University of Nebraska - Lincoln

DigitalCommons@University of Nebraska - Lincoln

CSE Conference and Workshop Papers

Computer Science and Engineering, Department
of

1991

PARALLEL TEST PATTERN GENERATION USING BOOLEAN SATISFIABILITY

V. Sivaramakrishnan

University of Nebraska-Lincoln

Sharad C. Seth

University of Nebraska-Lincoln, seth@cse.unl.edu

Prathima Agrawal

AT&T Bell Laboratories, Murray Hill, NJ

Follow this and additional works at: <https://digitalcommons.unl.edu/cseconfwork>

Part of the [Computer Sciences Commons](#)

Sivaramakrishnan, V.; Seth, Sharad C.; and Agrawal, Prathima, "PARALLEL TEST PATTERN GENERATION USING BOOLEAN SATISFIABILITY" (1991). *CSE Conference and Workshop Papers*. 54.

<https://digitalcommons.unl.edu/cseconfwork/54>

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

PARALLEL TEST PATTERN GENERATION USING BOOLEAN SATISFIABILITY

V. Sivaramakrishnan

Sharad C. Seth

University of Nebraska, Lincoln, NE 68588

Prathima Agrawal

AT&T Bell Laboratories, Murray Hill, NJ 07974

Abstract

Recently, Larrabee proposed a sequential test generation algorithm for combinational circuits based on boolean satisfiability and presented results on benchmark circuits in support of the viability of this approach. Parallel implementations of test generation algorithms are attractive in view of the known difficulty (NP-completeness) of the problem. In this paper we suggest parallel versions of Larrabee's algorithm, suitable for implementation on shared-memory and message-passing multicomputers.

1 Introduction

For a given logic circuit, a test pattern generator produces a set of input patterns that cause different responses from the faulty and fault-free circuits. These test pattern generators search for a solution in the input space and the worst-case search complexity is exponential in the number of primary inputs. All the well-known test pattern generators are topology based (e.g. [1] [2]). For circuits of VLSI complexity, most of their time is apt to be spent on a relatively small number of hard-to-detect (HTD) faults. Multiprocessors have been tried in an attempt to speed up the solution.

A well known test generation algorithm, PODEM, was recently implemented [3] on a 16 processor iPSC/2 hypercube with a dynamic search

space allocation strategy. Another implementation [4] used varying number of processors. In this paper we propose parallel implementations of a recent algorithm based on boolean satisfiability [5]. The main idea of this algorithm can be summarized as follows.

A circuit is modeled as a directed acyclic graph with each node representing a gate or a fanout point. With each gate, we associate a logic formula or a clause, in 3-element conjunctive normal form, or 3CNF. This logic formula is true if, and only if, the variables representing the gate's inputs and output take values consistent with its truth table. We obtain a formula for the entire circuit by logically ANDing the formulas for individual gates. This formula can easily be derived by traversing the circuit graph from primary outputs and forming the AND or conjunction of formulas of all nodes visited. It describes the fault-free circuit output, that will be true when variables are assigned values that are consistent with the truth tables of the logic gates. A faulty circuit is represented by a copy of the fault-free circuit with renamed variables for the wires affected by the fault. A formula describing the faulty circuit output in terms of the primary inputs is obtained. The Boolean difference with respect to the fault is defined as the XOR of unfaulted circuit output and its faulted circuit output [6]. Figure 1 shows an example circuit and Figure 2 shows XORing of the good and faulty circuits for a fault on line D. A composite formula for the combined circuit is constructed by taking the conjunction of the good circuit formula and the faulty circuit formula together with the XOR

Figure 1: An Example Circuit

gate. We call this the Boolean difference formula. A conflict-free assignment of values to the variables in the Boolean difference formula results in the output (Boolean difference) of the composite circuit to be a logical one or TRUE. This is known as the Boolean satisfiability condition. The values assigned to the primary inputs in this assignment are a test for the fault in question. Although the Boolean difference formula embodies all tests that distinguish between the fault-free and faulty circuits, any one of them suffices as a test.

A circuit with n outputs will generate n Boolean difference formulas. For detecting a given fault, at least one of these formulas should be satisfied. If none can be satisfied the fault is declared undetectable or redundant.

