Optimal Granularity and Scheme of Parallel Test Generation in a Distributed System

Hideo Fujiwara, *Fellow, IEEE*, and Tomoo Inoue, *Member, IEEE*

***Abstract*—A Client-Agent-Server model (CAS model) which can decrease the work load of the client by adding agent processors to the Client-Server model (CS model) is proposed and an approach to parallel test generation for logic circuits on the CAS model is presented. Two problems are considered: optimal granularity problem and optimal scheme problem. First, the problem of parallel test generation on the CAS model is formulated to analyze the effect of the granularity (grain size of target faults allocated to processors) in both cases of static and dynamic task allocation (optimal granularity problem). Then the relationship between the number of processors and the total processing time is analyzed (optimal scheme problem). From the analysis, it is shown that the CAS model can reduce the total processing time over the CS model and that there exists an optimal scheme (an optimal pair of numbers of agent processors and server processors) for the CAS model which minimizes the total processing time for a given number of processors. To corroborate the analysis, the proposed parallel test generation algorithm is implemented on a network of more than 100 workstations and experimental results for the ISCAS benchmark circuits are presented. It is shown that the experimental results are very close to the theoretical results which confirms the existence of optimal granularity and optimal scheme which minimizes the total processing time for the CAS model.**

***Index Terms*—Combinational circuits, client-server model, distributed systems, fault simulation, granularity, parallel processing, test generation.**

I. Introduction

THEORETICALLY, it has been shown that the problem of test generation for logic circuits is NP-hard [1], [2] even for combinational circuits, and hence it is very difficult to speed up the test generation process due to backtracking mechanism. On the other hand, efficient heuristics to speed up test generation have been proposed [3], [4], [5], but handling the increased logic complexity of VLSI circuits has been severely limited by the slowness of conventional CAD tools on a general purpose computer. Multiprocessing hardware has to be used to get orders of magnitude speed up for those circuits of VLSI or ULSI complexity.

There are several types of parallelism inherent in test-pattern generation: fault parallelism, search parallelism, heuristic parallelism, and topological parallelism [16]. *Fault parallelism* refers to dealing with different faults in parallel. Motohara et al. [7], Patil and Banerjee [12], and Fujiwara and Inoue [10] presented their methods of parallel processing for test generation based on fault parallelism. *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. Motohara et al. [7] and Patil and Banerjee [11] proposed their methods of parallel processing for test generation based on search parallelism. *Heuristic parallelism* refers to dealing with one fault using different heuristics in parallel. Chandra and Patel [8] reported an approach to heuristic parallelism. *Topological parallelism* refers to simulating different subcircuits in parallel. Kramer [6] and Hirose et al. [9] presented their methods of parallel processing for topological parallelism.

Manuscript received Dec. 13, 1993; revised Aug. 3, 1994.

The authors are with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma, Nara 630-01, Japan. e-mail: fujiwara@is.aist-nara.ac.jp.

IEEECS Log Number D95029.

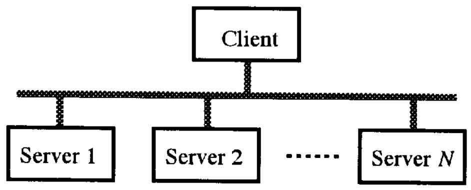
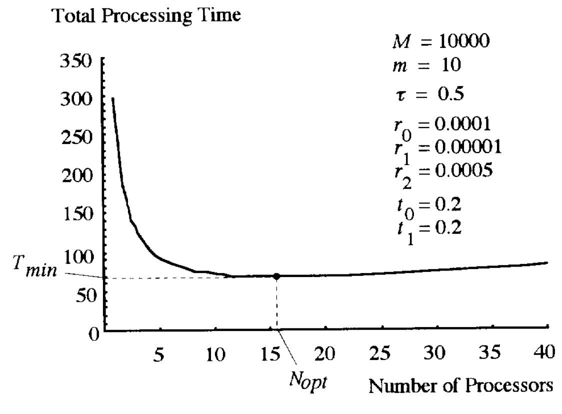


Fig. 1. Architecture of the Client-Server model.

