John von Neumann Institute for Computing

Towards Robustness in Parallel SAT Solving

W. Blochinger

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 301-308, 2006.

http://www.fz-juelich.de/nic-series/volume33

Towards Robustness in Parallel SAT Solving

Wolfgang Blochinger[a]

[a]Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 14, D-72076 Tübingen, Germany

This paper deals with a novel method for parallel Boolean satisfiability (SAT) solving. Our approach focuses on improving robustness, which plays a key role in enabling practical usability of parallel SAT. Specifically, we adaptively induce competition parallelism into parallel computations based on exploratory decomposition in order to control work-anomalies. We present initial performance measurements indicating the usefulness of our approach.

1. Introduction

The Boolean satisfiability (SAT) problem consists of finding a variable assignment for a Boolean formula F such that F evaluates to TRUE, resp. of proving that for F no satisfying variable assignment exists. Within the class of constraint satisfaction problems, SAT is increasingly gaining importance, since considerable research has been carried out on efficiently encoding real-world problems as SAT instances. Prominent application domains of SAT are electronic design automation (EDA) [19], software-verification [8], scheduling [10], ai planning [16], cryptography [20], and configuration of complex systems [17,21]. Today, SAT solving is recognized as a universal tool for tackling hard problems. Despite significant performance improvements of SAT solving algorithms realized in the last decade, there still exist unsolved SAT instances in all major application fields. For example, the constantly increasing complexity of chip designs is delivering extremely hard SAT instances which are far out of range of state-of-the-art sequential solvers.

Basically, parallel SAT solving is capable of substantially speeding-up the solving process. But a severe limitation of current parallel SAT solvers is that they often exhibit poor robustness. This means that for solving SAT instances with similar complexity, parallel SAT solvers achieve considerably different parallel efficiencies. Even for iterated parallel runs of the same problem instance, often totally different speedups can be observed. This behavior results from work-anomalies, where the total amount of work (in terms of the search space of variable assignments to be tested) differs significantly between sequential and parallel runs of a SAT problem instance.

Improving robustness is crucial for the practical applicability of parallel SAT solving. In this paper, we identify conditions which lead to work-anomalies and study a novel approach to parallel SAT solving which aims at improving robustness of parallel SAT solvers by adaptively combining different forms of parallelism.

The rest of the paper is organized as follows. In Section 2 we give a brief overview of state-of-the-art SAT solving algorithms. Section 3 presents our approach for improving robustness of parallel SAT. In Section 4 we report on initial performance measurements. Section 5 discusses related work.

2. SAT Solving

2.1. Basic Definitions

We consider Boolean formulae in conjunctive normal Form (CNF). In CNF, a formula is composed of conjunctions ($\wedge$) of *clauses*. A clause is the injunction ($\vee$) of one or more *literals*, and a literal is

```
boolean DPLL() {
  while(true) {
    if (decide()) {                              // decision
      while(deduce()==CONFLICT) {                // deducing
        if (current_level==0) {      // top-level conflict
          return false;                    // unsatisfiable
        } else {
          new_level=analyze_conflicts();         // learning
          back_track(new_level);            // backtracking
        }
      }
    } else {                      // all variables assigned
      return true;                             // satisfiable
    }
  }
}
```

Figure 1. DPLL Algorithm with Dynamic Learning and Conflict Driven Backtracking

a variable or the complement of a variable. Consider the following Boolean formula:

$$F = (x_1 \vee x_3) \wedge (x_2 \vee \overbrace{\overline{x_3}}^{literal}) \wedge (\underbrace{\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}}_{clause}) \wedge x_3$$

The variable assignment $x_1 \rightarrow$FALSE, $x_2 \rightarrow$TRUE, $x_3 \rightarrow$TRUE represents a satisfying assignment of F. A fundamental property of a formula in CNF is that it is satisfiable iff. in each clause at least one literal evaluates to TRUE. If for a clause all but one literals have already been assigned to FALSE, the remaining literal must be assigned to TRUE in order to satisfy the clause. Such clauses are called *unit clauses*. A situation when all literals of a clause are assigned to FALSE is called a *conflict*, and the clause is called a *conflicting clause*.

