47 1981

Chapter Eighteen

A Parallel Heterarchical Machine for High-level Language Processing

Adolfo Guzmán

1. INTRODUCTION AND PROJECT STATUS

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

1.1 Goals

The goals of the Project AHR (Arquitecturas Heterárquicas Reconfigurables) are:

(a) to explore new ways to perform parallel processing;
(b) to have a machine in which it will be possible to develop parallel processing languages and software;
(c) to have a tool for students to learn and practice parallel concepts in hardware and software.

1.2 Project status

Version 0 [3] of the machine has been designed and simulated. This produced Version 1 [12], which was simulated using SIMULA.

We are building Version 1 of the machine, expected to be operational [5] in the first quarter of 1981. Subsequently, a faster version will be built, possibly incorporating changes and ideas sprung from our experience with the first machine. Finally, this fast version will be used to try to attain the goals mentioned above.

In "Languages and Architectures for Image Processing." Duff and Levialdi, eds. Academic Press. 1981

About six people are involved in the project full-time. The expected uses of the machine also include picture processing, finite element methods, engineering calculations, and distributed processing.

1.3 Main features

The AHR machine has the following characteristics:

(a) general purpose;
(b) parallel processor;
(c) heterarchical. It means that there is no hierarchy among the processors; there is no "master" processor, or controller. All the processors are at the same level;
(d) asynchronous operation;
(e) it has LISP as its main programming language;
(f) processors do not communicate directly amongst themselves. They only "leave work" for somebody else to do it;
(g) no input/output. This is handled by a minicomputer to which the AHR machine is attached;
(h) no operating system (software). Most of the LISP operations, as well as the garbage collector, are written in Z-80 machine language;
(i) the AHR machine works as a slave of a general purpose computer (a mini);
(j) gradually expandable. More microprocessors can be added as additional computing power is needed [9].

1.4 Funtional notation

The AHR machine obtains its parallelism by parallel evaluation of the arguments of functions. 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; then g of the result, in parallel with a and b. That is, evaluation occurs from bottom up, or from the inside to the outside of the expression. This is in accordance with the rule for evaluation of a function: "to evaluate a function, the arguments have to be already evaluated".

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

The machine works with pure LISP, without SETQ's, GOTO's, Label's, RPLACA.

2. THE PARTS OF THE AHR MACHINE

In this section the constituents of the machine are described; section 3 explains how the machine works. Refer to Fig. 2—"The AHR machine".

2.1 Passive memory

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

Originally, the programs to be executed reside here, and they are copied to the grill (see below) for their execution. As new data structures are built as partial results of the evaluation, they come to the passive memory to reside.

2.2 The grill

This memory holds the programs that are being executed. A program, once in the grill, is being transformed into *results*, as the result of its evaluation.

Programs reside in the grill in the form of *nodes* (see Fig. 1). Each node is pointed at by its sons (its arguments), and its *nane* field contains the number of non-evaluated arguments. Nodes with nane = 0 are ready for evaluation.

2.3 The LISP processors

These active units are microprocessors (about several dozens of Z-80's) that obtain from the grill nodes ready for evaluation, and, after evaluation, return *results* (s-expressions) to the grill. Each LISP processor knows how to execute every LISP primitive. Each of them works asynchronously, without communicating with other processors.

The processors obtain 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 instance (CAR '(A B C)') will evaluate to A. The node (CAR '(A B C)') has become the result A. The LISP processors has to do, after evaluation, the following things:

(1) Insert the new result A in the cell (in the grill) pointed to by the node (CAR '(A B C)'). That is, insert such result in a slot of the father of the evaluated node (see such slots in Fig. 1).
(2) Release the grill space occupied by node (CAR '(A B C)').
(3) Subtract 1 from the nane of the father.

