Combining Strengths of Circuit-based and CNF-based Algorithms for a High-Performance SAT Solver

Malay K Ganai, Lintao Zhang[1], Pranav Ashar, Aarti Gupta, and Sharad Malik[1]
NEC USA CCRL, Princeton NJ
{malay,ashar,agupta}@nec-lab.com
[1]EE Dept, Princeton University
{lintaoz,malik}@ee.princeton.edu

Abstract

We propose Satisfiability Checking (SAT) techniques that lead to a consistent performance improvement of up to 3x over state-of-the-art SAT solvers like Chaff on important problem domains in VLSI CAD. We observe that in circuit oriented applications like ATPG and verification, different software engineering techniques are required for the portions of the formula corresponding to learnt clauses compared to the original formula. We demonstrate that by employing the same innovations as in advanced CNF-based SAT solvers, but in a hybrid approach where these two portions of the formula are represented differently and processed separately, it is possible to obtain the consistently highest performing SAT solver for circuit oriented problem domains. We also present controlled experiments to highlight where these gains come from. Once it is established that the hybrid approach is faster, it becomes possible to apply low overhead circuit-based heuristics that would be unavailable in the CNF domain for greater speedup.

Categories and Subject Descriptors

J.6 [**Computer Applications**]: Computer-aided Engineering – *Computer-aided design*.

General Terms

Algorithms, Performance, Experimentation,Verification.

Keywords

Boolean Satisfiability (SAT), Boolean Constraint Propagation (BCP), Conjunctive Normal Form (CNF), Bounded Model Checking (BMC).

1. Introduction

The Boolean Satisfiability (SAT) problem has extensive applications in VLSI CAD. Recent advances in SAT solvers based on Conjunctive Normal Form (CNF) representation have resulted in significantly improved performance. In particular, innovative techniques for decision variable selection [1], Boolean constraint propagation (BCP) [1, 2], and backtracking with conflict analysis based learning [1, 3] have led to high-performance CNF-based SAT solvers like Chaff [1]. For circuit application domains such as Automatic Test Pattern Generation (ATPG) [4], equivalence checking [5], and Bounded Model Checking (BMC) [6], the Boolean reasoning problem is typically derived from the circuit structure. This has also led to interest in circuit-based SAT solvers [4, 5, 7, 8], which use circuit specific knowledge to guide the search. In general, attempts to include circuit structure information into CNF-based solvers have been unsuccessful due to significant overhead. Furthermore, it is also difficult to integrate CNF-based solvers with other useful techniques like BDD sweeping and dynamic circuit transformation [5]. On the other hand, CNF-based solvers are better than circuit-based solvers in handling conflict analysis and learned clauses [1].

In this paper, we propose techniques that combine the strengths of circuit-based and CNF-based SAT solvers. In particular, we describe a *hybrid SAT solver* where the original logic formula is processed in circuit form, and the learned clauses are processed separately in CNF. We discuss important differences between the two approaches, and highlight how we reap benefits from both by employing state-of-art innovations in BCP, decision variable selection, and backtracking. Our technique leads to a consistent performance improvement of up to 3x over state-of-the-art SAT solvers like Chaff on representative problem applications.

2. Motivation for Hybrid SAT

Boolean Constraint Propagation (BCP) forms the core of deduction in many successful and widely studied SAT algorithms based on the DPLL algorithm [9]. When a variable is assigned, BCP is used to find the consequences of the assignment, like unit clauses produced and whether a conflict exists. It constitutes about 80% of the total SAT time in our experience. Therefore, any improvement in BCP significantly benefits the overall performance of a SAT solver. We start by describing the details of the circuit-based BCP and CNF-based BCP that we use in our approach. Their relative strengths motivate the specific hybrid scheme we use.

