comunicaciones técnicas

THE DESIGN AND CONSTRUCTION OF A PARALLEL HETERARCHICAL MACHINE: FINAL REPORT OF PHASE 1 OF THE AHR PROJECT*

Adolfo Guzmán
Kemer B. Norkin**

serie naranja: investigaciones

INSTITUTO DE INVESTIGACIONES
EN MATEMATICAS APLICADAS
Y EN SISTEMAS

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

Apdo. Postal 20-726 Admón. No. 20
Delegación de Alvaro Obregón
01000 México, D.F.
548-54-65

IIMAS
UNAM

Artículo 53

comunicaciones técnicas

1982 Serie Naranja: Investigaciones No. 308

THE DESIGN AND CONSTRUCTION OF A PARALLEL HETERARCHICAL MACHINE: FINAL REPORT OF PHASE 1 OF THE AHR PROJECT*

Adolfo Guzmán
Kemer B. Norkin**

*Technical report AHR-82-21

**Professor of Computer Science and Head of the Optimization Laboratory, Institute for Control Sciences, USSR Academy of Sciences, Moscow.

Recibida: 2 de junio de 1982.

INSTITUTO DE INVESTIGACIONES
EN MATEMATICAS APLICADAS
Y EN SISTEMAS
UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

APDO. POSTAL 20-726 ADMON. No. 20
DELEGACION DE ALVARO OBREGON
01000 MEXICO, D.F.
550-52-15 ext. 4559

IIMAS
UNAM

TABLE OF CONTENTS

ABSTRACT

The experience obtained, in the framework of the AHR project, in building a prototype of a parallel LISP computer is generalized.

The machine built may have up to several dozens of microprocessors (Z-80's), working together to evaluate at the same time (in parallel) different pieces of the same Lisp program. This computer uses a new type of data exchange betweeen processors working in parallel. All the parts of this computer, except components, were built in Mexico. Current version possesses five Z-80's

Critical analysis of achievements is given.

Main directions for future scientific research and applied work are discussed. These include.

1. Estimation of the field of possible applications of the results (distributed computer systems, large computers, real-time applications)
2. Data flow optimization
3. Generalization for systems other than Lisp.
4. Determination of best policies for industrial production of the machine.

Аннотация

Обобщен опыт построения прототипа параллельной ЛИСП - машины, полученный в ходе осуществления проекта АНР.

Машина состоит из нескольких десятков микропроцессоров (Z - 80), включенных параллельно и одновременно оценивающих различные участки заданной ЛИСП-программы. В машине использован новый тип организации управления параллельной работой микропроцессоров. Машина создана мексиканскими специалистами из импортных деталей.

В отчете дан критический анализ полученных результатов. Обсуждаются основные направления будущих исследований и прикладных работ. Сюда относятся:

I. Выбор области применения результатов (распределенные вычислительные системы, крупные мультимикропроцессорные системы, системы реального времени)

2. Оптимизация потока данных

3. Обобщение на системы программирования иные, чем ЛИСП.

4. Определение наивыгоднейшей технологической политики при организации серийного производства машины.

ACKNOWLEDGEMENTS

To all those people who stayed with the AHR Project until its completion, as well as those who fell by the wayside, we are grateful for their participation.

A special word of gratitude to the IIMAS administration and its director, Tomás Garza, who made their best effort to help the Project.

This project benefitted from the interchange of visiting scientists between the Institute for Applied Mathematics and Systems (Mexico) and the Institute for Control Science (USSR Academy of Sciences, Moscow), as a consequence of an Agreement for Scientific Collaboratorion between both countries. We are grateful to the people in CONACYT (The National Council for Science and Technology, Mexico) and the USSR Academy of Sciences who made this agreement possible.

The essential support of the Directors of the Institute for Control Sciences is gratefully appreciated.

Michael Gerszo greatly improved the English of this report.

The finantial support from CONACYT (Grant # 1632, PCAINAL 790250) is acknowledged.

Авторы с признательностью отмечают финансовую поддержку КОНАСИТА (Национального совета по науке и технике) в рамках заказа № I632.

Проект AHR существенно выиграл от сотрудничества между учеными Института прикладной математики и систем (Национальный автономный университет Мексики) и Института Проблем управления (АН СССР и Минприбор, Москва). Работники Конасита и АН СССР много сделали для осуществления данного соглашения и их вклад в данную работу должен быть отмечен.

Мы особо благодарны дирекции IIMAS , сделавшей все, что было в ее силах для осуществления проекта, и руководству Института проблем управления за существенную поддержку.

OBJECTIVES OF THE AHR PROJECT

The main direction for computer development seems to be parallel processing. The future of this field is very bright and at the same time this field is very difficult because of the complexity inherent in parallel systems. In 1973, IIMAS researchers proposed organizations for parallel computers [Naranja 133]*. At that time, these proposals showed little promise to be fruitful.

However, in 1980-81, doubts concerning parallel processing in the proposed way disappeared as a consequence of the results obtained from the AHR project: a computer designed along the lines proposed in 1973 was built and tested in spite of all the difficulties intrinsic to parallel systems, and partly due to specific conditions existing in Mexico.

The laboratory prototype of the AHR computer consists of approximately 600 chips (see table "AHR CHIPS"). These chips include many of large scale integration (LSI), which means that the AHR project represents, under contemporary classifications, a very complicated electronic design. This project, which demanded such a huge amount of hardware work devoted to a unique goal, is new for IIMAS and, so far as we know, pioneer for Mexico. Because of this, the main goal of executing AHR was educative. Without such "on the job education" we could never discharge Mexico from big investments in computer imports. But this educational goal was not unique. AHR was a multi-purpose project.

Other goals were:

* 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.

The AHR computer has the following characteristics:

* general purpose parallel processor.
* All the processors are heterarchical. This means that there is no "master" processor, or controller. All the processors are at the same hierarchy level; hence, there is no hierarchy.
* Asynchronous operation.
* Lisp as its main programming language. (Pure Lisp, without goto's, setq's,...)
* 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 added.
* No input/output. This is conducted by a microcomputer to which the AHR machine

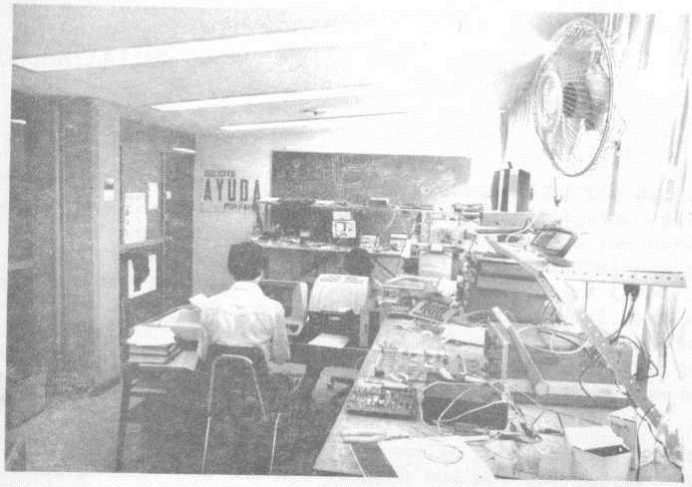
---

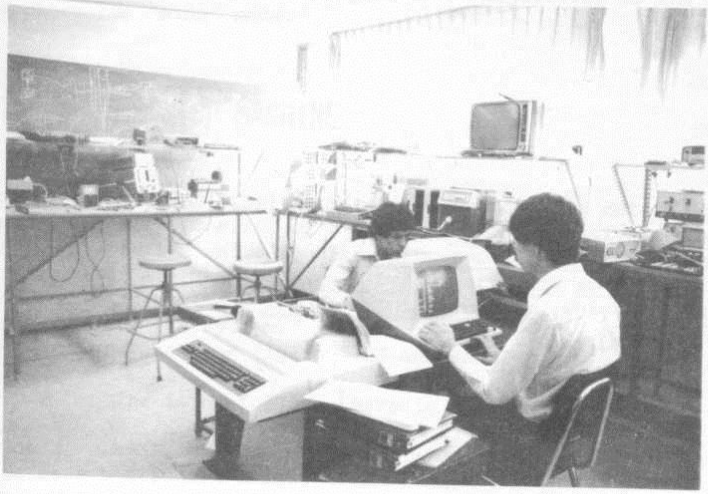
* References in square brackets are listed at the end of this report, but the AHR reports and publications are listed in page 11.

is attached.

* No operating system in software. There exists a normal operating system in the host processor, or i/o processor, but it is not considered to be part of the AHR machine. Also, if by "operating system" we mean "system's resources administrator", then the grill or active memory, together with the fifo and the distributor, do form an operating system. Nevertheless, it is embodied in hardware! 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-cell lists, and queues. The efficiency of the system is further increased by use of "intelligent" memories, which can be accesed in in several modes: "read", "write", "free this cell", "give a new cell", etc. These modes are implemented in hardware.
* The AHR machine works as a slave to the general purpose microcomputer.

Since the purposes of the AHR Project were mainly educative ans scientific, not much attention was paid to the normal things that give good service to users (input/output facilities, utility programs, service routines, and so on) but which are very well known, on the other hand. We concentrated instead in the new aspects of the computer, in its unique and novel parts, and in insuring that these new ideas will perform correctly.





A VIEW OF THE AHR LABORATORY

The picture was taken at the middle of 1980. It shows the general aspect of the AHR Laboratory.

PROJECT ORGANIZATION AND MAIN WORK PERFORMED

Work organization of the AHR project was determined by the difficult task at hand: to build and test the prototype of a complete computer system, which appeared to be very complicated, due to its novelty and to the parallelism of its execution. In the work organization, we faced a fundamental choice between "wide" or "deep" working modes. The "wide" mode is understood as involving practical work and experiments devoted to all parts of computer software and hardware. In this case, we would have to build all parts of the AHR machine using modern and fast components despite of its cost, complexity, difficulties in system building and debugging. These 'drawbacks' of "wide" mode are compensated by the evident educational advantages and the opportunities of increased learning.

In the "deep" mode, we would have to minimize risk and difficulties in designing reliable but perhaps not the most modern and not the faster prototype. However it could be used for a long time.

Taking into account our educational goal (See "Objectives of the AHR Project" above) we chose the "wide" mode.

SHORT DESCRIPTION OF THE LISP MACHINE

A short description of the AHR (or Lisp) machine is given in some detail in [Naranja 279, Naranja 280] . It consists of a memories grill, passive memory, variables memory and fifo; active units are Lisp processors or cajas ("boxes"); data circulation connections are the high speed bus and the low speed bus; and the distributor (with its arbiter) and Input-Output processor which can be regarded as administrators of the system.

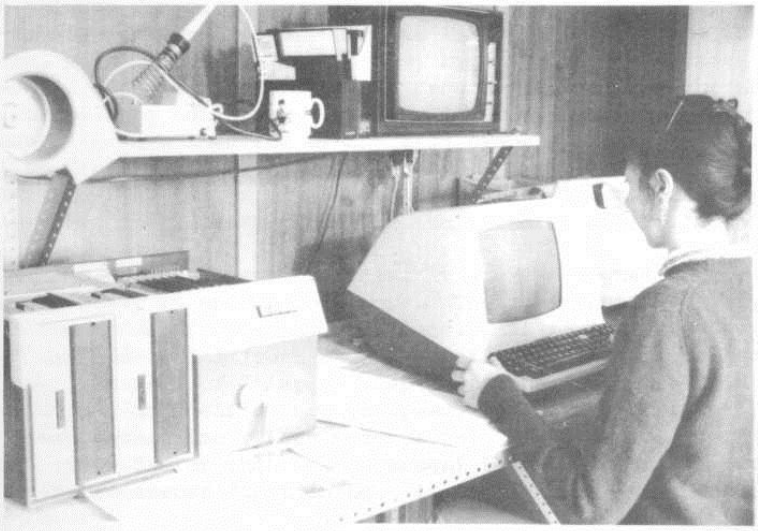
All these parts are shown on figure "THE AHR MACHINE."

Functioning of the AHR machine is as follows [Naranja 280] :

Input: The user uses a terminal of the micro (I/O processor) which is master of the AHR machine. He uses a standard editor, disks and the normal operating system of the micro. When the user is ready to run a program, he loads it from disk into a part of the address space of the micro (which is really the passive memory of the AHR machine. In this way, the program is loaded as list cells into 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.

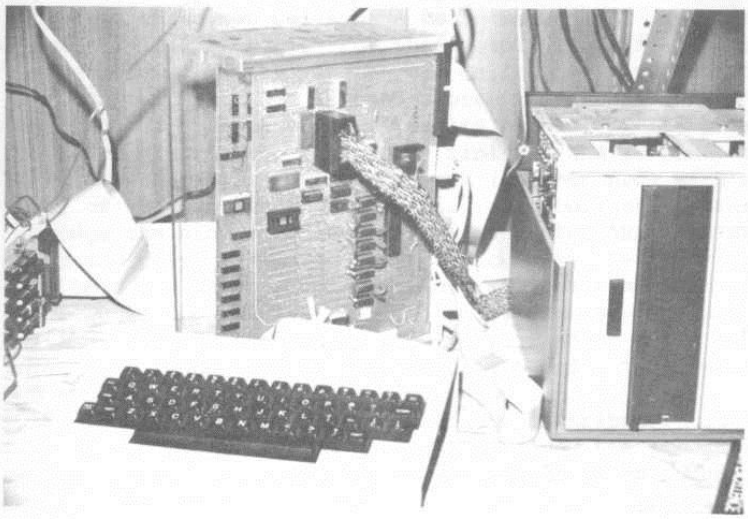
Starting: At this point it is assumed that each Lisp processor already has had its programs (the Lisp interpreter, written in PLZ and compiled into Z-80 machine language, with routines that handle the special hardware, the mail box, the memory access modes, and so on) loades into its private memory.

