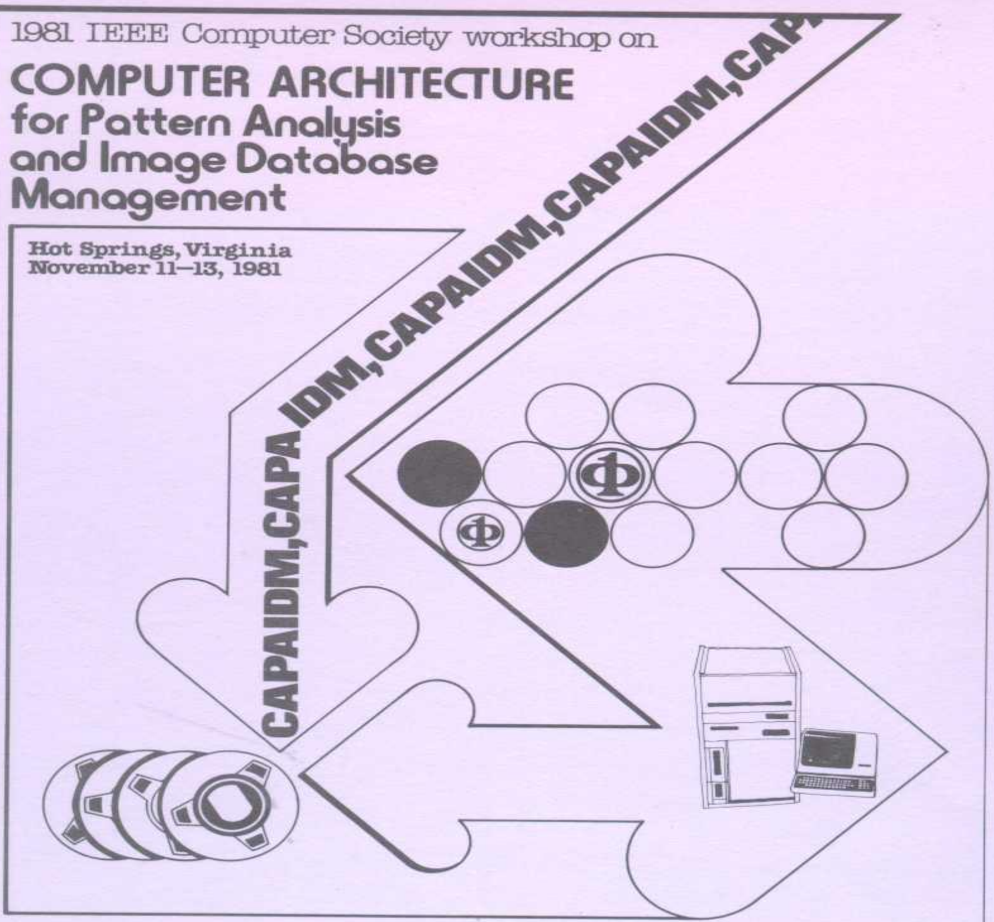
③

1981 IEEE Computer Society workshop on

COMPUTER ARCHITECTURE for Pattern Analysis and Image Database Management

Hot Springs, Virginia
November 11–13, 1981



IEEE Catalog No. 81CH1697-2
Library of Congress No. 81-82808
IEEE Computer Society Catalog No. 378

IEEE COMPUTER SOCIETY PRESS

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

ARTAGA 48

Reprinted from IEEE COMPUTER SOCIETY WORKSHOP ON COMPUTER ARCHITECTURE FOR PATTERN ANALYSIS AND IMAGE DATABASE MANAGEMENT, November 1981

A HETERARCHICAL MULTI-MICROPROCESSOR LISP MACHINE

Adolfo Guzmán

Computing Systems Dept., IIMAS
National University of Mexico
Apdo. Postal 20-276
México 20, D.F.

ABSTRACT

This article presents the architecture of a parallel, general purpose computer that uses Lisp as its main programming language. It is built from several dozens of microprocessors (Z-80's), each of them executing a part of the program.

The machine processes in parallel programs written in high level languages capable of being expressed in the lambda notation (applicative languages). It is formed by a collection of general purpose processors which are weakly coupled and without hierarchy among them. Asynchronous computation is permitted due to each processor evaluating a part of a program.

The architecture presented here has been developed for the Lisp language, though other applicative languages can be utilized. The implementation of function calls, argument passing, and sequencing of tasks are a part of hardware rather than software. The primitive operations of Lisp are executed by processors, of which each is a Z-80 microprocessor that has been programmed to do these operations.

Doing I/O is avoided in the machine for it operates through a general purpose minicomputer. Moreover, all interactions between the machine and the user(s) are done by the normal operating systems of the mini.

I. INTRODUCTION AND PROJECT GOALS

Within the Computing Systems Department, there are about six full-time people that are involved in a project of building a machine. The project is called AHR (Arquitecturas Heterárquicas Reconfigurables).

Goals

Some of the goals of the Project AHR are:

* To have a machine in which it will be possible to develop software and parallel processing languages. Currently, the AHR machine supports parallel pure Lisp.
* To explore new ways to perform parallel processing.
* For students to use this machine as a tool for learning and practicing parallel concepts in hardware and software.

