REPORT

K.Berkling

Visit to ICOT

from Jan.20, 83 to Feb.18, 83

A visit to Icot was first mentioned by Dr. Uchida and Dr. Onai, whom I met in Bristol, UK, at the Summer School on VLSI Architectures. Later, a formal invitation by Dr. Fuchi was accepted by GMD.

I was asked to give three lectures about the reduction machine, its language, and its underlying concepts. Also I gave a public lecture entitled "Functional Languages" for a mixed audience of researchers from manufacturers, research institutions, and universities.

Compared to the almost total involvement in the language PROLOG, which I observed in ICOT as well as other institutions, I was surprised that there is still interest in functional languages. On the other hand, however, the reduction concept is of fundamental importance for PROLOG implementations.

During my visit to ICOT, I had the opportunity to meet Prof. A. Robinson and Dr. D. Warren. Many interesting discussions involving ICOT researchers were conducted about subjects concerning the processing techniques for PROLOG programs, concepts of implementing them in hardware, and the connections of functional programming (LISP) and logic programming (LOGLISP).

The first week at ICOT consisted mainly in attending

"Mini-Workshops" on subjects of ongoing research in ICOT. The internal lectures of Prof. Robinson, of Dr. Warren, and my own lectures were mixed into this series of mini-workshops. A high rate of information turnover resulted. I was slightly disadvantaged in this turnover, since my knowledge of PROLOG and logic programming was very new and not very deep. In time, however, with the help of many discussions I managed to get on top.

The idea of these mini-workshops is a very good one to acquaint visiting scientists with the current research. The tight schedule puts, however, a lot of strain on the power of comprehension of the participants, at least on mine. Thus, in the following, comments and judgements vary very much with respect to detail, depth and amount. These variations are merely subjective, and no objective valuation should be construed from these variations.

Inference Machines: SIM and PIM

First, I will make some general remarks on the problem. Logic programming is about set generation. The enumeration of solutions is accomplished by an "exhaustive treewalk" of a tree which is generated and retracted during this process. Leaves encountered are labelled "success" or "failure" depending on being non-contradictory or not. In this context the notions of depth-first and breadth-first denote algorithms to walk the tree. Generally, depth-first is easier to implement, however, there is always the possibility of non-termination, even before any "success" leaf is encountered.

Recent advances in hardware permit the introduction of

"parallel" enumeration algorithms. In this context, the word "parallel" should be avoided and replaced by the word "concurrent", since the "treewalk" is now performed by many agents, which interact rather substantially. PROLOG implementations on a conventional von Neumann machine are sequential of necessity. Thus, single processor PROLOG machines are also sequential. It is natural, that the SIM-project follows rather closely well-known software implementations, that is, the general scheme of representation is retained, while special instructions are provided to support unification und resolution. This is very much the approach taken in the commercially available LISP-machines. They also do not deviate from the basic list-representation of functions and data, their interpretation and compilation. Execution takes place on a specialized von Neumann type processor.

This approach is quite reasonable for experimental purposes, it might, however, preclude the development of more innovative architectues, which might be dictated by the use of VLSI technology. On the other hand, it is certainly of utmost importance to have a complete system available at the end of the initial stage of the FGCS-project. From my insight into ongoing reasearch in ICOT I have doubts that an operational SIM will be ready at the end of 1983. The same might be true for a PIM at the end of 1984. The schedule is just too tight. I would be worried that in the rush of development not all possible implementation varieties are explored and too many concepts from software implementations of PROLOG are simply "hardened", that is, they are taken from software to firmware.

The attack of the whole programming environment right from the beginning is to be applauded. Very often in the

past hardware and software were developed separately by people without any mutual understanding, moreover, the whole software problem became obvious only after the completion of the hardware design. Many efforts to implement complicated instructions were in vain, since these instructions turned out to be either unusable or superfluous. In the light of these thoughts, it might be more advantageous to simulate the whole system, that is the inference machine, the database interface, and the programming environment, on an existing computer of sufficient performance. But I might underestimate the capability of Japanese Manufacturers to turn out experimental machines very fast.

Two multiprocessor inference (PROLOG) machines are developed in the first laboratory of ICOT. One is based on graph-reduction, the other is based on dataflow. The design considerations for the latter one seem to be further ahead. The one based on graph-reduction seems to employ to a large degree data flow concepts, too. I was asked to look more deeply into the proposal for the graph-reduction inference machine.

