Working Within the FGCS National Project

E. Tick

CIS-TR-92-20
October 1992

Abstract

The Japanese Fifth Generation Computer Systems (FGCS) extending from 1982–1992 has been enshrouded in hype since its inception. The national project made bold promises in the fields of artifical intelligence and computer engineering, and these goals were amplified by the media and overseas researchers. Now that the dust is settling, it is time to take a good long look at the project, its accomplishments, and failures. This essay is a personal account of my experiences working within, and around, the FGCS project at the Institute for New Generation Computer Technology (ICOT) and elsewhere. I focus on the development of hard and soft technologies, as well as human infrastructure.

This article will appear in the *Communications of the ACM*, March 1993.

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE
UNIVERSITY OF OREGON

"It is not very often that Westerners get to see the Japanese just as they are. The difficulty we have when we look at Japan — the layers-of-the-onion problem — can be so frustrating that we tend to raise our own screen of assumptions and expectations, or we content ourselves with images of the Japanese as they would like to be seen. If you live in Japan, you learn to value moments of clarity — times when you feel as if you'd walked into a room where someone is talking to himself and doesn't know you're there."

Letter from Japan
P. Smith [71]

1 Introduction

This article summarizes my views of the Fifth Generation Computer Systems (FGCS) project conducted at ICOT over the period of 1982–1992. My participation is somewhat unique because I was both an ICOT visitor in February 1987 and then a recipient of the first NSF-ICOT Visitors Program grant, from September 1987–September 1988. For that year I conducted basic research in the laboratory responsible for developing parallel inference multiprocessors (PIMs). In 1988 I joined the University of Tokyo, in the Research Center for Advanced Science and Technology (RCAST), with a visiting chair in Information Science donated by the CSK Corp. Thus over the period of 1987–1989 I had an "insider's view" of the FGCS Project. Furthermore, for the past three years, at the University of Oregon, I have continued to collaborate with ICOT researchers in the PIM groups.

Summaries of the FGCS Project successes and failures by most foreign researchers tend to categorize the abstract vision (of knowledge engineering and a focus on logic programming) as a great success and the lack of commercially competitive hardware and software as the main failure. I would like to place these generalizations in the specific context of my personal involvement with the project: my own corner of things, and my assessment of parallel inference machine (PIM) research as a whole. Furthermore, I would like to comment on more subtle successes and failures that fewer observers had a chance to evaluate. These results involve the training of a generation of computer scientists.

This article is organized as follows. In Section 2, I summarize my own research experience at ICOT. This leads to Section 3, where technology contributions are addressed: did ICOT take the "correct" road to next generation computer development? In Section 4, I switch my view to ICOT's influences on the development of human resources within Japan. Much of the analysis and conclusions drawn here are based on discussions with Japanese engineers and managers in both industry and ICOT. Conclusions are summarized in Section 5.

2 "Stranger in a Strange Land"

I had applied for the NSF–ICOT Visitor's Program grant with the goal of both extending my thesis research (evaluating the memory-referencing characteristics of high-performance implementations of Prolog [78]) as a post-doc, and living in Tokyo. Both of these goals were equally important, so the NSF–ICOT grant was ideal. Prior to graduating, I had studied Japanese at Stanford for two years, in addition to making two short visits to Tokyo. Strangely, rather than being interested in that vision of rural Japan professed by travel posters and paperbacks, my interest was limited almost exclusively to Tokyo, the farthest one can be from the "real" Japan, and still be on Japanese soil. Yet to me, this *was* the "real" Japan: the vitality of a sensory overload of sound trucks, a myriad of ever-changing consumer products, ubiquitous vending machines, electronics supermarkets, and a haywire of subway and rail systems.

It seemed only appropriate that ICOT was located in the midst of all this motion: downtown Mita in the Minato-ku ward of Tokyo. Located inside the Yamonote-sen, the railway line encircling the city, Mita has immediate access to all parts of cosmopolitan Tokyo either by rail, subway, taxi, bicycle, or foot. ICOT members were all "on loan" from parent companies and institutions, most of which were on the outer belts of the city, so they had long commutes. I was not so constrained, and found an apartment in Hiroo, a convenient 30 minute walk to ICOT. Furthermore, Hiroo was only 30 minutes by foot to Roppongi (nightclub district), and by bike to Ooi Futo, a circuit where local racers practiced on Sundays.

It was in this environment, not unlike Queens, New York where I grew up, that I started working in September 1987. My first day, I arrived at Narita airport at about 8 am and took the liberty of grabbing the first available bus to Mita, and then a taxi to ICOT. I had brought a Macintosh, so I figured I would first dump it off at work and say hello to everyone. Unfortunately, no one at ICOT was expecting my arrival *there*. To the contrary, they had informed an NHK film crew of my imminent arrival at *my hotel*. The bungle was diplomatically solved by calling the film crew over to ICOT, having me carry my backpack and Mac back down to the lobby, and staging an "official" arrival for Japanese TV. We then proceeded up the elevator, TV cameras glaring, to my desk, so they could record my unpacking of the Mac. The gist of it was "strange gaijin (in tennis shoes) brings own computer to Fifth Generation Project..." It was fairly amusing, although it is always disconcerting how naive the media are. Almost as funny was the next week when a newspaper requested my photo, which I had taken at a nearby film shop. The next day's paper displayed that photo, but with a necktie drawn in. Interesting cultural differences, but moreover an indication of a time of peaking national limelight for the project.

2.1 Expectations and Goals

My expectations were to continue my research in the direction of parallel logic programming languages implementation and performance evaluation. At the point when I finished my thesis, I had only begun to explore parallel systems, primarily collaborating with M. Hermenegildo.[1] In 1987, Hermenegildo was developing the prototype of what later evolved into &-Prolog, a transparent AND-parallel Prolog[2] system for shared-memory multiprocessors [34, 37, 36]. This work introduced me to the world of parallel processing during a period of great excitement in the logic programming community. One of the primary triggers of this excitement were the new shared-memory multiprocessors from Sequent, Encore and BBN in the mid-1980's. Many research groups were developing schemes for exploiting parallelism (e.g., [3, 8, 16, 26, 45, 50, 55, 89]). I think funding for the FGCS Project motivated many international researchers to continue in this area. Two particularly promising systems at the time were Aurora OR-parallel Prolog[3] [13, 51, 73] and the family of committed-choice languages [70, 69]. Aurora was being developed at the University of Manchester (later at the University of Bristol), Argonne National Laboratories (ANL) and the Swedish Institute of Computer Science (SICS). Committed-choice languages were being developed primarily at the Weizmann Institute, ICOT, and Imperial College.

Both &-Prolog and Aurora were meant to transparently exploit parallelism within Prolog programs. &-Prolog was based on the idea of "restricted AND-parallelism" [21] wherein Prolog goals could be statically analyzed to determine that they shared no data dependencies, and could thus be executed in parallel. Aurora exploited OR-parallelism wherein alternative clauses defining a procedure definition could be executed in parallel, spawning an execution tree of multiple solutions. The committed-choice languages represented a radical departure from Prolog: backtracking was removed in favor of stream-AND parallelism, similar to that in Communicating Sequential Processess (CSP) [39]. These approaches were promising because they potentially offered low overhead essential operations: variable binding, task invocation, and task switching.

It was in this whirlwind of activity that I mapped out my own project for ICOT. Such planning was a good idea in retrospect. Some other visiting researchers had failed to plan ahead, and floundered, unable to connect with a group to work with.

[1] Hermenegildo was at the Microelectronics and Computer Technology Corporation (MCC) at the time, now at the Technical University of Madrid (UPM).

[2] Logic programs are composed of procedures defined by Horn clauses of the form: H :- $B_1, B_2, ..., B_k$ for $k \geq 0$. The head H contains the formal parameters of the procedure corresponding to the clause. The body goals B_i contain actual parameters for procedure invocations made by the parent procedure. AND parallelism exploits the parallel execution of multiple body goals.

