Optimal Granularity of Test Generation in a Distributed System

HIDEO FUJIWARA, FELLOW, IEEE, AND TOMOO INOUE

Abstract—**The problem of test generation for logic circuits is known to be *NP*-hard, and hence, it is very hard to speedup the test generation process due to its backtracking mechanism. This paper presents an approach to parallel processing of test generation for logic circuits in a loosely-coupled distributed network of general purpose computers, and analyzes the effects of the allocation of target faults to processors, the optimal granularity (grain size of target faults), and the speedup ratio of the multiple processor system to a single processor system.**

I. INTRODUCTION

THE TEST generation process usually includes both test-pattern generation and fault simulation. For test-pattern generation, several efficient algorithms such as PODEM [1], FAN [2], and SOCRATES [3] have been reported. However, the problem of test-pattern generation for logic circuits is known to be *NP*-hard [4], [5], and hence, the computational requirements grow exponentially in general as the circuit size increases. For simulating very large faulted circuits, deductive and concurrent fault simulation are known to be efficient. The computational complexity of this fault simulation is less than $O(G^3)$ and seems to behave as an $O(G^2)$ algorithm [6]–[8]. These facts imply that a test generation system implemented on a single general purpose computer takes a prohibitive amount of computing time for very large circuits.

Handling the increased logic complexity of VLSI circuits is severely limited by the slowness of conventional CAD tools on a general purpose computer. To alleviate this, several kinds of special purpose hardware accelerators have been reported, e.g., IBM's Yorktown Simulation Engine (YSE) [9], NEC's Hardware Accelerator (HAL) [10], Zycad, Silicon Solutions, Daisy, etc. [11], [12]. There are attempts to accelerate fault simulation [13] and test-pattern generation [14] using hardware logic simulators. Kramer's approach [15] uses the MIT Connection Machine as a test-pattern generation engine. Until now, the results of [14], [15] were that the speedup advantage can be gained only for small circuits because of the exponential nature of the proposed algorithms in which all input combinations are wastefully analyzed. Elziq *et al.* [16] surveyed the state-of-the-art logic verification and fault simulation machines and presented their view such that a successful test 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.

Although these special purpose hardware engines certainly provide the fastest simulation, their hardware cost is very expensive and their special purpose architecture is inflexible to other applications such as test generation. An alternative to special purpose hardware engines is the use of a loosely-coupled distributed network of general purpose computers which are less expensive and much more flexible than special purpose hardware engines. There is a report of a distributed fault simulator implemented on a loosely-coupled network of general purpose computers in which a close to linear speedup is achieved [17]. For test generation, there is a report of parallel test generation on a loosely-coupled distributed system which predicted the performance of parallel processing by dealing with multiple heuristic schemes [20]. There is also a report, which appeared after this paper had been submitted, of parallel test generation on a tightly-coupled multiprocessor system which presented heuristics to partition faults for parallel test generation with minimization of the overall runtime and test length [26].

In Motohara *et al.* [18] and Fujiwara [19], two types of parallelisms, *fault parallelism* and *search parallelism* were presented, both of which parallelize test generation using tightly-coupled multiple processor systems. Fault parallelism refers to dealing with different faults in parallel, and search parallelism refers to searching different nodes of a decision tree (in a *branch-and-bound* search) or to searching different input-vectors in parallel. Usually, a large percentage of faults are easy-to-detect and only a small fraction of faults are hard-to-detect faults which remain undetected after a large number of backtracks. So, the approach combined with both fault parallelism and search parallelism might be most effective if fault parallelism is first performed and then search parallelism is applied to the remaining hard-to-detect faults. In this paper, we shall consider the fault parallelism on a loosely-coupled distributed network of general purpose computers instead of a tightly-coupled multiple processor system. In the fault parallelism of Motohara *et al.* [18] and Fujiwara [19], the number of target faults allocated to a processor each time is only one, and hence, no optimal *granularity* of target faults (size of target faults) is

Manuscript received March 14, 1989; revised September 1, 1989. This paper was recommended by Associate Editor S. C. Seth.

The authors are with the Department of Computer Science, Meiji University, Tama-ku, Kawasaki 214, Japan.

IEEE Log Number 9036175.

0278-0070/90/0800-0885$01.00 © 1990 IEEE

considered there. In this paper we shall consider the fault parallelism in which a cluster of faults will be allocated to each processor instead of allocating one fault each time. We shall analyze the effect of the number of faults allocated to a processor each time to find the optimal granularity in both cases of *static* and *dynamic* task allocation and derive the speedup ratio of the multiple processor system to a single processor system, in order to see the performance of the multiple processor system.

