Diagnostic Fault Simulation for Synchronous Sequential Circuits

Shung-Chih Chen and Jer Min Jou, *Member, IEEE*

***Abstract*—In this paper, a time and memory-efficient diagnostic fault simulator for sequential circuits is first presented. A distributed diagnostic fault simulator is then presented based on the sequential algorithm to improve the speed of the diagnostic process. In the sequential diagnostic fault simulator, the number of fault-pair output response comparisons has been minimized by using an indistinguishability fault list that stores the faults that are indistinguishable from each fault. Due to the symmetrical relationship of the fault-pair distinguishability, fault list sizes are reduced. Therefore, the different diagnostic measures of a given test set can be generated very quickly using a small amount of memory. To further speed up the process of finding the indistinguishable fault list for each fault, a distributed approach is proposed and developed. The major idea for this approach is that each processor constructs the indistinguishable fault lists for a certain percentage of faults only. Experimental results show that the sequential diagnostic fault simulator runs faster and uses less memory than a previously developed one and that the distributed algorithm even achieves superlinear speedup for a very large sequential benchmark circuit, s35932. To the authors' knowledge, no distributed diagnostic fault simulation system for sequential circuits has been proposed before.**

***Index Terms*—Diagnostic fault simulation, diagnostic measure, indistinguishable fault list.**

I. Introduction

THE GOAL of fault diagnosis is to identify the causes of circuit failures. One diagnosis strategy uses diagnostic fault simulation to provide a set of potential fault locations, followed by visual inspection or electron-beam probing of the faulty circuit to identify the cause of failure. As a result of the rapid progress in very large scale integration (VLSI) technology, millions of gates can easily be packed into an integrated circuit (IC). Diagnosing device failures with so many potential fault locations becomes a very time-consuming job. Therefore, developing an efficient diagnostic fault simulator is an urgent need.

Besides providing a set of potential fault locations, diagnostic fault simulation can also be used to determine the diagnostic capability of a test set. Several measures have been proposed to measure the diagnostic capability. They are diagnostic power (DP) [1], diagnostic resolution (DR) [1], equivalence class (EC) sizes [2], and indistinguishability class sizes [3]. The diagnostic fault simulator developed by Rudnick *et al.* [3], which was extended from the PROOFS fault simulator [4], used a two-dimensional (2-D) distinguishability matrix to record the distinguishability between each pair of faults, and two procedures were proposed to update the distinguishability matrix after each test pattern was simulated. After all the test patterns were simulated, the indistinguishability classes were found and reported by size based on the distinguishability matrix. Since the time and memory complexities of both procedures are $O(n^2)$, where n is the number of faults, a large memory and a long execution time are needed to obtain the indistinguishability classes for very large sequential circuits.

In this paper, we first present a sequential diagnostic fault simulator which minimizes the number of fault-pair output response comparisons by using an indistinguishable fault list to store the faults that are indistinguishable from each fault. By applying the symmetrical relationship of the fault-pair distinguishability, we can reduce the sizes of the fault lists maintained. Experimental results show that this method uses less memory but achieves a significant speedup as compared to the methods for updating the distinguishability matrix [3]. However, it still spends a lot of central processing unit (CPU) time on very large circuits. To further speed up the diagnostic process, a distributed diagnostic fault simulator is then developed and presented. To the authors' knowledge, no distributed diagnostic fault simulator for sequential circuits has been reported before. Experimental results show that this distributed diagnostic fault simulator indeed shortens the diagnostic time greatly, especially for very large circuits.

The rest of the paper is organized as follows. In Section II, the measures used to measure the diagnostic capabilities of a given test set are described. In Section III, we survey previous methods for updating the distinguishability matrix and identify their problems. Sequential diagnostic fault simulation is the topic of Section IV. Section IV-A describes how to minimize the number of fault-pair output response comparisons. Section IV-B shows how to extend diagnostic fault simulation to diagnose a single stuck-at device failure [3]. Section IV-C shows the experimental results for many ISCAS89 sequential benchmark circuits [5] and compares them with the results of previous approaches. Section V demonstrates a distributed diagnostic fault simulation system connected via ethernet. In this system, a workstation, which works as a HOST, activates other workstations, which work as SLAVE's, to obtain the indistinguishable faults of all the detectable faults of a circuit under a given test set. The algorithms for the HOST and each SLAVE are described in Section V-A and V-B, respectively. An example is shown in Section V-C to explain the work

Manuscript received May 26, 1995; revised March 30, 1997. This paper was recommended by Associate Editor, W. K. Fuchs.

The authors are with the Electrical Engineering Department, National Cheng Kung University, Tainan, 70101 Taiwan, R.O.C.

Publisher Item Identifier S 0278-0070(97)04325-X.

done by the HOST and by each SLAVE. Section V-D shows the efficiency of this distributed system by providing some experimental results. Finally, conclusions are made in Section VI.

II. Diagnostic Measures

In this section, various diagnostic measures are introduced. *Diagnostic resolution (DR)* is the fraction of fault pairs distinguished [1]

$$\text{DR} = \frac{\text{number of fault pairs distinguished}}{\text{total number of fault pairs}}.$$

Diagnostic power (DP) is the fraction of fully distinguishable faults [1]. A fault is fully distinguishable if it can be distinguished by the test set from all other faults

$$\text{DP} = \frac{\text{number of fully distinguishable faults}}{\text{total number of faults}}.$$

The above two measures may not adequately reveal the diagnostic capabilities of a test set since they are single numbers. To provide more accurate diagnostic information, Kubiak *et al*. proposed a third measure, *Equivalence Class (EC)* [2], which identifies sets of equivalent faults with respect to a given test set for a circuit with binary response, i.e., a combinational circuit or a sequential circuit with a reset state. Besides reporting DR and DP, which can be computed from EC, the number of equivalence classes by size is also reported. Therefore, this measure reveals the diagnostic capabilities more completely.

In sequential circuits, since the initial value for each flip-flop (FF) is unknown (X), a good or faulty circuit can produce a value of 0, 1, or X on any primary output (PO). Due to the uncertainty of X, values 0 and 1 are thought of as indistinguishable from value X. Therefore, *equivalence classes* cannot be directly applied to sequential circuits with unknown initial state at each FF. Rudnick *et al*. [3] introduced an alternative measure, *indistinguishability class*, to express the diagnostic capabilities of a test set for sequential circuits with an unknown initial state at each FF. An *indistinguishability class* is a set of faults such that every pair of faults in the class is indistinguishable. Two faults f and g are indistinguishable with respect to a test set, if for every PO, either the binary values for f and g are equal, or one of the two values is unknown X. Circuits containing faults which can only be potentially detected, i.e., faults that have unknown values at PO's while other faulty circuits have known zero or one values, will have some faults included in more than one indistinguishability class. Consequently, the indistinguishability classes may not be disjoint [3].