[3] OR parallelism exploits the parallel execution of alternative clauses defining the same procedure.

Although I should have learned from these experiences, I myself fell into the same trap at the University of Tokyo a year later. Both institutions had trouble integrating young visiting researchers into the fray because of both language differences and an unjustified assumption (in my case) that "visitors" were omniscient experts who didn't *need* mentoring.

My proposed research was to evaluate alternative parallel logic programming languages, executing on real (namely Sequent and Encore at the time) multiprocessors. Specifically, M. Sato of Oki was at that time building Panda, a shared-memory implementation of Flat Guarded Horn Clauses (FGHC)[4] [82] for the Sequent Balance. An early (Delta) version of Aurora was also obtained from ANL. My intent was to get a set of comparative benchmarks running in both FGHC and Prolog, for judging the merits of the approaches. It would have been a coup to include &-Prolog in the comparisons, but the enabling compiler technology was still under development [54].

The measurements I wanted were of the type previously collected for Prolog and &-Prolog: low-level memory behavior, such as frequency of data-type access, and cache performance. Work progressed on several fronts simultaneously: as Panda and Aurora were stablizing, a collaborative effort with A. Matsumoto to build a parallel cache simulator was underway, as was benchmark development. The Panda and Aurora systems were then adopted and harnessed for the simulator. The simulator was interesting in its own right: to enable long runs, it was not trace-driven, but rather ran concurrently with language emulators [31].

The overall goals of this research were to conduct some of the first detailed empirical evaluations of concurrent and parallel logic programming systems. Little work had been done in performance evaluation: most logic programmers were furiously creating systems and simply not analyzing them, or at best measuring the inference rate of list concatenation. The final word on performance characteristics, such as was later documented for imperative languages by Hennessy and Patterson [33], was impossible for logic programming languages because of the lack of an established body of application *programs.* Although Prolog had received widespread use and recognition, leading to industrial-strength applications [84, 5, 72], concurrent languages and parallel variants had no such history. As a result, my goals were practically oriented: to make an incremental improvement in the benchmarks over the standardized U.C. Berkeley and Edinburgh benchmarks [22, 88], and to continue to improve the range of our analysis, into multiprocessor cache performance. This also led to a book [80], written primarily at the University of Tokyo.

[4]KL1 is the supersetted FGHC that is supported by the PIM architectures.

2.2 Successes, Failures and Frustrations

During the summer of 1988, A. Ciepielewski of SICS visited ICOT and helped a great deal with instrumenting Aurora. This required rewriting the lock macros, the Manchester scheduler [12], and other nasty business. By summer's end, we had produced running systems evaluating a large benchmark suite, which had been refined over the months for performance considerations. In August I started writing a paper on our results [79], finishing off my ICOT visit and moving to Todai. The technical results of the empirical study were summarized in the Lisbon ICLP [79]:

> The most important result of this study was a confirmation that indeed *(independent) OR-parallel architectures have better memory performance than (dependent) AND-parallel architectures.* The reasons are that OR-parallel architectures can exploit an efficient stack-based storage model whereas dependent AND-parallel architectures must resort to a less efficient heap-based model. For all-solutions search problems, a further result is that *non-committed-choice architectures have better memory performance than committed-choice architectures.* This is because backtracking architectures can efficiently reclaim storage during all-solutions search, thereby reducing working-set size. Committed-choice architectures, like functional language architectures, consume memory at a rapid rate. Incremental GC can alleviate some of penalty for this memory appetite, but incremental GC also incurs its own overheads [62]. Thirdly, for single-solution problems, *OR-parallel architectures cannot exploit parallelism as efficiently as dependent AND-parallel architectures can.* Although OR-parallel goals may exist, they are often too fine-grained for the excessive overheads necessary to execute them in parallel. In this respect, *dependent AND-parallel architectures can execute fine-grain parallelism more efficiently than can OR-parallel architectures.*

There were some objections to these conclusions among those implementing the systems. The basic criticism was simply that the application domains of the languages differed, and so comparison was inappropriate. A more far-reaching question, that still has not been resolved by the FGCS Project, is the utility of an all-solutions search method as realized by backtracking. ICOT made a radical decision in 1982 to use logic programming as its base, and then another radical decision to switch to committed-choice languages a few years later. Although many techniques have been explored to recapture logical completeness [29], none of them have been entirely successful. This gap has led other research groups to develop more powerful language families, such as concurrent constraint languages (CCLs) [66] and languages based on the "Andorra Principle" [10, 17, 6]. First attempts at both of these families at ICOT are GDCC [77] and AND-OR-II [74], respectively.

I felt my greatest success at ICOT was collaborating with several engineers, contributing to both my project and others. These included both the members of the PIM laboratory, as well as ICOT visitors, such as J. Crammond of Imperial College. The congeniality at ICOT was unsurpassed, not just to foreign visitors, but among the different company members. Sometimes I thought that it was a bit *too* congenial, and that the lack of (externally directed) aggressive competitiveness was detrimental to the FGCS project overall. As in most Japanese institutions, foreign collaboration was somewhat carefree because the visitors were not treated as true members of the team. The main reason for this was lack of reading skills needed to fully participate in group meetings and prepare working papers. A related problem was a lackadaisical attitude towards citing the influences of foreign researchers. I attribute this somewhat to language differences, but primarily to lenient academic standards inherited from a corporate culture.

An implicit success, I hope, was that my technical analysis helped uncover systems' problems that were later fixed. During the project, development progressed on the Aurora schedulers [7, 12, 11], improved Prolog and FGHC compilers, and alternative systems such as JAM Parlog [18] and MUSE OR-parallel Prolog [2]. In a sense, these developments were frustrating because they made my small experiment somewhat obsolete. The half-life of empirical data of this type is very short: results rarely make it to journal form before the underlying systems have been "versioned-up." Furthermore, I hope the conclusions derived comparing OR-parallel Prolog to stream AND-parallel FGHC had some influence on the Andorra model [10], developed by D. H. D. Warren to combine the two.

A limited success was my influence on ICOT researchers in the PIM groups. I think the empirical slant influenced a number of researchers to devote more care to evaluating their designs. Still, I don't think enough emphasis was placed on producing quantitative performance analysis (concurred in [35]). Considering the massive efforts that went into building the Multi-PSIs and PIMs, few publications analyzing the key performance factors were generated (see [63]). I blame the three-period FGCS schedule for this. In 1988, the groups were struggling to complete the Multi-PSI-V2 demonstration. Yet the design of the PIM machines were largely underway, and little if any Multi-PSI experience collected after 1988 affected PIM. Still, the sophistication of PIM instrumentation and experimentation (e.g., Toshiba's PIM/k cache monitors and Fujitsu's ParaGraph [1]) have improved over the final period, even if the newer application benchmarks [63] have not yet been exercised on the PIMs.

