ReCo: An implementation of conditional Concurrent Rewriting on distributed memory machines

Ilies Alouini
INRIA-Lorraine & CRIN
`http://www.loria.fr/equipe/protheo.html`
BP 101, 54602 Villers-lès-Nancy FRANCE
E-mail: {`Ilies.Alouini@loria.fr`}

Introduction

Conditional concurrent term rewriting is a general model of concurrent computation. ReCo is the implementation of conditional concurrent using a fine-grained parallelism on terms. The term is represented as an acyclic graph distributed over a set of processors.

From an operational point of view, term representation as graph is a well-known technique used in symbolic and functional computations. Rewriting terms represented as acyclic graphs can lead to efficient implementations by sharing subgraphs of a term during the reduction process. We describe the system ReCo which is a direct implementation of the conditional concurrent model on distributed memory machines proposed in [4, 1, 2, 3].

The essential feature of our approach is to have executable specification without having to explicit the potential parallelism present in the specification.

On the one hand, ReCo is a scalable parallel application that runs on large variety of existing parallel machines. On the other, its implementation is totally asynchronous and avoids the usual synchronisation problems met in distributed computation.

The Implementation Model

Our system ReCo (implementation of conditional concurrent rewriting) uniformly implements the model of concurrent term graph rewriting on distributed memory machine. The implementation (ReCo) is written in the single program multiple data (SPMD) style. All processors apply the same program to different data. Initially, the term to be reduced is distributed on a given number of processors. All processes apply rewrite rules concurrently until a normal form of the term is obtained (if it exists).

Figure 1 and 2 shows the organisation of ReCo . We give a brief description of each ReCo component below.

Term distribution. The initialisation process in ReCo operates as a bottom up approach. The initial term to be reduced is distributed from the leaves to the root. The important feature here is that the rewriting process begins in some subterms without waiting for the end of the distribution phase.

Matching precompilation. It consists in computing tables giving for each state the set of rules applyable at some node knowing the matching state at that node. These matching state are updated using a bottom-up message mechanism.

Asynchronous replacement. ReCo offers two mechanisms for right-hand side rewrite rule creation called *local rewriting* and *distant rewriting.* Local rewriting corresponds to the creation of right-hand side within a processor. Distant rewriting corresponds to the creation of right-hand side whithin a distant processor.

Concurrent garbage collection algorithm[1]. The implemented algorithm is derived from the reference counting algorithm adapted to our model of computation. It reclaims active processes i.e nodes are viewed as rewriting processes. The garbage collector is composed of a set of local garbage collectors integrated into each rewriting process. Each one running on a processor detects and reclaims garbage nodes by using only information local to the node.

Termination. The concurrent reduction process terminates when a term normal form is obtained. A normalisation flag which is contained in each node structure, is used to detect the termination of the concurrent reduction process. This normalisation information is easily updated by a bottom-up message mechanism.

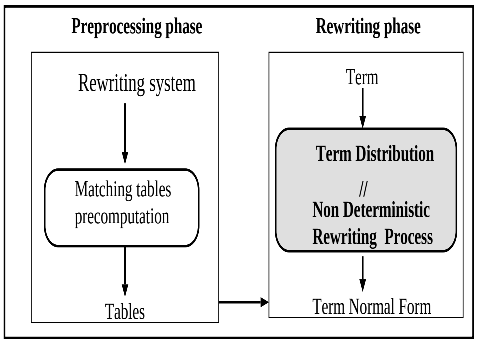

Figure 1: Functional scheme

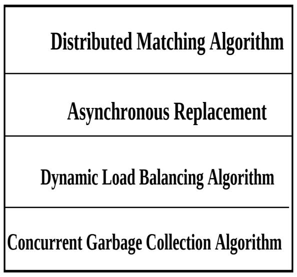

Figure 2: ReCo components

The ReCo system

ReCo is written in C (8000 lines) and the current version uses the message passing interface library (MPI) or the library PVM (Parallel Virtual Machine). The system runs on a cluster of workstations (SUN, HP,...) and on several parallel machines (Power Challenge Array, IBM SP/1). The normalisation process is non-deterministic. This leads to different possible executions for computing a normal form of a term concurrently. Processes running on each processor never wait for messages. All used MPI functions i.e the used send and receive are non-blocking calls. Load balancing is done in a distributed way. Its goal is to balance the number of rewriting done in each processor.

The main loop. The main function in ReCo consists in an infinite loop composed sucessively by two calls on each processor. The first call operates a rewriting step on a given node of the graph. The second calls is a non-blocking receive consuming the eventual received messages.

Experiments with ReCo . Experiments are based on different normalisations process executed on a cluster of Solaris Sun WorkStations, and on a Power Challenge Array (PCA) with 8 R8000 processors. Speed-ups up to 3.9 for computing term normal form can be obtained on 5 processors. These experiments done on the actual version of ReCo have shown the importance of the interprocess communication in our implementation. The ratio computation/communication is about 20% of the global execution time.

References

[1] Iliès Alouini. Concurrent garbage collection for concurrent rewriting. In J. Hsiang, editor, *Proceedings 6th Conference on Rewriting Techniques and Applications, Kaiserslautern (Germany)*, volume 914 of *Lecture Notes in Computer Science*, pages 132–146. Springer-Verlag, 1995.

[2] Iliès Alouini and Claude Kirchner. Conditional concurrent rewriting. Research report 2777, INRIA, Inria Lorraine, March 1996.

[3] Iliès Alouini and Claude Kirchner. Toward the concurrent implementation of computational systems. In *Proceedings 5th Conference on Algebraic and Logic Programming, to appear*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[4] Claude Kirchner and P. Viry. Implementing parallel rewriting. In B. Fronhöfer and G. Wrightson, editors, *Parallelization in Inference Systems*, volume 590 of *Lecture Notes in Artificial Intelligence*, pages 123–138. Springer-Verlag, 1992.