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

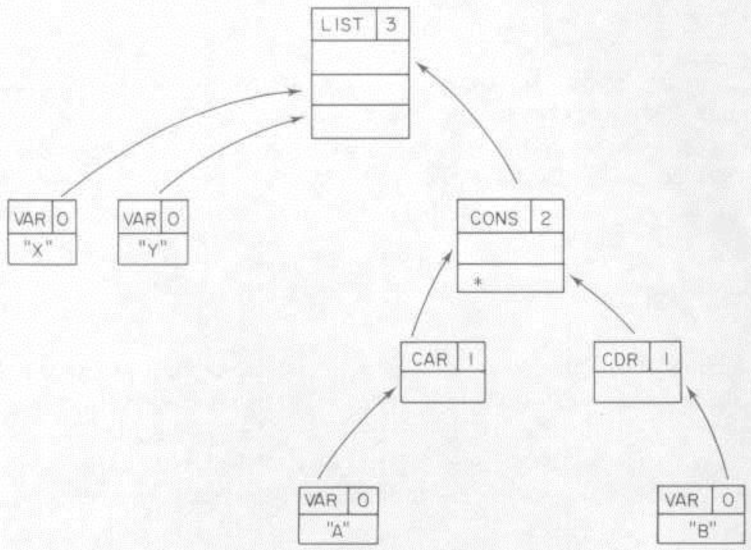

Fig. 1. Nodes in the grill. (above) the LISP expression to be evaluated. (below) how it is structured into nodes, each node being a function or a variable. Each node shows a number: its nane, *or number of non-evaluated arguments. When a node has a nane of zero, it means that node is ready for evaluation.*

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 "".*

(4) If the new nane (of the father) is zero, inscribe the father in the FIFO: the father is now ready for evaluation.

These steps are done by the processor simply by signalling to the distributor that it has finished, and that its results should be handled in mode "normal end" (burocracia de salida, in Spanish [12]); the distributor itself performs the requested steps.

Notice that in this form nobody has to search the grill looking for nodes with nane = 0, because as soon as they appear, they are inserted into the tail of the FIFO.

The LISP processors have access to the passive memory (where lists and atoms reside), and to the variable memory, where we have the vlaues of variables. A LISP processor is either *busy* (evaluating a node) or it is *ready* to accept more work (another node).

2.3.1 The high speed bus

Connecting each LISP processor with the distributor is a high speed bus that goes into the private memory of each processor. The new node that the distributor throws is inserted (through the high speed bus) into the memory of the selected processor. Then, the processor is signalled to proceed.

2.3.2 The slow speed bus

This bus runs from the I/O processor (the mini to which the AHR machine is connected) to each box. It is not shown in the diagrams, nor it is explained furthermore in this article [see 5]. Through this bus each processor is loaded with programs, prior to starting the machine. Also, in the debugging stage, the slow bus is used to pass statistical information to the I/O processor.

2.4 Variable memory

This memory contains pairs of (variable, values), and it is organized as a tree, or a collection of a-lists, where each pair of (variable, values) 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.

2.5 The distributor

This piece of hardware communicates the grill with the LISP processors. The distributor keeps in the *FIFO* (a memory) and array of nodes ready to be evaluated; these nodes are thrown, one in each cycle of the distributor, to the LISP processors that are ready to accept new work. An *arbiter* decides which LISP processor obtains the node; 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. The distributor stores such results in the grill, in the address indicated within the result. Generally, this result is stored in a slot of the node which is father of the node just evaluated. An overall view of the machine is shown in Fig. 2.