2.1 Circuit-Based BCP

Existing circuit-based SAT solvers [4, 5, 7, 8] use a circuit representation based on AND and OR gate vertices, with INVERTERs either as separate vertices, or as attributes on the gate inputs. Constant propagation across AND/OR gates is, of course, well known, but the speed tends to be very implementation dependent. We use a lookup table for fast implication propagation [5]. Based on the current values of the inputs and output of the vertex, the lookup table determines the

DAC 2002, June 10-14, 2002, New Orleans, Louisana, USA.

next "state" of the gate where the state encapsulates any implied values and the next action to be taken for the vertex. The algorithm **imply** (from [5]), shown in Figure 1 for a generic vertex type, iterates over the circuit graph. For each vertex, it determines new implied values and the direction for further processing. As an example, Figure 2 (from [5]) shows some cases from the implication lookup table for a two-input AND gate. For Boolean logic, only one case, a logical *0* at the output of an AND vertex, requires a new case split to be scheduled for justification. All other cases either cause a conflict and backtracking, or further implications, or a return to process the next element to be justified. Due to its low overhead, this implication algorithm is highly efficient. As an indication, on a 750MHz Intel PIII with 256 MB, it can execute over one million implications per second.

```
Algorithm imply (vertex, value) {
  assign (vertex, value);
  lvalue = get_value (vertex->left);
  rvalue = get_value (vertex->right);
  next_state = lookup (value, lvalue, rvalue);
  switch (next_state) {
   case CONFLICT:
    return 0;
   case CASE_SPLIT:
    add_vertex(vertex, justification_queue);
    return 1;
    ...
   case PROP_LEFT_AND_RIGHT:
    if (imply (vertex->left, next_state->lvalue) &&
      imply (vertex->right, next_state->rvalue)) {
       return 1;
    }
    return 0;
    ...
  }
  return 1;
}
```

Figure 1: Circuit-based BCP Procedure

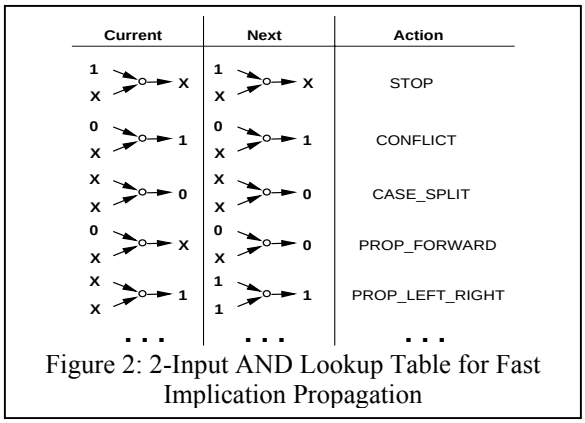

Figure 2: 2-Input AND Lookup Table for Fast Implication Propagation

2.2 CNF-Based BCP

For CNF, we use the BCP scheme called *lazy update* proposed by Chaff [1] :

1. For each clause, only two literals are monitored for state change.
2. The clause state is updated lazily when a variable is assigned, i.e., only when the two monitored literals coincide.
3. It does not require state change for clauses during the backtracking process, thus unassigning a variable takes constant time.

For clauses with many literals, lazy update works significantly better than other schemes like SATO [2], and GRASP[3]. It is especially useful for large clauses where it avoids unnecessary traversal.

2.3 Comparison of BCP Algorithms

There is an inherent overhead built into the translation of circuit gates into clauses. A two-input gate translates to three clauses in the CNF approach, while in the circuit-based approach a gate is regarded as a monolithic entity. Therefore, in the circuit approach an implication across a gate requires a single table lookup, while in the CNF approach it requires processing multiple clauses. In addition, the CNF-based BCP in Chaff does not keep track of the clauses that have been satisfied in order to reduce overheads. However, there is an inherent cost associated with visiting the satisfied clauses. Specifically, even if a clause gets satisfied due to an assignment to some un-watched literal, the watched literal pointers could still get updated. Overall, for the generally small clauses arising from circuit gates, these differences translate to significant differences in BCP time, usually in favor of the circuit-based approach.

On the other hand, learned clauses arising from conflict analysis are typically much larger than those arising from two-input gates. Adding a large learned clause as a gate tree can lead to a significant increase in the size of the circuit. This in turn, can increase the number of implications, thereby negating any potential gains obtained from circuit-based BCP. For such clauses, it is more appropriate to maintain them as monolithic clauses and take advantage of CNF-based 2-literal watching and lazy update to process them efficiently.

Based on these observations, we propose a hybrid approach in which we maintain the circuit-based logic expressions using a uniform-gate data structure, and the learned clauses as CNF. We process them separately using the appropriate BCP approach.