One impediment to my research was lack of compiler technology. This was prevalent throughout the systems: compile-time analysis was not yet on par with that of imperative languages, thus lessening the importance of what we were measuring. Recent work in logic program compilation (e.g., [76, 86, 32, 90]) indicate that significant speedups can be attained with advanced optimization methods. A related frustration during my stay in Japan was something as simple as lack of floating point arithmetic in parallel logic language implementations. Even SICStus Prolog had such an inefficient implementation of floating point numbers as to make it unusable. I accidentally discovered this when attempting to implement a new quadrature algorithm for the N-Body problem, developed by K. Makino at the University of Tokyo. I had just joined the school after leaving ICOT, and was eager to find that "killer application" which would link number crunching with irregular computation that logic programs are so good at. I had read a paper by Makino [52] describing a method for successively refining space into oct-trees, and exploiting this to approximate long-distance force interactions. I walked over to his office, surprising him one day, got the particulars of how to generate test galaxies, and hacked up the program in Prolog. For a long period afterwards I could not get the program to perform more than a few iterations before the heap overflowed. Tracing the problem it became apparent that all unique floating point results were interned! This was disappointing primarily because it was a lost opportunity for cross-disciplinary research within the University, which I sorely wanted. Furthermore, it indicated the early-prototype state of parallel logic programming at the time, since none of the systems could tackle applications driving the U.S. markets [15].[5]

2.3 20/20 Hindsight

My belief going into the ICOT visit was that transparently parallelizing sequential languages was best, e.g., in some gross sense, let's exploit parallelism in Prolog as in Fortran. I still believe this approach to be very useful, especially for users who cannot be bothered with concurrency. However, during my stay I discovered the additional value of expressing concurrent algorithms directly, and became an advocate of concurrent languages. My own intellectual development included above all learning how to write concurrent programs (and I am still learning how to make them execute in parallel!). I continue to believe that concurrent languages are great, but I realized, after

[5]It should be stressed that there are no exceptional technical problems preventing logic programming systems from having first-class floating point, e.g., Quintus Prolog implements the IEEE standard. Recently, SICStus Prolog floating point has been repaired, benefiting Aurora and &-Prolog, both based on it. The PIM systems also support floating point.

all the effort developing those benchmarks, that although concurrent languages often facilitate elegant means to implement algorithms [26, 69, 80], the languages *per se* were rarely used to model concurrent systems. The mainstream concurrent languages do not support temporal constraints, thereby making certain types of modeling no easier than non-logical languages. Furthermore, it was still quite messy to express collections and connections of nondeterminate streams in these languages (recent languages have been designed to assuage this problem, e.g., A'UM [91], Janus [67], and PCN [25]).

My greatest technical criticism of my own research was that it was limited in scope, both in systems (Aurora *vs.* Panda) and benchmark programs. If the languages had been standardized years earlier, especially among the committed-choice logic programming language community, we could have collected much more significant benchmarks. Alternatively, if I had the fruits of ICOT applications development conducted during the third period of the FGCS project [63], I would also have been in a stronger position. Advanced compilers are still, to this day, not available.

On the personal side, collaboration was not always smooth, but that made our successes more of an accomplishment. There were more than a few arguments during the years, as to alternative methods of implementing one feature or the other, or bureaucratic culture shock. An amusing example of the latter was that to bypass MITI software release restrictions, we would publish the source listing of the software in a technical report, e.g., [81]. My aggressiveness was not immediately understood for what it was (I like to think it is healthy competitiveness) until people got to know me better over the years. As in any organization, the Japanese were no different in that software developers become possessive and protective of their systems. When I made a branch modification of those systems, and bugs appeared, the first question was: did I cause this bug, or did I inherit it? Software systems were not as successfully partitioned among engineers as was hardware, usually resulting in a single guru associated with each system. This necessitated some rewriting of similar systems over the years.

ICOT researchers worked well together, especially considering the number of companies they hailed from. There were some communication difficulties among the research laboratories. My own stated problems with compilers and applications resulted in part from differing laboratory agendas: the hardware group needed help in these areas, whereas the languages group was working in other areas, e.g., or-parallel search, constraints, meta-programming, partial evaluation, and artificial intelligence [29]. Again, this was a conflict of advanced technology *vs.* basic research that depleted both.

I don't think that ICOT researchers were on average more efficient than those in other institutions. Recall that the end of 1987 brought an avalanche upon U.S.–Japan relations: the October Crash, yen appreciation, and "Super 301" (U.S. trade legislation) left people in a surly mood. Japan bashing began, with foreign media attention on Japanese long working hours, among other things. We didn't work any longer at ICOT than, say, at Quintus or IBM Yorktown Heights (hey, the last trains left around midnight). Long hours don't necessarily translate into insights or efficiency, anywhere. If anything, Tokyo was isolated in some sense, leading to a feeling of remoteness among the research community. For example, in the U.S., universities and industry tend to interact well informally, for instance through university-sponsored seminars. Tokyo shares a great diversity of computer industry and universities, but none of that informal get-togetherness. On the other hand, ICOT *did* have an advantage commanding member manufacturers' resources (namely large groups of engineers) to construct large software (e.g., PIMOS) and hardware (PIM) systems. Such cooperation was quite astounding.

In retrospect, with regard to my own participation, I would have done a few things differently. Primarily, I would have memorized my kanji every day! It was and is simply no fun at all, but Martin Nilsson of the University of Tokyo, now at SICS, demonstrated that it can be done. This would have allowed my increased participation in weekly ICOT meetings, and later at Todai.

3 Technology and the FGCS Project

Hirata *et al.* [38] and Taki [75] wrote excellent articles summarizing the design criteria and implementation decisions made at the software and firmware levels of the PIM. My own research was most closely tied to this laboratory at ICOT, so I will limit my comments to this research. Since all PIMs are organized as loosely connected shared-memory clusters (except PIM/m), each design problem requires both local and distributed solutions. Furthermore, static analysis (by the compiler) must be balanced with runtime analysis. The main components of their design include:

- memory management: concurrent logic programs have a high memory bandwidth requirement because of their single assignment property and lack of backtracking. ICOT has developed an integrated solution for garbage collection at three levels within the PIMs. Locally, incremental collection is performed with approximative reference counting. Specifically, Chikayama's Multiple Reference Bit (MRB) is incorporated in each data word [14]. Distributed data is reclaimed across clusters via their export tables. Finally, if local memory expires, a parallel stop & copy garbage collector is invoked in the cluster [41]. The effort in designing and evaluating alternative garbage collection methods is one of the most extensive of all ICOT projects (e.g., [62, 42, 30, 57]), primarily because the problem was recognized several years ago. However, compilation techniques, such as *statically* determining instances of local reuse [28, 23], were not explored. The tradeoffs between static analysis time *vs.* runtime overhead are still open questions for such techniques.

- scheduling: concurrent logic languages have inherently fine-grain process structures. The advantage is exploitable parallelism, but the disadvantage is the potential of a thrashing scheduler. The PIMs rely on explicit inter-cluster scheduling using *goal pragma* (an attribute telling where the goal should be executed), and implicit (automatic) load balancing within a cluster. Furthermore, goals can be assigned priorities, which steer load balancing in a nonstrict manner. Although functional mechanisms have been completed at ICOT, extensive evaluation has not yet been conducted. Higher-level programming paradigms, such as *motifs* [27], have not been designed to alleviate the complexity of user development and modification of pragma.

- meta-control: pure concurrent logic languages have semantics that allow program failure as a possible execution result. This has long been recognized as a problem because user process failure could migrate up to the operating system in a naive implementation. ICOT developed protected tasks called *shoen* [40], similar to work by I. Foster [24]. Functional mechanisms for shoen management have been implemented at ICOT over the past four years [58, 64, 65], although empirical measurements of the operating system running applications programs have not yet been analyzed. Intra-cluster mechanisms include termination and deadlock detection [43], and resource caching; inter-cluster mechanisms include weighted throw counts. Without further empirical data, it is difficult to judge the effectiveness of the mechanisms for reducing runtime overheads. Furthermore, it is not clear how this research should be viewed: as fine-grain control over concurrent logic programs, or as a full-blown operating system. The latter view would certainly run into commercial problems.

- unification: unification is peculiar to logic programs, and somewhat controversial, in the general computing community, in its utility. Concurrent logic programs reduce general two-way unification into *ask* and *tell* unifications, which correspond more directly to importation and exportation of bindings in imperative concurrent languages. Still, logical variables cause two serious problems.

  First, variables are overloaded to perform synchronization. This is both the beauty and horror of concurrent logic languages. The programmer's model is simplified by implicit synchronization on variables, i.e., if a required input variable arrives with no binding, the task suspends. Furthermore, if at any time that variable receives a binding (anywhere in the machine), the suspended task is resumed. Implementing this adds significant overhead to the binding time, primarily because mutual exclusion is required during binding, and suspension/resumption management.