When the sizes of indistinguishability classes for a given test set are found, the diagnostic measures, DP and DR, can then be computed. The number of fully distinguishable faults is the number of indistinguishability classes with size one. The number of distinguished fault pairs can be computed from the number of indistinguishable fault pairs since DR is equal to

$$\text{DR} = 1 - \frac{\text{number of distinguishable fault pairs}}{\text{total number of fault pairs}}.$$

III. Previous Work

Previous approaches used an n-by-n distinguishability matrix to represent the distinguishability information between each pair of faults under a given test set [3], where n is the number of faults. If two faults are distinguishable, the corresponding entries in the distinguishability matrix are set to one. Two ways were proposed to update the distinguishability matrix after every test pattern was simulated. The first way uses a 2-D array to store every PO value of all the faulty circuits after a test pattern is simulated. Then, each faulty circuit is compared to 32 other faulty circuits at a time. Since only half of the matrix need to be constructed because it is symmetric, the number of comparisons for each test pattern is $(n * m * \lceil n/32 \rceil)/2$, where m is the number of PO's. After each comparison, the corresponding entries in the distinguishability matrix for the distinguished fault pairs are set. The second way uses two buckets, the one and zero buckets, to store the faulty circuits with values of one and zero, respectively, at each PO. The distinguishability matrix is updated by traversing the bucket lists. Every fault in a zero bucket is distinguishable from every fault in the corresponding one bucket. Consequently, for every PO under every test pattern, the traversal complexity is $O(b_0 * b_1)$, where b_0 and b_1 are the number of faults in the two buckets.

The above methods have two problems. The first is that the memory requirement for the n-by-n distinguishability matrix is quite large. The memory complexity is $O(n^2)$. When the circuit under diagnosis is large, the circuit will be partitioned into smaller submodules, and the diagnostic capabilities will be reported separately. The second problem is that many operations are not necessary. For the first method, the output response of each fault is compared to that of all other faults. In fact, it is necessary only to compare its output response with those of the faults that are still indistinguishable from it. That is, the first method wastes a lot of time in comparing the PO values of distinguished fault pairs. For the second method, since every PO has zero and one buckets, a faulty circuit might have known zero or one values at different PO's and, therefore, appear in more than one bucket. Consequently, each distinguished fault pair, with respect to a test pattern, might be traversed more than one time. In the following section, we will propose our diagnostic simulation method which eliminates the drawbacks described.

IV. Proposed Diagnostic Method

As described in the previous section, the methods for updating the distinguishability matrix wasted some CPU time in comparing the values of all the PO's of the faulty circuits that had already been distinguished by a sequence of test patterns. In this section, we describe how our method minimizes the number of fault-pair output response comparisons and demonstrate it by using a simple example. An extension for diagnosing single stuck-at device failures is also made and described.

A. Minimization of the Number of Comparisons

If we want to minimize the number of comparisons among all the pairs of faulty circuits, we can compare the output

response of each fault with only those faults that are still indistinguishable from it. Consequently, a list must be associated with every fault to store its indistinguishable faults. We call such a list an indistinguishable fault list (IFL). Before fault diagnosis begins, every fault is indistinguishable from all other faults. To construct each fault's IFL, which contains all other faults, in advance is impractical since a very large memory would be necessary for very large circuits. To reduce the memory usage, our method does not construct IFL's for faults until they are detected, i.e., the IFL for each undetected fault is empty. Thus, each fault requires a detection flag to indicate its detection. Once a fault becomes detected, its detection flag is set, and the faults which are indistinguishable from it are put into its IFL. After each subsequent test pattern is simulated, each detected fault needs only to have its output response compared with those of the faults in its IFL, instead of all other faults. After each comparison, any newly distinguished faults are removed from its IFL. Consequently, the length of the IFL for each detected fault gets shorter while the number of comparisons gets smaller as the diagnostic process proceeds.

For each test pattern, there may be newly detected faults which are distinguishable from the good circuit response. Two kinds of faults are put into the IFL of each newly detected fault. The first one includes the detected faults that are indistinguishable from it so far. This is done by traversing the IFL of each detected fault to check whether it is in this IFL or not. If yes, then the detected fault is put into its IFL. The second kind consists of the undetected faults whose output responses are indistinguishable from its response with respect to the current test pattern. This is determined by comparing its output response with those of undetected faults.

In constructing the IFL for a newly detected fault, not all the faults that are indistinguishable from it are inserted into its IFL. This is because the distinguishability relation is symmetrical [3]. That is, if a faulty circuit A is distinguishable (indistinguishable) from a faulty circuit B, then the faulty circuit B is distinguishable (indistinguishable) from the faulty circuit A. To minimize the number of fault-pair output response comparisons in our diagnostic fault simulator, the output responses for each fault pair are compared once only, instead of twice. To do that, each fault is indexed with a different number, called a fault number, before starting simulation. By applying the symmetrical property, the IFL for a newly detected fault contains 1) the detected faults which are indistinguishable from it and whose fault numbers are larger than its fault number and 2) the undetected faults that are indistinguishable from it. The algorithm for maintaining the previously constructed IFL's and constructing the IFL's for newly detected faults after the simulation of each test pattern is shown in Fig. 1.

The *distinguishable* function shown in Fig. 1 is a boolean function to determine whether the output responses of a pair of faults are distinguishable or not. If yes, then the function returns a value of one. Otherwise, it returns a value of zero. A pair of faulty circuits is said to be distinguishable with respect to the current test pattern if any of their PO values is distinguishable. We describe how to determine the distinguishability of a PO value of two faulty circuits.

```
For each previously detected faulty circuit f
    For each faulty circuit g in the IFL of f
        If ( distinguishable( f, g ) )
            Remove g from the IFL of f.
For each undetected faulty circuit h
    If ( distinguishable( h, G ) )        /* G is the good circuit */
        For each detected faulty circuit f        /* h is a newly detected fault */
            If ( h is in the IFL of f and has smaller fault number )
                Insert f into the IFL of h and remove h from the IFL of f.
        For each undetected faulty circuit g
            If ( ! distinguishable( h , g ) )
                Insert g into the IFL of h.
        Move h from the undetected fault set to the detected fault set.
```

Fig. 1. Algorithm for constructing and maintaining the IFL's.

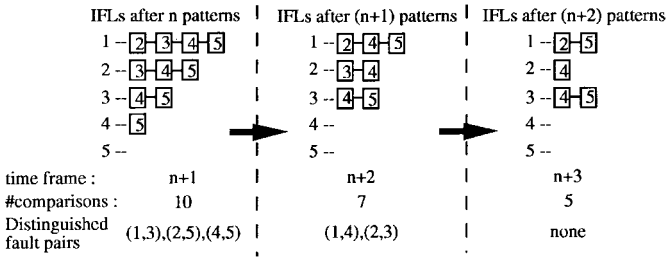

Fig. 2. Example illustrating the efficiency of our method.