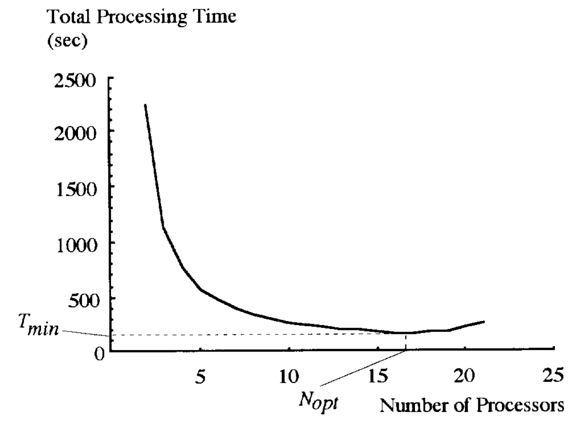
In [10], we presented an approach to parallel test generation based on fault parallelism in a loosely coupled distributed network of general purpose computers and analyzed theoretically the effect of the allocation of target faults to processors using a *Client-Server model (CS model)* illustrated in Fig. 1. We showed the existence of the optimal granularity or the optimal number of target faults allocated to processors which minimizes the total processing time for the CS model. The total processing time T on the CS model can be expressed by

$$T = \frac{M}{N}\left(r_0 + r_1\frac{M+1}{2} + r_2 mN\right)\left(\tau + \frac{t_0 + t_1 N}{m}\right) \qquad (1)$$

where M, N, and m are the total number of faults of a circuit under test, the number of processors, and the granularity, respectively, and τ is the mean (test-generation/fault simulation) processing time per fault, r_0, r_1, r_2 are constants that relate to test-generation/fault simulation, and t_0, t_1 are constants that relate to communication between processors [10]. Fig. 2(a) shows the total processing time versus the number of processors for the case of M = 10,000, m = 10, τ = 0.5, r_0 = 0.0001, r_1 = 0.00001, r_2 = 0.0005, t_0 = 0.2, and t_1 = 0.2. The parallel test generation system [10] was implemented on a network of workstations using the FAN algorithm [4]. Fig. 2(b) gives a curve of total processing time versus the number of processors for the experimental result using the ISCAS89 benchmark circuit s15850 modified into a combinational circuit with full-scan design. From Figs. 2(a) and 2(b) we can see that there exists an optimal number of processors which minimizes the total processing time for the CS model. This implies, even if the number of processors is increased beyond the optimal number for the CS model, higher speedups cannot be achieved.



(a)



(b)

Fig. 2. Total processing time versus number of processors. (a) Analysis. (b) Experimental result for circuit s15850.

In this paper, in order to get a more efficient scheme than the CS model of [10], we propose another model called a *Client-Agent-Server model (CAS model)* which can decrease the work load of the client by adding agent processors to the CS model. The CAS model proposed here is an extension of the CS model. In other words, the CS model can be regarded as a special case of the CAS model that has only *one dummy* agent processor. We extend here the result of [10] to the CAS model. In [10], we considered the *optimal granularity problem* for the CS model. The granularity of the CS model is defined as the number of faults transferred from the client to a server. In case of the CAS model, the granularity is defined as a pair of two numbers; one is the number of faults transferred from the client to an agent and the other is the number of faults transferred from an agent to a server during each communication. The optimal granularity problem for the CAS model is thus to find a pair of these two numbers which minimizes the total processing time. In this paper we consider this optimal granularity problem for the CAS model. There is another important problem, *optimal scheme problem.* The scheme of the CAS model is determined by the number of agents and the number of servers per agent. So, the optimal scheme problem is to find a pair of these two numbers which minimizes the total processing time. In [10], we have not considered the optimal scheme problem for the CS model. Here we consider the optimal scheme problem for the CAS model including the CS model.

We first formulate the problem of test generation for the CAS model and derive the equation of the total processing time as a function of several parameters, including the granularity and the scheme of the CAS model. Then we analyze the equation to solve both of the problems of optimal granularity and optimal scheme. We analyze the effect of the number of faults allocated to a processor to find the optimal granularity in both cases of *static* and *dynamic* task allocation. We analyze the relationship between the number of processors and the total processing time to find the optimal scheme which minimizes the total processing time for a given number of processors. From the analysis, it is shown that the CAS model can reduce the total processing time over the CS model and that there exists an optimal scheme (an optimal pair of numbers of agent processors and server processors) for the CAS model which minimizes the total processing time for a given number of processors. To corroborate the analysis, the proposed parallel test generation algorithm is implemented on a network of more than 100 workstations. We present experimental results for the ISCAS benchmark circuits. It is shown that the experimental results are very close to the theoretical results which confirms the existence of optimal granularity and optimal scheme that minimizes the total processing time for the CAS model.

II. Architecture of the Client-Agent-Server Model

The architecture of our loosely coupled multiple processor systems is illustrated in Fig. 3. This system is derived by inserting agent processors between a client and servers of the CS model. We call it a *CAS model.* This CAS model has a logical hierarchy such that N_a agents are connected to the client, and N_s servers are connected to each agent, though all processors are physically connected to a single communication network. The client requests an agent to execute a task and to return the result. An agent partitions a task into subtasks and distributes each subtask to a server connected to the agent. When a server finishes its assigned task, it sends the result to the agent and requests a new task. After an agent finishes the task from the client, it sends the result to the client and requests a new task. The client saves the result and sends a new task to the agent. This process is repeated until all tasks are processed.

