Scheduling in Or-Parallel Prolog Systems: Survey and Open Problems

Andrzej Ciepielewski

Received June 1991; Revised June 1992

Implementation of or-parallel Prolog systems offers a number of interesting scheduling problems. The main issues are the interaction between memory models and scheduling, ordering of multiple solutions, and scheduling of speculative work. The problems occur partly because of the design choices (e.g. the choice of a memory model), and partly because of the desire to maintain observational equivalence between parallel and sequential implementations of Prolog, while achieving high efficiency. In the first part of this paper a common framework for discussing scheduling in or-parallel systems is introduced, and also a collection of issues that must be addressed in such systems is presented. In the second part of the paper we survey a number of solutions to these problems comparing their efficiency whenever possible. We close the survey with a short discussion of open problems.

KEY WORDS: Prolog; parallel execution; scheduling; pruning; side-effects.

1. INTRODUCTION

The 80’s witnessed a growing acceptance of parallel processing as a tool for achieving superior performance and sometimes as the only possibility to tackle a problem. The main emphasis was placed on the use of parallel computers for solving hard scientific problems. Scientific computing is the reign of the traditional programming languages, mainly Fortran and C. In the same period, research on declarative programming languages, and in particular logic programming, was also heavily oriented towards utilizing existing and forthcoming parallel architectures. The work on parallelizing
Prolog, the most popular logic programming based language,\(^1\) is a substantial part of this effort.

Broadly speaking, two types of parallelism can be extracted from a Prolog program. The first, and-parallelism, stems from the possibility offered by Prolog semantics for simultaneous execution of several subproblems. The second, or-parallelism, originates from the ability of Prolog to find more than one solution to a problem. This paper is concerned only with the latter type of parallelism, in particular with scheduling problems arising in or-parallel systems. Other issues and other types of parallel Prolog systems are presented, for example, in a collection of papers edited by Biswas and Ciepielewski.\(^2\) Important papers on concurrent logic programming systems have been collected by Shapiro.\(^3\)

The main property which distinguishes or-parallel Prolog systems from other parallel systems is the source of parallelism—simultaneous search for several solutions. The ability to conduct the parallel search can be utilized to get several answers, or even to get a single solution when it is not known how it can be reached (a generalization of if-then-else). Some examples of problems where the parallel search can be utilized are: natural language parsing, expert systems, theorem proving, and circuit testing. Generally speaking, a solution to any problem which can be broken up into a set of independent subproblems, rather like independent iterations in do-loops in Fortran or recursive calls in a Quick-sort program, can be fairly easily and efficiently mapped on an or-parallel solution.

A distinguishing property of many problems solved in or-parallel systems is the unpredictable nature of their inherent parallelism. It is very difficult to determine at compile time how such problems should be partitioned in order to utilize the parallelism efficiently. Although some work on using static partitioning methods developed for Fortran on Prolog programs is being done,\(^4\) all systems to date use dynamic scheduling. An associated issue is the problem of scheduling speculative work which occurs when only one of many ways to solve a problem is needed.

The goal of this paper is to present the main results from the research on scheduling in or-parallel systems and to expose unsolved problems. The main issues to be discussed are the interaction between memory models and scheduling, techniques for automatic ordering of side-effects, and handling of speculative work. The first issue is somewhat analogous to the cache sensitive scheduling where the choice of a task is influenced by the content of caches.\(^5\) The second issue occurs when a language designed for sequential execution if implemented on a multiprocessor. Finally, the third issue occurs in parallel search algorithms.

The rest of this paper is organized as follows: In Section 2 a framework for discussing scheduling in or-parallel systems is presented, and