Research on Parallel Logic Language Implementation and Architecture at ICOT

A Trip Report

M. Hermenegildo
Department of Computer Science
The University of Texas at Austin
Austin, TX 78712 (USA)
herme@cs.utexas.edu

1 Introduction

This report summarizes my visit to ICOT from August 28 to September 13, 1988 as a result of a generous invitation from Dr. Fuchi. This visit was a remarkable experience for me and I am deeply grateful to Dr. Fuchi and Dr. Uchida for making it possible. The general purpose of the visit was to exchange views on several aspects of the design, implementation, and performance analysis of sequential and parallel logic languages and inference machines, and to discuss my own work on independent AND-parallelism and global analysis of logic programs. Because of my research background in parallelism and logic programming I was assigned to the 4th. research laboratory, headed by Dr. Uchida, which is the home of the PIM, Multi-PSI, PIMOS, and KAPPA projects. Unless otherwise stated, the reports of my meetings and my comments refer to this part of ICOT research.

I spent most of my limited time with different researchers, either presenting my own research, listening to their presentations, or discussing several issues. However, I also managed to keep up with my e-mail and even get some work done on my own. The visit was of benefit for me not only because of the excitement of being part of the historical fifth generation project (if

only for a short time), but also because I learned much about the current status and details of the research going on at ICOT, ICOT's future plans, and the general nature of research in Japan. I also hope that the visit was helpful to ICOT's fourth laboratory: that I was able to share whatever experience I have in the areas of language design and implementation (with particular emphasis on independent and-parallelism), global analysis and abstract interpretation, compilation, parallel execution model development, architecture design, and performance analysis. I didn't propose major global solutions or changes in direction (which I think may not be appropriate in the current phase of the project) but I believe I succeeded in offering suggestions, improvements, or alternative solutions to various aspects of the designs that were explained to me.

The rest of this report is devoted to summarizing my technical comments and suggestions concerning the different projects that I was exposed to. I also kept a daily chronological record of my activities during this visit to ICOT. I have included it at the end (as appendix A) in case it is of any interest to the curious reader. It summarizes my activities inside and outside ICOT, my presentations, and my discussions with ICOT researchers. The appendix also includes some comments on the ICOT research environment and some other bits of information about ICOT (perhaps at times somewhat anecdotical) which cannot usually be found in technical reports.

2 Technical and General Comments

In this section I would like to offer my comments, both from a technical and a more general point of view, about the research being conducted at ICOT and ICOT's activities and organization. First of all, I would like to say that I was in general very favorably impressed by the research being done at ICOT. I was very positively impressed by ICOT researchers and their accomplishments. However, little gain can come from complacency and real progress can only stem from the identification of points which need improvement and the devotion of resources to such improvement. Therefore, and for the purposes of constructive criticism, I have also included in the following paragraphs comments on areas which in my opinion could be improved. I sincerely hope that the reader (whether from ICOT or not)

doesn't find my comments too critical, and I hope that he or she can excuse me for them if they are out of place. In any case their only purpose is to perhaps help improve even more an already excellent research record, my reasoning being that, apart from my own personal interest in ICOT's high success, excellent results from ICOT can only be of benefit to the logic programming community in particular and to international computer science in general. In any case, my comments should be taken with a grain of salt, specially considering that they are based on information gathered during a relatively short stay. Also, they are based on simply technical arguments, which I do understand have to be tempered by many other constraints of a more "practical" nature (this, at least no less at ICOT than at any other research institution that I have some experience of).

Although my visit was rather short I did make some concrete suggestions during my discussions with ICOT researchers. I would like to first summarize some of these suggestions, as they refer to the different systems that I had an opportunity to study. In general I have made an effort not to comment on the high-level research-direction decisions taken at ICOT but rather concentrate, given those decisions, on the specifics of their implementation. Also, I have tried purposely not to include too much description of the actual activities of ICOT or details about the systems being developed there because this information can easily be found in existing or future technical reports. At the end I also offer some comments of a more general nature.

