PHYSICAL SHARING OF CLAUSES IN PARALLEL SAT SOLVERS

por

Juan Luis Olate Hinrichs

Patrocinante: Leonardo Ferres

Memoria presentada
para la obtención del título de

INGENIERO CIVIL INFORMÁTICO

Departamento de Ingenería Informática y Ciencias de la Computación

de la

UNIVERSIDAD DE CONCEPCIÓN

Concepción, Chile
Octubre, 2011

Abstract

Your abstract goes here...

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

One of the most well-known problems in computer science is the satisfiability (SAT) problem. This is because this was the first problem to be proved to be NP-complete [1], proof known as the Cook-Levin theorem[1]. One year later, in 1972, Karp proved in [2] that many common combinatorial problems could be reduced in polynomial time to instances of the SAT problem, thus drawing even more attention to SAT problems by the scientific community. Because many combinatorial problems can be reduced to SAT, it is not strange to find many practical problems with useful applications (such as circuit design and automatic theorem proving) that could be solved if there was an efficient algorithm to solve the SAT problem. Unfortunately, because of the NP-complete nature of SAT, such algorithm has not been found yet, but also has not been proven to be in-existent. Many researchers suspect such efficient algorithm to solve all SAT instances does not exist, so instead of trying to solve the NP-complete problem, they try to improve the current SAT solving algorithms.Over the years, SAT solvers have shown impressive improvement, the first complete algorithm, the Davis Putnam algorithm [3], was very limited and could only handle problems with around ten variables. Today, modern SAT solvers can handle instances with millions of variables, making such solvers suitable even for industrial application. In the next chapter we will point out the main features that have improved SAT solvers significantly.

In the last decade parallel computing has become increasingly popular. As CPU manufacturers have found difficult and expensive to keep increasing the clock speed of processors, they have instead turn to increase the number of cores each chip has. Unfortunately, if the algorithms are not thought to be run in parallel, more cores will bring small improvements. This is the reason why there is a growing concern to

[1]They both proved it independently.

parallelize algorithms so that they can take advantage of many-cores architectures of today's computers. In SAT solving it is no different. The annual SAT competition [1], an event to determine which is the fastest SAT solver, has two main categories; sequential SAT solvers and parallel SAT solvers. In the last years parallel SAT solvers have outperformed sequential solvers in total wall clock time, so the interest in parallel solvers has grown, new designs and approaches have been explored for this kind of solvers. One of the most successful approaches to implement a parallel SAT solver is the portfolio approach. This approach is basically to run different solvers in parallel and wait for one of them to solve the problem. It's a very simple and straight forward approach of parallelization, but we have also encountered one drawback to it: as we add more solvers to different cores of a single chip, the overall performance of the parallel solver decreases in around 20-40%. In this work we will attempt to find the source of this problem and explore possible solutions to it.

[1] www.satcompetition.org

Chapter 2

Background and Related Work

2.1 SAT solvers

2.1.1 The SAT problem

Given a set of boolean variables Σ, a literal L is either a variable or the negation of a variable in Σ, and a *clause* is a disjunction of literals over distinct variables[1]. A propositional sentence is in *conjunctive normal form* (*CNF*) if it has the form $\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n$, where each α_i is a clause. The notation of sentences in CNF we will be using are sets. A clause $l_1 \vee l_2 \vee ... \vee l_m$, where l_i is a literal, can be expressed as the set $\{l_1, l_2, ..., l_m\}$. Furthermore, the CNF $\alpha_1 \wedge \alpha_2 \wedge ... \wedge \alpha_n$ can be expressed as the set of clauses $\{\alpha_1, \alpha_2, ..., \alpha_n\}$. With these conventions, a CNF Δ is valid if Δ is the empty set: $\Delta = \emptyset$. A CNF Δ will be inconsistent if it contains the empty set: $\emptyset \in \Delta$. Given a CNF Δ, the SAT problem is answering the question: Is there an assignment of values for variables in Σ, such that Δ evaluates to true? The NP-completeness of this question lies in the combinatorial nature of the problem; to solve it one would need to try all different assignments of variables in Σ, the number of possible assignments grows exponentially as $|\Sigma|$ grows.

2.1.2 Resolution

The resolution inference rule [4] is defined as follows. Let P be a Boolean variable, and suppose that Δ is a CNF which contains clauses C_i and C_j, where $P \in C_i$ and $\neg P \in C_j$. The resolution inference rule allows us to derive the clause $(C_i - \{P\}) \cup (C_j - \{\neg P\})$, which is called a *resolvent* that is obtained by *resolving* C_i and C_j. A simple example is the CNF $\{\{A, \neg B\}, \{B, C\}\}$. Applying resolution to this two clauses would derive the clause $\{A, C\}$, which would be called a B-*resolvent*.