II. Architecture of the Distributed System

The architecture of our loosely-coupled multiple processor system is illustrated in Fig. 1. The *client* and *servers* are connected via a communication bus. In this network, a client processor requests a remote server processor to execute a task and to return the results to the client. When a server finishes its assigned task, it sends the result to the client and requests a new task. The client saves the result and provides a new task for the server. Client service discipline is first come, first served. In this distributed processing, the original problem is partitioned into subproblems or tasks (*task partitioning*), and each task is allocated to the servers (*task allocation*).

Since our problem is test generation, our goal is to generate test-patterns for all faults. The problem domain is thus a set of faults. Here, we shall consider distribution of a cluster of faults (called *target* faults) to servers to generate test-patterns for them. Therefore, a subproblem or a task allocated to a server is a test generation problem for a cluster of faults. For a given cluster of faults, each server can perform test-pattern generation and fault simulation. The result is a set of faults which are detected by a set of generated test-patterns, a set of redundant faults, and a set of aborted faults due to the exceeded backtrackings. This information is sent to the client and the fault table is updated by the client. The client then extracts a new cluster of faults which have not yet been processed by any server and sends it to the server. The server then executes test generation for the cluster of faults. This process continues until all faults in the fault table are processed.

When the client allocates tasks to servers, the size of tasks (granularity) has to be determined. In this task allocation, we can consider two ways: *static* and *dynamic* task allocation. Static task allocation refers to allocating the same number of target faults to each server throughout. In dynamic task allocation, the number of target faults will vary as time goes on. In both cases, if we decrease the granularity in order to exploit better parallelism, then servers complete the tasks more rapidly, and hence, send requests to the client more frequently. This increases the communication overhead and in turn slows down processors. In this way, the total computation time is influenced by the size of target faults allocated to each server, and hence, it is important to predict the optimal granularity which will yield the best performance for a given distributed system, though it is very difficult to predict the optimal dynamic granularity since the interaction among various performance factors is very complex.

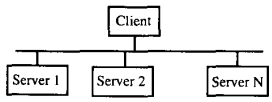


Fig. 1. Architecture of the distributed system.

III. Formulation of the Problem

We shall consider a distributed network consisting of a client and N servers as shown in Fig. 1. Let M be the total number of faults of a given circuit. A process of test-pattern generation for a fault f_i is called a *process for fault* f_i. The result of a process for a fault is whether 1) the fault is detected by a test-pattern, or 2) the fault is redundant, or 3) the process is aborted due to the exceeded backtracking. Let τ_{ij} be the processing time of server j for fault f_i, i.e., the computation time of server j to complete test-pattern generation process for fault f_i.

Let δ_{ij} be the probability that process for fault f_i is allocated to server j. Let λ_{ij} be the probability that server j communicates to the client after process for fault f_i. Let τ_c be the mean communication time which includes waiting time due to contention and data transfer time between the client and a server. Then, the average time necessary to complete all processes allocated to server j is

$$T_j = \sum_{i=1}^{M} \delta_{ij}(\tau_{ij} + \lambda_{ij}\tau_c). \tag{1}$$

The average time necessary to complete all processes is defined by the maximum of T_j:

$$T = \max\{T_j\}. \tag{2}$$

The problem is thus to find a task allocation schedule which minimizes T.

IV. Homogeneous Problem with Static Task Allocation

4.1. Optimal Granularity

First we shall consider *static* task allocation of faults where the same number of target faults will be transferred from client to a server at each communication.

To obtain the optimum task allocation schedule, it is important to equalize the load of servers, e.g., hard-to-detect faults are distributed equally among the servers as well as easy-to-detect faults. However, it is very difficult or even impossible to know which faults are hard- or easy-to-detect *a priori* before test-generation. One solution might be to estimate the testability of faults by testability analysis using controllability/observability measures [25]. Here, we shall adopt a more stringent assumption that every fault will be distributed equally among the servers as follows.

We assume a *homogeneous* case such that 1) all servers are uniform, i.e., $\tau_{ij} = \tau_i$ for all faults f_i and servers j and 2) for any fault i the probability that fault i is allocated to server j is equiprobable for all servers j independently of server j, i.e., $\delta_{ij} = \delta_i$ for all faults f_i and servers j. The second assumption that the probability δ_{ij} is equiprobable intends that every fault will be distributed equally among the servers.

