A Concurrent Portfolio Approach to SMT Solving

Christoph M. Wintersteiger[1], Youssef Hamadi[2], and Leonardo de Moura[3]

[1] Computer Systems Institute, ETH Zurich, Switzerland
christoph.wintersteiger@inf.ethz.ch
[2] Microsoft Research Cambridge, 7 JJ Thomson Avenue, Cambridge CB3 0FB, UK
youssefh@microsoft.com
[3] Microsoft Research, One Microsoft Way, Redmond, WA, 98074, USA
leonardo@microsoft.com

Abstract. With the availability of multi-core processors and large-scale computing clusters, the study of parallel algorithms has been revived throughout the industry. We present a portfolio approach to deciding the satisfiability of SMT formulas, based on the recent success of related algorithms for the SAT problem. Our parallel version of Z3 outperforms the sequential solver, with speedups of well over an order of magnitude on many benchmarks.

1 Introduction

Z3 is a Satisfiability Modulo Theories (SMT) solver from Microsoft Research [4]. It is targeted at solving problems that arise in software verification and analysis applications. Consequently, it integrates support for a variety of theories. Z3 participated in SMT-COMP'08, where it won 9 first places (out of 15), and 6 second places. Z3 uses novel algorithms for quantifier instantiation [2] and theory combination [3]. The first external release of Z3 was in September 2007, and the latest version was released in the beginning of 2009. Z3 integrates a modern DPLL-based SAT solver, a core theory solver that handles equalities and uninterpreted functions, satellite solvers (for arithmetic, arrays, etc.), and an E-matching abstract machine (for quantifiers). Z3 is implemented in C++; Figure 1 (left box) gives an overview of the architecture of the solver.

The ManySAT parallel SAT solver [6] won the parallel track of the 2008 SAT-Race.[4] It includes all the classical features of modern DPLL-based solvers like two-watched literals, unit propagation, activity-based decision heuristics, lemma deletion strategies, and clause learning. In addition to the classical first-UIP scheme, it incorporates a new technique which extends the classical implication graph used during conflict-analysis to exploit the satisfied clauses of a formula [1]. Unlike other parallel SAT solvers, ManySAT does not implement a divide-and-conquer strategy based on some dynamic partitioning of the search

[4] `http://www-sr.informatik.uni-tuebingen.de/sat-race-2008/`

space. On the contrary, it uses a portfolio philosophy which lets several sequential DPLLs compete and cooperate to be the first to solve the common instance. These DPLLs are differentiated in many ways. They use complementary restart strategies, VSIDS and polarity heuristics, and learning schemes. Additionally, all the DPLLs exchange learnt clauses up to some static size limits, which allows for super-linear speedups that would be difficult to reach in a portfolio without sharing.

The portfolio approach is very attractive for SMT because it allows the use of different encodings of a problem, as well as different theory solvers at the same time. Moreover, SMT solvers frequently have to handle problems that are in undecidable fragments (e.g., first-order logic and arithmetic, or non-linear arithmetic). In these fragments, the solvers are much more fragile, and different heuristics may dramatically affect performance.

On the following pages we present an integration of those two award-winning techniques. To the best of our knowledge, this constitutes the first implementation of a parallel SMT solver available to the public.

2 Portfolios

Modern SMT solvers are based on a decomposition of the input problem into a propositional problem, and theory-specific problems over the formula atoms. This allows the use of modern SAT solving techniques alongside theory-specific solvers. Each of them employs a great number of heuristics to solve industrial size problems efficiently.

In our parallel version of Z3, we parallelize the sequential solver by running multiple solvers, each configured to use different heuristics, i.e., by running a *portfolio* of solvers. If two heuristics are known to work well on disjoint benchmark sets, then we can expect speedups from this technique.

To further improve the performance of our solver, we *share* derived lemmas between solvers. New lemmas are derived as part of conflict analysis in the SAT and theory solvers. When running multiple solvers in parallel, they are likely to investigate different parts of the search space at a given point in time. Sharing lemmas may prevent a solver from entering a search space that was previously investigated by another solver, thus improving the performance of the first solver.

Z3 has a portfolio of theory solvers. For example, Z3 supports multiple arithmetic solvers: a trivial solver that treats all arithmetic functions as uninterpreted, two difference-logic solvers (based on the Bellman-Ford and Floyd-Warshall algorithms), as well as a simplex based solver. Since Z3 cannot tell which of the arithmetic solvers would be the quickest to decide the formula beforehand, we can run them in parallel and abort the computation as soon as one of them is able to deliver a definite result. Note that using the first three solvers is equivalent to constructing conservative over-approximations of the formula. If either of them decides the formula to be unsatisfiable, then so must the simplex-based solver. Sharing information in a portfolio of over-approximating solvers is sound, as solvers decide the formula only for unsatisfiable input formulas. It is also a way

of implicit approximation refinement, as approximations become more precise by importing lemmas derived from the precise formula. When sharing information in a portfolio of under-approximations, or mixed under- and over-approximations, sharing is only sound in one direction, i.e., from precise to less precise formulas.

