Summer 1988 at ICOT: Some Impressions and Preliminary Results

Andrzej Ciepielewski *SICS, Sweden*

Abstract

ICOT and SICS are two somewhat similar research organizations in two very different countries. One of the main research areas in both institutes is parallel execution of logic programming languages. I stayed at ICOT for three months working with subjects of common interest, and studying work done at the institute. The preliminary results presented in this report concern parallel implementation of OLDT resolution, and/or parallel execution of Prolog related languages, suitability of MIMD and SIMD architectures for logic programming languages, and finally architecture simulations.

1 Background

My visit to ICOT has been made possible thanks to mutual interest on many different levels. First comes my personal interest in Japan and in the research conducted at ICOT and elsewhere in Japan. That is closely followed by the common interests of ICOT and SICS, two somewhat similar institutes in two very different countries. The institutes agreed on exchanging researchers, or rather ICOT agreed to receive researchers from Sweden. Finally, there are surely some common interests at the national level, but of that I know very little.

The concrete plans had been setup by Koichi Furukawa and me, and approved by the respective directors. We easily found some research pursuits from which both ICOT and SICS could benefit. We agreed on the following subjects. (1) Investigation of suitability of or-parallel implementation techniques in the context of knowledge bases in general, and for implementing OLDT in particular. (2) Comparison of Andorra Prolog with ANDOR-II. (3) Cooperation with Evan Tick in his effort to compare cache behavior of the Aurora Prolog and the KL1 systems.

2 Overview

A three month visit to an exciting research institute in an enchanting country induces an extreme effort to pursue (and balance) what are essentially contradictory interests: professional and other. I will omit describing all of the excitement outside ICOT, but my advice is to come and see it, there is nothing like Tokyo at 9 p.m. (any evening), and there is already a lot of beautiful country side as soon as 1 hour from the downtown.

I wanted to carry out my outlined plan, but at the same time I wanted to learn as much as possible about all the different subjects being investigated at ICOT. The results of all the compromises are described below.

In Section 3, I present my work on knowledge bases which I carried out together with the members of the 3rd laboratory and Toshiba's Research and Development center. The first outcomes are a prototype parallel implementation of OLDT in Aurora Prolog (mostly by Y.Morita), a parallel scheduling algorithm which is an improvement over the multi-stage scheduling algorithm proposed by Tamaki [10], and an outline of extensions to the Aurora Prolog system necessary to implement OLDT efficiently. In Section 4, I discuss some language and implementation aspects of Andorra and ANDOR-II, and also give some more general comments on and-or parallel logic languages. In this part of my work I had many fruitful

exchanges with A.Takeuchi and K.Takahashi from Mitsubishi's Central Laboratory in Osaka. In Section 5, I compare ICOT's and SICS's approache to implementing logic languages on parallel computers, and also offer some thoughts on feasibility of using SIMD machines. This part of the work has been influenced by discussions with members of the 4th laboratory and M.Nilsson from Tanaka Laboratory at Tokyo University. In Section 6, I give a very short summary of my contributions towards instrumenting the Aurora system for cache simulations. Finally in Section 7, I mention some other events from my stay at ICOT.

3 Parallel OLDT

One of the main goals of ICOT is to design an integrated inference and knowledge base machine. Up to now two separate machines were ivestigated. One of the main differences in techniques used in the database area and in the problem solving area is the way queries are solved: in the DB area, mostly bottom up and in the problem solving area exclusively top-down. Another difference is the approach to search completeness. Prolog, for example, is not search complete, it is up to the programmer to write clauses and goals in a proper order, and to pose suitable queries so the program will terminate, e.g., when all solutions are required. Database systems must be search complete, at least for a class of programs (e.g. stratified programs), and the search completeness must not depend on the order of literals in a query or the ordering of rules and facts in the database. One attempt to solve the search completeness problem in a top-down manner is OLD resolution with tabulation (OLDT) [10], extended to OLDTNF to handle finite negation [7]. There are at least two other attempts to solve the problem in a similar way [2] [12]. The last one seems to be the most general of the three as it does not require that goals be chosen in a fixed order, which OLDT does. My aim was to investigate the feasibility of an or-parallel implementation of OLDT and OLDTNF.

