HAS USER MANUAL

Kevin N. Jagla

Oregon Graduate Center
Department of Computer Science
and Engineering
19600 N.W. von Neumann Drive
Beaverton, OR 97006-1999 USA

Technical Report No. CS/E 89-016

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.

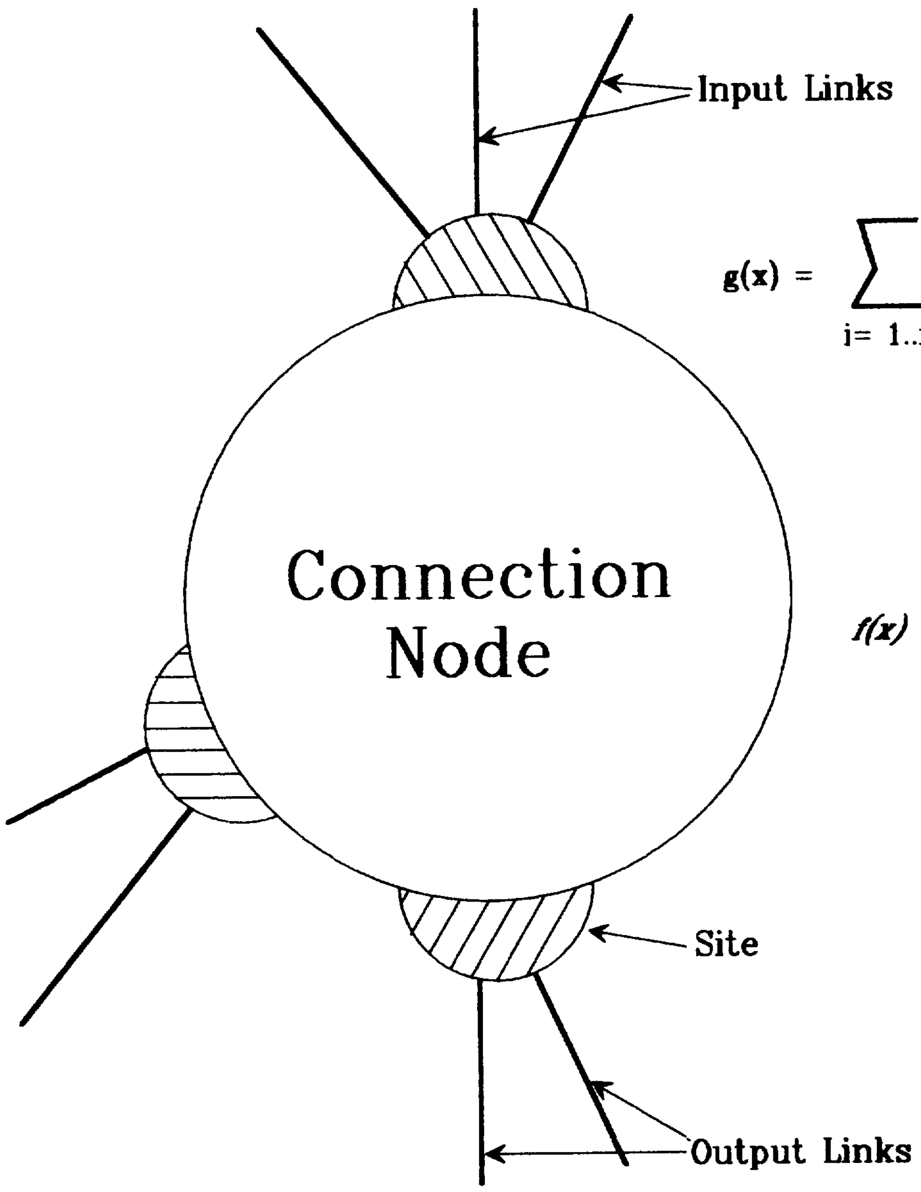
2. BIF Network File

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



$$g(x) = \sum_{i=1..n} (\textit{weight} * \textit{value})$$

$$f(x) = \frac{1}{1 + e^{-g(x)}}$$

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 e10, 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.