PAPER

Fast Test Pattern Generation Using a Multiprocessor System

Hideo FUJIWARA† *and* Akira MOTOHARA††, *Members*

SUMMARY An approach for parallel processing of test-pattern generation for combinational circuits is described. In general, difficulty in test-pattern generation lies in two points; how to deal with a large number of faults, and what to do with faults that are hard to generate their test patterns. These problems can be resolved in a parallel-processing environment on a variable number of processors. Test-pattern generation is shown to be a good application of parallel processing techniques. The proposed method has been implemented on a multi-microcomputer system called LINKS-1 in order to estimate its performance. The results show that the proposed parallel processing method can get a high parallelism and achieve a high degree of acceleration of test-pattern generation.

1. Introduction

Handling the increased logic complexity of very-large-scale integration (VLSI) is severely limited by the slowness of conventional computer-aided design (CAD) tools on general-purpose (Von Neumann) computers. To alleviate this, several kinds of special-purpose hardware for logic simulation have been reported[1]-[4].

The problem of test-pattern generation for logic circuits is known to be NP-complete even for monotone circuits composed of only AND and OR gates[5], and thus it is very hard to achieve a high fault coverage for large circuits within a reasonable computing time. Some attempts to accelerate test-pattern generation and/or fault simulation through parallel processing techniques have been reported[6]-[8]. The approach proposed by Kramer[6] utilizes the MIT Connection Machine as a test-pattern generation engine. The result was that speed-up advantage can be gained only for small circuits because of the exponential nature of the proposed algorithm in which all input combinations are wastefully analyzed. El-ziq et al.[8] surveyed the state-of-the-art logic verification and fault simulation machines and presented their view such that a successful test-pattern generation system should be built as an expert system or a knowledge-based system using a special purpose engine, though no concrete configuration of the test-pattern generation engine was presented.

In this paper we shall propose a new approach for parallel processing of test-pattern generation for combinational logic circuits. Though intended for combinational circuits, this approach is also extendable to sequential circuits. The proposed method has been implemented on a multi-microcomputer system called LINKS-1. The experimental results are presented for real combinational circuits of up to 1000 gates and the performance of the proposed parallel processing method is analyzed. The results show that the proposed method can get a high parallelism and achieve a high degree of acceleration of test-pattern generation.

2. Parallel Processing of Test Pattern Generation

Many algorithms for test-pattern generation have been proposed over the years. In our system proposed here, we use the PODEM algorithm[9] for test-pattern generation because of its conciseness and effiency of the algorithm, in which, however, the multiple backtrace technique of the FAN algorithm[10] is adopted in order to further improve the efficiency of the PODEM algorithm. Our system also adopts the concurrent fault simulation[11] as a technique for fault simulation. Hence, the modified PODEM algorithm combined with the concurrent fault simulation is used in our system.

Various techniques for parallel processing to accelerate test-pattern generation could be considered as follows:

(a) Separate the tasks of the algorithm into separate processors.

(b) Distribute the same kind of tasks to many processors in order to process many faults and test-patterns simultaneously.

(c) Partition the circuit under consideration into sub-circuits to distribute them to separate processors.

Since the test-pattern generation algorithms such as the PODEM and the FAN use the branch-and-bound method, the technique (c) may cause poor space-parallelism because very few elements are active at a time in test-pattern generation algorithms due to its backtracking mechanism. Here we adopt the techniques (a) and (b) to realize parallel processing of many faults and test-patterns. If faults A and B are

Manuscript received October 5, 1987.
Manuscript revised December 2, 1987.

† Faculty of Engineering, Meiji University, Kawasaki-shi, 214 Japan.
†† Advanced Device Laboratory, Semiconductor Research Laboratory, Matsushita Electric Industrial Co., Ltd., Moriguchi-shi 570 Japan.