Second, in a distributed environment, optimizing data locality over a set of unifications of arbitrary data structures is an impossibly difficult problem. Message passing mechanisms defining import/export tables and protocols were developed [40], but little empirical analysis has been published.

Compilation techniques to determine runtime characteristics of logical variables, such as "hookedness" and modes [83], and exploit them to speedup bindings, minimize suspensions, and minimize memory consumption, have not yet been implemented in the current PIM compilers.

The ICOT research schedule began with the development of the personal inference machines (PSI-I,II,III), followed by mockup PIMs (Multi-PSI-V1/2, the latter built of PSI-IIs), and finally the various PIMs: PIM/p (Fujitsu) [46], PIM/m (Mitsubishi) [59], PIM/i (Oki) [68], PIM/c (Hitachi) [56], and PIM/k (Toshiba) [4]. A great deal of credit must go to ICOT's central management of these efforts, based on a virtual machine instruction set called PSL used to describe a virtual PIM (VPIM) running on a Sequent Symmetry [75]. VPIM was shared (with slight modifications) by most of the member organizations, making design verification feasible. These designs are summarized in Taki [75]. Highly parallel execution of dynamic and non-uniform (but explicitly *not* data-parallel) applications is cited as the target of the project. The major design decisions were made for MIMD execution, of a fine-grain concurrent logic programming base language, on a scalable distributed-memory multiprocessor. The PIMs were also designed around a cluster organization. I recall visiting MCC (where research on virtual shared-memory multiprocessors was being conducted) in spring 1987 and giving a talk describing the organization, and fielding questions about why the design was not more innovative? My reply was that if ICOT could get the organization to work efficiently, that would be sufficiently innovative. I still think the design is plausible; a vote of confidence came later from DASH [48], with a similar organization. But the two-level hierarchy presents an irregular model to mapping and load balancing algorithms that has not yet been conclusively solved by ICOT.

Interestingly, one of MCC's prime directives was to design long-term future systems that were far beyond the planning window of its member companies. Thus for instance advanced human interfaces (including virtual reality) and D. Lenat's common-sense knowledge base CYC [47] were tackled. The FGCS project had a fixed duration whereas

MCC did not, and as ICOT wound down, through the second and third periods, the goals became shorter-term, in an effort to demonstrate working systems to MITI to continue funding. Similarly, the more futuristic projects within MCC were terminated at the first shortage of funds (e.g., the entire parallel processing program, including the virtual shared-memory efforts, which now look so promising). M. Hermenegildo states that, in hindsight, this has proved to be one of MCC's primary weaknesses: that it is privately funded and thus has slowly drifted to very short-term research.

ICOT succeeded in technology transfer to its member companies, even if the final systems were not ideal. In some overlapping areas of interest, MCC may have produced concepts and systems that were on par or superior to those of ICOT, but MCC had a more difficult time transferring the technology to the shareholders. ICOT's success was due to the dedication of many engineers, the investment in a single, integrated computational paradigm, and careful management from conception to execution to transfer. Personnel was transferred as well, ensuring that the technology would not be dead on arrival. The companies contributed resources, towards hardware and software construction, that are quite large by U.S. standards. The formula: top-down concept management cooperating with strong bottom-up construction, is discussed further in the next section.

In my own view, the future of this research area lies in the design and compilation of compositional languages, such as PCN, that attempt to bring logic programming more in-line with mainstream practices. Specific problems that need to be solved (many of these issues are currently active research topics):

- How to reuse memory automatically at low cost? How to retain data locality within distributed implementations?
- How to finesse, with static analysis, many overheads of symbolic parallel processing, such as dereferencing, synchronization, and communication?
- How to efficiently schedule tasks with a combination of static analysis, profiling information and motifs?
- Programming paradigms and compilation techniques for bilingual or compositional programming languages. Efficiently partitioning the control and data structures of a problem into two languages can be difficult.
- Find "killer applications" that combine the power of symbolic manipulation and supercomputer number crunching, to demonstrate the utility of logic programming languages.

- Gain experience with large benchmark suites, which requires standardizing some of these languages.

3.1 Commercial Success and Competitiveness

In this section I will address the validity or commercialization of the processing technologies developed by ICOT, specifically the idea of building a special-purpose multiprocessor to execute a fine-grain concurrent language. This seems to be the main concern of the media, and perhaps the key point upon which ICOT is being evaluated. One could criticize ICOT for attempting to naively leapfrog "fourth generation" RISC-based microprocessor technologies, which continue yearly to grow in performance. Ten years ago, Japanese companies did not have experience developing microprocessor architectures, much less second-generation (superscalar) RISC designs, nor MIMD multiprocessor designs. Building the various PIM machines gave some of the hardware manufacturers limited experience in microprocessor design, although presumably this experience could have been had with a more conventional target.

It should be emphasized that the key goal of the FGCS project was to develop computers for "dynamic and non-uniform large problems" [75]. This is distinctly different from the goal of supercomputing research in the U.S., developing computers for large data-parallel (regular) problems [15]. The result is different design decisions: for example, massively parallel SIMD computers cannot be effectively used to execute non-uniform applications, although some have tried [61]. Neither Japan nor the U.S. misevaluated future goals, but they each saw a part of it, neglecting the other part.

However, I do not disagree with the criticism that a more conventional target could have produced successful commercial technologies *and* influenced human infrastructure (see Section 4). The selection of the FGCS goals were certainly influenced by the individuals in charge, primarily K. Fuchi and K. Furukawa, who had a grand vision of logic programming integrating all aspects of high-performance symbolic problem solving. Human infrastructure development was influenced by these same architects. In some sense the two are connected: engineers were given less freedom to choose the direction of overall research, the top-down technologies being managed from above. This is in sharp contrast to U.S. research groups, which are closer to creative anarchy.

In any case, I believe some unique experience *was* attained in the FGCS Project: that of fabricating tagged, symbolic, parallel architectures. This endeavor covers the same ground as the more bottom-up approach to massively parallel computation, taken by conventional multiprocessor vendors. The overall problem of exploiting massively parallel symbolic computation can be seen from two vantage points. ICOT took the high road, first laying out a foundation of a family of symbolic, concurrent languages, and then attempting to implement them, and building layers of applications around them. U.S. vendors took the low road, first building massively parallel hardware, and then attempting to implement imperative languages, either through parallel library interfaces, through sophisticated compilation, or by language modification. In no instance, to my knowledge, have vendors developed massively parallel software for solving symbolic problems, as has been emphasized throughout the FGCS project. My strongest criticism of the FGCS project, however, was a lack of sufficient concern for compiler technology, something adamantly stressed in the U.S. for the past decade.

It is not surprising that the operating systems community are now developing lightweight threads, what I consider a bottom-up effort. Furthermore, languages such as object-oriented Smalltalk [19] and tuple-based Linda [53] form the cores of recent distributed processing efforts, these kernels developing top-down, similar to the ICOT approach with logic programming. A performance gap currently remains between these top-down and bottom-up approaches. To bridge this gap, significant work needs to be done in compilation, and further hardware design refinement is needed. I think the latter requirement is easier for the Japanese manufacturers than the former. I have always been impressed by the *responsiveness* of hardware development both in the universities and industry. It may be the case that compiler technology has made little progress in the FGCS project because the emphasis was placed elsewhere, on languages and applications. A more subtle reason is the inherent complexity of managing high-level language compilation and hardware development simultaneously. These languages have large semantic gaps that need to be covered, with concurrency adding to the analysis problem.

Let's consider what the situation will be if/when the performance gap between these approaches can be bridged. The key question is then *who will be in the better position*? The top-down approach (taken by ICOT) has advantage of programming and application experience in concurrent and symbolic, high-level languages. The bottom-up approach (predominantly taken by U.S. vendors) has the advantage of using imperative languages that evolved slowly, thus retaining market share. There is no clear answer to this question, but for the sake of illustration, let me rephrase it in terms of two specific technologies: wormhole-routed distributed networks [20] and concurrent constraint languages [44, 85].

