PHYSICAL SHARING OF CLAUSES IN PORTFOLIO APPROACH PARALLEL SAT SOLVERS

por

Juan Luis Olate Hinrichs

Patrocinante: Leo Ferres

Propuesta de

MEMORIA DE TÍTULO

Departamento de Ingenería Informática y Ciencias de la Computación

de la

UNIVERSIDAD DE CONCEPCIÓN

Concepción, Chile

Abril, 2012

Table of Contents

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 [2], proof known as the Cook-Levin theorem[1]. One year later, in 1972, Karp proved in [4] 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 inexistent. 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 [7], 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 with no physical sharing of clauses. This approach is basically to run different solvers in parallel, each thread keeps its own copy of the whole problem in memory, 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%. Experiments strongly suggest that this decrease in performance is caused by memory cache. Because all cores in a single chip share the same last level cache, and because each thread holds a copy of the original problem in memory, the more threads we add, the bigger the amount of data we have to handle will be. Since there is only one last level cache shared among all threads (assuming one thread per core), the amount of total accesses from the last level cache to main memory will increase, because now there is a bigger volume of data to handle.

Our idea is to keep the line of a parallel portfolio approach SAT solver, but implementing a shared clause database among all solvers in the different threads. By doing this we expect to lower the amount of data all threads have to handle, since all threads will be using a common pool of clauses, instead of copies of it. This would mean a lower amount of accesses from the last level cache to main memory, and thus better performance.

[1] www.satcompetition.org

Chapter 2

Objectives

The objectives for this work are the following:

- Empirically quantify the decrease in peformance, as you add threads, of paralel SAT solvers using the portfolio approach and that do not share clauses physically.
- Prove with experimental results that this decrease is due to memory cache.
- Propose and implement a CDCL (Conflict Driven Clause Learning, a modern type of SAT solver) parallel SAT solver, with portfolio approach and a physically shared clause database, that could lower the decrease in performance when adding threads.
- Test the new SAT solver, discuss the results and draw conclusions that could help future parallel SAT solver designers.

Chapter 3

Methodology

The following activities will be done to complete the work:

1. *Bibliographic research*: In this step we will gather and read articles and books of related SAT solving work. We are aiming to find the best performing parallel SAT solvers and get to know their internal designs.

2. *Testing of existing parallel solvers*: This work is based on experimental evidence that the best performing parallel SAT solvers suffer from considerable decrease in performance when adding more threads. We will perform various tests on some parallel SAT solvers to quantify and confirm the initial evidence.

3. *Cache tests*: After proving that there is a significant decrease in performance when adding threads to a typical portfolio approach SAT solver, we will prove through experimental results on existing solvers that the problem is memory cache.

4. *Propose a new solver*: We will propose a new design of a parallel SAT solver with a shared clause database that should lower the decrease in performance when adding threads.

5. *Result analysis*: With the new proposed solver, we will run experiments on it and analyze results.

6. *Report writting*: All the previous work will be written as a report to present as a graduation proyect.

Chapter 4

Background and Related Work

4.1 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.

4.2 SAT solvers

All modern solvers today are CDCL (Conflict Driven Clause Learning) [6] SAT solvers. Given a CNF φ, a partial assignment of variables ν, 1 outlines the general structure of a CDCL SAT solver, where x is a variable, v a truth value and β a number. We will shortly explain the main functions of this algorithm, but the details will be covered in the final work.

- UnitPropagation consists of iterately deducting the truth value of variables. The values are deduced by logical reasoning on φ and ν.

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

Algorithm 1: Typical CDCL algorithm

Input: A CNF φ and a variable assignment ν

if (UnitPropagation(φ,ν)==**CONFLICT**) **then**
    return UNSAT.
$dl \leftarrow 0$
while (**not** AllVariablesAssigned(φ,ν)) **do**
    (x,v)=PickBranchingVariable(φ,ν)
    $dl \leftarrow dl + 1$
    $\nu \leftarrow \nu \cup \{(x,v)\}$
    if (UnitPropagation(φ,ν)==**CONFLICT**) **then**
        β=ConflictAnalysis(φ,ν)
        if ($\beta < 0$) **then**
            return UNSAT
        else
            Backtrack(φ,ν,β)
            $dl \leftarrow \beta$
return SAT

- PickBranchingVariable consists of selecting a variable to assign, and the respective value. Heavily relies in heuristics to pick variables.

- ConflictAnalysis consists of analyzing the most recent conflict (a conflict occurs when no variable assignment with the current ν can satisfy φ) and learning a new clause from the conflict.

- Backtrack undoes variable assignments as computed by ConflictAnalysis.

- AllVariablesAssigned tests whether all variables have been assigned a truth value.

4.3 Parallel SAT solvers

As mentioned before, some parallel SAT solvers have performed at the top of the last SAT competitions, but even though they all fall into the parallel solvers category, their parallel strategies and implementations vastly differ from each other. We mainly classify parallel SAT solvers into two categories: Portfolio approach solvers and divide-and-conquer solvers.

The main idea behind portfolio approach solvers is the fact that different kinds of sequential solvers will perform differently for different kinds of SAT problems. The portfolio approach is a very straight forward strategy: They run a group of sequential solvers in parallel, each with different heuristic random values and/or different search strategies. The time they take to solve the problem will be the time of the fastest solver in the group of solvers running in parallel. Although all portfolio approach solvers share this same principle, they also have quite different kinds of implementations. We identify in this group the solvers that are pure portfolio approach, the ones that share clauses only logically, and the ones that share clauses physically and logically.