Main Features

The AHR machine has the following characteristics:

* parallel processors.
* general purpose.
* All the processors are heterarchical. This means there is no 'master' processor, or controller.
* asynchronous operation.
* Lisp is its main programming language.
* processors do not communicate directly one to another. They simply 'leave the work' that is needed to be done for the next processor without having to tell it what is expected from it.
* gradually expandible. As additional computing power is needed more microprocessors can be aded [9].
* no input/output. This is conducted by a minicomputer to which the AHR machine is attached.
* no operating system (software). The majority of the Lisp operations, as well as the garbage collector, are written in Z-80 machine language. Also, special hardware helps to handle list structures, free-cells lists, and queues.
* the AHR machine works as a slave to the general purpose minicomputer.

Project Status

There are now several versions of the AHR machine. The design of the first version of the machine, Version 0 [see reference 3] was not built, but it was used to produce Version 1 [12], which was simulated using the language SIMULA. This second version, Version 1, is being built and it is expected to be operational by the end of 1981 [5]. Afterwards a faster version will be built, possibly incorporating changes and ideas that resulted from our experiences with Version 1.

This last version will be used to try to attain the aboved-mentioned goals. Picture processing, finite element methods, engineering calculations, and distributed processing are also some of the expected uses of the machine.

Parallel Evaluation and Functional Notation

The AHR machine works with pure lisp, without

SETQ's, GOTO's, Label's, RPLACA, and other operators. It obtains its parallelism by parallel evaluation of the arguments of functions. This is in accordance with the rule for evaluation of a function:"to evaluate a function, the arguments have to be already evaluated". That is, evaluation occurs from bottom up, or from the inside to the outside of the expression. For instance, in f(a,b,g(u,g(x,b))), first x and b are evaluated; then g of them, in parallel with u; afterwards g of the result, in parallel with a and b.

Recursion is handled [3] by substituting the function name ("FACTORIAL") by its function definition (LAMBDA (N) (IF (EQ N 0) 1. . .)) when evaluating it.

II. THE CONSTITUENTS OF THE MACHINE

The parts of the AHR machine are described in this section (refer to figure 2 "The AHR machine"); section III explains how the machine works.

The Grill

Also known as "active memory", holds the programs that are being executed. Once in the grill, a program is evaluated and being transformed into results.

Programs residing in the grill are in the form of nodes and are about to be evaluated, as figure 1 illustrates. Each node is pointed to by its "sons" (its arguments), and its nane (números de argumentos non-evaluados) field contains the number of nonevaluated arguments. When nodes have nane = 0, they are ready for evaluation.

The grill consists of 2^{19} words of 32 bits and is divided logically in nodes, each with 7 words. Version 1 will have 8K words, with an access time of 55 nanoseconds.

Passive Memory

This memory holds lists and atoms; it also holds partial results and parts of programs that are not being executed at the moment.

In the beginning of the process the programs to be executed reside here in passive memory, and they are copied to the grill for their execution. As new data structures are built, representing partial results of the evaluation, they also come to reside in the passive memory.

It consists of upto 2^{20} words of 22 bits; it also has a parity bit. This memory contains the input ports, list space, output ports and atom space.

Version 1 will have only 64K words, with an access time of 150 nanoseconds.

Variable Memory

This memory contains pairs, a variable name and its value. It is organized as a tree, or a collection of a-lists, where each pair (variable, value) points to older pairs. It is accessed by the Lisp processors, and it is augmented (a branch of the tree grows) after each LAMBDA binding.

Since the evaluations are made in parallel, the a-lists grow in parallel as well. For instance,

```
(LIST (CONS (CAR A)
            (CDR B))
      X
      Y )
```