2.4 BCP Results

Our experimental results for BCP are shown in Table 1. The times shown are in seconds *per million implications* on a 750 MHz Intel PIII with 256 MB. The examples used are large logic formulas derived from the BMC application on a large industrial circuit. The columns `Hybrid` and `Chaff` show BCP times for the hybrid and CNF approaches, respectively, on learned clauses added during SAT as well as the original circuit formula. The size of the formula in terms of the number of primary inputs and gates is indicated in the columns `pi` and `gate`. The column `CH` is the ratio between the `Chaff` and `Hybrid` BCP times. It is clear that the hybrid approach is consistently faster than the CNF-based approach in Chaff on these large problems.

Logic Formula	pi	gate	BCP Time (sec/million implications)					
			Hybrid	Chaff	CH	Structure	CS	HS
230U	5016	228346	1.154	1.589	1.37	0.9	1.76	1.28
252U	5244	242170	1.168	1.586	1.35	0.914	1.73	1.28
272U	5244	238755	1.101	1.411	1.28	0.894	1.578	1.23
274S	3863	179217	1.11	1.52	1.369	0.908	1.67	1.22
275U	5472	255979	1.162	1.566	1.347	0.955	1.64	1.22
276U	5472	256000	1.172	1.571	1.34	0.91	1.72	1.29
405U	6612	325069	1.236	1.605	1.29	0.92	1.74	1.34
406U	6612	325090	1.207	1.639	1.35	0.94	1.74	1.28
407U	6612	325090	1.232	1.694	1.375	0.942	1.798	1.31
409U	6612	325090	1.245	1.639	1.316	0.923	1.775	1.35
435U	6840	338899	1.233	1.668	1.35	0.995	1.676	1.24
2dlx_100	557	33848	1.82	2.22	1.21	0.931	2.38	1.95

Table 1: BCP Time per million implications

To demonstrate the effect of learned clauses, we also present the BCP time per million implications for the circuit-based method on *only the circuit formula for the same problems*, shown in the Column `Structure`. Note that this is, in a sense, the best case possible, since BCP on the learned clauses is not strictly necessary for the search. Now, the column `CS (HS)` is the ratio between the `Chaff (Hybrid)` time and the `Structure` time, and reflects the overhead due to the extra work in performing BCP on the learned clauses as well. Clearly, large learned clauses introduce a significant overhead in comparison to the circuit formula alone. Our hybrid approach uses the better-suited CNF approach for large learned clauses.

3. Hybrid SAT Solver

A faster BCP with the hybrid approach is only the partial story. The additional benefit of a hybrid approach is that circuit-based decision heuristics can be easily applied, which are otherwise unavailable in a pure CNF approach. Thus, the overall speedup we observe with our hybrid solver can be attributed to the improved BCP in the deduction engine, and additional use of circuit-based heuristics in the decision engine. In this section we demonstrate the benefits derived from them. The diagnosis engine in our hybrid solver is similar to that in Chaff [1], originally proposed by GRASP [3]. In the hybrid approach, conflict driven learning records both clauses *and circuit nodes* as the reasons for the failure of search.

Our experimental results are presented in Table 2. The times are on a 750 MHz Intel PIII with 256 MB. The logic formulas are derived from the application of BMC on three large industrial circuits (bus, arbiter, and controller) and some public domain benchmarks[1] [10] for which circuit descriptions were available. (Note that use of our hybrid approach requires a circuit description, which is not available for most public benchmarks for SAT.) In order not to pollute the results, we ran the hybrid approach on more than 70 logic formulas, but report results only on those requiring more than 40s of CPU time. The formulas are distributed between unsatisfiable and satisfiable instances. The size of the formula can be determined from the `pi` and `gate` columns indicating the number of primary inputs and gates.

[1] In the table, 2dlx* is 2dlx_cc_mc_ex_bp_f_new

3.1 Comparison of Chaff and Hybrid SAT for Identical Heuristics

Our first comparison is between the hybrid solver and Chaff for exactly the same heuristics, i.e. without use of any circuit-based decision heuristics. Apart from the different BCP algorithms on the circuit formula, the two use identical algorithms for order of processing of implications, conflict-based learning, backtracking, and decision variable selection. In spite of the same heuristics, a minor difference can creep in due to the uncontrolled choice of the conflict node when several nodes are in conflict. This difference may have a pronounced effect in satisfiable instances, for which one of the two solvers may get lucky in hitting upon a solution early. However, it has relatively less effect in unsatisfiable instances since the entire search space must be explored. With this in mind, we consider only the unsatisfiable instances as reasonable data for this controlled experiment. The columns `Chaff` and `H` indicate the times for the Chaff and hybrid solvers, respectively. Due to unavailability of a circuit-based solver in public domain, we could not provide any comparison results with such solvers at this time.