In a sequence of meetings with Y.Morita, F.Itoh, and H.Monoi from the 3rd laboratory, researchers from Toshiba's R and D laboratory, and H.Yokota from Fujitsu, we discussed the database work done at ICOT and associated companies, and relational database techniques in general. We also discussed, extensively, or-parallel implementation techniques in general and the or-parallel Aurora Prolog system in particular. With that background we could identify the main implementation problems of OLDT as being: representation of terms for efficient subsumption testing, organization of OLDT tables for avoiding congestion on access by processors working on different branches, and creation of an efficient parallel scheduling algorithm to replace the multi-stage scheduling algorithm proposed by Tamaki.

H.Ito is investigating three methods for representing terms: superimposed code word techniques, TRI representation proposed by H.Yokota [16], and indexing with hashing as used in most Prolog implementations. Y.Morita is working on a pilot implementation of OLDT in Aurora Prolog. His implementation is quite realistic, it is expected to have fairly good speed and already shows good speed-ups (up to 8 times on 8 processors). Morita's implementation is one of the first extensive tests of Aurora's asynchronous predicates and of the "save area" facility [5]. I am working on a parallel scheduling algorithm, and on integrating OLDT resolution in the Aurora parallel Prolog system.

Tamaki's multi-stage scheduling algorithm is sequential. A computation proceeds in stages. In each stage OLD extensions and lookup reductions are executed in order until no more operations are possible. A new stage starts if the previous one has produced some new solutions. Tamaki's algorithms can be trivially parallelized by removing the ordering restriction for execution within a stage. The reduction within a stage can be done in any order or in parallel. The algorithm remains correct as long as the division into stages is kept. Such simple parallelization is not satisfying because it restricts parallelism unnecessarily. The algorithm is based on the worst case assumption, namely that all subtrees rooted in different solution nodes are interdependent (a lookup node in one tree is suspended on a solution node in another subtree). I am working on a less restrictive algorithm, a local multi-stage scheduling algorithm, based on the following observations: (1) The solution list of a solution node can only be expanded by the branches

below this node in the OLDT tree; (2) The solution list of a solution node cannot be expanded in the current stage (global stage as defined earlier) if every branch below this node is suspended either on this solution node or on its descendant. The main idea of the algorithm is to divide the computation into many local stages and allow as many stages as possible to be executed simultaneously. A local stage consists of reductions and extensions below a solution node. Local stages can be nested and interdependent. The algorithm determines when the stages are disjoint, and thus independent.

Morita's parallel interpreter implements a parallel version of Tamaki's algorithm. He is also trying to implement an approximation to the algorithm proposed by me. It seems fairly hard because it requires knowledge of the structure of the suspended part of the OLDT tree. Another inefficiency in the pilot implementation is the handling of suspension. The state of a suspended computation is copied to the internal database on suspension and copied back to the runtime area (code and heap) on activation. By extending the Aurora system the efficient scheduling algorithm can be implemented and copying of suspended branches avoided. Implementation of the algorithm is facilitated by the fact that even the suspended branches are kept in memory, and thus the structure of the whole tree is available. Copying can be avoided because in the Aurora system information about about conditional bindings [13] is available (from trails) even for suspended branches.

The pilot implementation is currently being used to estabkish the raw speed of the interpreter, and get an idea of the time consumed by different operations like subsumption testing and suspension. The speed is compared to the speed of a Prolog system, on feasible examples, and also to the speed of the sequential OLDT interpreter from which the parallel one originates. The speed-ups are being measured for different programs. When the investigation is completed we will try to answer the following questions: Is the implementation in Aurora a useful tool in itself; How much is gained by parallelism and what has to be done to extract it; Is it worth-while to extend Aurora with local multi-stage scheduling? A question which cannot be answered by this investigation is how subsumption testing can be improved.

4 Andorra and ANDOR-II

Programs written in logic programming languages have potential for both AND- and OR- parallelism. Because of the difficulties in combining the two, most of the systems designed so far utilize just one form of parallelism. Or-parallel Prolog (e.g. Aurora) allows parallel search for alternative solutions while keeping execution of goals sequential. This impairs efficiency, but does not limit the expressive power of Prolog. Committed choice languages allow parallel exploration of goals, but commit to one choice in each predicate, making programming of many problems inconvenient if not impossible. Two languages have been proposed recently, Andorra Prolog and ANDOR-II. Their main objectives are different.
The initial objective of the Andorra model [14, 15], was to allow and/or execution of Prolog, but it is now considered as a base for some other languages. One possibility is Andorra Prolog [4], a language intended to subsume both Prolog and committed-choice languages. In addition to don't-know and don't-care non-determinism the language supports control of or-parallel split, synchronization on variables, and selection of clauses.

