A Hybrid Distributed Test Generation Method Using Deterministic and Genetic Algorithms

Haidar Harmanani and Bassem Karablieh
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010
Lebanon

Abstract

Test generation is a highly complex and time-consuming task. In this work, we present a distributed method for combinational test generation. The method is based on a hybrid approach that combines both deterministic and genetic approaches. The deterministic phase is based on the D-algorithm and generates an initial set of test vectors that are evolved in the genetic phase in order to achieve a high fault coverage in a short time. The algorithm is parallelized based on a cluster of workstations using the Message Passing Interface (MPI) library. Several benchmark circuits were attempted, and favorable results comparisons are reported.

1. Introduction

Automatic test pattern generation (ATPG) is the process of generating patterns to test digital circuits. Deterministic test generation is achieved by injecting a fault into a circuit and then the use of a specific algorithm to activate the fault thus causing the fault's effect to propagate through the hardware and to manifest itself at a primary output [1]. If all possible circuit assignments are attempted and no vector is found, the ATPG declares the fault as untestable. Such algorithms usually utilize the gate level description of a circuit in order to generate a condensed set of tests. They require backtracking and lead to an exponential search space in the number of primary inputs [3]. Various deterministic techniques that are based on path sensitization methods were proposed such as the D-algorithm [9], PODEM [5], FAN [4] and SOCRATES [14].

Simulation based techniques, on the other hand, were proposed in order to avoid the long execution time by processing the circuit in the forward direction only [15, 16]. Such algorithms generate vectors that don't target a specific fault in a circuit and use simulation to find the effect of the new test vector on the circuit. In addition to the above algorithmic techniques that maybe used in order to improve the efficiency of an ATPG, parallel algorithms may also be used in order to reduce the test generation computational time. Various approaches have been proposed for fault generation and simulation parallelization such as *fault partitioning* where the fault list is distributed among available processors. Other alternatives for parallelization are *circuit partitioning* where the circuit that is being simulated is partitioned among available processors and *algorithmic partitioning*.

1.1. Genetic Algorithms and Test Generation

Genetic algorithms are optimization techniques that incorporate features of natural evolution. They are non-deterministic and search for the best solution by exploring favorable characteristics of previous solutions. A generation is a GA iteration where individuals in the current population are selected for crossover and offspring are created. Genetic algorithms have been used in digital test generation [2, 8, 10, 12] and high-fault coverages and fast execution times have been reported for several circuits. Genetic-based test generation algorithms typically start with a random initial population that is used to generate each test vector and a fault simulator is used to evaluate the fitness of each set. The genetic algorithm is evolved over several generations and the best test found is saved separately from the population. Generations of individual test vectors is repeated until no more improvements in fault coverage is obtained.

One of the main problems of genetic-based test generation is that, as stand-alone procedures, test generation may fail to achieve complete fault coverage in a reasonable time [11]. Furthermore, though they generate sometimes a high-fault coverage, the generated test sets often have lower fault coverage than those generated by a deterministic test generator due to hard-to-test faults. In order to alleviate this

1098-8068/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

problem, test generation procedures based on genetic optimization were combined with deterministic test generation [10, 13]. In the various reported hybrid approaches, the genetic algorithm is followed by a deterministic phase in order to detect the undetected faults.

This paper presents a hybrid distributed ATPG system for combinational circuits. The algorithm is composed of two main phases, a *deterministic phase* that is followed by an efficient *genetic algorithm*. The algorithm is parallelized using fault partitioning based on a master-slave model.

The remainder of the paper is organized as follows. Section 2 presents the deterministic phase of the test generation system, the non-deterministic phase, as well as the parallel algorithm. Section 3 presents the experimental results while concluding remarks are presented in section 4.

2. The Parallel ATPG Algorithm

Our proposed hybrid distributed ATPG system is composed of two main phases. The first phase is deterministic and based on the D-algorithm. It uses the single stuck-at fault model and the five-valued logic with logic values $0, 1, X, D, \overline{D}$ while the second phase is based on an efficient genetic algorithm.