Let m be the number of target faults transferred from the client to any server j at each communication. Suppose fault i is in the set of m target faults allocated to server j. Then the probability that server j communicates to the client after process for fault f_i is $1/m$ since such a communication occurs only once for those m faults. Hence, $\lambda_{ij} = 1/m$. Let τ be the mean processing time for each fault, i.e.:

$$\tau = \frac{\sum_{i=1}^{M} \tau_i}{M}. \tag{3}$$

Case I (Without Fault Simulation): First, suppose that test-pattern generation without fault simulation is performed for each fault. Then, the probability that process for fault f_i is allocated to server j is $\delta_{ij} = 1/N$. By substituting these expressions to (1) and (2), we have

$$\begin{aligned} T_j &= \sum_{i=1}^{M} \delta_{ij}(\tau_{ij} + \lambda_{ij}\tau_c) \\ &= \sum_{i=1}^{M} \frac{1}{N}\left(\tau_i + \frac{1}{m}\tau_c\right) \\ &= \frac{M}{N}\left(\tau + \frac{1}{m}\tau_c\right). \end{aligned} \tag{4}$$

Since this expression is a monotone deceasing function of m and $1 \le m \le M/N$, the minimum of T is obtained when $m = M/N$:

$$T_{\min} = \frac{M}{N}\tau + \tau_c. \tag{5}$$

The above expression means that for the homogeneous case such that $\tau_{ij} = \tau_i$ and $\delta_{ij} = \delta_i$ for all faults f_i and servers j and that test-pattern generation process without fault simulation is performed for each fault, the best performance is obtained when each server receives all target faults only once from the client. However, it may often occur that a test-pattern generated for a fault can also be a test-pattern for other faults. Hence, if we apply the fault simulation process after the test-pattern generation process, then all faults may not have to be processed.

Case II (With Fault Simulation): Suppose that the client requests a server process m target faults. The server generates a test-pattern for one of m faults, and finds out all detected faults by the test-pattern where fault simulation is performed for all faults in the circuit, not just those in the set of m target faults. It repeats test-pattern generation and fault simulation until all target faults are processed. Suppose that after this test generation process for m target faults completes, ρm faults are *newly* found to be either detectable or redundant. Note that those newly found ρm faults are from all faults in the circuit, not just in the set of m target faults. The term "newly" means that those faults were known to be neither detectable nor redundant but have been newly found to be either detectable or redundant. Let us call those faults *newly processed faults*.

Let us define the *ratio of newly processed faults to target faults* by

$$\rho = \frac{\text{number of newly processed faults per server}}{\text{number of target faults per server}}. \tag{6}$$

Note that this ratio will decrease as the number of processed faults increases. Therefore, it is expressed as ρ_i, the ratio for ith processed fault f_i.

During each iteration of the server process, m target faults are processed by the server and ρm faults are newly found to be either detectable or redundant through both test-pattern generation and fault simulation. Some faults are found to be either detectable or redundant only by test-pattern generation and other faults are found to be detectable after fault simulation. Hence, the probability that fault i is processed by some server is

$$\frac{m}{\rho_i m} = \frac{1}{\rho_i} \tag{7}$$

where ρ_i is the ratio of newly processed faults to target faults when fault f_i is processed.

On the other hand, the probability that fault i is processed by some server is defined by $\Sigma_{j=1}^{N} \delta_{ij}$. Therefore, we have

$$\sum_{j=1}^{N} \delta_{ij} = \frac{1}{\rho_i}. \tag{8}$$

From the assumption that $\delta_{ij} = \delta_i$, we have

$$\sum_{j=1}^{N} \delta_{ij} = \sum_{j=1}^{N} \delta_i = N\delta_i. \tag{9}$$

Hence, from (8) and (9) we have

$$\delta_{ij} = \delta_i = \frac{1}{N\rho_i}. \tag{10}$$

The average time necessary to complete all processes allocated to server j is obtained from (1) by substituting (10) and $\tau_{ij} = \tau_i$ and $\lambda_{ij} = 1/m$:

$$T_j = \sum_{i=1}^{M} \frac{1}{N\rho_i}\left(\tau_i + \frac{\tau_c}{m}\right). \tag{11}$$

The right side of the above equation is independent of the index j, i.e., the average processing time of server j is the same as that of other server. This is caused by the assumption of homogeneity that all servers are uniform and for any fault i the probability that fault i is allocated to

server j is equiprobable for all servers j. Hence, the total amount of processing and communication time T is

$$T = \sum_{i=1}^{M} \frac{1}{N\rho_i}\left(\tau_i + \frac{\tau_c}{m}\right). \tag{12}$$

4.1.1. Communication Time: τ_c: We assume that 1) the size of data transferred between the client and a server is constant, and hence, the data transfer time during communication between the client and servers is constant, and that 2) the number of requests from servers to the client is proportional to the total number of servers, N, and hence, the waiting time during communication between the client and servers is proportional to N. Hence, we have

$$\tau_c = t_0 + t_1 N \tag{13}$$

where t_0 and t_1 are constants.