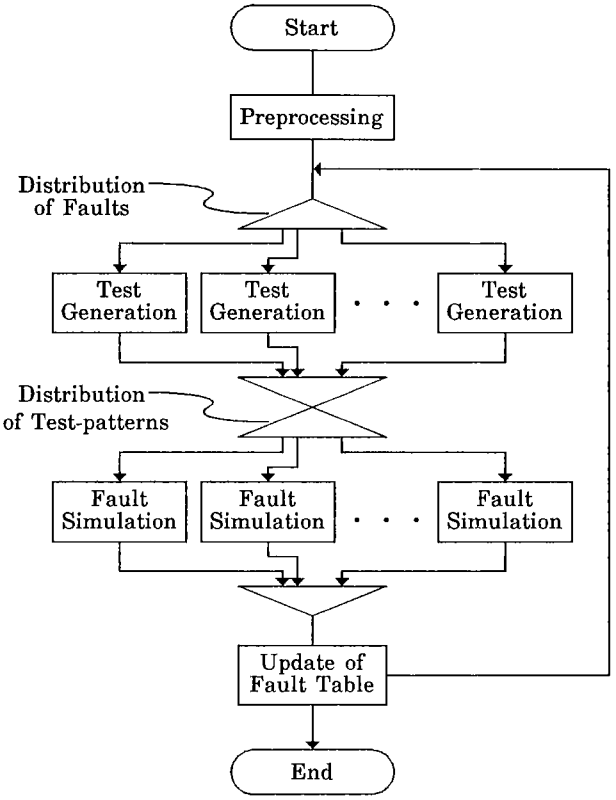

Fig. 1 Flow of concurrent process of faults.

topologically near each other, it may often occur that a test-pattern generated for fault A can also be a test-pattern for B. In this case, it is waste of time to generate tests for those faults independently by separate processors. However, for large circuits there are many faults that can be processed independently of each other because they are far separated topologically. For those faults, test-pattern generation and fault simulation can be done separately and simultaneously by distributing them to many processors. Hence, the speed-up of test-pattern generation and fault simulation could be achieved by distributing faults and tasks to separate processors and by performing the processing of many faults simultaneously. This parallel processing is illustrated in Fig. 1. First, some faults are taken out from the fault table to generate test-patterns simultaneously. To get a high parallelism without wase of time, those faults should be a set of faults which can be test-generated independently of each other. However, it is a hard and time-consuming problem to obtain suh a set of faults. Faults that are extracted *depthfirstly* from primary inputs toward primary outputs are often detected simultaneously by same test-patterns because faults on a path are often sensitized simultaneously. Therefore, we have adopted here a simple procedure as approximation such that faults are extracted not depth-firstly but *breadth-firstly* from primary inputs toward primary outputs of the circuit under test. Each fault is then distributed or assigned to a processor to generate a test

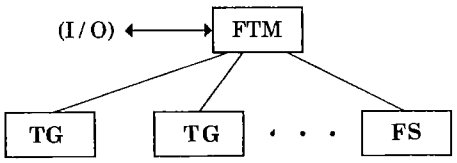

Fig. 2 System configuration.

pattern independently of other processors. Processors carry out test generation by themselves using the modified PODEM algorithm. Test-patterns generated by those processors are then distributed again to separate processors for fault simulation. Each processor performs the concurrent fault simulation for a given test-pattern independently of other processors.

In the parallel processing of Fig. 1, many faults are processed concurrently and thus the management of the fault table is important. The information, whether a fault has been processed, or is being processed, or not, is stored in the fault table and updated. The processor that manages the fault table is called the Fault Table Manager (FTM). The processors that generate test-patterns are called Test Generators (TGs). The processors that perform fault simulation are called Fault Simulators (FSs). Fig. 2 shows the configuration of the multiprocessor system corresponding to the flow of Fig. 1. The tasks of processors are as follows:

The Fault Table Manager performs the following four tasks:

(1) Communication from/to the Host Computer

FTM receives the information necessary to generate test-patterns (circuit configuration data, limited number of backtracks, etc.) from the host computer and sends the results of test-pattern generation (test-patterns, fault coverage, etc.) to the host computer.