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.



BUILDING THE SOFTWARE FOR THE LISP PROCESSORS

Using a Zilog microcomputer as a developing aid, the Lisp interpreter for the Lisp processors is being developed.



ONE OF THE LISP PROCESSORS BY ITSELF

The Lisp processor (background, center) is being debugged through an "umbillical cord" by the Zilog microprocessor (right). In front, the keyboard simulates commands coming from the slow bus.

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; copying is done in parallel). 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 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 or FACTORIAL) are coded in Z-80 machine language, instead of "simple" Lisp functions, such as CDR.

Due to the large difference in speed, the distributor can have and continuously keep working many Lisp processors. For example, if the distributor is one hundred times faster than the (average) Lisp function, it could keep one hundred Lisp processors functioning. Is it 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 (in a "mail-box" fashion). 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 "THE AHR MACHINE") 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 that 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.

Output: Finally, after the complete program has been converted into a single result (a list, let us say) and deposited in passive memory, the AHR machine then signals the micro (I/O processor) also giving to it the address in passive memory where the result --the final result-- is being stored. The micro makes

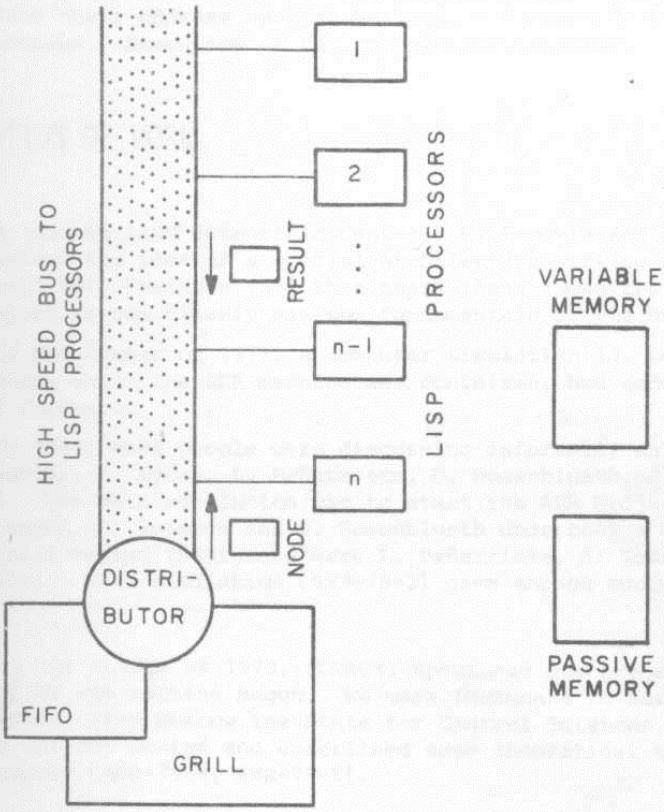


Figure 1

THE AHR MACHINE

Lisp processor 2 is ready to accept more work. The distributor fetckes 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. Version 2 may be built during Phase 2 of the Project.

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

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.

ORGANIZATION OF WORK

In 1973, A. Guzmán (IBM Scientific Center), R. Segovia and M. Magidin (CIMAS-UNAM) conceived the idea of a special architecture working with pure Lisp. It was not until 1976 [Naranja 133] that these ideas found their way in print. In that report we can clearly see the fundamentals of the modern AHR machine.

In the summer of 1977, a computer simulation (J. Ludlow) proved that the idea under which the AHR machine was conceived, had enough merits to be considered further.

In 1978, more people were discussing informally an initial design of the AHR machine: L. Lyons, L. Peñarrieta, D. Rosenblueth, J. M. López Acevedo, C. Velarde. The main conclusion was to start the AHR Project in January 1979. For this reason, C. Velarde and D. Rosenblueth undertook a detailed simulation of the initial design (designers were L. Peñarrieta, A. Guzmán and L. Lyons, among others). This simulation [AHR-79-2] gave enough encouragement to continue further.

In the middle of 1979, CONACYT sponsored our project, and serious construction of the machine begun. We were fortunate to have been visited by Prof. K. Norkin, from Moscow Institute for Control Sciences, who gave further ideas about current design and underlined some theoretical tradeoffs between speed and memory [AHR-79-4, AHR-79-5].

The design took most of 1979; we used the methodology of "democratic dictatorship": main issues of design were discussed between all member of the project (D. Gómez, A. Kuri, M. Correa and R. Gómez joined then the staff of the project); if an objective decision could be obtained by reasoning, it was adopted. In case of doubt or if the tradeoffs or advantages of different alternatives were not clear, the leader of AHR (A. Guzmán) will take the decision. All decisions were binding for all members, whether they agreed with them or not.

As a result of the design, a master file was formed. The results of design can be found in [AHR-79-2, AHR-79-5], as well as some results of the simulation.

The detailed design was divided then as follows:

* C. Velarde. Left the project at this time.
* L. Lyons. Responsible for overall hardware design.
  Responsible for design of Lisp processors ("cajas").
  Responsible for design of "acoplador" (coupler).
* L. Peñarrieta. Responsible for distributor, hardware and software versions [AHR-80-10, AHR-82-20]
* A. Kuri. Responsible for software distributor. Left the project after initial design of software distributor.

* M. Correa. Design of window from input/putput processor

* D. Rosenblueth and N. Apodaca. Garbage collector, software for input/output processor, screen editor [AHR-81-18]
* R. Gómez. Design of controllers for intelligent memories.
* M. Correa. Design of Memories.
* Dora Gómez. Design of the Lisp interpreter and high level routines (in Z-80 assembly language and in PLZ).
* Adolfo Guzmán. Leader of the project. Design of the low-level routines (software) of the Lisp interpreter [AHR-80-13, AHR-81-21]
* N. Apodaca. Arbiters [AHR-82-22]

Work proceeded during all 1980. More detail can be found in [AHR-80-10]

During January 1981, we had our AHR machine constructed, in the sense that all subsystems were put in the same cabinet.

By the end of 1981, the AHR machine was working [AHR-81-16], and the project was drawing to a close.

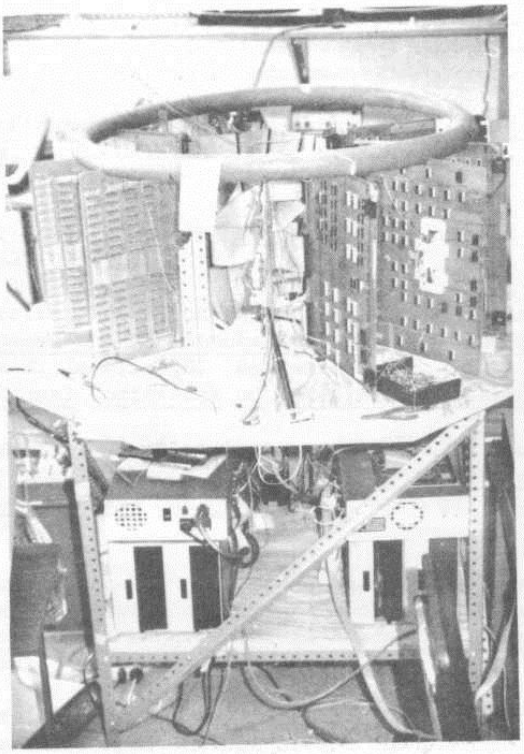
The above work was documented in about twenty publications and technical reports.

REPORTS AND PUBLICATIONS

A) TECHNICAL REPORTS FROM IIMAS

Series and Number	AUTHOR	TITLE	YEAR
Naranja 133	Adolfo Guzmán, Raymundo Segovia	A Configurable Lisp Machine. AHR-76-1	1976
Naranja 200	David Rosenblueth Carlos Velarde	The AHR machine for parallel processing. First Stage. AHR-79-2 (In Spanish)	1979
Naranja 206	Adolfo Guzmán	Heterarchical architectures for parallel processing of digital images. AHR-79-3	1979
Naranja 214	Kemer Norkin, Dora Gómez	A new description for data transformation in the AHR computer. AHR-79-4	1979
Naranja 215	Kemer Norkin, David Rosenblueth	Towards Optimization in AHR. AHR-79-5	1980
Naranja 216	Adolfo Guzmán	Distributed Computing as an alternative of the future. (In Spanish) AHR-79-6	1979
Amarilla 15	Adolfo Guzmán	125 Projects and Theses Topics in Computer Science (In Spanish). AHR-80-7	1980
Naranja 229	Adolfo Guzmán	Reconfigurable Geographic Data Bases. AHR-80-8	1980

Naranja 271	Luis Hugo Peñarrieta	DAHR: A debugging tool for AHR programming. (In Spanish) AHR-80-9	1981
Naranja 253	A. Guzmán et al.	The AHR Computer: construction of a multiprocessor with Lisp as its main language. (In Spanish). AHR-80-10	1980
Verde 17	A. Guzmán, D. Rosenblueth	Structures of Digital Computers. AHR-80-11	1980
Naranja 251	Víctor G. Sánchez	Reconfigurable information systems. AHR-80-12	1980
Naranja 246	Adolfo Guzmán	A parallel heterarchical machine for high level language processing. AHR-80-13	1980
Naranja 282	Luis Hugo Peñarrieta, Luis Lyons	A computer architecture for LANDSAT image management. (In Spanish) AHR-81-14	1981
(pending)	Dora Luz Gómez	The Lisp interpreter of the AHR Computer. (In Spanish) AHR-81-15	1981
Naranja 279	A. Guzmán et al.	Construction of a multiprocessor that handles Lisp. (In Spanish) AHR-81-16	1981
Naranja 280	Adolfo Guzmán	A heterarchical multi-microprocessor Lisp machine. AHR-81-17	1981
(Thesis ESIME)	Nelly Gayosso	The distributor of the AHR machine, hardware version (microprogrammable). (In Spanish)	1981
Naranja 309	D. Rosenblueth	A full-screen text editor for microcomputers (In Spanish). AHR-82-18	1982
Amarilla 25	Adolfo Guzmán	125 Projects and Theses Topics in Computer Science, Third Edition (In Spanish). AHR-81-19	1981
Naranja 302	L. Peñarrieta, Nelly Gayosso	Alternatives for AHR distributor AHR-82-20	1982
Naranja 308	A. Guzmán, K. Norkin	The design and construction of a parallel heterarchical machine: Final Report of the Phase 1 of the AHR project. AHR-82-21	1982
Naranja 306	Norma Apodaca	The arbiters of the AHR computer. (In Spanish) AHR-82-22	1982



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).

B) PUBLICATIONS IN THE SCIENTIFIC LITERATURE

Guzmán, A., and Segovia, R. A parallel reconfigurable Lisp machine. Proceedings of the International Conference on Information Science and Systems, August 1976. University of Patras, Greece. 207-211.

Guzmán, A. Heterarchical configurable architectures for parallel digital processing with high level languages. Présentée á l' I Salon International de Informatique et Equipment de Bureau. Mexique 1979.

Guzmán, A. Reconfigurable geographic data bases. In Pattern Recognition in Practice, E. S. Gelsema and L, N. Kanal (eds), 99-112, 1980, North Holland.

Guzmán, A. Distributed computing as an alternative for the future. Informática 46, 23-32, Dec. 79.

Guzmán, A. A parallel heterarchical machine for high level language processing. In Languages and Architectures for Image Processing, M. J. B. Duff and S. Levialdi (eds). 1981 Academic Press. Also in: Proc. 1981 International Conference on Parallel Processing. 64-71. 81CH-1634-5.

Guzmán, A. A Heterarchical multi-microprocessor Lisp machine. Proceedings of the 1981 IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management. Hot Springs, Va. 1981. IEEE Computer Soc. 81CH-1697-2.

Guzmán, A., Lyons, L., et al. Construction of a multiprocessor that handles Lisp. Part I: THeory of operation of the AHR machine. (In Spanish) Memorias del VII Congreso de la Academia Nacional de Ingeniería. Oaxaca, Oax., 1981.

Guzmán, A., Lyons, L., et al. Construction of a multiprocessor that handles Lisp. Part II: Architecture of the AHR machine. (In Spanish). Memorias del VII Congreso de la Academia Nacional de Ingeniería. Oaxaca. 1981.

Guzmán, A. A multi-microprocessor that executes pure Lisp in parallel. Submitted for publications to IEEE Transactions on Computers, Dec. 1982. Special Issue on Parallel and Distributed processing.

Peñarrieta, L., and Lyons, L. An Image processing system. IEEE Computer Workshop on Computer Architecture for Pattern Analysis and Image Database Managmt. Nov. 1981. 293-300. IEEE Catalog 81CH1697-2.

