Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions, volume B-2020-1 of Department of Computer Science Series of Publications B, University of Helsinki 2020.

ParaFROST, ParaFROST_CBT, ParaFROST_HRE, ParaFROST_ALL at the SAT Race 2020

Muhammad Osama and Anton Wijs
Department of Mathematics and Computer Science
Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands
{o.m.m.muhammad, a.j.wijs}@tue.nl

I. Introduction

This paper presents a brief description to our solver ParaFROST which stands for *Parallel Formal Reasoning Of SaTisfiability* in 4 different configurations. Our solver is based on state-of-the-art CDCL search [1]–[3], integrated with preprocessing as presented in our tool called SIGmA (SAT sImplification on GPU Architectures) [4], [5], and a new technique called Multiple Decision Making (MDM) [6]. Nevertheless, all submitted versions only permits a single-threaded CPU execution.

ParaFROST provides easy-to-use infrastructure for SAT solving and/or preprocessing with optimized data structures for both CPU/GPU architectures, and fine-tuned heuristic parameters. The *Parallel* keyword in ParaFROST intuitively means that SAT simplifications can be fully executed on variables in parallel as described in [4] using the Least Constrained Variable Elections (LCVE) algorithm. Moreover, via the MDM procedure [6], the solver is capable of making multiple decisions that can be assigned and propagated at once. In principle, choosing variables to preprocess or decisions relies heavily on *freezing* (that is where FROST is surfaced) mutually independent variables according to some logical properties.

II. Preprocessing

In previous work, we have shown how Bounded Variable Elimination (BVE) [7], [8] and Hybrid Subsumption Elimination (HSE) can be performed in parallel on Graphics Processing Units (GPU). The acceleration is proven to be effective in increasing the amount of reductions within a fraction of second, e.g. $66\times$ speedup compared to SatElite [8] when combined together in `ve+` mode [4]. This mode iterates over BVE and HSE in several rounds until no literals can be removed. Furthermore, we have added new implementations for Blocked Clause Elimination (BCE) and a new simplification technique, we call Hidden Redundancy Elimination (HRE) [5]. HRE 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 $C \in S$ for which $C \equiv C_1 \otimes_x C_2$ and C is not a tautology, then let $\mathcal{S} := \mathcal{S} \setminus \{C\}$. The clause C is called a *hidden redundancy* and can be removed without altering the original satisfiability. For example, consider the formula $\mathcal{S} = \{\{a, \bar{c}\}, \{c, b\}, \{\bar{d}, \bar{c}\}, \{b, a\}, \{a, d\}\}$. Resolving the first two clauses gives the resolvent $\{a, b\}$ which is equivalent to the fourth clause in $\mathcal{S}$. Also, resolving the third clause with the last clause yields $\{a, \bar{c}\}$ which is equivalent to the first clause in $\mathcal{S}$. HRE can remove either $\{a, \bar{c}\}$ or $\{a, b\}$ but not both.

In this submission, a sequential implementation of all simplifications described above is provided as part of ParaFROST. By default, in ParaFROST, all simplifications are disabled. In ParaFROST_HRE, the `ve+` is enabled with number of `phases` set to 2. The `phases=<n>` option applies `ve+` for a configured number of iterations, with increasingly large values of the threshold μ (maximum occurrences of a variable) [4], [5]. After all phases are done, the `hre` method is executed once. On the other hand, the ParaFROST_ALL submission enables all simplifications along with `bce`.

both ParaFROST_HRE and ParaFROST_ALL delay preprocessing by a user-defined number of restarts. This gives the solver enough time to solve trivial problems (solved in few seconds) before simplifications are executed. The number of restarts needed to activate preprocessing is set to 50 through the option `pre-delay=<n>`. The solver supports geometric [9], Luby [2], and dynamic restarts [10]. However, in all submissions, we only enable dynamic restarts.

III. Multiple Decision Making

We proposed a new approach [6] to make multiple decisions in such a way, they can be assigned and propagated simultaneously or sequentially without causing any implications or conflicts. Originally, we did so to introduce a possible parallelisation strategy. This strategy is yet to pay off, but surprisingly, the MDM turned out to have a positive impact on standard, sequential CDCL, for many different formulas. In all configurations, the solver periodically calls MDM with a maximum of 3 rounds per search. Otherwise, a single decision is made as the standard CDCL procedure does. The number of MDM rounds is controlled via the option `PDM=<n>`.

IV. Chronological Backtracking

We adopted the *chronological backtracking* (CBT) introduced by the authors in [11], to help CDCL solvers avoid

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).

jumping too far in certain situations. However, the procedure is computationally expensive in calculating the correct chronological level during a conflict. Therefore, we enabled this feature in a separate solver instance called ParaFROST_CBT (all simplifications are disabled). The CBT is triggered when the number of conflicts are multiple of 5000 (`cbt-conf=<n>`) and the jumping distance is 500 (`cbt-dist=<n>`). In ParaFROST_HRE, this option is disabled.

V. Automated Tuning

The GPU code tuner made by Ben van Werkhoven [12], [13] is used to optimize the parameter settings of all heuristics in ParaFROST_ALL. The tool is capable of tuning both CPU and GPU codes with support for many search optimization algorithms. In our case, we collected a sample of 48 different formulas, stemmed from different CNF families. The solving time per problem is expected to take 1000 seconds according to a solver experiment without tuning. Then, we ran a Python script to optimize the solver based on the accumulated running time of the selected benchmark suite. The tuned parameters are passed to the solver as command-line options. The basin hopping strategy is used to accelerate the tuning process.

Finally, the solver instance ParaFROST_ALL comprises all configurations described in the previous sections, in which HRE, CBT, and all simplifications are enabled.

References

[1] J. P. Marques-Silva and K. A. Sakallah, "Grasp: A search algorithm for propositional satisfiability," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 506–521, 1999.

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

[3] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern sat solvers," in *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, ser. IJCAI'09. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, p. 399–404.

[4] M. Osama and A. Wijs, "Parallel sat simplification on gpu architectures," in *Tools and Algorithms for the Construction and Analysis of Systems*, T. Vojnar and L. Zhang, Eds. Cham: Springer International Publishing, 2019, pp. 21–40.

[5] ——, "Sigma: Gpu accelerated simplification of sat formulas," in *Integrated Formal Methods*, W. Ahrendt and S. L. Tapia Tarifa, Eds. Cham: Springer International Publishing, 2019, pp. 514–522.

[6] ——, "Multiple decision making in conflict-driven clause learning," Submitted.

[7] 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.

[8] 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.

[9] T. Walsh *et al.*, "Search in a small world," in *Ijcai*, vol. 99, 1999, pp. 1172–1177.

[10] G. Audemard and L. Simon, "Refining restarts strategies for sat and unsat," in *Principles and Practice of Constraint Programming*, M. Milano, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 118–126.

[11] A. Nadel and V. Ryvchin, "Chronological backtracking," in *Theory and Applications of Satisfiability Testing – SAT 2018*, O. Beyersdorff and C. M. Wintersteiger, Eds. Cham: Springer International Publishing, 2018, pp. 111–121.

[12] B. van Werkhoven, "Kernel tuner: A search-optimizing gpu code auto-tuner," *Future Generation Computer Systems*, vol. 90, pp. 347 – 358, 2019.

[13] ——, "Kernel tuner," 2020. [Online]. Available: https://github.com/benvanwerkhoven/kernel_tuner