A two-bit coding technique [4] has been used to represent the circuit values 0, 1, and X as (1, 0), (0, 1), and (0, 0), respectively. Let $V(f_i)$ be the value of the faulty circuit f at the ith PO after a test pattern is simulated and $(V0_{f_i}, V1_{f_i})$ be its corresponding coded value pair. Then the distinguishability between a pair of faulty circuits f and g at the ith PO is defined as

$$BD(V(f_i), V(g_i)) = (V0_{f_i} \bullet V1_{g_i}) \oplus (V1_{f_i} \bullet V0_{g_i}) \quad (1)$$

where $\bullet$ and $\oplus$ are the AND and XOR bit operations, respectively. If the computed value of $BD(V(f_i), V(g_i))$ is equal to one, then this pair of faulty circuits is distinguishable. Otherwise, they are indistinguishable at the ith PO with respect to the current test pattern. The extension to comparing 32 PO values is obvious and, therefore, omitted.

We use a simple example to explain the efficiency of our algorithm and the IFL data structure. Without loss of generality, we assume that after n test patterns are simulated, there are five detected faults, numbered from one to five, that are indistinguishable from each other in a circuit with five PO's. Fig. 2 shows the IFL's, the number of comparisons, and the assumed distinguished fault pairs for each of the three subsequent test patterns. In total, the number of comparisons is 22 for the three test patterns. For the first method proposed in [3], the number of comparisons is 30, ten comparisons for each test pattern. This small example shows the efficiency of our method.

After all the test patterns have been simulated, the IFL for each undetected fault is then constructed which includes the detected faults that are indistinguishable from it and all other undetected faults. Since we applied the symmetrical property of the indistinguishability relationship into our diagnostic simulator, the faults with smaller fault numbers which are indistinguishable from a detected fault are also inserted into the IFL of this detected fault. The sets of indistinguishable faults are then obtained by finding all cliques of maximal sizes in the graph created by the IFL's of all the faults [3]. Having these indistinguishability classes, the diagnostic capabilities, diagnostic power, and diagnostic resolution are then computed.

B. Diagnosing Device Failure

To diagnose device failures given the output response for every test vector, a mismatch score for each faulty circuit was used in [3] to reflect the number of times each faulty circuit was distinguished from the actual circuit at a PO. The mismatch score was computed as follows. When a faulty circuit had a known one or zero value at a PO which was different from the actual circuit value, the mismatch score was incremented by two. When the faulty circuit value was unknown, the mismatch score was incremented by one. Otherwise, the mismatch score was not incremented. Mismatch scores were not modified when the good circuit value was unknown at a given PO. Faults corresponding to faulty circuits with lowest mismatch scores were identified as candidate faults. This method allowed for approximate diagnosis of multiple stuck-at faults and nonstuck-at faults. However, the candidate faults it reported may be distinguishable from the actual circuit. According to the results reported in [3], all single faults were diagnosed correctly. However, for most of the diagnosed multiple faults, only one out of five double faults was identified correctly. For this reason, only diagnosis of single stuck-at fault is tried in our diagnostic fault simulator.

To diagnose the location of a single stuck-at device failure, a new candidate fault list is used to store the faults whose output responses are indistinguishable from that of the failing device, and each fault in this list has an indistinguishability score indicating the indistinguishability degree between it and the failing device. After the first test pattern is simulated, the candidate fault list is constructed. Meanwhile, the indistinguishability scores are computed as follows. For each PO, when the faulty value of a fault in this list is the same as that of the failing device, the indistinguishability score of this fault is not incremented. Otherwise, the score is incremented by one. After each subsequent test pattern is simulated, each fault in the candidate fault list has its output response compared with that of the failing circuit. If these are distinguishable, this fault is removed from the candidate fault list. Otherwise, its indistinguishability score is updated. After all the test patterns have been simulated, the faults remaining in the candidate fault list are the possible locations for the failing device.

C. Experimental Results

The proposed sequential diagnostic fault simulator has been designed, coded in C language, and run on a SUN station SPARC LX with 32 megabytes memory. To test its performance, several ISCAS89 sequential benchmark circuits [5] were run and compared with the results obtained in [3], which used a SUN station SPARC IPC with 24 megabytes memory. Note that the performance of SPARC LX is about two times that of a SPARC IPC. Both test sets were generated by the STG3 sequential circuit test generator [6]. Table I shows the characteristics of the circuits and the diagnostic results obtained for the STG3 test vectors. The comparison results are shown in Table II. Table III shows the sizes of the indistinguishability classes for each circuit under the given test set.

TABLE I
CIRCUIT CHARACTERISTICS AND DIAGNOSTIC RESULTS FOR STG3 TEST VECTORS

circuit	#gates	#PI	#PO	#FF	#fault	#test	fault cov.	DR	DP	Mem used
s298	119	3	6	14	308	162	85.7	94.1	0.0	72K
s344	160	9	11	15	342	91	96.1	96.7	0.0	77K
s400	164	3	6	21	424	1282	82.8	92.3	0.0	173K
s526	193	3	6	21	555	754	75.3	89.5	0.0	60K
s641	379	35	24	19	467	133	86.2	97.6	39.0	387K
s713	393	35	23	19	581	107	80.8	95.8	31.0	84K
s820	289	18	19	5	850	411	81.8	96.3	8.7	110K
s832	287	18	19	5	870	377	81.3	96.2	8.5	116K
s953	395	16	23	29	1079	16	7.7	14.9	1.3	128K
s1238	508	14	14	18	1355	349	94.6	99.7	81.2	154K
s1423	657	17	5	74	1515	36	24.4	42.5	1.7	213K
s1488	653	8	19	6	1486	590	92.5	99.1	3.9	250K
s1494	647	8	19	6	1506	469	91.1	98.6	3.7	253K
s5378	2779	35	49	179	4603	408	74.0	93.1	25.6	1M
s35932	16065	35	320	1728	39094	86	92.5	98.5	0.0	11M

TABLE II
COMPARISON RESULTS BETWEEN THE METHODS IN [3] AND OUR METHOD

circuit	CPU time(Seconds)			speedup factor	
	our method	Proc. 1 of [3]	Proc. 2 of [3]	Proc. 1 ‡‡‡ / ours*2	Proc. 2 ‡‡‡ / ours*2
s298	3.8	9.4	25.4	1.24	3.34
s344	3.5	9.1	21.9	1.30	3.13
s400	58.2	121.2	364.2	1.04	3.13
s526	48.9	100.2	301.8	1.02	3.08
s641	3.4	29.6	29.5	4.35	4.34
s713	3.7	33.4	30.3	4.51	4.09
s820	15.1	193.8	198.0	6.42	6.56
s832	14.1	190.2	190.8	6.74	6.77
s953	1.5	7.5	5.6	2.50	1.87
s1238	13.7	196.2	304.2	7.16	11.10
s1423	5.8	20.7	16.4	1.78	1.41
s1488	76.4	904.2	1992.0	5.92	13.04
s1494	68.0	741.0	1585.8	5.45	11.66
s5378	217.7	11520.0	5616.0	26.46	12.90
s35932	1208.1+	45108.0+	68724.0+	18.67	28.44
s35932	16238.2‡‡	NA	NA	----	----

+ : CPU time for one-fifth faults
‡‡ : CPU time for all the faults
‡‡‡ : Our results were run on a SPARC LX, which has about twice speed of an IPC[3].