4.1.2. Ratio of Newly Processed Faults to Target Faults: ρ: Fig. 2(a) and (b) give the typical curves of the number of newly processed faults for two large circuits c5315 and c7552 of the ISCAS'85 benchmark circuits [24]. The axis of ordinates shows the number of faults which are *newly* found to be detectable by fault simulation after test-pattern generation for a target fault, and the axis of abscissas shows the number of processes. From this figure we can see that the number of newly processed faults will quickly decrease as the number of processed faults increases. Hence, we assume that the ratio of newly processed faults to target faults, $\rho(x)$, is

$$\rho(x) = \frac{1}{r_0 + r_1 x} \tag{14}$$

where x is the number of processed faults and r_0 and r_1 are constants.

The ratio $\rho(x)$ will also decrease monotonically as the number of target faults increases. After receiving the list of detected faults and redundant faults from a server, the client renews the fault table by flagging the newly detected faults and redundant faults. Since many servers are working simultaneously, some servers may find the same faults detected. These overlapped processes for the faults that are simultaneously detected by different servers are wasteful. The number of newly processed faults per fault will decrease as the number of target faults per server and the number of servers increases. Therefore, we can assume

$$\rho(x) = \frac{1}{r_0 + r_1 x + r_2 mN} \tag{15}$$

where r_2 is a constant. In the above expression, the factor $r_2 mN$ accounts for the decrease ratio of newly processed faults due to overlapped processing.

Suppose that the number of processed faults is i when fault $f_{\pi(i)}$ is processed where π (π: $\mathfrak{M} \rightarrow \mathfrak{M}$) is a permutation of $\mathfrak{M} = \{1, 2, \cdots, M\}$. Then, from (15), the ratio of newly processed faults when fault $f_{\pi(i)}$ is processed can be expressed as

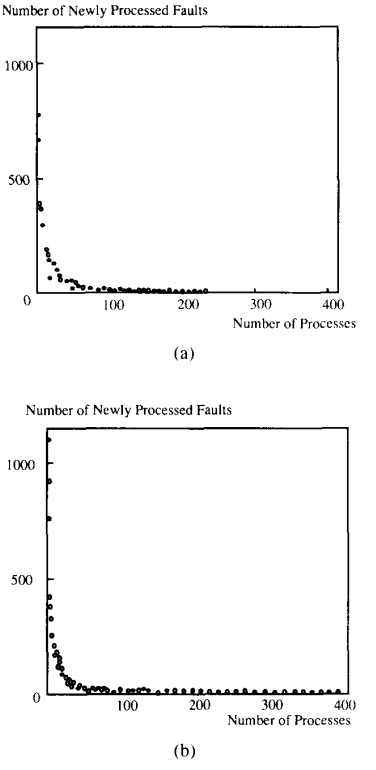


Fig. 2. Number of newly processed faults. (a) c5315. (b) c7552.

$$\rho_{\pi(i)} = \frac{1}{r_0 + r_1 i + r_2 mN}. \tag{16}$$

Let $\mathcal{P}$ be the set of all permutations of $\mathfrak{M}$. There is a one-to-one correspondence between permutations of $\mathfrak{M}$ and sequences of faults. The total number of sequences is $M!$.

Next we shall take an average of total processing time for all permutations. From (12) and (16) we have

$$T = \frac{\sum_{\pi\in\mathcal{P}} \sum_{i=1}^{M} \frac{1}{N}(r_0 + r_1 i + r_2 mN)\left(\tau_{\pi(i)} + \frac{\tau_c}{m}\right)}{M!}. \tag{17}$$

On the other hand we have

$$\sum_{\pi\in\mathcal{P}} \sum_{i=1}^{M} i\tau_{\pi(i)} = \sum_{i=1}^{M} i\left((M-1)! \sum_{i=1}^{M} \tau_i\right) = \sum_{i=1}^{M} i(M!\,\tau) \quad \text{(from (3))}. \tag{18}$$

Hence, substituting (17) by (18) we have

$$T = \sum_{i=1}^{M} \frac{1}{N}(r_0 + r_1 i + r_2 mN)\left(\tau + \frac{\tau_c}{m}\right) \tag{19}$$

$$= \frac{M}{N}\left(r_0 + r_1\frac{M+1}{2} + r_2 mN\right)\left(\tau + \frac{\tau_c}{m}\right). \quad (20)$$

