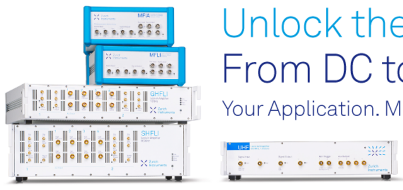
RESEARCH ARTICLE | JUNE 05 2017

A survey of SAT solver FREE

Weiwei Gong; Xu Zhou

AIP Conf. Proc. 1836, 020059 (2017)
https://doi.org/10.1063/1.4981999



A survey of SAT Solver

Weiwei Gong[1,a)] and Xu Zhou[1,b)]

[1]*School of Computer, National University of Defense Technology, China*

[a)]IssacGong@outlook.com
[b)]zhouxu@nudt.edu.cn

Abstract. In Computer Science, the Boolean Satisfiability Problem(SAT) is the problem of determining if there exists an interpretation that satisfies a given Boolean formula. SAT is one of the first problems that was proven to be NP-complete, which is also fundamental to artificial intelligence, algorithm and hardware design. This paper reviews the main algorithms of the SAT solver in recent years, including serial SAT algorithms, parallel SAT algorithms, SAT algorithms based on GPU, and SAT algorithms based on FPGA. The development of SAT is analyzed comprehensively in this paper. Finally, several possible directions for the development of the SAT problem are proposed.

INTRODUCTION

At present, software applications have penetrated every corner of developed human society. Meanwhile, software vulnerabilities have also brought security risks. According to the National Institute of Standards and Technology (NIST) on the software defect investigation[1], in the United States, the losses caused by software defects are as high as 59.5 billion US dollars, which is equivalent to 0.6% of US GDP. Software verification is an effective way to ensure software security. Automated execution of software verification usually applies the SAT Solver, which can convert software verification problems into Boolean Satisfiability Problem.

The SAT solving technology is key to promote the performance of software verification. The efficiency of software verification requires verification technology to find defects with speed and accuracy. A wide range of defect types and the low false-rate of all kinds of defects are also required to ensure an optimized software verification. Also, the locations and causes of the bugs need to be determined quickly. These requirements must be based on the execution of the algorithm. The key of the typical software verification algorithm is to solve the satisfiability problem, which needs a large amount of irregular calculation. When the size of the constraint clause is over 10 million, it must rely on a high-performance computing platform, and the problem can be solved within an acceptable amount of time.

Satisfiability Problems consist of a set of Boolean variables and a set of short clauses composed of these variables. The problem is to obtain a set of solutions satisfying the set of short clauses. SAT problem is the basic problems solving of mathematical logic, reasoning, machine learning, and many other theoretical and practical problems. The SAT problem is the first discovered NP-complete problem, and is also the core of a large class of NP-complete problems. Therefore, solving the SAT problem plays an important role in the study of artificial intelligence systems and computational theory. Any improvement of the SAT algorithm has a great effect on Computer Science.

SERIAL SAT ALGORITHMS

There are two kinds of mainstream serial SAT algorithms. They are complete algorithms (ie, backtracking search algorithms) and incomplete algorithms. The complete algorithms search for the entire solution space. If solutions exist, they will be returned. If not, the problem can be proven to be unsatisfiable. The incomplete algorithms cannot prove anything because of their randomness. These two kinds of algorithms are different, and jointly promote the development of efficient SAT algorithm.

Complete Algorithms

The complete algorithm searches for the solution space of a given SAT problem based on enumeration and backtracking search mechanisms. If the problem is satisfiable, then the solution of the logic formula will be given, and if not, a proof of completeness will be given. In fact, in the second case, it is extremely important and practical to give proof of completeness since many unsatisfiable instances require proof that they cannot be satisfied. The complete algorithm determines the value of variables in the propositional logic one by one until all variables are assigned and the value of the formula is true, or all possible assignments are tried but remain unsatisfied.

The advantage of the complete algorithm is that it can guarantee the solution of the corresponding SAT problem or prove that the formula is unsatisfiable. Unfortunately, the efficiency of the complete algorithm is extremely low. Although the average time complexity is a polynomial level, the worst-case time complexity is exponential .In fact, the complete algorithm is a process of depth-first search of the entire solution space, the search space is very large, which may make the computer unable to return the results in an acceptable time. Because of the combinatorial explosion problem, the calculation time spent cannot be tolerated.