2.2. DPLL SAT Solving Algorithm

The original Davis-Putnam-Logemann-Loveland (DPLL) SAT solving algorithm [12,11] still represents the algorithmic framework of modern complete SAT solvers, but has been significantly enhanced by sophisticated heuristics for pruning the search space of variable assignments to be tested. Most beneficial advances could be achieved by employing *dynamic learning* and *conflict driven backtracking* techniques [18]. Figure 1 shows the basic structure of the DPLL algorithm incorporating these heuristics. We will restrict the following discussion of the DPLL algorithm to a top-level treatment. A more detailed explanation can be found in [23].

Basically, the DPLL algorithm performs a backtrack search process. Partial variable assignments are speculatively extended to find a satisfying assignment. The procedure `decide()` determines according to a *decision heuristics* [14] which unassigned variable should be chosen next to extend the current partial variable assignment. Each decision is recorded on an *assignment stack* along with an associated *decision level*. The decision level of the first decision is 1. The procedure `deduce()` infers additional assignments that are logical consequences of the current partial variable assignment using a technique called *unit propagation*: After making a new decision, some clauses may have become unit clauses implying new assignments. Deduced assignments are called *implications* and are also recorded on the assignment stack at the current decision level. Unit propagation terminates

when either no unit clauses exist or a conflict occurs. In the first case, a new decision is made inducing the next decision level. In the second case, the procedure `analyze_conflicts()` is invoked which constitutes the core of modern SAT solvers. It performs two tasks:

- **Dynamic Learning**: A new clause called *lemma* is constructed by analyzing the reasons for the current conflict. A lemma reflects a minimal subset of the current assignments that implies the conflict. When added to the input formula, a lemma prevents the further search process from reproducing the same conflict in other regions of the search space. Since the underlying mechanism for constructing lemmas is resolution, adding a lemma to the input formula does not affect the correctness of the DPLL algorithm. Original clauses and lemmas constitute the *clause database*.

- **Conflict Driven Backtracking**: By construction, a lemma is initially a conflicting clause. The backtracking level is determined as the lowest level at which the lemma becomes a unit clause. Note that at this level the current conflict is also resolved.

The procedure `back_track()` releases all assignments recorded on the assignment stack up to the computed backtracking level. The newly added lemma, which is now a unit clause, takes the search to a new direction. When backtracking reaches decision level 0, the current lemma forces an assignment at level 0, called a *top-level assignment*. The corresponding conflict cannot be resolved by releasing any non top-level assignments. Thus, top-level assignments are a necessary condition for the formula to be satisfiable and are fixed for the further search process. If all variables have been assigned without a conflict, the input formula is satisfiable. Unsatisfiability of the input formula is proven when a *top-level conflict* occurs (i.e. a conflict at decision level 0), since it cannot be resolved by releasing assignments.

In order to prevent the search process from getting stuck in a futile part of the search space, often a technique called *search restarts* [1] is applied. Here, the search process is periodically cancelled by backtracking to level 1 and restarted keeping some results (typically lemmas) of the previous run.

3. Improving Robustness of Parallel SAT Solving

3.1. Decomposition vs. Competition

Basically, there are two approaches to the parallelization of heuristic search problems of the kind of SAT: *decomposition* and *competition*.

The decomposition approach splits the whole search space of variable assignments into disjoint subspaces to be treated in parallel (*exploratory decomposition*). Due to the highly irregular structure of the search space, particularly of real-world SAT instances, dynamic problem decomposition and consequently dynamic load balancing become inevitable.

The competition approach is based on the property that the effect of search heuristics can vary considerably for different problem instances and cannot be predicted. Here, parallelism is exploited by concurrently executing sequential SAT solvers on the same problem instance, each pursuing a different search strategy until the problem is solved by one of the solvers.