2.1 KL1 Compiler and Run-time Systems

In general I liked the overall design of the compiler and the bulk of the decisions made up to now. I was a little surprised to learn that the current KL1 compiler is written in Prolog (rather than in KL1), although I do understand the reasoning behind it: that a mature and stable Prolog system (rather than the still unstable KL1 system) was preferred for the development of the initial compiler. I was happy, however, to learn that Sekita-san is writing a new compiler in KL1. I clearly encourage this move (despite my openly stated fondness for Prolog) because a native compiler is obviously essential for being able to bootstrap a standalone system. In addition, a compiler written in KL1 should also be very useful as a benchmark, and as proof of the usability of the language.

Perhaps the major point where I feel there should be room for improvement is in the run-time performance of KL1 on any given system. This includes both the KL1 compiler and the emulator/microcode run-time systems on the various machines. In many ways KL1 can be considered a lower-level language than Prolog (hence, presumably, the existence of KL1-U) and it simply shouldn't run slower than Prolog (or other high-level, declarative languages) on most machines. Granted that the PSIs are optimized for Prolog execution, but on a relatively general-purpose machine (such as the Symmetry and perhaps also the PIMs) basic performance metrics such as uniprocessor execution speed and memory efficiency should at least be comparable. The situation will not be any different when the PIMs are ready and run KL1 at a good speed: the PIMs will also run fast parallel Prolog, and parallel C, and parallel LISP, since they are essentially general-purpose machines. As I mentioned before, irregardless of the fact that there may be several sources of inefficiency inherent in the language, I believe that there are still many ways in which it may be possible to attain better performance in KL1 execution, by improving both the compiler and the run-time system. I believe a significant amount of effort should be devoted to this, specially since improvements, specially in compilation technology, that result in better performance on the PSIs, Multi-PSIs, and Symmetries can only mean better performance on the future PIMs.

Given the importance of the performance issue I think a top-down, focused analysis effort is needed. The minimum objective of a parallel system should be to run faster than "competing" sequential systems of comparable cost. End performance is obviously the product of speedup and sequential performance. Therefore, both of these factors should be consciously optimized and issues affecting performance should perhaps be divided into those which affect sequential performance and those which affect parallel performance. I will concentrate mostly on the facts that affect sequential performance first, for the obvious reason that they are better understood. I will try to return to the facts affecting parallel performance (such as granularity) later, notably in the discussion of Multi-PSI. In any case it is important to avoid the all too frequent tendency to concentrate on optimizing details of the system while the main sources of overhead are at a higher level. In this regard all of the comments in the next paragraphs should be taken in the

light of the higher level comments regarding granularity analysis and load balancing later on.

In any case, it is my feeling that a more comprehensive performance analysis effort, based on a large set of benchmarks (preferably, real application programs) is necessary in order to precisely identify where the most time is spent in KL1 for a given architecture, i.e. whether it is fetching instructions, handling suspensions, context switching, garbage collection, etc. It is essential to establish a perhaps more effective feedback loop so that the knowledge obtained from the performance analysis guides the compiler and run-time system improvements which in turn hopefully affect the results of the benchmarking. This point of the importance of engaging in a serious performance analysis was apparently also raised by previous visiting researchers. Significant progress has been made in this direction, with emphasis on the area of memory performance, but more experiments using more and larger benchmarks are still needed in my opinion.

A result of previous analysis has been the relatively early detection of the potential for overhead and importance of GC (expected to represent perhaps 120% of the overhead on the PIMs). In view of these results, several methods have been developed aimed at reducing the incidence of this problem such as the MRB and LRC schemes. I believe it is important that substantial experiments be run with and without MRB, and perhaps also with and without LRC in order to assess what the real overhead involved in these dynamic garbage collection methods is and whether they are of real overall benefit. For a fair comparison, the overhead of maintaining free lists etc. should also be taken into account. My personal feeling (based on the limited data currently available) is that the overall final performance could very well be similar in the end for all methods! This may leave a difficult decision between the simplicity of standard stop and copy garbage collection (but with the inconvenience to the user of having periodical suspensions of execution for GC) and the elegance of a continuously running machine (but at significant complication in the implementation). The possibility should not be discarded that the insidiousness of the GC problem may be pointing out an intrinsic disadvantage of KL1 in this particular area with respect to systems which do partial garbage collection through backtracking, such as Prolog.