It is clear that the overall performance of the hybrid solver is much better than Chaff. The typical ratio of `Chaff` time to `H`

Example	pi	gate	Chaff	H	H-jft	C/H	C/H-jft
Unsatisfiable Instances							
53U	6384	546779	1893.43	1629.05	639.20	1.16	2.96
55U	6612	571185	1585.08	1402.80	715.20	1.13	2.22
51U	6156	524116	1283.82	1010.90	508.80	1.27	2.52
47U	5700	477047	1190.05	863.60	391.87	1.38	3.04
49U	5928	499710	1072.64	738.30	606.70	1.45	1.77
435U	6840	338899	922.61	326.50	317.00	2.83	2.91
3pipe	198	10810	920.50	245.60	566.90	3.75	1.62
45U	5472	452641	888.86	415.27	443.40	2.14	2.00
434U	6840	338899	880.84	532.10	318.00	1.66	2.77
407U	6612	325090	736.49	367.20	588.00	2.01	1.25
438U	6840	338899	647.71	427.20	283.00	1.52	2.29
437U	6840	338899	621.17	281.10	419.00	2.21	1.48
439U	6840	338899	599.14	441.80	453.00	1.36	1.32
436U	6840	338899	559.08	369.60	384.00	1.51	1.46
105U	1017	94528	547.68	248.90	506.30	2.20	1.08
409U	6612	325090	496.31	437.20	329.00	1.14	1.51
406U	6612	325090	486.86	353.90	317.00	1.38	1.54
405U	6612	325069	471.50	585.00	445.00	0.81	1.06
43U	5244	429978	471.13	353.87	181.25	1.33	2.60
109U	1055	101180	441.33	173.70	229.00	2.54	1.93
408U	6612	325090	433.21	358.10	347.00	1.21	1.25
41U	5016	405572	364.03	230.27	233.80	1.58	1.56
39U	4788	382909	307.59	208.87	120.20	1.47	2.56
101U	979	88088	230.97	141.00	117.00	1.64	1.97
276U	5472	256000	193.24	132.30	104.00	1.46	1.86
37U	4560	358503	174.47	146.96	139.12	1.19	1.25
252U	5244	242170	165.46	118.30	93.50	1.40	1.77
35U	4332	335840	165.20	128.90	76.02	1.28	2.17
275U	5472	255979	162.97	132.10	129.00	1.23	1.26
230U	5016	228346	132.54	100.60	77.00	1.32	1.72
33U	4104	311434	98.64	80.10	42.10	1.23	2.34
31U	3876	288771	92.22	51.30	29.30	1.80	3.15
2dlx*	414	25075	89.07	49.30	72.40	1.81	1.23
272U	5244	238755	47.54	27.70	8.00	1.72	5.94
Satisfiable Instances							
109S	2374	144590	811.12	330.4	182.5	2.45	4.44
105S	2286	134654	432.22	196.4	72.9	2.20	5.93
108S	2367	142198	423.63	766.8	138.5	0.55	3.06
101S	2198	125070	388.35	399.38	111.9	0.97	3.47
102S	2235	127558	344.34	312.7	144.4	1.10	2.38
103S	2242	129818	343.95	481.8	86.7	0.71	3.97
107S	2330	139578	286.1	256.4	178.2	1.12	1.61
104S	2279	132350	270.01	384.4	49.1	0.70	5.50
106S	2323	137230	226.57	198.58	94.4	1.14	2.40
100S	2191	120305	153.41	374.8	97.8	0.41	1.57
2dlx_100	557	33848	120.12	76.8	17	1.56	7.07

Table 2: Comparison of Hybrid SAT Solver and Chaff

time, shown in the Column C/H, is greater than 1.3, with the maximum being 3.75. The ratio of the total time spent in Chaff to the total time spent in the hybrid solver for all the unsatisfiable instances is 1.48. Due to possibly a bad choice for a conflict node, we do see a larger time with the hybrid solver in a few cases, e.g., 405U.

