Implementation of a Parallel SAT Solver

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Technology and Master of Technology

by

Sahil Agarwal

(*Roll no.* 10D070017)

Supervisor:

Prof. Madhav Desai

Department of Electrical Engineering

Indian Institute of Technology Bombay

2014

Abstract

SAT solvers are used in various domains like artificial intelligence, circuit design, automatic theorem proving, etc. With increasingly large search spaces, greater speed in solving is required. Modern SAT solvers are using available multi-core systems to achieve speedup in SAT solving. In this work we present a parallel SAT solver based on Orthonormal function decomposition [1]. The solver will be implemented on clusters of computers, using OpenMP and MPI (Message Passing Interface). The main features of the program are described. The solver could also be used to solve other NP-hard problems like Travelling Salesman problem, Discrete Logarithm problem and more.

Contents

Chapter 1

Introduction

1.1 Boolean Satisfiability Problem

The Boolean Satisfiability Problem (SAT) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. An interpretation is the assignment of TRUE or FALSE values to variables of the given formula. If an assignment exists such that the formula evaluates to TRUE, the formula is called satisfiable. If no such assignment exists, the function expressed by the formula is identically FALSE for all possible variable assignments and the formula is unsatisfiable. A *SAT solver* is the algorithm which solves the SAT problem.

Definitions and Terminology: A propositional logic formula, also called Boolean expression, is built from variables, operators AND (conjunction, also denoted by $\wedge$), OR (disjunction, $\vee$), NOT (negation, $\neg$), and parentheses. A literal consists of a variable A and is either positive (A) or negated ($\neg A$). A clause is a disjunction of literals. A formula is in the conjunctive normal form (CNF) if it is a conjunction of clauses. A boolean formula is generally given to the SAT solver in CNF.

1.2 Overview of SAT solvers

1.2.1 DPLL Algorithm

Most modern SAT-solvers are based on the Davis-Putnam-Loveland-Logemann (DPLL) algorithm [2]. It is a complete, backtracking-based search algorithm for solving the CNF-SAT problem. [1]

The *backtracking algorithm* runs by choosing a literal, assigning a truth value to it, simplifying the formula and then recursively checking if the simplified formula is satisfiable; if this is the case, the original formula is satisfiable; otherwise, the same recursive check is done assuming the opposite truth value. This is known as the *splitting rule*, as it splits the problem into two simpler sub-problems.

The DPLL algorithm enhances over the backtracking algorithm by using the following rules at each step:

- *Unit propagation* - If a clause is a unit clause, i.e. it contains only a single unassigned literal, this clause can only be satisfied by assigning the necessary value to make this literal true.

- *Pure literal elimination* - If a propositional variable occurs with only one polarity in the formula, it is called pure. Pure literals can always be assigned in a way that makes all clauses containing them true. Thus, these clauses do not constrain the search anymore and can be deleted.

Unsatisfiability of a given partial assignment is detected if one clause becomes empty, i.e. if all its variables have been assigned in a way that makes the corresponding literals false. This is known as the *conflict clause*. Satisfiability of the formula is detected either when all variables are assigned without generating the empty clause, or, in modern implementations, if all clauses are satisfied. Unsatisfiability of the complete formula can only be detected after exhaustive search.

[1] Henceforth, the input formula is in CNF unless specified otherwise.

1.2.2 Sequential SAT solvers

The idea to analyze the conflict clause further led to the conflict driven clause learning (CDCL) [3]. Resolving the conflict clause and the clauses which have been used in the implications, new clauses are learned. These learned clauses can be added to the given formula leading to an improved backtracking behavior, where larger parts of the search tree are closed by a single conflict. The most commonly used version of this algorithm is described in [4].

The most commonly used SAT-solver that implements most of the mentioned techniques is MINISAT ([5]). This solver provides a basis for many sequential and parallel SAT-solvers including the one implemented in this work.

1.2.3 Parallel SAT solvers

The recursive application of the split rule in the DPLL algorithm provides a natural way to parallelize the search for a satisfying assignment [6]. Initially, each computing node receives the given formula. Thereafter, only partial interpretations are communicated such that each computing node receives a different partial interpretation. This is the *Decomposition-based approach*. The first parallel SAT-solvers used single-core CPUs which communicated via a network. When shared memory architectures became available, shared memory communication was utilized. A popular decomposition-based parallel solver is PMSAT [7]. It is based on MiniSAT 1.14 and MPI (Message Passing Interface).

Another popular technique in parallel SAT solving is the *Portfolio-based technique*. Instead of implementing cooperative parallelism by splitting the search space, competitive parallelism is applied where all parallel solvers try to find a solution for the same SAT instance.

1.3 A new parallel SAT solver

1.3.1 High-level description

This work highlights a decomposition-based parallel SAT solving algorithm. The algorithm is based on the application of a well known generalized Boole-Shannon orthonormal (ON) expansion of Boolean functions [1]. This is basically an extension of the DPLL algorithm in which the splitting rule is generalized to several variables in terms of ON terms. The subproblems created recursively are solved on parallel computing nodes. After a threshold is reached, direct sequential search is performed on the subproblems at the individual nodes.

1.3.2 Motivation

Until now, modern CPUs contain only few cores. Recent parallel SAT-solvers for such shared-memory architectures are mainly based on the techniques developed for sequential SAT-solvers. As soon as the individual solvers running on different cores share memory, the efficiency of the individual solvers decreases. None of these parallel solvers seem to scale well if hundreds or even thousands of cores become available.

The algorithm suggested above has the potential to make use of large number of cores due to ease in its parallelization.

Chapter 2

Implementation of the SAT solver

2.1 Introduction

The parallel SAT solver will use the Orthonormal (ON) expansion based algorithm in [1]. This algorithm is a generalized version of the DPLL algorithm [2].

2.2 Plan

- *Sequential Stage* - A sequential form of the algorithm has been implemented in the first stage. Although this will not give an idea of execution time of the parallel algorithm, it can be used to verify the correctness of the approach.
- *Parallel Stage 1* - The algorithm will undergo slight modifications to achieve parallelism using OpenMP (Open Multi-Processing). OpenMP is an API that supports multi-platform shared memory multiprocessing programming in C, C++ ... [8]. This is a single machine multi-core implementation.
- *Parallel Stage 2* - Large-scale parallelization over multiple machines and cores will be achieved by using MPI (Message Passing Interface). MPI was also used in [7].