As early as the 1960s, Davis, Putname, Longemann, Loveland, and others have studied the SAT problem, and their basic algorithm is called DPLL algorithm[2], which is one of the most commonly used algorithms. And its ideas have, so far, been closely followed. In 1995, GRASP[3] was proposed, which summarized the basic DPLL algorithm flow, and the SAT algorithm was gradually used more frequently to solve practical problems. The most important contribution of GRASP was the introduction of the Conflict Driven Clause Learning (CDCL) learning process, which is very effective in narrowing the search space with the DPLL algorithm, which makes the SAT algorithm a quantum leap. Subsequent development of the technology can be summarized in the CDCL, including the clause-based learning of the Statute, a random restart, heuristic decision-making based on the active value, and effective reasoning to support the data structure. With the introduction of these new technologies, a number of more efficient SAT algorithms such as MiniSat[4], Chaff[5], BerkMin[6], CryptoMiniSAT[7], PicoSAT[8], and Lingeling[9] were introduced.

Incomplete Algorithms

The local search algorithm is the most commonly used in the incomplete algorithm. This method uses a certain strategy to search randomly in the assignment space of the variable, and gradually approaches the final solution under the guidance of the objective function.

There are two types of objective functions: one is to minimize the number of clauses with the true value of false, and the other is to satisfy the maximum number of constraint clauses. Local search algorithms mainly include greedy local search (GSAT[10]) and random walk GSAT (WSAT[11]). The two algorithms start randomly generating an initial assignment, and then execute the loop. In the loop, the GSAT algorithm repeatedly checks the current assignment's neighbors (eg, assignments that have only one variable assignment different to the current assignment) and selects a new assignment to maximize the number of clauses that are satisfied.

The WSAT algorithm randomly selects a variable in an unsatisfied clause and inverts its value. TSAT[12] is also based on the GSAT. TSAT maintains a fixed-length tabu list during the search. In 1997, David McAllester and Bart Selman proposed NSAT[13]. This algorithm is the same as the original WalkSat in framework. Two new ways to select variables (Noverty, R_Noverty) are proposed to avoid falling into local minima. These two methods have proven to be effective.Sparrow[14] introduces the probability distribution function in the selection of variables, each variable being selected with some probability. ProbSAT[15] is also dependent on the probability distribution of the solver, but the realization is very simple, there is no complex heuristic strategy, but is very effective.

Another way to solve SAT problems is to transform the satisfiability problem into another problem in another field, which can be solved quickly, such as Genetic Algorithm(GA)[16], Simulated Annealing(SA)[17], and Survey Propagation(SP)[18] in statistical physics and so on.The SP algorithm has been very effective in the study of the 3-SAT problem. The idea of the SP algorithm method is derived from the spin glass principle in statistical physics and has important significance in solving the SAT problem. It increases processing power from 10^4 to 10^6 orders of magnitude in variable size.

PARALLEL SAT ALGORITHMS

Serial SAT algorithm has a lot of optimization methods, such as heuristic decision-making and conflict-clause learning. It is difficult to increase the execution speed of these methods by orders of magnitude. These methods are very sensitive to parameters, thus increasing the uncertainty of the performance of the algorithm[19]. To further enhance performance, the solver must be developed in parallel to accommodate modern computing techniques such as multi-core processors.

Complete Algorithms Based On Different Parallel Mode

Based on Search Space

Search space partitioning is the most classical parallel search method. The search space of the decision tree is partitioned into disjointed subspaces by using some constraints, and each subspace is computed in different threads or compute units. As shown in figure 1, space division is now commonly used to guiding path[20]. Due to the irregularity of the search tree, the divide-and-conquer strategy will have load balancing problems. PSATO[20] implements the SAT solution with a divide-and-conquer strategy, which is parallelized by partitioning the search space and load-balancing.

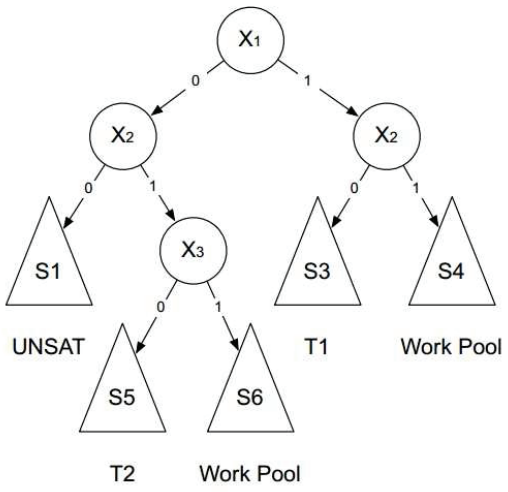


FIGURE 1. Figure 1 shows the search space partition using the guide path

PMiniSat[21] is a multi-threaded version of the serial solver MiniSAT 2.0 which uses a search path partitioning approach based on boot paths. The sharing of conflicting clauses is enhanced by the information of the boot paths. It also uses a cache-optimized data structure to speed up the propagation of the unit clauses.

