MallobSat and MallobSat-ImpCheck In the SAT Competition 2024

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

***Abstract*—We present the 2024 version of the distributed SAT solver MALLOBSAT with improved clause sharing techniques and data structures, updated solvers, and a new approach to trustworthy solving without explicit proof production.**

***Index Terms*—SAT solving, HPC, proofs**

I. INTRODUCTION

We submit the current upstream of the distributed job scheduling and SAT solving platform MALLOB[1] [6, 10, 11]. In line with the author's dissertation [8], we now use the following naming scheme: Our job scheduling framework is named MALLOB whereas its integrated SAT solving engine is named MALLOBSAT. While there is currently no way of running MALLOBSAT outside of MALLOB, the platform MALLOB can support applications beyond SAT solving.

For details on the design decisions and inner workings of MALLOBSAT, we refer to an article under review [10] and (slightly older) to the author's dissertation [8]. In these works we also describe some adjustments over last year's competition version, the most relevant of which are the following:

- We use a new sublinear function to bound the sharing buffer volume w.r.t. the number of involved processes, which first grows linearly and then converges to an upper bound of $L = 250\,000$ literals (per sharing).
- We now *increment* each incoming clause's LBD score rather than resetting it to the clause's length.
- We implemented a mechanism which compares the actual volume of successfully shared clauses (after filtering) to the *anticipated* sharing volume and tracks the difference to elastically adjust the next exchange's sharing limit.
- We fixed bugs in the exact distributed clause filtering, especially concerning the filtering of self-produced clauses.
- We updated some configuration options such as the sharing interval (now at 0.5 s).

We submit a performance-focused version of MALLOBSAT to the cloud track and two versions to the parallel track: a downscaled version of our cloud submission and a fully trustworthy version "MallobSat-ImpCheck" (Section III).

II. GENERAL CHANGES

In the following, we focus on changes made since the works mentioned above [8, 10] to all the versions we submit.

A. Solvers

We updated KISSAT and CADICAL to their 2023 version [3], with the consequence that KISSAT now features fewer overall configuration options in its portfolio. Using the latest version of CADICAL with full LRAT support [5] enables our trusted solving approach (see Section III). Motivated by last year's results, where our 64-thread configuration significantly outperformed the 32-thread configuration, we run 64 solver threads in the parallel track (and, as always, 16 in the cloud track). We no longer use diversification via input permutation, which had no apparent benefit in large-scale tests.

B. Clause Buffers

Once again, we reworked the clause buffering data structure after we found the prior version to perform disappointingly [10]. Rather than solver threads explicitly stealing space from "worse" slots in order to insert a better clause [7], we now allow unlimited insertions of clauses up to a certain slot, the *quality limit*.[2] Failed insertion attempts (i.e., beyond the quality limit) are still being counted for each slot in order to track the distribution over produced clauses. Each flush operation counts the number of actual and failed clause literals per slot until some total literal budget is exceeded. The quality limit is then updated to the current slot. This particular slot *at* the quality limit is the only one with an explicit maximum capacity, namely the remaining literal budget after considering all prior slots. Each slot's memory is a plain array which is grown as needed during insertions and resized by the flush operation when below a certain fill ratio. While this can lead to some slight memory peaks in between clause exchanges, it allows for very inexpensive insert operations.

C. Infinite Units

For some inputs, we noticed that all or most solver threads produce tens to hundreds of thousands of unit clauses, in particular at the beginning of solving. MALLOBSAT so far [7, 8] behaved as follows: Each unit clause is admitted *once* to each process' clause buffer—while duplicates are filtered, there is no limit to the number of units in a clause buffer. When the process is prompted to export its clauses, it exports only these units (since they are of highest priority) or even just

[1] https://github.com/domschrei/mallob

[2] As before, unit clauses are always admitted.

a subset if there are too many. These units are then aggregated to a single sharing buffer holding the set of *unique* exported units, which stays far behind the anticipated sharing volume due to the many duplicates. Until all units have been flushed, clause sharing is completely jammed with those units.

To address this issue, we now allow for *unlimited* export and sharing of unit clauses. On a technical level, we introduced a threshold x such that clauses of length $l \leq x$ no longer count towards a sharing buffer's budget that determines whether further clauses can be added. For $x = 1$, since the number of units that follow from a formula is trivially bounded by its number V of variables, we never communicate more than during the formula's initial transfer. As such, clause sharing can continue normally while the solvers synchronize all units they found. This is also reasonable in the sense that unit clauses do not come with any inherent cost (like added memory usage in the solvers) and thus should be exchanged without reservation *once*. While we believe that $x = 2$ could also be useful, we considered a worst-case sharing volume of $\mathcal{O}(V^2)$ to be unacceptable and use $x = 1$.

