ICOT Technical Memorandom: TM-1251

TM-1251

A Parallel CLP Language GDCC and Its Parallel Constraint Solvers for Non-Linear Equations

by
A. Aiba

March, 1993

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191 ~ 5

Institute for New Generation Computer Technology

A Parallel CLP Language GDCC and Its Parallel Constraint Solver for Non-linear Equations

Akira Aiba
Institute for New Generation Computer Technology
4-28, Mita 1-chome, Minato-ku, Tokyo 108, Japan
aiba@icot.or.jp

1 Introduction

A programming paradigm called constraint logic programming (*CLP*) was proposed by A. Colmerauer [3], J. Jaffar and J-L. Lassez [5] and M. Dincbas [4] as an extension of logic programming.

At ICOT, we started research on CLP in 1986, and we developed two CLP languages. One is a sequential CLP language called CAL (*Contrainte Avec Logique*) [1], and the another is a parallel CLP language called GDCC (*Guarded Definite Clauses with Constraints*) [9]. We introduce the latter in this article.

When we solve a problem using a conventional programming language, we usually have to give a of series of detailed instructions for each problem. If we use a high level language, we are sometimes exempt from this tedious task. The most successful examples of such languages are logic programming languages, such as Prolog. Such languages can handle logical inferences almost automatically. However, when we want to solve problems that depend highly on the structure of its domain, logical inferences are far from sufficient, and we must also give detailed instructions for each problem.

When we want to solve problems in real life, we often have to handle various constraints over several domains, such as integers, rationals, and boolean values. Such constraints are sometimes in the form of equations, and sometimes in the form of inequalities. CLP was proposed to realize a high level programming paradigm for these problems. CLP languages are given in the framework of logic programming languages with some additional facilities for constraint. Their language processors contain automated solvers for constraint. Such solvers together with strong faculty of logical inferences for logic programming enable us to have the extremely high level languages needed for various problem solving.

However, such high quality greatly increase the load on a language processor of a CLP language. Thus, the more general and powerful a CLP language, the bigger the issue of the efficiency of its processor. This is one of our reason for introducing parallelism in CLP languages.

GDCC is aiming at a programming language for efficient and flexible problem solving by constructing its language processor on a parallel inference machine.

2 Parallel Constraint Logic Programming Language GDCC

GDCC is a parallel constraint logic programming language which incorporates the framework of the cc language [7]. cc is a scheme of the parallel constraint logic programming languages that is an extension of parallel logic programming language scheme. In cc, the computation is modeled on a form where multiple agents exchange information by inquiring and modifying the constraints store. This model can be regarded as a computation model with guards (conditions). The truth or falsity of a question corresponds to the truth or falsity of a guard constraint, and a modification corresponds

to the process of a body constraint. Therefore, this framework is very suitable for KL1 [10], that is a committed-choice parallel logic programming language.

2.1 GDCC Language and System

The following is an example of a program written in GDCC. This problem is known as the Pony and Men problem.

```
pam(Heads, Legs, Ponies, Men) :- true |
          alg#Heads = Ponies + Men,
          alg#Legs = 4*Ponies + 2*Men.
```

In the example, expressions starting with "`alg#`" are constraints that are handled in the algebraic solver. In a guard, the user can place ask-constraints that are passive constraints, and built-in predicates of KL1. In the above program, "`true`" is placed in the guard that represents a constraint always reduced to *true*. As with GHC, GDCC is a committed choice language. Clauses that can be unified against a query are called *candidate clauses*. Among the candidate clauses, one clause whose guards are reduced to true is selected nondeterministcally.

When the program is executed with the following query:

```
?- pam(5, 14, Ponies, Men).
```

From this, the answer, `Ponies = 2, Men = 3`, is obtained. If query

```
?- pam(Heads, Legs, 2, 3).
```

is input, then the answer, `Heads = 5, Legs = 14`, is obtained.

GDCC system consists of *a GDCC shell*, that translates users query into its internal form and provides a simple debugging facility; *a compiler*, that compiles a GDCC source program into a KL1 program; *constraint solvers*, that solve active (tell) constraints in GDCC programs; and *an interface*, that solves passive (ask) constraints and connects the shell and the constraint solvers.

Unlike sequential systems, all components in GDCC system can run in parallel. That is, a KL1 program compiled from a user's GDCC program and constraint solvers that are invoked from the KL1 program can be executed in parallel, and are synchronized, if necessary.

2.2 Features of GDCC System: Block

Since GDCC has a facility to estimate the real roots of univariate nonlinear constraints, some variables may have more than two values. Because GDCC is a committed-choice parallel language, failure in a body may cause failure of the entire program. To facilitate searching in GDCC, we need a function to localize failure. Furthermore, when maximizing or minimizing a function with respect to a set of linear inequation constraints, we must have a function to designate a set of constraints. For these purposes, we introduce the mechanism of a "*block*" into GDCC.