The main objective of ANDOR-II [8] is investigation of distributed problem solving, with problems coming from AI. The language inherits most of its properties from FGHC, extending it with non-determinate (all solutions) predicates. The method of realization is a general transformation of or-parallelism into and-parallelism, using colors to identify solutions coming from alternative branches.

I have rather informally compared two aspects of the above languages: expressiveness and execution models. In the comparison I do not simply use the Andorra model, but Andorra Prolog, a language built upon the model.

The two languages seem approximately expressive power, assuming that built-in predicates in Andorra suspend until their arguments are nonground. There is one exception. Andorra, as currently defined, suffers from unfair scheduling. Because of this many programs which designers of ANDOR-II have in

mind, e.g. programs defining systems of communicating non-determinate process, processes giving all solutions, will loop in Andorra, not producing any results. This is so because Andorra's scheduling always chooses the leftmost among suspended non-determinate goals. This scheduling principle is not inherent in Andorra Prolog. It has been adopted to get a program behavior (order of goal execution) as close to that of Prolog as possible. My opinion is that the principle of choosing the leftmost go can be replaced by fair scheduling without any great loss. The close affinity to Prolog will be lost anyway, because of the necessary suspension of built-in predicates. Andorra with fair scheduling would run the same programs as ANDOR-II, and the programs would be very similar.

In contrast to the expressive power, the computational models of the two languages are different in at least one important respect: the handling of multiple solutions. ANDOR-II computational model handles multiple solutions using "colors", Andorra uses restrictive copying.

ANDOR-II is implemented on top of FGHC, but this is not of primary importance, as an FGHC implementation could be extended to support colors efficiently. The important fact is that each process must potentially manage solutions from different worlds (branches in the search tree), because inconsistent solutions (belonging to different worlds) might "meet" in a process, but must not be combined. Consistency checking is the main source of overhead in the ANDOR-II model. A nice property of the model is that the same solution is not recomputed in different worlds, as it is in Prolog and in Andorra.

In Andorra different worlds do not interact. Execution of a non-determinate predicate prepares for the creation of independent worlds. A world is actually created by restrictive and delayed copying of conditional bindings using the binding array technique.

Summarizing: in ANDOR-II inconsistent solutions can be produced and as a result a consistency check is needed. In Andorra, in contrast, worlds are separated by restrictive copying and no inconsistencies arise. It is an open question which method is better on the average.

A.Takeuchi has made an interesting observation after studying S.Gregory's proposal to implement Andorra in Parlog [3]. One of the main sources of overhead in this proposal is the copying of goals. It seems that copying the goals could be made efficient using the binding array technique used in Andorra. That would make Andorra in Parlog a viable alternative to the other two.

While studying the scheduling problems in Andorra, two reflections occured to me, one concerning generalizing the Andorra model, and one on scheduling in KL/1.

(1) An important property of Andorra is that at most one non-determinate goal per branch is executed at any time. In this way conflicting bindings are avoided. Currently a non-determinate goal is chosen when all goals in the branch suspend. The goal could be chosen earlier, e.g. when the last goal is about to suspend, or even earlier. In the latter case all goals in a branch would have to be stopped while the non-determinate goal is executed, and then restarted in different worlds. Such flexibility makes possible the design of schedulers which could give different priorities to different goals depending e.g. on the availability of processors.

(2) I think that similar problems will have to be considerd in a FGHC implementation running ANDOR-II programs. Two types of processes, all-solutions process and ordinary and-processes, will have to be introduced and scheduled appropriately. This is especially true in view of the sensitivity to the choice of scheduling shown by the existing implementations [9].

5 FGHC on MIMD and SIMD

Both ICOT and the Gigalips project have the ultimate goal of developing machine architectures for the efficient utilization of parallelism in logic programming. The approaches of the two organizations are very different.