I believe both these technologies required significant intellectual efforts to conceptualize, design, implement, and apply in real systems. The former represents a bottom-up technology and the latter a top-down technology. Bottom-up technologies are easier to introduce into designs, e.g., PIM/m [59] incorporates wormhole routing (and can execute GDCC, a constraint language [77]), whereas the Intel machines [49] (and other experimental supercomputers at this level, such as the CM/5 [60] and iWARP [9]) do not yet have implementations of constraint languages, or in fact any vendor-offered languages other than C or Fortran. Perhaps GDCC can be ported to general-purpose multiprocessors, but that is *not* the issue. Where GDCC came from, and where it is going, can only be determined from the foundation of the research expertise gained in its development. This is of course true about routing technologies, but again, bottom-up technologies are more easily imported and adapted. They are also more easily sold because they translate more directly to peak FLOPS, although this can be a grave misstatement of application performance, especially in the domain of nonscientific codes.

In 1991, the U.S. Congress passed the High Performance Computing and Communications (HPCC) initiative [15] stating several "grand challenges," such as climate modeling. These goals were indicative of the research interests and activities already underway in U.S. universities and national laboratories. Furthermore, these challenges echoed current government funding in supercomputer research, such as DARPA's 28% stake in Touchstone development costs [49]. Conspicuously, none of the challenges involved symbolic computation. Equally conspicuous was the lack of numerical computation in the FGCS project goals (the overlapping supercomputing project[6] was more tuned to this goal). Yet again, reasoning that if and when the bottom-up and top-down approaches coincide, one can imagine that these "traditional" number-crunching applications will also run efficiently on the resulting architectures, or that the symbolic and numeric engines will become so inexpensive that they will coexist in hybrid systems. It is perhaps more revealing that software applications to solve both climate modeling and genome mapping are being led by language paradigms with logic programming roots, namely PCN [25] and Lucy [92].

4 A New Generation

The following is a compendium of ICOT's major influences in developing its human capital. During the FGCS'92 conference in Tokyo, I had the opportunity to conduct extensive interviews with hardware engineers participating in the FGCS Project, filling me in on all that had transpired since I had left. All those interviewed were active throughout the length of the project, and thus had a complete perspective. Because they worked primarily in the area of computer architecture, their combined views form one in-depth analysis of ICOT, rather than broad-based analyses.

[6] "High-Speed Computing Systems for Scientific and Technological Uses," 1981–1989.

4.1 Increased Communication

ICOT infrastructure was unique for Japanese research organizations in the early 1980's in that it supplied researchers with various communication channels that normally did not exist in the corporate culture. In the following, I will summarize the forms of communication.

Company-to-company interaction was engendered by the cooperative efforts of engineers centrally headquartered at ICOT. All previous national projects were distributed among manufacturers. It is perhaps due to the general Japanese culture of consensus making that made the central location successful.

The introduction of electronic mail increased international as well as local information flow. This trend was generally occurring throughout Japanese organizations coinciding with, not engendered by, the FGCS project. The creation of electronic networks falls under the auspices of the Ministry of Communication. Because this delegation is separate from education and industry, network infrastructure has been slow to develop in Japan. Even the most advanced universities only developed high-bandwidth networks within the past five years.

Company-to-university interaction was engendered by the Working Groups (WGs) associated with the FGCS project. The WGs were started from the inception of ICOT, with the intent of fostering university and company communication. Initially there were 1–5 groups in the first period of the project, growing to 15 groups in the second period, and about 10 in the third period. Participating universities included Tokyo, Kyushu, Kobe, Kyoto, Keio, and the Tokyo Institute of Technology. As an example, the PIM WG, one of the oldest, meets monthly. My experience, from presenting talks at the PIM WG in 1988 and 1992, was an overly structured format and limited participation. Most participation is from universities in Tokyo (for economic reasons), but student participation is extremely limited, e.g., one or two Todai students might attend. In this respect, I doubt that the WGs were efficient in strengthing ties between industry and universities, for instance compared to the weekly Computer Systems Laboratory seminars held at Stanford University. Yet others disagree with this conclusion, pointing out that cooperation among U.S. universities is limited and among U.S. companies, almost nonexistent.

Interaction between research communities in Japan and abroad was engendered by the high value placed on the publication and presentation of research results. ICOT researchers and international researchers exchanged visits frequently. The exposure gained by young researchers was exceptional, even for Western organizations.

4.2 Post-Graduate Education

ICOT served as a substitute for OJT ("on-the-job training"), and in doing so, graduated a generation of engineer/managers educated in advanced areas of computer science and better able to manage their own groups in the future. The latter point applies to both the engineering management as well as political management, learned by a close relationship with MITI.

An argument can be made that separation of industry and higher education is beneficial to Japan, for instance it delivers engineers to industry ready to be trained in specific company technologies. My personal experience in Japan indicated that this argument is weak. The lack of popularity of graduate studies weakens Japan's ability to do computer science, and therefore to produce advanced long-term technologies. The issue is not so much of *where* advanced academic skills are learned, but that the infrastructure needed to successfully learn includes high communication bandwidth of all forms, and an open forum to discuss the latest research ideas. An indirect result of ICOT was to teach fresh university graduates (mainly with B.S. degrees) how to properly conduct research and construct hardware and software systems. Furthermore, experience of presenting papers at conferences gave the individuals much needed practice at social interaction with the international community.

Few ICOT researchers entered service with PhDs. Over the life of the project, about ten PhDs were granted for FGCS-related research (e.g., [91, 82]). This side-effect was unusual for national projects, indicating ICOT's emphasis on basic research, as well as more practical considerations of personal advancement: a large percentage of those completing PhDs became university professors.

My stay at ICOT, and the University of Tokyo, as well as various visits to industry, indicated that ICOT's infrastructure was carefully planned to bring about these results. Detail was paid to basic things, like the ICOT library which was quite extensive. More than any other workplace I have experienced, communication between engineers, unprotected by offices or cubicals, was extensive. Finding an expert for consultation was as simple as crossing the room.

4.3 Company Cultures

I believe that ICOT coincided with greater forces within Japan causing a movement away from the culture of lifetime employment. However, the revolution was certainly felt within the FGCS Project. A. Goto of NTT estimates that over 5% of all ICOT participants changed their affiliations after their tenure ended. Examples include moves from industry and the national laboratories to academia (both as professors and as researchers), and moves between companies. The former constituted the major group.

In general the most highly-productive researchers made the moves. ICOT may have implicitly influenced the individuals by empowering them to conduct world-class research programs. Once successful, they reevaluated their opportunity costs, which were not being adequately met by their employers. These costs involved both salary, as well as intellectual freedom. The explosion came as a surprise to the companies, which it should not have, given the highly technical nature of computer science. Certain companies took direct action to deal with it, such as SONY forming the Computer Science Laboratory (CSL), a small Western-style research lab in Tokyo. NEC took indirect action by forming a research laboratory in New Jersey.

In addition, the universities gained a significant number of professors "generated" at ICOT. K. Nakajima of Mitsubishi estimates this at about five directly from ICOT, and six from the ICOT-related groups within industry. Perhaps this was an accidental side-effect of the decade, but it certainly was not seen in the previous national projects.

An opposite effect, to the previous "explosion," was the cross-fertilization of company cultures. ICOT played a role of matchmaker to manufacturers, resulting in technology transfers, however indirect or inadvertent, over the ten years. Here I review two main transfers: engineering management techniques and multiprocessor technologies.

Large systems development, such as the PIM development efforts, required scheduling. Nakajima pointed out that ICOT would pool the scheduling techniques from the member companies without bias. This resulted in more efficient scheduling and project completion. Even if the companies themselves did not adopt the hybrid methodologies, the individuals involved certainly learned.