Here if we regard the task as test generation, the above process can be redescribed as follows: The client extracts a number of faults from the fault table as a set of target faults and sends the faults to an agent. When an agent receives the target faults from the client, the agent sends a subset of the target faults to a server connected to the agent as a set of target faults for the server. A server which receives the target faults generates a test-pattern for one of the target faults and finds out all detected faults by the test pattern by performing simulation for all faults in the circuit, not just those in the set of target faults. The server repeats test-pattern generation and fault simulation until all the target faults are processed and then sends the result to the agent. After receiving the result from the server, the agent saves it in its own storage. The agent then sends a new set of target faults which have not yet been processed by any server of the agent and sends it to the server. After all the target faults assigned to the agent are processed, the agent sends the results to the client and requests a new set of target faults. The client updates the fault table and sends new target faults to the agent. This process continues until all faults in the fault table are processed.

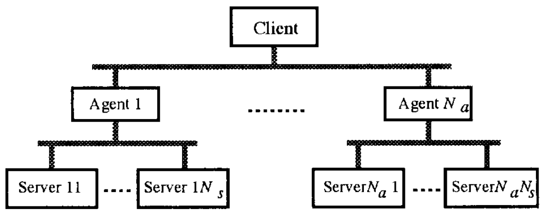


Fig. 3. Architecture of the Client-Agent-Server model.

III. Formulation of the Problem

We formulate the test generation problem for the CAS model. It consists of one client, N_a agents, and N_s servers per agent. Let the kth server connected to the jth agent A_j be server S_{jk}. A process of test-pattern generation for a fault f_i is called *a process for fault f_i*. The parameters used here are defined as follows:

M:	the total number of faults of a given circuit.
τ_{ijk}:	the processing time of server S_{jk} for fault f_i.
δ_{ijk}:	the probability that process for fault f_i is allocated to server S_{jk}.
λ_{aij}:	the probability that agent A_j communicates to the client after process for fault f_i.
λ_{sijk}:	the probability that server S_{jk} communicates to agent A_j after process for fault f_i.
τ_{ca}:	the mean communication time which includes waiting time due to contention and data transfer time between the client and agents.
τ_{cs}:	the mean communication time which includes waiting time due to contention and data transfer time between an agent A_j and servers.

Then, the average time necessary to complete all processes allocated to server S_{jk} is

$$T_{jk} = \sum_{i=1}^{M} \delta_{ijk}(\tau_{ijk} + \lambda_{aij}\tau_{ca} + \lambda_{sijk}\tau_{cs}). \tag{2}$$

The time necessary to complete all processes is defined by the maximum of T_{jk}:

$$T = \max\{T_{jk}\} \tag{3}$$

Here, we consider two problems: *optimal granularity problem* and *optimal scheme problem*. The granularity is the grain size of target faults transferred between two processors. In case of the CAS model, the granularity is a pair of two numbers (m_a, m_s) where m_a is the number of faults transferred from the client to an agent and m_s is the number of faults transferred from an agent to a server during each communication. Note that these values m_a and m_s will vary in *dynamic* task allocation strategy. The *optimal granularity problem* is thus to find a pair of two numbers (m_a, m_s) which minimizes the processing time T of (3). In other words, it is to find a task allocation schedule which minimizes T. On the other hand, the scheme of the CAS model is determined by the number of agents, N_a, and the number of servers per agent, N_s. So, the *optimal scheme problem* is to find a pair of two numbers (N_a, N_s) which minimizes the processing time T.

IV. Optimal Granularity With Static Task Allocation

First we consider *static* task allocation of faults where the numbers of target faults from the client to an agent and from an agent to a server are always constant, respectively.

A. Assumption of Homogeneous Problem

To obtain the minimum processing time on the CAS model, it is important to equalize the load of each server. Here, we shall assume a *homogeneous* case as follows:

1) All servers are uniform, i.e.,

$$\tau_{ijk} = \tau_i \tag{4}$$

for all faults f_i and servers S_{jk}.

2) For any fault f_i the probability that fault f_i is allocated to a server S_{jk} is independent of the server S_{jk}, i.e.,

$$\delta_{ijk} = \delta_i \tag{5}$$

for all faults f_i and servers S_{jk}.

B. Communication Probability: λ_{aij}, λ_{sijk}

Let m_a be the number of target faults transferred from the client to an agent A_j during each communication. Suppose that fault f_i is in the set of m_a target faults allocated to the agent A_j. Then the probability that the agent A_j communicates to the client after process for fault f_i is

$$\lambda_{aij} = \frac{1}{m_a} \tag{6}$$

since such a communication occurs only once for those m_a faults.

