The Clause-Diffusion theorem prover Peers-mcd (System description)

Maria Paola Bonacina [⋆]

Department of Computer Science
The University of Iowa
Iowa City, IA 52242-1419, USA
bonacina@cs.uiowa.edu

Abstract. Peers-mcd is a distributed theorem prover for equational logic with associativity and commutativity built-in. It is based on the Clause-Diffusion methodology for distributed deduction and the Argonne prover EQP. New features include ancestor-graph oriented criteria to subdivide the search among the parallel processes. Peers-mcd shows super-linear speed-up in a case study in Robbins algebra.

Challenge theorems may require the best sequential provers to run for days. A purpose of building a distributed theorem prover is to solve such problems by distributing the work among multiple computers. Various approaches to parallel theorem proving differ in the granularity of parallelism. While systems with fine-grain parallelism parallelize the basic algorithms of the strategy, systems for coarse-grain parallelism parallelize the search process. The distributed prover Peers-mcd belongs to the latter class.

Peers-mcd is the successor of Peers [5], and it is new in several ways: it implements the Modified Clause-Diffusion method of [2], it incorporates the code of the prover EQP0.9, which solved the Robbins problem [7], and it uses the MPI standard for message passing. Peers-mcd features new criteria to subdivide the inferences among the processes. These criteria are called *ancestor-graph oriented* (AGO) criteria, because they are based on the notion of *ancestor-graph* in the model of the search space in [4].

Modified Clause-Diffusion

Clause-Diffusion [2, 3] is a methodology for *parallel search* in theorem proving. Concurrent, asynchronous deductive processes search in parallel the space of the problem. Each process executes a theorem-proving strategy, develops its own derivation, and builds its own database of clauses. As soon as one of them succeeds, the parallel search halts. The search space is subdivided among the processes by assigning the clauses to the processes. This allocation of clauses is logical, not physical: each generated clause belongs to only one process, but may be stored in the database of many. The inferences are partitioned based on

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

the ownership of clauses. In order to preserve completeness, the processes communicate clauses by message-passing. Specific characteristics of Modified Clause-Diffusion include the following. First, it subdivides both expansion inferences and backward-contraction inferences. The latter is done without preventing or delaying the deletion of redundant clauses. The advantages include less duplication, and uniform treatment of all clauses, regardless of whether they were generated by expansion or backward contraction. Second, communication is done in such a way that both purposes (preserving completeness and assigning the clauses to the processes) are achieved by one form of communication (broadcasting clauses as *inference messages*). As a consequence, fewer messages are generated, there is only one type of message, and all processes can use the clauses sooner. Third, the *naming scheme* (the mechanism for the unambiguous identification of clauses across the network) guarantees that the successful process can reconstruct the global proof based on local information.

The AGO criteria

A key component of a Clause-Diffusion strategy is the *allocation criterion* to assign clauses to processes. The simplest criterion is *rotate*, which picks process $p+1 \ mod\ N$, if there are N processes and p was chosen last. The *ancestor-graph oriented* criteria allocate a clause based on information in its ancestor-graph. Assume that the search space is represented as a search graph with vertices labelled by clauses and arcs representing the inferences. Given a generated clause φ, its *ancestor-graph* is obtained by proceeding backward from φ, e.g., if φ is generated from ψ_1 and ψ_2, its ancestor-graph has φ as root and the ancestor-graphs of ψ_1 and ψ_2 as subgraphs. The data structures in EQP already contain information on the ancestors, for the purpose of proof reconstruction (the generated proof is the ancestor-graph of the empty clause). Thus the AGO criteria use information that is kept by the prover anyway, with no significant additional work. The motivation for working with ancestor-graphs is the following. If the parallel processes search the same part of the space, parallel search may not help. Thus, a goal of an allocation criterion is *to limit the overlap of the parallel searches.* The ancestor-graphs of generated clauses represent the available information on the search space during the derivation. Therefore, the AGO criteria use this information to reduce the overlap.

The AGO criteria *parents* and *majority* work as follows. The *parents* criterion is based on the intuition that clauses which have the same parents should be assigned to the same process. The idea is that clauses which have the same parents are spatially close in the search graph. If we assign such clauses to different processes, we may increase their overlap. Given a clause φ, the *parents* criterion takes the identifiers of the parents of φ and feeds them into a function f, which returns the number of the process φ should be assigned to. Since f is a function (unique image), all clauses with same parents receive the same process. The criterion is parametric with respect to f.

The *majority* criterion is also based on an intuitive notion of proximity in the search space. Rather than considering proximity between clauses, it considers proximity between clauses and processes. A clause φ should be assigned to a process which is active near φ. Another choice could augment the overlap of the processes. Since what is known of the search space around φ is its ancestor-graph, the criterion looks for a process whose search overlaps the most with the ancestor-graph of φ. Therefore, φ is assigned to a process which owns a majority of its ancestors.

Experiments

The experiments were executed on a local area network of workstations (HP 715 with 64M), where each process of Peers-mcd runs on a separate workstation. The times in the following tables are *average wall-clock times*, expressed in seconds, for different formulations of a lemma in Robbins algebra.