The results obtained by our method have been compared with those obtained by the two procedures proposed in [3]. Note that the speedup factors shown in Table II have been divided by two since the workstation used in this experiment is about two times faster than the one used in [3]. The results show that our method is more efficient than both procedures for every circuit. For the circuit s35932, our method achieves more than 18 times speedup for one-fifth of the original faults. No result for diagnosing all the faults of s35932 has been obtained by [3] since the memory was not sufficient to handle so many faults. Due to the efficient list structure used in our method, the results for processing all the faults of circuit s35932 have also been obtained by our method, as shown

TABLE III
SIZES OF THE INDISTINGUISHABILITY CLASSES

	number of indistinguishability classes by size									
circuit	2	3	4	5	6	7	8	9	10	> 10
s298	3	9	7	15	32	13	9	31	23	6(11),4(12),1(13),1(14), 2(15),1(18),1(44)
s344	15	6		12	95	89	3	3	10	4(11),1(12),1(13)
s400	25	18	9			3	20	9	8	28(11),55(12),12(13), 5(14),2(15),3(16),1(17), 2(20),1(59),1(73)
s526	6	14	4	5	3	2	15	19	18	6(11),9(12),25(13),24(14), 19(15),12(16),5(17),3(19), 1(20),1(23),1(25),1(52), 1(65),1(137)
s641	34	80	17	2	5	2	3			1(64)
s713	27	67	20	2	4	6	6		6	1(111)
s820	63	389	51	10	1	2		1		1(14),1(154)
s832	463	51	7	2	2		2			1(14),1(162)
s953	13	5		2	2	2		1	2	1(13),1(14),1(16),1(995)
s1238	80	6	1							1(72)
s1423	18	11	8	7	5	3	2	4	2	2(11),2(12),2(14),1(15), 1(16),1(17),1(18),1(22), 1(23),1(34),1(36),1(55), 1(1145)
s1488	109	90	943	58	10	3				1(110)
s1494	92	36	19	76	907	62	10	2	1	1(11),1(134)
s5378	534	319	122	91	43	31	15	15	10	10(11),5(12),3(13),4(14), 2(15),1(17),2(23),1(25), 1(27),1(50),1(61),1(71), 1(1196)
s35932	4945	8279	4118	2156	641	245	167	110	82	27(11),7(12),6(13),1(16), 1(4696)

in Table II. The memory required for each circuit under the given test set is shown in the last column of Table I. The results show that the memory usage for circuit s35932 is 11 megabytes. This proves that our method is not only fast but also memory-efficient.

The function for diagnosing a single stuck-at failure is also incorporated into our simulator. Besides the test patterns, the output response of the randomly selected failing device for each test pattern is also used as input in our diagnostic fault simulator. The diagnostic results are shown in Table IV. The faults with lowest indistinguishability scores are reported as the possible locations of the failing device. Note that the candidate faults and the selected failing device are in the same indistinguishability class(es). All single stuck-at faults are diagnosed correctly. Including the diagnosis process into our fault simulator causes a small overhead in execution time.

V. DISTRIBUTED DIAGNOSTIC FAULT SIMULATION

Although the above sequential diagnostic fault simulation approach is very efficient as compared to the previous one [3], it still takes 4.4 hours of CPU time for a very large circuit, s35932, to complete the whole diagnostic process. It would be useful to look for a new method that could further reduce the time spent in diagnostic fault simulation.

One way of improving the speed of fault simulation is to partition either the gates [7], the test patterns [8], or the faults [9] and then to run on a distributed workstation system. Although a diagnostic fault simulator is usually extended from a fault simulation algorithm, the above partitioning approaches cannot be applied to diagnostic fault simulation since each fault's indistinguishable fault list cannot be determined in advance. For this reason, any prior partition might lead to reporting incorrect diagnostic capabilities.

TABLE IV
DIAGNOSTIC RESULTS FOR SINGLE STUCK-AT FAULTS

circuit	Selected Fault	Fault in Indist. Class(es)	Diagnosed Faults (indistinguishability score)	Exec. Time
s298	90	2	90(0), 266(960)	3.90
	37	4	37(0), 56(0), 44(931)	3.90
	2	5	2(0), 18(864), 3(883)	3.93
s526	279	2	279(0), 469(4512)	49.75
	24	3	24(0), 37(4128), 469(4508)	49.93
	273	5	273(0), 463(0), 510(0)	50.10
s713	1	2	1(0), 442(0)	3.73
	8	4	8(0), 346(0), 180(104)	3.73
	3	5	3(0), 191(0), 212(0), 265(0)	3.75
s953	431	2	431(0), 443(0)	1.51
	17	3	17(0), 447(1), 551(1)	1.51
	279	5	279(0), 281(0), 623(9)	1.51
s1423	49	2, 22	49(0), 50(0), 51(0), 59(0)	5.87
	131	4	131(0),132(0),135(0),133(14)	5.85
	829	3	829(0), 259(17), 1066(17)	5.85
s5378	0	2	0(0), 2(0)	219.85
	60	3	60(0), 1120(0), 3266(9606)	220.03
	198	4	198(0), 216(0), 3864(468)	220.08

One of the objectives for the diagnostic fault simulation is to find the indistinguishable fault list for each fault. In fact, only the detectable faults' IFL's need to be obtained during the diagnostic process since the undetectable faults' IFL's can be obtained from those of the detectable faults. This is because the fault-pair distinguishability has the symmetrical relation described in Section IV-A. Therefore, the objective for a distributed diagnostic fault simulation system is that each computation node in the distributed system constructs the IFL's for some detectable faults. Overall, the computation nodes in this distributed system construct the IFL's of all the detectable faults.

The problems now are 1) how to determine which faults are detectable and 2) how to distribute them to each computation node evenly, so that all the computation nodes complete their own jobs in the same time, as quickly as possible. The first problem can be solved by using a fault-dropping fault simulator. Given a test set, a fault-dropping fault simulator can determine which faults are first detected by each test pattern. With the development of efficient fault-dropping fault simulators [10]–[12], the time spent on fault-dropping fault simulation can be small as compared to that spent on the whole diagnostic process. The second problem is more difficult to solve since the faults in the IFL of each detectable fault for each circuit cannot be determined in advance. For this reason, we currently allow users to adjust the percentage of faults, which are detectable with respect to a given test set, whose IFL's are going to be constructed by each computation node, so as to try to find the best load distribution.