The problem of test generation is thus translated to a problem of assigning true and false values to circuit variables so that the Boolean difference formula is satisfied. In other words, suitable assignment will ensure that none of the constituent clauses reduce to false.

Figure 2: XORing of the Faulty and the Good Circuit to derive the Boolean Difference Formula

The problem of test generation is known to be NP-complete [7] as is the problem of satisfying 3CNF [8]. Thus, the above formulation of test generation as a satisfiability problem, may be said to provide just a different view of a difficult problem. However, the satisfiability formulation suggests a solution method which is not readily apparent from the circuit topology. It is based on the observation that at least two thirds of the clauses generated for the Boolean difference of a combinational circuit have only two literals, i.e., they are 2CNF clauses. Further, finding a satisfying solution to 2CNF (a 2SAT solution) can be carried out in linear time [9]. Hence Larrabee suggested the following approach to test generation: generate 2SAT solutions to the Boolean difference formula in some fixed order until one is found which also satisfies the 3CNF clauses (i.e. is a 3SAT). Since there may be an exponential number of 2SAT solutions to try, heuristics are employed to speed up the search for one that is also a 3SAT.

Figure 3: Implication Graph for the circuit in figure 2

The sequential algorithm, described below, incorporates this idea and forms the basis for our parallel implementation. It is a variation of the original algorithm due to Larrabee [5]. The parallel algorithm would normally be used as the second part of a test generation system, where tests may be attempted only for hard-to-detect faults. The first part of test pattern generation filters out easier to detect faults using one of the standard test generation algorithm, such as PODEM, using a small backtrack limit (say, 25).

2 Sequential Algorithm

Each 2CNF clause of the form $(a+b)$ can be viewed as two implications, $a' \Rightarrow b$ and $b' \Rightarrow a$ (' represents the complement) and an implication graph can be derived. Figure 3 shows the implication graph for the example circuit. Satisfying the 2CNF terms requires assigning true and false values to the nodes in the implication graph so that no node assigned true leads to a node labeled false. When a true node precedes a false node in the implication graph, both literals that make up the 2CNF clause are assigned false and the Boolean difference formula cannot

be satisfied. The strongly connected components can be collapsed in a preprocessing phase such that there is no cycle in the graph. A strongly connected component represents variables in an equivalence class that must all be assigned the same value. If a strongly connected component contains both a literal and its negation, the formula is not satisfiable since it introduces an edge that leads from a node assigned true to a node assigned false.

Our sequential algorithm for boolean satisfiability is shown in Figure 4, where additions to Larrabee's original algorithm are shown within dotted lines. We will first describe the original algorithm. It assumes that the 2CNF variables have been ordered and placed in the array V in a preprocessing phase. An index i points to the first unbound variable in the array, and the variable dir keeps track of whether or not we are backtracking. In the forward direction, each iteration through the loop is made either in the forward or in the backtracking direction and results in a new (partial) assignment of variables which is consistent with the 2CNF clauses. This is checked for consistency with the 3CNF clauses at the end of the loop and the dir variable is set appropriately before the next iteration is started.

algorithm SAT

```

V ← all Unbound; i ← 0; dir ← Forward;
loop
  if 2SAT then
    if 3SAT then exit successfully
    else dir ← Backward
  if dir = Forward then
    while i ≠ size(V) and V[i] is bound do
      i ← i+1
    end;
    if i = size(V) then exit successfully;
    V[i] ← 0;
    set direct implications of V[i];
    i ← i+1;
  end;
  elseif dir = Backward then
    if i < 0 then exit unsuccessfully end;
    temp ← V[i];
    Undo direct implications of V[i];
    V[i] ← Unbound;
    if temp = 0 then
      V[i] ← 1;
      Set direct implications of V[i];
    else
      i ← i-1;
    end
  endif
  if no clause falsified then dir ← Forward
  else dir ← Backward
end
endloop

```

algorithm 3SAT

```

limit ← i; dir ← Forward;
Reorder unbound variables in V[i], V[i+1], ..., V[size(V)-1];
loop
  if dir = Forward then
    while i ≠ size(V) and V[i] is bound do
      i ← i+1
    end;
    if i = size(V) then exit successfully end
    V[i] ← 0;
    i ← i+1;
  elseif dir = Backward then
    if i < limit then exit unsuccessfully end;
    temp ← V[i];
    V[i] ← Unbound;
    if temp = 0 then
      V[i] ← 1;
    else
      i ← i-1;
    end
  endif
  if no clause falsified then dir ← Forward
  else dir ← Backward
end
endloop