Regarding the compiler, I made some comments based on the current design itself and the performance data known to me. In general I suggest adding more sophisticated analysis at both the front and back ends.

At the front-end of the compiler, I suggest that global analysis and abstract interpretation techniques be applied to determining which clause in a procedure is likely to commit, reordering the body goals in the way that best minimizes suspensions during sequential (and parallel) execution, and recognizing cases where it is more advantageous to run goals locally than making them available for parallel execution. Of course, the global analysis should take into account all the possible goal orderings and producer-consumer patterns. Kimura-san and I discussed some ways in which such an analysis could be approached.

At the core of the compiler, some changes in the basic architecture may bring improved performance. For example, reevaluating the entire procedure after a suspension seems wasteful. A pointer in the suspended goal frame containing the offset from the top of the procedure code where the suspension occurred could do the job of restarting at the point of suspension while taking care of relocation problems during garbage collection. Of course the situation is complicated by the necessity of recreating the temporary register status upon return. A simple solution to this problem is to just skip the execution of the clauses above the one which caused the suspension but reexecute the head of the current clause thus recreating the temporary register state. Better performance can be obtained by generating reentry code for each point of potential suspension. This code simply recreates the state before suspension, taking into account the indexing scheme used. As usual, there is a spectrum of possible solutions offering different levels of complication and potential performance improvement. Again, measurements are necessary to estimate the real impact on performance of restarting at the beginning of the procedure after suspension. However, even if the impact appears to be low, this doesn't mean that it shouldn't be further optimized. It is important to note that if no given part of the abstract machine appears to be the single major cause for the overhead, if overall performance is low, this simply means is that *everything* needs to be sped up equally!

Another point of improvement might be the scheme used for implementing the case when several goals are waiting on a variable instantiation or

when a goal is waiting on several variables. At first sight, at least, it looks complicated and perhaps inefficient in terms of both space and time. Depending on the frequency of use of this scheme it may make sense to work more on simplifying it.

Finally, the new indexing scheme proposed by Kimura-san seems good and I specially liked the fact that it frequently avoids repeating work (such as dereferencing) done during indexing when executing the head. A possible improvement in this area is to include hashing. Also, the current scheme builds an indexing tree (which has logarithmic traversal time). This could be optimized by building a matrix including all the possible cases so that the address of the matching clause is computed in one operation.

At the back-end of the compiler (and perhaps also at the core) if more performance is to be attained, the compiler has to be tailored more to the particular architecture and organization on which the program is to be run. However, one of the problems in trying to do this is the fact that a single compiler is being used for architectures dramatically different such as the PSI-II, Multi-PSI, Sequent Symmetry, and the PIM RISC processor. The simplest way to tackle this problem is by adding several different backends to the compiler. Suggested roles for these backends for the different architectures under consideration at ICOT could be as follows: for execution on a CISC machine like the PSI or on a byte-code emulator system some collapsing of the instructions is probably needed. I do understand that a fast instruction prefetch unit may bring the advantages of collapsing in part, but collapsing instructions also makes *cross-instruction optimizations* possible (such as, for example, avoiding multiple dereferencing). Related to the subject of collapsing instructions, type-checking instructions which are currently separate from builtin calls may better be included within the definition of the builtins themselves.

For implementation on machines that are more RISC-oriented than the PSIs, such as conventional sequential workstations and multiprocessors or the PIMs, it may prove useful to redefine the instruction set in terms of smaller units (such as, perhaps, "dereference," or "check tag") so that a peephole optimizer could do cross-instruction optimizations. It may perhaps be advantageous to directly generate machine code, depending on the tradeoff point desired between code size and execution speed.