Partially differentiating T by m, we have

$$\frac{dT}{dm} = \frac{M}{N}\left(r_2 N\tau - \frac{\left(r_0 + r_1\frac{M+1}{2}\right)\tau_c}{m^2}\right). \quad (21)$$

Then, we have the minimum of T

$$T_{\min} = \frac{M}{N}\left(\sqrt{\left(r_0 + r_1\frac{M+1}{2}\right)\tau} + \sqrt{r_2 N\tau_c}\right)^2 \quad (22)$$

when

$$m = \sqrt{\frac{\left(r_0 + r_1\frac{M+1}{2}\right)\tau_c}{r_2 N\tau}}. \quad (23)$$

Fig. 3(a)–(c) presents the total amount of processing and communication time as a function of the number of target faults per server for four cases of different parameters. Fig 3(a) shows two cases of $t_0 = t_1 = 0.1$ and $t_0 = t_1 = 0.01$. From this figure we can see that as the communication time decreases, both the optimal granularity and the total processing time decrease. Fig. 3(b) compares two cases of $r_2 = 0.00001$ and 0.000001. From this figure, we can observe that as the r_2 decreases, the total processing time decreases. This is because the factor $r_2 mN$ which accounts for the decrease ratio of newly processed faults due to overlapped processing decreases. From Fig. 3(c) which compares two cases of $r_1 = 0.00001$ and 0.000001, we can see that both the total processing time and the optimal granularity decreases as the value of r_1 decreases. This is because as the value of r_1 decreases, the ratio of newly processed faults increases, and hence, the effect of fault simulation increases.

The parallel test generation system has been implemented on a network (ethernet) of eleven SUN workstations (one SUN3/60 for client and ten SUN3/50's for servers) where the FAN algorithm [2] was used. Fig. 4 gives the curves of total processing time versus the number of target faults for the largest circuit c7552 of the ISCAS'85 benchmark circuits [24] on the network. From this figure, we can see that the shape of the curve coincides closely with those of Fig. 3 obtained from the above analysis under a homogeneous model.

4.2. *Speedup of Multiple Processor Systems*

The *speedup* of a multiple processor system is defined as the ratio of the time required to complete test generation for all faults on a *single* processor to the time required to execute the same process on an N-processor system. Therefore, we have

$$S = \frac{T_{\text{single}}}{T_{\text{multi}}}. \quad (24)$$

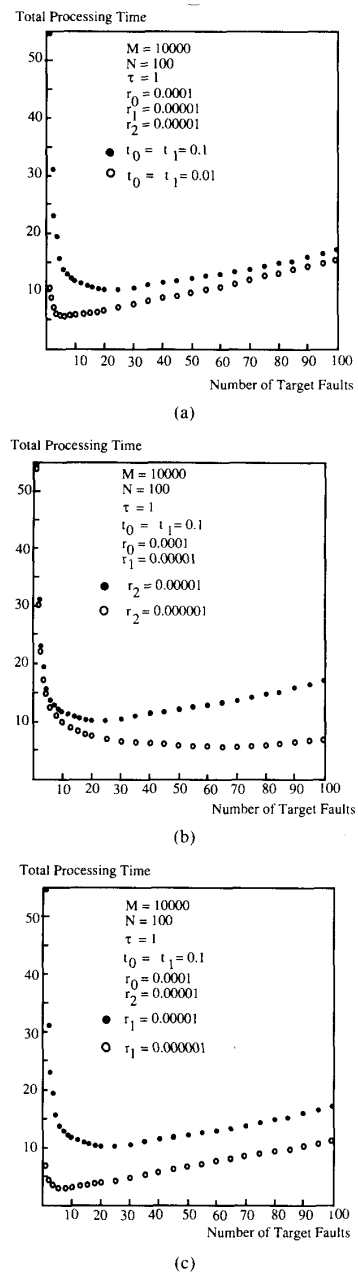


Fig. 3. Computation time versus number of target faults.

T_{single} can be considered as a special case of T_{multi}, and hence, the computation time required to complete test generation for all faults on a single processor can be derived from (19) and (20) by deleting both terms $r_2 mN$ and τ_c/m in (19) and (20). Hence, we have

$$T_{\text{single}} = \sum_{i=1}^{M}(r_0 + r_1 i)\tau$$

$$= M\left(r_0 + r_2\frac{M+1}{2}\right)\tau. \quad (25)$$

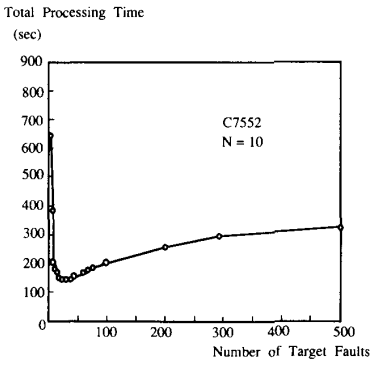


Fig. 4. Experimental result of 10-server system.