2.1. Deterministic Phase

A fault is detected if it can be observed at a primary output. The deterministic algorithm considers each fault in a circuit and tries to find the appropriate test vector for detecting this fault through path sensitization. Thus, for every fault in the circuit, the algorithm activates the fault by assigning a D or $\overline{D}$ on the target line and drives the fault by assigning the uncontrollable values for a gate, and propagates the values of the gates to all the circuit. The fault is propagated until it reaches the primary output. If the fault reaches a primary output successfully then the values that were used to drive the fault should be justified. To justify these values appropriate values are placed at the inputs of the justified gates. The number of backtracks that a system will perform in justification of a single d-drive path can be approximated to the order of 2^n where n is the number of levels to be traversed in the justification process. To improve the performance, constraints were added on the number of backtracking operations that the algorithm will perform. For each fault the algorithm limits the number of backtrack operations in the circuit to 16 levels. Thus, a maximum of 2^{16} backtrack operations are performed per injected fault; if the algorithm backtracks for all the cases in the 16 levels and fails to reach the primary input, then the fault is not testable using the chosen drive path. On the other hand and in order to reduce the depth of the search tree, the number of drives that the algorithm performs is limited as well. Therefore, if a fault cannot be justified in the first forward path then the fault will be declared as untestable. The algorithm will not be allowed to backtrack and try another path though a longer search might find a vector to test that fault. Thus, this constraint improves the timing performance of the algorithm tremendously, yet will cause degradation in the fault coverage. Finally, during the deterministic test generation, some inputs are left unspecified. In order to increase the fault coverage, such unspecified inputs are randomly assigned to either 0 or 1. The unspecified values are further explored by the genetic operators in order to achieve a better fault coverage. Figure 1 illustrates the flowchart of the deterministic phase of the test algorithm.

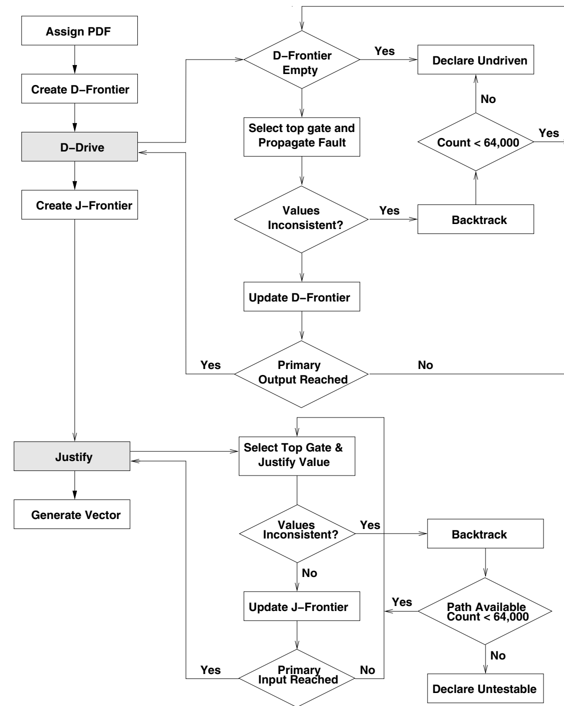


Figure 1. Deterministic fault detection

2.2. Non-deterministic Phase

The deterministic algorithm finds in a very short time an efficient test set, though the fault coverage is not high due to the restriction on the backtrack operations as well as the limitation on the depth of the search tree. The non-deterministic algorithm improves the quality of the test set using a genetic algorithm. Genetic-based test generation problems suffer due to the randomness nature of the genetic operators [8]. For example, the crossover operator does not take advantage of the circuit property and relies on the consideration of single input values. Thus, genetic-based test generation algorithms tend to be slower and more complex. In order to alleviate the above problem, we follow the deter-