(2) Fault Table Management

Before test-pattern generation, FTM creates a table of the faults to be tested. This is carried out by fault modeling and collapsing. The fault table is sorted is sequence so that faults are taken out breadth-firstly from primary inputs toward primary outputs of the circuit under test. During the test-pattern generation mode, FTM extract a fault, which is not processed yet, from the top of the fault table and transfers it to an idle TG. FTM receives a list of detected faults from each FS and then eliminates (or flags) those faults from the fault table.

(3) Test Pattern Management

FTM receives a test-pattern from each TG and stores it in an FIFO (first-in first-out) queue. While the FIFO queue is not empty, FTM takes out a test-pattern one by one from the queue and sends it to an idle FS.

(4) Watch of TGs and FSs

When FTM is free from above (1)-(3) tasks, FTM always watches all TGs and FSs. If an idle processor is found, FTM gives a task to it.

When a TG (Test Generator) receives a fault from

FTM, it begins to generate a test- pattern for the fault. After ending of test-pattern generation, TG sends the result to FTM. The result is one of the following three cases: (1) A test-pattern is obtained. (2) Test-pattern generation for the fault is aborted when the number of backtracks in the branch-and-bound processing exceeds the predefined limit value, e.g. 100 or 1000. (3) The given fault is found to be undetectable or redundant.

When a FS (Fault Simulator) receives a test-pattern from FTM, it begins to simulate all the faults. After ending the fault simulation, FS sends a list of detected faults to FTM.

3. The Multi-Microcomputer System

The above-mentioned approach to parallel processing of test-pattern generation was implemented on a multi-microcomputer system called LINKS-1 which was developed for three-dimentional image generation [12]. The basic architecture of LINKS-1 is shown in Fig. 3. The system consists of a root computer (RC) and 64 (up to 256) node computers (NCs) of the same structure. RC and NC are called unit computers (UCs) consisting of a CPU Z8000 and 1 MB memory as shown in Fig. 4. Each node computer NC_i is linked radially to RC through an intercomputer memory swapping unit (IMSU) and a bus switch (BS). RC is connected with the host computer (Zilog System 8000) through the parallel communication line. Each $IMSU_i$ has two memory areas (2×64 KB) which are connected to RC and NC_i through a bus exchange switch to transfer a great amount of data at a time. Data transfer between RC and NC_i is perfomed in the following way: The sender first loads data to the memory area (64 KB) of $IMSU_i$ connected to the sender through the data bus of 16 bits width and sends a switching request to the receiver. When the receiver becomes idle, it sends a bus exchange signal to $IMSU_i$. Then, $IMSU_i$ exchanges memory areas of RC and NC_i by turning over the bus exchanges switch and sends an acknowledgment signal to the sender. Receiving the acknowledgment signal, the sender knows that switching has been completed.

In this multi-microcomputer system, LINKS-1, we use RC as FTM and each NC as TC or FS. Since each NC is so programmed as to act both TG and FS, the numbers of TGs and FSs can change dynamically. For every NCs, the task of fault simulation has priority over the task of test-pattern generation. This means that if there are some test-patterns to simulate, a task as FS is given to NC. FTM, TG, and FS are all programmed in the C language.