[1] That all literals in a clause have to be over distinct variables is not standard.

Resolution is incomplete in the sense that it is not guaranteed to derive every clause that is implied by the CNF, but it is *refutation complete* on CNFs. It is guaranteed that resolution will derive the empty clause if the given CNF is unsatisfiable.

Unit resolution is an important special case of resolution. It's a resolution strategy which requires that at least one of the resolved clauses has only one literal. Such clause is called a *unit clause*. The importance of unit resolution does not rely on its completeness (it's actually not refutation complete, as resolution is), but one can apply all possible unit resolution steps in time linear to the size of a given CNF. Its efficiency makes it a key technique employed by modern solvers.

2.1.3 Conditioning

Conditioning a CNF Δ on a literal L consists of replacing every occurrence of L by the constant **true**, replacing $\neg L$ with **false**, and simplifying accordingly. The result of conditioning Δ on L will be denoted by $\Delta|L$ and can be defined as follows:

$$\Delta|L = \{\alpha - \{\neg L\} | \alpha \in \Delta, L \notin \alpha\}$$

This means that the new set of clauses $\Delta|L$ will be all the clauses in Δ that do not contain L, but with the literal $\neg L$ removed. The clauses that contain L are removed because they are now satisfied, since we made L **true**. $\neg L$ is removed from the remaining clauses because it was set to **false** and no longer has any effect.

The definition of conditioning can be extended to multiple literals. For example, the CNF:

$$\Delta = \{\{A, B, \neg C\}, \{\neg A, D\}, \{B, C, D\}\}$$

can be conditioned as $\Delta|CA\neg D = \{\emptyset\}$ (an inconsistent CNF). Moreover, $\Delta|\neg CD = \emptyset$ (a valid CNF).

2.1.4 Search trees

One way to picture the search for a possible assignment of variables that satisfies the CNF formula is to use a tree. For example, given $\Sigma = \{A, B, C\}$ and $\Delta = \{\{\neg A, B\}, \{\neg B, C\}\}$, figure 2.1 shows the search tree for this CNF. Each node of the tree represents a variable, for each level we have a different variable. The branches are the different truth values the variable can be assigned. Each w_i represents a possible truth assignment of the variables in Σ. Note that w_1, w_5, w_7 and w_8 are all assignments that satisfy the CNF, while w_2, w_3, w_4 and w_6 do not.

2.1.5 The DPLL algorithm

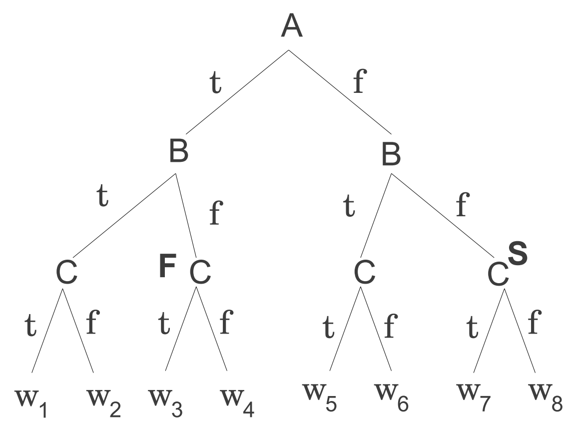

Figure 2.1: A search tree for the CNF $\Delta = \{\{\neg A, B\}, \{\neg B, C\}\}$.

The DavisPutnamLogemannLovel (DPLL) algorithm is the base of all modern SAT solvers. Many refinements have made to this algorithm over the last decade, which have been significant enough to change the behaviour of the algorithm, but it is still important to know it for understanding modern solvers.

2.1.6 PaMiraXT

PaMiraXT [5] is one of the first attempts to use a shared clause database in a parallel SAT solver.

Chapter 3

Important stuff...

Chapter 4

Conclusions

Bibliography

[1] S. A. Cook. The complexity of theorem-proving procedures. *proc. 3rd ann. ACM symp. on theory of computing*, pages 151–158, 1971.

[2] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

[3] Davis Putnam and Martin Hillary. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[4] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[5] Tobias Schubert, Matthew Lewis, and Bernd Becker. Pamiraxt: Parallel sat solving with threads and message passing. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:203–222, 2009.