The Constraints Journal, Vol. 2, no. 1, 63–69 (1997)

Some Challenges for Constraint Programming

MANUEL HERMENEGILDO herme@fi.upm.es • http://www.clip.dia.fi.upm.es/~herme/
Computer Science Department, Technical University of Madrid (UPM), Spain

Abstract. We propose a number of challenges for future constraint programming systems, including improvements in implementation technology (using global analysis based optimization and parallelism), debugging facilities, and the extension of the application domain to distributed, global programming. We also briefly discuss how we are exploring techniques to meet these challenges in the context of the development of the CIAO constraint logic programming system.

Keywords: Constraint Programming, Implementation Technology, Global Analysis, Parallelism, Debugging, Visualization, Distributed Programming, Global Programming.

1. Introduction

Constraint programming languages [24, 17] are among the very few new language proposals that are seeing industrial success while being based on a novel programming paradigm. Intervening factors in this fact appear to be the expressiveness awarded by the high level of these languages, the combination of search and incremental constraint solving capabilities, and the relative efficiency of the resulting applications. These characteristics are added to the strengths in general-purpose symbolic processing of the underlying logic programming kernel on which they are often built. Constraints extend this kernel to numerical domains and beyond, offering a natural platform in which applications combining symbolic and numeric computing can be easily developed. This initial success notwithstanding, it seems important to identify significant challenges which lay ahead for constraint programming. Without any pretense of being exhaustive, we herein touch upon some challenges in the areas of implementation technology, network-wide programming, and language and system design.

2. Faster performance through global analysis

The performance of current constraint logic programming systems often compares favorably to other constraint solving and symbolic programming tools. However, their performance is still lower than that of traditional imperative languages, specially in (non constraint related) numerical computations. A possible solution is to develop advanced compilation technology capable of detecting the cases where limited or no constraint solving is involved and compiling those cases in the most efficient way. Some promising progress has already been made towards this goal. The efficiency and effectiveness of global analysis of logic programs is already established [40], and essentially the same performance as imperative languages has been achieved in some experimental systems (e.g., [34]). Some results on the possible speedups attainable in constraint logic programming have been reported (e.g., [32, 12]), some practical frameworks for global analysis developed (e.g., [8]), and a few systems described which perform global analysis based optimization [25, 12]. Such global analysis has also been applied to concurrent constraint programming systems, where one of the most important objectives is to reduce suspension and resumption of goals and synchronization overhead [7, 29, 30, 26, 31, 14]. Recent progress in incremental and compositional global analysis [29, 20, 2] appears to solve most remaining problems related to dealing with large programs and the interactive program development environment that is common in constraint programming systems. However, the application of extensive optimization in commercial or widely used public domain systems still remains a goal to be achieved. Also, much research remains to be done in finding accurate abstraction domains for standard constraint systems.

3. Faster performance through automatic parallelization

Program parallelization is becoming a more and more interesting optimization since multiprocessing hardware is starting to be in many cases the default installation platform. This is often the case, for example, for departmental servers where multiprocessors using using fast, inexpensive, off-the-shelf processors are replacing mainframes at a fraction of their cost. Some high-end workstations are also multiprocessors. It appears likely that this trend towards increased use of parallelism will continue as multiprocessor architectures are better understood, interconnection network performance increases with new technologies (specially if the promise of optical interconnect is finally delivered), and feature size diminishes allowing placement of several processors on the same chip.

The interest of the study of parallelism in logic and constraint logic programs goes beyond the paradigms themselves. In fact, these languages allow studying challenging issues in parallelization while remaining in the context of a relatively clean and well understood framework for which application codes are available. Such challenging issues include inter-procedural parallelization, irregular computations, recursion, complex data structures with (well behaved) pointers, and speculative parallelism. These issues have already been studied in the context of logic programs [4], where significant speedups have been obtained. In the context of constraint logic programming, exploitation of parallelism in the search (or-parallelism) is comparatively easy and has been shown to achieve significant speedups in constraint programs containing independent search [18, 11, 10]. On the other hand, and as pointed out in [24], comparatively little work has been devoted so far to exploiting parallelism within a given path of the search (and-parallelism) and in the solver itself. Although traditional concepts of independence used in imperative programming (e.g., the "Bernstein conditions") or even those of logic programming, do not apply in the context of CP [13], notions of independence appropriate for (concurrent) CP have been recently proposed [13, 3]. Our group has since been tackling the issue of automatic parallelization and has recently developed an automatic parallelizing compiler for constraint logic programs, with encouraging initial results [12]. While it may prove the case that there is often not much parallelism in constraint solving kernels, we believe that over the set of all programs the technology will eventually prove as successful as in the context of logic programming. However, proving this and developing new parallelization technology for constraint logic programs remains an area for future work.