EQP implements several strategies for AC-completion. The following tables refer to the strategies *start-n-pair* and *basic-n-pair* of [6]. "Start" means that the strategy uses (AC)-paramodulation, (AC)-simplification, subsumption and deletion by weight. "Basic" is the same as "start," with basic paramodulation [1, 8] in place of paramodulation. "Pair" means that the search plan works by the *pair algorithm*: it selects a pair of equations, performs all inferences between them, and repeats. The "n" in *start-n-pair* and *basic-n-pair* means that the pairs to be selected are sorted by increasing length (*best-first search*).

A Robbins algebra is presented by the axiom $n(n(x+y)+n(x+n(y)))=x$, where $+$ is AC. A well-known lemma [9] is to prove that the condition $\exists x \exists y\ x+y=x$ implies the Huntington axiom (H) $n(n(x)+y)+n(n(y)+n(x))=x$. (An algebra which satisfies H is Boolean.) Either Otter or EQP can prove that H follows from $\exists x\ x+x=x$ in less than 20 sec. Thus, the problem reduces to show that $\exists x \exists y\ x+y=x$ implies $\exists x\ x+x=x$. In [6], EQP0.9 finds a proof after 4400 sec with *start-n-pair* (best "start" strategy), and after 1902 with *basic-n-pair* (best result), using `max-weight` $=30$. Peers-mcd can do the problem in 522 sec with six nodes, strategy *start-n-pair*, the AGO criterion *parents*, and the same `max-weight`. The speed-up with respect to EQP0.9 is 7 and the efficiency is 1.2:

Strategy	*Criterion*	*EQP0.9*	*1-Peers*	*2-Peers*	*4-Peers*	*6-Peers*
start-n-pair	rotate	3705	3953	1349	1340	1631
start-n-pair	parents	3705	3953	933	915	**522**
start-n-pair	majority	3705	3953	997	1043	1187

With the *basic-n-pair* strategy, 6-Peers marks average wall-clock time 551 with the AGO criterion *parents.* The average wall-clock time of EQP0.9 is 1661, so that the speed-up is 3, with efficiency 0.5. A more general target theorem is $\exists y \forall x\ x+y=x$. With this formulation, it is sufficient to employ only two nodes to succeed in 485 sec, by using the AGO criterion *majority.* The speed-up with respect to EQP0.9 is 7.5, with efficiency 3.7:

Strategy	*Criterion*	*EQP0.9*	*1-Peers*	*2-Peers*
start-n-pair	rotate	3649	3809	2220
start-n-pair	parents	3649	3809	1591
start-n-pair	majority	3649	3809	**485**

Last, if H itself is given as target theorem, 4-Peers solves the problem in 709 sec, which represents a speed-up of 6.8 with efficiency 1.7:

Strategy	*Criterion*	*EQP0.9*	*1-Peers*	*2-Peers*	*4-Peers*
start-n-pair	rotate	4857	4904	3557	1177
start-n-pair	parents	4857	4904	1437	2580
start-n-pair	majority	4857	4904	872	**709**

A main strength of Peers-mcd is the capability of exhibiting super-linear speed-up. This is possible, because Clause-Diffusion does not parallelize the steps of the sequential strategy. It generates a new parallel search. Because of the subdivision of work (e.g., a process does not execute certain steps because the premises belong to others), the processes may generate and select clauses in different order than the sequential one. Thus, the search space is traversed in a different way. This approach may be weak in terms of scalability, because adding more processes may affect adversely the partition of the search space.

For algorithms, a super-linear speed-up by parallelization means that there is a better sequential algorithm. Analogously, super-linear speed-up by parallel search may suggest that a better sequential search plan for that problem exists. This is not unreasonable, since most search plans used in theorem proving are essentially exhaustive, and not target-oriented. Future developments include more work with the AGO criteria, and tools for the comparison of distributed and sequential proofs.

Acknowledgements I would like to thank Bill McCune for the source code of EQP0.9, and for answering my questions on EQP.

References

1. L. Bachmair, H. Ganzinger, C. Lynch, and W. Snyder. Basic paramodulation and superposition. In D. Kapur, editor, *Proc. CADE-11*, volume 607 of *LNAI*, pages 462–476. Springer, 1992.
2. M. P. Bonacina. On the reconstruction of proofs in distributed theorem proving: a modified Clause-Diffusion method. *J. of Symbolic Computation*, 21:507–522, 1996.
3. M. P. Bonacina and J. Hsiang. The Clause-Diffusion methodology for distributed deduction. *Fundamenta Informaticae*, 24:177–207, 1995.
4. M. P. Bonacina and J. Hsiang. On the representation of dynamic search spaces in theorem proving. In C.-S. Yang, editor, *Proc. Int. Computer Symp.*, Dec. 1996.
5. M. P. Bonacina and W. McCune. Distributed theorem proving by Peers. In A. Bundy, editor, *Proc. CADE-12*, volume 814 of *LNAI*, pages 841–845. Springer, 1994.
6. W. McCune. 33 Basic test problems: a practical evaluation of some paramodulation strategies. MCS Division, Argonne National Laboratory, Pre-print P618, 1996.
7. W. McCune. Solution of the Robbins problem. Submitted manuscript, 1996.
8. R. Niewenhuis and A. Rubio. Basic superposition is complete. In B. Krieg-Brückner, editor, *Proc. ESOP*, volume 582 of *LNCS*, pages 371–389. Springer, 1992.
9. S. Winker. Robbins algebra: Conditions that make a near-Boolean algebra Boolean. *J. of Automated Reasoning*, 6(4):465–489, 1990.