy of the cmgr1 executable in the directory also. The inputfil and the setupfile need to be in the directory with the simulator, and a copy of a BIF file to be simulated. The user then types "starthas". The script file first reloads a copy of the operating system into all of the nodes. This is a mild type of initialization. It usually works. If nothing happens, then perhaps you have forgotten to issue the "getcube" command. If the cube is unable to initialize it will return with a message that says "unable to initialize cube". Other possible messages are: "node X does not respond", or "checksum error in node 10". All of these messages mean that the cube was unable to get off the ground. The best thing to try at this point is a "load -R". This issues a hard reset to the machine. Usually this will be successful. If it is not, try again. If after a few times nothing seems to be working, your only option is to notify the systems administration. Usually they can re-initialize the cube and get it working again.

If the script file does work it will say "load successful" three times as it loads in copies of the executable code into the nodes. Then the cmgr1 will be started. After this it will ask for the name of the BIF file. You enter the name of your BIF file and the simulator will immediately begin to load the file.

Appendix A

The following is an example of user_fx.c. It uses a summation of the inputs followed by the use of a sigmoid function to generate the next cycle's outputs. All of the code within each mode step may be changed by the user to suit their particular need.

```
/*                                                        */
/*  user_function - This is supplied by the user and      */
/*  calculates the needs of the network node. It gives the*/
/*  user a full copy of the connection node, and also a   */
/*  copy of the input message is available to the user in */
/*  the global buffer buf.                                */
/*  all changes to the connection node record occur here. */
/*********************************************************/
#include "common.h"
#include <math.h>
#include <stdio.h>
#include "escan.h"
#define E 2.7182818

void user_function(C,T,S,L,buf)
   struct CNode *C;
   struct CNtype *T;
   struct sitemem *S;
   struct linkmem *L;
   char *buf;
   {
      extern struct step_variables step;
      extern int apid;
      extern int userfx_mode;
      extern char cmgr_buf[256];
      extern void send_output();
      extern void main();
      extern int send_cmgr();
#if CUBE
      extern int sprintf();
#endif
      int mes_cnx, mes_value, mes_time;
      int siteval, current_inval;
      int cn_index;
      double dblval;
      float wt, inval;
      int output_value, out_pid;
```

```
     /* This section is used in debug to insure the records are */

     /* properly retrieved from the database.                  */

       if( apid == 100 )
       {
       /*sprintf(cmgr_buf,"Apid %d Cn type %d Cn index %d cn state
%d \n",apid,C->type,C->index,C->state);*/
       /*send_cmgr(0);*/
       /*sprintf(cmgr_buf,"Apid  %d  type_index  %d  Type_name  %s
Initpot %d \n",apid,T->index,T->name,T->initpot);*/
       /*send_cmgr(0);*/
       /*sprintf(cmgr_buf,"Apid  %d  Site  name  %d Site iotype %d
Site value %d \n",apid,S->name,S->iotype,S->value);*/
       /*send_cmgr(0);*/
       /*sprintf(cmgr_buf,"Apid %d Link index %d Link  cnx %d Link
weight %f \n",apid,L->index,L->cnx,L->weight);*/
       /*send_cmgr(0);*/
       }
   /* A variable userfx_mode is used to identify which stage of the */
   /* processing is occuring. Mode == 1 is the stage where messages */
   /* are begin received. At this point in time the inputs are begin*/
   /* stored in their input links. Mode 1 continues until all of the*/
   /* inputs are received, and the node has let the cube manager    */