I received as material to work on the "Proposal of PROLOG Machine based on Reduction Mechanism" by R. Onai, H. Shimizu, N. Ito, and K. Masuda. The proposal considers OR-parallel execution and AND-parallel execution. The decision to process the AND-connected literals in sequences is most probably a good decision, since the communication between processes processing AND-connected literals might very easily become unsurmountable in real applications (See also my remarks on AND-parallelism below). I cannot agree with the definition of reduction on Page 3 of the proposal (At this moment, I cannot determine to what degree this dis-

agreement is also one with P. C. Treleaven.) To point (1): Data driven and demand driven exists always together, but one or the other might be more implicit. Scanning an expression for a possible reduction is an implicit "demand" operation, returning results is an explicit driving by data. This is the case for string and graph reduction. To point (2): This statement is true, all reduction systems actually do subsume iteration under recursion. To point (3): Yes, this is no point of discussion. To point (4): Graph manipulation avoids copying. As stated on page 3 one could think that cause and effect is reversed. I would define "reduction" as selfmodifying code as opposed to reentrant code. This covers string and graph reduction. The paper proposes several AND-reduction units (ARU) and several OR-reduction units (ORU). However, a clause body is processed in a single AND-reduction unit. Thus, clause bodies are stored in the ARU's with pointers to clause heads (of other clauses to be invoked later) in the ORU's. Along these pointers match and unification request migrate from one ARU to several ORU's in sequence, since the AND-connected clause is processed in sequence. There is a distribution network one to many from ARU's to ORU's. As first presented to me, there was only a one to one connection back from every ORU to an associated ARU. I pointed out, that this restricts the loading of the machine too much and suggested to provide in this direction also a one-to-many network.

It is a major problem to distribute clause information to the ARU's and ORU's to achieve a good load distribution.

We proceed now to describe the concurrent execution of a PROLOG program in such an architecture. Concurrency is possible when several ARU's in an involved problem are

processing a clause body each in sequence and thus keeping all ORU's busy, provided that a requested clause head process finds a free ORU loaded with the corresponding clause head information. There was a discussion about the location of clause information in the units. I do not recommend a central location since the resulting communication caused by all requests for processing might be prohibitive. However, a dynamic loading scheme may be viable. Given an idle ORU and a requesting ARU in a wait state, because the ORU does not have the corresponding clause head information, they might as well use the time to load it dynamically. The selfmodifying code, that is the process representation, is currently stored in the PM's of the ARU's. PM means packet memory. This seems to me the weakest point of the proposal. First of all, I would recommend to separate two issues, the representation of the process state and packets modifying it. Secondly, I would recommend to keep the process state representation in a central place, sending and receiving messages (packets). This introduces some central control, but facilitates the coordination. Finally, all clause and fact information will be stored originally in a central place. One might as well rely completely on dynamic loading, that is a unit requests that what it needs when it needs it. Some rigid scheme like body in ARU(k) and heads of clauses to be invoked later in ORU(k+1), ORU(k+2) ... a.s.o. might work. Full concurrency is only available with unlimited resources. The solution of a large problem with limited resources permits only limited concurrency. Thus, the depth-first method still has to be used, but in terms of larger subtrees. I regret that I could not get involved in more detail like working out all formats, determining the control-information need, a.s.o. Simulation should start as soon as possible to expose some

loose ends which cannot be solved in the present state of development.

Remarks on Parallelism

The following observations have to be weighted by the fact that the author is a novice with respect to use and implementation of PROLOG. In handouts and in presentations three sources of potentially concurrent processing are given: OR-parallelism, AND-parallelism, and parallelism of arguments. According to my insights (qualified by the above remark) these types of concurrency cannot be mentioned on even terms. PROLOG programming is backwards in some sense, the result of a computation is specified first, and the execution of a PROLOG program produces all the data which resolve the goal. The so-called OR-parallelism, is operational. It describes the possibilities to be investigated in finding a solution.

The AND-connective involves a certain kind of binding structures which one might describe as the reverse of the lambda calculus binding structure. In the lambda calculus a distribution structure from one location in an expression to many places is the key operation called beta-reduction. In a clause the reverse is to be enforced, namely a collection from many places in the clause body to one place in the clause head. Obviously, concurrent processing of the clause body cannot start before the reverse binding structure is established.

It is a technical problem to achieve the many-to-one binding structure in a clause. It should not show up in the

language. According to my understanding, "concurrent PROLOG" is doing just this. Concurrent unification of literals entails intensive communication between corresponding processes. To introduce the concept of channels (read-only annotation) to variables seems to me connterintuitive to the concept of referential transparency and to the concept of variables.

Functional Programming and Logic Programming

In many discussions one point came up over and over again, namely the relation between functional and logic programming. The shift of attention from procedural languages to functional languages, and later to logic languages, reflects the concern over the predominance of issues which do not directly contribute to problem solving in the production of application software. Generally, von Neumann software is overloaded with solving the problems of the von Neumann computer architecture. While procedural languages require to a large degree specifications "how" to get to a solution, functional languages allow the specifications of one solution and emphasize the "why". Functional programs are not executed in the sense of program execution. Functional programs really denote a solution, which is then reduced, that is transformed or rewritten in a form of reduced complexity. As an example consider ((7 + 8) > (6 * 9)) = true. This implies that a representation of the solution has to be known beforehand. All functional program examples exhibit this "denotation of the solution" property. One might ask, whether some sort of problem solving takes place at all in in the rewriting process. In contrast, this looks like procedural languages

do really prescribe some work. This is true, but the work done is mostly not concerned with problem-solving issues. In contrast to functional programming, logic programming is not concerned with specifying one solution. A logic program is a question, a goal, and produces all data consistent with that goal. This includes the case where no data exist which resolve the goal.