The syntax of a block is as follows:

`call(`*Goals*`)using` *Solver-Package* `for` *Domain*
`initial` *Input-Con* `giving` *Output-Con*

where, "*Goals*" represents a sequence of goals, "`using` *Solver-Package* `for` *Domain*" indicates that the block uses a constraint solver designated by *Solver-Package*, for the constraint domain *Domain*. "`Initial` *Input-Con*" indicates an initial constraint set for this block, and "`giving` *Output-Con*" indicates a constraint set which is the result for the block.

Within a block, there are two kinds of variables: one is local variables and the other is global variables. Local variables are specified within a block by the built-in predicate of GDCC, `alloc/2`. Variables that are not specified within a block by `alloc` are treated as global variables.

The following is a GDCC program using blocks:

```
test :- true |
          alloc(200, A),
          alg#A=-1,
          call( alg#A=1 ) initial nil giving C0,
          call( alg#A=0 ) initial nil giving C1.
```

When this program is executed by executing the goal "test", constraint set C0 has a constraint set "A=1", C1 has a constraint set "A=0", and the entire computation succeeds.

3 Parallel Constraint Solver for Non-Linear Equations

For a constraint solver to evaluate nonlinear equations in GDCC, we employ the *Buchberger Algorithm* [2] to compute the so called Gröbner Bases that have been used in the field of computer algebra. This algorithm works on equations on complex numbers, and it satisfies the following conditions required for constraint solvers: (1) Can the solver decide whether a given constraint is satisfiable? (2) Given satisfiable constraints, can the solver compute the simplest form (called the canonical form of the constraints) in a certain sense?

We parallelize the algorithm on top of the distributed memory machine, called Multi-PSI by using the KL1 language. After a series of experiments, we finally achieve a relatively good speedup for the parallel system on a distributed memory machine, and we achieve relatively good absolute speed as shown in table 1.

Table 1: Comparison between existing systems and our system

Benchmarks	System	Number of Processors				
		1	2	5	12	16
Katsura-4 (Seconds)	**GDCC**	8	9	5	6	8
	Vidal's system	17	10	4	4	–
	Giovini's system	40	–	–	–	–
Katsura-5 (Seconds)	**GDCC**	82	84	23	21	23
	Vidal's system	1103	551	146	79	–
Cyc 4-roots (Seconds)	**GDCC**	1	1	2	3	4
	Siegl's system	218	–	–	–	36
Cyc 5-roots (Seconds)	**GDCC**	23	25	14	15	18
	Giovini's system	143	–	–	–	–
T-6 (Minutes)	**GDCC**	443	438	155	80	72
	Backelin's system	90	–	–	–	–

For a full descripton of **Vidal's** , **Giovini's** , **Siegl's** , and **Backelin's** systems, please refer to the paper on GDCC [9].

4 Concluding Remarks

Experiments to improve the efficiency of GDCC are being carried out with the focuse being on the algebraic constraint solver. To date, some application programs have been written, and are being used for evaluation, using GDCC. Application programs include a design support system for handling robots [8], Voronoi diagram construction, and solving hierarchical constraints in parallel.

Constraint logic programming is the paradigm now drawing the most attention for its future potential, due to its high-levelness and declarativeness. However, many improvements are still required in areas such as, functionfalities, implementation, and heuristics representation.

Low productivity often results when an ad hoc approach to problem solving is utilized. Such low productivity requires new programs to be coreated to address each problem. This could, to some extent, be overcome with the constraint logic programming language.

Finally, we would like to remark that the KL1 programming environment will be ported on UNIX machine in FGCS Follow-on Project. In the project, we plan to port GDCC language processor on top of the KL1 programming environment on UNIX to be able to use GDCC on UNIX machine.

References

[1] A. Aiba, K. Sakai, Y. Sato, D. J. Hawley, and R. Hasegawa. Constraint Logic Programming Language CAL. In *Proceeding of the International Conference on Fifth Generation Computer Systems 1988*, November 1988.

[2] B. Buchberger. Gröbner bases: An Algorithmic Method in Polynomial Ideal Theory. In N. Bose, editor, *Multidimensional Systems Theory*, pp. 184 232. D. Reidel Publ. Comp., Dordrecht, 1985.

[3] A. Colmerauer. Opening the Prolog III Universe: A new generation of Prolog promises some powerful capabilities. *BYTE*, pp. 177–182, August 1987.

[4] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The Constraint Logic Programming Language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems 1988*, November 1988.

[5] J. Jaffar and J-L. Lassez. Constraint Logic Programming. In *4th IEEE Symposium on Logic Programming*, 1987.

[6] K. sakai and A. Aiba. CAL: A Theoretical Background of Constraint Logic Programming and its Applications. *Journal of Symbolic Computation*, (8): pp.589–603, 1989.

[7] V. Saraswat. *Concurrent Constraint Programming Languages*. PhD Thesis, CMU, Computer Science Department, January 1989.

[8] S. Sato and A. Aiba. An Application of CAL to Robotics. Technical Memorandum TM–1032, Institute for New Generation Computer Technology, February 1991.

[9] 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 *Proceedings of the International Conference on Fifth Generation Computer Systems 1992*, June 1992.

[10] K. Ueda and T. Chikayama. Design of the Kernel Language for the Parallel Inference Machine. *Computer Journal*, December 1990.