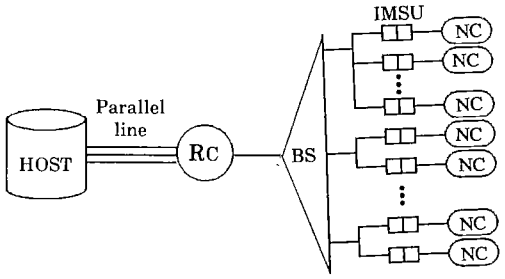

Fig. 3 LINKS-1

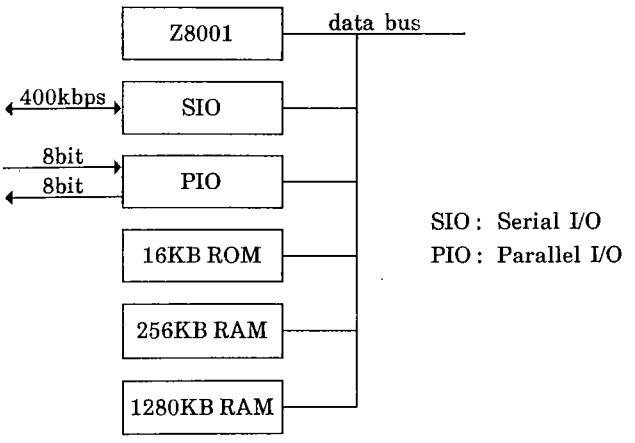

Fig. 4 Unit computer.

4. Performance Estimate

We present here our estimates for the performance of our system on the following points:

(a) What degree of acceleration or parallelism can be achieved by the parallel processing approach proposed in this paper?

(b) When will the performance of the system become saturated as the number of unit computers increase?

In order to clarify this, we consider the following measures:

(1) Computing Time: $t(n)$

Let $t(n)$ be the total computing time of the multi-microcomputer system with n unit computers (see Fig. 5).

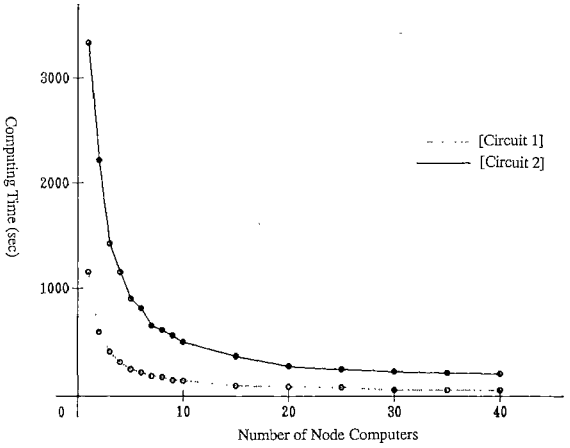

Fig. 5 Computing time vs number of node computers.

Table 1 Characteristics of test circuits.

circuit name	circuit type	number of				fault coverage*
		gates	faults	inputs	outputs	
circuit 1	ECC	464	1338	8	7	99.8%
circuit 2	ECC	876	1871	33	33	99.3%

* Fault coverages are obtained by sequential processing using single Unit Computer. The faults remained undetected after more than 100 backtracks are aborted.

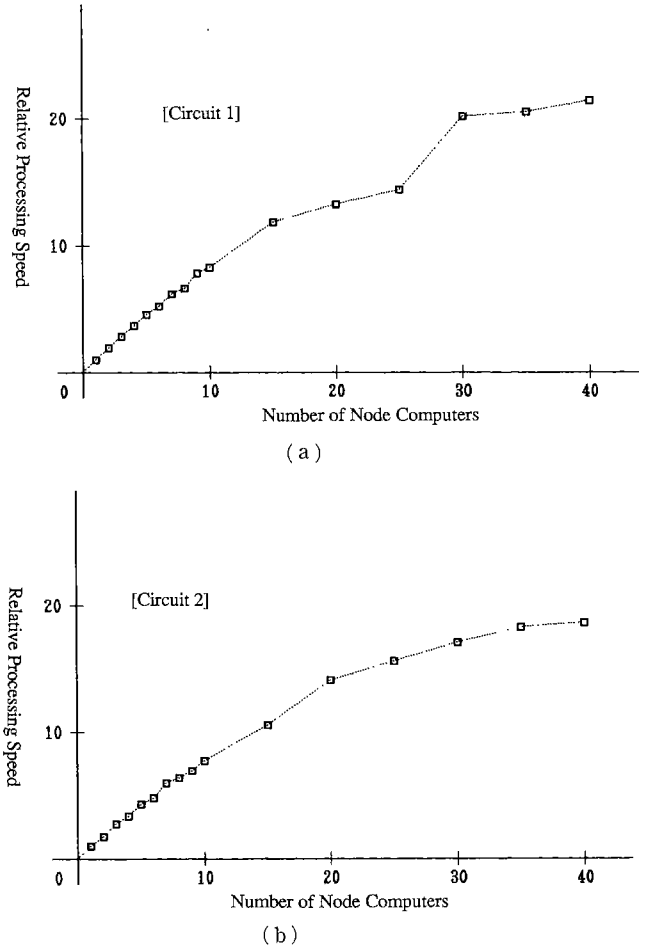

Fig. 6 Relative processing speed vs number of node computers.