ICOT was a mixture of manufacturers and their engineers, and the experience of introducing these groups was beneficial to all. The engineers are exposed to how things are done in other companies. K. Kumon of Fujitsu stressed that PIM/p [46] and PIM/m [59] could not *both* be built by *both* Fujitsu and Mitsubishi — each manufacturer had its own technology expertise, and thus the designs evolved. Designers from both companies learned, first hand, alternatives that were not (yet) feasible in their own environments.

5 Conclusions

Considering technology, I conclude that the top-down, vertically integrated approach in the FGCS project failed to achieve a revolution, but was a precursor to evolutionary advances in the marketplace. The Japanese supercomputing project involved vector processor technology that was well understood compared to symbolic computation. Thus the projects cannot be compared on that basis. Furthermore, comparisons to U.S. research efforts, which are driven by strong national laboratories and universities in the direction of "traditional" scientific computation, is also inappropriate. Perhaps further comparison of the FGCS and HPCC projects would be appropriate, but a subject of another paper. U.S. research and development concerning symbolic processing in Lisp and Smalltalk might be the most valid benchmark, if comparisons are desired. The rise and fall of the Lisp machine market, over this decade, does not place the U.S. in a more successful light. Refinement, rather than abandonment, of the concepts developed in the FGCS project may well serve the Japanese manufacturers in the upcoming decade.

Considering human capital, I think all the influences cited in this article are natural results of "market forces." The action of these influences on young ICOT researchers were by and large positive. Increased communication between engineers, managers, professors, students, and government bureaucrats leads to more rapid progress in developing basic research ideas into successful commercial products.

The question remains as to whether a National Project of this magnitude is necessary to create these human networks each generation, or if this first network will propagate itself without help from another project. An optimistic view has the networks weakening with age, but remaining in place. Thus in the future, it may not require such a grand-scale project to strengthen ties. For example, current ICOT graduates, understanding the importance of free and flexible discussion of results at national conferences, will increase the participation of the researchers in their care, thus enabling the next generation to form their own friendships and working relationships.

However, few ICOT people believe this scenario. Some believe that most ICOT researchers *implicitly* understand the importance of ICOT's contributions in this area, but not *explicitly*. Without explicit self-awareness, this meta-knowledge may be lost without another national project, or an equivalent, to reinforce the lessons. The current generation of engineers, without an experience similar to ICOT, will be at a disadvantage to the ICOT generation. Communication will be strictly limited to technical conferences, where information flow is restricted. In this sense, ICOT did not create a revolution because it did not fundamentally change the manufacturers. The human networks will *not* be self-generating from the bottom-up, by the few seedling managers trained at ICOT. Although a manager's own bias may be consistent with ICOT's flexible style of research and management, the higher one gets in the company hierarchy, the less managers tend to share this sentiment.

Either another project, or a radical restructuring of the diametric cultures of education and industry, will be required to propagate the advances made in the FGCS project. The Japanese, certainly amenable to hedging their bets, have already started upon both avenues. A "sixth generation" project involving massive parallelism, neural networks, and optical technologies is already underway. However, the research is distributed among many institutions, potentially lessening its impact. Furthermore, the Ministry of Education is currently making plans to approximately double funding for basic research in the universities [87].

On a more personal note, I highly respect the contribution made by the FGCS project in the academic development of the field of symbolic processing, notably implementation and theory in logic programming, constraint and concurrent languages, and deductive and object-oriented databases. In my specific area of parallel logic programming languages, architectures, and implementations, ICOT made major contributions, but perhaps the mixed schedule of advanced technology transfer and basic research was ill-advised.

This basic research also led to a strong set of successful applications, in fields as diverse as theorem proving and biological computation. In a wider scope, the project was a success in terms of the research it engendered in similar international projects, such as ALVEY, ECRC, ESPRIT, INRIA, and MCC. These organizations learned from each other and their academic competitiveness in basic research pushed them to achieve a broader range of successes. In this sense, the computer science community is very much indebted to the "fifth generation" effort.

Sometime during my stay in Tokyo, I was invited by some U.S. Congressmen to a breakfast meeting at a plush Roppongi hotel one Sunday morning. It was so early, I had to attend directly from Saturday night socializing, leaving me somewhat weakened. However, I was clearheaded enough to listen to the voices around the table stating what was wrong with the electronics/computer trade imbalance. I was perhaps the only attendee who was not a salesman or a politician, and certainly the only one who wasn't quite groking that Big American Breakfast sitting in front of me. When it was my turn to speak, I couldn't think of much to say: the issues were as large as billions in chip dumping and unfair markets, not collaborative research efforts. Well, collaboration is of long-term importance, I thought. The same as the basic research itself.

Acknowledgements

The author is now supported by an NSF Presidential Young Investigator award, with matching funds from Sequent Computer Systems Inc. My stay at ICOT was generously supported by Y. T. Chien and A. DeAngelis of the National Science Foundation. I would like to thank the numerous people who graciously aided me in writing this article.

References

[1] S. Aikawa, M. Kamiko, H. Kubo, F. Matsuzawa, and T. Chikayama. ParaGraph: A Graphical Tuning Tool for Multiprocessor Systems. In *International Conference on Fifth Generation Computer Systems*, pages 286–293, Tokyo, June 1992. ICOT.

[2] K. A. M. Ali and R. Karlsson. The Muse Or-Parallel Prolog Model and its Performance. In *North American Conference on Logic Programming*, pages 757–776. Austin, MIT Press, October 1990.

[3] H. Alshawi and D. B. Moran. The Delphi Model and Some Preliminary Experiments. In *International Conference and Symposium on Logic Programming*, pages 1578–1589. University of Washington, MIT Press, August 1988.

[4] S. Asano, S. Isobe, and H. Sakai. The Unique Features of PIM/k: A Parallel Inference Machine with Hierarchical Cache System. Technical Report TR-767, ICOT, 1-4-28 Mita, Minato-Ku Tokyo 108, Japan, April 1992.

[5] Association for Logic Programming. Applications of Prolog International Conference and Exhibition. London, April 1992.

[6] R. Bahgat and S. Gregory. Pandora: Non-deterministic Parallel Logic Programming. In *International Conference on Logic Programming*, pages 471–486. Lisbon, MIT Press, June 1989.

[7] A. Beaumont, S. Muthu Raman, P. Szeredi, and D. H. D. Warren. Flexible Scheduling of Or-Parallelism in Aurora: The Bristol Scheduler. In *PARLE91: Conference on Parallel Architectures and Languages Europe*, pages 403–420. Springer Verlag, June 1991.

[8] P. Biswas, S-C. Su, and D. Y. Y. Yun. A Scalable Abstract Machine Model to Support Limited-OR (LOR)/Restricted-AND Parallelism in Logic Programs. In *International Conference and Symposium on Logic Programming*, pages 1160–1179. University of Washington, MIT Press, August 1988.

[9] S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H. T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. S. Tseng, J. Sutton, J. Urbanski, and J. Webb. iWARP: An Integrated Solution to High-Speed Parallel Computing. In *International Supercomputing Conference*, pages 330–339, Orlando, November 1988. IEEE Computer Society.

[10] P. Brand, S. Haridi, and D.H.D. Warren. Andorra Prolog—The Language and Application in Distributed Simulation. *New Generation Computing*, 7(2–3):109–125, 1989.

[11] R. Butler, T. Disz, E. L. Lusk, R. Olson, R. A. Overbeek, and R. Stevens. Scheduling OR-Parallelism: an Argonne Perspective. In *International Conference and Symposium on Logic Programming*, pages 1565–1577. University of Washington, MIT Press, August 1988.

[12] A. Calderwood and P. Szeredi. Scheduling Or-Parallelism in Aurora. In *International Conference on Logic Programming*, pages 419–435. Lisbon, MIT Press, June 1989.