C) PUBLICATIONS IN NEWSPAPERS, MAGAZINES AND PERIODICALS OF GENERAL CIRCULATION

"Información Científica y Tecnológica" Julio 1, 1981
COMPUTADORA MEXICANA DE PROCESAMIENTO EN PARALELO. pp.42-43.

"Gaceta UNAM" Agosto 20, 1981
Se diseña y construye en el IIMAS
PRIMERA COMPUTADORA DE PROCESAMIENTO EN PARALELO Y DE PROPOSITO GENERAL.
pp. 14-15.

"Novedades" Oct. 28, 1981
IPN Y UNAM DESARROLLAN NOVEDOSA COMPUTADORA. p 21

"Computerworld" Oct 12, 1981 pp 13-19
LA COMPUTACION DISTRIBUIDA COMO UNA ALTERNATIVA DEL FUTURO.

D) PRIZES AND AWARDS

PREMIO "CONFERENCIA ALEJANDRO MEDINA", Morelia, Mich. January 1982. Prize given to Adolfo Guzmán "for his contributions to the Computer Science field". During the award ceremony, the AHR project was specifically mentioned.

"Gaceta UNAM" February 8, 1982. p. 5.
FUE OTORGADO EL PREMIO "CONFERENCIA ALEJANDRO MEDINA" 1982.

E) THESES DIRECTED

L. Peñarrieta. Multiprocessing based in a memory shared by n processors. M. Sc. Thesis; School of Engineering. National University of Mexico. 1978. (In Spanish)

N. Gayosso. Distributor for the AHR computer, in hardware, microprogrammable. B. Sc. Thesis, ESIME, National Polytechnic Institute. 1981. (In Spanish).

Víctor G. Sánchez. General Information System with reconfigurable structure. M. Sc. Thesis, IIMAS, National Univ. of Mexico. 1981. (In Spanish).

A. Guzmán. 125 Projects and theses topics in Computer Science. AHR-81-19. (In Spanish). About ten projects and theses topics in this edition were inspired by the AHR Project.

WEAK POINTS IN PROJECT ORGANIZATION

In work organization within AHR, we can mention some drawbacks which caused a reduction in results. The main ones were:

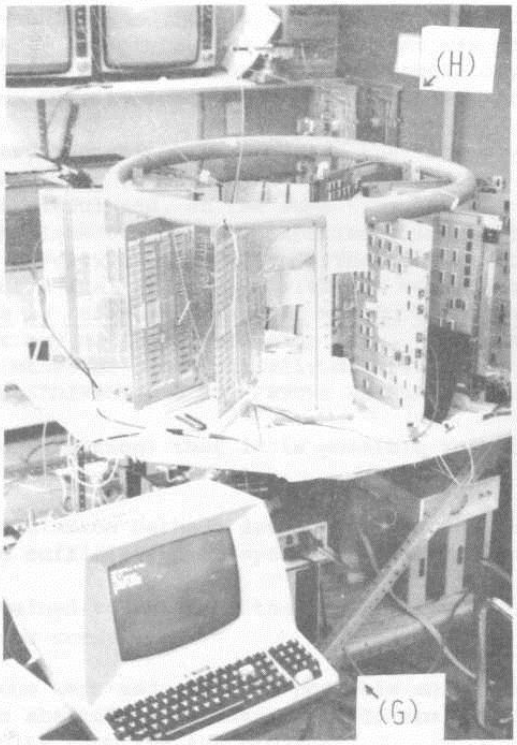
* Weak attention of IIMAS executives to the vital needs of some essential members of AHR. This produced undesirable turn over of people during AHR execution.
* It is a bad practice to have research assistants (students) without salary for a long period of time: for some reason, they tend to leave the project.

* Loading AHR members with very time consuming additional projects and duties.

* Absence of redundant personnel and resources. This redundancy is absolutely necessary, due to the presence of some random and unpredictable obstacles in such time-long and complicated project. It would have been nice to have had some surplus resources (manpower, computer circuits) available, to better meet these unforseen obstacles.

* Insufficient attention to design methodology. This caused, in the beginning, wrong estimation of man-months for the project, as well as the extra man-months for debugging and fixing the machine.

Finding these drawbacks in project organization is in fact, part of the educational goals of AHR. Because of this, we include in Appendix I of this report a summary of our understanding of "good organization" in AHR-like projects.



THE USER CONSOLE

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

MAIN RESULTS

Using new principles for computing process organization, the design of the AHR computer was finished, and a working prototype of the computer was built and checked. As a prototype, the AHR computer was not destined for normal use, but for verification of main ideas. It was completely sufficient for this purpose, and it became "living proof" of the feasibility of AHR premises.

From the very beginning, it was felt that our way to organize the parallel work of constituent computers in a multiprocessor system [Naranja 133] promised to be very fruitful and, at the same time, it was very doubtful. These doubts existed for the lack of concrete experience in the construction of such systems, and for the novelty of the design. The only way to solve these doubts was by construction of a real computer. With the conclusion of the AHR project, we now know that this is indeed a practical way to interconnect several machines. These machines may or may not be of the same type. These machines may or may not be close to each other or geographically distributed. This will be explained in more detail in part "DIRECTIONS FOR FUTURE RESEARCH."

* It was shown (by construction) that it is possible to build an entire parallel computer in Mexico.

* We can trust the simulation methods developed, because they were checked in practice and showed sufficient correspondence between simulation and measurements.

* Enough data was obtained to estimate the possibility of competing or not competing with other commercial computers.

* Practical estimations were made for the possible costs in producing AHR-like computers. This is absolutely necessary for determination of best ways to use the scientific results from the AHR project.

* Practical data about complexity of different parts of the computer (see table "AHR CHIPS"), and about time delays in different parts of the computer were obtained [AHR-80-10].

* The AHR design is very flexible and it gives essentially a well designed and practically checked concept about interconnection and parallel work of many homo and heterogeneous systems.

* We gained considerable experience in hardware and software during this project.

  In hardware,

  * experience in high speed digital electronics.
  * practical experience in building microprocessors modules and software.
  * shared access to common memories.
  * spies (which make unnecessary input/output with the user)
  * windows which map one address space to another
  * how to make a smart memory, with "sensible" addressing modes

* arbiters and acopladores (couplers)
* fast data loading using "mail box" techniques.
* broadcast of data to all Lisp processors (cajas).
* How to go from a piece of complex software (the distributor in this case) to a piece of hardware that executes the same function.

In software,

* recursive programming for microcomputers.
* programming in the presence of parallel computation.
* programming for a purely applicative language.
* garbage collection helped by special hardware.
* multiprocessor system debugging.
* experience in applicative programming.
* synchronization of applicative processes.
* we gained experience in simplifying the software, in particular resource management software which takes care of computation control ("bureaucratic software"). This software was later replaced by hardware ("bureaucratic hardware"), and we also gained experience in the production of hardware for computation control.

After this project, the members of IIMAS acquired a practical background in the design and construction of digital computer systems, as well as a feeling of participation in a leading research avenue. In addittion, the AHR project triggered the formation of a new department in IIMAS called the Computing Systems Department, as an affirmation of qualification growth.

ENDING PHASE 1 OF AHR.

The goal of Phase 1 of the AHR project was to construct a prototype of a computer designed using AHR principles and ideas.

* The machine was built.
* It operated successfully, according to its design.
* The machine was not sufficiently reliable for sustained (several hours) operation. It failed frequently, due to our insufficient experience in some technological aspects of its construction, such as connections, weak contacts, short circuits, and reflections of pulses.
* Since the prototype proved success in the design, and since the machine was not able to support practical applications, it was decided to close Phase 1 of the AHR Project.
* As discussed in the section "Directions for future research and applied work, Phase 2 of the AHR Project should tackle the important problems of
    * building a realible (high mean-time-between failures) version
    * Improving further the design, for instance by incorporating distributed computation.

	ACOPLADOR (coupler) x 5	LISP PROCESSOR x 5	CONTROLLER passive memory	CONTROLLER variables memory	GRILL INTER-FACE	VARIABLES MEMORY	PASSIVE MEMORY	VIDEO MAPPER x 5	WINDOW	WINDOW INTER-FACE	TOTAL
Large Scale Integration	1 x 5	18 x 5	2	2	4	96	96	1 x 5	0	4	224-304
Medium Scale Integration	16 x 5	4 x 5	21	21	0	10	10	10 x 5	10	2	104-224
Small Scale Integration	49 x 5	15 x 5	71	71	0	9	9	17 x 5	15	9	265-589
TOTAL	66 x 5	37 x 5	94	94	4	115	115	28 x 5	25	15	593-1117

T A B L E "A H R C H I P S"

If we count each different circuit only once, their total number of chips is 593. Taking into account five Lisp processors with their five couplers and video mappers, the number increases to 1117. This is the total number of chips in the current version of the AHR machine.

The laboratory prototype of the AHR computer consists of approximately 600-1100 chips. These chips are mostly large scale integration, which means that the AHR project represents, under contemporary classifications, a very complicated electronic design. This project, which demanded such a huge amount of hardware work devoted to a unique goal, is new for IIMAS and, so far as we know, for Mexico.

CRITICAL ANALYSIS OF VERSION 1 OF THE AHR MACHINE

As a whole, AHR Version 1 met the strong requirements imposed by the functional approach, and completely verified the theoretical premises given in [Naranja 133] . The organization for parallel work in homo and heterogeneous units appears to be a very effective way of eliminating all of the difficulties inherent in conventional parallel computing systems. The price of these advantages is that the user has to wite programs in pure Lisp.

In practice however, the necessity to obtain concrete results given limited resources motivated the members of AHR to make some simplifications to the initial design. In other words, those parts of the design which were not essential for checking the main idea were eliminated at the cost of possible future commercial applications of the AHR machine. Specifically, the initial design was simplified in the following way: elimination of any traditional software except Lisp; elimination of traditional input/output facilities; elimination of large system software overheads such as complex utilities and maintenance routines.

AHR Version 1 was built as a tool for checking the special properties of the Lisp language for performing parallel computation [Naranja 133, AHR-81-17]. The AHR computer proved both the suitability of Lisp for parallel processing, and the convenience of our hardware-software architecture. In addition, it confirmed our initial guess that it will be extremely easy to program in pure Lisp, without the programmer having to worry about the parallel execution of his program: it "executed in parallel" in a natural manner.

To generalize these results, we need to look at how our Lisp computer works as a whole. We may think of a Lisp program as no more (and no less!) than a description of a function L working on an initial data X. In general, this initial data is a set of lists of arbitrary length and arbitrary nature (letters, numbers, etc.). Function L is written as a superposition of Lisp primitives or more complicated mathematical constructions using simple rules. Since a Lisp program and its result are lists, Lisp function superposition always has a tree-like structure. Branch points of this tree are Lisp primitives and the branches themselves reflect the use of the results of the function evaluation (outputs) as initial data for subsequent calculations (inputs). For some functions, all input variables are known at some time (the function is ready for evaluation), and therefore they permit this function to be evaluated. As a consequence of this evaluation, other functions will become ready and may be evaluated. Therefore, the only signal or condition for function evaluation is the readiness of its arguments. This readiness can easily be checked during the replacement of the operand's name by its value. The checking operation is independent (each function can do the checking by itself) given the structure of the function operands. In the AHR system, the main rule is that any ready function must be evaluated thus giving rise to the phenomena that the computation in the system is spontaneous and is self-synchronizing. The lesson gained from this experiment is that if a user takes on the description of a computation using the Lisp language, he automatically rids himself of synchronization problems.

SIMILARITY BETWEEN PURE LISP PROGRAMS AND TREES OF HARDWARE COMPUTING STRUCTURES

To begin, let us underline the fact that pure Lisp corresponds to circuit designs in hardware computational structures (hardware that computes values from inputs) which have no feedback. For example, if we match the two figures "Calculation of Algebraic Expression Using Hardware .." and "Lisp program Structure ..", we can demonstrate this similarity. The non essential differences appear when using "multi-input" multiplications in Lisp.

It is well known that this type of hardware configurations permits the solution of any known computer problem, if the number of computational elements or structures is unbounded. In practice, however, the purely general solution has limited applicability for they require a very large amount of computing elements. The same is true for Lisp programs and Lisp computers. Lisp programs may contain a very large amount of constituent functions especially during a computational process. However, for a Lisp computer, this drawback is in no way fatal: each constituent function requires only a few words in memory which turns out to be very cheap. Consequently, it is possible to write very long programs. Despite the fact that pure Lisp in a Lisp machine requires little memory for each constituent function, it is nevertheless very desirable to reduce the length of the programs given specific and "real world" problems.

By using feedback in hardware and reconnecting used computational elements to places in which the computational process has not started yet (in these computational elements, the complete set of evaluated arguments does not exist), the difficulties mentioned above are annuled.

A related issue to number of elements necessary for carrying out a computation is that of what determines the number. That is, the number of necessary elements is not determined by the length of the program, but by the size of the problem. In this way, theoretically, it is possible to write programs of arbitrary length, which in practice means very long programs, and build complicated hardware computer structures.