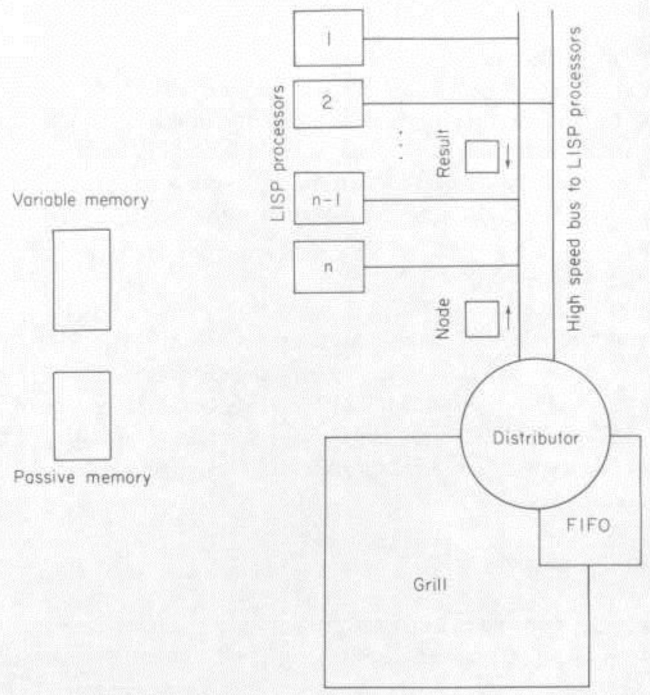

Fig. 2. The AHR machine. LISP processor 2 is ready to accept more work. The distributor fetches a node (to be evaluated) from the FIFO and sends it to processor 2, while accepting the results of the previous evaluation performed by such a processor. That result is stored in the grill, in a place indicated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

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

2.5.1 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].

2.5.2. The arbiter

If several LISP processors become ready to accept more work, the arbiter (a hardware) selects one of them, which will receive the node thrown by the distributor. If every processor is busy, the cycle of the distributor is wasted, since no processor accepts the node that the distributor is offering.

2.6 The I/O processor

It has been said that 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; we thus talk of such a mini as the I/O processor.

Input/ouput will be described in the next section.

3. HOW THE MACHINE WORKS

3.1 Input

The user sits at a terminal of the mini (I/O processor) which is master of the AHR machine. He uses a common editor, discs and the normal operating system of the mini. When he is ready to run a program, he loads it from disc into a part of the address space of the mini which is really the passive memory of the AHR machine (see Fig. 3). In this way, the program is loaded (already as list cells) in the passive memory. A signal from the I/O processor to the AHR machine signifies that execution should begin. Together with this signal an address is passed, indicating where in passive memory the program to be evaluated resides.

3.2 Starting

It is assumed that each LISP processor already has its programs loaded in its private memory.

When the AHR machine receives the "start" signal, the distributor throws a node (called the RUN node) to the LISP processor. This node points to the program which will start.

The program (in passive memory) is copied by more and more LISP processors (the more leaves or branches a program has, the more processors help to copy it) into the grill. Nodes with nane = 0 are inserted by the LISP

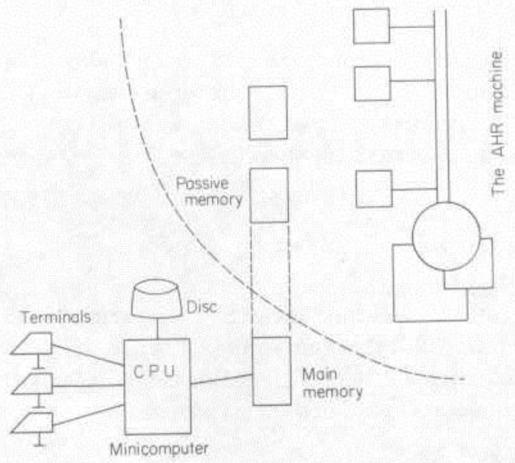

Fig. 3. The AHR machine as a slave. The AHR computer is shown as another peripheral of a general purpose minicomputer. The address space of the mini comprises the passive memory of AHR, through a movable window of 4 K addresses.

processors into the FIFO, so that some other LISP processor with execute them. Finally, the program has been copied into the grill. Notice that at the same time of copying, some nodes with nane = 0 could have been evaluated by some other LISP processors.

3.3 Evaluation

When a LISP processor is idle, it signals to the distributor, meaning that it is ready to accept more work.