(2) Relative Processing Speed and Efficiency: $s(n)$ and $e(n)$

The relative processing speed of the system with n unit computers is defined by

$$s(n)=\frac{t(1)}{t(n)}$$

and efficiency by

$$e(n)=\frac{s(n)}{n}$$

(3) Rate of RC Working: RRC

Communications between RC and NCs are carried out by polling of RC. RC watches switching requests from NCs, and if a request is found, RC switches IMSU

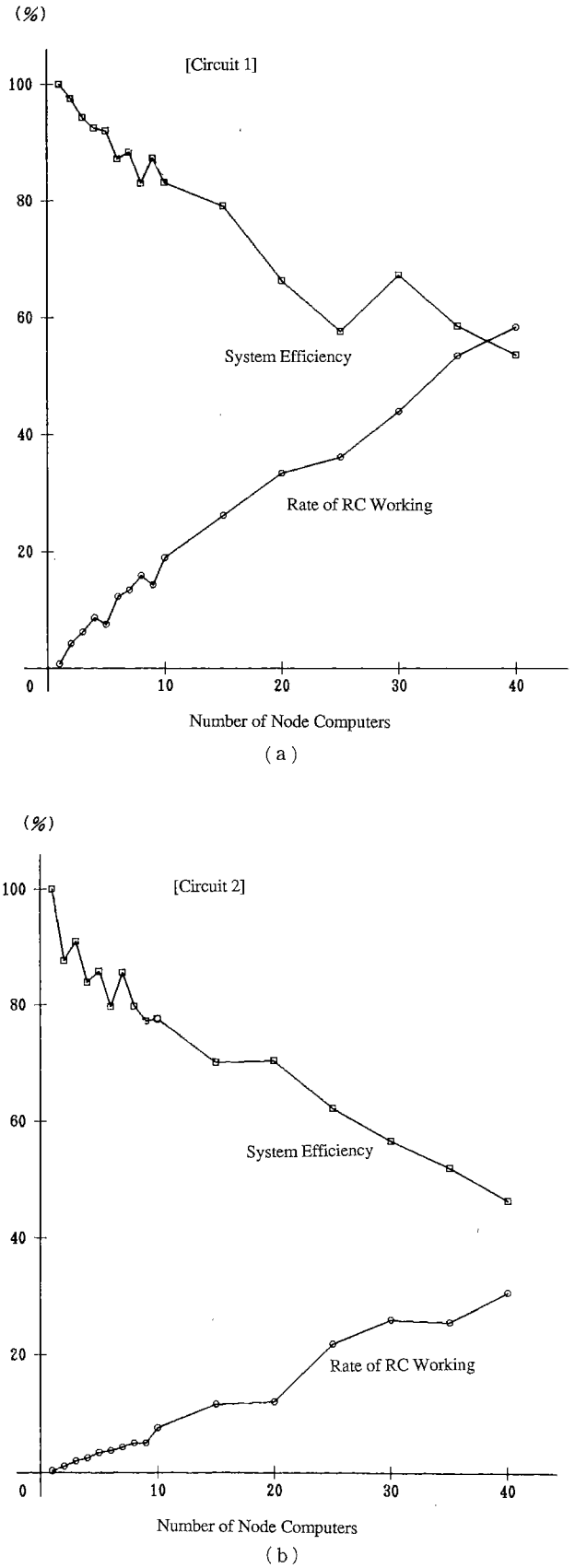

Fig. 7 System efficiency vs number of node computers.