SArTagnan[22] is a solver that uses search space partitioning, sharing CNF formulas in a similar way to ySAT, while each thread holds a private learning clause database. Unlike other solvers, SArTagnan does not perform the SAT solution on all threads. Instead, it uses one thread to simplify clauses, and one thread to optimize the decision process.

Competitive Parallelism

The search space partitioning method has a problem: the stability of the solution is not good enough as the solution performance depends on the heuristic mode and parameter setting. To solve this problem, a competitive solver can be used to combine multiple DPLL solvers. This method does not partition the search space of the problem. Each solver runs in the same search space sharing conflict clauses, but each solver uses different strategies, such as heuristic-strategy and restart-strategy.As shown in figure 2. Without clause-sharing, the competing parallel run time is equal to the best solver amongst solvers used. Using clause sharing, competitive parallelism can be faster than the best solver among solvers used.

In[23], MiniSAT1.14 with multiple different restart strategies is executed in the grid, each node is independent and there is no communication between the nodes. This approach can solve multiple instances of SAT at the same time, but does not achieve super-linear acceleration effect. Although it uses fewer nodes and achieves good results, it does not scale well. In the author's subsequent implementation[24], the sharing of conflict-learning clauses has been

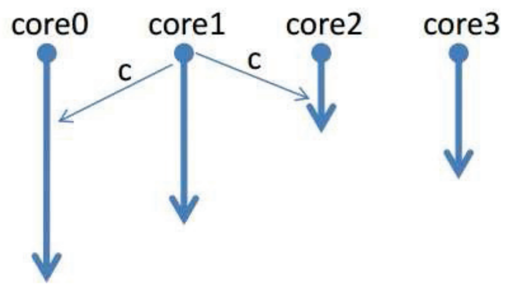


FIGURE 2. Competitive Parallelism

added, and performance has been improved. Many serial solvers cannot solve the problem. The disadvantage is that there is not a good sharing strategy.

ManySAT[19] runs four DPLL processes, each with a different restart strategy: increasing the restart interval in equal proportion, increasing the restart interval with an arithmetic sequence, increasing the restart interval with the luby sequence, and the last is the dynamic restart strategy. The clause sharing method is: the dynamic restarting kernel passes the learned clauses to two DPLL processes, the other DPLL processes do not share clauses. ManySAT also controls the total number of shared clauses. The selection of the length threshold of the shared clause changes from static to dynamic, and increases with time. Compared to the serial algorithm Minisat 2.1, ManySAT can achieve an average super-linear acceleration ratio of 6.02.

Other Parallel Methods

The parallel solver C-SAT[25] combines the parallelism of the two approaches: cooperative solution of search space partitioning, and competitive solver combination of different configuration solvers. The solver based on MiniSAT1.14 and MPI and the use of two different variable decision-making method of heuristic has good scalability, and can attain a high super-linear acceleration ratio. There is also a new approach to obtaining parallelism by partitioning the input SAT problem instances. In[26], a scattering partitioning method is proposed, which is slower than the DPLL forward partitioning method, but can solve more SAT instances at the same time.

Complete Algorithms Based On The Different Memory Structures

The realization of parallel SAT solver can be divided into distributed memory structure and shared memory structure according to memory structure. Parallel SAT was first implemented on single-core processors, and with the development of multi-core processors, parallel SAT solvers for many shared-memory architectures have emerged.

Distributed Storage Structure

In the distributed storage structure of the computer, the storage is distributed, and the nodes communicate by network. Usually, the master-slave structure is adopted, and the learning clauses are shared amongst the nodes.

GridSAT[27] was the first solver to use grid technology. The core of the solver is the serial solver zchaff, but it implements a distributed learning clause database system, and uses the master-slave approach to assign tasks. Unlike other parallel solvers, GridSAT makes its solution as serial as possible.

PMSAT[28] is a parallel solver based on MiniSAT1.14 and MPI. It has a master-slave structure and different processing processes sharing learning clauses. The author proposes several heuristic methods to partition the search space. Experiments show that different heuristic settings will have different effects. In many cases, it can achieve the super-linear acceleration ratio.

Shared Storage Structure

In a computer with a shared storage structure, the communication between nodes uses shared memory, and the access time of the nodes is uniform. The access time of the serial and parallel solvers becomes almost simultaneous, so that the shared memory structure can be further utilized to accelerate the SAT solution process.