[13] M. Carlsson. *Design and Implementation of an OR-Parallel Prolog Engine*. PhD thesis, Royal Institute of Technology, SICS Dissertation Series 02, March 1990.

[14] T. Chikayama and Y. Kimura. Multiple Reference Management in Flat GHC. In *International Conference on Logic Programming*, pages 276–293. University of Melbourne, MIT Press, May 1987.

[15] Committee on Physical, Mathematical, and Engineering Sciences. Grand Challenges: High Performance Computing and Communication. NSF, 1800 G Street, N.W., Washington D.C., 1991.

[16] J. S. Conery. Binding Environments for Parallel Logic Programs in Non-shared Memory Multiprocessors. *International Journal of Parallel Programming*, 17(2):125–152, April 1988.

[17] V. S. Costa, D. H. D. Warren, and R. Yang. The Andorra-I Engine: A Parallel Implementation of the Basic Andorra Model. In *International Conference on Logic Programming*, pages 825–839. Paris, MIT Press, June 1991.

[18] J. A. Crammond. The Abstract Machine and Implementation of Parallel Parlog. *New Generation Computing*, August 1992.

[19] W. J. Dally. A Universal Parallel Computer Architecture. In *International Conference on Fifth Generation Computer Systems*, pages 746–758, Tokyo, June 1992. ICOT.

[20] W. J. Dally and C. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[21] D. DeGroot. Restricted AND-Parallelism. In *International Conference on Fifth Generation Computer Systems*, pages 471–478, Tokyo, November 1984. ICOT.

[22] T. P. Dobry. *A High Performance Architecture for Prolog.* Kluwer Academic Publishers, Norwell MA, 1988.

[23] S. Duvvuru, R. Sundararajan, E. Tick, A. V. S. Sastry, L. Hansen, and X. Zhong. A Compile-Time Memory-Reuse Scheme for Concurrent Logic Programs. In *International Workshop on Memory Management*, pages 264–276, St. Malo, September 1992. ACM Press.

[24] I. Foster. Logic Operating Systems: Design Issues. In *International Conference on Logic Programming*, pages 910–926. University of Melbourne, MIT Press, May 1987.

[25] I. Foster, R. Olson, and S. Tuecke. Productive Parallel Programming: The PCN Approach. *Scientific Programming*, 1(1), 1992.

[26] I. Foster and S. Taylor. *Strand: New Concepts in Parallel Programming.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[27] I. Foster and S. Taylor. A Compiler Approach to Scalable Concurrent Program Design. Technical Report MCS-P306-0492, Argonne National Laboratory, 1992.

[28] I. Foster and W. Winsborough. Copy Avoidance through Compile-Time Analysis and Local Reuse. In *International Symposium on Logic Programming*, pages 455–469. San Diego, MIT Press, November 1991.

[29] K. Furukawa. Summary of Basic Research Activities of the FGCS Project. In *International Conference on Fifth Generation Computer Systems*, pages 20–32, Tokyo, June 1992. ICOT.

[30] A. Goto, Y. Kimura, T. Nakagawa, and T. Chikayama. Lazy Reference Counting: An Incremental Garbage Collection Method for Parallel Inference Machines. In *International Conference and Symposium on Logic Programming*, pages 1241–1256. University of Washington, MIT Press, August 1988.

[31] A. Goto, A. Matsumoto, and E. Tick. Design and Performance of a Coherent Cache for Parallel Logic Programming Architectures. In *International Symposium on Computer Architecture*, pages 25–33. Jerusalem, IEEE Computer Society, May 1989.

[32] D. Gudeman, K. De Bosschere, and S. K. Debray. `jc`: An Efficient and Portable Sequential Implementation of Janus. In *Joint International Conference and Symposium on Logic Programming*. Washington D.C., MIT Press, November 1992.

[33] J. L. Hennessy and D. A. Patterson. *Computer Architecture A Quantitative Approach*. Morgan Kaufmann Publishers Inc., 1990.

[34] M. V. Hermenegildo. An Abstract Machine for Restricted AND-parallel Execution of Logic Programs. In *International Conference on Logic Programming*, number 225 in Lecture Notes in Computer Science, pages 25–40. Imperial College, Springer-Verlag, July 1986.

[35] M. V. Hermenegildo. Research on Parallel Logic Language Implementation and Architecture at ICOT. In *The Research Exchange Report: Intermediate Stage (1985 to 1988)*, pages 157–174. ICOT, Tokyo, 1992.

[36] M. V. Hermenegildo and K. Greene. &-Prolog and its Performance: Exploiting Independent And-Parallelism. In *International Conference on Logic Programming*, pages 253–268. Jerusalem, MIT Press, June 1990.

[37] M. V. Hermenegildo and E. Tick. Memory Performance of AND-Parallel Prolog on Shared-Memory Architectures. In *International Conference on Parallel Processing*, Penn State, August 1988.

[38] K. Hirata, R. Yamamoto, A. Imai, H. Kawai, K. Hirano, T. Takagi, K. Taki, A. Nakase, and K. Rokusawa. Parallel and Distributed Implementation of Concurrent Logic Programming Language KL1. In *International Conference on Fifth Generation Computer Systems*, pages 436–459, Tokyo, June 1992. ICOT.

[39] C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[40] N. Ichiyoshi, K. Rokusawa, K. Nakajima, and Y. Inamura. A New External Reference Management and Distributed Unification for KL1. In *International Conference on Fifth Generation Computer Systems*, pages 904–913, Tokyo, November 1988. ICOT.

[41] A. Imai and E. Tick. Evaluation of Parallel Copying Garbage Collection on a Shared-Memory Multiprocessor. *IEEE Transactions on Parallel and Distributed Computing*, 1992. in press.

[42] Y. Inamura, N. Ichiyoshi, K. Rokusawa, and K. Nakajima. Optimization Techniques Using the MRB and Their Evaluation on the Multi-PSI/V2. In *North American Conference on Logic Programming*, pages 907–921. Cleveland, MIT Press, October 1989.

[43] Y. Inamura and S. Onishi. A Detection Algorithm of Perpetual Suspension in KL1. In *International Conference on Logic Programming*, pages 18–30. Jerusalem, MIT Press, June 1990.

[44] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *SIGPLAN Symposium on Principles of Programming Languages*, Munich, 1987. ACM Press.

[45] L. Kale. Parallel Execution of Logic Programs: the REDUCE-OR Process Model. In *International Conference on Logic Programming*, pages 616–632. University of Melbourne, MIT Press, May 1987.

[46] K. Kumon, A. Asato, S. Arai, T. Shinogi, A. Hattori, H. Hatazawa, and K. Hirano. Architecture and Implementation of PIM/p. In *International Conference on Fifth Generation Computer Systems*, pages 414–424, Tokyo, June 1992. ICOT.

[47] D. B. Lenat, M. Prakash, and M. Shepherd. CYC: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks. *AI Magazine*, Winter 1985.

[48] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy. The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor. In *International Symposium on Computer Architecture*, pages 148–159. Seattle, IEEE Computer Society, May 1990.

[49] S. L. Lillevik. The Touchstone 30 Gigaflop DELTA Prototype. In *International Conference on Supercomputing*, pages 671–677. IEEE Computer Society, 1991.

[50] Y. J. Lin and V. Kumar. AND-Parallel Execution of Logic Programs on a Shared Memory Multiprocessor: A Summary of Results. In *International Conference and Symposium on Logic Programming*, pages 1123–1141. University of Washington, MIT Press, August 1988.

[51] E. Lusk, R. Butler, T. Disz, R. Olson, R. Overbeek, R. Stevens, D. H. D. Warren, A. Calderwood, P. Szeredi, S. Haridi, P. Brand, M. Carlsson, A. Ciepielewski, and B. Hausman. The Aurora Or-Parallel Prolog System. In *International Conference on Fifth Generation Computer Systems*, pages 819–830, Tokyo, November 1988. ICOT.