In general, more sophisticated compilers and run-time systems should be the key to better KL1 performance, rather than putting hopes on future hardware: a KL1 that runs fast on the Symmetry or on PSI-II will probably also run very fast on the PIMs.

2.2 PIMOS

I have fewer comments on PIMOS, probably due to my limited expertise in language-oriented operating systems. I liked the structuring of the tasks and resource tree. I think it might be interesting to consider the possibility of extending the system to support multiple users. This may be facilitated if a Shoen is made to look more like a conventional (Unix-like) process environment, with some notion of locality of code and independence of data areas. It also seems like it might be possible to integrate the treatment of interrupts in a tighter way. Another area of possible improvement would be making priority management hierarchical: i.e. making it possible to specify the priority of each sub-Shoen within each Shoen in a *relative* way. The actual priority of any Shoen and all its children would then be determined by the priority of its parent, thus incorporating a notion of fairness.

There is no doubt that very interesting research issues are being tackled in the PIMOS group and that research should therefore continue. However, I would not recommend greatly increasing the amount of resources devoted to the task. There is no question about the theoretical interest of higher-level language based operating system research, especially since, to my knowledge, not many groups seem to be doing this sort of research in the world right now. I also understand that, for ICOT, presenting as the final parallel inference machine a machine where the operating system is UNIX and the native language is C would not be acceptable. Unfortunately, there is also the reality of the little current commercial success of related single-language systems (such as the lisp-machine)...

2.3 PSI Machine and Multi-PSI

I didn't have much time to work with the PSIs, but I did have enough to learn how to log in, create an ESP program and run it. The operating system and windowing system seems relatively robust and usable, although not blindingly fast. It seemed to have a much smaller learning curve than

a lisp-machine, although functionality also seems to be less, which is understandable considering the relative ages and stages of development of both systems. The Pmacs editor is very nice, although limited in comparison with gnu. I found ESP a little convoluted, due to the object-orientation support, but definitely usable. My main concern with the PSI is that there still seems to be no complete benchmarking and performance analysis for it. I think it is important to try to determine the strengths and weaknesses of architecture and organization. I understand that there are resource limitations, but such knowledge might be of importance, specially considering the HW is being used in the Multi-PSI systems and its performance will be compared to the results obtained with the PIM processor in order to quantitatively study the tradeoffs between CISC and RISC.

Regarding the Multi-PSI machines, I understand that one of their objectives is to serve as a testbed for the concepts used in the PIM. In relation to this, I think a nice medium-level PIM simulator could be built on top of the Multi-PSI with relatively small effort by making each Multi-PSI processing element simulate a PIM-cluster using a modified version of the Pseudo-Multi-PSI software, and making the Multi-PSI network simulate the PIM interconnection. It is a pity that given the current timing it is possible that the results and experience from Multi-PSI II perhaps won't be gathered before the PIM design is closed and the first prototype built.

I think that the choice of a PSI as the processing element and a mesh network may not have been ideal (further suggested by the later adoption of a RISC-based processor, shared-memory, clustered system for the PIM). A matter of perhaps some concern might also be the proposed code distribution method being a bottleneck. A broadcast facility in the network might perhaps have been helpful in this respect. Another possible area where performance might suffer is I/O, since I/O operations are all centralized through the PSI master machine. This may be a problem, for example, in parallelizing compilation, where sequential I/O time can reduce the speedup of an otherwise highly parallelizable task.

On the other hand, I do understand that the combination of the urgent need for a fast experimental machine, the issues of researcher motivation and impact, and the good sense in "using existing technology" (i.e. the PSI's) warranted the approach taken. However, under no timing or other

constraints a simulation-first approach might have been better.

