A Component-Based Parallel Constraint Solver

Peter Zoeteweij* and Farhad Arbab

CWI, P.O. Box 94079, 1090 GB Amsterdam, the Netherlands
{P.Zoeteweij,Farhad.Arbab}@cwi.nl

Abstract. As a case study that illustrates our view on coordination and component-based software engineering, we present the design and implementation of a parallel constraint solver. The parallel solver coordinates autonomous instances of a sequential constraint solver, which is used as a software component. The component solvers achieve load balancing of tree search through a time-out mechanism. Experiments show that the purely exogenous mode of coordination employed here yields a viable parallel solver that effectively reduces turn-around time for constraint solving on a broad range of hardware platforms.

Keywords: component-based software engineering, coordination, parallelization, constraint solving.

1 Introduction

Over the past years, constraint solving has been a useful test case for coordination techniques. [3, 8, 9, 14]. One of the reasons is that because of the general nature of the constraint programming paradigm, any constraint solver inevitably supports only a specific class of constraint satisfaction problems (CSP’s). By having different solvers, supporting different classes of specialized problems cooperate to solve a more general problem, a broader range of problems can be addressed. The goal of having stand-alone constraint solvers cooperate in a uniform and structured way brings solver cooperation into the area of coordination programming and component-based software engineering.

Constraint solving is an NP-complete problem. Efficient algorithms exist for some classes of CSP’s, and when completeness is not important we may be able to find a solution to a CSP quickly by using local search techniques. Nevertheless, generally constraint solving comes down to tree search. In this paper, we deal with a specific mode of solver cooperation that aims at reducing the turn-around time of constraint solving through parallelization of tree search. Contrary to other modes of solver cooperation, parallel constraint solving has received little attention from a coordination point of view.

The primary aspect of our approach is to equip a tree search based constraint solver with a time-out mechanism. When a CSP can be solved before the elapse

* Supported by NWO, the Netherlands Organization for Scientific Research, under project number 612.069.003.
of a given time-out, such a solver simply produces all solutions that it has found
(or the solution that it has found, if we are not interested in all solutions). Otherwise it also produces some representation of the work that still needs to
be done. For tree search, this is a collection of (disjunct) subproblems that must
still be explored: the search frontier. These subproblems are then re-distributed
among a set of identical solvers that run in parallel. The initial solver is part of
this set, and each solver in the set may split its input into further subproblems,
when its time-out elapses. The aim of the time-out mechanism is to provide
an implicit load balancing: when a solver is idle, and there are currently no
subproblems available for it to work on, another solver is likely to produce new
subproblems when its time-out elapses. We expect to be able to tune the time-out
value such that it is both sufficiently small to ensure that enough subproblems
are available to keep all solvers busy, and sufficiently large to ensure that the
overhead of communicating the subproblems is negligible. The idea of using time-
outs is quite intuitive, but to our knowledge, its application to parallel search is
novel.

Rather than a parallel algorithm, we present this scheme as a pattern for
constructing a parallel constraint solver from component solvers. The only re-
quirement is that these components can publish their search frontiers. We believe
that this requirement is modest compared to building a parallel constraint solver
from scratch. Our presentation of the scheme in Section 3 uses the notion of ab-
stract behavior types, and the Reo coordination model. These, and the relevant
aspects of constraint solving are introduced in Section 2. To test the concept, we
equipped a constraint solver with the time-out mechanism, and implemented the
coordination pattern as a stand-alone distributed program. In Section 4 we give
an account of this implementation, and in Section 5 we describe the experiments
that were performed to test the parallel solver. Compared to parallelizing an
existing constraint solver, the component-based approach has further benefits.
These are discussed in Section 6, together with related work and directions for
future research.

2 Preliminaries

To make the paper self-contained, in this section we provide the necessary back-
ground on constraint solving (2.1), abstract behavior types, and Reo (2.2).

2.1 Constraint Solving

Constraint solving deals with finding solutions to constraint satisfaction prob-
lems. A CSP consists of a number of variables and their associated domains (sets
of possible values), and a set of constraints. A constraint is defined on a subset
of the variables, and restricts the combinations of values that these variables
may assume. A solution to a CSP is an assignment of values to variables that
satisfies all constraints. Tree search in constraint solving performs a systematic
exploration of assignments of values to variables: at every node of the search