More precisely, without reducing the computational speed, we can solve problems which demand more computing elements that there are in the structure. More over, in some recent work (K. Norkin, "Specialized hybrid equipment for computation and control"*--in Russian--) it was proved that for hardware computing structures, optimal degree of parallelization is not always the same as maximum possible degree. There are some cases known in which the reducing of parallelization degree does not produce any reduction in speed. For example, in some cases evaluation results are not immediately consumed, and in this case, we can delay some computation without increasing total time. In some other cases, using performance cost criteria, it is possible to reduce the degree of parallelization despite some increase in computation time. For example, the maximum degree of parallelization can be reached only during a short period of time. That being so, the optimum will move to lower degrees. The same approach is valid for optimum memory size and optimum number of Lisp processors ("cajas") as in AHR. In the current design of AHR, the number of "cajas" is nowhere near the optimum in the sense of [AHR-79-5], but it can be easily changed in the future.

* Норкин К.Б. Специализированные гибридные управляющие вычислительные устройства. М. "Энергия" 1980 г.

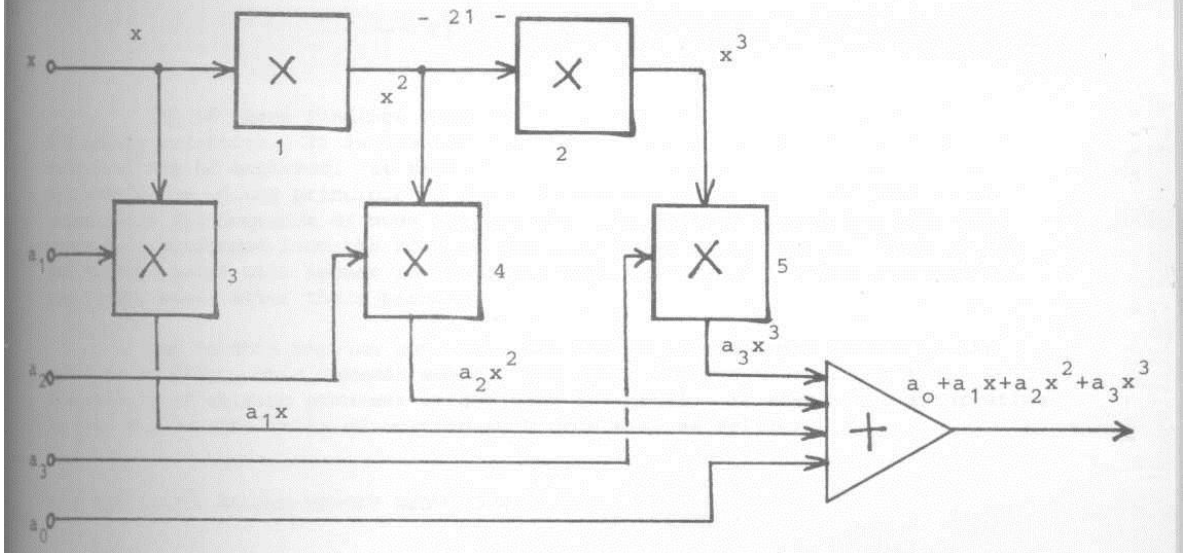


FIGURE 2 "CALCULATION OF ALGEBRAIC EXPRESSIONS USING HARDWARE COMPUTATIONAL STRUCTURES"

Multipliers are marked with X; adders with +.

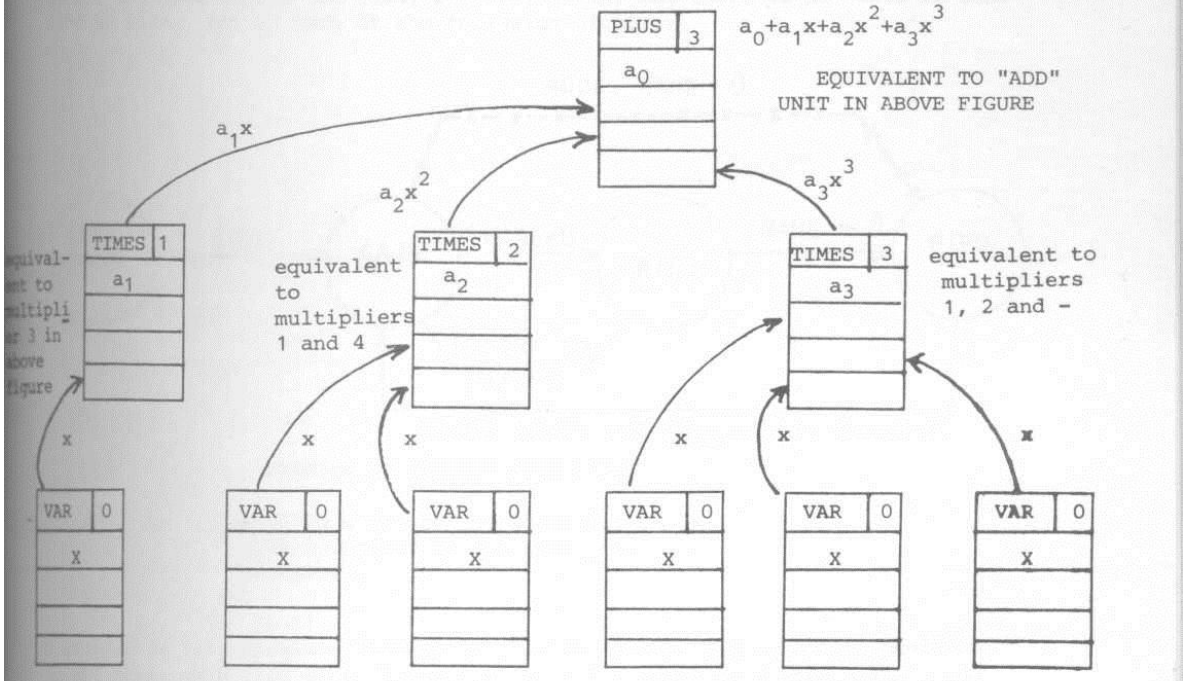


FIGURE 3 "LISP PROGRAM STRUCTURE FOR Y= $a_0 + a_1x + a_2x^2 + a_3x^3$ CALCULATION."

All of these findings were not fully applied to Version 1 of AHR for memory reduction. It is now our purpose to describe how additional memory reduction may be achieved. It seems that the only possible obstacle for the systematic use of the principle of feedback and reconnection of computational elements is the sequence of node copying that the current design now has. This copying is performed from the root of the tree towards its leaves. That is why the "ready" nodes will appear in the grill near the end of a copying operation, but in any case, after their parents.

Due to this copying approach, the entire program must reside in the grill or in fifo. This demands memory, and (more important) excludes the possibility of solving problems larger than the grill. It seems very interesting to find the possibilities of solving such problems, as follows.

SUCH SIMILARITY ALLOWS MEMORY REDUCTION

In the current design of AHR, node copying and node execution are assumed to be two instances of the same type of work to be performed. These two operations are similar in the sense of its executing rules, and are performed by "cajas." As it will be shown later, this similarity can be generalized. It can also reduce redundant data flow: nodes which have been copied from passive memory and have nane=0 are ready for evaluation, and there is no sense to send them to fifo. Let us look at the following picture:

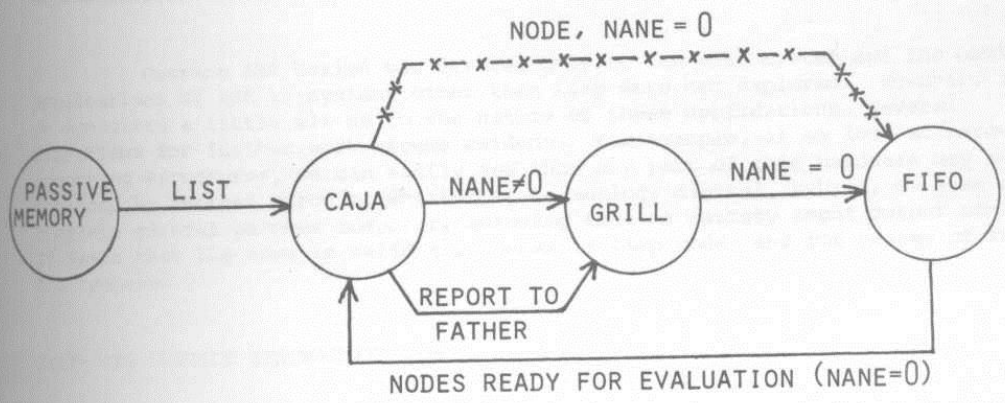


DIAGRAM OF CURRENT DATA PATHS

With -x-x-x- the redundant data path described in the text is signaled.

A program in list notation resides in passive memory. Sequentially, beginning at the root, each list goes (irrespective of its nane) to a free "caja", and then a node is produced. Independently of its nane, this node is spit out of the "caja" and is sent to the grill (if nane $\neq$ 0) or to the fifo (if nane = 0). Further, it is executed under common rules: from fifo to "caja", from "caja"

to "output bureaucracy"; and its father stays in the grill or is sent to fifo, depending on its new nane. It is evident that for nodes with nane=0, the circulation "caja"-fifo-"caja" is useless (The redundand path for this cycle is marked with -x-x-x- in the figure). The percentage of "ready for evaluation" nodes using current sequence of copying is rather small, and that is why the software simplification used now in AHR, which is responsible for this redundant circulation, is tolerated. But if we assume [the reality of this assumption will be explained later] that we could find some ways to carry out program execution from the very beginning of the copying process, the percentage of the "ready for evaluation" nodes will increase, and the drawback introduced by the redundant path will become more visible.

ITERATION AS A SPECIAL CASE OF DATA FLOW

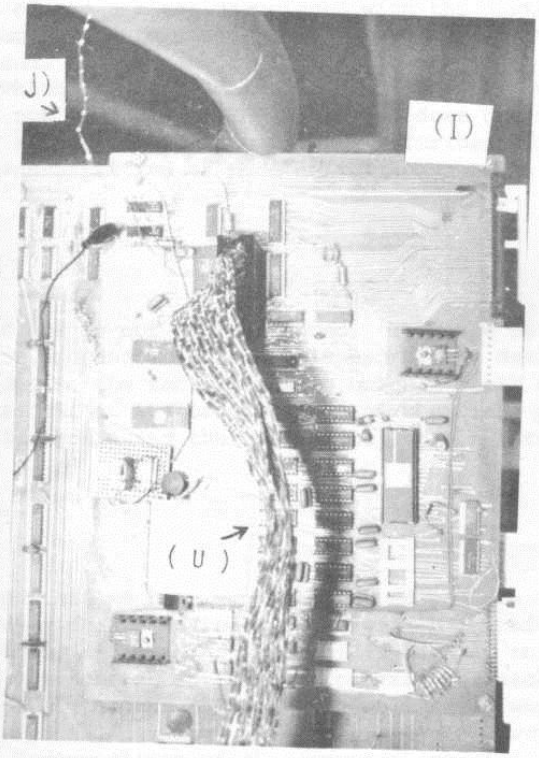
The drawback mentioned above is not the only source of redundant data flow in AHR. Cases frequently appear in real programs in which the father has the same name as a son, and the input variables of the father are the output variables of his only son. As example of this is the operator "DO". Existing AHR software treats this case in the standard manner, producing redundant data exchanges and overloading the grill with redundant relatives. It is now easy (see Data Circulation Improvement later in this report) to see how to change the AHR software to get rid of these useless relatives.

GENERALIZATION OF AHR COMPONENTS

Current AHR design was devoted only to the Lisp system and the obvious applications of AHR to systems other than Lisp were not explored. However, if we speculate a little bit as to the nature of these applications, several directions for further work become evident. For example, if we look at hardware computing structures, we can easily see than any part of this hardware may be replaced by another circuit, whether it be analog, digital, hybrid, or even a complete general purpose computer, assuming that we satisfy input/output conditions. It seems that the same is valid for pieces of Lisp code, and for pieces of the AHR computer.

SELF- AND OUTSIDE SYNCHRONIZATION

Another area of investigation has to do with the possibilities of synchronization of computing processes with other processes in the environment outside the computer, or with processes inside the AHR system. On the other hand, in the AHR Version 1 design, the approach was exactly the opposite: the goal was to eliminate completely any artificial synchronization. The current design proved this goal to be attainable. However, in sompe practical applications, such as in real time control systems, it is necessary to synchronize the computations with a real time clock or with other events.



DEBUGGING A LISP PROCESSOR

The software of the Lisp processor (I) shown is being debugged from a Z80 -- microprocessor (D) through an "umbillical cord" (U). (J) is the line to the TV spy (A), from the Lisp processor.

RECONFIGURABILITY

Included in the initial concept of the AHR machine was that of reconfigurability. However, in the first prototype version of AHR, we discarded the possibility of reconfigurability and made the cajas universal. Consequently, this simplified the software, required a single fifo and simplified the design of the arbiter associated with the distributor. The effect of not including the feature of reconfigurability was that the "caja's" memory was increased and, at the same time, that it very much limited the number of primitives which can be used in a real program. The fact that reconfigurability was not included in AHR Version 1 does not mean that it is not worthy of further attention and research. Any further effort should include the above mentioned generalization of node definition.