3 Implementation

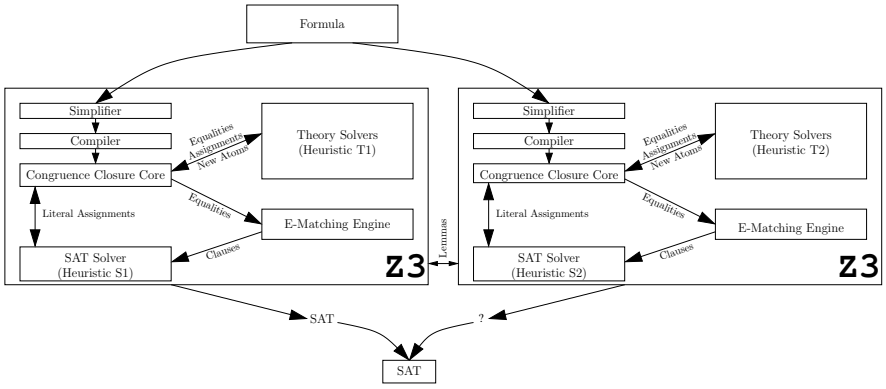


Fig. 1: Schematic overview of the Z3 parallelization.

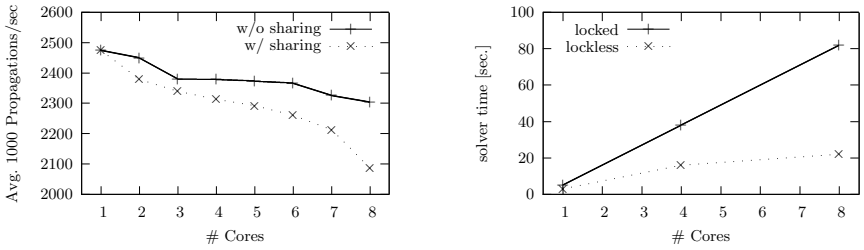
In this section we discuss the main components of our implementation. The parallel version of Z3 is based on the newest version of the sequential Z3 SMT solver. The implementation of the multi-core (shared-memory) functionality was implemented with the help of the OpenMP library, while multi-node functionality was added using an MPI library. Here, we focus on multi-core operation.

We made the observation that most benchmarks in the SMT library require a relatively small amount of memory to solve, when compared to SAT benchmarks, e.g., it is rare to see benchmarks that require more than 100 MB of memory. We thus made the design decision to copy all input data to all cores, once it is read from a file or handed to the solver using the Z3 API. This eliminates the need for locks on the formula, and enables us to use unlocked reference counters. Furthermore, it allows us to modify the input formula before (or after) handing it to a core, which is important if different (precise or approximating) encodings are to be solved, or if we would like to split the search space (e.g., by adding constraints on variables). Our experiments indicate that the additional memory overhead is of consequence only on very large instances.

Lemma exchange between cores is achieved by the help of lockless queues that hold references to all lemmas that a core wants to export. Upon reaching decision level 0 in the SAT solver, every core checks his queues for new lemmas to import.

3.1 Challenges

While working on our parallelization of Z3, we met two important challenges that we propose to discuss. First, we noticed that there is a considerable overhead when simply running multiple identical solvers in parallel. This overhead is induced by cache and memory bus congestion, and has a considerable impact on the performance of each solver in our portfolio. Figure 2a shows the average number of propositional propagations that each core is able to perform in a setup of identical solvers. This number drops dramatically from almost 2.5 million propagations per second to only 2.3 million when eight cores are employed. This corresponds to a 7 % decrease in solver performance.[5]



(a) Avg. Core performance with and without sharing of lemmas up to size 2.

(b) Locked vs. lockless memory manager.

Fig. 2: Parallelization and Sharing Overhead

Another challenge that we faced was memory allocation. While the sequential version of Z3 uses a global memory manager, we found that locking the global manager in the parallel version imposes far too much overhead. We solved this problem by reserving memory in separate heaps for each thread. Figure 2b depicts the difference in runtime that we witnessed on a small, but memory intense benchmark when running identical solvers on all cores. Clearly, the reduction in runtime by using a lockless memory manager is significant.

3.2 Portfolio

The default configuration of our implementation uses the same portfolio as ManySAT on the first four cores, i.e., it only diversifies based on heuristics specific to the built-in SAT-solver. Additional cores are configured by using different combinations of settings. Other configurations may be set from the command line, a feature available to the user in order to define their own portfolios.