corresponding to the NC. RC is therefore watching NCs' switching requests when RC is free from the task of management of faults and test-patterns in the fault table and the FIFO queue. Therefore, while RC is busy about the data management, idle NCs are kept waiting until RC completes the task. Let TRC be the total processing time of RC and let TMRC be the processing time of RC related to the management of faults and test-patterns. The larger the value of TMRC becomes, the longer the waiting time of idle NCs will be. Hence, the increase of TMRC would cause the decrease of the system performance. To estimate the performance of RC, the rate of RC working on data management (RRC) is defined by

$$\mathrm{RRC}=\frac{\mathrm{TMRC}}{\mathrm{TRC}}$$

Table 1 shows the characteristics of two combinational circuits used as the experimental example. The experimental results are shown in Figs. 5–7. The fault coverage for circuits # 1 and # 2 is 99.8% and 99.3%, respectively.

The results in Fig. 6 shows that the system with 40 unit computers is about 20 times faster than the system with one unit computer. Considering that the test-pattern generation algorithm uses a branch-and-bound method and has poor parallelism in general due to its backtracking mechanism, we can say that the degree of parallelism obtained here is very high.

The rate of RC working is in proportional to communication volume such as test-patterns and lists of faults between RC and NC's, and hence it increases in proportion to the number of NCs. This tendency is well shown in Fig. 7, where the graph of RRC is almost linear. As mentioned above, the larger the value of RRC becomes, the longer the waiting time of idle NCs will be. To increase the system performance more, it is important to decrease the waiting time of NCs. This can be done by distributing the tasks of RC to many processors, i.e., by lightening the burdern of RC. In the next section, we shall present an extended method of the above-mentioned parallel processing approach, in which the fault table in divided into sub-tables to be processed by separate processors.

5. Division of Fault Table

In the preceding system of Fig. 2, only one FTM manages a fault table. To lighten the burden of the FTM, we consider here a tree-structured multiprocessor system with many FTMs as shown in Fig. 8. In this figure, the root processor is called the Fault Table Divider (FTD), whose tasks are (1) to communicate from/to the host computer, (2) to divide a fault table into smaller subtables, and (3) to distribute those divided sub-tables to FTMs. Note that the circuit under test is not divided into sub-circuits. Since a test-pattern for a fault can detect many other faults in general,

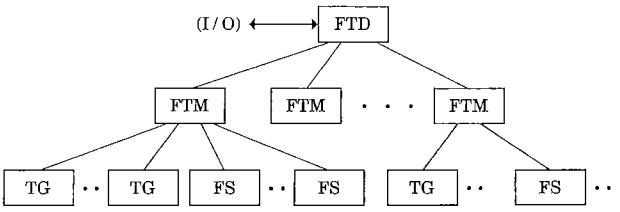

Fig. 8 System configuration.

restriction on the range of simulated faults will cause the wasteful repetition of fault simulation and will decrease the efficiency of fault simulation. Therefore, this approach of dividing the fault table and distributing those sub-tables to FTMs might increase this kind of overlap work between processors. To minimize this overlap work, we divide here the fault table so that all faults in the same sub-table are topologically near.

The communication between the FTD and FTMs occurs only before and after the test-pattern generation process. The division of the fault table and its distribution to FTMs are performed as preprocessing. Receiving a divided fault sub-table, each FTM manages only those faults in the fault sub-table. Each sub-system composed of an FTM and its successors (TGs and FSs) generates a set of test-patterns for the faults distributed by FTD, and sends the results (a set of test-patterns and a list of detected faults) to FTD. Collecting all the results from all FTMs, the FTD compiles them into a result (the fault coverage and test-patterns).

There are some merits as well as demerits of this approach. The main merit is that the communication overhead can be reduced by dividing the fault table, i.e., by distributing the task of fault-table management to several FTMs. Moreover, from the viewpoint of the storage, the concurrent fault simulation in FSs will come to deal with very large circuits by dividing the fault table. On the other hand, this approach has a demerit of overlap work between processors as mentioned above; the wasteful repretition of fault simulation due to restriction on the rage of faults to be simulated by each processor.