Logic programming represents currently the highest degree of abstraction from procedural details. Since the latter do not disappear, the requirements on automatic transformations of goal specifications, facts, and rules into procedural detail are large, and compromises between abstract specifications and procedural hints seem unavoidable. Finding and producing the set of consistent solutions is combinatorially explosive. At this point of the discussion, I would like to draw the attention to what one might call "the single language syndrom". In spite of people getting enthused with some language, there is no single language sufficient and practical for all applications. Not only are the problem oriented concepts of different application areas too different to have them all efficiently supported in one language, but there is also a level structure of languages existing of necessity. One needs an orderly descent from a very abstract language expressed in terms of a lower language level and so on until the level of flip-flops and gates is reached. This latter level is again an abstraction of even more primitive elements. Also, any program in terms of one language is object of a higher level language dealing with the logistics of bringing in programs and getting out data. Thus, procedural languages and functional languages provide a more direct and straight forward way to obtain a solution, where the algorithm is known or

the solution can be specified, respectively. There is no need to search for something which is already known. A typical example is the use of "cut", "not", and "fail" to steer the evaluation process in PROLOG. Here, knowledge about details of the implementation is necessary to achieve effects which are more directly accessible in other languages.

Comments to visits to universities and manufacturers.

Courtesy of ICOT the following visits were arranged: Kyoto University, University of Tokyo, ETL, NEC and ECL. All visits were well organized, the presentations well prepared, and written material was always available. The latter is particularly important, since the impressions of a visit have to be digested later again. I would like to summarize my impressions in three points. 1) There seems to be a policy to make the best and latest equipment available to researchers in sufficient amounts, with some differences, of course. 2) There seems to be a general recognition of the fact, that architectural concepts have to be tried by real engineering or else significant deficiencies of the concept will not be exposed and detected. Researchers obviously get sufficient support by engineering departments 3) The discussions arranged were very substantial due to the well-informed participants. Under current conditions of expensive travel, high postal rates combined with long delivery times the particular effort necessary is applandable.

Impressions and comments in ICOT

This was my first trip to Japan. Although I travelled a lot, this was a completely new experience. My deepest impression is the generally positive climate of working and living together. I cannot believe that this is due to being a guest and due to the language barrier which might conceal the truth.

All efforts were made to get settled in what seems at first to be a very foreign environment. It takes, however, only hours or a day to find out that this environment is quite livable.

The four weeks went by rather quickly. Except for preparing the lectures, I was more or less receptive since the whole area of logic programming is new to me. Occasional prior acquaintance to PROLOG by some lectures does not count. Attempts to get involved in ongoing research were dwarved by the schedule of talks and visits. To produce tangible results it is necessary to sit at the desk for a couple of weeks without obligation to lecture or to pay visits.

The working climate in ICOT seems to be determined by the synergetic effect of the "Fifth Generation Idea", combining several current trends in information and computer science. Originating from the severe language barrier, even between Japanese and computers, the drive for logic programming and PROLOG seem to be a must for natural language processing and translation, speech input and output, while relational database techniques seem to wait for inference machines to be real useful. Natural language access to data

bases seem to make applications possible hitherto unheard of. Natural language translation contributes to international understanding and commerce. Although I saw simular projects in many places in Japan, they all get their incentive and drive by a visible confluence point like ICOT. Many nations in the world will look at it with envy and admiration. The idea of combining current trends cannot be repeated elsewhere, it will be unique and will determine developments for decades.

Dr. Klaus Berkling

Eisenachstrasse 31
5205 St. Augustin, West-Germany
(02241) 332720

EDUCATION:	
1951 - 1957	Studied Physics at the Free University in West-Berlin and at the University of Bonn, West-Germany.
1957 - 1961	Ph.D. work at the University of Bonn, West-Germany.
DEGREES:	
1957	M.A. in Physics: Design of a Partial-Pressure Gauge, based on the three-dimensional massspectrometer of Prof. Paul.
1961	Ph.D. in Physics: Scattering of Gallium Atoms in different Zeemanstates on Atoms of Inert Gases.
EXPERIENCE:	
1961 - 9164	IBM Laboratories in Boeblingen, West-Germany.
1964 - 1972	IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA.
1971 - 1972	Guestlecturer for Computerscience at the University of Freiburg, West-Germany.
1972 -	Gesellschaft fuer Mathematik und Datenverarbeitung, Bonn-Birlinghoven, West-Germany.

1980 - 1981	Visiting Scholar, Stanford University, Stanford, CA 94305, Center for Integrated Systems, study of very large scale integration techniques.

WORKAREAS:

Higher Level Language Computer Architecture.
Computer Architecture and -Engineering.
Microprogramming.
Functional Programming Languages, their Theory, Application and Support by Hardware Architecture.

ADDITIONAL INFORMATION:

Glider Pilot.
Private Pilot.
Learned Machinist.