Let m_S be the number of target faults transferred from an agent A_j to a server S_{jk} during each communication. Suppose that fault f_i is in the set of m_s target faults allocated to the server S_{jk} from the agent A_j. Then the probability that the server S_{jk} communicates to the agent A_j after process for fault f_i is

$$\lambda_{sijk} = \frac{1}{m_s} \tag{7}$$

since such a communication occurs only once for those m_s faults.

C. Probability of Process Allocation: δ_{ijk}

Suppose that the client requests an agent to process m_a target faults. The agent extracts m_s faults from the m_a target faults and requests a server to process the m_s target faults. Note that $m_s \le m_a$. The server generates a test pattern for one of the m_s faults and seeks all the faults detectable by the generated test pattern by performing fault simulation for all faults in the circuit, not just those in the set of m_s target faults. It repeats test-pattern generation and fault simulation until all target faults are processed. Let ρm_s be the number of faults that are newly found to be detectable or redundant at completion of of test generation for m_s target faults. Let us call those faults *newly processed faults.*

Let us define the ratio of newly processed faults to target faults:

$$\rho = \frac{\text{number of newly processed faults per server } (\rho m_s)}{\text{number of target faults per server } (m_s)}. \quad (8)$$

Note that this ratio will decrease as the number of processed faults increases. Here, let ρ_i be the ratio when fault f_i is processed.

During each iteration of the server process, m_s target faults are processed by the server and ρm_s 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 f_i is by some server is

$$\frac{m_s}{\rho_i m_s} = \frac{1}{\rho_i} \quad (9)$$

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 f_i is processed by some server is defined by

$$\sum_{j=1}^{N_a}\sum_{k=1}^{N_s}\delta_{ijk}. \quad (10)$$

Therefore, we have

$$\sum_{j=1}^{N_a}\sum_{k=1}^{N_s}\delta_{ijk} = \frac{1}{\rho_i}. \quad (11)$$

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

$$\sum_{j=1}^{N_a}\sum_{k=1}^{N_s}\delta_{ijk} = \sum_{j=1}^{N_a}\sum_{k=1}^{N_s}\delta_i = N_a N_s \delta_i. \quad (12)$$

Hence, we have

$$\delta_{ijk} = \delta_i = \frac{1}{N_a N_s \rho_i}. \quad (13)$$

D. Ratio of Newly Processed Faults to Target Faults: ρ

The number of newly processed faults will quickly decrease as the number of processed faults increases. Further, the number of newly processed faults per fault will decrease as the number of target faults per server and the number of servers increase. In [10], we assumed the ratio of newly processed faults to target faults for the CS model to be

$$\rho(x) = \frac{1}{r_0 + r_1 x + r_2 m N} \quad (14)$$

where m is the number of target faults to a server per communication, N is the number of servers, x is the number of processed faults, and r_0, r_1, and r_2 are constants. In this expression, the factor $1 / (r_0 + r_1 x)$ expresses the effect of fault simulation, and the factor $r_2 m N$ accounts for the decrease ratio of newly processed faults due to overlapped processing.

About the factor for decrease ratio of newly processed faults on the CAS model, we have to consider the overlapped processing among agents, in addition to the overlapped processing among servers. After receiving the list of the result from a server, an agent renews its own fault table, which is the copy from the client. Since multiple agents are working simultaneously, some agents may save the same faults detected by servers. These overlapped processes will increase and hence ρ_i will decrease as the number of target faults per agent (m_a) and the number of agents (N_a) increase. By introducing this factor ($m_a N_a$) into (14), we have

$$\rho(x) = \frac{1}{r_0 + r_1 x + r_2 m_s N_s + r_3 m_a N_a} \quad (15)$$

where r_0, r_1, r_2, and r_3 are constants. In the above expression, the factor $r_3 m_a N_a$ accounts for the decrease ratio due to the overlapped processing among agents.

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

$$\rho_{\pi(i)} = \frac{1}{r_0 + r_1 i + r_2 m_s N_s + r_3 m_a N_a}. \quad (16)$$

E. Communication Time: τ_{ca}, τ_{cs}

Here we have the following assumptions:

1) The size of data (fault table) transferred between the client and an agent or between an agent and a server is fixed, and hence, the data transfer time during communication between the client and agents or between an agent and servers is a constant.

2) All agents communicate with the client through a single communication network. All servers also communicate with respective agents through the same network. Agents and servers cannot consequently communicate while one of the other processors communicates. Hence, the waiting time during communication between the client and an agent or between an agent and a server is proportional to the number of agents plus the total number of servers, i.e., $N_a + N_a N_s$.

3) After receiving the result from an agent, the client updates the fault table and sends a new set of target faults. This work load increases in proportion to the number of agents, N_a. Hence, the waiting time during working of the client is proportional to the N_a, and the work load of an agent increases in proportion to the number of the servers connected to the agent,

N_s. Hence, the waiting time during working of an agent is proportional to the N_s.

