Future directions of automated deduction: Distributed automated deduction

Maria Paola Bonacina [*]
Department of Computer Science
University of Iowa
Iowa City, IA 52242-1419, USA
bonacina@cs.uiowa.edu

Keywords: theorem proving, contraction, parallelism, distributed search.

Distributed automated deduction is concerned with the design and realization of strategies where multiple deductive processes work concurrently towards the solution of theorem-proving problems. These strategies are usually implemented in distributed environments, such as a network of computers, or a loosely coupled, asynchronous multiprocessor, where each deductive process runs on a node of the system. While this area has emerged in automated deduction only in the past few years, a complete survey of the work done so far is beyond the scope of this document. Two surveys are available in [4] and [11], and most approaches proposed since those surveys were written may be found in [6] or [7]. In the following we give first some motivation and introduction, then we focuse on a fundamental problem for future research, the design of techniques to partition the search space among cooperating concurrent processes.

Motivation The aim of distributed automated deduction is to design distributed strategies that may improve the performance of theorem proving by employing in parallel more than one machine. Higher performance means obtaining proofs in shorter time, and possibly proving theorems that are beyond the reach of contemporary sequential methods. In addition, distributed theorem proving poses a number of new problems, both theoretical and practical, arising from having multiple processes performing logical inferences in parallel and cooperating to prove theorems. These problems are relevant to automated reasoning at large, and some are significant instances of general problems in distributed computing.

Coarse-grain parallelism Some of the most successful theorem-proving programs existing today implement either subgoal-reduction strategies (e.g., [3, 10]) or contraction-based strategies (e.g., [2, 8, 9]). The former are *backward-reasoning* strategies that work on a stack of goals, and counter the combinatorial explosion of the search space by using *depth-first search with iterative deepening*, so that only one path needs to be kept in memory at any given time. The latter

[*]Supported in part by the National Science Foundation with grant CCR-94-08667.

are *forward-reasoning* strategies that work on a database of clauses, and counter the combinatorial explosion of the search space by the ability of detecting and eliminating *redundant* clauses. Contraction-based strategies proved to be more powerful than forward-reasoning strategies without contraction on significant classes of problems, and are the strategies of choice for equational reasoning. In [4], we analyzed the parallelizability of strategies in these categories with respect to the granularity of parallelism. We defined three types of parallelism for deduction: *fine-grain* parallelism, or *parallelism at the term level* (each parallel process performs a subtask of the inference steps), *medium-grain* parallelism, or *parallelism at the clause level* (each parallel process performs one or relatively few inference steps with a common premise), and *coarse-grain* parallelism, or *parallelism at the search level* (each parallel process generates a derivation). Based on a qualitative analysis of the operations of the strategies, and on the experimental work done up to then by several authors, we discussed how coarse-grain parallelism is suitable for both subgoal-reduction and contraction-based strategies, and appears to be the most appropriate for contraction-based strategies, that are our primary interest.

Parallelism at the search level Concurrent, asynchronous deductive processes search in parallel for a solution of the given theorem-proving problem. In contrast with fine-grain parallelism, each process may be given a large portion of the data, (e.g., a fairly large set of clauses), and the data sets of the processes do not need to be shared. Each process develops its own derivation, while communicating with the other processes. As soon as one of them succeeds, the whole distributed strategy succeeds. The key concept in parallelism at the search level is *to partition the search space* among the concurrent deductive processes, so that each of them faces a smaller search task than the sequential process. The effect of partitioning the search space and of asynchronous communication is that the deductive processes generate portions of the search space which may be radically different from those generated sequentially. On one hand, this is an advantage and a reason for the interest in parallelism at the seach level. Since the portions of the search space can be different, the distributed strategy may find a much faster proof than the sequential one. On the other hand, it is a challenge. First, if the search space is not partitioned properly, the distributed prover may generate a proof similar to the sequential one from fragments of the search space seen by the sequential prover, so that it may be at a disadvantage. Second, like in all approaches to distributed deduction, one needs to preserve completeness, while keeping redundancy and communication in check.

Impact Parallelism at the search level does not consist solely in parallelizing the operations of the sequential strategy, but in a new mode of search. Therefore, it may have an impact in different ways, within the field of deduction and outside. In terms of performances, many applications of deduction may benefit from high-performance distributed theorem provers. This is especially true if good results can be achieved by employing as distributed environment local networks of workstations, since such networks are commonly available. Some of the experiences reported in the literature (e.g., [1, 5, 12]) are encouraging in this respect. In terms of understanding of search problems and design of search mechanisms, the techniques and search plans developed for distributed theorem proving may be used towards distributed solutions of other search problems.

This includes problems in distributed artificial intelligence and multi-agent systems.

References

[1] Avenhaus, J., Denzinger, J. and Fuchs, M., DISCOUNT: A system for distributed equational deduction, in [7], 397–402.

[2] Anantharaman, S. and Hsiang, J., Automated proofs of the Moufang identities in alternative rings, *J. Automated Reasoning* **6**(1) (1990) 76–109.

[3] Astrachan, O. L. and Loveland, D. W., METEORs: High performance theorem provers using model elimination, in R. S. Boyer, (ed.), *Automated Reasoning: Essays in Honor of Woody Bledsoe*, Kluwer Academic Publisher, Dordrecht, The Netherlands, 1991, 31–60.

[4] Bonacina, M. P. and Hsiang, J., Parallelization of Deduction Strategies: An Analytical Study, *J. Automated Reasoning*, **13** (1994) 1–33.

[5] Bonacina, M. P. and Hsiang, J. , The Clause-Diffusion methodology for distributed deduction, in D. A. Plaisted (ed.), *Fundamenta Informaticae* **24** (1995) 177–207, special issue on term rewriting systems.

[6] Hong, H., (ed.), *Proceedings of the First Symposium on Parallel Symbolic Computation*, Linz, Austria, September 1994, World Scientific, Lecture Notes Series in Computing **5** (1994), and *J. Symbolic Computation*, special issue dedicated to the symposium, forthcoming.

[7] Hsiang, J. (ed.), *Proceedings of the Sixth Conference on Rewriting Techniques and Applications*, Kaiserslautern, Germany, April 1995, Springer Verlag, Lecture Notes in Computer Science **914** (1995).

[8] Kapur, D. and Zhang, H., RRL: a Rewrite Rule Laboratory, in E. Lusk, R. Overbeek (eds.), *Proc. Nineth International Conference on Automated Deduction*, Argonne, Illinois, May 1988, Lecture Notes in Computer Science 310, Springer-Verlag, New York, 1988, pp. 768–770.

[9] McCune, W. W., Otter 3.0 Reference Manual and Guide, Technical Report ANL-94/6, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Illinois, January 1994.

[10] Stickel, M. E., A Prolog Technology Theorem Prover: Implementation by an Extended Prolog Compiler, *J. Automated Reasoning* **4** (1988) 353-380.

[11] Suttner, C. B. and Schumann, J., Parallel Automated Theorem Proving, in L. Kanal, V. Kumar, H. Kitano and C. B. Suttner (eds.), *Parallel Processing for Artificial Intelligence*, Elsevier, 1994.

[12] Zhang, H., Bonacina, M. P. and Hsiang, J., PSATO: a Distributed Propositional Prover and Its Application to Quasigroup Problems, in [6].