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

The LISP processor "discovers" the node in its own memory, with all the arguments already evaluated. The LISP processor proceeds to perform the evaluation that the node demands. Suppose it is LIST, and its arguments are (A B), M and N. It then has to address the passive memory in the mode "give a new cell". Such a cell is given by a cell dispatcher (hardware attached to passive memory). Three new cells have to be requested. Then the LISP processor forms the result: ((A B) M N). For this, it has to store pointers to (A B), to M and to N, into passive memory, in the new cells already obtained. Then, it stores the result (which is a pointer to passive memory) into a special place ("results place") of its private memory. It has finished. It signals to the distributor that it 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 (through the high speed bus; see Fig. 2) from the "results place" in private memory, the result ((A B) M N). The distributor will store this result into a slot in a node in the grill. The address in the grill of this slot was 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: such address is found also in the "results place", together with the result ((A B) M N).

The distributor has to do one more thing: it has to substract one from the nane of the father (which has just received the result ((A B) M N)). And if such nane becomes zero, then a pointer to the father is insterted by the distributor into the FIFO through its tail. Finally, the distributor has to free the cell of the node (LIST (A B) M N), so that this grill space could be re-used [10].

The distributor is very fast compared with the speed of the LISP processor. This will be even more true if we code "complicated" LISP functions (such as MEMBER OR FACTORIAL) in Z-80 machine language, instead of "simple" LISP functions, such as CDR.

Due to such differences in speed, the distributor can keep many LISP processors working; if the distributor is 100 times faster than the (average) LISP function, it could keep 100 LISP processors functioning. It pays to make a fast distributor.

3.4 Output

Finally, the whole program has been converted into a single result (let us say, a list) deposited in passive memory. The AHR machine now signals the mini (or I/O processor), giving it also the address in passive memory where the result lays. The mini now accesses the passive memory as if it were part of its own memory (remember, their address spaces overlap), and proceeds to the (serial) printing process. Execution has now finished.

4. HARDWARE CONSIDERATIONS

4.1 LISP processors

The first version of the machine will have 5 LISP processors, and the "mini" or I/O processor is another Z-80. Each LISP processor will have 4 K bytes of private memory, where a pure-LISP interpreter will reside [8].

The maximum number of LISP processors is 64. It could be increased further, but a new arbiter needs to be designed in that case.

4.1.1 The high speed bus

The distributor inserts a node (7 words of 32 bits) into the private address space of the selected LISP processor, through the high speed bus. It does this in 0.5 μs. It runs from the distributor to all LISP processors. It carries nodes and results.

4.1.2 The low speed bus

In a 16 bits low speed bus, 8 of the bits indicate which LISP processor is addressed, the other 8 bits carry data. It runs from the I/O processor to the LISP processors.

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

4.2 Passive memory

It consists of up to 1024 K words of 22 bits; it contains the input ports, list space, output ports and atom space. Version 1 will have only 64 K words. Access time is 150 ns. It has a parity bit.

4.3 The grill

It consists of up to 512 K words of 32 bits. It is divided logically in nodes, each with 7 words. Version 1 will have 8 K words. Access time is 55 ns. The grill contains the nodes that are about to be evaluated.

4.4 Variable memory

It consists of up to 512 K words of 32 bits. This memory contains names of variables and their values at a given time. 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 16 K words. Access time is 150 ns.

4.5 The distributor

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

4.5.1 First version of the distributor

This first version [10] is implemented through a Z-80, using a program that performs all the functions of the distributor. It runs slowly, in the sense that it distributes nodes at low speed. It is further described in section 5.

4.5.2 Second version: fast distributor

This version is not yet built; it will become part of version 1 of the machine. It will be built either from bit-slice microprocessors, or from PAL's.

4.5.3 The FIFO

Of a maximum of 512 K words of 19 bits, the FIFO contains pointers to the nodes in the grill. Version 1 will be of 4 K words. Its access time is 55 ns.

4.5.4 The arbiter

There are really three arbiters—for passive memory, variable memory and for the grill. Each arbiter takes 400 ns to respond, and it may handle up to 64 processors. Each processor has a fixed priority, varying from 1 to 64. Each processor has a different (unique) priority.