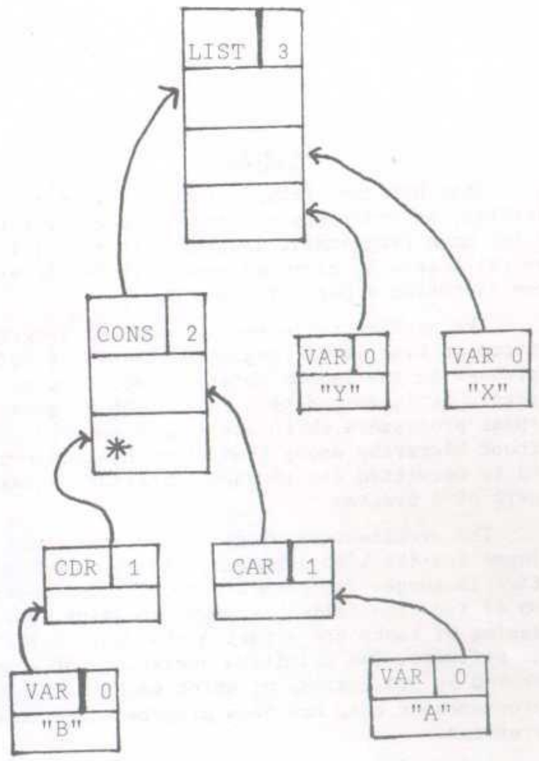


Figure 1

NODES IN THE GRILL

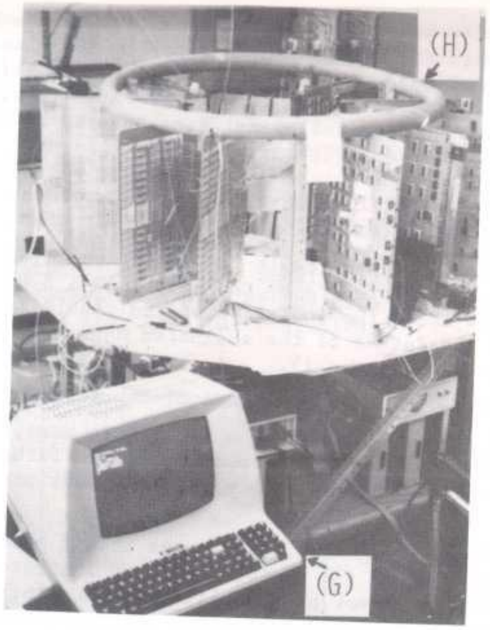
Above, the Lisp expression to be evaluated. Below, how it is structured into nodes, each node being a function or a variable. Each node shows a number: its nane, or number of non-evaluated arguments. When a node has a nane of zero, it means that such node is ready for evaluation.

Empty words are slots where the results of evaluation will be inserted. For instance, the results of (CDR B) will be inserted in the slot marked "".*

consider the following expression: **BODY0**

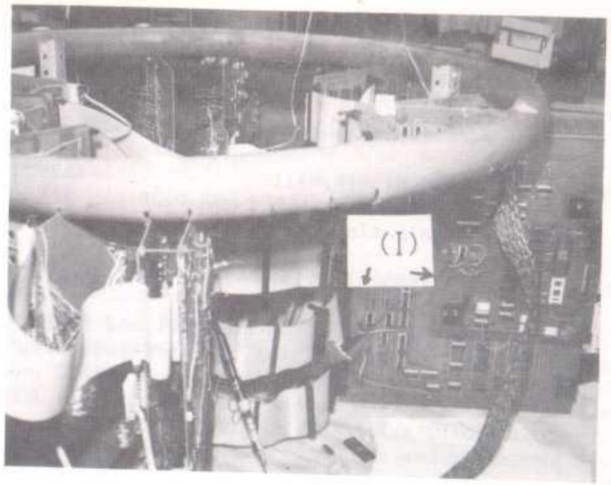
(list((lambda(X) BODY1) 3) (lambda(X) BODY2) 4))

that will be evaluated with the following



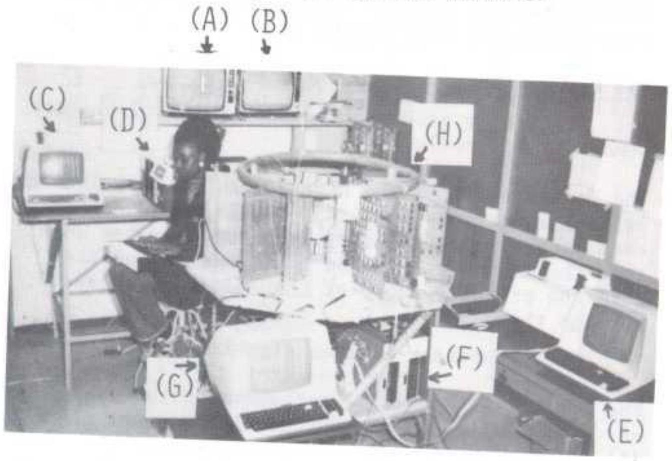
THE USER CONSOLE

In the foreground, the user console (G).
In the background, the AHR machine.



ONE LISP PROCESSOR

One of the several Lisp processors (I) of the AHR machine.



OVERVIEW OF THE AHR MACHINE

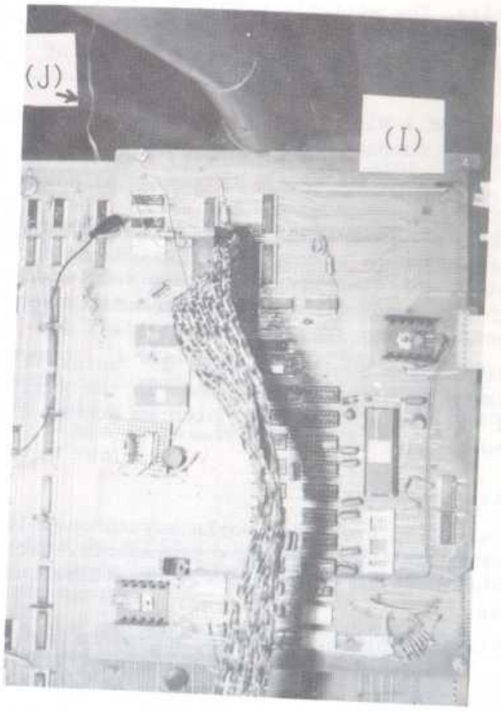
(A) and (B) are televisions that spy the contents of the private memory of the Lisp processors ((I) in the upper right picture).

(C) is the console of the Z80 microprocessor (D) that helps to debug the software of the Lisp processors (I).

(E) is the console of Version 0 of the distributor (F).

(G) is the user console. More than one console can be added: the AHR machine is a multiuser machine.

(H) is the upper part of the AHR machine.