1098-8068/05 $20.00 © 2005 **IEEE**
IEEE COMPUTER SOCIETY

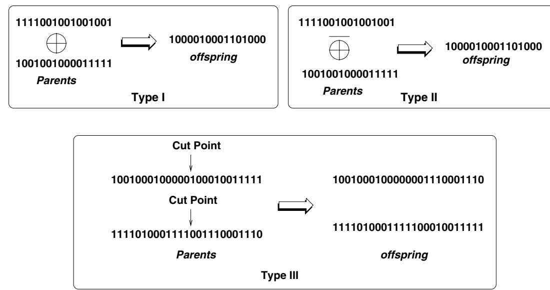


Figure 2. Example crossover operation

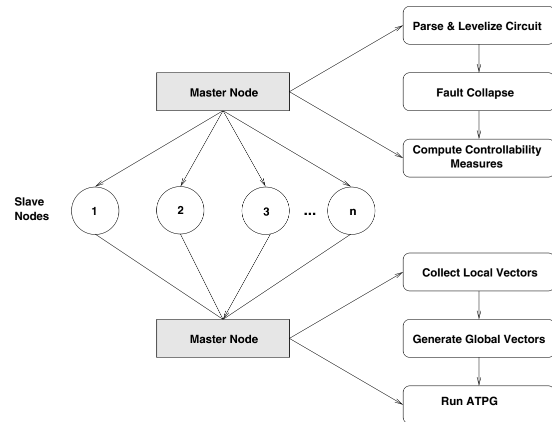


Figure 3. Parallel architecture

ministic phase with a genetic phase that uses the deterministically generated test vectors as an initial population. The population is evolved based on a series of genetic operators. The test set quality is evaluated using fault simulation.

2.2.1 Chromosomal Representation

The set of test vectors correspond to a population. Every individual or chromosome in the population represents a test vector. Thus, every gene in a chromosome is mapped to a primary input.

2.2.2 Initial Population

An initial population is important in order to generate a certain predefined number of solutions. The quality of the final solution is affected by how the initial solution has been constructed. The initial population is directly generated based on the deterministic phase thus resulting in a highly fit population.

2.2.3 Selection, Reproduction, and Cost Function

The initial population is evolved through several generations by randomly selecting two individuals and mating them. Every generation affects only a fraction of the population. Thus, the fitness of the population increases in successive generations. The population constitutes a test vector depository where every chromosome detects a unique set of faults. Chromosomes are not replaced unless another chromosome is generated that detects the same faults in addition to other faults. The fitness of the population is the number of distinct faults a chromosome detects. The fitness of the overall population is the sum of fitness of all the chromosomes in the population.

2.2.4 Genetic Operators

1. *Mutation:* The mutation operator produces incremental random changes in the offspring. Since the parent is deleted after this operation, only genes that are left *unspecified* during the deterministic phase are subject to mutation. The mutation operator randomly selects a chromosome and complements all bits that are in the *unspecified* positions.

2. *Crossover Operators:* In order to efficiently explore the search space, we use three crossover operators that provide an efficient mechanism for the offspring to inherit the characteristic of better parents. The parent vectors are not deleted as a result of the crossover operations, and thus all genes are subjected to these operations. The crossover operators are:

   - A differential crossover that favors non-equal allele values. The operation randomly selects two chromosomes and applies the binary XOR operation.
   - An equalizing crossover that favors equal allele values. The operation randomly selects two chromosomes and applies the XNOR operation on the two selected chromosomes.
   - A combination crossover that randomly selects two chromosomes from the population and combines them at a random cut point.

2.3. The Parallel ATPG Algorithm Implementation

The proposed ATPG algorithm was parallelized through fault partitioning based on a master-slave architecture. The master process reads the circuit, levelizes it, and computes the controllability and observability measures based on SCOAP [6]. The fault list is next generated and partitioned among the available slave processes on the cluster so that to reduce idle time. Every slave process applies the path sensitization algorithm on its fault list in order to generate the appropriate test patterns. Once the test patterns are generated, they are sent to the master where they are added

IEEE COMPUTER SOCIETY

circuit	inp	det.able	det.ed	patt	tests
c880	60	942	942	942	191
c1355	41	1566	1566	1564	150
c1908	33	1870	1870	1743	177
c2670	233	2630	2630	2050	500
c3540	50	3291	3291	3686	325
c5315	178	5291	5291	5889	717
c6288	32	7710	7710	809	162
c7552	207	7419	7419	7843	607

Table 1. ISCAS results based on our system

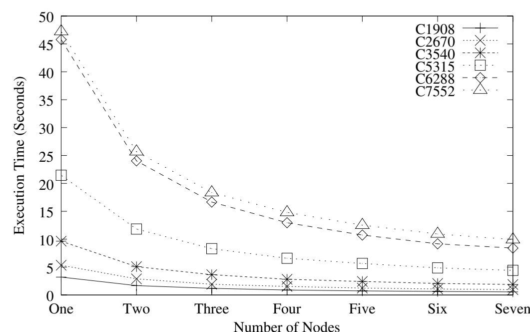


Figure 4. Benchmark execution time

to the fault depository and fed back to the genetic algorithm as an initial population. The genetic algorithm is executed sequentially on the master process. Figure 3 shows the general architecture of the parallel design.