   /* know that it is ready to proceed. Mode == 2 begins when the   */
   /* cube manager signals for the beginning of next cycle. The node*/
   /* then begins by summing all of the inputs from the Links into  */
   /* the S->siteval. Next Mode == 3 occurs in which the activation */
   /* function is performed on the stored S->siteval. Since this    */
   /* network has only one input site, only one is processed.       */
   /* The result of the activation function is stored in the        */
   /* C->output field. When this is completed, the next step is     */
   /* Mode == 4. In Mode 4 the Activation values are formatted into */
   /* a message and the message is output to The BroadCast Hierarchy*/
   /* These messages become the messages recieved in Mode == 1, and */
   /* the cycle continues.                                          */
   if (userfx_mode == 1)
      {
       sscanf(buf,"%d %d %d",&mes_cnx, &mes_value, &mes_time);
       /*sprintf(cmgr_buf,"apid %d  mes_cnx %d  mes_value %d mes_time %d
\n",apid,mes_cnx,mes_value,mes_time);*/
       /*send_cmgr(0);*/
         L->inval = mes_value;
         /*sprintf(cmgr_buf,"Apid %d mode  %d  C->index  %d  L->inval %d
L->weight %f \n",apid,userfx_mode,C->index,L->inval,L->weight);*/
         /*send_cmgr(0);*/
      }
   else if (userfx_mode == 2)
      {
      /*sprintf(cmgr_buf,"apid %d S->siteval %d",apid,S->siteval);*/
      /*send_cmgr(0);*/
```

```
        siteval = S->siteval;
        wt = L->weight/1000;
        current_inval = L->inval;
     /*sprintf(cmgr_buf,"Apid %d  mode %d  L->index %d siteval %d wt %f
current_inval
%d",apid,userfx_mode,L->index,siteval,wt,current_inval);*/
     /*send_cmgr(0);*/
        inval = (int)((float)current_inval)*wt;
        /*sprintf(cmgr_buf,"apid    %d    Cnode:     %d     inval    %f
\n",apid,C->index,inval);*/
        /*send_cmgr(0);*/
        siteval +=inval;
        S->siteval = siteval;
        /*sprintf(cmgr_buf,"apid,    %d    mode    %d    S->siteval    %d
\n",apid,userfx_mode,S->siteval);*/
        /*send_cmgr(0);*/
     }
  else if(userfx_mode == 3)
     {
        siteval = S->siteval;
        /*sprintf(cmgr_buf,"userfx_mode %d  S->siteval  %d  C->index %d
\n",userfx_mode,siteval,C->index);*/
        /*send_cmgr(0);*/
        /* Here for the 8x8 it is necessary to distinguish */
        /* between the first set of nodes and all others.  */
        if(C->index < 8)
          {
            C->output = siteval;
          }
        else
          {

              dblval = ((double)(siteval/1000.0));
              dblval = (1.0/(1.0 + exp(-1.0 *dblval)));
              C->output = (int)(dblval*1000);
          }
        /*sprintf(cmgr_buf,"Apid   %d   userfx_mode   %d   C->index  %d
C->output %d \n",apid,userfx_mode,C->index,C->output);*/
        /*send_cmgr(0);*/
     }
   else if(userfx_mode == 4)
     {
        if(C->output != 0)
           {
           sprintf(buf,"%d %d %d",C->index,C->output,step.time_stamp);
           send_output(L->cnx);
           /*sprintf(cmgr_buf,"Apid   %d    L->cnx    %d    outbuf    %s
\n",apid,L->cnx,L->weight,buf);*/
           /*send_cmgr(0);*/
          }
     }
```

```
  else
      {
         /*sprintf(cmgr_buf,"Apid    %d    incorrect    userfx_mode   %d
\n",apid,userfx_mode);*/
         /*send_cmgr(0);*/
      }

    }
```

References

[Bah88] Bahr, C., "ANNE User Manual," Tech. Report CS/E-88-029, Dept. of Computer Science/Engineering, Oregon Graduate Center, Beaverton, OR, 1988.

[Bai88] Bailey, J., "A VLSI Interconnect Structure for Neural Networks," Ph.D. Dissertation, Dept. of Computer Science & Engineering, OGC, 1988. In preparation.

[Joh88a] Johnson, M. A., "NDL User's Manual," CSE Technical Report, Oregon Graduate Center, Department of Computer Science/ Engineering, Beaverton, OR, July 1988. In preperation.