DEBUGGING A LISP PROCESSOR

The software of the Lisp processor (I) shown is being debugged from a Z80 microprocessor ((D) in the picture at the left) through an "umbillical cord."

(J) is the line to the TV spy (A), from the Lisp processor.

a-list:

((X,A) (Y,B) (Z,9)) ALISTO

Then, when evaluating BODY1, the a-list is:

((X,3) (X,A) (Y,B) (Z,9)); ALIST1

and when evaluating BODY2, the a-list is:

((X,4) (X,A) (Y,B) (Z,9)). ALIST2

But since the evaluation of BODY1 and BODY2 can be carried in parallel (by two different Lisp processors), this means that ALIST1 and ALIST2 co-exist at the same time in variable memory, but BODY1 points to ALIST1 and BODY2 points to ALIST2. So each processor has its "appropriate" a-list to work with.

ALISTO grew in two directions, like a tree, giving rise to ALIST1 and ALIST2 simultaneously. Both ALIST1 and ALIST2 share ((X,A) (Y,B) (Z,9)) between them. This explains the affirmation that "the variable memory contains a tree of a-lists".

The variable memory consists of up to 2^{19} words of 32 bits. The variable memory also contains real numbers, in its lower half. In its upper half it has "environments", which are lists of cells of 5 words each.

Version 1 will have 16K words, with an access time of 150 nanoseconds.

The Fifo

The fifo is a first-in-first-out memory that holds pointers to nodes (in the grill) ready to be evaluated. The distributor fetches such nodes through the head of the fifo, while new nodes to be evaluated are inserted through its tail [5].

It is of a maximum size of 2^{19} words of 19 bits containing pointers to the nodes in the grill. Version 1 will be of 4K words, with an access time of 55 nanoseconds.

The Lisp Processors

Each Lisp processor works asynchronously, without communicating with other processors. Each of them knows how to execute every primitive function of Lisp.

The Lisp processors have access to the passive memory, (where lists and atoms reside), and to the variable memory, (where the values of variables are stored).

A Lisp processor is always either occupied (evaluating a node) or ready to accept more work (another node).

These active units are microprocessors (about several dozen of Z-80's). Nodes that are ready for evaluation are taken from the grill by these microprocessors and after evaluation return results (s-expressions) to the grill.

The Lisp processors get new work to be done from the distributor, through the high speed bus. This work comes as a node ready to be evaluated.

Only nodes with nane = 0 come up to the Lisp processors for evaluation. So, for example, (CAR '(A B C)') will evaluate to A. The node (CAR 'A B C)') has become the result A. After evaluation of this node, the following steps have to be performed by the distributor:

1.- Free the grill space that was occupied by the node (CAR'(ABC)').

2.- Insert the new result 'A' in the cell (of the grill) pointed to by the node (CAR'(A B C)'). That means the insertion of the result in a slot of the father of the evaluated node (see such slots in figure 1).

3.- Subtract 1 from the nane of the father.

4.- If the new nane (of the father) is zero, it writes in the fifo a pointer to the father, which means the father is now ready for evaluation.

Even though the distributor itself performs the above steps (1) to (4), they are initiated by the Lisp processor by signalling to the distributor that it has finished evaluating a node and that the results should be handled in the "normal termination" mode (12).

Notice that with this approach the grill doesn't have to be searched looking for nodes with nane=0, because as soon as they appear they are inserted into the tail of the fifo.

The Private Memory of the Lisp Processors

Each Lisp processor has 16K bytes of private memory (ram + rom).

The High-Speed Bus. The high-speed bus goes into the private memory of each Lisp processor and connects each processor with the distributor. The new node that the distributor makes available is inserted into the private memory of the selected processor, through this bus.

The processor is then signaled to proceed.

The Low-Speed Bus. It is not shown in the diagrams, nor is it explained any further in this article (see [5]). There are 16 bits to this bus; 8 of them indicate which Lisp processor is addressed, the other 8 bits carry data.

An additional use of the low speed bus is to transmit to the Lisp processors the number of a program that needs to be stopped or aborted.

It runs from the I/O processor (the mini or micro to which the AHR machine is connected) to each of the Lisp Processors. Prior to starting the machine each processor is loaded withprograms through this bus. Also, in the debugging stage, the slow bus is used to pass statistical information to the I/O processor. The low-speed bus is not used during normal execution of Lisp programs.

The Distributor

This piece of hardware provides communication of the grill with the Lisp processors. The distributor keeps in the fifo (a memory) an array of nodes ready to be evaluated. These nodes are made available, one of them in each cycle of the distributor, to the Lisp processors that are ready to accept new work. An arbiter decides which Lisp processor receives the node, after which an exchange is done (through the high speed bus) between that Lisp processor and the distributor, the processor accepting the node and releasing the result of the previous evaluation to the distributor. The distributor stores the result in the grill, (in the address indicated within the result). Specifically, this result is stored in a slot of the node which is the father of the node just evaluated.

