ICOT Technical Report: TR-0761

TR-0761

MGTP: A Parallel Theorem Prover Based on Lazy Model Generation

by
R. Hasegawa, M. Koshimura & H. Fujita

April, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
(03)3456-3191～5
Telex ICOT J32964
Minato-ku Tokyo 108 Japan

Institute for New Generation Computer Technology

MGTP: A Parallel Theorem Prover Based on Lazy Model Generation

Ryuzo HASEGAWA
Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan

Miyuki KOSHIMURA
Toshiba Information Systems (Japan)

Hiroshi FUJITA
Mitsubishi Electric Corporation

hasegawa@icot.or.jp, koshi@icot.or.jp, fujita@sys.crl.melco.co.jp

Abstract. We have implemented a model-generation based parallel theorem prover in KL1 on a parallel inference machine, PIM. We have developed several techniques to improve the efficiency of forward reasoning theorem provers based on lazy model generation. The tasks of the model-generation based prover are the generation and testing of atoms to be the elements of a model for the given theorem. The problem with this method is the explosion in the number of generated atoms and in the computational cost in time and space, incurred by the generation processes. Lazy model generation is a new method that avoids the generation of unnecessary atoms that are irrelevant to obtaining proofs, and to provide flexible control for the efficient use of resources in a parallel environment. With this method we have achieved a more than one-hundred-fold speedup on a PIM consisting of 128 PEs.

1 Introduction

The aim of this research is to make high performance theorem provers for first-order logic by using the programming techniques of the parallel logic programming language, KL1. As a first step in developing KL1-technology theorem provers, we adopted the model generation method of SATCHMO as the basis. Our reasons were as follows: 1) SATCHMO has a good property that full unification is not necessary and that matching suffices for range-restricted problems. This makes it very convenient for us to implement provers in KL1 since KL1, as a committed choice language, provides very fast one-way unification. 2) It is easier to incorporate mechanisms for lemmatization, subsumption tests, and the other deletion strategies that are indispensable in solving difficult problems such as condensed detachment problems [Overbeek 90].

Two types of model generation provers were built: one for ground models (MGTP/G) and the other for nonground models (MGTP/N). Utilizing merit 1). above, MGTP/G becomes very simple and efficient [Fujita 91] where a new algorithm, called RAMS, is implemented to avoid redundancy in conjunctive matching. MGTP/G can prove non-Horn problems very efficiently on a distributed memory multi-processor, the Multi-PSI, by exploiting OR parallelism.

MGTP/N, on the other hand, aims at proving hard Horn problems by exploiting AND parallelism. For MGTP/N, we developed a new parallel algorithm that runs with optimal load balancing on a distributed memory architecture, and with the minimal amount of computation and memory to obtain proofs. This article provides a short introduction to MGTP/N and its main features.

2 Model Generation Method

A clause is represented in an implicational form:

$$A_1, A_2, \ldots, A_n \rightarrow C_1; C_2; \ldots; C_m$$

where $A_i (1 \leq i \leq n)$ and $C_j (1 \leq j \leq m)$ are atoms; the antecedent is a conjunction of $A_1, A_2, \ldots, A_n$; and the consequent is a disjunction of $C_1, C_2, \ldots, C_m$. A clause is said to be *tester* if its consequent is *false* ($m = 0$), otherwise it is called *generator*.

The model generation method incorporates the following two rules:

- Model extension rule: If there is a generator clause, $A \rightarrow C$, and a substitution σ such that $A\sigma$ is satisfied in a model candidate M and $C\sigma$ is not satisfied in M, then extend M by adding $C\sigma$ into M.
- Model rejection rule: If a tester clause has an antecedent $A\sigma$ that is satisfied in a model candidate M, then reject M.

We call the process of obtaining $A\sigma$, a *conjunctive matching (CM)* of the antecedent literals against the elements in a model candidate.

To achieve high performance in model-generation based provers, it is important to avoid redundant computation in conjunctive matching[1]. For this, we developed RAMS [Fujita 91], MERC [Hasegawa 91], and Δ-M [Hasegawa 92] algorithms.

