A Study on Parallel Garbage Collection for Parallel TRAM

Tsuguharu Saitou

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

February 13, 1998

Keywords: parallel term rewriting, abstract machines, TRAM, multiprocessors, concurrent garbage collection.

1 Introduction

Algebraic specification languages such as CafeOBJ have been widely attracting attention since they have clear semantics and the ability to write down lucid, non-ambiguous, and non-inconsistent specifications of software systems. Many algebraic specification languages use (order-sorted conditional) term rewriting systems as a general computational model. This makes it possible to verify and prove various properties of algebraic specifications, and to execute the specifications as programs by using rewrite engines. For implementing the rewrite engines efficiently on conventional computers, abstract machines for term rewriting systems are designed. TRAM is one of these abstract machines.

On the other hand, with the rapid and steady advance of hardware technology, small-sized and low-priced multiprocessor computers have been developed. These machines are the next generation of uniprocessor workstations and personal computers and will undoubtedly become standard for the future computers.

Judging from these facts, it would be very important to develop a method of implementing rewrite engines efficiently on multiprocessors. The first version of Parallel TRAM used a globally synchronizing garbage collector that collected garbage after the collector forced all running processes to be suspended. The collector was one of the main source of the overhead in the first version. Therefore, we have designed a parallel garbage collector that could collect garbage independent of other running processes in order to decrease the overhead, and also have incorporated it into Parallel TRAM.

2 TRAM and Parallel TRAM

TRAM is an abstract machine for order-sorted conditional term rewriting systems. This can be used as an intermediate level or a target machine for complication in the area of algebraic specification languages. TRAM can be easily parallelized thanks to its architecture. The parallel variant of TRAM is called Parallel TRAM. Both of them have similar structure. However, the interpreters for abstract instructions and the mutable memory spaces whose contents may be changed during rewritings are replicated and given to each processor.

TRAM consists of seven regions and three processing units. The seven regions are DNET, CR, CODE, SL, STACK, VAR and CANDS. The three processing units are the rule compiler,the term compiler and TRAM interpreter.

DNET is the region for a discrimination net encoded from LHSs of a rewrite program. CR is the region for compiled RHSs of a rewrite program. Matching Programs compiled from subject terms are allocated on the region CODE. SL is the region for a strategy list representing a reduction strategy. STACK is the working place for pattern matching. VAR contains variable bindings for pattern matching. CANDS keeps the rule candidates.

In Parallel TRAM, the rules and the terms are compiled by the main processor sequentially.

Parallel TRAM has the four additional abstract instructions for rewriting terms in parallel. The labels at which the instructions are stored are put in the strategy list appropriately.

FORK : This instruction creates a new process for reducing a term independently and allocates it on an idle processor and some care for exterior reference table. If there are no such idle processors at the moment, this instruction is ignored and the processor calling this instruction executes the new process by itself.

WAIT : This instruction is used to synchronize a processor with its child processes. The processors calling this instruction suspend their executions until all of their child processes (that are allocated on the other processors by FORK) are finished. During they suspend, their states are changed into idle and they may execute some reductions from the other processors.

EXIT : This instruction tells the parent processor that the reduction allocated by FORK is finished. This instruction is put in the strategy list of a child process. If the parent process waits for only one child process to finish rewriting a term, the instruction awakes the parent process, and otherwise the instruction decreases a value indicating how many child processes the parent process waits for.

SLEEP : This instruction is called when a processor finishes all of its allocated reductions and changes its state into idle. This instruction makes in the initialize period.

3 Garbage Collection

Most of recent programming languages provide dynamic memory allocation. Parallel TRAM is one of them. And dynamically allocated storage may become unreachable. Cells that are not live, but are not free either, are called garbage. And, collecting this garbage is called Garbage Collection. Dynamic storage allocation raises the issue of dynamic storage management. The way of dynamic storage management can be divided into two categories: explicit storage management; and automatic storage management. In the former case, since the programmer must decide when to deallocate objects, deallocation of object is complex and error-prone. While in the latter case, objects that are no longer needed are automatically deallocated by language run-time environment. This type of deallocation is called garbage collection. It releases the programmer from complex storage management, and helps to make programs shorter and easier to maintain, because they don't have to include part of storage management.

4 Concurrent Garbage Collection in Parallel TRAM

In TRAM, CODE is the garbage-collected region. TRAM uses a copying garbage collector because

- TRAM allocates a lot of objects on CODE for a short period of time and most of them become garbage quickly, and
- the size of each object is not constant.

So we have chosen the copying garbage collector as the basic algorithm of the parallel garbage collector for Parallel TRAM.

Dijkstra's on the fly garbage collection is typical of concurrent garbage collection. But this algorithm is based on mark and sweep method. So this method is not suited to Parallel TRAM. Baker's incremental garbage collector seems to be suited to Parallel TRAM because the collector is based on the copying garbage collection. The collector is suited to real time systems because each pause time for collecting garbage is very short. However, the collector does not improve the overall performance of systems.

Then, we have adopted a similar algorithm of parallel garbage collectors used for distributed systems. In this algorithm, each processor usually does a usual work as a mutator. If it is time to collect garbage on its own region, the process collects garbage as a collector mostly independent of the other processors and basically does not force the other processors to be suspended. Only if a processor tries to access an object on the region of another processor collecting garbage, the processor has to be suspended.

To manage such inter-processor references, we use the exterior reference table in which inter-processor references are recorded. Before a processor starts to collect garbage on its own region, it first locks all the entries in the exterior reference table that hold references to the region from the other processors. A processor can access to the other region only via the exterior reference table. Just after a processor locks all the entries in the exterior

reference table, it cannot start to collect garbage because the other processors might be accessing to the region. So it waits for the other running processors to each rewrite terms in one step.

5 Conclusion

We have designed a parallel garbage collector for Parallel TRAM and have incorporated it into Parallel TRAM on the multiprocessor workstation Ultra Enterprise 4000 system carrying six Ultra SPARC processors in C programming language. We have also evaluated the parallel garbage collector by executing some benchmarks on the Parallel TRAM system. The parallel garbage collector was found to make Parallel TRAM 1.2 times faster than the first version adopting a globally synchronizing garbage collector.