From the above assumptions, we have

$$\tau_{ca} = t_{a0} + t_{a1}N_a(N_s + 1) + t_{a2}N_a \qquad (17)$$

where t_{a0}, t_{a1}, and t_{a2} are constants. And we have

$$\tau_{cs} = t_{s0} + t_{s1}N_a(N_s + 1) + t_{s2}N_s \qquad (18)$$

where t_{s0}, t_{s1}, and t_{s2} are constants.

Here we assume $t_{a0} = t_{s0} = t_0$, $t_{a1} = t_{s1} = t_1$, and $t_{a2} = t_{s2} = t_2$. Then we have

$$\tau_{ca} = t_0 + t_1N_a(N_s + 1) + t_2N_a \qquad (19)$$

and

$$\tau_{cs} = t_0 + t_1N_a(N_s + 1) + t_2N_s \qquad (20)$$

where t_0, t_1, and t_2 are constants.

F. Total Processing Time: T

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

From (2), (13), (16), (19), and (20), we can derive the average of total processing time for all permutations:

$$T = \frac{\sum_{\pi\in P}\sum_{i=1}^{M} \frac{1}{N_aN_s}(r_0 + r_1 i + r_2 m_s N_s + r_3 m_a N_a)\left(\tau_i + \frac{\tau_{ca}}{m_a} + \frac{\tau_{cs}}{m_s}\right)}{M!}. \qquad (21)$$

On the other hand we have

$$\sum_{\pi\in P}\sum_{i=1}^{M} i\tau_{\pi(i)} = \sum_{i=1}^{M} i\left((M-1)!\sum_{i=1}^{M}\tau_i\right). \qquad (22)$$

Substituting the mean processing time for each fault:

$$\tau = \frac{1}{M}\sum_{i=1}^{M}\tau_i \qquad (23)$$

into the right side of (22), we have

$$\sum_{\pi\in P}\sum_{i=1}^{M} i\tau_{\pi(i)} = \sum_{i=1}^{M} i(M!\tau). \qquad (24)$$

Hence, from (21) and (24) we have

$$T = \sum_{i=1}^{M}\frac{1}{N_aN_s}(r_0 + r_1 i + r_2 m_s N_s + r_3 m_a N_a)\left(\tau + \frac{\tau_{ca}}{m_a} + \frac{\tau_{cs}}{m_s}\right) \qquad (25)$$

$$= \frac{M}{N_aN_s}\left(r_0 + r_1\frac{M+1}{2} + r_2 m_s N_s + r_3 m_a N_a\right)\left(\tau + \frac{\tau_{ca}}{m_a} + \frac{\tau_{cs}}{m_s}\right). \qquad (26)$$

From (26) we can see that if we can reduce the processing time for each fault (τ) and/or the communication time (τ_{ca} and τ_{cs}), we can decrease the total processing time (T). We can also observe that as the r_0, r_1, r_2, and r_3, decrease, the total processing time decreases. This is because the term

$$\left(r_0 + r_1\frac{M+1}{2} + r_2 m_s N_s + r_3 m_a N_a\right)$$

accounts for the decrease ratio of newly processed faults due to the overlapped processing among servers and agents as mentioned earlier (see (16)). If we can reduce this type of overlapped processing, we can increase the effect of fault simulation and hence decrease the total processing time. Furthermore, since the total processing time T is a function of granularity (m_a and m_s) and scheme (N_a and N_s) in (26), we can reduce the total processing time by choosing appropriate values for the granularity and scheme. In the following section we shall consider the effect of scheme or the numbers of agents and servers per agent. Here we consider the effect of granularity (m_a and m_s)

In (26), partially differentiating T by m_s, we have

$$\frac{\partial T}{\partial m_s} = \frac{M}{N_aN_s}\left(r_2N_s\left(\tau + \frac{\tau_{ca}}{m_a}\right) - \frac{\left(r_0 + r_1\frac{M+1}{2} + r_3 m_a N_a\right)\tau_{cs}}{m_s^2}\right). \qquad (27)$$

Then, we have

$$T_{s\min} = \frac{M}{N_aN_s}\left(\sqrt{r_2N_s\tau_{cs}} + \sqrt{\left(r_0 + r_1\frac{M+1}{2} + r_3 m_a N_a\right)\left(\tau + \frac{\tau_{ca}}{m_a}\right)}\right)^2 \qquad (28)$$

when

$$m_{s\text{opt}} = \sqrt{\frac{\left(r_0 + r_1\frac{M+1}{2} + r_3 m_a N_a\right)\tau_{cs}}{r_2N_s\left(\tau + \frac{\tau_{ca}}{m_a}\right)}}. \qquad (29)$$