The YSAT[29] solver shares the original CNF and learning clauses. Each thread maintains its own local learning clause database using the master-slave model, where the master is responsible for dynamic job stealing. The sharing of the original CNF will adversely affect the performance of the Cache. The paper shows that the write blocking overhead of YSAT is 10%.

MiraXT[30] is a multi-core parallel SAT solver that uses a shared storage clause database. Using the SatELite preprocessor in the solver, MiraXT can remove some bad splitting variables. In order to propagate the unit clauses of dual-word monitoring, each task of the solver uses an additional text index to store the current monitoring text. Each task can determine which clauses need to be used, there will be no delay in clause sharing in YSAT.

A Combination Of Distributed And Shared Storage

The paper[31] implements a hybrid strategy of composition and competition based on the NUMA framework. Space partitioning is implemented using MPI. Each node runs the OpenMP version of ManySAT Multi-core can be used to achieve better speed ratio. Compared to 24 threads of ManySAT, the acceleration rate reached 3.3.

Incomplete Algorithms Parallelism

There are two kinds of incomplete algorithms: competitive parallelism and multi-flip parallelism. In competitive parallelism, Pha[32] uses competitive parallelism of gNovelty+.There is no interaction between the solvers. Arbelaez[33] gives seven strategies for collaboration. Each solver shares the best value found in the variable assignment as the s-tarting point for the next run. The Prob-NormalizeW method is the best of these methods, and won the silver medal in the 2011 SAT Contest. The method uses a high probability method to select a good starting point, but retains a small probability of choosing a bad starting point. Arbelaez[34] also evaluated various local sat solvers, including sparrow, adaptiveNovelty+, paws, and vw. The same test set exhibits similar acceleration curves as it spreads to several hundred cores. The linear acceleration ratio is achieved on the crafted and verified instances; sub-linear acceleration ratios are achieved on random and quaigroup instances;. Shylo[35] gives a theoretical explanation of this phenomenon: The acceleration effect is related to the distribution of the program during the run.

PGSAT[36] is a parallel version of GSAT. In this algorithm, the variables are divided into subsets, which are assigned to different processors. In each iteration of the local search algorithm, if the global solution is not found, each subset flips the respective best variable. A more interesting finding is that when the value exceeds a certain limit, performance degrades. Moreover, this boundary is different for each instance, but it is related to the average degree of connectivity of the variable subset graph. PGWSAT[37] improves PGSAT, which randomly selects a variable to be flipped in an unsatisfiable clause with a certain probability during PGSAT execution. GenSAT implements a parallel GSAT that flip variables that improve the overall performance by tie-breaking evaluation. The authors used a Boltzmann machine to select a subset of variables and then flip it in parallel. The experimental results show that this method has fewer flip-flops than the original GSAT.

There are some algorithms that combine the complete and incomplete algorithms together. MiniWal[38] combines Minisat and Walksat to solve the MaxSAT problem. The two solvers execute in parallel, using MiniSAT to guide WalkSAT execution in different regions of the search space. Minisat saves the assignment state of the current variable and reports it to Walksat. The Walksat assignment is covered, and in the following run, only variables that are not assigned in Minisat are flipped. The hybrid algorithm presented in[39] has two phases. The first phase runs the DPLL algorithm to divide the program into multiple subspaces. The second phase is to allocate these subspaces to different processors to run the local search algorithm WalkSAT.

SAT ALGORITHMS BASED ON GPU

GPU is widely used in large-scale data processing problems with high computational efficiency. By using the GPU to accelerate the SAT solver, it can help to solve large-scale SAT problems.

Complete Algorithms

Meyer[40] proposed a 3-SAT complete algorithm based on the GPU parallel technique, and solved the SAT problem using the kernel pipeline. This method did not use some of the most commonly used SAT optimization techniques, such as BCP, but proposed a large-scale parallel approach to make better use of the GPU for SAT solutions.

Fujii[41] accelerated the BCP process of 3-SAT on the GPU, which is similar to Davis acceleration methods on the FPGA. The testbed is the 2.93 GHz Intel Core i3 and NVIDIA GeForce GTX480, and the test sets are 10 instances of the 2011 SAT competition stochastic set, each one has 50K of variables and 210K of clauses. Compared to the CPU, GPU average speedup ranges from 1.5 up to 6.7.