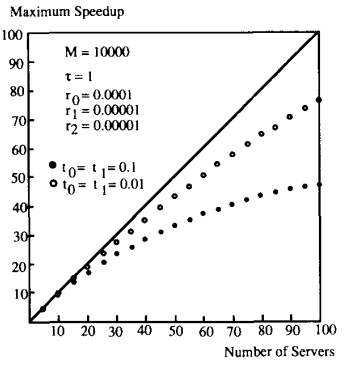


Fig. 5. Maximum speedup ratio versus number of servers.

From (22), (24), and (25), we have the maximum speedup $S_{\max}$ as follows:

$$S_{\max} = \frac{T_{\text{single}}}{T_{\min}} = \frac{N}{\left(1 + \sqrt{\left(\frac{r_2 N}{r_0 + r_1 \frac{M+1}{2}}\right)\left(\frac{t_0 + t_1 N}{\tau}\right)}\right)^2} < N. \tag{26}$$

The above expression indicates that $S_{\max}$ approaches to N if

$$r_2 mN \ll r_0 + r_1 \frac{M+1}{2} \tag{27}$$

and

$$\frac{t_0 + t_1 N}{m} \ll \tau. \tag{28}$$

In other words, if the decrease ratio of newly processed faults due to overlapped processing is much smaller than the ratio of newly processed faults, and if the data transfer time per fault and the waiting time per communication are much smaller than the processing time per fault, then $S_{\max}$ approaches to N.

Fig. 5 shows the maximum speedup ratio as a function of the number of servers for two cases of different parameters of $t_0 = t_1 = 0.1$ and 0.01. The experimental result on the network of fifteen SUN workstations (one SUN3/60 for client and fourteen SUN3/50's for servers) is shown in Fig. 6. The figure gives the curve of the maximum speedup ratio as a function of the number of servers for the largest circuit c7552 of the ISCAS'85 benchmark circuits [24] on the network.

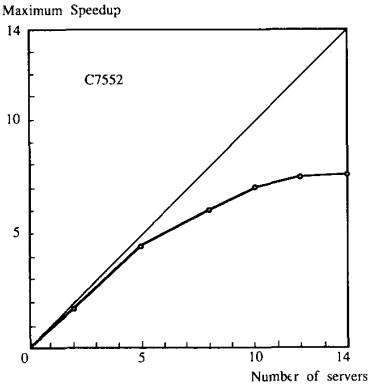


Fig. 6. Experimental result of maximum speedup.

V. Homogeneous Problem with Dynamic Task Allocation

In the previous sections we have considered static task allocation of faults where the number of target faults transferred from the client to a server is always the same. In this section we shall consider *dynamic* task allocation of faults where the number of target faults transferred from the client to each server will vary as time goes on.

Here, we consider again the homogeneous case; i.e., $\tau_{ij} = \tau_i$ and $\delta_{ij} = \delta_i$ for all faults f_i and servers j. Suppose that the number of processed faults is i when fault $f_{\pi(i)}$ is processed where π (π: $\mathfrak{M} \to \mathfrak{M}$) is a permutation of $\mathfrak{M} = \{1, 2, \cdots, M\}$. Let m_i be the number of target faults allocated to a server when i faults have been processed by all servers till then. Then the total amount of processing and communication time T can be obtained by replacing m by m_i in (17) as follows:

$$T = \frac{\sum_{\pi \in \mathcal{P}} \sum_{i=1}^{M} \frac{1}{N}(r_0 + r_1 i + r_2 m_i N)\left(\tau_{\pi(i)} + \frac{\tau_c}{m_i}\right)}{M!} \tag{29}$$

$$= \sum_{i=1}^{M} \frac{1}{N}(r_0 + r_1 i + r_2 m_i N)\left(\tau + \frac{\tau_c}{m_i}\right). \tag{30}$$

Partially differentiating the above expression by m_i, we have

$$\frac{dT}{dm_i} = \frac{M}{N}\left(r_2 N\tau - \frac{(r_0 + r_1 i)\tau_c}{m_i^2}\right). \tag{31}$$

Then, we have the minimum of T for dynamic allocation:

$$T_{\text{dynamic}} = \sum_{i=1}^{M} \frac{1}{N} \left(\sqrt{(r_0 + r_1 i)\tau} + \sqrt{r_2 N \tau_c} \right)^2 \quad (32)$$

when

$$m_i = \sqrt{\frac{(r_0 + r_1 i)\tau_c}{r_2 N \tau}}, \qquad \text{for all } i. \quad (33)$$