Researchers at ICOT have made an a priori decision that a MIMD architecture with local memories is an appropriate base for a parallel inference machine. Such an architecture has the very important advantage of being scalable, but it also has a crucial disadvantage in not efficiently supporting for global address space. This decision has influenced the form of the language used (committed-choice, one way head unification), and the pragmas (explicit control of scheduling). The results of the 4th laboratory are

very impressive, they have already managed to design a successful parallel machine that will be able to run many programs at excellent speed. At the same time, the simulation results show clearly that the machine is not quite satisfactory, because it does not allow transparency [9]. A program moved from a sequential machine to PIM will require extensive rewriting in order to run efficiently. This is due to the time taken by the remote accesses, and the resulting high sensitivity to scheduling decisions.

The consensus in the Gigalips project is that logic programming on a parallel computer should not be much different than on a sequential computer, and thus, that efficient support of a global address space is essential. The strategy is to extend the existing technology, and search for an architecture which would support the "dream language" efficiently, all the while keeping a check on the appropriateness of this direction of research. This approach is at the same time more cautious and more uncompromising. The difference in approach could be caused just by the 10 years limit hanging over ICOT, but there might be some deeper reasons.

There is an alternative to the directions of ICOT and the Gigalips project. Recently a very successful SIMD computer, the Connection Machine, has been constructed. It was intended for AI applications, but it seemed at first that it was most suitable for numerical computations. Now there is growing evidence that it is good for AI, after all. There is also some evidence that it suitable for committed-choice languages too. M.Nilsson [6] shows that massively parallel FGHC programs can be executed efficiently on the Connection Machine. The solution proposed by M.Nilsson is a parallel SIMD interpreter for FGHC (via FLENG). The parallel data structure evaluated by the interpreter is the list of goals. Nilsson's solution shows very good peak performance, i.e. speed with all processors busy. His results also reveal some serious weaknesses of the approach, at least on currently existing SIMD computers. Due to the one bit processors of the CM, Nilsson's interpreter performs very poorly when there is not enough parallelism in executed programs. This makes the system impractical except in very special cases. Most of the problems considered suitable to be solved using committed-choice languages do not have the amount of parallelism that would make a CM-like computer competitive as a general purpose computer.

It is an interesting question whether a 1000-processor SIMD with powerful 32 bit processors could be an alternative to a 1000 processor MIMD. The main problem is still the sequential performance. Nilsson's solution is a parallel interpreter. The interpreted speed is known to be an order of magnitude lower than the compiled speed on conventional uniprocessors. Does this rule of thumb still apply? If so, can a single SIMD processor be made 10 times faster than the corresponding MIMD processor? Other problems concern load distribution and garbage collection. Are there reasons to believe that these problems have more efficient solutions on SIMD architectures?

These are some open questions which must be answered in order to establish whether machines with SIMD architectures could be practical engines for logic programming.

6 Aurora Instrumentation

A non-trivial step in the design of a new architecture is simulation of the different aspects of the architecture. E.Tick is investigating cache behavior of Aurora and KL/1 systems [11] using a parallel cache simulator connected to current implementations on Sequent's Symmetry. During my stay at ICOT I applied my expertise concerning the internals of Aurora to make the system run together with Matsumoto's cache simulator, and to behave properly when instrumented. Some quite central changes to the system were required, probably the most essential being modification of the process management, new choice of idling times, and allocation of binding arrays in the shared memory. It seems that the instrumented system now behaves reasonably. In the course of this work I learned a lot about simulation, and discovered that it is indeed an art.

I am taking home with me Matsumoto's cache simulator, hoping that the work of Tick will be continued and extended at SICS.

As a spin-off from my work with Tick I have got some ideas about caches for Aurora, ideas that happen to coincide with the ideas of Tick, Hermenegildo, and Shen. In the current Aurora implementation, the memory of a processor is divided into private read/write memory, shared read-only memory, and

shared read/write memory, the last one being used for storing nodes of a search tree. The need for read/write shared memory can be eliminated if a scheduler using the "wave-front" data structure is used [1]. Caches for a system where only private memory is read/write would require a very simple cache coherency protocol. Namely, when a part of worker's memory is made shared, part of its cache must be either written back to memory or broadcast to other caches. The question is how much of the cache contents must be "made public".

7 Other Activities

During my stay at ICOT I gave two formal talks, visited 3 companies (Toshiba, Fujitsu, Mitsubishi) and talked to many ICOT members. Below I shortly summarize those activities.