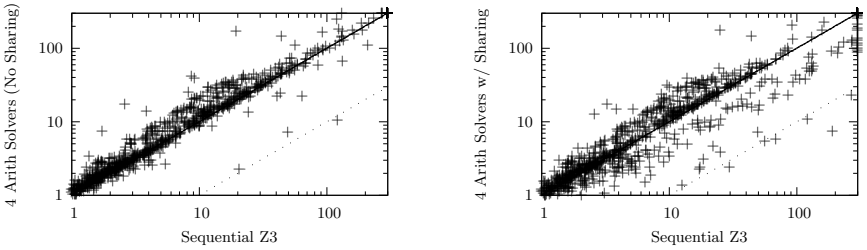
[5] These numbers were obtained from a single, statistically non-relevant file. The number of propagations per second varies greatly across different benchmarks; the relative change in performance is typical, however.

3.3 Lemma Sharing

We experimented with different lemma sharing strategies, and found that sharing of lemmas up to a size of 8 literals performed best on our benchmarks. This is not surprising, as it is also the best performing sharing strategy in ManySAT (cf. [5]).

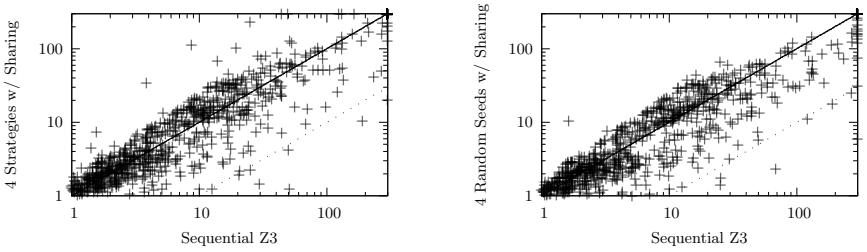
4 Experimental Evaluation

We present results on experiments on benchmarks from the SMT library[6], more specifically, the `QF_IDL` category, which consists of 1673 benchmark files. The results we present here were obtained on AMD Opteron 2.0 GHz machines with 8 cores and 16 GB of RAM. For our experiments we used four cores on each of the machines, and we compare against the sequential version of Z3 (using the configuration that was used in SMT-COMP'08); the timeout was set to 300 seconds. Our results indicate, that running four different arithmetic solvers



(a) without sharing (b) with sharing

Fig. 3: Four arithmetic solvers



(a) Four strategies (b) Four Random Seeds

Fig. 4: Four SAT solvers with sharing

[6] `http://www.smt-lib.org`

achieves speedups only on very few benchmarks, and due to the parallelization overhead many benchmarks take more time to solve than in the sequential version (Fig. 3a). This changes considerably when sharing is enabled, as demonstrated in Fig. 3a. Many benchmarks are now solved faster, with speedups between 1 and 25. Due to the large number of small benchmarks in the set, the average speedup is only 1.06. Excluding files that can be solved in less than 60 seconds by the sequential version, the average speedup is 3.2.[7]

We achieve similar results by diversifying the SAT strategies in our solver (Fig. 4). Using the four strategies used in ManySAT, we see an average speedup of 1.14, resp. 2.6 on the 60+ seconds benchmarks. As Fig. 4a shows, the speedup on many of the benchmarks is super-linear, with a maximum speedup of 40.

The best results we have seen during our experiments, were produced by a portfolio of identical solvers with sharing, each initialized by a different random seed. Those results are presented in Fig. 4b; the average speedup is 1.28 and the speedup on the 60+ seconds benchmarks is 3.5, i.e., close to linear on average. The maximum speedup we see in this configuration is 50.

5 Status & Availability

We currently investigate different portfolios and their performance on SMT-lib benchmarks. The parallel version of Z3 is also used on internal projects to improve the performance of multiple software-verification projects, and we expect more and better portfolios to arise from the use of our solver in practice.

The parallel version of Z3 is released alongside the sequential version. It is available from the Z3 project website.[8]

References

1. G. Audemard, L. Bordeaux, Y. Hamadi, S. Jabbour, and L. Sais. A Generalized Framework for Conflict Analysis. In *SAT'08*. Springer, 2008.
2. L. de Moura and N. Bjørner. Efficient E-matching for SMT Solvers. In *CADE'07*. Springer-Verlag, 2007.
3. L. de Moura and N. Bjørner. Model-based Theory Combination. In *SMT'07*, 2007.
4. L. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *TACAS'08*, 2008.
5. Y. Hamadi, S. Jabbour, and L. Sais. ManySAT: A Parallel SAT Solver. *Journal of Satisfiability*, to appear, 2008.
6. Y. Hamadi, S. Jabbour, and L. Sais. ManySAT: Solver Description. Technical Report MSR-TR-2008-83, Microsoft Research, May 2008.

[7] Below this point the parallelization overhead dominates. For further investigation the full benchmark data set may be obtained from `http://research.microsoft.com/~leonardo/z3-cav2009-results.zip`

[8] `http://research.microsoft.com/projects/z3/`