The experimental results for the circuits of Table 1 are shown in Fig. 9. This figure shows the relative processing speed of the system versus the number of unit computers in the system. The relative processing speed represents a kind of degree of acceleration of the system. From this figure, we can see that the degree of acceleration, i.e., the relative processing speed can be higher through the fault table division. The most effective division strategy, however, is dependent much on the circuit size. For example, for circuit # 2, much higher acceleration is obtained by both methods of dividing a fault table into halves and quarters (Fig. 9 (b)). For circuit # 1, however, halving is effective but quartering is not (Fig. 9(a)). For large circuits, the communication volume between processors is large

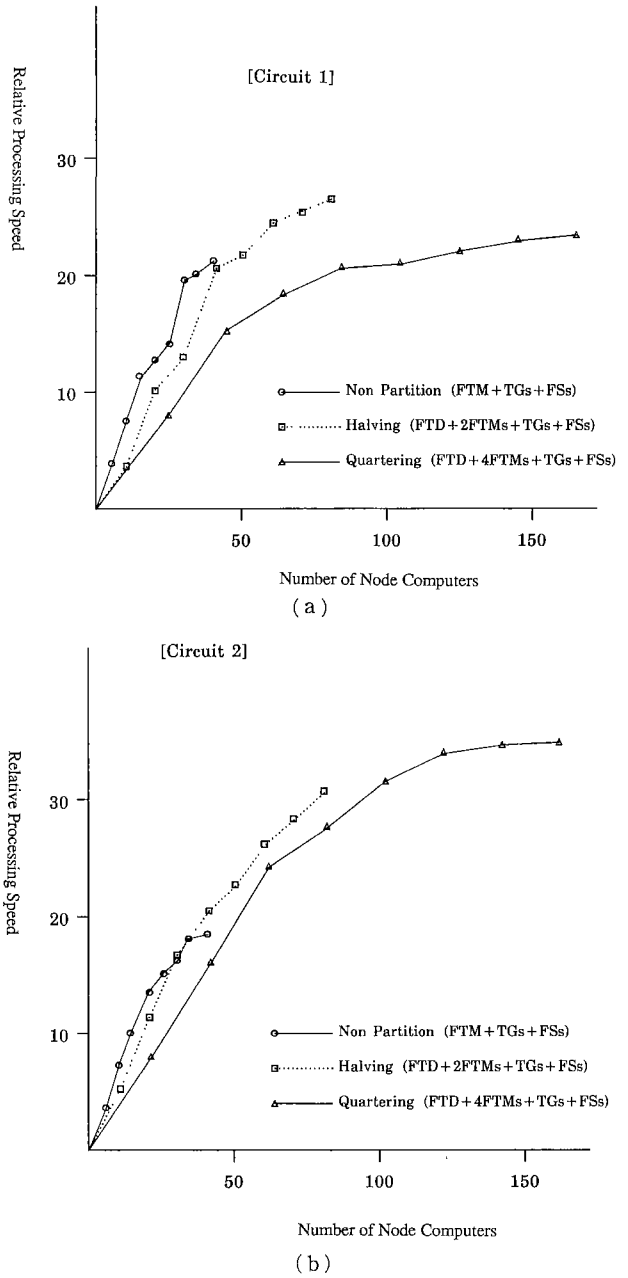

Fig. 9 Relative processing speed vs number of node computers.

enough to be partitioned. Hence, in general, the more the fault table is divided, the better the effect on the acceleration of test-pattern generation will be. This is not true for small circuits since the algorithm overhead caused by the increase of test-patterns is not negligible.

6. Conclusions

In this paper, we have presented an approach for parallel processing of test-pattern generation, which was implemented on a multi-microcomputer system called LINKS-1. From our experimental results, our approach based on the concurrency in processing many faults simultaneously is very effective to accelerate test-pattern generation especially for large circuits. Though the proposed system is intended for combinational circuits, this approach can be extended to sequential circuits too, and the similar results could be obtained. In general, the test-pattern generation algorithm has a poor parallelism due to its backtracking mechanism. Considering this latent difficulty, the parallelism obtained in this paper is very high and a degree of acceleration of test-pattern generation has been achieved.

