New Concurrent and Distributed Painless solvers: P-MCOMSPS, P-MCOMSPS-COM, P-MCOMSPS-MPI, and P-MCOMSPS-COM-MPI

Vincent Vallade*, Ludovic Le Frioux†, Razvan Oanea*, Souheib Baarir*§, Julien Sopena*†,
Fabrice Kordon*, Saeed Nejati¶, Vijay Ganesh¶
*Sorbonne Université, LIP6, CNRS, UMR 7606, Paris, France
†INRIA, Delys Team, Paris, France
‡Université Paris Nanterre, France
§University of Waterloo, Waterloo, ON, Canada

***Abstract*—This paper describes the solvers P-MCOMSPS and P-MCOMSPS-COM submitted to the parallel track of the SAT Competition 2021; as well as P-MCOMSPS-MPI and P-MCOMSPS-COM-MPI submitted to the cloud track of the SAT Competition 2021. P-MCOMSPS and P-MCOMSPS-MPI are LBD-based, and P-MCOMSPS-COM and P-MCOMSPS-COM-MPI are community and LBD-based.**

I. Introduction

P-MCOMSPS is a concurrent SAT solver built by using the Painless framework [1]. It is a portfolio-based [2] solver implementing a diversification strategy [3], fine control of learnt clause exchanges [4] based on LBD [5], using MapleCOMSPS [6] as a core sequential solver, and where learnt clause strengthening [7] has been integrated. P-MCOMSPS-COM is based on P-MCOMSPS and uses COM and LBD sharing [8]. P-MCOMSPS-MPI (resp. P-MCOMSPS-COM-MPI) is a distributed solver using on each node P-MCOMSPS (resp. P-MCOMSPS-COM), and relying on MPI for termination and clause sharing between nodes.

Section II details the implementation of P-MCOMSPS using Painless and MapleCOMSPS. Section III explains how community and LBD sharing has been implemented in P-MCOMSPS-COM. Finally, section IV explains how our concurrent solvers have been adapted to implement P-MCOMSPS-MPI and P-MCOMSPS-COM-MPI.

II. P-MCOMSPS

This section describes the overall behaviour of our competing instantiation named P-MCOMSPS. Its architecture is highlighted in Fig. 1. It implements the Painless strengthening described in [9].

A. MapleCOMSPS

MapleCOMSPS [6] has been adapted for the parallel context as follows: (1) we parametrized the solver to use either LRB [10], or VSIDS [11] (resp. L and V); (2) we added callbacks to export and import clauses; (3) we added an option to activate or not the Gaussian elimination (GE) preprocessing; (4) we parametrized the solver to use as a variable score comparator either $<$ or $<=$ (resp. head: H and tail: T).

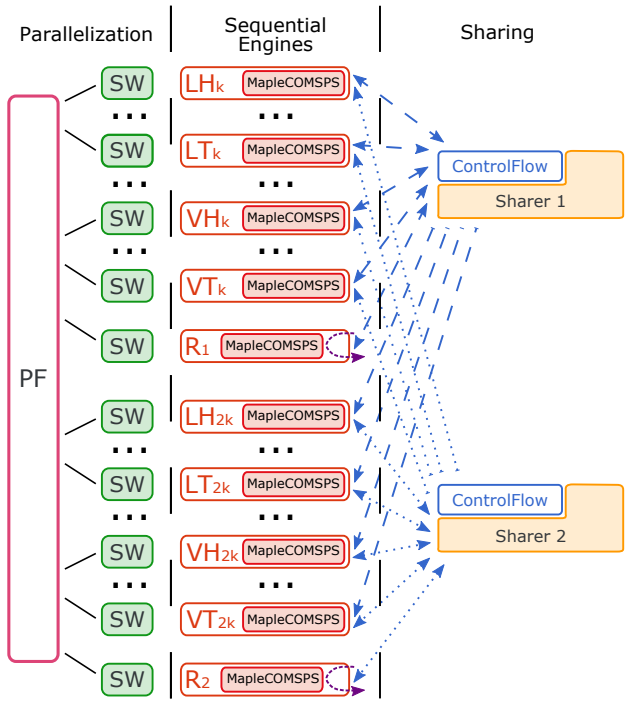

Fig. 1. Architecture of P-MCOMSPS

B. Strengthener

Two *reducer* engines (R in Fig. 1) implement the algorithm introduced in [7]. We implemented the strengthening operation as a decorator of *SolverInterface*. This decorator uses, by delegation, another *SolverInterface* to apply the strengthening, in the present case a MapleCOMSPS solver.

C. Portfolio and Diversification

As depicted in Fig. 1, P-MCOMSPS implements a portfolio strategy (PF), where two solvers are used as reducers, and the other underlying core engines are either LH, LT, VH or VT instances (i.e., combination of V or L, and H or T). For each type of instances, we apply a sparse random

diversification [3]. Moreover, only one of the solvers performs the GE preprocessing.

D. Controlling the Flow of Shared Clauses

In `P-MCOMSPS`, the sharing strategy `ControlFlow` is inspired by the one used by [3], [4]. As highlighted in Fig. 1, we instantiate two sharers, for each half of the solvers and one reducer are producers. It gets clauses from this producer and exports some of them to all others (the consumers).