This change did necessitate some further changes: So far, we used a fixed-size block of shared memory to transfer clauses between the MALLOB process and the SAT process, which is no longer adequate. Instead we now use UNIX pipes, which can add some delays but save some memory.

III. MALLOBSAT-IMPCHECK

We introduce a new "trustworthy solving" approach powered by the IMPCHECK (**Im**mediate **M**assively **P**arallel **P**ropositional **P**roof **Check**ing) tool chain.[3] Rather than producing a single persistent proof file and then checking it, which is possible but suffers from bottlenecks [4], each solver's LRAT proof output [5] is checked *on-the-fly* by a separate checker process without writing it to disk. To transfer the guarantee of a clause's soundness across processes, each checker signs each checked clause with a cryptographic *signature* (based on 128-bit SIPHASH [1]) which is bundled with the clause during sharing. The receiving checker validates the signature before adding the clause like an original, "axiomatic" clause. Our approach builds on the assumption that checker processes are the only parties which are able to compute such signatures, which is ensured via confidential access to the key K used by SIPHASH. Our trusted processes (a parser, which emits the parsed formula together with a signature, and the checkers) are written in around 1000 lines of C99 (with the ultimate goal of formally verifying them to obtain a verified distributed solver). Communication between a solver thread and a checker process is achieved by a named (UNIX) pipe in each direction, which is indistinguishable from plain file I/O in terms of the checker's code. We describe this approach in full detail in a future publication [9], where we found the overhead of our approach to be nearly independent of the scale of solving, reaching 42% of median running time overhead over non-trusted solving at 2432 cores (32 nodes).

[3]https://github.com/domschrei/impcheck

We have added two further features to this approach after its submission [9]: First, checkers now also check and sign satisfying assignments (with the consequence that at least one checker per SAT process needs to preserve all of its original problem clauses throughout solving). Secondly, we run a few solver threads specifically tuned to find satisfiability (e.g., YALSAT [2] and SAT presets for CADICAL and KISSAT) and run them *without* emitting proof information or clauses. If such a solver reports unsatisfiability, the result is discarded.

We only submit this configuration to the parallel track because only a single submission to the cloud track is allowed (not because of any technological barrier). Since the checkers require some added CPU time, we spawn 58 solver threads and leave the remaining six hardware threads for the checkers.

ACKNOWLEDGMENT

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 882500). Some of this work was performed on the HoreKa supercomputer funded by the Ministry of Science, Research and the Arts Baden-Württemberg and by the Federal Ministry of Education and Research. The authors gratefully acknowledge 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] Jean-Philippe Aumasson and Daniel J Bernstein. "SipHash: a fast short-input PRF". In: *International Conference on Cryptology in India*. Springer. 2012, pp. 489–508.

[2] Armin Biere. "Yet another local search solver and Lingeling and friends entering the SAT Competition 2014". In: *SAT Competition*. 2. 2014, p. 65.

[3] Armin Biere, Mathias Fleury, and Florian Pollitt. "CaDiCaL_vivinst, IsaSAT, Gimsatul, Kissat, and TabularaSAT Entering the SAT Competition 2023". In: *SAT Competition*. 2023, p. 14.

[4] Dawn Michaelson et al. "Unsatisfiability proofs for distributed clause-sharing SAT solvers". In: *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer. 2023, pp. 348–366.

[5] Florian Pollitt, Mathias Fleury, and Armin Biere. "Faster LRAT checking than solving with CaDiCaL". In: *Theory and Applications of Satisfiability Testing (SAT)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023.

[6] Peter Sanders and Dominik Schreiber. "Decentralized online scheduling of malleable NP-hard jobs". In: *Proc. Euro-Par*. Springer. 2022, pp. 119–135.

[7] Dominik Schreiber. "Mallob{32,64,1600} in the SAT Competition 2023". In: *SAT Competition*. 2023, pp. 46–47.

[8] Dominik Schreiber. "Scalable SAT Solving and its Application". PhD thesis. Karlsruhe Institute of Technology, 2023.

[9] Dominik Schreiber. "Scalable Trusted SAT Solving with on-the-fly LRAT Checking". Submitted to SAT. 2024. URL: https://dominikschreiber.de/papers/2024-sat-scalable-pre.pdf.

[10] Dominik Schreiber and Peter Sanders. "MALLOBSAT: Scalable SAT Solving by Clause Sharing". Submitted to *Journal of Artificial Intelligence Research* (JAIR). 2024. URL: https://dominikschreiber.de/papers/2024-jair-mallobsat-pre.pdf.

[11] Dominik Schreiber and Peter Sanders. "Scalable SAT Solving in the Cloud". In: *Theory and Applications of Satisfiability Testing (SAT)*. Springer. 2021, pp. 518–534.