In any case, I feel that KL1 on the PSI or on one element of the Multi-PSI should be made to run faster, perhaps applying some of the suggested improvements in the previous section on the KL1 compiler, although I agree that the fact that the architecture is tuned for Prolog execution is an issue. However, I believe the main bottleneck in the Multi-PSI, perhaps even more important than the single processor performance, may be the remote access time. I liked the address translation technique using import and export tables in order to support references across the overlapping address spaces of the different processors in the Multi-PSI. However, it seems like it may be a major source of overhead. The optimizations for bypassing some of the steps in this process under certain conditions seemed encouraging and it might be interesting to continue this effort and see if there are other ways in which cross-network references can be sped up. Note also that although the network delay itself may be small in comparison to the software overheads, the network could get very slow under load because of the single path routing. This is another area where some simulations might have been useful.

Given that foreign references can be expensive and since execution speed is affected by the remote access time, it seems of utmost importance to devote a serious amount of effort to devise granularity control, load balancing, and scheduling methods which will reduce the portion of foreign references. This, I feel is the main research issue in the Multi-PSI, not only because of the dramatic way in which progress in these areas will affect Multi-PSI performance, but also because such progress may be directly applicable towards solving the problem of the cost of out-of-cluster references in the PIM. The power plane idea proposed by Chikayama-san might be a good starting point. I believe that it may eventually be possible to perform *almost* automatic load balancing starting from just reduced information obtained from the user if a global analysis of the program is performed.

2.4 PIM

The direction taken by the PIM design team seems very reasonable. It appears that single-language machines don't have a real place in the marketplace currently or in the near future. Therefore, one can only support

the choice of an architecture based on taking relatively proven concepts (coherent cache-based multiprocessors and RISC technology) an evolving them (by adding the cluster concept and special support for KL1) to create a system capable of increased parallelism and of supporting both KL1/PIMOS and conventional languages and operating systems. Of course, from the point of view of the computer architect it is more interesting to design a special-purpose organization than to build on existing concepts or squeeze out more performance form a general-purpose RISC processor but previous experiences in these areas wouldn't make this a sensible choice at ICOT given the expected delivery dates for the PIMs.

The choice of RISC in the PIM (vs. CISC on the PSI) also seems reasonable, even more considering that KL1 is a simpler language than Prolog. The choice of broadcast coherent caches also seems like a good idea. The simulations performed by Tick and myself on the suitability of caching mechanisms for and-parallel execution of logic programs (applied to the case of independent and-parallelism) and the recent related work at ICOT seem to substantiate this. I do think more effort should be devoted to performing some sort of caching of foreign references (i.e. references from another cluster), even if it is perhaps only done on data marked at compile-time as read-only. Apart from code, the single assignment character of KL1 and the fact that there is no backtracking should make it possible to label a fair number of references as read only at compile-time.

As suggestions, I would like to mention that it may make sense to seriously study alternatives to the macrocall facility, such as using a simple "swap registers" instruction instead of the indirect argument access. Also, perhaps some of the arguments of the macrocall (specially the main condition code) should be collapsed into the opcode creating special cases for faster performance. In any case, an emulator should be written and simulations run to evaluate the tradeoffs involved in these choices. Different alternatives should be considered and compared, documenting the design choices and the reasoning behind them. This is specially important considering that the price being paid for the extra hardware added to support the macrocall facility is to have only one processor per board as opposed to two in commercial designs: this appears at first sight to be a disadvantage in cost-performance, so it would make sense to prove it otherwise through

simulation before committing to it.