The exchange strategy is defined as follows: each solver exports clauses having an LBD value under a given threshold (2 at the beginning). Every 1.5 seconds, 1500 literals (the sum of the size of the shared clauses) are selected from each producer by the sharers and dispatched to consumers. The LBD threshold of the concerned solver is increased (resp. decreased) if an insufficient (resp. a too big) number of literals are dispatched (75% and 98%).

E. Online Strengthening

There is one reducer engine that is both consumer and producer in each of the two sharing groups. It receives clauses from half of the sequential solvers, strengthened them, in case of success it then exports them back. The sharing mechanism will then share this strengthened clauses to all the other solvers. Since a strengthened clause subsumes the original one, it is likely that cores will forget the original clause over time.

III. `P-MCOMSPS-COM`

`P-MCOMSPS-COM` has exactly the same behaviour than `P-MCOMSPS` except for its clause sharing policy where clauses are filtered through community and LBD as presented in [8].

A. Community Structure

It is well admitted that real-life SAT formulas exhibit notable "structures". One way to highlight such structures is to represent the formula as a graph and analyze its community structure [12]. In `P-MCOMSPS-COM`, community structure is computed using the Louvain method [13] on the variable incident graph (VIG) of the simplified formula. The result is a disjoint partition of the variables present in the formula. Since this process can take some time, only one of the solver is responsible for the computation; until it ends sharing is based only on LBD as in `P-MCOMSPS`.

B. Community and LBD Sharing

We call *"COM of a clause"* the number of communities in which a clause spans. To compute the COM of a clause, we consider the variables corresponding to the literals of the clause and we count the number of distinct communities represented by these variables. When information on community structure is available, sharing policy switches and clauses are filtered using COM and LBD: shared clauses are those with LBD ≤ 3 or (LBD ≤ 4 and COM ≤ 3); this threshold has been highlighted in [8].

IV. `P-MCOMSPS-MPI` AND `P-MCOMSPS-COM-MPI`

This section presents `P-MCOMSPS-MPI` (reps. `P-MCOMSPS-COM-MPI`) which is a distributed adaptation of `P-MCOMSPS` (resp. `P-MCOMSPS-COM`). In order to adapt our concurrent solvers for the cloud track we added distant clause sharing, and termination. Moreover, since in the cloud track nodes have less CPUs, underlying concurrent solvers only use one reducer and one sharing group.

A. Distant Clause Sharing

On each node a concurrent solver (described in previous sections) runs. We added a component called `VirtualSolverAsynchronous` which supports the MPI-based communications between nodes. This solver receives clauses from other workers on the same node and send them to the other nodes. It also receives distant clauses (from other `VirtualSolverAsynchronous`) which are spread over the local node using the classical sharing mechanism.

B. Termination

Termination is handled by regularly synchronising main threads of each concurrent solver and is implemented thanks to the `MPI_Allgather` function.

References

[1] L. Le Frioux, S. Baarir, J. Sopena, and F. Kordon, "Painless: a framework for parallel sat solving," in *Proceedings of the 20th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pp. 233–250, Springer, 2017.

[2] Y. Hamadi, S. Jabbour, and L. Sais, "Manysat: a parallel sat solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 6, pp. 245–262, 2009.

[3] T. Balyo, P. Sanders, and C. Sinz, "Hordesat: A massively parallel portfolio sat solver," in *Proceedings of the 18th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pp. 156–172, Springer, 2015.

[4] Y. Hamadi, S. Jabbour, and J. Sais, "Control-based clause sharing in parallel sat solving," in *Autonomous Search*, pp. 245–267, Springer, 2011.

[5] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers.," in *IJCAI*, vol. 9, pp. 399–404, 2009.

[6] J. H. Liang, C. Oh, V. Ganesh, K. Czarnecki, and P. Poupart, "Maplecomsps lrb vsids, and maplecomsps chb vsids," pp. 20–21, 2017.

[7] S. Wieringa and K. Heljanko, "Concurrent clause strengthening," in *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pp. 116–132, Springer, 2013.

[8] V. Vallade, L. Le Frioux, S. Baarir, J. Sopena, V. Ganesh, and F. Kordon, "Community and LBD-based clause sharing policy for parallel SAT solving," in *Proceedings of the 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT'20)*, pp. 11–27, Springer, 2020.

[9] V. Vallade, L. Le Frioux, S. Baarir, J. Sopena, and F. Kordon, "On the usefulness of clause strengthening in parallel sat solving," in *Proceedings of the 12th NASA Formal Methods Symposium (NFM)*, Springer, 2020.

[10] J. H. Liang, V. Ganesh, P. Poupart, and K. Czarnecki, "Learning rate based branching heuristic for sat solvers," in *Proceedings of the 19th International Conference on Theory and Applications of Satisfiability Testing (SAT)*, pp. 123–140, Springer, 2016.

[11] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient sat solver," in *Proceedings of the 38th Design Automation Conference (DAC)*, pp. 530–535, ACM, 2001.

[12] C. Ansótegui, J. Giráldez-Cru, and J. Levy, "The community structure of sat formulas," in *int. conf. on Theory and Applications of Satisfiability Testing*, pp. 410–423, Springer, 2012.

[13] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.