Exploiting Parallelism in Tabled Evaluations*

Juliana Freire, Rui Hu, Terrance Swift, David S. Warren

Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
Email: {juliana, ruihu, tswift, warren}@cs.sunysb.edu

Abstract. This paper addresses general issues involved in parallelizing tabled evaluations by introducing a model of shared-memory parallelism which we call table-parallelism, and by comparing it to traditional models of parallelizing SLD. A basic architecture for supporting table-parallelism in the framework of the SLG-WAM[14] is also presented, along with an algorithm for detecting termination of subcomputations.

1 Introduction

The deficiencies of SLD resolution are well known, and extended efforts have been made to remedy these deficiencies. For instance, while SLD can be combined efficiently with negation-by-failure in SLDNF, the semantics of SLDNF have proven unacceptable for many purposes, in particular for non-monotonic reasoning. Even without negation SLD is susceptible to infinite loops and redundant subcomputations, making it unacceptable for deductive databases.

The latter deficiency, that of repeating subcomputations, has given rise to several systems which table subcomputations: OLDT [16], SLD-AL [18], and SLG [4, 3] are three tabling methods which have been implemented. At an abstract level, systems which use magic evaluation can also be thought of as tabling systems. Substantiation for this claim stems both from the asymptotic results of [12] and the experimental results of [15]. Tabling is also be relevant for computing the well-founded semantics: besides SLG, well-founded ordered search [13] and the tabulated resolution of [2] also use tabling.

While nearly all of these approaches are sequential, there is a natural parallelism inherent in these evaluation methods which we call table-parallelism. At a general level, the idea behind table-parallelism is simple: the table can be thought of as a large structured buffer through which cooperating threads communicate. Parallelization then can take place at each tabled subgoal. If a subgoal is called and is not in the global table, an entry for it is created, and answers are derived for the subgoal using program clause resolution. Otherwise, if this subgoal is a variant of subgoal present in the table, it will consume the answers stored in the table.

* This work was partially supported by CAPES-Brazil and NSF grants CCR-9102159, CCR-9123200, CCR-9404921, CDA-9303181.
t(x,u)

r(1,2), r(1,3), r(3,2),
b(2,3), b(3,4), b(3,5).
s(X,Y) :- b(X,Z), s(Z,Y), s(2,3) s(2,4) s(2,5) s(3,4) s(3,5) s(2,3) s(2,4) s(2,5).
t(X,Y) :- r(X,Z), s(Z,Y).

Fig. 1. Join in Parallel Prolog

Fig. 1 illustrates how, by reducing redundant subcomputations, tabling can improve performance over an SLD-based resolution method ([1, 15]). An or-parallel (or any sequential) Prolog would have to compute the relation s(2,Y) twice. In a tabling system, if the predicate s were declared as tabled it would be computed only once, and any other subsequent call would simply consult the table. This avoidance of redundant computation has long been recognized as necessary for data-oriented queries. On the other hand, a bottom-up method such as tabling can also introduce an overhead for copying information into and out of tables, making it unsuitable for programs such as list recursion.

We believe that, just as a mixture of SLD and tabled evaluation is arguably most useful for sequential evaluation, a combination of SLD-based and- and or-parallelism and table-parallelism will prove most useful for parallel evaluation. In terms of practical programs, the mixture of SLD and tabling has proven useful for program analysis [7] over flat domains [5], and for the Unification Factoring compiler optimization [6]. While implementation has so far been sequential, both of these algorithms contain bottom-up subcomputations that are amenable to table-parallelism, along with top-down computations amenable to traditional SLD parallelism.

The structure of the paper is as follows:

- We briefly overview general concepts underlying tabling using SLG formalism, and its implementation in the SLG-WAM.
- Using SLG terminology, we introduce the abstract model of table-parallelism and discuss its relation to more traditional and- and or- parallelism, and briefly discuss issues for the analysis of table-parallelism.
- We describe and prove the correctness of a parallel completion algorithm which detects termination of subcomputations, allowing a great deal of concurrency between subcomputations.
- In the framework of the SLG-WAM [14], we present the extensions to tabling operations necessary to implement table-parallelism.

2 A brief overview of SLG

SLG evaluates programs by keeping a table of answers to subgoals, and resolving repeated subgoals against answer clauses from the table rather than against