3.2. Robustness of Exploratory Decomposition

For reasons detailed subsequently, applying exploratory decomposition for parallel SAT solving can cause considerable work-anomalies and thus limited robustness. The dynamic learning process of modern SAT solvers relies on accumulated knowledge (in the form of lemmas) which is continuously deduced during the solving process. Employing exploratory decomposition techniques on a

distributed-memory architecture results in a (partially overlapping) partition of the clause database consisting of several distributed clause databases, one on each node. All clause databases comprise the problem clauses and also locally deduced lemmas. Since dynamic learning can considerably prune the search space, it is crucial to exchange lemmas among the different clause databases, establishing a distributed learning process. Due to the high irregularity of SAT, a synchronous approach for exchanging lemmas among the nodes (for example employing an SPMD style all-to-all broadcast) would cause significant processor idling. Moreover, the total amount of deduced knowledge increases with the number of nodes, making a total exchange approach highly unscalable. Consequently, an asynchronous and selective method of communicating lemmas is more appropriate for realizing a distributed learning process. In [4], we proposed a corresponding scheme that uses mobile agents for gathering and exchanging pertinent lemmas among the distributed clause databases. But with asynchronous communication it is virtually impossible to make the deduced knowledge available on every node in a way such that the resulting (parallel) search space remains exactly identical to the sequential search space. In some cases, even minimal initial differences may rapidly lead to totally different search spaces causing a considerable potential of work-anomalies.

3.3. Adaptive Competition

In order to overcome the identified limitations of exploratory decomposition for parallel SAT solving, we propose to combine decomposition and competition parallelism in an adaptive fashion. Competition parallelism is employed when a particularly hard region of the search space is encountered (which may not be present in sequential program runs). The rationale behind this combined approach is that decomposition is able realize speedup while competition can increase robustness. However, pure decomposition can lead to poor robustness, while pure competition is not able to deliver speedup (over the optimal search strategy). Specifically, our approach to parallel SAT solving starts with exploratory decomposition and adaptively induces competition parallelism when the solving process of a particular subproblem doesn't make sufficient progress.

A subproblem is represented by a corresponding assignment stack. For carrying out dynamic problem decomposition, we apply the guiding path technique [22]. It splits the search space of a subproblem by generating two assignment stacks which define disjoint subspaces (see Figure 2). For dynamic load balancing we employ a distributed task-pool model. Splitting is initiated if the size of the local task-pool falls below a given threshold. A randomized work stealing scheme is used to transfer tasks between the task-pools.

In order to steer the transition from decomposition to competition parallelism a *transition heuristics* is employed which assesses the progress of the solving process of an individual subproblem and decides when to switch the treatment of the subproblems to competition parallelism. Parameters of the solving process that can be considered by a transition heuristics are e.g. the number of unassigned variables, the number of conflicts, or the number of splitting operations the subproblem has already been involved in. An example of a concrete transition heuristics is given in the next section.

When a transition is initiated, the respective subproblem is additionally treated using (one or more) different search heuristics. Also further decomposition of the subproblem is disabled. For current SAT-solving algorithms, examples of possible competition disciplines are decision heuristics, lemma construction methods, or clause database management strategies.

4. Experimental Evaluation

In order to test the usefulness of our approach, we carried out performance measurements on a 15 node cluster which was equipped with 2.6 GHz Intel Xeon processors and 2 GB of main memory per

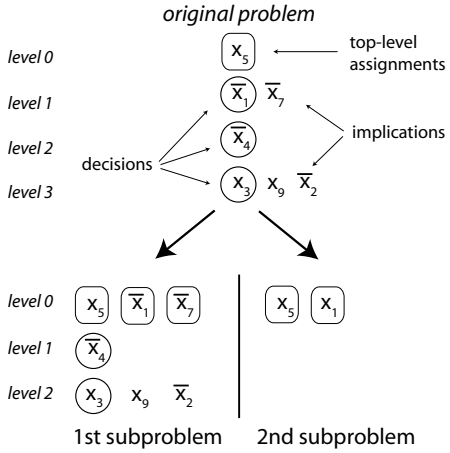

Figure 2. Search Space Splitting

node. All nodes were connected by a GigaBit Ethernet network. Our prototypical implementation is based on the sequential SAT Solver MiniSAT [13]. For the subsequently presented run-time measurements, we chose the grid-10-20 and the longmult15 benchmark from the SAT solver competition benchmark suite.