Incomplete Algorithms

McDonald[42] implemented a parallel WalkSAT incomplete algorithm with clause learning on the GPU. Each thread executes a serial WalkSAT algorithm, and each one has a different pseudo-random sequence.Wang[43] proposed an incomplete algorithm, which uses the random walk of the genetic algorithm to solve the SAT problem.Deleau[44] also proposed an incomplete algorithm in which the SAT instance is expressed as a 0-1 matrix, and the matrix multiplication is applied for SAT solutions.

Combination Of Incomplete And Complete Algorithms

There is also an algorithm that combines incomplete and complete algorithms. In the MESP (MiniSAT enhanced with Survey Propagation) implemented by Gulati[45], the GPU runs the incomplete algorithm called SurveySAT, and the CPU runs the full algorithm called MiniSAT. The output of the former is used for the decision-making of the latter. MESP has a 2.35x speedup compared to MiniSAT when testing 3-SAT on a 2.67GHz Intel i7 CPU and NVIDIA GeForce 280GTX GPU platform.

SAT ALGORITHMS BASED ON FPGA

According to the classification in[46], the hardware implementation of SAT solver can be divided into two kinds: a) the method for specific instance and b) the method for specific applications, as shown in Figure 3. The current research trend is gradually moving from the former to the latter.

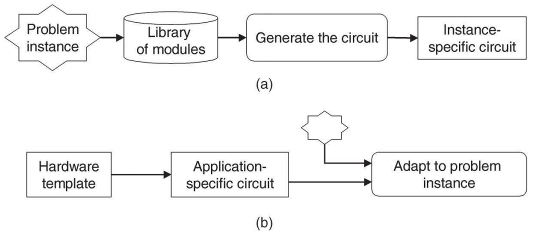

Problem instance
Library of modules
Generate the circuit
Instance-specific circuit
(a)
Hardware template
Application-specific circuit
Adapt to problem instance
(b)

FIGURE 3. (a) the method for a specific instance (b) the method for a specific application

The Method For A Specific Instance

In earlier work, reconfigurable hardware SATs focused on methods specific to a particular instance (eg, [47],[48], [49], [50]) and each circuit generated a specific SAT instance specific to it. When a new instance is to be calculated, a large amount of computation is required to re-synthesize and configure the circuit, which limits the size of the input instance.For example, Zhong et al[47] implements an accelerator in which each variable corresponds to a state machine FSM, and all FSMs join together to perform the search function.

When the left FSM completes the assignment, the control is passed to the right. When a conflict occurs, it goes back to the left. To improve usability, the author implements a C program that converts SAT instances to VHDL code. This implementation is serial, the search method is sequential, and in the computers at the time it took a few hours to compile. The authors[51] and[52] have improved the method by adding pipelining and non-sequential searching using incremental compilation and optimizing the process of generating the FPGA configuration file reducing compilation and downloading time. According to the article, the whole process only takes a few seconds.

The Method For A Particular Application

The hardware implementation for a particular application uses a more generic architecture that does not require reconfiguration of the hardware when addressing different instances. In this mode, clause is typically stored in a storage component and accessed when needed.

The most popular approach to hardware implementations of a particular application is to implement the most intensive work step BCP in hardware, while placing more complex task processing, such as conflict analysis and heuristic decisions, on the software side. For example, Dandalis[53],[54] divides a given CNF into a fixed number of modules (group of clauses), and executes the reasoning and collision detection processes in parallel and with pipelining. The merging unit then merges the values of the variables that have no collisions and passes the results back to the input of the pipeline until no new reasoning is generated, or there is a conflict. A reconfigurable template in the paper is provided to dynamically determine the optimal number of pipelines and parallel modules during execution.John D. Davis[55] designed the BCP Accelerator to group the clauses. Clauses that contain the same variables are grouped into different groups, each of which uses a tree-like clause indexing structure that can find relevant clauses through variables in a fixed clock cycle, which accelerates the propagation of constraints. Compared to the software implementation, BCP process enhances the efficiency from 5 to 16 times.

There are also designs[56], [57], [58]that implement complete SAT solver algorithms in FPGA. such as Gulati[58]. Gulati solves the global satisfiability by solving local satisfiability. The author saves multiple problem instances into off-chip DRAM. Each instance is split into smaller frames. In the calculation, an instance is loaded into the BRAM, which is then passed to the SAT solver in units of frames, and the satisfaction of the data of the frame is calculated. According to the satisfaction and Conflict selection order, it imports the next frame or rolls back to the frame causing the conflict, while updating the global variables stored in the BRAM state. Finally, the final solution of the instance is returned.

