PARAFROST at the SAT Race 2021*

Muhammad Osama and Anton Wijs
Department of Mathematics and Computer Science
Eindhoven University of Technology, Eindhoven, The Netherlands
{o.m.m.muhammad, a.j.wijs}@tue.nl

I. INTRODUCTION

This paper presents a brief description of our solver PARAFROST which stands for *Parallel Formal Reasoning On SaTisfiability* in 2 different configurations. Compared to the solver submission in the last year competition [1], it is completely redesigned from scratch based on our recent work in [2]. It is a parallel SAT solver with GPU-accelerated inprocessing capable of harnessing NVIDIA CUDA-enabled GPUs in applying modern simplification techniques in parallel. The CDCL search is built from scratch with various optimisations based on CADICAL [3] heuristics. The inprocessing engine extends our previous work in [4], [5] with space-efficient data structures, parallel garbage collection and more. However, all submitted versions of the solver are single-threaded.

PARAFROST provides easy-to-use infrastructure for SAT solving and/or inprocessing with optimized data structures for both CPU and GPU architectures. The solver can run under Linux and Windows operating systems. The *PARAllel* keyword in PARAFROST intuitively means that SAT simplifications can be fully executed on variables in parallel as described in [2] using the Least Constrained Variable Elections (LCVE) scheduler [4], [5]. Moreover, via the Multiple Decision Making (MDM) procedure [6], the solver is capable of making multiple decisions that can be assigned and propagated at once. In principle, choosing variables to simplify or decide relies heavily on *freezing* variables, hence the name PARAFROST. The scheduled variables are *mutually independent* according to some logical properties.

II. DECISION MAKING

Based on our work in [2], the solver improves the decision heuristics of the solver submitted last year by adopting another decision queue called Variable Move To Front (VMTF) [7] where the score of a variable is defined as the number of conflicts in which the variable was involved. VMTF is implemented in CADICAL and our solver with a doubly-linked list. At the decision making step, we apply both VSIDS and VMTF for the selection of multiple decisions [8]. One can alternate between VSIDS and VMTF queues based on the restart *mode* [3] in a ping-pong manner. In CADICAL, different restart sequences are interleaved together to remedy the shortcomings of each individual one and alleviate the strengths of all. The idea is to start with the *geometric* [9] style with less frequent restarts (i.e. called in CADICAL the *stable* mode) then switch to a more aggressive style using dynamic restarts [10], [11] after some interval based on the number of conflicts. The interval is increased using the total number of propagations. In the KISSAT solver [3], the propagation metric was replaced by estimating the number of cache lines accessed by the watches in the unit propagation procedure. Currently, in PARAFROST, we adopt the same technique assuming a more realistic line size of 64-bit. Moreover, we observed that clauses are extensively checked for being *deleted* or not during propagation, simplifications, and garbage collection. To avoid dereferencing a clause only to check its state, a stencil array is created explicitly as part of the CNF data structure inspired by our parallel garbage collector proposed for the GPU solver [2].

Making a decision can always lead to conflicts, but making more of them increases the likelihood of a conflict occurring. To avoid repeatedly selecting sets of decisions that cause conflicts, they are constructed in such a way that it is guaranteed that no conflicts will occur. However, multiple decisions cannot be always selected, as the production of implications cannot indefinitely be avoided. The main question is therefore when MDM should be applied. Per search, MDM is called a number of decaying *rounds* (default is 3) if there are enough free variables to assign. If no rounds are left, it can be reset again to the initial value (e.g. 3), periodically based on conflicts, in an $n\ log(n)$ increasing step [8].

To further strengthen MDM, we add an implementation for local search using the WALKSAT strategy [12]. Besides running the local search frequently to improve the decision phases [3], we call it in MDM regularly per first round at the top level to improve the quality of the multiple decisions picked [8]. Our WALKSAT version is powered by a random number generator based on the *Xorshift32* technique discovered by George Marsaglia. The initial decision phases are assumed to be negative which goes back to MINISAT. Regarding clause minimization, we still keep the strengthening method with binaries from the previous submission which is crucial when MDM is turned on.