In our experiments, competition has been established by employing different decision heuristics. Modern SAT solvers typically use a small amount of randomization within the decision procedure, e.g. in MiniSAT 2% of the decisions are made randomly. In order to preserve reproducibility, a fixed random seed is used. We take advantage of this feature to generate different decision procedures by using different (fixed) random seeds, one for each node. For the grid-10-20 benchmark, the sequential run-times for the resulting 15 different decision heuristics ranged from 160.0 to 5142.9 seconds, for the longmult15 benchmark from 159.6 to 300.0 seconds. We choose randomization for establishing competition, because it ensures an unbiased evaluation of the proposed approach. For a later production use, more specific competition disciplines might deliver even more distinct results.

Our transition heuristics is based on a limit of the number of conflicts encountered until the treatment of a subproblem is switched from decomposition to competition. Every node maintains a corresponding limit value which is initialized by the number of variables of the considered SAT instance as an estimate of the problem size. Every time a transition operation takes place, the corresponding limit is increased by a factor of 2 in order to adapt to the total computation time. Additionally, a transition is only carried out, if the considered subproblem has been part of at least one splitting operation. This ensures, that the computation will never degenerate to pure competition.

Figure 3 shows the results of 100 program runs each for employing adaptive competition and pure decomposition. The given sequential run-times represent the measured times of the best performing decision procedure. The run-times for pure decomposition result from parallel runs using the best performing decision procedure.

The results for the longmult15 benchmark show for both approaches nearly identical run-times and also only minimal dispersion. Thus, for this problem instance, decomposition does not induce work-anomalies. However, adaptive competition does not increase the run-time in this situation.

For the grid-10-20 benchmark, the run-times of the decomposition approach exhibit a significant spread indicating a high degree of work-anomalies. Even run-times greater than the sequential time (slow-downs) can be observed. For the program runs which employ adaptive competition in almost all cases considerable speedups could be realized.