BALANCE IN AHR

Version 1 of AHR is not well balanced. The size of its memories, the number of "cajas" (Lisp processors), and the speed of the data exchange do not mesh well. This was due to the fact that such balance could not have been known in advance, but as a result of the construction of the first prototype, we now posseess enough data for achieving the desired balance in future designs.

WEAK ENGINEERING POINTS

The handicaps that we have mentioned up to now are structural in nature. But there are engineering drawbacks too. These will be mentioned briefly and in little detail. One of the engineering weak points is that the AHR machine has a redundant amount of interconnection wires. For example, each "caja" has 290 connections to the rest of the AHR machine. The well known ways of reducing the number of connections by multiplexing, time sharing, etc., were not used. Another drawback was that the decomposition of the computer into its constituent printed circuit cards was not done in such a way as to make each card simple and therefore easy to test. And finally, because of the random collaboration of many people in the project, the interconnection system of the computer becomes cumbersome.

All of the drawbacks mentioned up to now* do not invalidate the main idea of the project. Moreover, we have some proposals to eliminate them or at least soften their effect. Most of these proposals do not require making any essential change in the hardware, but we prefer to expose them later in the report.**

---

* It is important in a final report to include for completion all weak points and handicaps; today, it is easy to know and to remember them. As time passes, our understanding of them will become more fuzzy, less clear and eventually they will be, naturally, forgotten.

** We separate them because of their different natures. A final report should also contain proposals for future realization that render the design more complete and closer to perfection.

In the next section, we will only trace the possible solutions, assuming that further work will be done for improving the AHR design and ideas.

However, before we go to the next section, we would like to mention that during the AHR machine design, we did not preoccupy ourselves with issues of its mass production. This is not a drawback of the design, but we should, nevertheless, devote later in the report some space to this issue.

DIRECTIONS FOR FUTURE RESEARCH AND APPLIED WORK

AREAS OF APPLICATIONS

Paradoxically, despite the fact that we believe in the high efficiency and wide range of applications in the future of AHR-like machines, the particular implementation of AHR ideas or concepts in terms of a machine which processes only Lisp and consists of units which are not useful by themselves does not have a sufficiently large commercial market to warrant mass production. Even in the United States, home of Artificial Intelligence, the demand for such machines numbers between 10 or 20 machines a year. In Mexico, the demand may be 1 or 2 machines every six years. It is therefore obvious that it will never be profitable to produce Lisp machines on a mass production basis in Mexico, unless of course, the Federal Government decides that areas of research in which Lisp machines are necessary must be heavily funded. But a more realistic view of the situation would cause us to define the problem as to what to do with the results of the AHR research in terms of finding the proper decomposition of the whole computer into separate units, which in turn, can be sold separately. The market would no longer be that of Lisp machines, but of parts that have commercial, industrial or business applications in a much larger market. It goes without saying that if the need arose, the parts could be assembled into a Lisp machine. Each of these separate parts will then be (a) useful by itself, and (b) useful as component of a Lisp machine.

If we solve this problem of decomposing the AHR machine into commercially viable parts, we can profit from the fact that the AHR machine is not only the first computer built and designed in Mexico with some original ideas, but also that it demonstrated new and rather efficient protocol conventions for data exchange between separate parts working in parallel. (*)

Without going into details, we can describe an AHR computer as being made up of several microprocessors, three types of memory devices, a monitor system, and a data exchange system. Let us look at the characteristics of each one of these components in terms of what they have in common with standard computer components, and also in terms of their uniqueness.

The microprocessor's activities in an AHR computer is as follows. The normal state of the processor is that it is waiting for work. It holds in its memory the necessary programs for calculation. The other parts of the AHR computer may then cause the microprocessor to begin a computing process, by sending it some data in terms of a node from the fifo, or a list from the passive memory. The data is passed to the microprocessor with all of the necessary information for computation: program name, pointers to input variables or the input variables themselves, and a definition of a rule for using the result, which in turn, may be a list. It should be noted that the function name is actually the starting address in program memory, and consequently, each microprocessor can be replaced by almost any mass-produced

(*) That is, we can both sell the AHR computer as a whole, or sell each of its parts separately, since they are useful by themselves.

microcomputer containing the same memory size (32K bytes in our case). When comparing the AHR microprocessor with a conventional microcomputer, we notice the presence of an "acoplador" (coupler). This acoplador permits data to be exchanged very fast. Such speed, which is vital for Lisp machines, is also very useful for conventional systems, where data produced by one computer should be readily available to others. In addition, the AHR microprocessor and the microcomputer differ in their memory access logic. AHR's memories are smart and can be accessed in a variety of useful modes [AHR-80-10, AHR-81-17]. But these good access schemes also seem to be very useful for most standard applications. Therefore we may conclude that all "specific" AHR designs of the microprocessor are also of "general" use.

All of these properties of the AHR machine are desirable in any application, and because of this, we must realize that just because we use the "caja" in a Lisp machine, does not imply that some of its properties are incompatible to commonly used computers.

We can also use ROM for Lisp programs resident in each Lisp processor (each caja), but this does not seem appropriate even for Lisp calculations, because by rewriting the content of program memory, we can produce reconfigurability very easily. The situation will change considerably if we are thinking of manufacturing our own Lisp chip, which is the path chosen by MIT's SCHEME chip.

Memory units of the AHR computer perform specific tasks. Nevertheless, it is evident that all of these functions may be performed using memory with "intelligence" (a variety of reading and writing modes). It is vital for a Lisp machine to have high speed read/write operations and data transmission, as well as in many other applications. Thus, an "intelligent" memory which is necessary for Lisp machine can be profitably used in general applications.

AHR monitor computer (the "input/output" processor) is a conventional Zilog MCZ microcomputer.

So, "cajas", memories and the monitor system are units that can be produced independently as useful "everyday" computer devices, as well as constituent Lisp machine devices.

TRANSFORMING CONVENTIONAL MULTIMICROPROCESSORS INTO AHR MACHINE

Taking the observations above, we can outline two possible approaches for applying the technical results of the AHR project.

The first approach is taking the new ideas developed in AHR and applying them to conventional multimicroprocessor systems. A typical example of such a system is shown in Figure "Conventional Multimicroprocessor System". It usually consists of one monitor computer, a number of multimicroprocessor boxes each of which has its own memory, and a common memory. This last memory usualy possesses some intelligence to solve access comflicts and avoid time delays. A common difficulty in building this system is assuring a fast data exchange between the multimicroprocessor boxes and the common memory. In AHR, this problem also exists.

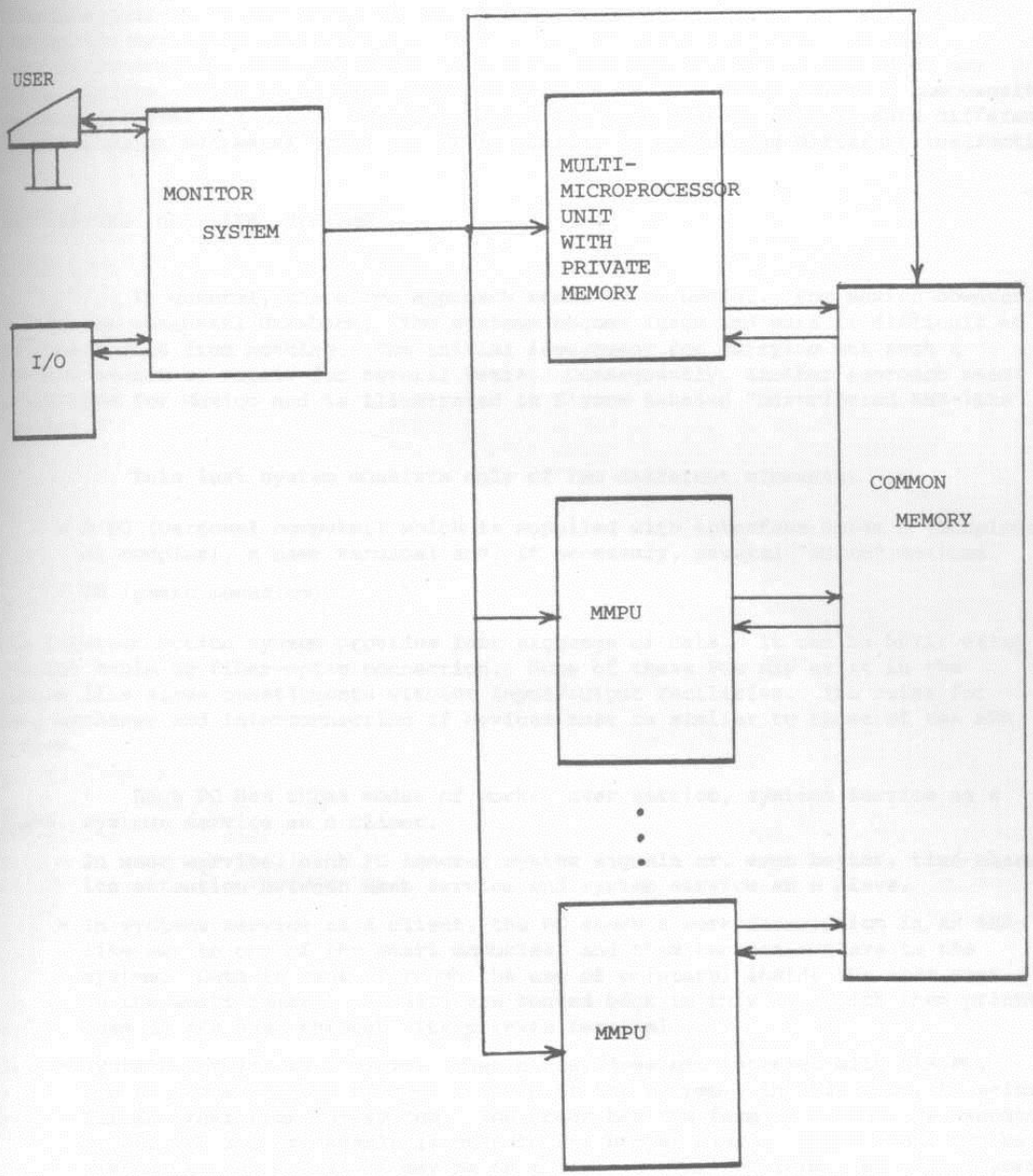


FIGURE 4 "CONVENTIONAL MULTIMICROPROCESSOR SYSTEM"

However, it seems that the application of AHR ideas may be possible through the simple expedient of modifying the multimicroprocessor software, and will help reduce the mentioned difficulties. More over, it seems that even for SIMD (single instruction multiple data) computers, the application of AHR ideas may be profitable. This is in fact possible because in Lisp there exists a homogeneity between programs and data. "Multiple data" may mean that we send as data different starting point addresses (That is, it is similar to performing different instructions).

DISTRIBUTED AHR-LIKE SYSTEMS

In general, the above approach seems to be useful. For Mexico however, it has one essential drawback. The systems become large and make it difficult to be constructed from nothing. The initial investment for carrying out such a project cannot be repaid for several years. Consequently, another approach seems preferable for Mexico and is illustrated in figure labeled "Distributed AHR-like Systems."

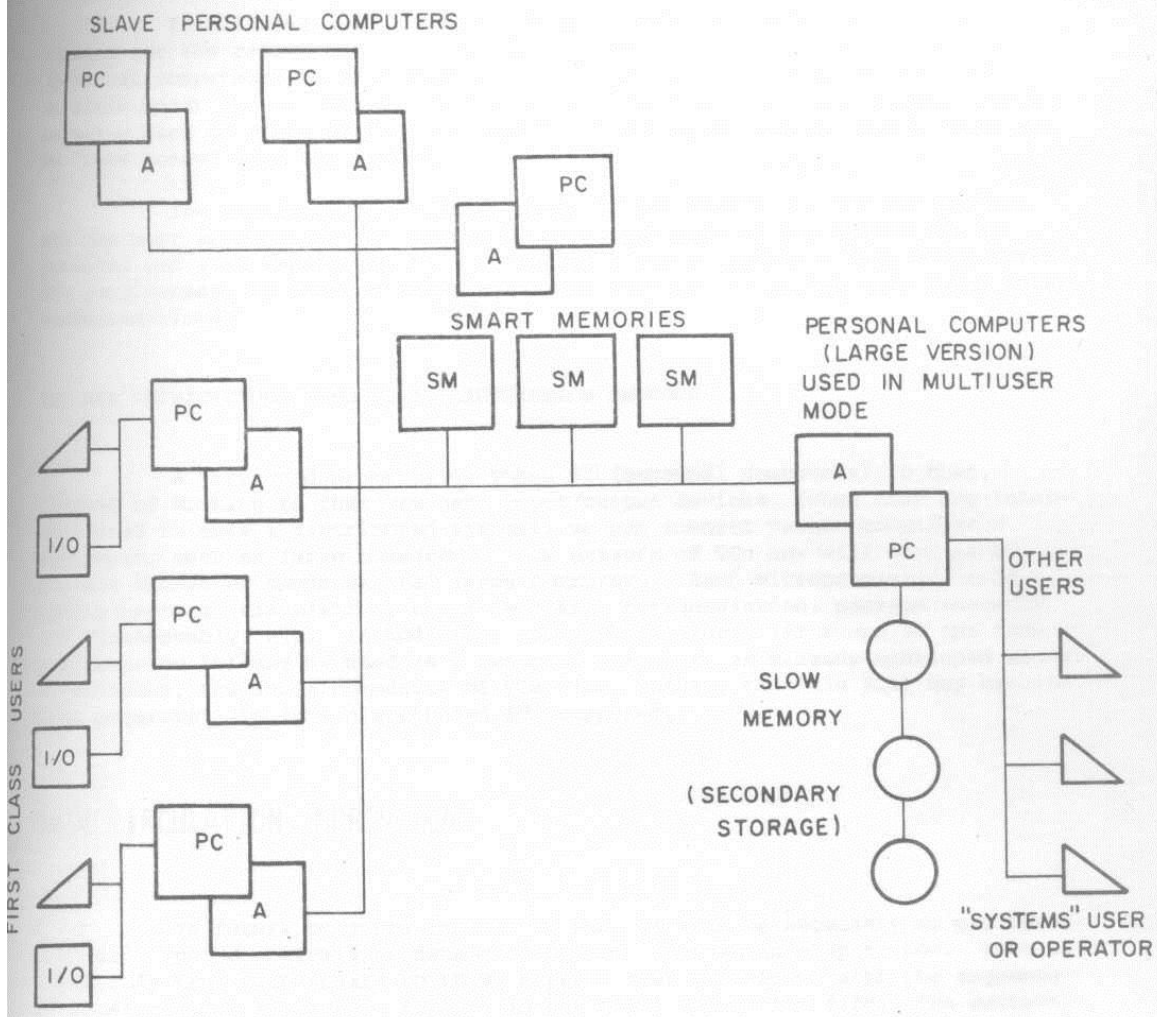
This last system consists only of two different elements:

- = A PC (personal computer) which is supplied with interface cards A (acoplador, or coupler), a user terminal and, if necessary, several "extra" devices.
- = SM (smart memories).