III. INPROCESSING

Recently, we applied GPUs in SAT solving to accelerate *preprocessing* [4], [5] and *inprocessing* [2]. In these operations, a given SAT formula is simplified, i.e., it is rewritten to a formula with fewer variables and/or clauses, while preserving satisfiability, using various simplification rules. In

* This work is part of the GEARS project with project number TOP2.16.044, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

preprocessing, this is only done once before the solving starts, while in inprocessing, this is done periodically during the solving. PARAFROST supports bounded variable elimination (BVE) [13], backward subsumption elimination (SUB) [14], Blocked Clause elimination (BCE) [15], and a new simplification technique called *eager redundancy elimination* (ERE) [2]. BVE eliminates variables by either applying the *resolution rule* or *substitution* (also known as gate equivalence reasoning) [14], [16]. Substitution detects patterns encoding logical gates, and substitutes the involved variables with their gate-equivalent counterparts. Previously [1], we only considered AND gates. In the current inprocessor, we add support for *Inverter*, *If-Then-Else* and *XOR* gate extractions. For all logical gates, substitution can be performed by resolving non-gate clauses (i.e., clauses not contributing to the gate itself) with gate clauses [16]. However, for the inverter gates we do substitution in-place without adding new clauses to the formula which saves time and memory.

ERE is a new elimination technique that we propose, which repeats the following until a fixpoint has been reached: for a given formula $\mathcal{S}$ and clauses $C_1 \in \mathcal{S}, C_2 \in \mathcal{S}$ with $x \in C_1$ and $\bar{x} \in C_2$ for some variable x, if there exists a clause $C \in \mathcal{S}$ for which $C \equiv C_1 \otimes_x C_2$, then let $\mathcal{S} := \mathcal{S} \setminus \{C\}$. In this work, we restrict removing C to the condition (C_1 is $learnt \vee C_2$ is $learnt$) $\implies$ C is $learnt$. If the condition holds, C is called a *redundancy* and can be removed without altering the original satisfiability.

In this submission, a sequential implementation of all simplifications described above is provided as part of PARAFROST. Moreover, the inprocessing engine uses a 20-byte data structure to store a clause (with at least one literal) different from the solver side which requires 24 bytes. By default, all simplifications are enabled except for BCE which was not effective in practice. The `ve+` option is always enabled with number of `phases` set to 5. The `phases=<n>` option applies `ve+` for a configured number of iterations, with increasingly large values of the threshold μ (maximum number of occurrences of a variable). If there are any unit clauses produced along the simplification process, their propagation is delayed and run in the next phase. Finally, at the last phase, the ERE method is executed once. Inprocessing is scheduled periodically based on the function $n \log^2(n)$ when at least 4,000 of the fixed (root) variables are removed. Forward subsumption is scheduled on all clauses with the same scaling function but applied within the learnt-clauses reduction procedure.

The solver version evaluated in [2] was missing important inprocessing techniques (thanks to Armin Biere for pointing this out) such as probing [17], autarky reasoning [3], and vivification [18]. In the latter, binary clauses are considered for the histogram but ignored in the actual vivification. This gives proper indication of which literals are more important to vivify. Regarding autarky, we remove autarkic variables in our implementation as in KISSAT but treat them as fixed roots (i.e. make the solver think they are deduced in the search) in both solution reconstruction and variable mapping. This saves memory and time spent in storing these variables and their satisfied clauses as witnesses. Autarky is applied only once after local search. With this submission we add them all to PARAFROST and are enabled by default.

IV. THREATS TO VALIDITY

Incorrect values of literals and variables due to ill logic or type casting breaks the solver fidelity. Therefore, PARAFROST always checks the invariants ($0 < x \leq m$) and ($1 < \ell \leq 2 \times m$) as preconditions, where m is the number of variables in the formula and ℓ encodes the variable x by a logical shift to the left. The least significant bit represents the sign. The generated model for satisfiable formulas can be verified against the original formula by enabling the `modelverify` option. The generation of DRAT proofs is also supported for the sequential solver.