3 Lazy Model Generation

A more important issue with regard to the efficiency of model-generation based provers is how to reduce the total computation amount and memory space required for proof processes. This problem becomes more critical when dealing with harder problems that require deeper inferences (longer proofs). To solve this problem, it is important to recognize that proving processes are viewed as *generation-and-test* processes and that generation should be performed only when required by the testing. We achieved high performance by introducing a new method called *lazy model generation*, based on the idea of "generate-only-at-test", designed for a demand-driven style of computation.

Figure 1 shows the lazy algorithm. In this algorithm, it is assumed that two processes, an algorithm for generator clauses and the other for tester clauses, run in parallel and communicate with each other.

Tester process 1) requests a set of atoms, Δ, to the generator process, 2) performs a subsumption test on Δ against $M \cup D$ where D represents a set of atoms to be added to M, and 3) performs a rejection test on $\Delta(CJM_T)$. For the generator process, 1) if a buffer, Buf, that stores a set of atoms that are the results of an application of the model extension rule, is empty, select an atom, e out of D and set a code for model

[1] The operation may be performed for identical combinations of atoms more than once.

process tester:
    repeat forever
        $request(generator, \Delta)$:
        $\Delta' := subsumption(\Delta, M \cup D)$:
        if $CJM_T(\Delta', M \cup D) \supset \perp$ **then return**(success);
        $D := D \cup \Delta'$

process generator:
    repeat forever
        while $Buf = \phi$ **do begin**
            $D := D - \{e\}$; $Buf :=$ **delay**$CJM_G(e, M)$; $M := M \cup \{e\}$ **end**;
        $wait(tester)$:
        $\Delta :=$**force**Buf;
    until $D = \phi$ **and** $Buf = \phi$

Figure 1: Lazy algorithm

Table 1: Comparison of complexities (for unit tester clause)

Algorithm	T	S	G	M
Basic	ρm^2	$\mu\rho^2 m^{1.6}$	$\rho^2 m^3$	$\rho^3 m^4$
Full test/Lazy	ρm^2	$\mu m^{2\alpha}$	m^2	ρm^2
Lazy & Lookahead	m^2	$(\mu/\rho)m^{\alpha}$	m/ρ	m

† m is the number of elements in a model candidate when *false* is detected in the basic algorithm.
† ρ is the survival rate of a generated atom, μ is the rate of successful conjunctive matchings ($\rho \leq \mu$), $\alpha (1 \leq \alpha \leq 2)$ is the efficiency factor of a subsumption test.

extension (**delay** CJM_G) for e and M onto Buf. 2) waits for a request of Δ from the tester process, and 3) forces the buffer, Buf, to generate Δ.

The lazy mechanism can be used to control the difference in speed between the generator and tester processes, thereby avoiding the unnecessary consumption of time and space caused by over generation.

From a simple analysis, it is estimated that the time complexity of the model extension and subsumption test decreases from $O(m^3)$ in the algorithms[2] without lazy control to $O(m)$ in the lazy one. Table 1 compares the complexities of the algorithms, where T(S/G) represents the number of rejection tests(subsumption tests/model extensions), and M represents the number of atoms stored. For details, refer to [Hasegawa 92].

[2]The basic algorithm taken by OTTER[McCune 90] generates a bunch of new atoms before completing rejection tests for previously generated atoms. The full-test algorithm completes the tests before the next generation cycle, but still generates a bunch of atoms each time. Lookahead is an optimization method for testing wider spaces than in Full-test/Lazy.

4 AND Parallel Implementation

We have several choices when parallelizing model-generation based theorem provers: 1) proof changing or unchanging according to the number of PEs, 2) model sharing (copying in a distributed memory architecture) or model distribution, and 3) master-slave or master-less.

A proof changing prover may achieve super linear speedup while a proof unchanging prover can achieve at most linear speedup.

The merit of model sharing is that time consuming subsumption testing and conjunctive matching can be performed at each PE independently with minimal inter-PE communication. On the other hand, the merit of model distribution is that we can obtain "memory scalability". The communication cost, however, increases as the number of PEs increases, since generated atoms need to be flown to all PEs for subsumption testing.