Based on the above descriptions, we develop a distributed diagnostic fault simulator for sequential circuits. This distributed network system is connected via ethernet. All the computation nodes are SUN SPARC II workstations, in which one is called HOST and the others are called SLAVE's. The algorithms for the HOST and each SLAVE are listed in Figs. 3 and 4, respectively, and described separately. Before continuing, we give some notations:

M number of SLAVE's to be used in the distributed system;

S_i ith SLAVE, where i is from 1 to M;

```
Accept user-defined parameters.
Read the circuit to be processed.
Do fault-dropping fault simulation.
Activate SLAVEs.
While( test pattern number < TF_H )
    Do fault simulation.
while( test patterns not exhausted )
    Do fault simulation.
    Construct new IFLs and maintain old IFLs.
Read the results saved by SLAVEs.
Construct the IFL for each undetectable fault.
Find the number of indistinguishability classes by size.
Save results.
```

Fig. 3. Algorithm for the HOST.

```
Read the circuit to be processed.
Preprocessing.
while( test pattern number < TF_i )
    Do fault simulation.
while( test pattern number < TL_i )
    Do fault simulation.
    Construct and maintain the IFL of each detected fault.
Remove the distinguished faults not assigned to i from the fault list.
while( test pattern is not exhausted )
    Do fault simulation.
    Maintain the IFL of each detected fault.
    Remove the newly distinguished faults not assigned to i from the fault list.
Save results.
```

Fig. 4. Algorithm for SLAVE S_i.

F_i percentage of faults, which are detectable with respect to the given test set, whose IFL's are going to be constructed by S_i;
P summation of all F_is;
G_i group of faults whose IFL's are constructed by S_i;
G_H fault list in the HOST;
TF_i test pattern at which the first IFL is constructed in S_i;
TL_i test pattern at which the last IFL is constructed in S_i;
TF_H test pattern at which the first IFL is constructed in the HOST.

A. Algorithm for the HOST

The parameters determined by the user include M and all the F_is. The fault-dropping fault simulation algorithm, which we developed [12], is performed until P is met. It generates a file containing the faults detected by each test pattern. This file is read by each SLAVE to determine which faults' IFL's it is going to construct, according to all the F_is. This process also determines the value of TF_H. All the SLAVE's are activated by the HOST using the "rsh" system call [13].

During the fault-dropping fault simulation, the faults whose IFL's are going to be constructed by all the SLAVE's are removed from the fault list. Consequently, the HOST only deals with the undropped faults. Since no IFL in HOST is constructed until TF_H is reached, the test patterns before TF_H only do fault simulation to record the faulty states for the faults in the fault list. For each of the subsequent test patterns, new IFL's for the newly detected faults are constructed, and old IFL's for the previously detected faults are maintained. The algorithm for constructing new IFL's and maintaining old IFL's was described in Section IV-A and listed in Fig. 1. After all the test patterns are processed, the HOST sometimes has to wait until all the SLAVE's construct their IFL's before the sizes of the indistinguishability classes are obtained.

B. Algorithm for Each SLAVE

The detectable faults stored in the file generated by the fault-dropping fault simulator are partitioned into M groups based on F_is, according to their detection sequence. The IFL's of the detectable faults in group i, i.e., G_i, are to be constructed in S_i.

For S_i, the detectable faults in fault groups G_1 to G_{i-1} are not included in its fault list. This is because the fault-pair distinguishability relation is symmetrical. If a detectable fault in fault group G_i is indistinguishable from a fault, say f, in any of fault groups G_1 to G_{i-1}, then it must be in the IFL of f. We can finally put fault f into the IFL of this detectable fault. Consequently, the detectable faults in fault groups G_1 to G_{i-1} can be removed from the fault list in S_i. This is done in the *preprocessing* function. This function also determines the values of TF_i and TL_i.

Since no IFL is constructed until TF_i is reached, only fault simulation is done for the test patterns before TF_i. As for the test patterns between TF_i and TL_i, they need to construct the IFL for each newly detected fault and maintain the IFL for each previously detected fault. After the IFL's for all the faults in G_i have been constructed, only the faults in G_i or the faults in these constructed IFL's require output response comparison. Therefore, the faults which are neither in G_i nor in these constructed IFL's are dropped from the fault list. Usually, the dropped are a large percentage of the faults in the fault list. For each of the subsequent test patterns, any newly distinguished fault is removed from the faults list. Consequently, the fault simulation time gets shorter due to the decreasing length of the fault list.

After all the test patterns are processed, the faults in fault group G_i and the faults in their IFL's are saved. When the HOST is constructing the IFL's of all the detectable faults, some extra faults should be inserted into some other detectable faults' IFL's. For example, suppose that a detectable fault f is in fault group G_2 and a detectable fault g in its IFL is in fault group G_4. In this situation, fault f does not appear in the IFL of fault g since fault f is excluded from the initial fault list of S_4. To correctly get the diagnostic capabilities, fault f is inserted into the IFL of fault g.

C. An Illustrative Example

We use an example, as shown in Fig. 5, to illustrate our distributed system. Suppose that there are n faults in a circuit, two SLAVE's are used, and t test patterns are simulated. Without loss of generality, we index all the faults in accordance with their detection sequence. We further assume that fault group G_1 contains the faults f_1 to f_k, fault group G_2 contains the faults f_{k+1} to f_m, and the rest of the faults constitute the fault list of the HOST, i.e., G_H. These values can be determined

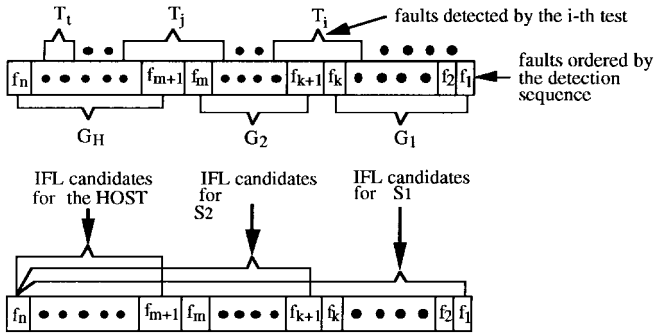

Fig. 5. Example illustrating our distributed diagnostic fault simulator.

after the user-defined parameters, i.e., M and F_is, are assigned by the user. With these values, we use S_2 to illustrate the algorithm for a SLAVE. In the *preprocessing* function, faults f_1 to f_k are removed from the fault list, and TF_2 and TL_2 are determined, which are equal to T_i and T_j, respectively. Consequently, this fault list contains f_{k+1} to f_n. For the test patterns T_1 to T_{i-1}, only fault simulation is performed to record the faulty state of each fault in the fault list, since no fault in this fault list can be detected. For test patterns T_i to T_j, there may be newly detected faults for each test pattern, so the fault-pair output response comparison is also performed to construct new IFL's and to maintain old IFL's. For the test patterns after T_j, since the IFL's for all the faults in fault group G_2 are constructed, the faults that are neither in fault group G_2 nor in the IFL's of faults f_{k+1} to f_m are removed from the fault list since they are distinguishable from all faults assigned to processors two. In this way, the fault simulation time for these test patterns can be reduced due to the reduction of the fault list.