Partially differentiating $T_{s\min}$ by m_a, we have

$$\frac{\partial T_{s\min}}{\partial m_a} = \frac{M}{N_aN_s}\left(1 + \sqrt{\frac{r_2N_s\tau_{cs}}{\left(r_0 + r_1\frac{M+1}{2} + r_3 m_a N_a\right)\left(\tau + \frac{\tau_{ca}}{m_a}\right)}}\right)\left(r_3N_a\tau - \frac{\left(r_0 + r_1\frac{M+1}{2}\right)\tau_{ca}}{m_a^2}\right). \qquad (30)$$

Then, we have the minimum of T

$$T_{\min} = \frac{M}{N_aN_s}\left(\sqrt{r_2N_s\tau_{cs}} + \sqrt{r_3N_a\tau_{ca}} + \sqrt{\left(r_0 + r_1\frac{M+1}{2}\right)\tau}\right)^2 \qquad (31)$$

when

$$m_{a\text{opt}} = \sqrt{\frac{\left(r_0 + r_1\frac{M+1}{2}\right)\tau_{ca}}{r_3N_a\tau}} \qquad (32)$$

and

$$m_{s\text{opt}} = \sqrt{\frac{\left(r_0 + r_1\frac{M+1}{2}\right)\tau_{cs}}{r_2N_s\tau}}, \qquad (33)$$

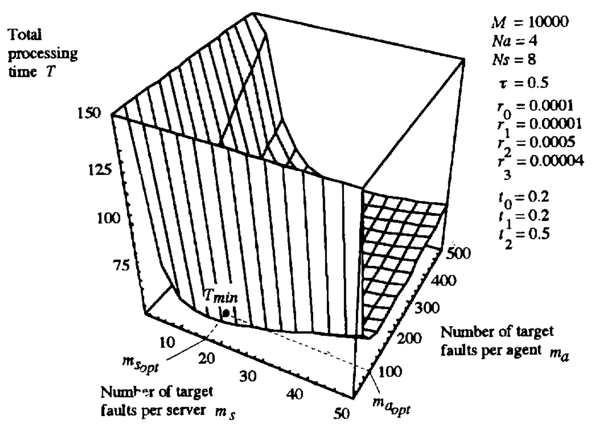


Fig. 4. Total processing time versus granularity: analysis.

which is derived from (29) by substituting m_a in the expression for $m_{a\text{opt}}$.

The analysis so far tries to take into account the existence of an optimal granularity or a pair of two numbers (m_a and m_s) which minimizes the total processing time T. To see the characteristic of (31), (32), and (33), consider Fig. 4 which shows the 3D graph of the total processing time T as a function of the number of target faults for an agent, m_a, and the number of target faults for a server, m_s, for the case of M = 10,000, N_a = 4, N_s = 8, τ = 0.5, r_0 = 0.0001, r_1 = 0.00001, r_2 = 0.0005, r_3 = 0.00004, t_0 = 0.2, t_1 = 0.2, and t_2 = 0.5. From Fig. 4 we can see that there exists an optimal number of target faults for an agent, $m_{a\text{opt}}$, and an optimal number of target faults for a server, $m_{s\text{opt}}$, which minimize the total processing time. These equations, (31), (32), and (33) for the CAS model, are extensions of the equation for the CS model of [10]. Hence the CAS model is an extension of the CS model, i.e., the CS model can be regarded as a special case of the CAS model that has only *one dummy* agent processor.

G. Experimental Results

The parallel test generation system of the CAS model was implemented on a network (Ethernet) of 140 workstations (SUN4/LCs). The FAN algorithm [4] was used as a test-pattern generator. Figs. 5 (a), (b), and (c) give the 3D graphs of the total processing time T as a function of the number of target faults for an agent, m_a, and the number of target faults for a server, m_s, for circuits s9234, s13207, and s15850, respectively, of the ISCAS'89 benchmark circuits [15] modified into combinational circuits by assuming full-scan design. Circuits s9234, s13207, and s15850 have 9234, 13207, and 15850 signal lines, respectively. In these figures, we can see that the shape of the graphs coincides closely with that of Fig. 4 obtained from the above analysis and there exists an optimal granularity pair which minimizes the total processing time. The optimal points are (m_a = 100, m_s = 25), (m_a = 100, m_s = 15), and (m_a = 100, m_s = 15) for circuits s9234, s13207 and s15850, respectively.