```

Figure 4: Sequential Algorithm

In the forward direction, the current prefix of bound values is extended by assigning a zero value to the first unbound variable in V . Assigning a zero value to a node implies that all its preceding nodes in the implication graph must also be set to zero (a count is kept of how many times each node is forced to a value so that the effect of the assignment can be correctly undone in a later backtracking step). In the backtracking direction, the one-value is tried for a variable before making it unbound and backtracking to a previous variable in V . Whenever a variable is changed from a bound to an unbound state,

it is necessary to undo the implications of the previous binding of this variable.

The order of variables in V has great effect on the performance of this algorithm. Several heuristics, based on the connectivity of the implication graph, are given by Larrabee to determine this order and it is suggested to switch heuristics if a solution is not found within a reasonable time. However, any ordering based on implication graph, becomes irrelevant as soon as the current assignment of variables becomes a 2SAT solution (since all the constraints stated in the im-

assignments to the first n variables. An initial process can be used to partition search space and start each group. It is assumed that each SAT processor has its own images of the implication graph and the 3CNF clauses to work with. Each group runs independently until a 3SAT solution is found by any group.

At the second level, concurrency is achieved by SAT spawning off 3SAT as a new process, if a processor is available to run it, otherwise continuing to run it sequentially itself. Each 3SAT process must be passed on the variable assignments and the resulting 3CNF at the time of initiation. Thereafter, it can run independently until either a solution is found or no solution is possible. Further concurrency may be possible by implementing SAT as a pipelined process.

The parallel algorithm could be implemented effectively on either a shared-memory or a message-passing architecture. Both kinds of implementations will be tried to validate this claim. We will use the Sequent for a shared-memory system and either MARS [10] or Transputer for a message-passing system. Single processor versions will be used to measure the actual speedup obtainable on the standard benchmark circuits.

4 Conclusion

We have proposed a variant of the boolean satisfiability algorithm for test generation as the basis of parallel implementation. The parallel scheme involved both data-parallel and pipelined forms of concurrency. These algorithms are currently being implemented. We plan to study their effectiveness using benchmark circuit examples. Efficient ways of partitioning the implication graph and pruning the search process are also being explored.

5 References

- 1 P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Computer*, vol. C-30, pp. 215-222, March 1981.
- 2 H. Fujiwara and T. Shiono, "On the acceleration of test generation algorithms," *IEEE Trans. Computer*, vol. C-32, pp. 1137-1144, December 1983.
- 3 S. Patil and P. Banerjee, "A parallel branch and bound algorithm for test generation," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 313-322, March 1990.
- 4 S. Arvindham, V. Kumar, V.N. Rao and V. Singh, "Automatic Test Pattern Generation on Multiprocessors," *MCC Technical Report*, 1989.
- 5 T. Larrabee, "Efficient generation of test patterns using Boolean difference," *Proc. 1989 Int. Test Conf.*, pp. 795-801, August 1989.
- 6 F.F. Sellers, M.Y. Hsiao and L.W. Bearnson, "Analysing errors with Boolean difference" *IEEE Trans. Computer*, vol. C-17, pp. 676-683, July 1968.
- 7 P.H. Ibarra and S.K. Sahni, "Polynomially complete fault detection problems" *IEEE Trans. Computer* Vol. C-24, pp.242-249, May 1975.
- 8 S.A. Cook, "The complexity of Theorem Proving Procedures" *Proceedings of the third annual ACM Symposium of Theory of Computing*, 1971, 151-158.
- 9 B. Aspvall, M.F. Plass and R.E. Tarjan, "A linear-time algorithm for testing the truth of certain quantified Boolean formulas," *Information Processing Letters*, Vol. 3, No. 8, pp. 121-123, March 1979.
- 10 P. Agrawal et al "Architecture and design of the MARS hardware accelerator," *Proc. 24th Design Automation Conf.*, pp. 101-106, June 1987.