Report on Visit to ICOT
July 18th - August 4th 1989

Jim Crammond

Dept. of Computing
Imperial College, London

Introduction

I was invited to ICOT by Dr. Chikayama after he visited Imperial College in April to exchange experiences with implementation of committed-choice languages on parallel machines - I have been developing the Parallel Parlog system that runs on shared memory machines such as the Sequent Symmetry and many issues relevant to the tightly coupled (intra-cluster) part of the PIM machine also arise in the implementation of the Parallel Parlog system.

My main objectives of this three week visit were (a) to get an overview of the parallel implementations being developed at ICOT and how they are progressing and (b) to carry out a performance comparison between my Parlog system and the parallel KL1 system which runs on the Sequent Symmetry, (c) to exchange ideas on problems common to our implementation efforts; particularly in memory management and scheduling.

This report describes some of the work carried out and discussions I had on these matters during my stay.

Overview of ICOT Implementations

There are many KL1 systems at ICOT! On the Sequent Symmetry there is the sequential system PDSS which is the main one used to write programs - this system has a well developed environment with a micro-PIMOS user-interface, tools for running within emacs and debugging facilities. I hope to have time for a proper demonstration of this system before I go! I understand that this system is also compatible with the KL1 systems on the PSI and multi-PSI machines (although this may change).

The main system being developed at present is the VPIM (Virtual PIM) system, written in PSL - an in-house variant of the C language which will enable real registers to be specified on real PIM machines. A PSL to C translator exists so that VPIM can be compiled on the Symmetry using the C compiler in the usual way. This system runs in parallel on the Symmetry but it is a highly instrumented emulator and includes features such as MRB garbage reclamation which will be supported in hardware/firmware on real PIM machines, so it is pretty slow. PSL compilers for the 5 different PIM machines will be written by the manufacturers of these machines.

ICOT now has multi-PSI machines including a 64 processor machine. This runs the PIMOS operating system and has 4 front-end processors - PSI machines running SIMPOS. Parallel applications are generally developed on PDSS first and then copied to the multi-PSI machine, perhaps via the PSI machine. Microcode compiled on the PSI machine can run on the multi-PSI machine

but PDSS has a better debugging environment for KL1 than the PSI machine. This should obviously change as the multi-PSI PIMOS environment improves.

The third system on the Symmetry is the KL1-ps system (invoked as clsim). This system is a parallel KL1 system which is designed to run reasonably efficiently on the Symmetry - it was this last system with which I made my comparisons with the Parlog system. KL1-ps provides a very basic user-interface - this system was apparently written mostly by Masatoshi Sato who left ICOT some time ago and no further work has been done on this system other than performance measurements by Evan Tick for his comparisons with the Aurora Or-Parallel Prolog system.

Performance Studies

When I arrived at ICOT I already had 6 benchmark programs which had been developed and used by Evan Tick in his technical report [2][1] which I had adapted from FGHC to (Flat) Parlog and ran on the parallel Parlog system. However, Evan had made significant improvements to most of these programs and had produced a number of other benchmark programs so we selected 10 of his new KL1 benchmarks and I translated these into Parlog for us to make our comparisons.

Before getting down to making comparisons, I decided to rewrite the garbage collector in the Parlog system to use a semi-space copying algorithm (which is also used by the KL1-ps system). With the new set of benchmarks I was able to accumulate some statistics about the new garbage collector and compare it with my old (sliding) algorithm. It does run faster - but requires more space, of course.

Evan and I measured execution times for the benchmarks on 1, 2, 4, 8 and 12 processors[2] and also the number of reductions performed. The reduction counts were essentially the same on both systems; the timing results, however, did not show a clear pattern. For six of the benchmarks Parlog was 20-45% faster; it was 70% faster in one case and then slower in the last two cases: *semigroup* was 15% slower and *zebra* was 30% slower.

I spent some time investigating the two benchmarks which were slower; the first case was due to spending a lot of time in the garbage collector. The KL1-ps system has only two options for memory (heap) allocation - 4Mb and 40Mb(!). The *semigroup* benchmark would only run with the large memory option, but would then do no garbage collections. By increasing the heap size in the Parlog system (which can be set to any size) so that only two garbage collections were done, the Parlog system was the faster than KL1-ps by 5%.