One of the main advantages for our distributed diagnostic fault simulator is that it introduces a very low network communication load. After reviewing our algorithm, we find that only when each machine is reading the circuit for fault diagnosis, when SLAVE's are saving their results, or when the HOST is reading these results is it necessary to transfer data within the network. In other words, when our distributed diagnostic fault simulator is running in the network system, it hardly influences the jobs on other machines that need to use the network to transfer data.

D. Experimental Results for Distributed Diagnostic Fault Simulation

The proposed distributed diagnostic fault simulator for sequential circuits has been designed, coded in C language, and run on a SUN SPARC network system. The HOST and SLAVE machines are SUN station SPARC II workstations with 32 megabytes memory. To test the efficiency of this approach, some ISCAS89 sequential benchmark circuits [5] were used. The circuits selected took much longer CPU time to complete their diagnostic process than other circuits.

As described previously, it is very difficult to find a way for all the circuits to distribute their load to each computation node evenly. At the moment, we allow users to adjust the percentage of faults F_is whose IFL's are to be constructed by each SLAVE. After making several trials for each circuit, the best load distributions we have found for each case under the given test sets, and the experimental results are summarized in Tables V and VI. Up to four SLAVE's were used in this experiment.

TABLE V
LOAD AND CPU TIME (IN SECONDS) FOR EACH MACHINE UNDER THE BEST DISTRIBUTION OF EACH CASE

circuit		Faults dealt with by each machine (%)				CPU time for each machine(sec)			
		1H+1S	1H+2S	1H+3S	1H+4S	1H+1S	1H+2S	1H+3S	1H+4S
s400	F_is	33	28 12	25 9 16	21 9 7 21	31.2	23.4 23.7	17.8 18.2 17.6	15.3 15.6 16.1 15.9
	H	67	60	50	42	31.4	23.2	18.6	15.9
s526	F_is	27	21 16	18 9 14	18 6 9 10	24.9	19.1 19.5	15.7 16.0 15.4	11.3 12.9 12.4 12.5
	H	73	63	59	57	25.0	19.0	13.8	12.0
s1488	F_is	31	20 27	17 15 22	13 12 16 19	35.2	27.1 26.3	22.2 22.1 22.3	19.6 19.1 19.1 19.0
	H	69	53	46	40	35.5	26.8	22.5	19.8
s1494	F_is	30	22 25	18 14 22	13 14 13 17	28.4	21.2 21.5	18.2 18.1 18.2	14.7 15.8 16.1 15.9
	H	70	53	46	43	28.8	21.7	17.9	16.0
s5378	F_is	34	24 25	15 21 20	14 15 16 17	107.1	77.8 82.3	66.4 64.8 66.5	55.1 56.1 57.4 56.8
	H	66	51	44	38	110.5	79.6	65.9	56.2
s35932	F_is	34	26 24	21 13 24	16 12 15 18	3859.6	2207.2 2111.7	1491.5 1419.4 1499.2	1233.2 1235.6 1186.5 1053.7
	H	66	50	42	39	3611.5	2081.5	1510.9	1140.4

TABLE VI
CPU TIME (IN SECONDS) AND SPEEDUP FOR THE MOST TIME-CONSUMING MACHINE OF EACH CASE

circuit		1H	1H+1S	1H+2S	1H+3S	1H+4S	speedup factor			
							1H+1S	1H+2S	1H+3S	1H+4S
s400	(1)	54.9	32.4	24.7	19.6	17.1	1.69	2.22	2.80	3.21
	(2)	53.9	31.4	23.7	18.6	16.1	1.72	2.27	2.90	3.35
s526	(1)	43.9	26.7	21.2	17.7	14.6	1.64	2.07	2.48	3.01
	(2)	42.2	25.0	19.5	16.0	12.9	1.69	2.16	2.64	3.27
s1488	(1)	74.0	41.4	33.0	28.4	25.7	1.79	2.24	2.61	2.88
	(2)	68.1	35.5	27.1	22.5	19.8	1.92	2.51	3.03	3.44
s1494	(1)	66.1	40.0	32.9	29.4	27.3	1.65	2.01	2.25	2.42
	(2)	54.9	28.8	21.7	18.2	16.1	1.91	2.53	3.02	3.41
s5378	(1)	225.9	116.3	88.1	72.3	63.2	1.94	2.56	3.12	3.57
	(2)	220.1	110.5	82.3	66.5	57.4	1.99	2.67	3.31	3.83
s35932	(1)	13257.8	5249.6	3597.2	2900.9	2625.6	2.53	3.69	4.57	5.05
	(2)	11867.8	3859.6	2207.2	1510.9	1235.6	3.07	5.38	7.85	9.60

(1) : Total CPU time.
(2) : CPU time excluding the time for finding the indistinguishability classes.

Table V shows the loads and CPU time of each machine under the best distribution of each case we have found. The time for finding the sizes of indistinguishability classes is not included. Based on the results shown in this table, we find that the variation in the load dealt with in each machine is large for each case. This confirms our statement that finding a best load distribution is a very difficult job.

The CPU times, with and without including the time for finding the indistinguishability classes, and their speedup factors for the most time-consuming machine of each case are summarized in Table VI. The data shown in the third column was obtained by using the HOST machine only. For the medium and small size circuits, these two speedup factors vary slightly since finding the sizes of indistinguishability classes only occupies a very small percentage of the total diagnostic fault simulation time. For the very large circuit

s35932, since the number of faults in it is very large, finding the sizes of indistinguishability classes takes a great deal of CPU time, even longer than the CPU time spent on diagnostic fault simulation when four SLAVE's are used. Therefore, the difference in speedup factors increases as the number of SLAVE's increases.

One more important observation for the circuit s35932 is that the speedup factor is greater than the number of machines used. In other words, for this circuit, our distributed diagnostic fault simulator achieves superlinear speedup results. To investigate the reason why superlinear speedup can be obtained, we have rerun our sequential and distributed diagnostic fault simulators to get some analytic data for circuits s5378 and s35932. Fig. 6 shows the results obtained from the sequential diagnostic fault simulator for different percentages of faults diagnosed. The results show that the time complexities of circuits s5378 and s35932 are $O(n)$ and $O(n^2)$, respectively, under the given test sets. Table VII shows the percentages of faults, tests, and CPU time for each phase in the most time-consuming machine for the cases shown in Table V. The distributed diagnostic fault simulation has been separated into four phases: a) fault simulation phase, which does fault simulation only, b) IFL construction and maintenance phase, which does fault simulation, constructs IFL's for newly detected faults, and maintains constructed IFLs, c) IFL maintenance phase, which does fault simulation and maintains constructed IFLs, d) indistinguishability class phase, which finds the indistinguishability classes. Note that the HOST machine does not have IFL maintenance phase and that SLAVE S_1 does not have the fault simulation phase. From Table VII, we find that most of the test patterns in each case only deal with a small percentage of faults. We also find that a great many distinguished faults have been removed from the fault list for further diagnosing. Based on the results shown in Fig. 6 and Table VII, we can understand why our distributed diagnostic fault simulator obtains superlinear speedup for circuit s35932. The Table also shows that the fourth phase occupies a small percentage of CPU time for circuit s5378 which indicates that the diagnostic fault simulation time can still be shortened by further increasing the number of computer machines. For circuit s35932, the fourth phase occupies more than 50 percent of the CPU time when four SLAVE's are used. That is, whenever the time spent in finding the indistinguishability classes is large, further increasing the number of computer machines can get little improvement in shortening the diagnostic fault simulation time. Is it possible to reduce the time spent in finding indistinguishability classes? One possible solution is to find a graph partitioning method to partition the graph generated by all the IFL's into subgraphs, and the indistinguishability classes of each subgraph is obtained by one computer machine in a distributed system. In this way, the time spent in finding indistinguishability classes can be reduced, and therefore, the time spent in diagnostic fault simulation can be significantly shortened by further increasing the number of computer machines. Min-cut graph partitioning [14] is perhaps an appropriate solution for developing a distributed algorithm to find the indistinguishability classes via a network system.