With all of these hardware solvers, only small and medium-sized SAT problems can be solved, no solvers can solve the problems faced by large-scale problems. For example, the BCP Accelerator [43], which handles a relatively large scale, can accommodate 64K variables and 64K clauses of length 9. The solver[58] can hold 10K variables and 280K fixed-length clauses. The actual scale, however, of the SAT problem in practical applications goes far beyond these limits.

A number of approaches have been proposed to improve hardware-based solutions such as using larger FPGAs[49] or multiple FPGAs [59][55], splitting problems into smaller subproblems[50], on-demand loading[58], a combination of hardware and software, or solving small subproblems with hardware[56].

DEVELOPMENT DIRECTION OF SAT SOLVER

Parallel Scalability Of SAT Solver

Parallel SAT solvers can be divided into two types: parallel SAT solvers based on Search Space and parallel SAT solvers based on competitive parallelism. The SAT problem solving process can be regarded as a proof of the concluding refutation of generating the SAT problem, and it is this structure that hinders the search space division[60]. At the same time, due to the uncertainty of the SAT algorithm, the search space division effect is not good. Competitive parallelism is an independent solution of multiple SATs, and it is difficult to guarantee the orthogonality of the search space. Current parallel SATs performance is still limited. For example, in the 2011 SAT race, parallel SATs could use 32 cores, but the best parallel solvers CryptoMiniSAT-MT[11] and Plingeling[61] had an average speedup of only 3. When the number of processors is 1,000 or 10,000, the parallel SAT scalability problem becomes more pronounced. It is necessary, therefore, to study large-scale parallel satisfiability constraint solving techniques.

Dynamic Allocation Of Computing Resources

Most of the computational problems are deterministic, and the parallel strategy of partitioning can be used to achieve good results. The solution of the SAT problem, However, has natural uncertainties, and the divide-and-conquer s-trategy relies more on luck. Good partitioning shortens subspace search time. on the contrary, another partition may also cause performance degradation. Increasing computing resources may weaken performance because the division produces more sub-space search needs. Increasing the degree of parallelism may lead to reductions of the efficiency because more parallels lead to an increase in clause sharing. There may be a cache congestion problem. With a given instance, however ,it is very difficult to know in advance how to allocate computing resources, the dynamic allocation of resources needs to studied more in future work.

The Lack Of A Unified Parallel Framework

The problem of SAT has always belonged to the research of parallel computing because the concept of SAT problem is simple and the search space is exponential. The current research does not really focus on the parallel algorithm. On the one hand, it is thought that there are still many research points in serial algorithm. On the other hand, researchers lack the appropriate parallel framework to systematically study the problem. In addition, as for heterogeneous computing platforms, in order to achieve a better overall speedup,there is needs for a unified framework for a variety of satisfiable algorithms to carry out a unified configuration and scheduling

ACKNOWLEDGMENTS

This work is partially supported by National High-tech R&D Program of China (863 Program) under Grants 2012AA01A301 and 2012AA010901, by program for New Century Excellent Talents in University, by National Science Foundation (NSF) China 61272142, 61402492, 61402486, 61472437 and by the State Key Laboratory of High-end Server & Storage Technology (2014HSSA01). All support is gratefully acknowledged.