The *zebra* benchmark is more complicated to explain. A large factor of the poor performance is due to the lack of clause indexing in the Parlog system - this was evident when abstract machine instruction counts were compared: the Parlog system executed 2.4 times more instructions than KL1-ps. Another improvement could be made to the Parlog version of this benchmark using sequential conjunction operator to reduce the number of calls to the scheduler - the two improvements combined made Parlog run 5% faster than the KL1-ps version; though we

[1] this report contained timings for a "model A" Symmetry. I had only run my benchmarks on a "model B" Symmetry with the improved cache algorithm and so wished to run all benchmarks on the same machine. Both Symmetries at ICOT are now model B Symmetry's.

[2] obtaining semi-reliable figures for 12 processors was the hardest as the Symmetry was often busy with many users active on the machine!

still should examine more closely why KL1-ps is performing so well.

I had discussions with Goto-san about the memory organisation and word formats in KL1-ps which is quite different in some respects to the Parlog system. Two key differences are that (a) the KL1 system uses 2 words (8 bytes) for each heap cell, compared with 1 word (4 bytes) for the Parlog system and (b) the KL1 system has one area for storing goal records (processes) and heap terms, while Parlog uses separates data areas for processes, goal arguments and heap terms. My garbage collector thus performed perhaps 3 times faster than KL1-ps but only operated on the heap. It also did not perform so well if the *goal argument stack* was large (containing large holes indicating garbage) and the active heap was small. Such characteristics were displayed by two of the benchmarks and after discussions with Goto-san and Imai-san I decided to make changes to my system to enable this stack to be garbage collected.

Of course, we also compared speed-ups, but found that (apart from one benchmark) these were about the same on both systems - hopefully we can investigate further the one notable exception. After discussions with Chikayama-san, Goto-san and Imai-san about the scheduling algorithm on KL1-ps I had expected the Parlog system to produce slightly better speedups than the KL1-ps system, but this was not so - at least on the 8 processors figures. Briefly, the main difference between their schedulers is that an idle processor in Parlog will search other processors run queues for work, while in the KL1-ps system it will set a flag and *wait* for a busy processor to give it work (at the next reduction step). I would like to compare speed ups on 12 or 15 processors, though.

This comparison work, though still incomplete, has been extremely useful for finding some of the weaknesses and strengths of the two systems. For example, after comparing the figures for abstract machine instructions executed, Evan Tick examined the KL1-b code for the benchmark where KL1-ps was 70% slower since it was executing more instructions than Parlog. This reveiled that the compiler was doing a poor job on certain constructs - this compiler is also used in the PDSS and VPIM systems, so perhaps should be investigated further. In general Parlog executed 30% more instructions for the benchmarks where it was 30% faster - this could well be due to better engineering of the C emulator. I feel this work has also highlighted the importance for having many reasonably non-trivial benchmarks to test such systems.

Presentations and Discussions

I gave two presentations during my stay; one on the second day at ICOT and one on the second-last day at ICOT. In the first talk I presented an overview of the Parallel Parlog system, giving a high level description of the internals of the Parlog emulator, and also described some experiments I had carried out earlier this year on with different scheduling algorithms.

My second talk presented the results of my work at ICOT, described above, in comparing the Parlog and KL1 systems and I described some of the main known differences between the two systems. I had hoped that this presentation would in fact be more of a discussion with plenty of audience participation and I was not disappointed! We covered many aspects of the different systems and also thought of further statistics to examine.

Various members of the VPIM team in the 4th Laboratory presented outlines of their work to me; in particular, Imai-san presented an overview of the PIM project and of the VPIM, and Kawai-san discussed the Shoen structure to be implemented on the VPIM.

Taki-san showed me a video of the multi-PSI demonstrations used in the FGCS conference last year and Kawai-san and Furuichi-san of Mitsubishi were able to show me live demonstrations on the 64 processor multi-PSI, all of which was most interesting.

I has a few discussions with Chikayama-san, Goto-san and Imai-san about scheduling in the VPIM, which essentially extends the algorithm used in KL1-ps by having prioritised run queues. I also had fruitful discussions with Goto-san and Imai-san about memory management and some ideas I have for a future Parlog version, which may also be of use in the tightly coupled part of the VPIM system.

Comments

I think the introduction of multi-PSI machine has encouraged more work on parallel applications in ICOT than has been done before - it is always nice to run programs on the real thing - and this is good news.