The interconnection system provides fast exchange of data. It can be built using coaxial cable or fiber-optic connection. Some of these PCs may exist in the system like slave constituents without input/output facilities. The rules for data exchange and interconnection of devices must be similar to those of the AHR system.

Each PC has three modes of work: user service, systems service as a slave, systems service as a client.

- = In user service, each PC ignores system signals or, even better, time-shares its attention between user service and system service as a slave.
- = In systems service as a client, the PC sends a work description in an AHR-like way to one of the smart memories, and them becomes a slave to the system. Data is sent (through the use of pointers) inside the node sent to the smart memory. Results are routed back to this PC, which then prints them to its user through its private terminal.
- = Systems service as a slave. When a user does not interact with his PC, the PC automatically becomes a slave of the system. In this mode, it waits for an order from the system. The order has the form of a node, is executed by the PC, and its result is sent to the proper place. These nodes may be similar to AHR nodes, or may be or a more generalized form. We will describe this general form later. But for the time being, we can say that these generalized nodes may produce some specific new desired performance of the PC, such as printing some output, loading the private memory of PC with new data (i.e., new programs), etc.



PC PERSONAL COMPUTER
A ACOPLADOR (COUPLER)
SM SMART MEMORIES

FIGURE 5 "AHR-LIKE DISTRIBUTED SYSTEM"

In building such a system, we also have to design the rules of billing for the resources used. These rules must force the user of PCs (personal computers) to make them available for system work (to place them in slave mode) for as long as possible, since in this way his personal computer is being used by other people, through the distributed system, and in this way he "gets money" from the system.

The applications outlined above of the results of the AHR project are the most appropriate for Mexico because each part of the system can be produced and sold separately thus providing a way to access a very large market. For conciseness, we will be thinking in the rest of the report only about these recommendations.

THE AHR ARCHITECTURE USED AS A TRANSMISSION NETWORK

A very good property of these PC (personal computers) is that, instead of hooking to them ordinary input/output devices (when they are interconnected to form a distributed system), we can connect rather complicated equipment, such as large computers. Our network of PCs now will work as a packets exchanger among several large computers. Each microprocessor could handle protocol translation, error recovery, retransmission, package assembly and dissassembly, etc. In addition, each microcomputer (if there is cpu time available on it) may be used as a personal computer, as already mentioned above. In addition, the whole community of PCs (or, rather, the idle PCs) may execute Lisp programs: it is a distributed AHR computer.

DATA CIRCULATION IMPROVEMENT

In future projects related to AHR, it will be necessary to pay more attention to the analysis of data circulation than we have up to now. This is especially true with regard to those aspects that have to do with the sequence of loading nodes in passive memory, in the grill and in the fifo. The section on Critical Analysis of this report seems to suggest that it is more efficient to perform the loading of nodes in a sequence that is opposite to that used in AHR, that is, it will be more efficient to load them from the leaves to the root of the program. This change seems feasible and, not only that, we may consider using the proposals described in [Naranja 214]. In that cut-line representation, each string contains pointers to the sons and a pointer to the father (it seems that many other representations are also possible). Consequently, the nodes can be sent in any desired sequence to passive memory and to the grill. Moreover, it is possible to select the fathers in accordance with the sons that are sent to passive memory, requiring that only part of the computing program resides in fast memory --and we may begin its execution immediately. This is similar to virtual memories found in many computing systems. However, in our system there exists the -- added advantage that the AHR system sends to the -- grill (where the nodes are evaluated) nodes which are guaranteed to be evaluated in predictable time in the future. Thus, we have a scheme of "predictive paging"

PREDICTIVE PAGING

or "predictive swapping", because each node "knows" that his father should be evaluated soon after him.

GENERAL NODE DEFINITION

In addition, it seems useful to generalize the definition of a node in AHR. For example, we can permit some nodes not to have a real father. And we can have a node that is considered as "no operation." Also, with reference to figure "Improving Data Paths in AHR", when the PC is working as a client, it sends nodes to the "smart" memories (which are equivalent to passive memory) in a sequence that guarantees that the nodes appearing in passive memory are ready to evaluate.

This process of sending nodes is done continuously, and at the same time, passive memory starts to send nodes to the "cajas" for their transformation fron list notation to node notation.

HOW TO AVOID SUPERFLUOUS CIRCULATION OF NODES

The transformation is performed as in AHR Version 1 except that, if the transformed node has nane=0, it does not leave the caja, but is evaluated immediately. Its value is sent to its father. If the father exists in the grill, the result is processed in the normal way. However, if the father does not exist, a new node with the function-name "no operation" is created with nane = 0 and is then sent to the fifo. After being in the fifo, the node goes to the caja, then to the fifo again, etc., until its father appears in the grill.

The sequence of copying just described guarantees that the father will exist in the grill in most cases, and the dummy operation is necessary only in a few cases in which the father does not exist.

HOW TO PERFORM ITERATIONAL PROCEDURES

There are other ways to guarantee the existance of the father. When the node come -- out of the fifo, it already has a pointer to the father, which means that we can check the presence of the father immediately (as soon as his son comes out of fifo) and , while the son is being evaluated by a caja, another caja could use that time to ensure the existence of the father. All of this does not require any change in hardware, but it does require reprogramming. If the father exists in the grill, further flow is as in Version 1. It is very important that we allow the pointer to the father to be "IBID". In this case, the node stays in the caja and the calculations are repeated until some condition becomes true. When this happens, the "pointer to father" replaces "IBID" by a pointer to its true father (the true father pointer was stored in the caja, outside the node, while the iteration was in progress) and the iteration ends. If we make this specific pointer to father to be a function of computational results, we can very easily perform most of iterational procedures, reducing in this way the need for duplicating similar fathers in the grill.

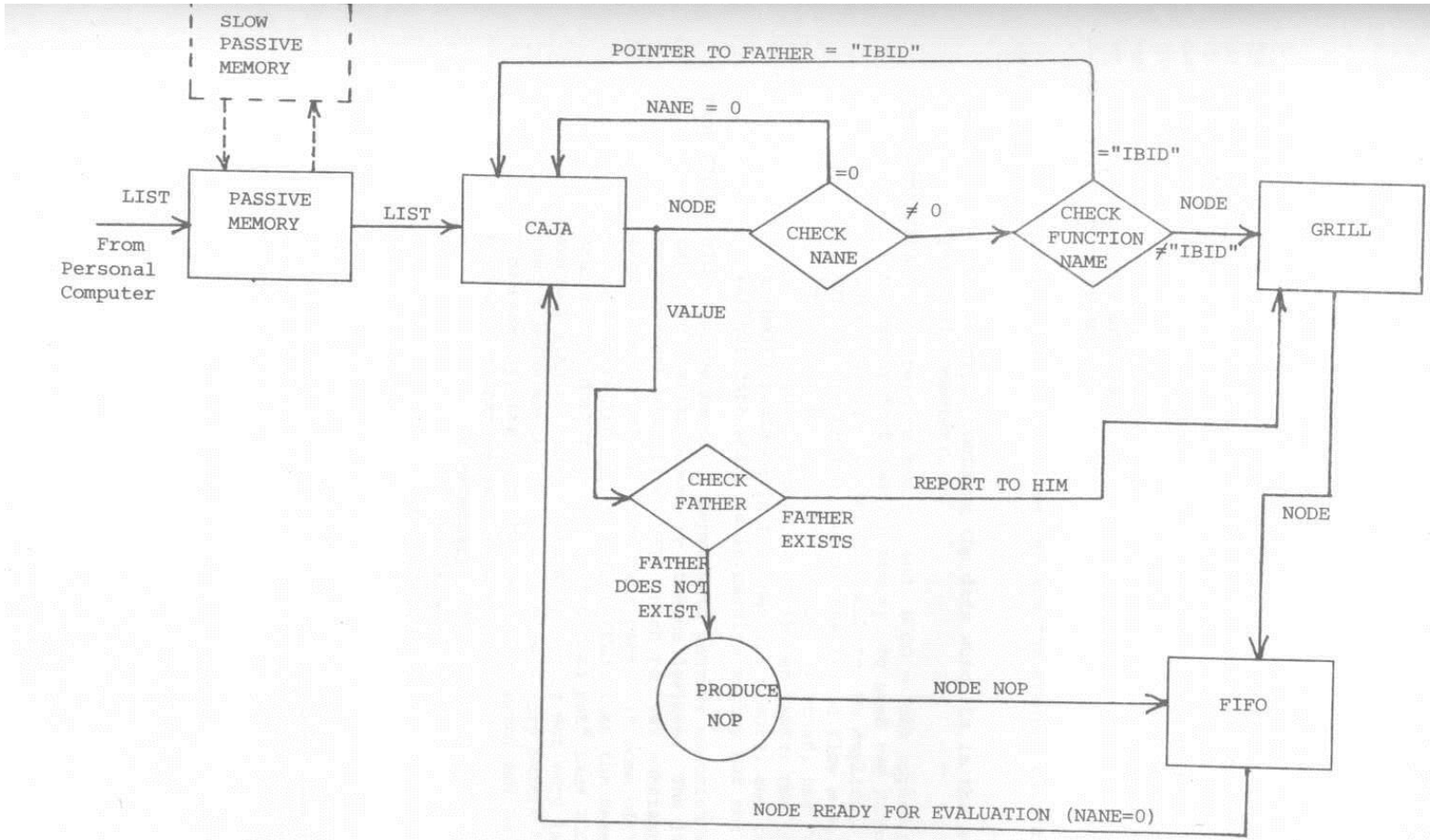


FIGURE 6 "IMPROVING DATA PATHS IN AHR"

With the introduction of node NOP, as well as father=IBID, the data circulation in the AHR machine is enhanced, within the context of applicative languages.

LIQUID FLOW ANALOGY

To clarify our proposals, let us use a chemical-hydraulic analogy. We will consider data as a suspension of different substances in a liquid. Each substance represents a particular type of data. For example, one may represent nodes, and another some final results of a node evaluation (that is, values). We can consider calculations as similar to chemical processes which transform nodes substances to value substances. To illustrate this analogy, we use the figure "Liquid Flow Model For AHR". In it appears the "necessary chemical equipment" such as tanks, filters, reactors, pumps, cocks, valves and mixers. We arrange the cemical processors to produce substances which are equivalent to final results, that is, values.

By inspecting the diagram, we can see the analogy with different parts of the AHR computer:

Tank USEM = user memory. Contains the data which is in the user's PC (personal computer).

Two tanks FAPA = fast passive memory and NOCO = "not copied yet", both represent fast passive memory. We need two tanks to represent data copying. In the AHR machine, we duplicate data in the process of copying, but in the liquid flow model, it is impossible to duplicate liquid. Instead, use "double quantity" of the liquid by putting it in two tanks: one to reflect data actually residing in fast passive memory, and the other to reflect data which is in passive memory but not yet copied.

Tank SLOP = slow passive memory. Represents passive memories which are disks, tapes or similar secondary storage devices. The liquid arrives to this tank via filter VFL3 which removes substances from the liquid coming out of above tank FAPA. If some of the substances are not extracted by the filter VFL3, then the same substances are brought dowm from the "not copied yet" tank NOCO through a special slanted tube. Naturally, it is not very intelligent to send a large amount of data to slow memory before they are copied. Consequently, the fact that some substances are not affected by VFL3 will guarantee zero flow through the tube that connects NOCO and VFL3. If necessary, PUMP1 returns these substances to FAPA and NOCO but they are filtered by VFL1 first. This last operation represents the addressing of the contents in slow passive memory.