For the satisfiable instances shown in Table 2, the Chaff to Hybrid ratio is distributed evenly on either side of 1.0 with a large standard deviation. This is most likely due to the randomness in choice of a conflict node, as explained earlier.

3.2 Incorporation of Circuit-Based Heuristics

Once it is established that the hybrid approach is, in fact, faster, it becomes possible to apply low overhead circuit-based heuristics that would be unavailable in the CNF domain to obtain an even higher speedup. Information regarding gate fanout/fanin, paths, and signal direction becomes readily available without overhead. We find that such an approach works much better than the circuit-oblivious decision heuristics used in the pure CNF-based methods. As an example, the column H-jft in Table 2 shows the time in our hybrid SAT solver with the use of the justification frontier heuristic [7]. This heuristic restricts the decision nodes to be those that justify the values on their fanout node. The use of this heuristic further improves performance in 24 of the 34 unsatisfiable instances. The ratio of the total time spent in Chaff to the total time spent in the hybrid solver for the unsatisfiable instances is now 1.89 versus the earlier 1.48, with the maximum ratio being 5.94 versus the earlier 3.75.

Now consider the satisfiable instances in Table 2. Where there was wide variation in the speedups (or lack thereof) earlier, we now have consistent and significant speedup over the CNF approach. The ratio of the total time spent in Chaff to the total time spent in the hybrid solver is now 3.24 for the satisfiable instances with the maximum ratio being 7.07. Clearly, the circuit heuristics lead to a satisfying solution much faster than the decision heuristics in Chaff.

Based on these data, there appears to be a clear benefit in using the circuit-based heuristics for both the satisfiable and unsatisfiable instances. In a sense, the hybrid approach allows us to exploit the benefits of the circuit-based as well as the CNF-based heuristics.

3.3 Impact on Verification Applications

SAT is a core engine in many verification applications like equivalence checking [5], and BMC [6]. In fact, it is typical for a long BMC run to last for multiple days, requiring thousands of applications of the SAT solver. A speed up by a factor of two in the core engine, therefore, can prove to be very significant since it would lead to a large absolute saving in the run time. Another practical aspect of our hybrid approach is that it is unnecessary to incur the overhead of copying the entire circuit into a CNF data structure for the purpose of SAT. This has the benefit of reducing the memory requirement of these applications, allowing them to scale to larger circuits, or in the case of BMC, also to a larger number of time frames.

4. Conclusions

We have presented a hybrid SAT solver that combines the strengths of circuit-based and CNF-based SAT solvers. We demonstrated it to be the highest performing SAT solver in circuit-based application domains. Our approach is based on an analysis of the source of efficiencies in state-of-art techniques for BCP, decision variable selection, and backtracking. We apply this understanding to develop SAT techniques leveraging off the circuit nature of these application domains while maintaining advantages arising from the CNF representation. Moreover, any future improvements in the performance of circuit-based and CNF-based SAT solvers can directly translate into improvements of the hybrid SAT solver as well.

References

[1] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of Design Automation Conference*, 2001.

[2] H. Zhang, "SATO: An efficient propositional prover," in *Proceedings of International Conference on Automated Deduction*, vol. 1249, *LNAI*, 1997, pp. 272-275.

[3] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Transactions on Computers*, vol. 48, pp. 506-521, 1999.

[4] M. Schulz, E. Trischler, and T. Sarfert, "SOCRATES: A highly efficient ATPG System," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, pp. 126-137, 1988.

[5] A. Kuehlmann, M. Ganai, and V. Paruthi, "Circuit-based Boolean Reasoning," in *Proceedings of Design Automation Conference*, 2001.

[6] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," in *Proceedings of Workshop on Tools and Algorithms for Analysis and Construction of Systems (TACAS)*, vol. 1579, *LNCS*, 1999.

[7] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Transactions on Computers*, vol. C-32, pp. 265-272, 1983.

[8] P. Goel, "An implicit enumeration algorithm to generate tests for Combinational circuits," *IEEE Transactions on Computers*, vol. C-30, pp. 215-222, 1981.

[9] M. Davis, G. Longeman, and D. Loveland, "A Machine Program for Theorem Proving," *Communications of the ACM*, vol. 5, pp. 394-397, 1962.

[10] M. N. Velev, "Benchmark Suites. http://www.ece.cmu.edu/~mvelev," October 2000.