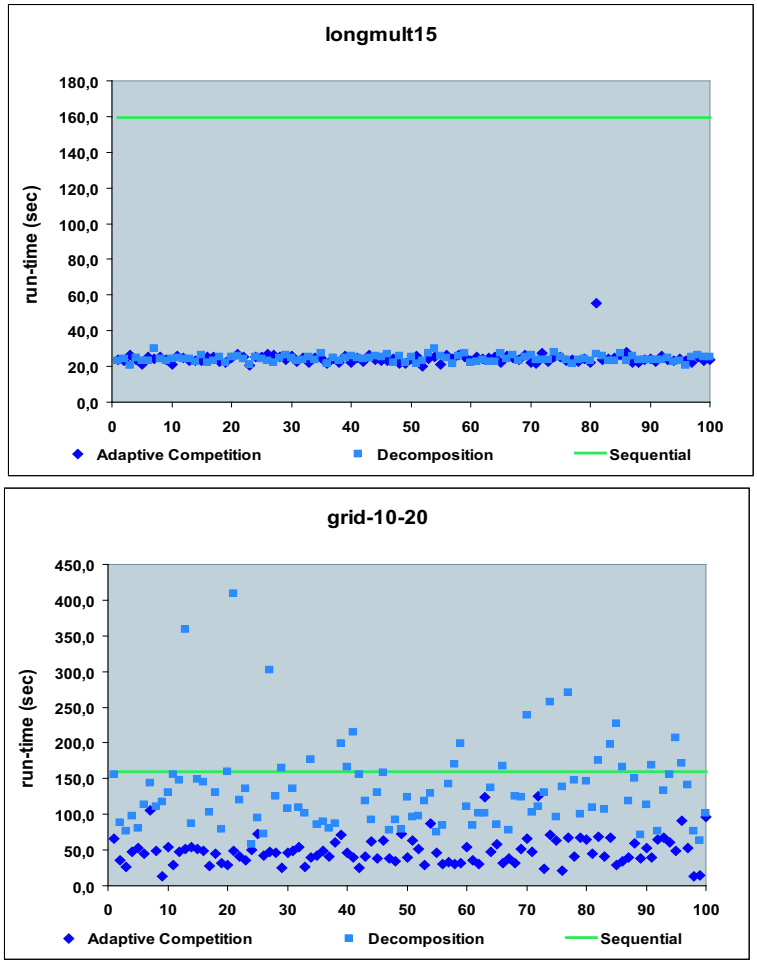

Figure 3. Results of Performance Measurements

5. Related Work

In this section, we discuss other approaches to parallel SAT solving. We restrict our treatment to parallel SAT solvers that are designed for general-purpose parallel hardware. Böhm and Speckenmeyer proposed a parallel SAT solver specifically designed for Transputer systems [7]. This work focuses on studying efficient load balancing techniques for d-dimensional mesh network-topologies in the context parallel SAT solving. Zhang's PSATO [22] is a distributed parallel SAT solver targeted for networks of workstations. PSATO introduced the *guiding path* technique for exploratory problem decomposition, taking advantage of the decision heuristics of sequential solvers for splitting the search space. PSATO is based on external parallelization of the sequential solver SATO. The parallel solver PSatz [15] by Jurkowiak *et al.* is a parallel variant of the sequential solver Satz. Concerning the parallelization approach, PSatz is very similar to PSATO, but uses work-stealing techniques for load-balancing. The parallel SAT solver PaSAT by Blochinger *et al.* [4] focuses on establishing an efficient distributed parallel learning process between the nodes of a cluster or a network of workstations. Also cross-fertilization of different kinds of heuristics is addressed [5]. PaSAT has been successfully employed in an industrial application [3] from the field of automotive

product configuration. It is based on the parallel platform DOTS [2]. GridSAT [9] developed by Chrabakh and Wolski is a parallel SAT solver targeted for parallel resources that are managed by the Globus grid platform. It employs grid technology to make reservations of appropriate dedicated resources (like compute clusters) for executing a parallel SAT solver based on the sequential solver zChaff. ZetaSAT [6] by Blochinger *et al.* is a framework for parallel SAT solving on Desktop Grids, that allows to use idle cycles of desktop computers for solving SAT instances. It is based on the ZetaGrid Desktop Grid platform and uses parallel versions of the SAT solvers zChaff and MiniSAT as solving engines.

All discussed parallel SAT Solvers exclusively apply exploratory decomposition techniques for realizing parallelism. PaSAT and GridSAT additionally establish a distributed learning process. PaSAT employs mobile agents to continuously exchange lemmas among the nodes executing the search process. GridSAT transfers a part of the clause database to newly created tasks when a search space split is performed.

6. Conclusion and Future Work

In this paper, we presented a novel approach to parallel SAT solving that adaptively employs competition parallelism in order to increase the robustness of the parallel solving process. To some extend, the presented approach can be regarded as a refinement of the search restart technique of sequential SAT solvers by using parallelism. But in contrast to search restarts, with adaptive competition work already carried out is not discarded at some point but enters into competition with other strategies.

Our future work will include investigations on sophisticated transition heuristics, other cross-fertilization issues of decomposition and competition parallelism, and on specific load balancing methods.

References

[1] L. Baptista and J. P. Marques-Silva. Using randomization and learning to solve hard real-world instances of satisfiability. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP)*, 2000.

[2] Wolfgang Blochinger, Wolfgang Küchlin, Christoph Ludwig, and Andreas Weber. An object-oriented platform for distributed high-performance Symbolic Computation. *Mathematics and Computers in Simulation*, 49:161–178, 1999.

