Z3-Parallel at the SMT Competition 2026

Ilana Shapiro[1], Sorin Lerner[2,1], Nikolaj Bjørner[3]
[1]University of California, San Diego, La Jolla, CA, USA
[2]Cornell University, Ithaca, NY, USA
[3]Microsoft Research, Redmond, WA, USA
{ilshapiro@ucsd.edu, sorin.lerner@cornell.edu, nbjorner@microsoft.com}

I. Overview

Z3-PARALLEL implements a novel dynamic partitioning framework within Z3 [1] that systematically exploits feedback from active search to guide partitioning and pruning decisions. We introduce a binary partition tree whose splitting atoms are selected on-the-fly from worker threads' VSIDS statistics. By leveraging the conflict-driven reasoning of CDCL, we enable non-chronological backjumping across the partition tree, using per-thread unsatisfiable cores to actively prune the search space. Our partition tree also supports cross-thread information sharing and on-demand termination of workers on shared subproblems. We further augment our framework with core minimization and online backbone detection for additional search-space pruning. Full details of our implementation are in our paper. Our code is available here [2]. More information about Z3 can be found on its website [3].

II. Preliminaries

The *partitioning* approach divides φ into n independent subproblems $\varphi_1, \ldots, \varphi_n$ for parallel solving such that $\bigvee_i \equiv \varphi_i$. Thus, if any one φ_i is SAT, then φ is SAT, and if all φ_i are UNSAT, then φ is UNSAT. Our approach is based on the *cube-and-conquer* strategy. Here, a set of n atoms $A = \{a_1, \ldots, a_n\}$ is selected. A *cube* is a conjunction of literals over these atoms, that is, a formula of the form $C = \ell_1 \wedge \cdots \wedge \ell_n$, where each $\ell_i \in \{a_j, \neg a_j\}$ for some $a_j \in A$. The 2^n resulting cubes over these atoms yield 2^n independent subproblems of the form $\varphi \wedge C$ to be solved in parallel.

A popular data structure for search-space partitioning is the *partition tree* [4], where the root node represents the input formula φ. Given a parent node associated with formula P, its i-th child represents a formula $P \wedge C_i$ produced by a *partitioning function* [5] such that for all i, the disjunction $\bigvee_i C_i$ holds, and for all $j \neq i$, $\neg(C_i \wedge C_j)$. Our approach is based on this data structure.

III. Architecture

On the main thread, the *batch manager* coordinates thread synchronization. It maintains a dynamically evolving *binary partition tree* for cube storage, distributes cubes to n worker threads, and facilitates information exchange between workers. Each worker w runs a sequential Z3 instance with a progressively increasing conflict budget and repeatedly requests cubes from the batch manager. If the cube is SAT, the entire problem is SAT. If w exhausts its conflict budget without solving the cube, it returns control to the batch manager, which may perform subcubing using the top atom from w's VSIDS statistics, and selects a new cube for w. If the cube is UNSAT, the batch manager prunes the partition tree using the solver's UNSAT core, and assigns w a new cube. Our pruning algorithm performs non-chronological propagation of cores across the partition tree, integrating independently derived cores from different workers and lifting CDCL-style clause resolution to search-space partitioning.

Cubes are assigned to nodes based on the node selection policy in [6]. Although cubes are often easier, some deep cubes are harder than the original problem. SMT runtimes are also highly erratic based on solver configuration, so it is important to revisit already-attempted nodes. Therefore, we first try solving fresh, deep nodes, and if these continue to time out, we gradually revisit shallower/internal nodes. The decision to split a node into subcubes is based on the tree expansion policy in [6], which adopts a conservative approach to splitting that ensures the tree remains largely balanced.

To mitigate redundant computation when multiple workers are assigned to the same cube, we employ a *terminate-on-demand* policy that aborts workers operating on the now-stale cube. Throughout execution, w accumulates learned lemmas that are reused across cubes.

Outside the partition tree, the batch manager maintains a dynamic ranking backbone candidate literals based on phase age, and attempts to prove them as backbones via two parallel *backbone detection threads* threads operating in complementary modes. In *negative mode*, the thread negates candidates and attempts to quickly prove them as backbones by deriving a short UNSAT proof. In *positive mode*, the thread assumes candidates directly and attempts to extend the assignment to a full model. The backbone threads use a modified version of the chunked backbone algorithm [7]. The negation of a proven backbone literal prunes the partition tree.

The batch manager also maintains a dedicated *core minimization thread*, which asynchronously attempts to reduce workers' UNSAT cores using the deletion-based core extraction paradigm [8]. Smaller cores enable more powerful partition tree pruning.

IV. Acknowledgements

Z3 is a project of Microsoft Research. It was created by Nikolaj Bjørner and Leonardo de Moura. A full list of contributors to Z3 is available here.

REFERENCES

[1] L. M. de Moura and N. S. Bjørner, "Z3: an efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, ser. Lecture Notes in Computer Science, C. R. Ramakrishnan and J. Rehof, Eds. Springer, 2008, pp. 337–340. [Online]. Available: https://doi.org/10.1007/978-3-540-78800-3_24

[2] I. Shapiro, S. Lerner, and N. Bjørner, "Parallel SMT solving via dynamic partitioning, core-guided pruning, and backbone detection," https://ilanashapiro.github.io/files/parallel_z3.pdf, 2026, under review.

[3] "Z3 website," https://z3prover.github.io/, 2026, accessed: 2026-05-25.

[4] A. E. J. Hyvärinen, M. Marescotti, and N. Sharygina, "Search-space partitioning for parallelizing SMT solvers," in *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, ser. Lecture Notes in Computer Science, M. Heule and S. A. Weaver, Eds. Springer, 2015, pp. 369–386. [Online]. Available: https://doi.org/10.1007/978-3-319-24318-4_27

[5] A. E. J. Hyvärinen, T. A. Junttila, and I. Niemelä, "Partitioning search spaces of a randomized search," *Fundam. Informaticae*, vol. 107, no. 2-3, pp. 289–311, 2011. [Online]. Available: https://doi.org/10.3233/FI-2011-404

[6] T. Kolárik, A. E. J. Hyvärinen, S. Asadzadeh, and N. Sharygina, "Parallel smt solving via iterative tree partitioning," in *TACAS 2026*, 2026.

[7] M. Janota, I. Lynce, and J. Marques-Silva, "Algorithms for computing backbones of propositional formulae," *AI Commun.*, vol. 28, no. 2, pp. 161–177, 2015. [Online]. Available: https://doi.org/10.3233/AIC-140640

[8] O. Guthmann, O. Strichman, and A. Trostanetski, "Minimal unsatisfiable core extraction for SMT," in *2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016*, R. Piskac and M. Talupur, Eds. IEEE, 2016, pp. 57–64. [Online]. Available: https://doi.org/10.1109/FMCAD.2016.7886661