4. Improved debugging and visualization facilities

An area in which current constraint logic programming systems have been found lacking by users developing industrial applications is debugging, in terms of ensuring correctness as well as understanding and improving performance of the program. Possible solutions can be based on both static and dynamic techniques. One technique is the use of assertions [9, 2]. Assertions can be seen as a generalization of type systems (often including directional types), in which relatively general preconditions and postconditions can be declared for procedures. Assertions can be provided by the user and checked at compile-time or run-time. Compile-time checking can in some cases be done via global analysis. Furthermore, assertions can be generated by the compiler, which the user can then inspect in order to detect high-level errors (here, the same technology used by the compiler for optimization purposes can be used). Another possible technique is the use of visualization, both of the search space and of the constraint store at different points of the execution (e.g., [28]). The development of more useful assertions, assertion checking and generation algorithms, and visualization paradigms for constraint programming remains a challenge.

5. Distributed and network-wide programming

Another challenge for constraint programming systems is related to the role of such systems in network-wide programming. This type of programming is likely to be of growing importance since the recent wider diffusion of the Internet and the popularity of the "World Wide Web" –WWW– protocols, which effectively provide a platform that is standard and ubiquitous and allows a new class of highly sophisticated distributed applications. Constraint logic programming systems already offer many characteristics which appear to place them well for making an impact in this area. Some of these characteristics, including dynamic memory management, well behaved structure and pointer manipulation, robustness, and compilation to architecture independent byte-code, are shared with other proposed network programming tools. In addition, constraint logic programming systems offer a quite unique set of other built-in features including dynamic databases, search facilities, grammars, sophisticated meta-programming, constraint solving capabilities, and well understood semantics. In addition, the theory of concurrent constraint programming offers a novel and very rich notion of inter-process communication based on constraint entailment [27, 37]. Some of these characteristics, such as the built-in database and grammars, can facilitate the rapid coding of relatively simple applications, even by naive end users. Furthermore, it seems natural that the more sophisticated network applications, such as intelligent information retrievers or distributed decision making systems, will require complex symbolic and numeric capabilities, including natural language processing, at which constraint programming (and, in particular, constraint logic programming) has already been shown particularly successful [33].

Sophisticated distributed applications can be developed with current constraint logic programming systems using the available low-level primitives to build communication abstractions such as blackboards [1, 5, 38] and even incorporating distributed variable-based communication (which can be supported, for example, using attributed variables [16]). In our experience the main feature found lacking in standard systems is native support for concurrent execution. While many systems allow starting a whole subprocess via the operating system interface and this can provide the desired functionality, it is very inefficient. A more direct support for concurrency (perhaps along the lines of that present in the AKL [23], Oz [36], and CIAO [21, 15] systems) seems highly desirable and we feel should be provided in all future constraint programming systems. One additional important use of concurrency is in implementing complex, delay based constraint solving algorithms. However, concurrency brings important new challenges in many areas. As mentioned before, an important one from the implementation point of view is developing effective analysis and optimization techniques.

Distributed concurrent constraint systems are currently being worked on [5, 22], distribution and application development libraries are being offered (e.g., [16, 6]), and network and WWW applications are being reported [39]. CP is a promising foundation for many aspects of the next generation of distributed systems, but it still remains as a challenge to develop simple, elegant and practically usable environments, and to demonstrate applications of such environments.

6. The CIAO system

We are developing a concurrent constraint logic programming system, "CIAO", which illustrates our own ideas on how to approach some of the challenges that we have set forward in the previous paragraphs [21, 15]. At the user level CIAO is a programming environment offering support for full standard Prolog, as well as several constraint domains, several control rules, a module and object system, and concurrency, synchronization, and distribution primitives. As opposed to other concurrent constraint languages such as AKL [23] or OZ [36], the language is "sequential by default", in the sense that concurrency is annotated explicitly by the user. Optional control rules include for example the determinate-first principle [35]. A rich set of assertions is available which the compiler understands and both generates and checks by means of global analysis techniques based on abstract interpretation. These assertions include a flexible type system.

At the implementation level most functionality is implemented via source to source transformation into a simple kernel language, which is then the subject of extensive analysis and optimization, using a single set of analysis and transformation tools. High-level optimizations include parallelization, granularity control, reduction of concurrency and synchronization, reordering of goals, code simplification, and specialization. Distributed and network-wide programming are implemented via libraries [16, 6]. One of the main features of the kernel language and the abstract machine underlying it is the support for concurrency and parallel execution, based on an extension of the capabilities of the &-Prolog engine [19]. CIAO is currently being used both as a workbench for testing implementation techniques (many of the program analysis and transformation systems referenced elsewhere have been prototyped on CIAO) and a powerful programming environment. Current versions of the &-Prolog and CIAO systems, and the related libraries are publicly available for experimentation (see `http://www.clip.dia.fi.upm.es/` for both software and publications).