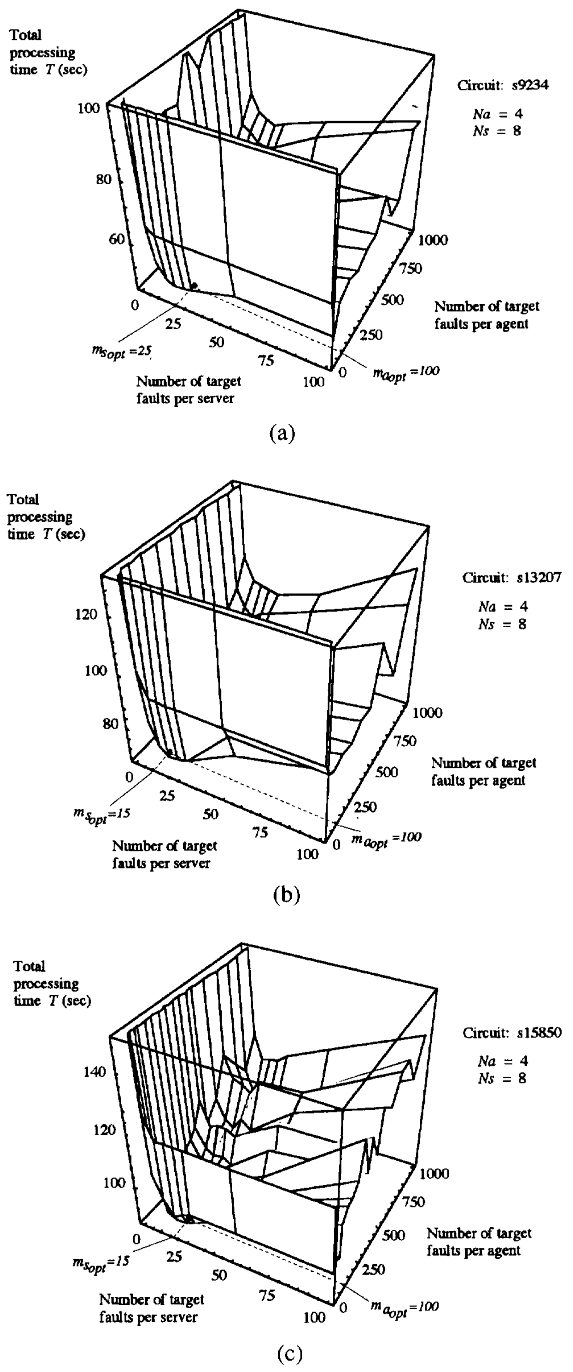


Fig. 5. Total processing time versus granularity: experimental results. (a) Circuit s9234. (b) Circuit s13207. (c) Circuit s15850.

V. Optimal Granularity With Dynamic Task Allocation

In this section we shall consider *dynamic* task allocation of faults where the numbers of target faults for an agent and for a server will vary as time goes on.

Here, we consider again the homogeneous case; i.e., $\tau_{ijk} = \tau_i$ and $\delta_{ijk} = \delta_i$ for all faults f_i and servers S_{jk}. Suppose that the number of processed faults is i when fault $f_{\pi(i)}$ is processed where π is a permutation of $I_M = \{1, 2, \ldots, M\}$. Let m_{ai} and m_{si} be the numbers of target faults allocated to an agent and a server, respectively, when i faults have been processed by all servers till then. Then the average of total processing time T can be obtained by replacing m_a by m_{ai} and m_s by m_{si} in (21) as follows:

$$T = \frac{\displaystyle\sum_{\pi \in P}\sum_{i=1}^{M} \frac{1}{N_a N_s}(r_0 + r_1 i + r_2 m_{si} N_s + r_3 m_{ai} N_a)\left(\tau + \frac{\tau_{ca}}{m_{ai}} + \frac{\tau_{cs}}{m_{si}}\right)}{M!} \tag{34}$$

$$= \sum_{i=1}^{M} \frac{1}{N_a N_s}(r_0 + r_1 i + r_2 m_{si} N_s + r_3 m_{ai} N_a)\left(\tau + \frac{\tau_{ca}}{m_{ai}} + \frac{\tau_{cs}}{m_{si}}\right). \tag{35}$$

Partially differentiating the above expression by m_{si}, we have

$$\frac{\partial T}{\partial m_{si}} = \frac{M}{N_a N_s}\left(r_2 N_s\left(\tau + \frac{\tau_{ca}}{m_{ai}}\right) - \frac{(r_0 + r_1 i + r_3 m_{ai} N_a)\tau_{cs}}{m_{si}^2}\right). \tag{36}$$

Then, we have

$$T_{s\min} = \sum_{i=1}^{M} \frac{1}{N_a N_s}\left(\sqrt{r_2 N_s \tau_{cs}} + \sqrt{(r_0 + r_1 i + r_3 m_a N_a)\left(\tau + \frac{\tau_{ca}}{m_a}\right)}\right)^2 \tag{37}$$

when

$$m_{si} = \sqrt{\frac{(r_0 + r_1 i + r_3 m_a N_a)\tau_{cs}}{r_2 N_s\left(\tau + \frac{\tau_{ca}}{m_a}\right)}}. \tag{38}$$