From (33), the optimal granularity (the optimal size of target faults) of time t can be expressed as

$$m(t) = \sqrt{\frac{(r_0 + r_1 x_t)\tau_c}{r_2 N \tau}} \quad (34)$$

where x_t is the total number of faults processed by all servers till the time t. Hence, the best performance or the test generation with the minimum computation time will be achieved if the dynamic task allocation is scheduled in accordance with the above expression as follows: the client counts up the total number x_t of processed faults till now (at time t), calculates the number $m(t)$ of target faults from (34), and then allocates $m(t)$ target faults to an idle server. Note that although (34) represents a continuous function, $m(t)$ is defined as an integer.

Let us consider next how much reduction of computation time will be achieved by dynamic task allocation compared with static one. The minimum of T for static allocation is

$$T_{\text{static}} = \frac{M}{N} \left(\sqrt{\left(r_0 + r_1 \frac{M+1}{2} \right)\tau} + \sqrt{r_2 N \tau_c} \right)^2. \quad (35)$$

Hence, the difference between T_{static} and T_{dynamic} is

$$T_{\text{static}} - T_{\text{dynamic}} = \frac{2\sqrt{r_2 N \tau_c \tau}}{N} \sum_{i=1}^{M} \cdot \left(\sqrt{r_0 + r_1 \frac{M+1}{2}} - \sqrt{r_0 + r_1 i} \right) > 0. \quad (36)$$

This equation is always positive for $M > 1$, that is, the dynamic task allocation is always more efficient than the static one.

VI. Conclusions

In this paper, we have presented an approach to parallel processing of test generation for logic circuits in a loosely-coupled distributed network of general purpose computers, and analyzed the effects of the allocation of target faults to processors, the optimal granularity (grain size of target faults), and the speedup ratio of the multiple processor system to a single processor system.

For the homogeneous case such that all servers are uniform and for any fault i the probability that fault i is allocated to server j is equiprobable for all servers j, if the test-pattern generation process is applied to each fault with no fault simulation, the total processing and communication time T becomes a monotone decreasing function of the number of target faults m, and hence, the best performance is obtained when each server receives all target faults only once from the client, i.e., when $m = M/N$. However, it may often occur that a test-pattern generated for one fault can also be a test-pattern for other faults if one apply fault simulation. To analyze this case, we have introduced a ratio of newly processed faults to target faults, and derived the expressions of optimal granularity in both cases of static and dynamic task allocation.

We have also derived an expression of the speedup of a multiple processor system in the homogeneous case. The analysis indicates that the speedup $S_{\max}$ approaches N if the data transfer time per fault and the waiting time per communication are much smaller than the processing time per fault, and if the decrease ratio of newly processed faults due to overlapped processing is much smaller than the ratio of newly processed faults. From this, we can see that the task allocation which minimizes the factor of overlapped processing in the ratio ρ will yield the best performance for a multiple processor system, provided that the parameters associated with the hardware are given, though those parameters are hard to determine in advance.

References

[1] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Comput.*, vol. C-30, pp. 215-222, Mar. 1981.

[2] H. Fujiwara and T. Shimono, "On the acceleration of test pattern generation algorithms," *IEEE Trans. Comput.*, vol. C-32, pp. 1137-1144, Dec. 1983.

[3] M. H. Schulz and E. Auth, "Advanced automatic test pattern generation and redundancy identification techniques," in *Dig. Papers, FTCS-18*, June 1988, pp. 30-35.

[4] O. H. Ibarra and S. K. Sahni, "Polynomially complete fault detection problems," *IEEE Trans. Comput.*, vol. C-24, pp. 242-249, Mar. 1975.

[5] H. Fujiwara and S. Toida, "The complexity of fault detection problems for combinational logic circuits," *IEEE Trans. Comput.*, vol. C-31, pp. 555-560, June 1982.

[6] P. Goel, "Test generation costs analysis and projections," in *Proc. 17th Ann. Design Automation Conf.*, 1980, pp. 77-84.

[7] T. W. Williams and K. P. Parker, "Design for testability—A survey," *Proc. IEEE*, vol. 71, pp. 98-112, Jan. 1983.

[8] W. A. Rogers, J. F. Guzolek, and J. A. Abraham, "Concurrent hierarchical fault simulation: A performance model and two optimizations," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 848-862, Sept. 1987.