The same comment as in the Multi-PSI regarding import and export tables applies here: the overhead involved in their management and use appears to be too high. This is even worse in the PIM since the processor speed is higher and the network latency is lower. As an alternative, using a global addressing space (across all clusters) doesn't seem out of the question, since the word-size is 64 bits, thus pushing address translation issues down to the network controller level. I also find peculiar the large amount of unused memory due to the difference between memory and processor word-size. Extra bits in the word should perhaps be used for reference counting, although I understand the problems with chip area incurred into if register size is increased (another related issue is the need to reduce the cost of suspensions: this could perhaps be done by adding more registers, but that is also limited by chip area considerations). In any case, and as I mentioned in the sections on the compiler, the cost-performance of the on-the-fly garbage collection techniques should be very carefully studied before the considerable extra complication involved in LRC and MRB is wired into the machine (specially if HW support is being considered). From the current simulations LRC or MRB don't really seem to be much better than normal GC (except for the increased locality, but this decreases as the number of processors increases) specially if the overhead in maintaining all the free lists is taken into account. The possibility of not having to stop the whole machine for garbage collection is of course very attractive but none of the approaches appears to guarantee completely avoiding GC for all cases (e.g. memory fragmentation).

As far as performance goals, a set of significant benchmarks should be prepared to test how close the different steps of evolution of the PIMs come to the desired goal. Also, the performance should be compared to other systems, in particular to that of parallel Prolog systems running on commercial shared-memory multiprocessors. Also, some serious study should be done of the impact of network latency on overall performance in the PIM. The ratio of local to remote access time in the PIM could be as high as 1/500. I suggested before reviewing the caching strategy so that some of the out-of-cluster references are also cached. In general, the overhead involved in accessing out-of-cluster data, coupled with the granularity, scheduling,

and load balancing issues may well be the biggest bottlenecks in the machine and the single highest priority problem to solve. In this sense the same comments mentioned in the discussion of the Multi-PSI regarding the utmost importance of tackling these problems are appropriate, and the results obtained in the Multi-PSI studies may very well be applicable. This is obviously a very hard and difficult task, but it needs to be done! There is no point in devoting effort to cutting a couple of microcycles off from a low-level instruction if that same instruction has a high probability of referencing in the next microcycle an address in another cluster that it may take thousands of cycles to fetch.

2.5 General Comments

I was pleasantly surprised in many cases with the knowledge that ICOT researchers have of their fields. On the other hand I do feel that researchers at ICOT could sometimes be more aware of (and reference!) other work being done outside ICOT. More importantly, I think the results and approaches used in ICOT work should be more frequently compared with other research results and approaches external to ICOT, on a cost-performance basis. Examples: a comparison of the PSI-II to other Prolog processors (Xenologic's, ECRC's, or even Quintus on a Sun-4), and a comparison of the KL1/Multi-PSI/PIM systems to other languages (Prolog), parallel execution models (Prolog OR- and AND-parallelism), and architectures (conventional multiprocessors). Much progress has been done in these areas, specially considering the very understandable language problems involved, but there may still be room for improvement. In this regard I would like to point out as very positive the results obtained by Dr. Tick during his NSF/ICOT-sponsored visit.

Researchers seem to be relatively aware of what is going on in the rest of the lab. This is indeed very good (especially considering what happens in other research institutions) and is probably due to the frequent meetings (which everybody seems to attend), the "open room," the fact that most researchers are part of several projects in addition to being in charge of one, and the general japanese inclination towards team work. This seems particularly clear in the development of hardware. On the other hand it seems like there is more difficulty in working collectively on a software system,

using conventional software engineering techniques. As a result most of the pieces of software seem to be developed by only one person in a relatively monolithic fashion, which has the undesirable side effect of perhaps limiting program size to that which can be tackled by a single person. Also this person may be the only one who can maintain the piece of software. This seems to be a specially acute problem since researchers eventually have to return to their companies and may leave behind orphan codes.

While communications within a laboratory seem excellent, they appear as more limited between the different laboratories, even though the researchers are in the same room or at most separated by a hall (although I have seen this phenomenon at many other places). For example, some of the work done in the first research Lab. appears to be perhaps too theoretical and could definitely benefit from application to practical problems. Some of their efforts are devoted to program transformations which don't necessarily address the main performance bottlenecks facing KL1. On the other hand, just across the hall, researchers in the 4th. lab are in great need for that technology and know-how for improving the quality and efficiency of their compiled code, essential for high-performance KL-1 execution.