I have the impression, however, that it is not so easy to write programs that run well on the multi-PSI as expected. I have seen clever tricks in programs such using extra arguments to goals to produce extra copies of terms which in turn reduces the number of remote references (and so improve speed) when such goals are moved to different processors. In addition I note that the programmer has to write the code to distribute the work load (either by static or dynamic load balancing). Hopefully as the PIMOS environment improves, such things will become easier.

The multi-PSI system already has a nice graphical tool to show how much time each processor is spending on executing a problem; but this gives no indication of *what* goal are being executed *where*. I believe it is important to develop tools for aiding *applications writers* to understand how their program is being executed on a parallel machine. This is equally important for the Parlog system and the tightly coupled part of the PIM machine.

What I have in mind is a tool equivalent to the WAM Trace facility [1] developed for the Aurora system. This analyses a post-mortem dump of a parallel execution run and animates how the search tree was explored by the processors. This can often help a programmer understand why his program does not speed up in the way he expected, or how it can be improved. This particular facility is not directly applicable to committed-choice languages, so we need to consider what kind of information would help the application writer understand how his program is executing. Also, we may prefer to attempt a real time analysis of what is going on (like the existing multi-PSI tool) rather than a post-mortem dump. Of course, any such tool should be graphical, so that we can use it in demonstrations!

I was surprised that (apart from Evan Tick who frequently visits ICOT) no one is writing parallel programs to run on the KL1-ps system. Admittedly it is a primitive system with no debugging aids, but I think more complex applications are needed to test the tightly coupled component of the PIM design and currently VPIM is too slow to use for developing applications. Perhaps a more performance oriented version of VPIM should be produced for this purpose.

I suspect that having to deal with five different flavours of PIM machine has added some unwelcome complexity to the VPIM design and implementation effort being done at ICOT - VPIM having to be suitably portable for all five machines. I hope this does not hinder the progress towards the completion of the final stage of ICOT's plan.

Acknowledgements

I found ICOT to have a very friendly environment in which to conduct research and greatly enjoyed my stay here. I would firstly like to thank Iwata-san and Karakawa-san for arranging my accomodation here in Tokyo, which was excellent.

I would like to thank Evan Tick for helping me with some of the comparison work I have carried out at ICOT. I would like to thanks all those members of the 4th Laboratory with whom I had many interesting discussions, lunches, dinners (Kawai-san in particular deserves much praise for his efforts in translating many Japanese menus for me!). Thanks also to the table tennis players for inviting me to play!

Special thanks to Imai-san for showing me the sights of Kyoto and Nara last weekend and arranging my stay in a Japanese Inn which was excellent. Finally I would like to thank Fuchi-san, Uchida-san and Chikayama-san for inviting me to ICOT.

References

1. T. Disz and E. Lusk, "A Graphical Tool For Observing The Behaviour of Parallel Logic Programs," pp. 46-53 in *1987 Symposium on Logic Programming*, Computer Society Press ().
2. E. Tick, "Performance of Parallel Logic Programming Architectures," Technical Report TR-421, ICOT (September 1988).

名前:　James A. Crammond (通称 Jim Crammond)
所属:　ロンドン大学インペリアルカレッジ計算学部 (ロンドン)
Dept. of Computing
Imperial College of Science and Technology
University of London
(研究助手)
番地:　180 Queen's Gate, London SW7 2BZ
Telephone:　01-589-5111
Telex:　261503

学位:　1988 年 PhD. 取得
ヘリオットワット大学計算機科学部 (エジンバラ)
Department of Computer Science
Heriot-Watt University

論文題目:
"Implementation of Committed Choice Logic Languages on Shared Memory Multiprocessors"

内容:
Parlog, FCP, GHC などの並列論理型言語をメモリ共有型のマルチプロセサ上に実装する方式に関する研究。

現在の研究分野:
学位取得時の研究 (並列論理型言語の実装方式) を継続している。

これまでの主な発表論文:

☆ J.A.Crammond and C.D.F.Miller,
"An Architecture for Parallel Logic Languages",
Proc. of 2nd Int. Conf. on Logic Programming, 1984.

☆ J.A.Crammond,
"A Comparative Study of Unification Algorithms for Or-Parallel Execution of Logic Languages",
IEEE Trans. on Computers C-34(10), 1985.

☆ J.A.Crammond,
"An Execution Model for Committed-Choice Non-Deterministic Languages",
Proc. of 1986 Symposium on Logic Programming, 1986.