[52] J. Makino. On an O(NlogN) Algorithm for the Gravitational N-Body Simulation and its Vectorization. In *Proceedings of the First Appi Workshop on Supercomputing*, pages 153–168, Tokyo, 1987. Institute of Supercomputing Research. ISR Technical Report 87-03.

[53] Network Linda Creates Supercomputers with Links. *MIPS World*, 1(3). December 1991.

[54] K. Muthukumar and M. Hermenegildo. Determination of Variable Dependence Information Through Abstract Interpretation. In *North American Conference on Logic Programming*, pages 166–188. Cleveland, MIT Press, October 1989.

[55] L. Naish. Parallelizing NU-Prolog. In *International Conference and Symposium on Logic Programming*, pages 1546–1564. University of Washington, MIT Press, August 1988.

[56] T. Nakagawa, N. Ido, T. Tarui, M. Asaie, and M. Sugie. Hardware Implementation of Dynamic Load Balancing in the Parallel Inference Machine PIM/c. In *International Conference on Fifth Generation Computer Systems*, pages 723–730, Tokyo, June 1992. ICOT.

[57] K. Nakajima. Piling GC: Efficient Garbage Collection for AI Languages. In *IFIP Working Conference on Parallel Processing*, pages 201–204. Pisa, North Holland, May 1988.

[58] K. Nakajima. Distributed Implementation of KL1 on the Multi-PSI. In *Implementation of Distributed Prolog*. John Wiley & Sons, Ltd., Sussex England, 1992.

[59] H. Nakashima, K. Nakajima, S. Kondo, Y. Takeda, Y. Inamura, S. Onishi, and K. Masuda. Architecture and Implementation of PIM/m. In *International Conference on Fifth Generation Computer Systems*, pages 425–435, Tokyo, June 1992. ICOT.

[60] PR Newswire. Thinking Machines Corporation Announces CM-5 Supercomputer. October 29, 1991.

[61] M. Nilsson and H. Tanaka. Massively Parallel Implementation of Flat GHC on the Connection Machine. In *International Conference on Fifth Generation Computer Systems*, pages 1031–1040. Tokyo, ICOT, November 1988.

[62] K. Nishida, Y. Kimura, A. Matsumoto, and A. Goto. Evaluation of MRB Garbage Collection on Parallel Logic Programming Architectures. In *International Conference on Logic Programming*, pages 83–95. Jerusalem, MIT Press, June 1990.

[63] K. Nitta, K. Taki, and N. Ichiyoshi. Experimental Parallel Inference Software. In *International Conference on Fifth Generation Computer Systems*, pages 166–190, Tokyo, June 1992. ICOT.

[64] K. Rokusawa and N. Ichiyoshi. A Scheme for State Change in a Distributed Environment Using Weighted Throw Counting. In *Interational Parallel Processing Symposium*, pages 640–645, Beverly Hills, March 1992. IEEE Computer Society.

[65] K. Rokusawa, N. Ichiyoshi, T. Chikayama, and H. Nakashima. An Efficient Termination Detection and Abortion Algorithm for Distributed Processing Systems. In *International Conference on Parallel Processing*, pages 18–22, Penn State, August 1988.

[66] V. A. Saraswat. *Concurrent Constraint Programming.* MIT Press, Cambridge MA, 1990.

[67] V. A. Saraswat, K. Kahn, and J. Levy. Janus: A Step Towards Distributed Constraint Programming. In *North American Conference on Logic Programming*, pages 431–446. Austin, MIT Press, October 1990.

[68] M. Sato, K. Kato, K. Takeda, and T. Oohara. Exploiting Fine Grain Parallelism in Logic Programming on a Parallel Inference Machine. Technical Report TR-676, ICOT, 1-4-28 Mita, Minato-ku Tokyo 108, Japan, August 1991.

[69] E. Y. Shapiro, editor. *Concurrent Prolog: Collected Papers*, volume 1,2. MIT Press, Cambridge MA, 1987.

[70] E. Y. Shapiro. The Family of Concurrent Logic Programming Languages. *ACM Computing Surveys*, 21(3):413–510, September 1989.

[71] P. Smith. Letter from Japan. *The New Yorker*, pages 89–99. April 13, 1992.

[72] L. Sterling, editor. *The Practice of Prolog.* MIT Press, Cambridge MA, 1990.

[73] P. Szeredi. Performance Analysis of the Aurora Or-Parallel Prolog System. In *North American Conference on Logic Programming*, pages 713–732. Cleveland, MIT Press, October 1989.

[74] A. Takeuchi, K. Takahashi, and H. Shimizu. A Description Language with AND/OR Parallelism for Concurrent Systems and its Stream-Based Realization. Technical Report 229, ICOT, 1-4-28 Mita, Minato-ku Tokyo 108, Japan, February 1987.

[75] K. Taki. Parallel Inference Machine PIM. In *International Conference on Fifth Generation Computer Systems*, pages 50–72, Tokyo, June 1992. ICOT.

[76] A. Taylor. LIPS on a MIPS: Results From a Prolog Compiler for a RISC. In *International Conference on Logic Programming*, pages 174–185. Jerusalem, MIT Press, June 1990.

[77] S. Terasaki, D. J. Hawley, H. Sawada, K. Satoh, S. Menju, T. Kawagishi, N. Iwayama, and A. Aiba. Parallel Constraint Logic Programming Language GDCC and its Parallel Constraint Solvers. In *International Conference on Fifth Generation Computer Systems*, pages 330–346, Tokyo, June 1992. ICOT.

[78] E. Tick. *Memory Performance of Prolog Architectures.* Kluwer Academic Publishers, Norwell MA, 1987.

[79] E. Tick. A Performance Comparison of AND- and OR-Parallel Logic Programming Architectures. In *International Conference on Logic Programming*, pages 452–470. Lisbon, MIT Press, June 1989.

[80] E. Tick. *Parallel Logic Programming.* MIT Press, Cambridge MA, 1991.

[81] E. Tick and K. Susaki. TOESP: A Prolog Benchmark. Technical Memo TM-451, ICOT, 1-4-28 Mita, Minato-ku Tokyo 108, Japan, January 1988.

[82] K. Ueda. *Guarded Horn Clauses.* PhD thesis, University of Tokyo, March 1986.

[83] K. Ueda and M. Morita. A New Implementation Technique for Flat GHC. In *International Conference on Logic Programming*, pages 3–17. Jerusalem, MIT Press, June 1990.

[84] M. Van Caneghem and D. H. D. Warren, editors. *Logic Programming and Its Applications.* Ablex, 1986.

[85] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* MIT Press, Cambridge MA, 1989.

[86] P. L. Van Roy and A. M. Despain. High-Performace Logic Programming with the Aquarius Prolog Compiler. *IEEE Computer Magazine*, pages 54–68, January 1992.

[87] K. Wada. Institute of Information Sciences and Electronics, Tsukuba University, personal communication, June 1992.

[88] D. H. D. Warren. *Applied Logic — Its Use and Implementation as a Programming Tool.* PhD thesis, University of Edinburgh, 1977. Also available as SRI Technical Note 290.

[89] M. J. Wise. *Prolog Multiprocessors.* Prentice Hall, Englewood Cliffs, NJ, 1987.

[90] K. Yanoo. An Optimizing Compiler for a Parallel Inference Language. In H. Tanaka, editor, *Annual Report of the Research on Parallel Inference Engine*, pages 71–94. University of Tokyo, April 1992. (in Japanese).

[91] K. Yoshida. *A'UM: A Stream-Based Concurrent Object-Oriented Programming Language.* PhD thesis, Keio University, March 1990.

[92] K. Yoshida, C. Smith, T. Kazic, G. Michaels, R. Taylor, D. Zawada, R. Hagstrom, and R. Overbeek. Toward a Human Genome Encyclopedia. In *International Conference on Fifth Generation Computer Systems*, pages 307–320, Tokyo, June 1992. ICOT.