Tank FIFO = fifo. Represents the data in fifo. We assume that this tank is long and thin, and suppose that PUMP2 has an elastic input tube which is always touching the liquid surface. The long and thin characteristic of the tank guarantees the first-in-first-out effect of the liquid.

Tank PARA. It represents the grill of the AHR system.

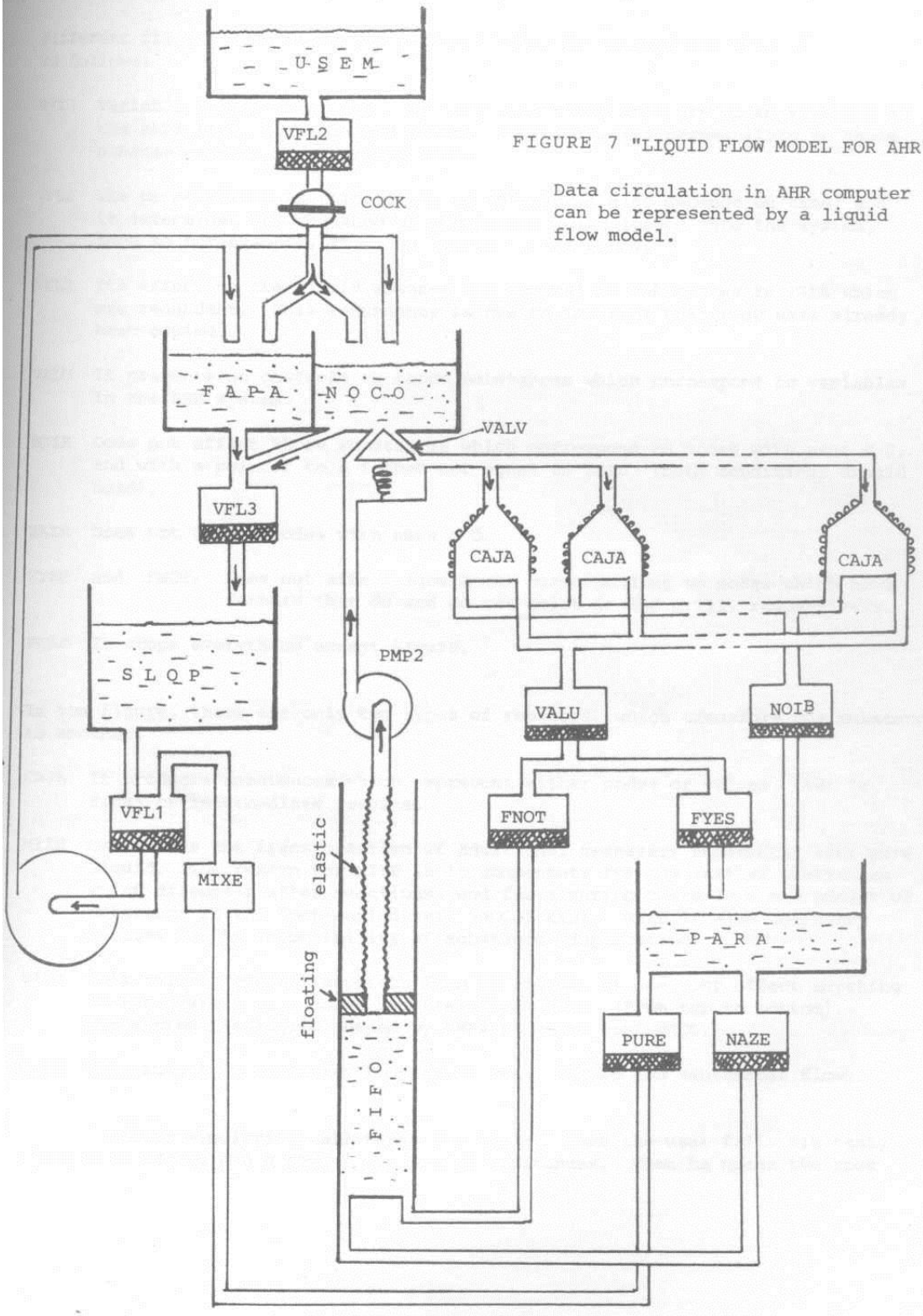


FIGURE 7 "LIQUID FLOW MODEL FOR AHR"

Data circulation in AHR computer can be represented by a liquid flow model.

Different filters have different permeabilities to substances. This is specified as follows:

VFL1 Variable filter which does NOT stop substances that are to be returned to the FAPA tank, as explained above. Therefore, its permeability to these substances must vary durying time.

VFL2 The permeability of this filter on substances also depends on time, and it determines the sequence of substances being loaded into the system, such as for example, from the leaves to the roots.

VFL3 Its effect on the liquid assures the removal of substances in FAPA which are redundant. This redundancy is due to the fact that they have already been copied.

VALU It presents no obstacle to those substances which correspond to variables in the AHR system.

NOIB Does not affect those substances which correspond to nodes with nane $\neq 0$, and with a pointer to a father not equal to IBID (both conditions should hold).

NAZE Does not affect nodes with nane = 0.

FYES and FNOT. Does not affect substances corresponding to nodes which have fathers that do and do not exist in the grill, respectively.

PURE It stops everything except liquid.

In the figure, there are only two types of reactors, which transform one substance to another.

CAJA It produces substances which represent either nodes or values, that is, final or intermediate results.

MIXE Guarantees the transformation of additional necessary substances into pure liquid. The reason for MIXE is to compensate for the initial substances which disappear after reactions, and for assuring there is a new amount of substance in the "not-copied-yet" (NOCO) tank. MIXE is also necessary because of the impossibility of substance duplication.

VALV This valve prevents incorrect flow of liquid. It does not affect anything in the system; it ensures unidirectional flow (from top to bottom), preventing the liquid pumped by PMP2 to enter tank NOCO.

COCK This spigot is necessary to stop or start liquid and substances flow.

Before simulation, all tanks are empty. Then the user fills his tank, USEM, with liquid and a proper mixture of substances. Then he opens the cock

and the liquid flows and fills FAPA and NOCO. The tank FAPA holds the liquid. The liquid from NOCO flows to "cajas", and in those "cajas", reaction begins. The substances will stay in cajas until they become variable-substance or nodes with nane ≠ 0 and pointer to the father ≠ IBID. The products of this reaction drop through filters FYES and NOIB to the tank PARA (which simulates the grill) or through the filter FNOT to the FIFO. If FIFO is not empty, the pump PMP2 will rise substances back to the "cajas." At the end of the simulation, the tanks USEM and NOCO become empty and all of the substances in the grill PARA become replaced by variables-substance. This is the end of the reaction.

This analogy, as almost any one, is not perfect. But it seems to us rather close to reality, and very flexible. It can be used as a background for future simulations and, in particular, it can help to reach proper balance between different parts of the AHR computer.

POSSIBILITIES ARISING FROM DATA CIRCULATION IMPROVEMENT

The data path innovations that have been proposed in this section may allow the following possibilities:

1. A very big effective size of passive memory. This is possible because we can use a hierarchy of memories of different speed (such as fast ram; disks) to help passive memory --more to the point, they "become part" of passive memory (they enlarge it)--. We already mentioned that this kind of virtual memory is especially effective in AHR because prediction of the nodes that need to be evaluated next (and, therefore, need to be brought to fast passive memory) is straightforward.

2. Reducing the grill and fifo. In this case, most of the nodes residing in these memories will be nodes that are close to be evaluated.

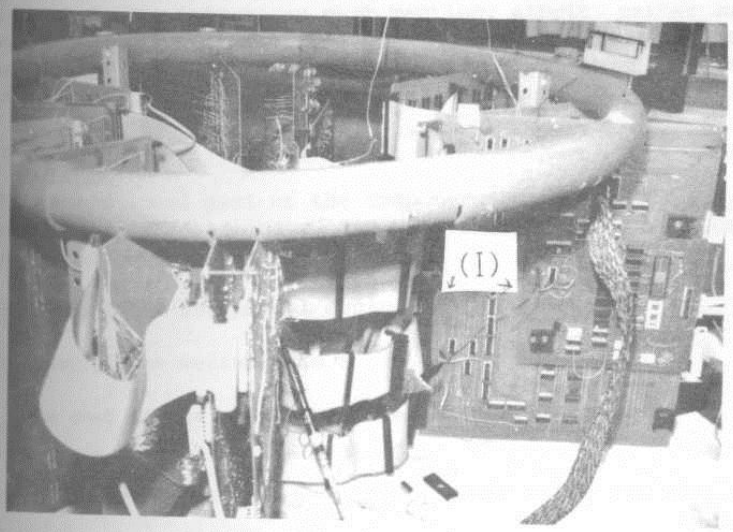
3. Synchronization with a real clock or external events. To achieve synchronization, we can use the "IBID" father and replace it by a real name of a father if a time value or external event satisfies a given condition.
   This is "busy waiting." All other types of waiting: hardware waiting (interrupts), monitor waiting, semaphores, can be performed by using specialized dummy "waiting nodes," which entails only periodic checking of the predicates. The process of "busy waiting" actually uses the universal caja as if it were specialized hardware. Therefore, when such use is constant, there are cases in which it seems profitable to replace the universal "caja" by simpler hardware which does the waiting.

   "Busy waiting" cajas or similar hardware can also be used as "daemons" for the synchronization of processes, including application processes, as is described in the Thesis of Rafael Domínguez "Daemons in the administration of a computer center", UPIICSA-IPN, 1980. (In Spanish).

4. Execution of iterative and recursive processes. As explained above, this can be achieved by using the "IBID" pointer.

COMMUNICATING THROUGH THE LOW-SPEED BUS

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



ONE LISP PROCESSOR

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

GENERALIZATIONS OF THE LISP MACHINE (AHR) FOR OTHER SYSTEMS AND RETURN TO RECONFIGURABILITY

The AHR architecture permits the execution of programs written in languages other than Lisp with very little changes in hardware. In fact, to guarantee this, we need only to replace one string in program memory by another. For example, in a PC, a program string can first be devoted to Lisp primitives but later it can be replaced by a string devoted to a specialized Lisp function.

As a second example, we can imagine the caja performing the sequential execution of a Fortran program, or any other type of non-applicative program, after its program memory has been reloaded with a Fortran interpreter. Incidentally, the reloading of PC program memory may be thought of as the execution of a generalized node with a non-existant dummy father.

The capability of loading and reloading the program memory is already part of the current AHR design. In this way, by mainly changing the software, we can achieve reconfigurability and the execution of programs other than those of Lisp. Naturally, we reduce parallelism in computations by executing code of non-applicative languages, but sometimes this may be better than to rewrite complicated old programs.

COMMERCIAL PRODUCTION OF PARALLEL COMPUTERS

At the end of this report, we must mention, albeit rather superficially, something about the future technological alternatives for mass producing parallel computers based on the AHR concepts. However, first we should review some simple but basic and essential properties of parallel computers.

Let us assume that:

1. The computational part of the computer costs $C \cdot N$, where C is the cost of the microprocessor, and N is the number of such processors in the system.

2. The cost of the peripheral equipment is P.

3. The cost of the software is S.

Consequently, the cost of the computer system will be:

$$L = P + C \cdot N + S/n$$

where n is the number of computer systems produced using the same software.

Computing speed of N parallel computers is always less than $V \cdot N$, where V is the speed of a single computer. In AHR, its speed is better than $V \cdot N^{0.5}$, as we can see in AHR-79-5, pages 25-26, for programs which possess enough parallelibility.

Let us assume first that this speed is $V \cdot N^{0.5}$. This means that a reasonable price for parallel computer is

$$R = (P + C) \cdot N^{0.5}$$

Profit gain conditions may be written as

$$\frac{(P + C \cdot N + S/n)}{(P + C) \cdot N^{0.5}} \leq A$$

where A is a given constant $0 \leq A < 1$.

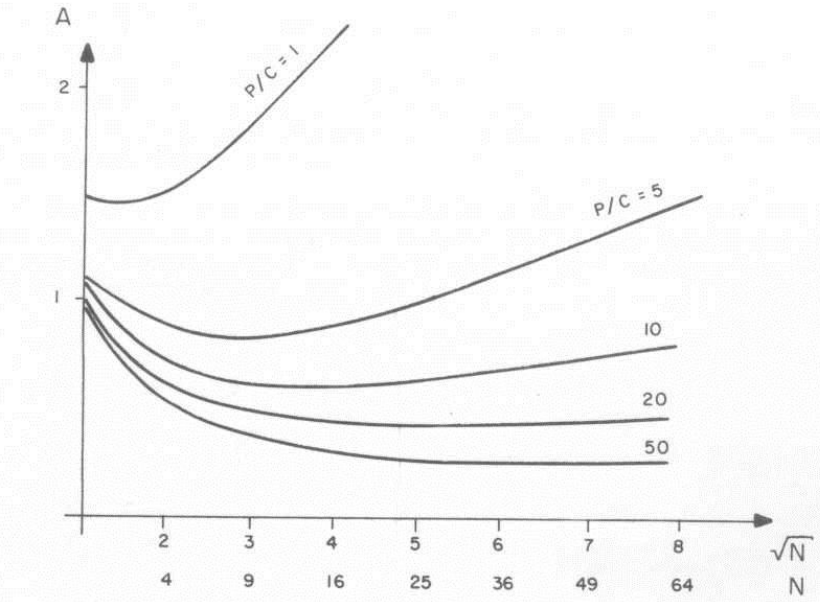
In the case that all of the components are imported, (1-A) determines those manufactured computers which can be distributed in the country where they are produced. A determines that portion of the production bulk which must be exported, to assure non-negative import/export balance. This non-negative balance is important in the case of Mexico because Mexican regulations for minicomputer manufacturers establish that a computer company that sells computers in Mexico should have imports whichincure preferably less than or equal to (but never more than) the costs of the exports. From manipulating the formula mentioned above, it seems appropriate that the value of A be about 0.5 Naturally, if A is near 1, most of the computers must be exported, thus obtaining only income from the computer production. If A is near 0, then the situation is very favorable because all of the computers will remain in the producing country. However, this value of A is very difficult to reach, as it will become clear later.