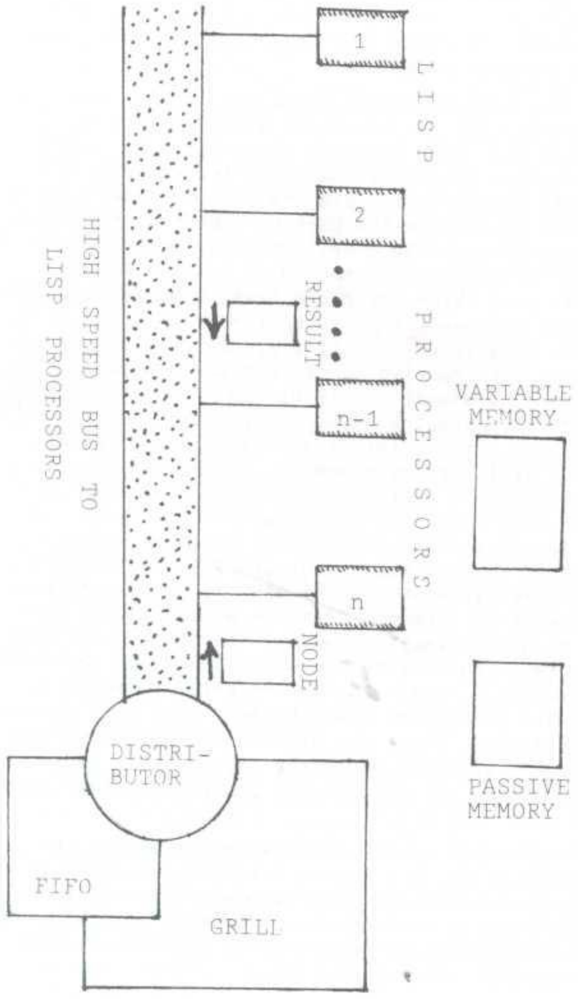


FIGURE 2

THE AHR MACHINE

Lisp processor 2 is ready to accept more work. The distributor fetches a node (to be evaluated) from the fifo and sends it to processor 2, while accepting the results of the previous evaluation performed by such processor. That result is stored in the grill, in a place indicated in the destination address of the result.

Such exchange of new work↔previous result is performed at each cycle of the distributor.

Version 2 of the AHR machine will gain speed over Version 1, mainly by building a faster distributor.

The Lisp processors also have access (connections not shown) to the variable and passive memories.

The Arbiter. If several Lisp processors become ready to accept more work, the arbiter (a hardware) selects one of them, to receive the node made available by the distributor.

If every process is busy, the cycle of the distributor is wasted, since no processor accepts the node the distributor has made available.

The I/O Processor

As it has been said, the AHR machine can be seen as a peripheral of a general purpose minicomputer. But this mini can also be considered as a peripheral of the AHR machine; therefore we speak of such mini as the I/O processor.

I/O will be described in the following section.

III. FUNCTIONING OF THE AHR MACHINE.

Input

The user uses a terminal of the mini (I/O processor) which is master of the AHR machine. He uses a common editor, disks and the normal operating system of the mini. When the user is ready to run a program, he loads it from disk into a part of the address space of the mini (which is really the passive memory of the AHR machine-See figure 3). In this way, the program is loaded as list cells in the passive memory. A signal from the I/O processor to the AHR machine causes Lisp execution to begin. Along with this signal, an address is also passed, indicating where in passive memory the program to be evaluated resides.

Starting

At this point it is assumed that each Lisp processor already has had its programs loaded into its private memory.

When the AHR machine has received the "start" signal, the distributor makes available a node (called the RUN node) to some Lisp processor. This node points to the program which will start to be evaluated.

The program (in passive-memory) is then copied (i.e., transformed from its passive-memory representation, which is in list notation, to its grill-representation, which is composed of nodes) by more Lisp processors into the grill. (The amount of leaves or branches a program has decides the number of processors that will be needed to help copy it). Nodes with nane=0 are inserted by the Lisp processors into the fifo, so that other Lisp processors will execute them.

NOTE: At a given time, there are some Lisp

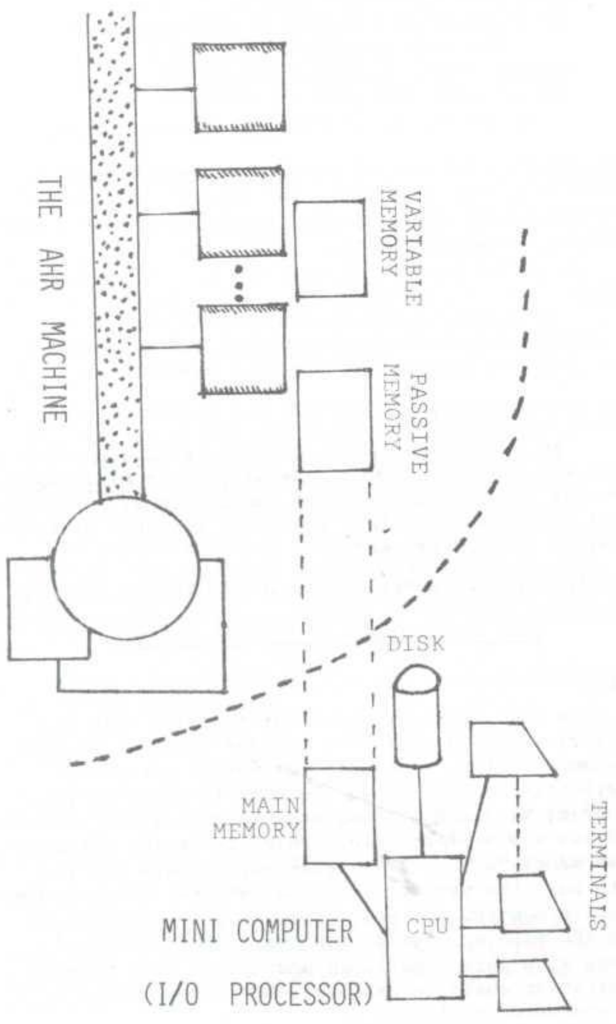


FIGURE 3

"THE AHR MACHINE AS A SLAVE"

