Mallob in the SAT Competition 2021

Dominik Schreiber
Institute of Theoretical Informatics
Karlsruhe Institute of Technology
Karlsruhe, Germany
dominik.schreiber@kit.edu

***Abstract*—We describe our contribution to the parallel and cloud tracks of the SAT Competition 2021. Notable differences over last year's submission include additional diversification, a simple kind of memory awareness, lock-free clause import in Lingeling, and updated parametrization of clause sharing.**

***Index Terms*—Parallel SAT solving, distributed SAT solving**

I. Introduction

After the great success of the distributed SAT solving system named *mallob-mono* in the SAT Competition 2020 [1], we submit a new version of this system with a number of improvements to this year's SAT Competition. We submit our system not only to the cloud track but to the parallel track as well in order to see how it compares to state-of-the-art shared memory solvers at a smaller scale.

We decided to name our submission *Mallob* and omit the suffix "-mono" which was meant to emphasize the mode of operation where only one instance at a time is solved. Mallob as a whole is capable of solving many instances at once and performing decentralized malleable load balancing on top of these jobs [2]. Furthermore, since last year, Mallob gained promising new solver interfaces to CaDiCaL [3], Glucose [4], and (work in progress) MergeSAT [5]. However, due to the rules of the competition, we cannot make use of more than one CDCL solver and can only solve one instance at a time. In productive environments free of such constraints, our system can reach better performance by employing a careful mix of these solvers and by solving several instances in parallel.

II. System and Solver Setup

The setup of our system remains mostly unchanged compared to last year's submission. We subdivide each physical compute node into groups of four hardware threads each and run one MPI process on each such group. Each MPI process then runs four core solvers in parallel. We make use of Lingeling and YalSAT [3] as last year with the same kind of "native" diversification options.

We revisited the concurrent program code in Lingeling's solver interface and made the clause import lock-free by introducing a concurrent lock-free ring buffer[1] instead of a simple array guarded by a mutex. While the array used to grow indefinitely if a solver did not import any clauses for a long time, the size of the ring buffer is now limited to a small multiple of the payload that may be shared each round, and if the buffer is full, further incoming clauses are discarded.

[1]https://github.com/rmind/ringbuf

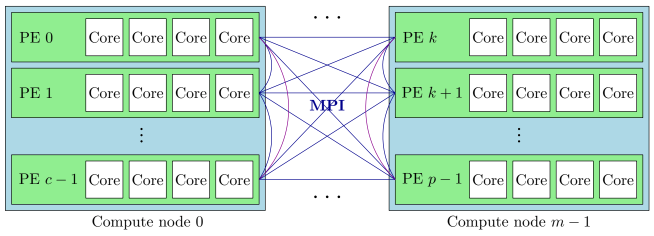

Fig. 1. Architecture of Mallob in the cloud track [2]

As an experimental change, we introduce a new kind of diversification based on permuting the input: With probability p, a solver will randomly shuffle all clauses and the literals within each clause. Preliminary experiments at a small scale showed that performing this permutation in *each* solver is detrimental to the overall performance, but we believe that letting a small ratio of solvers operate on a permutation of the input may provide different insights to the problem and improve performance in large scale environments where all native diversification options are exhausted. We set $p = 0.03$ which, in the cloud track with 1600 solver threads, leads to an expected number of 48 "permuted solvers" and still leaves the great majority of solvers running on the original problem.

III. Memory Awareness

Last year's benchmarks featured a few very large instances for which we experienced out-of-memory errors on the compute nodes running Mallob. We introduce a simple kind of memory awareness to alleviate this problem: As described in [2], we limit the total number of literals (including clause separation zeroes) which may be imported to the solver threads of a particular MPI process. This measure provides a coarse estimate on the memory the solver threads will use. If the input size were to exceed this limit, the number of solver threads to spawn is reduced such that either the import size is below the limit or only a single thread remains. We allow a total import size of $50 \cdot 10^6$ for each thread which we expect to prevent most out-of-memory issues given that in both tracks 4 GB of main memory are available per hardware thread.

IV. Clause Exchange

We repaired a subtle issue within the merging step of our distributed clause aggregation scheme: In the set data structure used for bookkeeping inserted clauses and for detecting duplicate clauses, we did use a hash function which is insensitive to the ordering of literals [6] but checked the equality of two clauses in an order-sensitive manner. We made the equality check order-insensitive as well such that our filtering during aggregation should now reliably detect duplicates even if the literals are ordered differently.

Based on large scale experiments [2] we adjusted the clause sharing parameters of our system compared to last year's submission. We significantly increased the clause length limit from five to 30: As our clause aggregation scheme only shares the globally shortest clauses, we found that it is not particularly harmful (but can rather be beneficial) to let individual solvers export some longer clauses. Likewise, experiments indicated that at the scale of the cloud track, it is beneficial to slightly increase the overall volume of clauses which can be shared compared to last year's configuration. As a result, we increased the clause buffer discount factor α from 0.75 to 0.9. In the parallel track, we set $\alpha = 1$ because we believe that employing only 64 solvers on a single machine allows for even more clauses to be shared with less of a penalty. This is the only difference between our parallel track submission and our cloud track submission.

Acknowledgment

The author expresses his heartfelt thanks to Armin Biere for providing the core solvers of Mallob, and to Laurent Simon and Armin Biere for allowing the use of Glucose and CaDiCaL in the competition even though it was ultimately not possible to make use of them.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500).

European Research Council
Established by the European Commission

Evaluation of this work was partially performed on the supercomputer ForHLR funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the Federal Ministry of Education and Research. The author gratefully acknowledges the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC-NG at Leibniz Supercomputing Centre (www.lrz.de).

References

[1] D. Schreiber, "Engineering HordeSat towards malleability: mallob-mono in the SAT 2020 cloud track," in *Proc. of SAT Competition*, pp. 45–46, 2020.

[2] D. Schreiber and P. Sanders, "Scalable SAT solving in the cloud," in *International Conference on Theory and Applications of Satisfiability Testing*, 2021. In review.

[3] A. Biere, "CaDiCaL, Lingeling, Plingeling, Treengeling and YalSAT entering the SAT competition 2018," *Proc. of SAT Competition*, pp. 13–14, 2018.

[4] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Twenty-first International Joint Conference on Artificial Intelligence*, 2009.

[5] N. Manthey, "MergeSAT," in *Proc. of SAT Competition*, pp. 40–41, 2020.

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