[9] G. Pfister, "The Yorktown simulation engine: Introduction," in *Proc. Ann. 19th Design Automation Conf.*, 1982, pp. 51-54.

[10] T. Sasaki, N. Koike, K. Ohmori, and K. Tomita, "HAL: A block level hardware logic simulator," in *Proc. Ann 20th Design Automation Conf.*, 1983, pp. 150-156.

[11] T. Blank, "A survey of hardware accelerators used in computer-aided design," *IEEE Design and Test*, pp. 21-39, Aug. 1984.

[12] B. Milne, "Put the pedal to the metal with simulation accelerators," *Electron. Design*, pp. 39-52, Sept. 1987.

[13] L. T. Smith and R. R. Rezac, "Methodology for and results from the use of hardware logic simulation engine for fault simulation," in *Proc. Int. Test Conf.*, 1984, pp. 224-228.

[14] F. Hirose, K. Takayama, and N. Kawato, "A method of generate tests for combinational logic circuits using an ultra-high-speed logic simulator," in *Proc. Int. Test Conf.*, 1988, pp. 102-107.

[15] G. A. Kramer, "Employing massive parallelism in digital ATPG algorithm," in *Proc. Int. Test Conf.*, 1983, pp. 108-114.

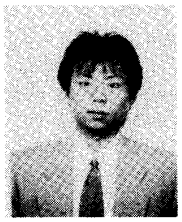
[16] Y. M. Elziq, H. H. Butt, and A. K. Bhatt, "An automatic test pattern generation machine," in *Proc. IEEE Int. Conf. Computer-Aided Design*, 1984, pp. 257-259.
[17] P. A. Duba, R. K. Roy, J. A. Abraham, and W. A. Rogers, "Fault simulation in a distributed environment," in *Proc. 25th Design Automation Conf.*, 1988, pp. 686-691.
[18] A. Motohara, K. Nishimura, H. Fujiwara, and I. Shirakawa, "A parallel scheme for test-pattern generation," in *Proc. IEEE Int. Conf. Computer-Aided Design*, Nov. 1986, pp. 156-159.
[19] H. Fujiwara and A. Motohara, "Fast test pattern generation using a multi-processor system," *Trans. IEICE*, vol. E-71, pp. 441-447, Apr. 1988.
[20] S. J. Chandra and J. H. Patel, "Test generation in a parallel processing environment," in *Proc. IEEE Int. Conf. on Computer Design*, October 1988, pp. 11-14.
[21] S. Shimojo, H. Miyahara, and K. Takashima, "Process assignment on multi-processor with communication contentions," *Trans. IECE*, (in Japanese) vol. J68-D, No. 5, pp. 1049-1056, Mar. 1988.
[22] Y-T. Wang and R. J. T. Morris, "Load sharing in distributed systems," *IEEE Trans. Comput.*, vol. C-34, pp. 204-217, Mar. 1985.
[23] Z. Cvetanovic, "The effects of problem partitioning, allocation, and granularity on the performance of multiple-processor systems," *IEEE Trans. Comput.*, vol. C-36, pp. 421-432, Apr. 1987.
[24] F. Brglez and H. Fujiwara, "A neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN," in *Proc. IEEE Int. Symp. on Circuits and Systems*, Kyoto, Japan, June 5-7, 1985.
[25] L. H. Goldstein, "Controllability/observability analysis of digital circuits," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 685-693, Sept. 1979.
[26] S. Patil and P. Banerjee, "Fault partitioning issues in an integrated parallel test generation/fault simulation environment," in *Proc. Int. Test Conf.*, 1989, pp. 718-726.

Hideo Fujiwara (S'70-M'74-SM'83-F'89) received the B.E., M.E., and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively.

He is currently Professor in the Department of Computer Science, Meiji University, Kawasaki, Japan. 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. His research interests include logic design, design for testability, test pattern generation, fault simulation, built-in self-test, computational complexity, parallel processing, and neural networks for design and test. He is the author of *Logic Testing and Design for Testability* (MIT Press, 1985).

Dr. Fujiwara is the Far East International Editor of *IEEE Design and Test of Computers*. He is a member of the Institute of Electronics, Information, and Communication Engineers of Japan and the Information Processing Society of Japan. He received the IECE Young Engineer Award in 1977.

*



Tomoo Inoue received the B.E. degree in electronics and communication engineering from Meiji University, Kawasaki, Japan, in 1988. He is currently working toward the M.S. degree at the Department of Electrical Engineering, Meiji University.

His research interests include hierarchical fault simulation, parallel processing for test generation, and design for testability.

Mr. Inoue is a student member of the Institute of Electronics, Information, and Communication Engineers of Japan.