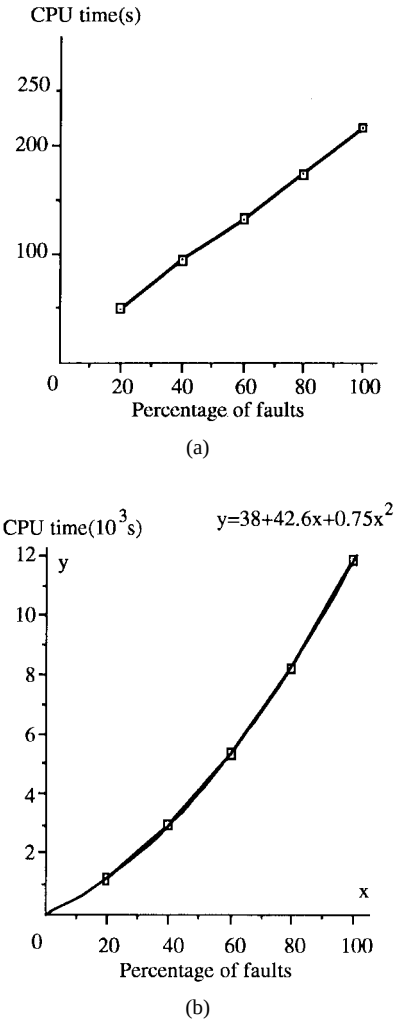

Fig. 6. Diagnostic time for different percentage of faults. (a) s5378. (b) s35932.

TABLE VII
PERCENTAGES OF FAULTS, TESTS, AND CPU TIME FOR EACH PHASE OF THE DIAGNOSTIC FAULT SIMULATION IN THE MOST TIME-CONSUMING MACHINE OF EACH CASE SHOWN IN TABLE V

		s5378				s35932			
		1H+1S	1H+2S	1H+3S	1H+4S	1H+1S	1H+2S	1H+3S	1H+4S
fault (%)	(3)	65.5	76.0	64.0	71.0	100.0	100.0	40.6	83.2
	(4)	65.5	76.0	64.0	71.0	100.0	100.0	40.6	83.2
	(5)	0.0	26.5	20.9	17.3	35.0	27.0	0.0	12.6
test (%)	(3)	2.2	1.2	3.0	1.7	0.0	0.0	33.7	3.5
	(4)	97.8	8.8	10.0	5.9	11.6	10.5	66.3	8.1
	(5)	0.0	90.0	87.0	92.4	88.4	89.5	0.0	88.4
CPU time (%)	(3)	1.5	1.4	3.1	2.6	0.1	0.1	1.4	0.4
	(4)	93.4	13.1	12.7	10.6	24.0	19.4	50.3	24.0
	(5)	0.0	78.7	76.1	77.4	49.8	42.7	0.0	24.1
	(6)	5.1	6.8	8.1	9.4	26.1	37.8	48.3	51.5

(3): Fault simulation.
(4): Fault simulation combined with constructing and maintaining IFLs.
(5): Fault simulation combinedwith maintaining IFLs.
(6): Finding the indistinguishability classes.

From the results obtained above, we find that it is very difficult to find a best load distribution for all the circuits. We also find that our system is inconvenient for users since every best case must go through several trials. Therefore, we heuristically find an easy way to determine the load for each computation node. Based on the best load distri-

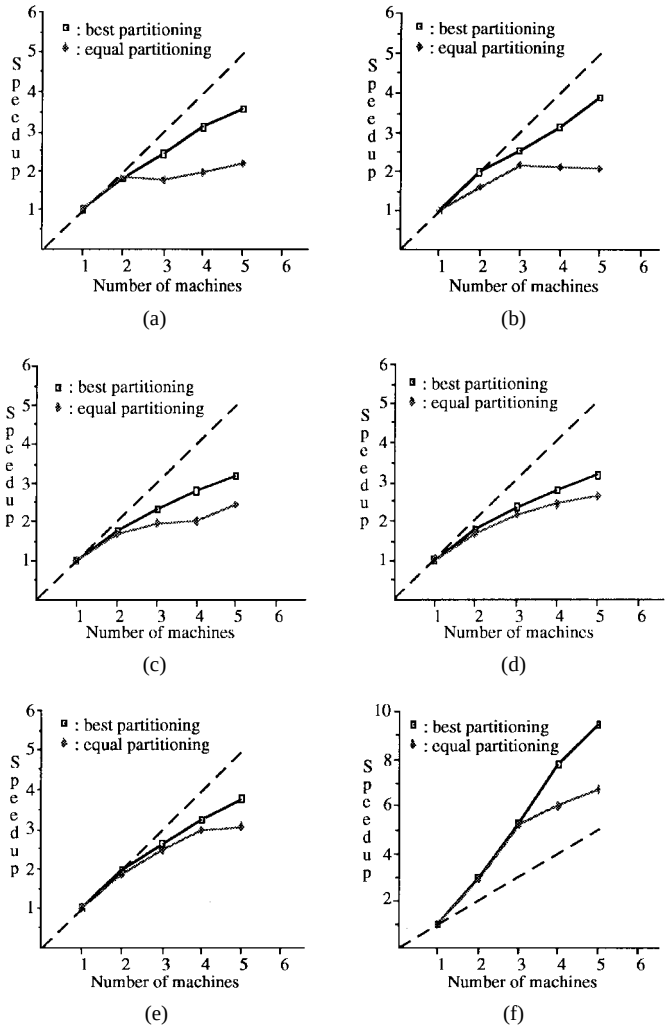

Fig. 7. Performance comparison for different load distribution approaches. (a) s400, (b) s526, (c) s1488, (d) s1494, (e) s5378, (f) s35932.