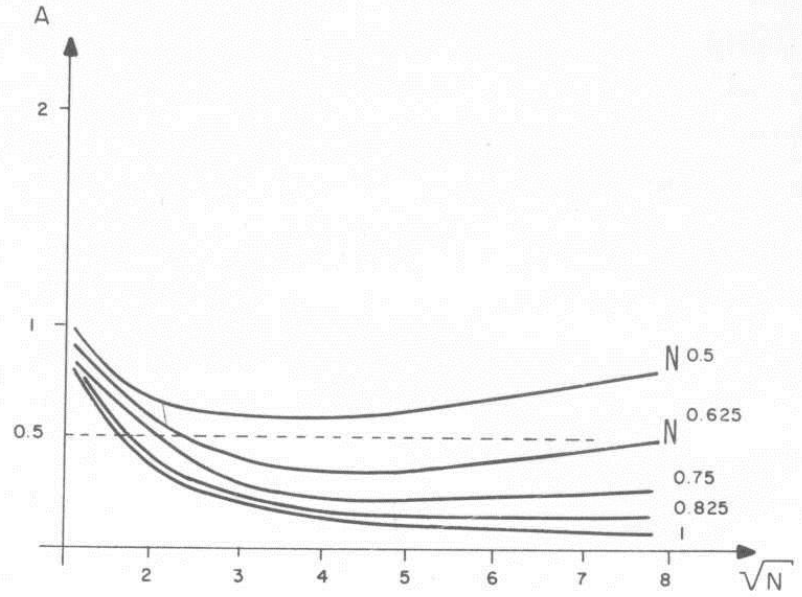
In the meantime, let us consider the following analysis. Taking into account the relation P/C, and if we assume that the software contribution for one computer is also the cost C (in reality, it will be more), we can calculat- the set of curves which are shown in the figure "RELATION BETWEEN THE NUMBER OF PROCESSORS AND A".

The set at the top of this figure is calculated for P/C = 1,5,10,20,50. From this set, it is clear that parallel computers made from imported components may be appropriate for mass production in positive trade balance countries, only in situations where parallelization is associated with relatively expensive peripherals. This gives us the key to being able to determine more exactly the efficient applications.

In addition, we may note that we can assume S/n = C. If software becomes more expensive, the competitiveness of parallel computers goes down. For this reason, we may assume that the competitiveness of AHR-like computers will be better than other ones because of the simplicity of the AHR software. To underscore this fact, it should be mentioned that it took two people six months to finish the software of version One of AHR. Another factor which assures the competitiveness of AHR is that its speed is in fact better than $V \cdot N^{0.5}$.

The bottom set of curves in the above mentioned figure illustrates the influence of the speed on A. The highest curve in the set is for $N^{0.5}$ dependence. Nearest to the $\sqrt{N}$ axis is for N dependence. Intermediate ones are for $N^{0.625}$,

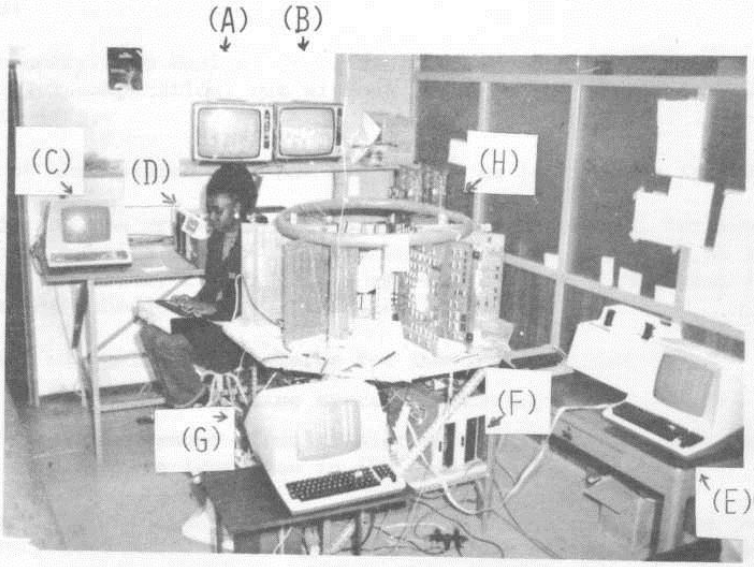




FIGURES 8, 9 "RELATION BETWEEN NUMBER OF PROCESSORS AND RESULTING COMPETITIVENESS UNDER DIFFERENT CONDITIONS."

$N^{0.75}$ and $N^{0.825}$ respectively. It is very important to understand that the increase of the power from 0.5 to 0.625 also increases the competitiveness significantly, thus making A less than 0.5 for an entire range of systems including simple ones in which P/C = 10. But for AHR, as it can be seen in the report AHR-79-5, page 26, even if high parallel instructions are 40%, the power is almost equal to 0.625.

Naturally, what we have presented is not comprehensive and does not provide an unequivocal answer with respect to the best policy for mass producing AHR systems. However, it does seem to provide good guidance in estimating a policy for the future.



OVERVIEW OF THE AHR MACHINE

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

(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.

CONCLUSIONS AND RECOMMENDATIONS

CONCLUSIONS:

1. For a country like Mexico, it is fruitful to continue research in the field of parallel computation, but without losing sight of its practical applications and spin-offs.

2. We are of the opinion that AHR ideas will be in wide use in the future.

3. Built during the project stage which has now concluded, Version 1 of AHR shows the practicality of a multiprocessor system based on an applicative language. However, it seems that non-applicative (iterative) languages can also be embedded naturally in the AHR design.

4. Two areas of applications emerge for immediate concentration:

   a. Distributed computing systems operating under AHR data exchange rules.

   b. The improvement of multi-microprocessor system performance by using applicative languages in its design and programming.

5. The degree to which hardware in AHR takes charge of resource management seems to be very reasonable. We expect that in the future that hardware will assume more system administrative and data flow control chores than are commonly done by present hardware systems.

6. The design of AHR seems to be very flexible, because many important improvements in its performance can be achieved by very small changes.

RECOMMENDATIONS:

We of the AHR project recommend the following:

1. To investigate the applications of AHR ideas to parallel computation, for SIMD and MIMD computers.

2. To do the preliminary design of distributed systems. This entails specifying in detail the interconnection of computational elements, the data flow paths, and data circulation, but not, however, going into the internal design of each element. In accordance with this report, each part of the system (smart memories, personal computers, "acopladores" (couplers), etc.) should be useful by themselves, and preferably should be stand-alone.

3. To evaluate the results of (1) and (2), and determine the technological policy for this area, both by hand calculation and by computer simulation.

4. To investigate the current state of the national industry with respect to its capability in mass producing parallel computer systems. Then, add a set of recommendations as to how this state can be improved.

5. In carrying out this project, the Institute (IIMAS) should follow the recommendations given in Appendix 1 of this report.

APPENDIX I- HOW TO ORGANIZE COMPLICATED PROJECTS IN THE FUTURE.

To us, the following list of rules seems to be very useful for assuring the successful completion of a project which has, as an important part, the design and construction of a complicated piece of hardware:

1-The goals of the project should be defined exactly, including the exact definition of the "width-depth" balace. (to be explained)

2-The variety of problems to be solved in the project should be limited, but in accordance to (1).

3-A project leader must have a reputation for reliably producing new ideas, for finishing what he has started, and for motivating people under and around him to work hard but without negative psychological effects.

4-A project leader must have an administrative assistant who is in charge of:
a-carrying out administrative chores.
b-purchasing equipment.
c-providing visitors with information about the project.
d-controlling the project administration.

5-A project leader should not be working on other projects. And in so far as possible, he should not be involved in other adminstrative or scientific organizational functions.

6-A project leader must have the ability to motivate the most useful members of the project. He must also have the power to remove those members he considers unproductive in the project.

7-By working very hard himself, the project leader demonstrates to the other members how to work. But he must understand that, on the average, most of the members of his group will tend to work a little less than he will.

8-It is obligatory that the project leader retain control on the work in progress on a weekly basis. It is bad for him to delegate this control to the administrative assistant. The control must be carried out in a firm, but tactful manner. This may even improve the psychological climate of the project.

9-The actual number of members of the project must be 10 to 20 percent greater than is needed to meet unforseen problems.

10-All of the vital needs of the members of the project must be satisfied no later than six months after the commencement of the project. As is defined in the field of scientific

management, vital needs are those necessities such as a good salary, job stability, an office, good working conditions, etc, which are essential to correct performing of a human being.

11-Increases in salary and other incentives during the project must correspond to the contribution of the members of the project, judged among same level people

12-The project must enforce an "early-death" policy. This means that, from the beginning of the project, the work load must be heavy. In this manner, the "weak" members will withdraw from the project early on, and the project leader will therefore have time to find the appropriate replacements. Also, the remaining members will get accostomed to the high standards of productivity.

13-Exceptional attention must be paid to the adminstrative aspects of the design process, including the design sequence and the design rules.

14-At the beginning of the project, the man-month requirements for carrying out the project (hardware and software) must be precalculated. These requirements depend on the forcasted complexity of the project at hand, which may be measured using the probable number of chips in the design. The following formula seems to be applicable:

$$M = K\ N^{\alpha}$$

where N is the number of chips, K and α are constants, and M is the man-month requirement.

15-First of all, there must be a master design file. All final decisions and changes in the design must be recorded in this file. Secondly, all of the members of the project must carry out their work according to the specifications in the master file, and must abstain from making changes in the design without authorization and without documenting them in the master file.

16-The first step in the design process must be an exact specification of the machine to be developed. The final version of the specification may filed in many different places, but it is obligatory that it be always in the master file. The "master file version" of the design is to be considered always correct.

17-The next step must be the choice of the technology. There must be a written report on the analysis of the possible alternatives, as well as, the final decision that was taken. This report must also reside in the master file.

18-Next, the design is decomposed into cabinets, units, pc boards, and standard design solutions. The description of the decomposition must be in the master file, together with the specifications of the standard design solutions. All members of the project must apply these standard design solutions as much as possible. However, if this cannot be done, they are encouraged to propose additions to the specifications of the standard design solutions. And only if this is not possible, then they have the right to invent new non-standard solutions.

19-What has been described above with regard to hardware can also be applied to software: each member of the project is encouraged,as much as possible, to use already existant standard subroutines. However, later, he may propose additions or alterations to these subroutines if they are warranted. But only as a last resort, he may write a subroutine from scratch. In either case, he must file the specifications of the subroutines in the master file.

20-Next, the interconnection design between the pc boards, and cabinets is worked out. Also, the input-output parameters are specified. The final version of this effort must be in the master file.

21-Only after finishing steps 15 to 20, can the team proceed with the final design of the pc boards and units. It should be noted that the preliminary design (step 14) was carried out but without using a master file. Moreover, this preliminary design can come in useful while carrying out steps 15 through 20. However, in the exact design of the hardware, project members must posses exact information on the availability of components which they intend to specify in the design of the system. This may require being in direct contact with suppliers or using databases containing information as to the where abouts of the products required.

22-Upon acquiring pc cards and other components, the project must perform tests on them so as to assertain their quality.

a-some components should be tested individually, others can be approved by testing a sampling in a batch, others under a variety of temperatures, etc.

b-it is advisable to buy chips from known manufactures, such as, National Semiconductor, Texas Instruments, etc. In this case, the need for (a) dissappears.

c-if the chip to be used is new to the designer, he must become familiar with its characteristics by carefully reading the technical sheets supplied by the manufacturer, and possibly by developing some designs which use it.

The methods for checking completely the componets must also be specified at this stage.

23-Software must be developed before hardware. It is very useful to simulate hardware components or subsystems with software in a way that permits the gradual replacement of different subroutines with hardware. For example, in the AHR project, we first wrote a software version of the distributor, and later replaced it by a hardware distributor with no difficulty.

24-After each "big" project, the project leader is responsible for writing or editing the rules which we have just described. We write "big" because we are sure that changing the rules too frequently is bad. In any case, this practice will permit the accumulation of recorded experiences in the participation of large projects. A final remark: one must always
    a-finish a project (we do not believe in eternal projects).
    b-write a report on the entire project.