Acknowledgments

The author wishes to thank Kim Marriott, Philippe Codognet, Ken Mcaloon, and the members of the CLIP group at UPM, F. Bueno, D. Cabeza, M. Carro, M. García de la Banda, P. López, and G. Puebla for their comments on previous drafts of this paper.

References

1. J. Almgren, S. Andersson, L. Flood, C. Frisk, H. Nilsson, and J. Sundberg. *Sicstus Prolog Library Manual.* Po Box 1263, S-16313 Spanga, Sweden, October 1991.
2. F. Bueno, D. Cabeza, M. Hermenegildo, and G. Puebla. Data–flow Analysis of Standard Prolog Programs. In *European Symposium on Programming*, number 1058 in LNCS, pages 108–124, Sweden, April 1996. Springer-Verlag.
3. F. Bueno, M. Hermenegildo, U. Montanari, and F. Rossi. From Eventual to Atomic and Locally Atomic CC Programs: A Concurrent Semantics. In *Fourth International Conference on Algebraic and Logic Programming*, number 850 in LNCS, pages 114–132. Springer-Verlag, September 1994.
4. J. Chassin and P. Codognet. Parallel Logic Programming Systems. *Computing Surveys*, 26(3):295–336, September 1994.
5. D. Cabeza and M. Hermenegildo. Distributed Concurrent Constraint Execution in the CIAO System. In *Proc. of the 1995 COMPULOG-NET Workshop on Parallelism and Implementation Technologies*, Utrecht, NL, September 1995. U. Utrecht / T.U. Madrid. Available from `http://www.clip.dia.fi.upm.es/`.
6. D. Cabeza, M. Hermenegildo, and S. Varma. The PiLLoW/CIAO Library for INTERNET/WWW Programming using Computational Logic Systems. In *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP"96, Bonn, September 1996. Text and code available from `http://www.clip.dia.fi.upm.es/miscdocs/pillow/pillow.html`.

7. M. Codish, K. Marriott M. Falaschi, and W. Winsborough. Efficient analysis of concurrent constraint logic programs. In *Twentieth International Coll. Automata, Languages and Programming*, Lund, Sweden, July 1993.
8. M. García de la Banda, M. Hermenegildo, M. Bruynooghe, V. Dumortier, G. Janssens, and W. Simoens. Global Analysis of Constraint Logic Programs. *ACM Transactions on Programming Languages and Systems*, 18(5):564–615, 1996.
9. W. Drabent, S. Nadjm-Tehrani, and J. Maluszynski. Algorithmic debugging with assertions. In H.Abramson and M.H.Rogers, editors, *Meta-programming in Logic Programming*. MIT Press, 1989.
10. L. Li et al. APPLAUSE: Applications using the ElipSys parallel CLP system. In *Proceedings of the International Conference on Logic Programming*, pages 847–848. MIT Press, 1993.
11. European Computer Research Center. *Eclipse User's Guide*, 1993.
12. M. García de la Banda, F. Bueno, and M. Hermenegildo. Towards independent And-Parallelism in CLP. In *International Symposium on Programming Language Implementation and Logic Programming, PLILP'96*, volume 1140 of *LNCS*, pages 77–91. Springer Verlag, September 1996.
13. M. García de la Banda, M. Hermenegildo, and K. Marriott. Independence in Constraint Logic Programs. In *1993 International Logic Programming Symposium*, pages 130–146. MIT Press, Cambridge, MA, October 1993.
14. M. García de la Banda, K. Marriott, and P. Stuckey. Efficient Analysis of Constraint Logic Programs with Dynamic Scheduling. In *1995 International Logic Programming Symposium*, pages 417–431, Portland, Oregon, December 1995. MIT Press, Cambridge, MA.
15. M. Hermenegildo, F. Bueno, M. García de la Banda, and G. Puebla. The CIAO Multi-Dialect Compiler and System: An Experimentation Workbench for Future (C)LP Systems. In *Proceedings of the ILPS'95 Workshop on Visions for the Future of Logic Programming*, Portland, Oregon, USA, December 1995.
16. M. Hermenegildo, D. Cabeza, and M. Carro. Using Attributed Variables in the Implementation of Concurrent and Parallel Logic Programming Systems. In *Proc. of the Twelfth International Conference on Logic Programming*, pages 631–645. MIT Press, June 1995.
17. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, 1989.
18. P. Van Hentenryck. Parallel Constraint Satisfaction in Logic Programming. In *Sixth International Conference on Logic Programming*, pages 165–180, Lisbon, Portugal, June 1989. MIT Press.
19. M. Hermenegildo and K. Greene. The &-Prolog System: Exploiting Independent And-Parallelism. *New Generation Computing*, 9(3,4):233–257, 1991.
20. M. Hermenegildo, G. Puebla, K. Marriott, and P. Stuckey. Incremental Analysis of Logic Programs. In *International Conference on Logic Programming*, pages 797–811. MIT Press, June 1995.
21. M. Hermenegildo and the CLIP group. Some Methodological Issues in the Design of CIAO - A Generic, Parallel, Concurrent Constraint System. In *Principles and Practice of Constraint Programming*, LNCS 874, pages 123–133. Springer-Verlag, May 1994.
22. S. Haridi, P. VanRoy, and G. Smolka. An Overview of the Design of Distributed Oz. In *Proc. of the 1996 JISCLP Workshop on Multi-Paradigm Logic Programming*. T.U.Berlin, September 1996.
23. S. Janson and S. Haridi. Programming Paradigms of the Andorra Kernel Language. In *1991 International Logic Programming Symposium*, pages 167–183. MIT Press, 1991.
24. J. Jaffar and M.J. Maher. Constraint Logic Programming: A Survey. *Journal of Logic Programming*, 13/20:503–581, 1994.
25. A. Kelly, A. Macdonald, K. Marriott, P. Stuckey, and R. Yap. Effectiveness of optimizing compilation for CLP(R). In *Proceedings of Joint International Conference and Symposium on Logic Programming*, pages 37–51. MIT Press, 1996.
26. Andy King and Paul Soper. Schedule Analysis of Concurrent Logic Programs. In Krzysztof Apt, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 478–492, Washington, USA, 1992. The MIT Press.