Solvers which are pure portfolio approach have the most simple design. They run completely independent solvers in parallel and wait for one of them to give an answer. Despite their simplicity, the solver ppfolio [8], a pure portfolio approach solver, was the winner of the crafted and random categories, and second place in the application category of the 2011 SAT competition of parallel solvers.

On the other hand, we have more elaborated portfolio approach solvers, which can also share clauses logically between their different solvers. One of the advantages of CDCL solvers is the fact that they can learn new lemmas as they solve a SAT problem. These new lemmas will provide additional information during the solution search, so that the solver doesn't fall into previous fruitless search paths (there are also some drawbacks to adding new lemmas, which are addressed by clause database cleanups). The idea is that different solvers running in parallel can share their learned lemmas so that they all benefit from what other solvers have learned and improve their own search. An example of these kind of solvers is ManySAT [3], which won the 2009 SAT competition in the parallel solver application category. ManySAT has its own sequential state-of-the-art SAT solver and runs different instances of it in parallel, using different VSIDS [10] heuristics (branching heuristics) and restart policies for each of it, both of which account for random factors in the solver. The difference with pure portfolio approach solvers, is that ManySAT also shares learned lemmas between solving threads. It is called logical sharing of clauses, because the lemmas are passed as messages between threads and they never share the same physical information in memory. The advantage of logical sharing is that it is easier to implement message passing between threads, than having threads reading and modifying the same memory locations, which often requires locks that could hinder the overall solver performance. One of the best parallel performing solvers, Plingeling [1], also shares clauses logically. It is a very weak sharing though, since it only shares unit lemmas and it does so through messafe passing, using a master thread to coordinate messages between worker threads.

Portfolio approach solvers that share clauses physically have the same strategy as mentioned before, but they share clauses by allowing threads to access the same memory locations, instead of message passing. One solver in this category is SarTagnan [5], which shares clauses logically and physically.

Divide-and-conquer solvers do not try to run different solvers in parallel, they run one solving instance, but try to parallelize the search and divide it between the different threads. A common strategy to divide the search space is to use guiding paths. A guiding path is a partial assignment of variables in Σ, which restricts the search space of the SAT problem. A solver that divides its search space with guiding paths will assign threads to solve the CNF with the given partial assignment from the guiding path the thread was assigned with. Once a thread finishes searching a guiding path with no success, it can request another to keep searching. MiraXT [9] is a divide-and-conquer SAT solver which uses guiding paths. Moreover, different threads solving different guiding paths also share a common clause database, in which they store their learned lemmas. This is another example of physical clause sharing.

The work will mainly consist of building a parallel portfolio approach solver, which uses a smilar shared clause database system as proposed by MiraXT.

Chapter 5

Planification

Image 5.1 shows the planification of work for this project.

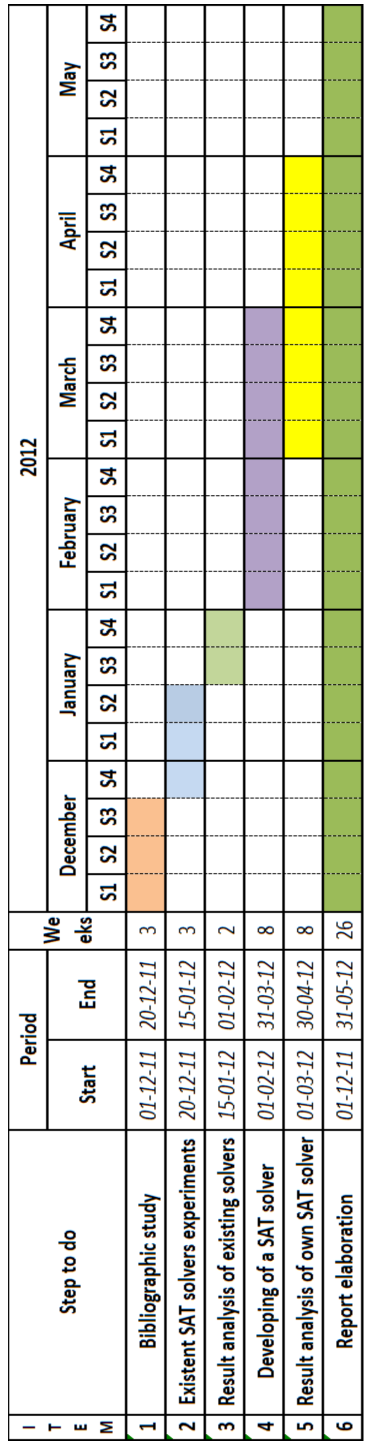

Figure 5.1: Planification of work to be done in the project.

Bibliography

[1] Armin Biere. Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010. http://baldur.iti.uka.de/sat-race-2010/descriptions/solver_1%2B2%2B3%2B6.pdf, 2010. [Online; accessed 01-April-2012].

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

[3] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. Manysat: a parallel sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, 6:245–262, 2009.

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

[5] Stephan Kottler and Michael Kaufmann. SArTagnan - A parallel portfolio SAT solver with lockless physical clause sharing. In *Pragmatics of SAT*, 2011.

[6] Joao P. Marques-Silva, Ines Lynce, and Sharad Malik. *Conflict-Driven Clause Learning SAT Solvers*, chapter 4, pages 131–153.

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

[8] O. Rousell. Description of ppfolio. http://www.cril.univ-artois.fr/~roussel/ppfolio/solver1.pdf, 2010. [Online; accessed 01-April-2012].

[9] Tobias Schubert, Matthew Lewis, and Bernd Becker. Pamiraxt: parallel sat solving with threads and message passing. *Journal of Satisfiability, Boolean Modeling and Computation*, 6(4):203–222, 2009.

[10] Jaeheon Yi. The effect of vsids on sat solver performance. 2009.