REFERENCES

[1] G. Tassey, National Institute of Standards and Technology, RTI Project **7007** (2002).
[2] M. Davis, G. Logemann, and D. Loveland, Communications of the ACM **5**, 394–397 (1962).
[3] J. P. M. Silva and K. A. Sakallah, “Graspła new search algorithm for satisfiability,” in *Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design* (IEEE Computer Society, 1997), pp. 220–227.
[4] N. Eén and N. Sörensson, Springer Berlin/Heidelberg **10**, 978–3 (2004).
[5] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Chaff: Engineering an efficient sat solver,” in *Proceedings of the 38th annual Design Automation Conference* (ACM, 2001), pp. 530–535.
[6] E. Goldberg and Y. Novikov, Discrete Applied Mathematics **155**, 1549–1561 (2007).
[7] K. Wu, T. Wang, X. Zhao, and H. Liu, “Cryptominisat solver based algebraic side-channel attack on present,” in *International Conference on Instrumentation* (2011), pp. 561–565.
[8] A. Biere, Journal on Satisfiability, Boolean Modeling and Computation **4**, 75–97 (2008).
[9] A. Biere, Proceedings of SAT Challenge 33–34 (2012).
[10] B. Selman, H. J. Levesque, D. G. Mitchell, *et al.*, “A new method for solving hard satisfiability problems.” in *AAAI*, Vol. 92 (1992), pp. 440–446.
[11] o, łI **9**, p. 934 (1994).
[12] B. Mazure, L. Saïs, and É. Grégoire, “Tabu search for sat,” in *AAAI/IAAI* (1997), pp. 281–285.
[13] D. Mcallester, B. Selman, and H. Kautz, “Evidence for invariants in local search,” in *IN PROCEEDINGS OF AAAI-97* (2003), pp. 321–326.
[14] A. Balint and A. Fröhlich, “Improving stochastic local search for sat with a new probability distribution,” in *International Conference on Theory and Applications of Satisfiability Testing* (Springer, 2010), pp. 10–15.
[15] A. Balint and U. Schöning, “Choosing probability distributions for stochastic local search and the role of make versus break,” in *International Conference on Theory and Applications of Satisfiability Testing* (Springer, 2012), pp. 16–29.
[16] C. Solnon, IEEE transactions on evolutionary computation **6**, 347–357 (2002).
[17] W. M. Spears, Cliques, Coloring and Satisfiability: Second DIMACS Implememntation Challenge **26**, p. 33 (1993).
[18] A. Braunstein, M. Mézard, and R. Zecchina, Random Structures & Algorithms **27**, 201–226 (2005).
[19] Y. Hamadi, S. Jabbour, and L. Sais, Journal on Satisfiability, Boolean Modeling and Computation **6**, 245–262 (2008).
[20] H. Zhang, M. P. Bonacina, and J. Hsiang, Journal of Symbolic Computation **21**, 543–560 (1996).
[21] G. Chu, P. J. Stuckey, and A. Harwood, SAT race (2008).
[22] S. Kottler and M. Kaufmann, Pragmatics of SAT (2011).

[23] A. E. Hyvärinen, T. Junttila, and I. Niemelä, “Strategies for solving sat in grids by randomized search,” in *International Conference on Intelligent Computer Mathematics* (Springer, 2008), pp. 125–140.

[24] A. E. Hyvärinen, T. Junttila, and I. Niemelä, “Incorporating learning in grid-based randomized sat solving,” in *International Conference on Artificial Intelligence: Methodology, Systems, and Applications* (Springer, 2008), pp. 247–261.

[25] K. Ohmura and K. Ueda, “c-sat: A parallel sat solver for clusters,” in *International conference on theory and applications of satisfiability testing* (Springer, 2009), pp. 524–537.

[26] A. E. Hyvärinen, T. Junttila, and I. Niemelä, “Partitioning sat instances for distributed solving,” in *International Conference on Logic for Programming Artificial Intelligence and Reasoning* (Springer, 2010), pp. 372–386.

[27] W. Chrabakh and R. Wolski, “Gridsat: A chaff-based distributed sat solver for the grid,” in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing* (ACM, 2003) p. 37.

[28] L. Gil, P. Flores, and L. M. Silveira, Journal on Satisfiability, Boolean Modeling and Computation **6**, 71–98 (2008).

[29] Y. Feldman, N. Dershowitz, and Z. Hanna, Electronic Notes in Theoretical Computer Science **128**, 75–90 (2005).

[30] M. Lewis, T. Schubert, and B. Becker, “Multithreaded sat solving,” in *2007 Asia and South Pacific Design Automation Conference* (IEEE, 2007), pp. 926–931.

[31] N. Totla and A. Devarakonda, .

[32] D. N. Pham and C. Gretton, Solver description. SAT Competition (2009).

[33] A. Arbelaez and Y. Hamadi, “Improving parallel local search for sat,” in *International Conference on Learning and Intelligent Optimization* (2011), pp. 46–60.

[34] A. Arbelaez and P. Codognet, “From sequential to parallel local search for sat,” in *European Conference on Evolutionary Computation in Combinatorial Optimization* (Springer, 2013), pp. 157–168.

[35] O. V. Shylo, T. Middelkoop, and P. M. Pardalos, Parallel Computing **37**, 60–68 (2011).

[36] A. Roli, “Criticality and parallelism in structured sat instances,” in *International Conference on Principles and Practice of Constraint Programming* (Springer, 2002), pp. 714–719.

[37] A. Roli, M. Blesa, and C. Blum, (2005).

[38] L. Kroc, A. Sabharwal, C. P. Gomes, and B. Selman, “Integrating systematic and local search paradigms: A new strategy for maxsat.” in *IJCAI*, Vol. 9 (2009), pp. 544–551.

[39] W. Zhang, Z. Huang, and J. Zhang, “Parallel execution of stochastic search procedures on reduced sat instances,” in *Pacific Rim International Conference on Artificial Intelligence* (Springer, 2002), pp. 108–117.