The master-slave configuration makes it easy to build a parallel system by simply connecting a sequential version of MGTP/N on a slave PE to the master PE. However, its devices must be designed to minimize the load on the master process. On the other hand, a master-less configuration such as ring connection allows us to achieve pipeline effects with better load balancing, whereas it becomes harder to implement suitable control to manage collaborative work among PEs.

Our policy in developing parallel theorem provers is that we should distinguish between the speedup effect caused by parallelization and the search-pruning effect caused by strategies. In the proof changing parallelization, changing the number of PEs is merely betting, and may cause the strategy to be changed badly even though it results in the finding of a shorter proof.

Given the above, we implemented a proof unchanging version of MGTP/N in a master-slave configuration based on the lazy model generation. In this system, generator and subsumption processes run in a demand-driven mode, while tester processes run in a data-driven mode. The main features of this system are as follows: 1) Proof unchanging allows us to obtain greater speedup as the number of PEs increases. 2) By utilizing the synchronization mechanism supported by KL1, sequentiality in subsumption testing is minimized; 3) Since slave processes spontaneously obtain tasks from the master and the size of each task is well equalized, good load balancing is achieved; 4) By utilizing the stream data type of KL1, demand driven control is easily and efficiently implemented.

By using demand-driven control, we can not only suppress unnecessary model extensions and subsumption tests but also maintain a high running rate, that is the key to achieving linear speedup.

Table 2 shows performance obtained by running the MGTP/N prover for Theorems 5 and 7 [Overbeek 90] on Multi-PSI with 64 PEs. We did not use heuristics such as sorting, but merely limited term size and eliminated tautologies. Full unification is written in KL1, which is thirty to hundred times slower than that written in C on SUN/3 and SPARC. Note that the average running rate per PE for 64 PEs is even a little bit higher than that for 16 PEs. With this and other results, we were able to obtain almost linear speedup.

Recently we obtained a proof of Theorem 5 on PIM/m with 127 PEs in 2870.62 sec and 43939240329 reductions (thus 120 KRPS/PE). Taking into account that the CPU of PIM/m is about two times faster than that of Multi PSI, we found that almost

Table 2: Performance of MGTP/N (Theorem 5 and Theorem 7)

Problem		16 PEs	64 PEs
Theorem 5	time(sec)	11725.98	11056.12
	Reductions	380709405.58	407596894.19
	KRPS/pe	57.03	57.60
	speedup	1.00	3.77
Theorem 7	time(sec)	48629.93	13514.47
	Reductions	312812114.17	374075314.27
	KRPS/pe	40.20	43.25
	speedup	1.00	3.60

linear speedup can be achieved at least up to 128 PEs.

5 Conclusion

It is important to avoid explosive increases in the amount of time and space consumed when proving hard theorems that require deep inferences. For this we proposed the lazy model generation method and techniques to improve its performance. Experimental results show that a significant saving in the computation and memory amounts can be realized by using the lazy algorithm. The lazy model generation method can be easily extended from Horn clauses to non-Horn clauses.

We developed AND parallelization methods to solve Horn problems. The recent results of our experiments show that we could achieve linear speedup up to 128 PEs. The key technique is laziness in model generation that avoids unnecessary computation and memory space while maintaining a high running rate.

References

[Fujita 91] H. Fujita and R. Hasegawa. A Model-Generation Theorem Prover in KL1 Using Ramified Stack Algorithm. In *Proc. of the Eighth International Conference on Logic Programming*. The MIT Press, 1991.

[Hasegawa 91] R. Hasegawa. A Parallel Model Generation Theorem Prover: MGTP and Further Research Plan. In Proc. of the Joint American-Japanese Workshop on Theorem Proving, Argonne, Illinois, 1991.

[Hasegawa 92] R. Hasegawa, M. Koshimura and H. Fujita. Lazy Model Generation for Improving the Efficiency of Forward Reasoning Theorem Provers. ICOT TR-751, 1992.

[Manthey 88] R. Manthey and F. Bry. SATCHMO: a theorem prover implemented in Prolog. In *Proc. of CADE 88, Argonne, Illinois*, 1988.

[McCune 90] W. W. McCune. OTTER 2.0 Users Guide. Argonne National Laboratory, 1990.

[Overbeek 90] R. Overbeek. Challenge Problems. (private communication) 1990.