butions obtained above, we find that the performance for the HOST is about two times that of each SLAVE. Therefore, we assume that each SLAVE deals with the same percentage of faults and that the HOST deals with twice the number of faults that each SLAVE deals with. For example, if two SLAVE's are used, then the HOST deals with 50 percent of the faults, and each SLAVE deals with 25 percent of the faults. We call this equal partitioning. Under this condition, we did another experiment to show the CPU time for each computation node in Table VIII. From these results, we found that, for some cases, the most time-consuming machine spends four times as much time as the least time-consuming one. The comparisons of the speedup factors, without including the time spent for finding indistinguishability classes between the best partitioning and equal partitioning, are summarized in Fig. 7. From these figures, we find that the variation in speedup factors between these two partitioning approaches get larger as the number of SLAVE's used increases. Obviously, this is not a good load distribution approach. However, for circuit s35932, it still achieves superlinear speedup. Up to now, we have not found a better heuristic approach to obtain a best load distribution

TABLE VIII
CPU TIME (S) FOR EACH SLAVE DEALING WITH THE SAME PERCENTAGE OF FAULTS

circuit	1H+1S		1H+2S		1H+3S			1H+4S		
	H	S	H	S	H	S		H	S	
s400	21.1	31.5	17.9	18.1 32.9	10.4	14.6 29.6	18.1	10.3	12.5 26.2	18.2 14.6
s526	19.7	31.2	7.7	23.2 22.7	6.2	17.0 23.7	15.4	5.4	10.3 24.0	15.5 8.0
s1488	34.4	37.1	28.4	31.9 24.2	21.4	31.3 26.1	21.1	17.5	24.7 25.7	17.3 17.8
s1494	29.3	30.1	19.9	23.7 21.6	14.8	19.3 20.8	17.8	13.4	18.0 19.4	15.2 17.8
s5378	110.0	117.1	76.9	87.4 79.5	60.2	72.4 64.4	61.7	40.1	70.2 54.5	58.5 66.2
s35932	3403.9	3968.4	2081.5	2123.8 2247.0	1238.2	1433.1 1944.2	1090.5	630.4	1247.8 1744.7	1052.4 971.1

for every circuit under a given test set. This is the challenge of our future work.

VI. CONCLUSION

In this paper, we have first presented an efficient sequential diagnostic fault simulator for sequential circuits. In it, the number of fault-pair output response comparisons has been minimized by using an efficient list structure. Several ISCAS89 sequential benchmark circuits have been run to test its performance. Experimental results show that this sequential diagnostic method achieves a significant speedup but uses smaller memory as compared to the previously developed methods. To further speed up the diagnostic process, a distributed diagnostic fault simulator was then proposed and developed. To the authors' knowledge, no related approaches for sequential circuits have been reported before. The major idea for this distributed diagnostic fault simulator is that each computation node finds the IFL's for a certain percentage of faults only. Experimental results show that, for a very large circuit, s35932, the CPU time spent on fault diagnosis, excluding the time spent in finding the sizes of indistinguishability classes, is reduced from 3.3 h on a single SUN SPARC station II to less than 21 min on a network of four such machines, and superlinear speedup is exhibited. This is quite an improvement. However, a best load distribution might go through several trials. The future work is to find a better way of automatically distributing loads evenly without going through manual trials.

ACKNOWLEDGMENT

The authors thank the reviewers, especially reviewer two, for their helpful suggestions and corrections. The authors also thank Prof. W. K. Fuchs for his great help and S. Venkataraman for his correction on calculating the diagnostic resolution.

REFERENCES

[1] P. Camurati, D. Medine, P. Prinetto, and M. Sonza Reorda, "A diagnostic test pattern generation algorithm," in *Proc. Int. Test Conf.*, 1990, pp. 52–58.
[2] K. Kubiak, S. Parkes, W. K. Fuchs, and R. Saleh, "Exact evaluation of diagnostic test resolution," in *Proc. 29th Design Automation Conf.*, 1992, pp. 347–352.
[3] E. M. Rudnick, W. K. Fuchs, and J. H. Patel, "Diagnostic fault simulation of sequential circuits," in *Proc. Int. Test Conf.*, 1992, pp. 178–186.

[4] T. M. Niermann, W.-T. Cheng, and J. H. Patel, "PROOFS: A fast, memory-efficient sequential circuit fault simulator," *IEEE Trans. Computer-Aided Design*, pp. 198–207, Feb. 1992.
[5] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. Int. Symp. Circuits Syst.*, 1989, pp. 1929–1934.
[6] W.-T. Cheng and S. Davidson, "Sequential circuit test generator (STG) benchmark results," in *Proc. Int. Symp. Circuits Syst.*, 1989, pp. 1938–1941.
[7] R. B. Mueller-Thuns, D. G. Saab, R. F. Damiano, and J. A. Abraham, "Portable parallel logic and fault simulation," in *Proc. Int. Conf. Computer-Aided Design*, 1989, pp. 506–509.
[8] W. C. Wu, C. L. Lee, J. E. Chen, and W. Y. Lin, "Distributed fault simulation for sequential circuits by pattern partitioning," in *Proc. 2nd Asian Test Symp.*, 1993, pp. 176–180.
[9] P. A. Duba, R. K. Roy, J. A. Abraham, and W. A. Rogers, "Fault simulation in a distributed environment," in *Proc. 25th Design Automation Conf.*, 1988, pp. 686–691.
[10] H. K. Lee and D. S. Ha, "New method of improving parallel fault simulation in synchronous sequential circuits," in *Proc. Int. Conf. Computer-Aided Design*, 1993, pp. 2–5.
[11] C.-P. Kung and C.-S. Lin, "HYHOPE: A fast fault simulator with efficient simulation of hypertrophic faults," in *Proc. Int. Conf. Computer-Aided Design*, 1994, pp. 714–718.
[12] J. M. Jou and S.-C. Chen, "A new fault simulator for large synchronous sequential circuits," in *Proc. IEEE Asia-Pacific Conf. Circuits Syst.*, 1994, pp. 466–471.
[13] *SunOS Reference Manual*, vol. I, SUN Microsystems, 1991.
[14] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–308, 1970.

Shung-Chih Chen received the B.S. degree and M.S. degree in electrical engineering from National Cheng Kung University, Tainan, Taiwan, R.O.C., in 1984 and 1991, respectively. He is currently pursuing the Ph.D. degree at National Cheng Kung University, Tainan, Taiwan, R.O.C.

He worked at Siliconix Taiwan from 1986 to 1989. He is currently an Instructor in the Electronic Department, Nan-Tai Institute of Technology, Tainan County, Taiwan, R.O.C. His research interests include ATPG, fault simulation, and diagnostic fault simulation.

Jer Min Jou (M'89) was born in Taipei, Taiwan, R.O.C., on October 21, 1957. He received the B.S. degree in engineering science and the M.S. and Ph.D. degrees in electrical engineering and computer engineering from National Cheng Kung University, Tainan, R.O.C., in 1981, 1983, and 1987, respectively.

Since 1989, he has been an Associate Professor in the Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan, R.O.C. His current research interests include VLSI CAD/Design, data compression chip design, and asynchronous circuits and architecture design.

Dr. Jou received a Distinguished Paper Citation at the IEEE ICCAD Conference in Santa Clara, CA, in 1987.