4.6 The I/O processor

This is actually built around a Z-80 that works as a general purpose computer. Its main functions are to:

(a) talk to the users; to read their input and to print their results;
(b) store user files in its disc;
(c) initialize the AHR machine;
(d) load into passive memory, through the window, the programs loaded from disc;
(e) begin garbage collection;
(f) end garbage collection;
(g) Actually, the garbage collector runs in the I/O processor.

5. SOFTWARE CONSIDERATIONS

5.1 The LISP interpreter

A LISP interpreter runs in each LISP processor. It interprets pure LISP (only evaluations; no SETQs, RPLACDs or other operators). The garbage collection is not done by the LISP processors at this moment.

For the first version, 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.

5.2 The garbage collector

For the first version of the machine, this will be a "normal" serial garbage collector, running in the I/O processor. While it is working, the LISP processors remain idle. For 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 numbers region of variable memory (where it compactifies memory). In the "environments" zone of variable memory and in the grill (nodes), there is no need to re-collect garbage, because used space, as soon as it is abandoned in these two places, is inserted (by hardware) into a list of free environment cells (for variable memory) or into a list of free nodes (for the grill).

5.3 The distributor (first version)

This is a piece of software [10] running in a Z-80, that emulates all the functions that the "real" (hardware) distributor performs. It is slow in this sense, but it is flexible and helps in the debugging of the AHR machine; it may be run "step-by-step" to see the flow of information. It also keeps statistics of use of hardware and software.

5.4 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 disc. From here, a loader (running in the I/O processor) converts it into list cells and brings the program to passive memory (see Fig. 3).

6. RELATED WORK AND MACHINES

6.1 Greenblatt's LISP machine

This is a single processor machine [14] built for high speed LISP computations. It does not pretend to be an experiment in parallel hardware; it gains 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.

6.2 Parallel LISP machine

This machine [7] is a loosely coupled multiprocessor for applicative languages such as LISP. It is the machine most closely resembling ours, in its application.

6.3 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 colours of tokens to mark "this result", "previous result", and so on.

6.4 ZMOB

A collection of Z-80's around a conveyor belt, this machine [11] may be applied to image processing and numerical calculations. Each microprocessor has its own private memory. They do not have direct access to a common memory (as AHR does), but behind one of the micros, a huge central memory or mass memory may reside.

6.5 PM4

This is a machine [2] suitable for image processing. 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.

6.6 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. The language is described elsewhere [1] in this book. It was designed mainly as a result of our experience in picture processing of multispectral images [6]. "L" has not been implemented.

7. CONCLUSIONS

The architecture of the AHR computer shows 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 are there, or what they are doing. It is not possible to address a processor: "here I have a message for processor number 4."

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

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

Finally, the AHR machine shows how it is possible to design a heterarchical system, where none of the processors tells the others what to do, in what order to do it, or what resources are available to whom.

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.