The AHR computer is shown as another peripheral of a general purpose minicomputer. The address space of the mini comprises the passive memory of AHR, through a movable window of 4k addresses.

processors copying the program while nodes with nane=0 are being evaluated by other Lisp processors.

Evaluation

When a Lisp processor is idle, it gives a signal to the distributor indicating it is ready to accept more work.

The distributor is very fast in comparison to the speed of the Lisp processor. This is even more evident if "complicated" Lisp functions (such as MEMBER OF FACTORIAL) are coded in Z-80 machine language, instead of "simple" Lisp functions, such as CDR.

Due to a large difference in speed, the distributor can have and continuously keep working many Lisp processors. For example, if the distributor is 100 times faster than the (average) Lisp function, it could keep 100 Lisp processors functioning. It is therefore worthwhile to have a fast distributor.

The distributor selects (with the help of an arbiter) one of several idle processors, and through the high speed bus it introduces a new node (taken from the grill through the head of the fifo) into the private memory of the processor. It then signals that processor to start.

The Lisp processor finds the node in its memory with all the arguments already evaluated. The Lisp processor proceeds to perform the evaluation that is needed by the node. For example if it is LIST, and its arguments are (A B), M and N, it then has to address the passive memory in the "give a new cell" mode. Such cell is given by a cell dispatcher (hardware attached to passive memory). In this case three new cells have to be requested. The Lisp processor then forms the result: ((A B) M N). For this result the Lisp processor has to store pointers into passive memory [in the new cells that have been obtained] to (A B), to M and a pointer to N. It then stores the result (which is a pointer to passive memory) into a special place ("results place") of its private memory. It then signals to the distributor that it is finished and is ready to accept more work. The distributor will insert new work (another node with nane=0) into the private memory of the processor, but it will also collect the result ((A B) M N) (through the high speed bus; see figure 2) from the "results place" in the private memory of that processor. The distributor will store this result into a slot in a node in the grill. The address of this slot in the grill is known to the (LIST (A B) M N) node, because each node points to its father. Thus, the distributor has no problem in finding where to store the result: this address is found also in the "results place", together with the result ((A B) M N).

After all of the above is accomplished the distributor has to subtract one from the nane of the father (which has just received the result ((A B) M N). If the nane becomes zero, then a pointer to the father is introduced by the distributor into the fifo through its tail.

The last thing that the distributor does is to free the cell of the node (LIST (A B) M N), so that this grill space can be reused [10].

Output

Finally, after the complete program has been converted into a single result (let us say, a list) and deposited in passive memory, the AHR machine then signals the mini (I/O processor) also giving to it the address in passive memory where the result is being stored. The mini makes access to the passive memory as if the passive memory were a part of its own memory (since their address spaces overlap), and proceeds to the (serial) printing process. Execution of the program has finished.

IV. SOFTWARE ISSUES

Editing

Editing of Lisp programs is done outside the AHR machine, using the operating system and editor of the I/O processor. After editing, the program is filed on disk and a loader (running in the I/O processor) converts the Lisp program into list cells and brings the program to passive memory. (See figure 3).

The Lisp Interpreter

A lisp interpreter runs in each Lisp processor. It interprets pure Lisp (only evaluations; no setq's, rplacd's or other operators). The garbage collection is not done at this time by the Lisp processors.

In the first version of the machine, the Lisp interpreter will do argument checking of the Lisp functions. This will remain as an option in the second version of the AHR machine.

First Version of the Distributor

This is a piece of software [10] running in a Z-80 microprocessor, that emulates all the functions that the "real" (hardware) distributor performs. Here it functions slowly, but it is flexible and helps in the debugging of the AHR machine, (it can be ran "step by step" to see the flow of information). It also keeps statistics of the uses of the hardware and software.

The Garbage Collector

In the first version of the machine, the garbage collector will be a "normal" serial garbage collector, running in the I/O processor. While it is working, the Lisp processors remain idle. But in the second version, it will be a parallel incremental garbage collector, running in the Lisp processors.

Garbage collection is done for passive memory (list cells) and for the real number regions of variable memory (where it compactifies memory). There is no need to recollect garbage in the "environment" zones of variable memory and in the grill (nodes), because in these two places, as soon as used space is abandoned the used space is inserted (by hardware) into a list of free environment cells (for variable memory) or into a list of free nodes (for the grill).

V. HARDWARE ISSUES

The distributor

The distributor dispatches nodes from the grill to the Lisp processors, and stores in the grill the results that are coming from the Lisp processors. There are two versions of the distributor.

First version of the distributor. This first version [10] is implemented through a Z-80 microprocessor, using a program that performs all the functions of the distributor. It runs slowly, in the sense of distributing nodes at low speed.