Acknowledgement

We would like to thank Professors K. Ohmura and I. Shirakawa of Osaka University for their giving us an opportunity of using LINKS-1. Thanks also to Mr. K. Nishimura for his assistance in obtaining the experimental results of this work.

References

(1) G. F. Phister: "The Yorktown simulation engine: Introduction", Proc. 19th Design Automation Conf., pp. 51-54 (June 1982).
(2) M. Abramovici, Y. H. Levendel and P. R. Mennon: "A logic simulation machine", Proc. 19th Design Automation Conf., pp. 65-73 (June 1982).
(3) T. Sasaki, N. Koike, K. Ohmori and K. Tomita: "HAL: A block level hardware logic simulator", Proc. 20th Design Automation Conf., pp. 150-156 (June 1983).
(4) N. V. Brunt: "The Zycad logic evaluator and its application to modern system design", Proc. IEEE Int. Conf. on Computer Design, pp. 232-233 (Oct. 1983).
(5) H. Fujiwara and S. Toida: "The complexity of fault detection problems for combinational logic circuitrs", IEEE Trans. Comput., **C-31**, 6, pp. 555-560 (June 1982).
(6) G. A. Kramer: "Employing massive parallelism in digital ATPG algorithm", Proc. Int. Test Conf., pp. 108-114 (1983).
(7) L. T. Smith and R. R. Rezac: "Methodology for and results from the use of hardware logic simulation engine for fault simulation", Proc. Int. Test Conf., pp. 224-228 (1984).
(8) Y. M. Elziq. H. H. Butt and A. K. Bhatt: "An automatic test pattern generation machine", Proc. IEEE Int. Conf. on Computer-Aided Design 84, pp. 257-259 (Nov. 1984).
(9) P. Goel: "An implicit enumeration algorithm to generate tests for combinational logic circuits", IEEE Trans. Comput., **C-30**, 3, pp. 215-222 (March 1981).
(10) H. Fujiwara and T. Shimono: "On the acceleration of test pattern generation algorithms", IEEE Trans. Comput., **C-32**, 12, pp. 1137-1144 (Dec. 1983).
(11) E. G. Ulrich and T. Baker: "The concurrent simulation of nearly identical digital networks", Proc. 10th Design Automation Workshop, pp. 25-27 (June 1973).
(12) H. Nishimura, H. Ohno, T. Kawata, I. Shirakawa and K. Ohmura: "LINKS-1: A parallel pipelined multimicrocomputer system for image creation", Proc. 10th Int. Symp. on Computer Architechture, pp. 387-394 (June 1983).
(13) H. Fujiwara: "Logic Testing and Design for Testability", The MIT Press (1985).

Hideo Fujiwara was born in Nara, Japan, on February 9, 1946. He received the B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively.

Since 1974 he has been with the Department of Electronic Engineering, Faculty of Engineering, Osaka University. In 1981 he was a Visiting Research Assistant Professor at the University of Waterloo, and in 1984 he was a Visiting Associate Professor at McGill University, Canada. Since 1985 he has been an Associate Professor in the Department of Electronics and Communications, Faculty of Engineering, Meiji University, Tokyo, Japan.

His research interests include logic design, design for testability, test pattern generation, fault simulation, built-in self-test, and expert systems for design and test. He is the author of Logic Testing and Design for Testability (MIT Press, 1985). He is a senior member of IEEE and a member of the Information Processing Society of Japan.

Akira Motohara was born in Kumamoto Prefecture, Japan, on February 22, 1961. He received the B.E. and M.E. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1983 and 1985, respectively. He joined Matsushita Electric Industrial Co., Ltd., Osaka, Japan, in 1985. He is currently a member of Advanced Devices Laboratory of Semiconductor Research Center. His research interests include test pattern generation, design for testability, fault simulation, computer architecture, and parallel processing. He is a member of IEEE.