Design and Implementation of Massively Parallel TRAM based on Parallel TRAM

Hiromichi Hirata

School of Information Science,
Japan Advanced Institute of Science and Technology

February 13, 1998

Keywords: parallel term rewriting, abstract machines, TRAM, Parallel TRAM, message passing.

Introduction

We can use algebraic specification languages such as OBJ3 and CafeOBJ to write formal specifications of software and/or hardware systems. Algebraic specification languages are usually executable, we can verify some correctness of the specifications and can reason about some properties of the systems through the specifications (semi) automatically by using computers. Since the specifications can be executed as programs in the specification languages, we may use the specification languages as rapid prototyping tools. Many algebraic specification languages are based on (order-sorted conditional) term rewriting systems (TRSs) as a general computational model. To implement the rewrite engines efficiently on conventional computers, abstract machines for term rewriting systems are designed. TRAM (Term Rewriting Abstract Machine) is one of these abstract machines that adopts the E-strategy as its reduction strategy, which makes it possible to control the reduction reasonably. Parallel TRAM is a parallel version of TRAM, which adds ability of parallel rewriting to TRAM. It has the parallel E-strategy that is an extension of the E-strategy, and can control the parallelism of the reduction suitably.

On the other hand, nowadays, since hardware technologies are enhanced rapidly, and parallel processing has come into wide use, a workstation cluster has becomes popular to realize the parallel distributed environments economically. Furthermore, recent massively parallel computers have been used to various areas, include TRSs.

Considering the above things, it is very important to develop a method of implementing rewrite engines efficiently on massively parallel computers and/or workstation clusters. Our approach, we have massively parallelized TRAM based on Parallel TRAM so that the rewrite engines can be implemented efficiently on the massively parallel computers. In addition, we have implemented the massively parallelized TRAM (Massively Parallel TRAM) on the Cray T3E which is a massively parallel computer, and have evaluated the performance of the implementation by executing several benchmark programs on Massively Parallel TRAM.

Massively Parallel TRAM

The characteristic points of the Massively Parallel TRAM are as follows:

- The Massively Parallel TRAM is designed to be executed on massively parallel computers and/or workstation clusters.
- The Massively Parallel TRAM adopts the message passing model as its parallel computational model.
- The simple master/worker model is used as the basic parallel algorithm for Massively Parallel TRAM.
- The master maintains three pieces of information about workers, and each worker is has the three states: *idle*, *busy* and *waiting*.
- In the Massively Parallel TRAM, the reference table called *INFO* is put into the Strategy List.

Design

Since Parallel TRAM was originally designed to be executed on multiprocessors, its parallel computational model was the shared-memory model. However, it is difficult to implement Massively Parallel TRAM by using shared-memory model on massively parallel computers and/or workstation clusters. On the other hand, the message passing model can be applied to many kinds of parallel architecture. Therefore, we have decided to make use of the message passing model as the parallel computational model of the improved TRAM.

To manage the load of works on the processors, the simple master/worker model is used as the basic parallel algorithm. The model is generally used to manage the process by using message passing. In the model, a parallel work is scheduled to work efficiently by a host processor, this host processor is called *master*, and the processor in the parallel work is called *worker*.

Worker states

The master has three pieces of information about workers. Each worker has the three states: *idle*, *busy* and *waiting*. Just after booting Massively Parallel TRAM, all workers are the idle states. The master holds all the idle workers in *Idle Stack*. The workers which is rewriting terms are the busy states. A busy worker may ask idle workers to rewrite some subterms, and wait to receive the results of the rewriting from the workers. The workers waiting the results from other workers are the waiting states. The master holds all the waiting workers in *Wait List*. Not only idle workers but also waiting workers may undertake tasks from busy workers.

INFO:Reference Table

Since Parallel TRAM was originally designed to be executed on multiprocessors, interprocess memory access had no problem. On the other hand, Massively Parallel TRAM is designed to be executed on massively parallel computers and/or workstation clusters, thus its parallel computational model is distributed-memory model. Accordingly, an interprocess memory references have some problem; some process must have synchronization for garbage collection, etc. Because this problem may be serious overhead for the performance of term rewriting, we must remove interprocess memory references. We make use of reference table called *INFO* that extends the Strategy List.

Conclusion

We have improved Parallel TRAM for massively parallel computers and implemented it on Cray T3E has 128 processing elements by using the message passing library MPI(the Message-Passing Interface). We obtained following results:

- We parallelized TRAM very massively by using a concept of Parallel TRAM on distributed memory computers, without losing the advantages of TRAM.
- Since we improved the message passing model by using *INFO*, we can collect garbage asynchronous.
- The maximum performance of Massively Parallel TRAM was 67 times as faster as the performance of TRAM when it was executed with 127 workers.