Efficient fault partitioning is important in order to ensure a comparable load on all processors and to increase the efficiency of the parallel algorithm [17]. We partition the fault list among all processors based the difficulty of controlling and observing the faults using the SCOAP's controllability measure that approximately quantifies how hard it is to set and observe the internal signals of the circuit [6]. The fault lists are ranked according to the ratio of their controllability to the level of the fault in the circuit. The fault list is partitioned based on the following simple algorithm:

- Partition the fault list F into four groups, F_1, F_2, F_3, and F_4.
- Let F_1 contains all the faults whose controllability measure is three times its level.
- Let F_2 contains all the faults whose controllability measure is two times its level.
- Let F_3 contains all the faults whose controllability measure is 1.5 times its level.
- Let F_4 contains all the faults whose controllability measure is less than 1.5 times its level.
- Divide F_1, F_2, F_3, and F_4 equally among all the MPI processes where each process will handle a portion of each set and generates the corresponding test vectors.

3. Experimental Results

We implemented the proposed distributed hybrid ATPG algorithm using the Message Passing interface (MPI) and the C language. The algorithm was tested on a 7-nodes Linux Beowulf cluster. Each node in the cluster is an Intel Pentium 4 processor with 1.4 GHz clock speed with 512 megabyte of physical memory. The cluster runs Red Hat Enterprise Linux. The cluster is organized in two different subnets for maximum performance.

We fixed the number of generations to 600 in order to limit the execution time. The population size did expand beyond the initial size as new test vectors were added to the initial test set that was generated by the deterministic phase. The probability of the mutation operator as well as the probability of the combination crossover operator were chosen to be 0.2 while the other crossover probabilities were chosen to be 0.6. The combination crossover was applied every generation while all other operators were applied every 10 generations.

3.1. Test Generation Results

The algorithm was tested using the ISCAS 85 combinational logic circuits. Some easy circuits such as *c880* and *c6288* achieved the highest possible fault coverage even before applying the genetic algorithm. The results of the test generation are shown in Table 1 where we show under the column *det.able* the number of stuck-at faults in each circuit. Under column *det.ed* we show the number of faults detected by our method. The number of patterns simulated before the final test set was obtained is shown in under column *patt*. Finally, we show under column *tests* the size of the derived test set. It should be noted that the test set was not compacted. It is clear that our algorithm detects all detectable faults in the circuit. Figure 5 illustrates the fault simulation as well as speedup for the remaining ISCAS circuits. The fault simulation graphs show a period of saturations with fault coverage. This is the state where all the easy to test faults are detected by the deterministic algorithm. A noticeable increase in fault coverage occurs after the genetic algorithm is executed. Table 2 compares the performance of our method with [7, 8, 11, 12]. Though most shown algorithms achieve the same fault coverage, it is noticeable that algorithms that are based on pure non-deterministic test techniques require a lot more test patterns than our hybrid algorithm especially in the case of circuits

1098-8068/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY

circuit	Ours		[8]			random [8]		[7]		[11]		[12]	
	det.ed	patt	det.ed	patt	tests	det.ed	patt	det.ed	patt	det.ed	tests	det.ed	patt
c880	942	942	942	3000	115	942	10102	942	15000	942.0	35.7	937	5309
c1355	1566	1564	1566	3000	109	1566	2529	1566	3000	1566.0	84.1	1536	-
c1908	1870	1743	1870	3000	176	1870	6742	1870	50000	1868.5	115.2	1852	4501
c2670	2630	2050	2630	138000	187	2357	128000	2630	330000	2526.3	62.4	2290	-
c3540	3291	3686	3291	24000	276	3291	37686	3291	45000	3288.8	121.2	3277	8000
c5315	5291	5889	5291	4000	221	5291	3934	5287	1500	5291.0	83.6	5258	5258
c6288	7710	809	7710	-	-	-	-	7710	1500	7710.0	17.9	7709	2822
c7552	7419	7843	7419	787000	389	7270	128000	7362	9000000	7356.0	133.2	7120	-

Table 2. Comparison with other methods

that are not random pattern testable. The number of test patterns in our case is consistently small and related to the number of faults in the circuit. Finally, the execution time of our system is shown in Figure 4. It should be noted that in the case of [11], which is a hybrid algorithm where the genetic algorithm is followed by a deterministic algorithm, the system failed to detect all the faults at each run and the shown values are averages that were derived at various runs.

