Scalable SAT Solving on Demand (Abstract)

Dominik Schreiber
Peter Sanders
dominik.schreiber@kit.edu
sanders@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Baden-Württemberg, Germany

ABSTRACT

We present our line of work on scalable distributed-memory solvers for the satisfiability of formulas in propositional logic.

KEYWORDS

HPC; SAT solving; malleable scheduling; decentralized algorithms

1 INTRODUCTION

The problem of propositional satisfiability (SAT), i.e., to decide whether a formula in propositional logic is satisfiable, is the original NP-complete problem [4] and an essential building block for a plethora of important applications such as formal verification [21], cryptanalysis [20], and electronic design automation [9] (see [17, Sect. 2.2.5] for further examples). We describe our line of work on scalable distributed-memory algorithms for SAT solving. Our two main thrusts are (a) improving distributed SAT solving approaches themselves and (b) making efficient use of the computational resources at hand through malleable job scheduling. The featured parallel algorithms range from decentralized resource negotiation protocols to compact data exchanges over a fluctuating set of processes. Our system Mallob and its integrated distributed SAT solver MallobSat have attracted international attention, in particular due to their role in the International SAT Competitions [3, 5].

Context. This overview is based on the 2021 SAT conference paper "*Scalable SAT Solving in the Cloud*" [18] and its more recent follow-ups [17, 19]. We also touch on a 2022 Euro-Par publication [11], which expands on our decentralized scheduling algorithms. Furthermore, the outlined system was published at the Journal of Open Source Software (JOSS) [12] and participated in the International SAT Competition 2020–2023 [13–16]. Lastly, our system was the first distributed solver to gain support for producing *proofs of unsatisfiability* through a cooperation with Amazon researchers [10], which is not the focus of this overview.

2 BACKGROUND

The SAT problem is to find an assignment to all Boolean variables in a propositional formula F such that F evaluates to `true`, or to report *unsatisfiability* if no such assignment exists. Today's most efficient sequential SAT solvers are based on the *Conflict-Driven Clause Learning* (CDCL) paradigm: The solver performs a careful search of the space of partial variable assignments, backtracks and restarts its search frequently and non-chronologically, and (most importantly) learns *redundant conflict clauses* when encountering a logical conflict [8]. The performance of SAT solvers crucially depends on bookkeeping these conflict clauses to prune the search space. Parallelizing this search by partitioning the search space [6, 7] is problematic because good partitionings are difficult to find in general [2]. The more successful approach is to run many sequential solvers in parallel on the original formula and to let them share some of their conflict clauses from time to time. This *clause-sharing portfolio* paradigm has been applied before to massively parallel scales, with mixed results. While Balyo et al.'s HordeSat [1] reportedly achieved super-linear speedups for individual instances, its median speedup was 13 at 2048 cores (i.e., efficiency 0.6%).

3 OVERVIEW

When faced with parallel algorithms that scale sub-linearly, a natural means to still exploit massively parallel hardware efficiently is to process many inputs at once. Indeed, an on-demand, multi-user platform for SAT solving tasks appears promising for many applications, such as formal verification, where a single task can emit many propositional formulas [21]. In line with these observations, we explore online scheduling of distributed tasks whose running time is unknown in advance. In order to make the best use of computational resources despite this lack of knowledge, we exploit *malleable scheduling*, where the computational resources allotted to a task can fluctuate during the task's execution. Our approach is fully decentralized. Each process is affiliated with at most one job at any given time (but can hold data of further jobs in which the process is currently inactive). Each job is organized as a *binary tree of processes* that can grow and shrink at any time to reflect the job's current *fair volume* $v_j \in \mathbb{N}$, i.e., the number of processes the job is *supposed* to own at that point in time. These job volumes are computed with a decentralized protocol that takes into account individual job demands and priorities [18] and can be implemented in logarithmic span [11]. Idle processes are assigned to job trees by

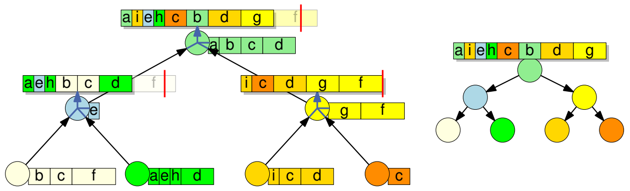

Figure 1: Compact tree-based clause exchange approach.

routing *requests* r^i_j (each demanding a process for the i-th job tree node of job j) through the system. We explored random walks of requests; routing requests along a global process tree; and matching requests and idle processes via asynchronous prefix sums [17]. The latter two strategies empirically worked the best. In evaluations on up to 6144 cores of an HPC cluster, our system MALLOB (**mall**eable **lo**ad **b**alancer) achieved scheduling latencies in the range of *few milliseconds*, both for allotting an initial process for an incoming job and for adding new processes to an existing job. The flexible and rapid reallocation of resources achieves near-optimal utilization.

In terms of the distributed SAT algorithm running within each particular job tree, we consider HORDESAT as a point of departure to design a malleable and more scalable distributed SAT solver. HORDESAT's clause exchange uses an all-gather operation that concatenates a fixed amount of data from each processor to one large array. This array often features gaps, where no information at all is present, and can contain many duplicate clauses. By contrast, Fig. 1 illustrates our clause exchange approach via an example with seven processes. Each letter corresponds to a clause, the surrounding box size representing its length and the color representing its process of origin. Each process writes its locally best clauses to a space-limited buffer. We then hierarchically merge these clause sets along a binary tree of processes. Each input and output is sorted by clause length in increasing order, which allows to immediately detect and discard duplicates. (Short clauses are considered the most useful.) Moreover, we limit each intermediate output in size by a function that grows sublinear in the number of involved processes, which guarantees that the operation remains scalable even in huge systems. As such, the root process obtains a compact buffer that holds the *globally most valuable distinct clauses*, which is then broadcast. Following this operation, the processes perform a second aggregation operation where they decide, based on tracking their own exported clauses, which of the shared clauses are new and should indeed be imported. This generalizes HORDESAT's approximate *clause filtering* approach and renders it exact.