[3] Wolfgang Blochinger, Carsten Sinz, and Wolfgang Küchlin. Parallel consistency checking of automotive product data. In G. R. Joubert, A. Murli, F. J. Peters, and M. Vanneschi, editors, *Proc. of the Intl. Conf. ParCo 2001: Parallel Computing – Advances and Current Issues*, pages 50–57, Naples, Italy, 2002. Imperial College Press.

[4] Wolfgang Blochinger, Carsten Sinz, and Wolfgang Küchlin. Parallel propositional satisfiability checking with distributed dynamic learning. *Parallel Computing*, 29(7):969–994, 2003.

[5] Wolfgang Blochinger, Carsten Sinz, and Wolfgang Küchlin. A universal parallel SAT checking kernel. In Hamid R. Arabnia and Youngsong Mun, editors, *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications PDPTA 03*, volume 4, pages 1720–1725, Las Vegas, NV, U.S.A., June 2003. CSREA Press.

[6] Wolfgang Blochinger, Wolfgang Westje, Wolfgang Küchlin, and Sebastian Wedeniwski. ZetaSAT – Boolean satisfiability solving on desktop grids. In *Proc. of the IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2005)*, Cardiff, UK, 2005.

[7] M. Boehm and E. Speckenmeyer. A fast parallel SAT-solver – efficient workload balancing. *Annals of Mathematics and Artificial Intelligence*, 17(3-4):381–400, 1996.

[8] Sagar Chaki, Edmund M. Clarke, Alex Groce, Somesh Jha, and Helmut Veith. Modular verification of software components in C. *IEEE Transactions on Software Engineering*, 30(6):388–402, June 2004.

[9] Wahid Chrabakh and Rich Wolski. GridSAT: A Chaff-based distributed SAT solver for the grid. In *Proc. of Supercomputing 03*, Phoenix, Arizona, USA, 2003.

[10] James M. Crawford and Andrew B. Baker. Experimental results on the application of satisfiability algorithms to scheduling problems. In *Proc. of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, volume 2, pages 1092–1097, Seattle, Washington, 1994. AAAI Press/MIT Press.

[11] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[12] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, 1960.

[13] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing: 6th International Conference, SAT 2003*, volume 2919 of *LNCS*, pages 502 – 518, Santa Margherita Ligure, Italy, 2003. Springer Verlag.

[14] J. N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15(3):359–383, 1995.

[15] Bernard Jurkowiak, Chu Min Li, and Gil Utard. A parallelization scheme based on work stealing for a class of sat solvers. *Journal of Automated Reasoning*, 34(1):73–101, 2005.

[16] Henry A. Kautz and Bart Selman. Planning as satisfiability. In *Proc. of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992.

[17] W. Küchlin and C. Sinz. Proving consistency assertions for automotive product data management. *Journal of Automated Reasoning*, 24(1-2):145–163, February 2000.

[18] J. P. Marques-Silva and K. A. Sakallah. Grasp - a new search algorithm for satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, 1996.

[19] J. P. Marques-Silva and K. A. Sakallah. Boolean satisfiability in electronic design automation. In *Proceedings of IEEE/ACM Design Automation Conference*, June 2000.

[20] F. Massacci and L. Marraro. Logical cryptanalysis as a SAT problem. *Journal of Automated Reasoning*, 24(1-2):165–203, February 2000.

[21] Carsten Sinz, Andreas Kaiser, and Wolfgang Küchlin. Formal methods for the validation of automotive product configuration data. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 17(1):75–97, January 2003. Special issue on configuration.

[22] H. Zhang, M. P. Bonacina, and J. Hsiang. PSATO: A distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 21:543–560, 1996.

[23] L. Zhang and S. Malik. The quest for efficient boolean satisfiability solvers. In *Proc. of 8th International Conference on Computer Aided Deduction(CADE 2002)*, Copenhagen, Denmark, 2002.