Regarding the issue of funding and cooperation with industry and academia, I found very interesting that the funding from the companies finances only the support functions of ICOT, while the researchers are fully funded by MITI. This seems like a very good idea, since it makes it possible for the research results to be public, i.e. owned by MITI instead of by the participating companies, while promoting very high cooperation with industry. However, it seems like the cooperation with universities appears to be relatively low (beyond the "working group" meetings) in comparison with other research consortia. A reason for this could be the fact that universities are funded by the Ministry of Education instead of MITI. An interesting fact is that comparing high-level budget/head-count ratios it seems like the cost per researcher of research at ICOT is about twice that of similar consortia in the US. This is a little surprising considering that the same amount of money in the US buys workstations and offices for every researcher and (presumably) higher salaries. On the other hand it is true that US consortia seem to have a much lower level of spending in other areas, most notably hardware prototyping, since more emphasis is put on simulation (for better or worse).

This brings me to the subject of the bold "build first-measure later" (if at all) approach that seems to be taken at ICOT towards hardware design. In comparison to the US there seems to be much less simulation and performance evaluation effort. I was surprised to see that some decisions (which directly affected actual hardware) were based on relatively small simulations of perhaps a single benchmark. I remember, for example, a discussion in the WG meeting regarding the usefulness of MRB in improving locality and coherent cache performance which was based on the results for only one benchmark. I feel that more and much larger benchmarks have to be used (specially since it is a large parallel machine).

I would like to finish my comments on a more technical note. Although a repetition of points that I have made before, I believe much more emphasis should be put on performance analysis (and comparison to other approaches), and to studying high-level issues such as load distribution, granularity analysis, parallel algorithms, communication locality, standard programming styles, etc., etc. These issues are bound to affect performance more than other lower-level optimizations. I am not saying that it is easy to do this: it is hard, but very important. The situation is complicated by the fact that application people understandably don't want to worry about what they consider "low level" issues. They expect things like load balancing to be done automatically for them. The problem is that in order to do the task automatically a minimum understanding of how to do it by hand is first needed and that expertise similar to that of application people is required in order to do this. In the end it all boils down to the realization of the limitations which exist in current parallel compilation technology. We know how to compile for sequential machines (even worrying about pipeline breaks etc.) but our knowledge is still very limited in some basic aspects of compilation for parallel execution. This is not a problem that only ICOT faces, but to underscore the importance of solving the problem let me repeat the basic question, formulated in terms which directly affect ICOT's goals: what is more important for Multi-PSI (and for PIM) performance, optimal pipeline continuity or optimal goal scheduling?

3 Conclusions

¿From a technical point of view, I would like to restate that I was in general very favorably impressed by the research being done at ICOT, by the quality of ICOT researchers, and by their accomplishments. I am very honored to have been able to be part of ICOT even if only for a brief period.

¿From a personal point of view I feel profoundly indebted to everyone at ICOT for their warm welcome, for pampering me during my visit, and for making it a most memorable experience. I would like to thank Dr. Fuchi, director of ICOT research, and Dr. Uchida, director of the 4th. and 2nd. labs for the invitation and the opportunity to share a "time-slice" of the life of ICOT in general, and of the 4th. research lab. in particular. I am deeply indebted to my hosts Kimura-san and Yamamoto-san which went out of their way to make my visit very enjoyable. I also very much enjoyed my technical and personal interactions with Goto-san, Ueda-san, Taki-san, Chikayama-san, Yoshida-san, Sekita-san, Nakashima-san, Rokusawa-san, Nakajima-san, and with all the other members of the 4th. lab (too many to list here) with whom I had enrichening conversations and who made me feel always welcome. Also my appreciation for Dr. Iwata whose detailed planning was instrumental in the smooth flow of my visit and whose help (and his assistant's - Momose-san) was invaluable for mi sightseeing trips. Finally, I would like to thank Dr. Tick for his hospitality and for sharing with me some of his (by September 1988) definitely very good knowledge of Tokyo.