Second version of the distributor. This version will be a faster distributor. It has not yet been built. It will become part of version 1 of the machine, being built either from bit-slice microprocessors or from PAL's.

The Arbiter. There are really three arbiters, one for passive memory, a second for variable memory, and the third for the grill.

Each arbiter takes 400 nanoseconds to respond, and it may handle up to 64 processors. Each processor has a different unique fixed priority, varying from 1 to 64. Since all of the processors are equal (they are able to perform exactly the same tasks), the assignment of priorities to processors really does not matter. Of course, if there are many processors available, those with lowest priorities will never obtain work (nodes) to do.

Lisp Processors

The first version of the machine will have 5 Lisp processors, and the I/O processor will be a normal mini, or perhaps another Z-80 micro. Each Lisp processor will have 16K bytes of private memory, where will reside a pure-Lisp interpreter [8].

The maximum number of Lisp processors is 64. The number of Lisp processors could be increased, but a new arbiter would need to be designed.

The High-Speed bus. Through this bus, the distributor inserts a node (7 words of 32 bits) into the private address space of the selected Lisp processor. This is accomplised in 0.5 microseconds. The high speed bus runs from the distributor to all Lisp processors, carrying nodes and results.

The I/O Processor

Although initially contemplated to be a minicomputer, it is actually built around a Z-80 micro processor that works as a general purpose computer. Its main functions are:

* to communicate with the users; reading their input and printing the results.
* to store user files in its disk.
* to initialize the AHR machine.
* to load into passive memory, through the window, the programs loaded from disk.
* to begin garbage collection.
* to end garbage collection.

(The garbage collector actually runs in the I/O processor).

VI. RELATED WORK AND MACHINES

Parallel Lisp Machines

The machine of [7] is a loosely coupled multi processor for applicative languages such as Lisp. In its application, this machine closely resembles ours. Another Lisp machine [15] uses three processors that collaborate with each other.

Greenblatt's Lisp Machine

This is a single processor machine [14] built for high speed Lisp computations. It does not claim to be an experiment in parallel hardware. This Lisp machine acquires its speed and power from careful design of the software and machine architecture, as well as from the experience of the builders with the Lisp language.

Data Flow Machines

These machines [13] resemble the AHR architecture in that data is directed through "boxes" that process them. The flow of executions is controlled, like in our design, by what previous results are ready (available). The cited article describes a machine that uses different color tokens to mark "this result", "previous result", and so on.

The Language "L" for Image Processing

"L" is a language suitable for processing of images. It is mentioned here because it may be implemented in a parallel machine [4], such as the AHR computer. This language is described elsewhere [1]. The language was designed mainly as a result of our experience in picture processing of multispectral images [6]. "L" has not been implemented.

Zmob.

This machine is a collection of Z-80 microprocessors around a conveyor belt [11]; it may be applied to image processing and numerical calculations. Each microprocessor has its own private memory. The microprocessors have direct access to a common memory (as AHR does), but behind one of the micros, a hugh central memory or mass memory may reside. It does not process Lisp.

PM4

This is a machine suitable for image processing [2]. It is a dynamically reconfigurable multimicroprocessor- based machine. It can be partitioned into several groups of processors which may be assigned to execute multiple independent SIMD processes and MIMD processes.

VII. CONCLUSIONS

Performance of the Machine

At this time no figures can be given, since the AHR machine isn't yet completed.

New Advances as of August 1981.

The hardware is now functioning and the software is completed. Extensive tests are under way, and several bugs have been found and fixed.

Final Remarks

The architecture of the AHR computer demonstrates that it is possible to build a multiprocessor of the MIMD type, where each processor does not explicitly communicate with other processors. In the AHR design, a processor does not know how many other processors exist, or what they are doing. It is not possible to address a processor: "here I have a message for processor number 4."

Finally, the AHR machine demonstrates how it is possible to design a heterarchical system, whereby all of the processors have the same priority level.

Once the machine has been built, experimentation will begin in the design of parallel languages and ways to express "powerful" commands in heterarchical fashion. It may be possible to place each micro in a remote place, consequently achieving some class of distributed computing, if the amount of access to memories for each processor is low. That is, a micro can process local work (through Basic, for instance) as well as remote (Lisp) work.

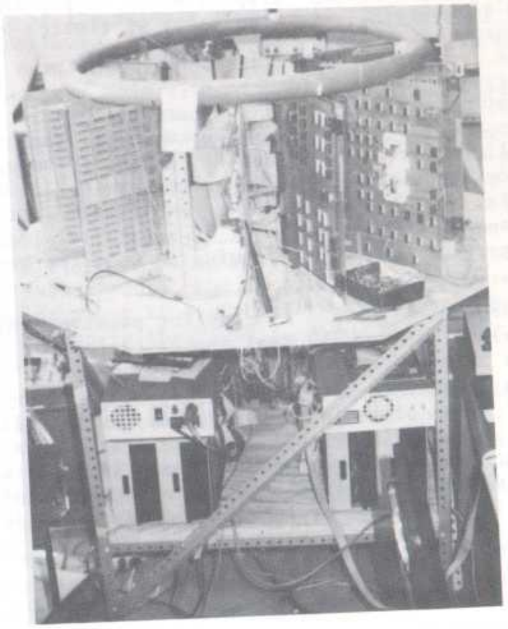
By connecting the machine to a general purpose computer, therefore being able to use already available operating systems for time sharing, text editors and loaders, the construction of new software has been kept low.