We integrated state-of-the-art sequential solvers in our system and compared HORDESAT and our solver MALLOBSAT on up to 2560 cores, using diverse benchmarks from the International SAT Competition [5]. In our most recent experiments [19], MALLOBSAT more than doubles HORDESAT's geometric mean speedups and solves 10.7% more instances. As such, MALLOBSAT has dominated the massively parallel track of the International SAT Competition 2020–23 [19]. Moreover, we show that repurposing the resources of a finished job for remaining jobs can significantly reduce response times in a many-user setting without using any additional resources.

ACKNOWLEDGMENTS

Some of this work was performed on the HoreKa supercomputer and the supercomputer ForHLR, both 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).

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).

REFERENCES

[1] Tomáš Balyo, Peter Sanders, and Carsten Sinz. 2015. Hordesat: A massively parallel portfolio SAT solver. In *Theory and Applications of Satisfiability Testing (SAT)*. Springer, 156–172. https://doi.org/10.1007/978-3-319-24318-4_12

[2] Wolfgang Blochinger, Carsten Sinz, and Wolfgang Küchlin. 2003. Parallel propositional satisfiability checking with distributed dynamic learning. *Parallel Comput.* 29, 7 (2003), 969–994. https://doi.org/10.1016/s0167-8191(03)00068-1

[3] Byron Cook. 2021. Automated reasoning's scientific frontiers. https://www.amazon.science/blog/automated-reasonings-scientific-frontiers. Amazon Science.

[4] Stephen A. Cook. 1971. The complexity of theorem-proving procedures. In *ACM Symposium on Theory of computing*. 151–158. https://doi.org/10.1145/800157.805047

[5] Nils Froleyks, Marijn J. H. Heule, Markus Iser, Matti Järvisalo, and Martin Suda. 2021. SAT competition 2020. *Artificial Intelligence* 301 (2021), 103572. https://doi.org/10.1016/j.artint.2021.103572

[6] Maximilian Heisinger, Mathias Fleury, and Armin Biere. 2020. Distributed Cube and Conquer with Paracooba. In *Theory and Applications of Satisfiability Testing (SAT)*. Springer, 114–122. https://doi.org/10.1007/978-3-030-51825-7_9

[7] Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. 2011. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *Haifa Verification Conference*. Springer, 50–65. https://doi.org/10.1007/978-3-642-34188-5_8

[8] João Marques-Silva, Inês Lynce, and Sharad Malik. 2021. CDCL SAT Solving. In *Handbook of Satisfiability*. IOS Press, 131–153. https://doi.org/10.3233/faia200987

[9] João P. Marques-Silva and Karem A. Sakallah. 2000. Boolean satisfiability in electronic design automation. In *Annual Design Automation Conference*. 675–680. https://doi.org/10.1145/337292.337611

[10] Dawn Michaelson, Dominik Schreiber, Marijn J. H. Heule, Benjamin Kiesl-Reiter, and Michael W. Whalen. 2023. Unsatisfiability proofs for distributed clause-sharing SAT solvers. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 348–366. https://doi.org/10.1007/978-3-031-30823-9_18

[11] Peter Sanders and Dominik Schreiber. 2022. Decentralized online scheduling of malleable NP-hard jobs. In *European Conference on Parallel Processing (Euro-Par)*. Springer, 119–135. https://doi.org/10.1007/978-3-031-12597-3_8

[12] Peter Sanders and Dominik Schreiber. 2022. Mallob: Scalable SAT Solving On Demand With Decentralized Job Scheduling. *Journal of Open Source Software* 7, 76 (2022), 4591. https://doi.org/10.21105/joss.04591

[13] Dominik Schreiber. 2020. Engineering HordeSat towards malleability: mallob-mono in the SAT 2020 cloud track. In *SAT Competition*. 45–46.

[14] Dominik Schreiber. 2021. Mallob in the SAT Competition 2021. In *SAT Competition*. 38–39.

[15] Dominik Schreiber. 2022. Mallob in the SAT Competition 2022. In *SAT Competition*. 46–47.

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

[17] Dominik Schreiber. 2023. *Scalable SAT Solving and its Application*. Ph. D. Dissertation. Karlsruhe Institute of Technology. https://doi.org/10.5445/IR/1000165224.

[18] Dominik Schreiber and Peter Sanders. 2021. Scalable SAT Solving in the Cloud. In *Theory and Applications of Satisfiability Testing (SAT)*. Springer, 518–534. https://doi.org/10.1007/978-3-030-80223-3_35

[19] Dominik Schreiber and Peter Sanders. 2024. MALLOBSAT: Scalable SAT Solving by Clause Sharing. *Journal of Artificial Intelligence Research (JAIR)* (2024).

[20] Mate Soos, Karsten Nohl, and Claude Castelluccia. 2009. Extending SAT Solvers to Cryptographic Problems. In *Theory and Applications of Satisfiability Testing (SAT)*. Springer, 244–257. https://doi.org/10.1007/978-3-642-02777-2_24

[21] Yakir Vizel, Georg Weissenbacher, and Sharad Malik. 2015. Boolean Satisfiability Solvers and Their Applications in Model Checking. In *Proc. IEEE*, Vol. 103. 2021–2035. https://doi.org/10.1109/JPROC.2015.2455034