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

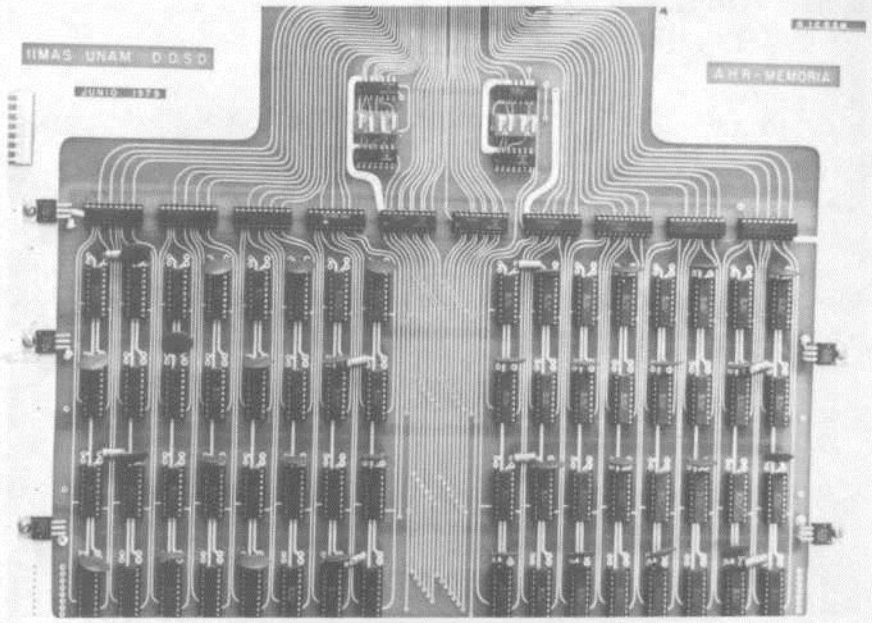

Fig. 4. The FIFO. This figure shows the FIFO memory that stores pointers to the grill, containing nodes ready to be evaluated.

REFERENCES

[1] Barrera, R., Guzmán, A., Ginich, A. and Radhakrishan, T. (1981). "Design of a high level language for image processing". In *Languages and architectures for image processing*. M. J. B. Duff and S. Levialdi (eds). Academic Press, London and New York.

[2] Briggs, F. A., Fu, K. S., Hwang, K. and Patel, J. H. (1979). "PM4: A reconfigurable multiprocessor system for pattern recognition and image processing". Tech. Rep. TR-EE-79-11, School of Electr. Eng., Purdue University.

[3] Guzmán, A. and Segovia, R. (1976). "A parallel reconfigurable LISP machine". Proc. Int. Conf. on Inf. Sci. and Syst., University of Patras, Greece, pp. 207–211.

[4] Guzmán, A. (1979). "Heterarchical architectures for parallel processing of digital images". Tech. Rep. AHR-79-3, IIMAS, Nat. Univ. of Mexico.

[5] Guzmán, A., Lyons, L. *et al.* (1980). "The AHR computer: construction of a multiprocessor with LISP as its main language" (in Spanish). Tech. Rep. AHR-80-10, IIMAS, Nat. Univ. of Mexico.

[6] Guzmán, A., Seco, R. and Sanchez, V. (1976). "Computer analysis of LANDSAT images for crop identification in Mexico". Proc. Int. Conf. on Inf. Sci. and Syst., University of Patras, Greece, pp. 361–366.

[7] Keller, R. M., Lindstrom. G. and Patil, S. (1979). "A loosely-coupled applicative multi-processing system". AFIPS Conf. Proc. **48,** 613–622.

[8] Norkin, K. and Gomez, D. (1979). "A new description for data transformations in the AHR computer". Tech. Rep. AHR-79-4, IIMAS, Nat. Univ. of Mexico.

[9] Norkin, K. and Rosenblueth, D. (1979). "Towards optimization in AHR". Tech. Rep. AHR-79-5, IIMAS, Nat. Univ. of Mexico.

[10] Penarrieta, L. (1980). "Error detection in the AHR computer" (in Spanish). Tech. Rep. AHR-80-9, IIMAS, Nat. Univ. of Mexico.

[11] Rieger, C., Bane, J. and Trigg, R. (1980). "ZMOB: a highly parallel multiprocessor". Tech. Rep. TR-911, Dept. of Comp. Sci., University of Maryland.

[12] Rosenbleuth, D. and Velarde, C. (1979). "The AHR machine for parallel processing: 1st stage" (in Spanish). Tech. Rep. AHR-79-2, IIMAS, Nat. Univ. of Mexico.

[13] Watson, I. and Gurd, J. (1979). "A prototype dataflow computer with token labeling". AFIPS Conf. Proc. **48,** 623–628.

[14] Weinreb, D. and Moon, D. (1979). *LISP machine manual*. M.I.T.A.I. Lab., Cambridge, Massachusetts.