Another point of interest in our parallel implementation is the *speed-up, S(p)*. Typically, the maximum possible speedup is the linear speedup, that is p with *p processors*. However and in practice, the maximum achievable speedup is limited by the fraction of the computation that cannot be divided into concurrent tasks. The speedup factor increased as the number of processors increased. The achieved speedup using *seven* processors varied between 1 and 4.74 (in the case of the $C7552$ benchmark). Figure 5 illustrates the speedup factor for the same benchmark examples.

4. Conclusion

We presented an efficient and fast algorithm for the combinational ATPG problem, an NP complete problem. The method is based on a hybrid deterministic method and a genetic algorithm. The algorithm was parallelized on a cluster of workstations and favorable results were reported.

References

[1] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing for Digital, Memory, and Mixed Signal VLSI Circuits*, Kluwer, 2000.

[2] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda, “GATTO: a Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits,” *IEEE Trans. on CAD*, Vol. 15, No. 8, pp. 943-951, 1996.

[3] H. Fujiwara and S. Toida, “The Complexity of Fault Detection Problems for Combinational Logic Circuits,” *IEEE Trans. on Computers*, pp. 555-560, 1982.

[4] H. Fujiwara and T.Shimono, “On the Acceleration of Test Generation Algorithms,” *IEEE Trans. On Computers,* Vol. C-32, pp. 1137-1144, 1983.

[5] P. Goel, “An Implicit Enumeration Algorithm to Generate Tests for Computation Logic Circuits,” *IEEE Trans. On Computers*, Vol. C-30, No. 3 pp. 215-222, 1981.

[6] L. Goldstein and E. Thigpen, “SCOAP: Sandia Controllability/Observability Analysis Program,” *in Proc. DAC,* pp. 190-196, 1980.

[7] H. K. Lee and D. S. Ha, “An Efficient Forward Fault Fault Simulation Algorithm Based on the Parallel Pattern Single Fault Propagation,” *in Proc. ITC*, pp. 946-955, 1991.

[8] I. Pomeranz and S. M. Reddy, “On Improving Genetic Optimization Based Test Generation,” *in Proc. European Design and Test Conf.*, pp 506-511, 1997

[9] J. P. Roth, W. G. Bouricius and P. R. Schneider,“Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits,” *IEEE Trans. On Electronic Computers*, Vol. EC-16, No. 10, pp 567-579, Oct. 1967.

[10] E. Rudnick and J. Patel, ”Combining Deterministic and Genetic Approaches for Sequential Circuit Test Generation,” *in Proc. DAC*, pp. 183-188, 1995.

[11] E. Rudnick and J. Patel, ”A Genetic Algorithm Framework for Test Generation,” *IEEE Trans. on CAD*, Vol. 16, No. 9, 1997.

[12] D. G. Saab, Y. Saab and J. Abraham, “CRIS: A Test Cultivation Program for Sequential VLSI Circuits,” *in Proc. ICCAD*, pp. 216-219, 1992.

[13] D. G. Saab, Y. G. Saab, and J. A. “Iterative [simulation-based genetic + deterministic techniques] = complete ATPG,” *in Proc. ICCAD*, pp. 40-43, Nov. 1994.

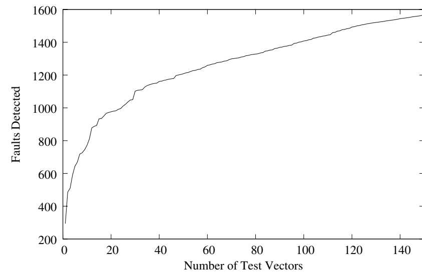
[14] M. H. Schulz, E. Trischler, and T.M. Sarfert, “SOCRATES: A Highly Efficient Automatic Test Pattern Generation System,” *IEEE Trans. CAD,* Vol. 7, pp. 126-137, 1988.

[15] H. D. Schnurmann, E. Lindbloom, and R. G. Caroenter, “The Weighted Random Test Pattern Generator,” *IEEE trans. on Computers,* Vol. C-24, No. 7, pp. 695-700, 975.