V. SUBMISSIONS

The solver instance PARAFROST comprises all configurations described in the previous sections, in which MDM with local search, and all simplifications are enabled. The second configuration submitted is called PARAFROST-NOMDM which disables MDM using the command `mdmrounds=0`. The initial settings of the PARAFROST instance have been tuned on the DAS-5 cluster [19] and the Dutch national supercomputer CARTESIUS.

REFERENCES

[1] M. Osama and A. Wijs, "ParaFROST, ParaFROST CBT, ParaFROST HRE, ParaFROST ALL at the SAT Race 2020," *SAT Competition 2020*, pp. 42–43, 2020.

[2] M. Osama, A. Wijs, and A. Biere, "SAT Solving with GPU Accelerated Inprocessing," in *TACAS 2021, Luxembourg, 2021, Proceedings, Part I*, ser. Lecture Notes in Computer Science, vol. 12651. Springer, 2021, pp. 133–151.

[3] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, "CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020," in *SAT Competition 2020*, 2020, pp. 51–53.

[4] M. Osama and A. Wijs, "Parallel SAT Simplification on GPU Architectures," in *TACAS*, ser. LNCS, vol. 11427. Cham: Springer International Publishing, 2019, pp. 21–40.

[5] ——, "SIGmA: GPU Accelerated Simplification of SAT Formulas," in *iFM*, ser. LNCS, vol. 11918. Springer, 2019, pp. 514–522.

[6] M. Osama and A. Wijs, "Multiple Decision Making in Conflict-Driven Clause Learning," in *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, 2020, pp. 161–169.

[7] A. Biere and A. Fröhlich, "Evaluating CDCL Variable Scoring Schemes," in *SAT*, ser. LNCS, vol. 9340. Springer, 2015, pp. 405–422.

[8] M. Osama and A. Wijs, "Improving Decision Making in CDCL SAT Solvers," in *Journal of Automated Reasoning*, 2021, to be submitted.

[9] N. Eén and N. Sörensson, "An Extensible SAT-solver," in *SAT*, ser. LNCS, vol. 2919. Springer, 2004, pp. 502–518.

[10] G. Audemard and L. Simon, "Refining Restarts Strategies for SAT and UNSAT," in *Principles and Practice of Constraint Programming*, M. Milano, Ed. Springer, 2012, pp. 118–126.

[11] A. Biere and A. Fröhlich, "Evaluating CDCL Restart Schemes," in *Proceedings of Pragmatics of SAT 2015 and 2018*, ser. EPiC Series in Computing, D. L. Berre and M. Järvisalo, Eds., vol. 59. EasyChair, 2019, pp. 1–17.

[12] B. Selman and H. A. Kautz, "An Empirical Study of Greedy Local Search for Satisfiability Testing," in *Proceedings of the 11th National Conference on Artificial Intelligence. Washington, USA, 1993*. AAAI Press / The MIT Press, 1993, pp. 46–51.

[13] S. Subbarayan and D. K. Pradhan, "NiVER: Non-increasing variable elimination resolution for preprocessing SAT instances," in *SAT*, ser. LNCS, vol. 3542. Springer, 2004, pp. 276–291.
[14] N. Eén and A. Biere, "Effective Preprocessing in SAT Through Variable and Clause Elimination," in *SAT*, ser. LNCS, vol. 3569. Springer, 2005, pp. 61–75.
[15] T. Balyo, A. Fröhlich, M. J. H. Heule, and A. Biere, "Everything You Always Wanted to Know about Blocked Sets (But Were Afraid to Ask)," in *SAT*, C. Sinz and U. Egly, Eds. Cham: Springer International Publishing, 2014, pp. 317–332.
[16] M. Järvisalo, M. J. Heule, and A. Biere, "Inprocessing Rules," in *IJCAR*, ser. LNCS, vol. 7364. Springer, 2012, pp. 355–370.
[17] I. Lynce and J. Marques-Silva, "Probing-based preprocessing techniques for propositional satisfiability," in *ICTAI*. IEEE, 2003, pp. 105–110.
[18] C. Piette, Y. Hamadi, and L. Saïs, "Vivifying Propositional Clausal Formulae," in *ECAI*. NLD: IOS Press, 2008, pp. 525–529.
[19] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff, "A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term," *IEEE Computer*, vol. 49, no. 5, pp. 54–63, 2016.