Acknowledgements

The AHR machine is being built by the members of the AHR Project, to whom I express my appreciation for their time, effort and enthusiasm.

I wish also to add thanks to Sintra Duke, one of our Computer Science students, who entirely rewrote an earlier manuscript [4], giving a better English version of it.

Work herein described has been partially supported by Grant #1632 from CONACYT, the National Council for Science and Technology (Mexico).



THE AHR COMPUTER

The picture shows the circular structure of the AHR machine. Under the table, two Zilog Z80 microcomputers: the I/O processor (left), and the distributor, Version 0 (right).

References

1. Barrera, R., Guzmán, A., Ginich, A., and Radhakrishnan, T. Design of a high level language for image processing. In Languages and Architectures for Image Processing, M.J.B. Duff and S. Levialdi (eds). 1981. Academic Press

2. Briggs, F.A., Fu, K.S., Hwang, K., and Patel, J.H. PM4: a reconfigurable multiprocessor system for pattern recognition and image processing, 1979. Technical report TH-EE-79-11. School of Electr. Eng., Purdue University (USA)

3. Guzmán, A., and Segovia, R. A parallel reconfigurable LISP machine. 1976. Proceedings of the International Conference on Information. Sci. and Systems. Univ. of Patras, Greece. 207-211.

4. Guzmán, A. Heterarchical architectures for parallel processing of digital images. In Languages and Architectures for Image Processing, M.J.B. Duff and S. Levialdi (eds). 1981. Academic Press.

5. Guzmán, A., Lyons, L., et al. The AHR Computer: construction of a multiprocessor with LISP as its main language. (in Spanish). 1980. Technical report AHR-80-10. IIMAS, National University of Mexico.

6. Guzmán, A., Seco, R., and Sánchez, V. Computer Analysis of LANDSAT images for crop identification in Mexico. 1976. Proceedings of the International Conference on Information Sciences and Systems. University of Patras, Greece. 361-366.

7. Keller, R.M., Lindstrom, G., and Patil, S. A loosely-coupled applicative multi-processing system. AFIPS 1979 Conference Proceedings, Vol. 48, 613-622.

8. Norkin, K., and Gómez, D. A new description for data transformations in the AHR computer. 1979. Technical report AHR-79-4, IIMAS, National University of Mexico.

9. Norkin, K., and Rosenblueth, D. Towards optimization in AHR. Technical report AHR-79-5, IIMAS, National Univ. of Mexico, 1979.

10. Peñarrieta, L. Error detection in the AHR computer. (In Spanish). 1980. Technical report AHR-80-9. IIMAS, National University of Mexico.

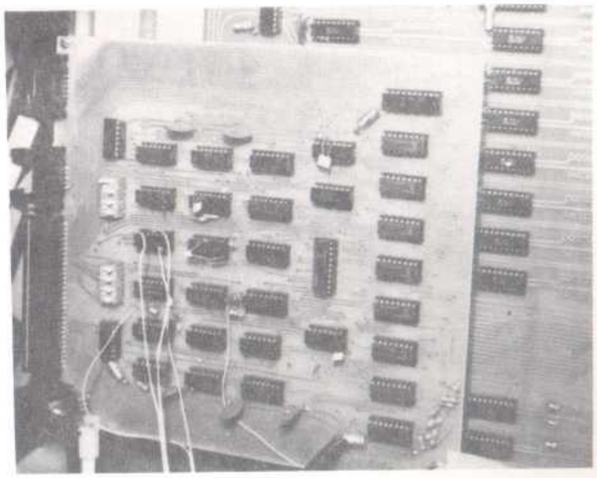
11. Rieger, C., Bane, J., and Trigg, R. ZMOB: a highly parallel multiprocessor. 1980. Technical report TR-911, Dept. of Comp. Science, Univ. of Maryland (USA).

12. Rosenblueth, D., and Velarde, C., The AHR machine for parallel processing: 1st. stage. (In Spanish). 1979. Technical Report AHR-79-2, IIMAS, National University of Mexico.

13. Watson, Ian, and Gurd, John. A prototype dataflow computer with token labeling. AFIPS 1979 Conference Proceedings, 48, 623-628.

14. Weinreb, C., and Moon, D. Lisp machine manual. 1979. M.I.T.A.I. Laboratory, Cambridge, Mass. (USA)

15. Williams, R. A Multiprocessing system for the direct execution of Lisp. 4th Workshop on Computer Architecture for non-numeric processing. 1978. ACM Sigarch Vol. VII No.2 (Sigmod Vol. X No.1). Pages 35-41.



Not to be confused with the fifo (Sec. II), the fifo-fifo (not discussed in the text) is a piece of hardware that the Lisp processors use to do input/and output with the I/O processor: it holds S-expressions to be read or printed.
The variable memory is behind the fifo-fifo.

COMMUNICATING THROUGH THE LOW-SPEED BUS

Sintra Duke uses a keyboard to issue commands to the Lisp processors through the lowspeed bus (Section II), normally connected to the I/O processor.