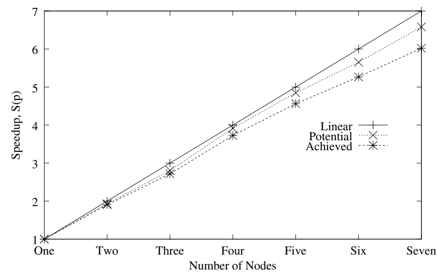
[16] S. Seshu and D. Freeman, ”The Diagnosis of Asynchronous Sequential Switching Systems,” *IRE Trans. Electronic Computing*, Vol. 11, pp. 459-465, 1962.

[17] J. Wolf, L. Kaufman, R. Klenke, J. Aylor, and R. Waxman, “An Analysis of Fault Partitioned Parallel Test Generation,” *IEEE Trans. on CAD*, Vol. 15, No. 5, pp. 517-534, 1996.

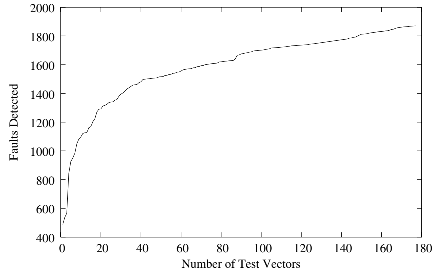
IEEE COMPUTER SOCIETY



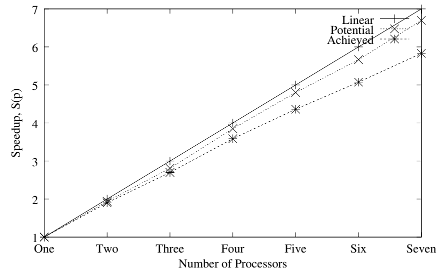
(a) C1355 Results



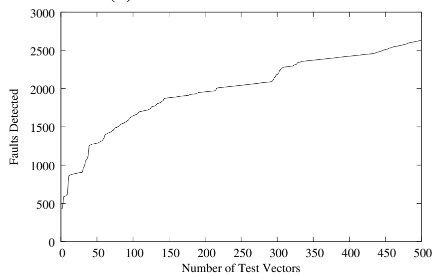
(b) C1355 Speedup



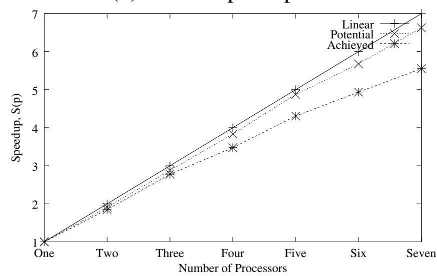
(c) C1980 Results



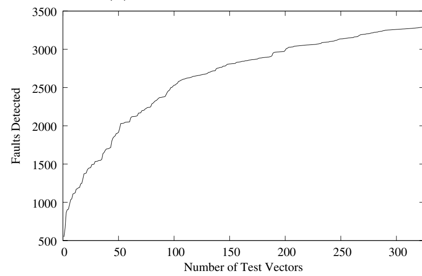
(d) C1908 Speedup



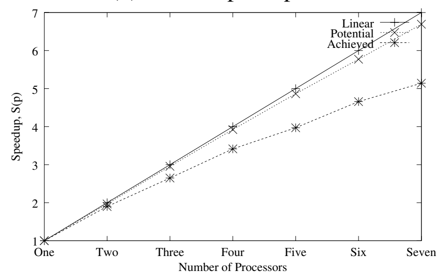
(c) C2670 Results



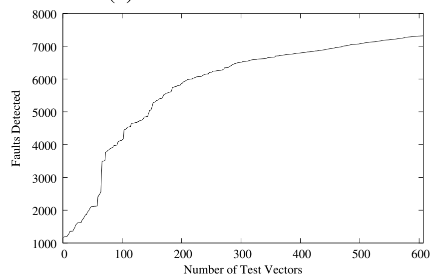
(d) C2670 Speedup



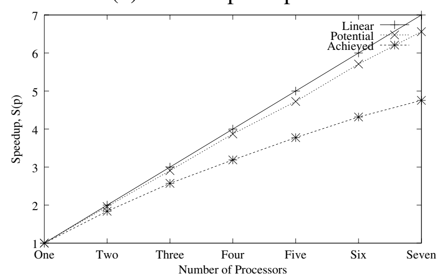
(c) C3540 Results



(d) C3540 Speedup



(c) C7552 Results



(d) C7552 Speedup

Figure 5. Simulation results for the ISCAS combinational benchmarks

1098-8068/05 $20.00 © 2005 **IEEE**