[40] Q. Meyer, F. Schönfeld, M. Stamminger, and R. Wanka, “3-sat on cuda: Towards a massively parallel sat solver,” in *High Performance Computing and Simulation (HPCS), 2010 International Conference on* (IEEE, 2010), pp. 306–313.

[41] H. Fujii and N. Fujimoto, “Gpu acceleration of bcp procedure for sat algorithms,” in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)* (The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012) p. 1.

[42] A. McDonald, Parallel walksat with clause learning. data analysis project papers, 2009.

[43] Y. Wang, “Nvidia cuda architecture-based parallel incomplete sat solver,” Ph.D. thesis, Rochester Institute of Technology 2010.

[44] H. Deleau, C. Jaillet, and M. Krajecki, “Gpu4sat: solving the sat problem on gpu,” in *PARA 2008 9th International Workshop on State–of–the–Art in Scientific and Parallel Computing, Trondheim, Norway* (2008).

[45] K. Gulati and S. P. Khatri, “Boolean satisfiability on a graphics processor,” in *Proceedings of the 20th symposium on Great lakes symposium on VLSI* (ACM, 2010), pp. 123–126.

[46] I. Skliarova and A. de Brito Ferrari, IEEE Transactions on Computers **53**, 1449–1461 (2004).

[47] P. Zhong, M. Martonosi, P. Ashar, and S. Malik, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **18**, 861–868 (1999).

[48] T. Suyama, M. Yokoo, H. Sawada, and A. Nagoya, IEEE Transactions on Very Large Scale Integration (VLSI) Systems **9**, 109–116 (2001).

[49] M. Platzner and G. De Micheli, “Acceleration of satisfiability algorithms by reconfigurable hardware,” in *International Workshop on Field Programmable Logic and Applications* (Springer, 1998), pp. 69–78.

[50] M. Abramovici and J. T. De Sousa, Journal of Automated Reasoning **24**, 5–36 (2000).

[51] P. Zhong, P. Ashar, S. Malik, and M. Martonosi, "Using reconfigurable computing techniques to accelerate problems in the cad domain: a case study with boolean satisfiability," in *Proceedings of the 35th annual Design Automation Conference* (ACM, 1998), pp. 194–199.

[52] P. Zhong, M. Martonosi, and P. Ashar, IEE Proceedings-Computers and Digital Techniques **147**, 135–141 (2000).

[53] A. Dandalis and V. K. Prasanna, ACM Transactions on Design Automation of Electronic Systems (TODAES) **7**, 547–562 (2002).

[54] M. Redekopp and A. Dandalis, "A parallel pipelined sat solver for fpgas," in *International Workshop on Field Programmable Logic and Applications* (Springer, 2000), pp. 462–468.

[55] J. D. Davis, Z. Tan, F. Yu, and L. Zhang, "A practical reconfigurable hardware accelerator for boolean satisfiability solvers," in *Proceedings of the 45th annual Design Automation Conference* (ACM, 2008), pp. 780–785.

[56] I. Skliarova and A. d. B. Ferrari, IEEE Transactions on Very Large Scale Integration (VLSI) Systems **12**, 408–419 (2004).

[57] M. Waghmode, K. Gulati, S. P. Khatri, and W. Shi, "An efficient, scalable hardware engine for boolean satisfiability," in *2006 International Conference on Computer Design* (IEEE, 2007), pp. 326–331.

[58] K. Gulati, S. Paul, S. P. Khatri, S. Patil, and A. Jas, ACM Transactions on Design Automation of Electronic Systems (TODAES) **14**, p. 33 (2009).

[59] J. D. Davis, Z. Tan, F. Yu, and L. Zhang, "Designing an efficient hardware implication accelerator for sat solving," in *International Conference on Theory and Applications of Satisfiability Testing* (Springer, 2008), pp. 48–62.

[60] G. Katsirelos, A. Sabharwal, H. Samulowitz, L. Simon, *et al.*, "Resolution and parallelizability: Barriers to the efficient parallelization of sat solvers." in *AAAI* (Citeseer, 2013).

[61] A. Biere, SAT Challenge (2012).

[62] J. T. de Sousa, J. Da Silva, and M. Abramovici, "A configurable hardware/software approach to sat solving," in *Field-Programmable Custom Computing Machines, 2001. FCCM'01. The 9th Annual IEEE Symposium on* (IEEE, 2001), pp. 239–248.

[63] A. Strohmaier, "Multi-flip networks: parallelizing gensat," in *Annual Conference on Artificial Intelligence* (Springer, 1997), pp. 349–360.