7.1 Talks

The Aurora Or-Parallel Prolog System
Aurora is a prototype or-parallel implementation of the full Prolog language for shared-memory multiprocessors, developed as part of an informal research collaboration known as the "Gigalips Project". It currently runs on Sequent and Encore machines. It has been constructed by adapting Sicstus Prolog, an existing, portable, state-of-the-art, sequential Prolog system. The techniques for constructing a portable multiprocessor version follow those pioneered in a predecessor system, ANL-WAM. The SRI model was adopted as the means to extend the Sicstus Prolog engine for or-parallel operation. We describe the design and main implementation features of the current Aurora system, and present some preliminary experimental results. We conclude with our plans for the continued development of the system and an outline of future research directions.

The system is available at ICOT's Symmetry machines. I plan to demonstrate the system both directly after the talk and at some later time we can agree upon.

Andorra Prolog: its Computational Model, Applications and Implementation
The Andorra Model, invented by D.H.Warren and R.Young is a solution to the problem of exploiting dependent and-parallelism transparently in Prolog programs. The solution is to execute determinate goals in and-parallel, with the determinacy analysis being performed mainly at compilation time. The model extended with wait declarations is a base for Andorra Prolog, designed by S.Haridi, a coroutining language more powerful than Prolog. I will present the model and the language. Then I will briefly describe thh application of Andorra Prolog to distributed simulation, and outline an implementation of the model. Finally, I will try to compare Andorra Prolog with ANDOR-II, a language proposed by K.Takeuchi et al.

7.2 Visits

Toshiba:
I visited the Information System Laboratory at Toshiba's Research and Development Center. My host was Mr. M.Minami. The laboratory receives corporate, division and state funding. It consists of 5 groups working on advanced architectures, knowledge base software, man-machine interface (multimedia), speech understanding and vision. The results of the laboratory range from close to exploitation (like an attached Prolog processor) to the very speculative (like a parallel AI machine). The laboratory cooperates with ICOT in the knowledge base research. Among the results are a hardware accelerator for RBU (retrieval by unification), and a multiprocessor database machine Mu-X.

Fujitsu
I visited the AI laboratory in Kawasaki. My host was Y.Yokota. The laboratory consists of 3 groups working on basic AI, neural computing, and hardware. The laboratory cooperates with ICOT on building PIM. A Sequent Symmetry is used for experimentation in implementation techniques for GHC. Other

activities include construction of parallel hardware for image recognition and display; I was shown CAP (cellural array processor), a MIMD computer constructed at the laboratory, meant to be used e.g. for car interior design using high quality images; and robot control by neural networks; I was shown a neural network based system used to control a set of toy robots playing cops and robbers.

Mitsubishi
I visited the Central Research Laboratory in Osaka. My host was A.Takeuchi. This is one of several computer research laboratories supported by Mitsubishi. It is fairly small (slightly over 10 people), and fairly theoretical. The main themes are parallel problem solving, intelligent programming systems (verification, synthesis, analysis, algorithmic debugging) and expert systems. I spent most of my time discussing ANDOR-II and Andorra systems mentioned earlier in this report. I also had an opportunity to discuss the Pegasus chip, a one chip processor to be used in embedded systems, e.g. home appliances.

7.3 Presentations and Discussions

I was given a general introduction to ICOT by Dr.Iwata, an overview of the second laboratory by ???, and an overview of the third laboratory by Y.Morita. I talked with K.Yokota about Kappa, with K.Yoshida about A'UM, with H.Seki about OLDTNF, with Goto,Taki,Sato and Chikayama about PIM and multi PSI, and with K.Ueda about program transformations and constraint programming in FGHC. I also showed a simple integer constraint solver written by S.Haridi to K.Ueda and his colleagues, who are comparing it to a constraint solver written in FGHC. Finally I talked with M.Hermenegildo about compiler time analysis for RAP.

8 General Remarks

The keys to a successful visit are common interests, good organization, and openness on both sides. I think that in my case all the keys were given. I have benefited from the visit both by being exposed to new issues and by being forced to explain my work.

9 Acknowledgements

I am very much impressed by ICOT's procedures for welcoming guests and new coworkers. I have never felt so important in my life. I want to express my special gratitude to Dr.Iwata who helped organize my trip, and to Y.Morita who always had time to help me with my daily problems in and outside ICOT (e.g. when the air-conditioning started to flood my apartment).

