pmcSAT

Ricardo Marques, Luis Guerra e Silva, Paulo Flores and L. Miguel Silveira
INESC-ID / IST-TU Lisbon
Rua Alves Redol, 9, 1000-029 Lisbon, Portugal
{rsm,lgs,pff,lms}@algos.inesc-id.pt

***Abstract*—This document describes the SAT solver PMCSAT, a conflict-driven clause learning (CDCL) portfolio solver that launches multiple instances of the same basic solver using different heuristic strategies, for search-space exploiting and problem analysis, which share information and cooperate towards the solution of a given problem.**

I. Introduction

PMCSAT is a portfolio-based multi-threaded, multi-core SAT solver, built on top of MINISAT [1]. The general strategy pursued in this solver is to launch multiple instances of the same solver, with different parameter configurations, which cooperate to a certain degree by sharing relevant information when searching for a solution. This approach has the advantage of minimizing the dependence of current SAT solvers on specific parameter configurations chosen to regulate their heuristic behavior, namely the decision process on the choice of variables, on when and how to restart, on how to backtrack, etc.

II. Main Techniques

The solver uses multiple threads (eight currently), which explore the search space independently, following different paths, due to the way each thread is configured.

In order to ensure that each thread follows divergent search paths, we defined distinct priority assignment schemes, one for each thread of PMCSAT. Note that the priority of a variable will determine its relative assignment order.

Below are described the different priority schemes that were used.

- **Thread #0/#1** - All the variables have the same priority, therefore this thread mimics the original VSIDS heuristic.
- **Thread #2** - The first half of the variables read from the file have higher priority than the second half.
- **Thread #3** - The first half of the variables read from the file have lower priority than the second half.
- **Thread #4** - The priority is sequentially decreased as the variables are read from the file.
- **Thread #5** - The priority is increased according to its number of occurences in the file.
- **Thread #6** - The priority is decreased according to its number of occurences in the file.
- **Thread #7** - The priority is decreased according to the number of variables that have the same number of common variables.

Threads #0 and #1 use the same priority-scheme, however they have different learnt clause deletion methods.

In [2] the authors show that using a more agressive clause deletion strategy could lead to good results in a CDCL SAT solver, as a result of the overhead reduced in propagation on learnt clauses. Threfore, thread #1 uses a more agressive deletion strategy, while all the other threads follow the MINISAT deletion scheme.

Although each PMCSAT thread exploits independently the search space, this is not just a purely competitive solver. All the threads cooperate by sharing the learnt clauses resulting from conflict analysis, leading to a larger pruning of the search space.

To reduce the communication overhead introduced by clause sharing, and its overall impact in performance, we designed data structures that eliminate the need for read and write locks. These structures are stored in shared memory, which is shared among all threads.

Each thread owns a queue, where the clauses to be shared are inserted. Associated to this queue is a pointer, which marks the last inserted clause, manipulated by the source thread, while every other targed thread owns a pointer that indicates the last read clause from the queue. Therefore, this data structure eliminates the need for a locking mechanism.

A more detailed explanation of the techniques used in this solver can be found in [3].

III. Main Parameters

The internal parameters of PMCSAT are the same as in MINISAT, with the addition of the following:

1) The learnt clauses size condition to be exported. For the SAT competition the clause size limit was set to 8, i. e., only learnt clauses with less than 8 literals are exported and shared with other threads.
2) The threshold for the thread's learnt clause database to be reduced. The initial condition defined for thread #1 is 4000 learnt clauses, with an increment value of 300, as in GLUCOSE 2.1.

IV. Implementation details

1) The programming language used is C++, using pthread for parallel computing.
2) The solver was implemented on top of MINISAT V2.2.0.

V. SAT Competition 2013 Specifics

1) The solver was submitted to all Parallel Tracks: Application SAT+UNSAT, Hard-Combinatorial SAT+UNSAT, Random SAT and Open Track.

2) The compiler used is g++.
3) The optimization flag used is ”-O3”
4) 64-bit binary.
5) The only command-line parameter is the input file

VI. Availability

More information about the PMCSAT solver, including its source code, can be found on the ALGOS research group publicily available website:

http://algos.inesc-id.pt/algos/software.php

Acknowledgment

The authors would like to thank the authors of MiniSat 2.2.0 for making available the source code of their solvers.

This work was partially supported by national funds through FCT, Fundação para a Ciência e Tecnologia, under project ”ParSat: Parallel Satisfiability Algorithms and its Applications” (PDTC/EIA-EIA/103532/2008) and project PEst-OE/EEI/LA0021/2011.

References

[1] N. Een and N. Sorensson, “An extensible sat-solver,” in *SAT*, ser. Lecture Notes in Computer Science, E. Giunchiglia and A. Tacchella, Eds., vol. 2919. Springer, 2003, pp. 502–518.

[2] G. Audemard and L. Simon, “Predicting learnt clauses quality in modern sat solver,” in *Twenty-first International Joint Conference on Artificial Intelligence(IJCAI’09)*, jul 2009, pp. 399–404.

[3] R. S. Marques, L. G. e Silva, P. Flores, and L. M. Silveira, “Improving sat solver efficiency using a cooperative multicore approach,” *International FLAIRS Conference, May 22 - 24, 2013*.