27. M. J. Maher. Logic Semantics for a Class of Committed-choice Programs. In Jean-Louis Lassez, editor, *Proceedings of the Fourth International Conference on Logic Programming*, Series in Logic Programming, pages 858–876, Melbourne, 1987. MIT Press.
28. M. Meier. Grace User Manual, 1996. Available at `http://www.ecrc.de/eclipse/html/grace/grace.html`.
29. K. Marriott M. Falaschi, M. Gabbrielli and C. Palamidessi. Compositional analysis for concurrent constraint programming. In *IEEE Symposium on Logic in Computer Science (LICS)*, Montreal, June 1993.
30. K. Marriott, M. García de la Banda, and M. Hermenegildo. Analyzing Logic Programs with Dynamic Scheduling. In *20th. Annual ACM Conf. on Principles of Programming Languages*, pages 240–254. ACM, January 1994.
31. U. Montanari, F. Rossi, F. Bueno, M. García de la Banda, and M. Hermenegildo. Towards a Concurrent Semantics based Analysis of CC and CLP. In *Principles and Practice of Constraint Programming*, LNCS 874, pages 151–161. Springer-Verlag, May 1994.
32. K. Marriott and P. Stuckey. The 3 R's of Optimizing Constraint Logic Programs: Refinement, Removal, and Reordering. In *19th. Annual ACM Conf. on Principles of Programming Languages.* ACM, 1992.
33. The practical application of constraint technology conference series. The Practical Application Company, 54 Knowle Avenue, Blackpool, Lancs FY2 9UD, U.K.
34. P. Van Roy and A.M. Despain. High-Performace Logic Programming with the Aquarius Prolog Compiler. *IEEE Computer Magazine*, pages 54–68, January 1992.
35. V. Santos-Costa, D.H.D. Warren, and R. Yang. Andorra-I: A Parallel Prolog System that Transparently Exploits both And- and Or-parallelism. In *Proceedings of the 3rd. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.* ACM, April 1990.
36. Gert Smolka. The Oz programming model. In Jan van Leeuwen, editor, *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000, pages 324–343. Springer-Verlag, Berlin, 1995.
37. V. Saraswat, M. Rinard, and P. Panangaden. Semantic Foundation of Concurrent Constraint Programming. In *Proceedings of the 18th. Annual ACM Conf. on Principles of Programming Languages.* ACM, 1991.
38. Paul Tarau and Koen De Bosschere. Virtual World Brokerage with BinProlog and Netscape. In *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP"96, Bonn, September 1996. Available from `http://clement.info.umoncton.ca/~lpnet/lpnet11.html` http://clement.info.umoncton.ca/~lpnet/lpnet11.html.
39. Paul Tarau, Andrew Davison, Koen De Bosschere, and Manuel Hermenegildo, editors. *Proceedings of the 1st Workshop on Logic Programming Tools for INTERNET Applications*, JICSLP"96, Bonn, September 1996.
40. R. Warren, M. Hermenegildo, and S. Debray. On the Practicality of Global Flow Analysis of Logic Programs. In *Fifth International Conference and Symposium on Logic Programming*, pages 684–699, Seattle, Washington, August 1988. MIT Press.

Received Date
Accepted Date
Final Manuscript Date