References

[1] Per Brand. Wavefront scheduling. Internal Report, Gigalips Project, 1988.

[2] Extension Tables: Memo Relations in Logic Programming. In *Proceedings of the 1987 Symposium on Logic Programming*, San Francisco, 1987.

[3] Steve Gregory. Thoughts on the Relationship between David's Proposal and Parlog. In *Gigalips Workshop*, Manchester, 1988.

[4] Seif Haridi and Per Brand. Andorra Prolog – An Integration of Prolog and Committed Choice Languages. In *International Conference on Fifth Generation Computer Systems 1988*, ICOT, 1988.

[5] Ewing Lusk et al. The Aurora Or-parallel Prolog System. In *International Conference on Fifth Generation Computer Systems 1988*, ICOT, 1988.

[6] Martin Nilsson and Hidehiko Tanaka. Massively Parallel Implementation of Flat GHC on the Connection Machine. In *International Conference on Fifth Generation Computer Systems 1988*, ICOT, 1988.

[7] Hiroshita Seki and Hidenori Ito. A Query Evaluation Method for Stratified Programs under the extended CWA. In *Proceedings of the 1988 International Conference on Logic Programming*, Seattle, 1988.

[8] Akikazu Takeuchj, Kazuko Takahashi and Hiroyuki Shimizu. A Parallel Problem Solving Language for Concurrent Systems. In *Proceedings of IFIP WG 10.1*, 1988.

[9] Kazuo Taki. Measurements and Evaluation of the Multi-PSI/V1 System - A Study of Inter-PE Communication versus System Performance. ICOT.

[10] Hisao Tamaki and Taisuke Sato. OLD Resolution with Tabulation. In *Proceedings of the 1986 International Conference on Logic Programming*, London, 1986.

[11] Evan Tick. Performance of Parallel Logic Architectures. ICOT Technical Report TR-421, 1988.

[12] L.Vieille. A Database-complete Proof Procedure Based on SLD-resolution. In *Proceedings of the 1987 International Conference on Logic Programming*, Melbourne, 1987.

[13] David H. D. Warren. The SRI model for or-parallel execution of Prolog—abstract design and implementation issues. In *Proceedings of the 1987 Symposium on Logic Programming*, pages 92–102, 1987.

[14] David H. D. Warren and Rong Young. The Andorra Model and Its Implementation. In *Gigalips Workshop*, March, 1987.

[15] Rong Yang. Programming in Andorra-I. Internal Report, Gigalips Project, August 1988.

[16] Haruo Yokota et al. Knowledge Retrieval and Updating for Parallel Problem Solving. Fujitsu Limited.

Ciepielewski Andrzej

Researcher at the Laboratory of Logic Programming Systems
SICS Swedish Institute of Computer Science

Home Address
Renstiernas Gata 31, II 11631Stockholm, Sweden, Phone +46 8 417344

Academic Degrees
. Ph.D. in Computer Science from the Royal Institute of Technology (KTH) Stockholm, 1984.
. Docent, in Computer Science from the Royal Institute of Technology (KTH) Stockholm, 1988.

Employment
. Current employement is stated above, Dec./85.
. Extra university lecturer Oct./84 - Nov.85 at KTH.
. Research Engineer at KTH, department of computer science, July/80 - Sept./84.
. Assistent at KTH, department of computer science, July/76 - June/80.
. Programmer at Institute of Applied Mathematics (ITM), Jan./78 - June/78
. Programmer at Mydata AB, June/75 - Dec./77.

Referee for:
. Journal of Logic Programming
. The International Journal of Parallel Programming
. International Conference in Logic Programming, Melbourne 87, Seatle 88
. IEEE Symposium on Logic Programming, San Francisco -87, Salt Lake C. - 1986, Atlantic C. - 1984
. IFIP, Dublin -1986, Paris - 1983.
. Annual Symposium on Computer Architecure, Tokyo -1986, Stockholm - 1981.
. International Conference on FGCS, Tokyo, 1984

Program Committee Member:
. Parallel Architecture and Language Europe, PARLE'89
. International Conference in Logic Programming, Seatle 88
. IEEE Symposium on Logic Programming, San Francisco 87