Partially differentiating $T_{s\min}$ by m_{ai}, we have

$$\frac{\partial T_{s\min}}{\partial m_{ai}} = \frac{M}{N_a N_s}\left(1 + \sqrt{\frac{r_2 N_s \tau_{cs}}{(r_0 + r_1 i + r_3 m_{ai} N_a)\left(\tau + \frac{\tau_{ca}}{m_{ai}}\right)}}\right)\left(r_3 N_a \tau - \frac{(r_0 + r_1 i)\tau_{ca}}{m_{ai}^2}\right). \tag{39}$$

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

$$T_{\text{dynamic}} = \sum_{i=1}^{M} \frac{1}{N_a N_s}\left(\sqrt{r_2 N_s \tau_{cs}} + \sqrt{r_3 N_a \tau_{ca}} + \sqrt{(r_0 + r_1 i)\tau}\right)^2 \tag{40}$$

when

$$m_{ai} = \sqrt{\frac{(r_0 + r_1 i)\tau_{ca}}{r_3 N_a \tau}} \tag{41}$$

and

$$m_{si} = \sqrt{\frac{(r_0 + r_1 i)\tau_{cs}}{r_2 N_s \tau}}, \quad \text{for all } i. \tag{42}$$

From (41) and (42), the optimal granularity (the optimal size of target faults) of time t can be expressed as

$$m_a(t) = \sqrt{\frac{(r_0 + r_1 x_t)\tau_{ca}}{r_3 N_a \tau}} \tag{43}$$

and

$$m_s(t) = \sqrt{\frac{(r_0 + r_1 x_t)\tau_{cs}}{r_2 N_s \tau}} \tag{44}$$

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 $m_a(t)$ of target faults from (43), and then allocates $m_a(t)$ target faults with the number x_t to an agent. The agent calculates the number $m_s(t)$ of target faults from (44), picks the $m_s(t)$ target faults out of the $m_a(t)$ target faults, and then allocates the $m_s(t)$ target faults to an idle server. Note that although (43) and (44) represent continuous functions, $m_a(t)$ and $m_s(t)$ are respectively defined as integers.

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

$$T_{\text{static}} = \frac{M}{N_a N_s}\left(\sqrt{r_2 N_s \tau_{cs}} + \sqrt{r_3 N_a \tau_{ca}} + \sqrt{\left(r_0 + r_1 \frac{M+1}{2}\right)\tau}\right)^2. \tag{45}$$

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

$$T_{\text{static}} - T_{\text{dynamic}} = \frac{2\sqrt{\tau}\left(\sqrt{r_2 N_s \tau_{cs}} + \sqrt{r_3 N_a \tau_{ca}}\right)}{N_a N_s}\sum_{i=1}^{M}\left(\sqrt{\left(r_0 + r_i \frac{M+1}{2}\right)\tau} - \sqrt{(r_0 + r_1 i)\tau}\right). \tag{46}$$

This equation is always positive for $M > 1$, that is, the dynamic task allocation is always more efficient than the static one. This implies that the dynamic task allocation can be better than the static one though the dynamic case requires extra calculation of $m_a(t)$ and $m_s(t)$ which vary as time goes on.

VI. Optimal Scheme for Parallel Test Generation

So far, we have considered the optimal granularity problem of parallel test generation on the CAS model. Here, we consider the optimal scheme problem, i.e., the relation between the number of processors and the total processing time, and show the existence of an optimal scheme for a given number of processors.

The equation (26) of the total processing time T can be also expressed as

$$T = \frac{M}{N_a N_s}(c_{a0} N_a + c_{a1})(c_{a2} N_a + c_{a3}) \tag{47}$$

where

$$c_{a0} = r_3 m_a,$$

$$c_{a1} = r_0 + r_1 \frac{M+1}{2} + r_2 m_s N_s,$$

$$c_{a2} = \left(\frac{1}{m_a} + \frac{1}{m_s}\right) t_1 (N_s + 1) + \frac{t_2}{m_a},$$

and

$$c_{a3} = \tau + \left(\frac{1}{m_a} + \frac{1}{m_s}\right) t_0 + \frac{t_2}{m_s} N_s \,. \tag{48}$$

Partially differentiating T by N_a, we have

$$\frac{\partial T}{\partial N_a} = \frac{M}{N_s}\left(c_{a0}c_{a2} - \frac{c_{a1}c_{a3}}{N_a^2}\right). \tag{49}$$

Then, we have the minimum of T,

$$T_{a\min} = \frac{M}{N_s}\left(\sqrt{c_{a0}c_{a3}} + \sqrt{c_{a1}c_{a2}}\right)^2 \tag{50}$$

when

$$N_{a\mathrm{opt}} = \sqrt{\frac{c_{a1}c_{a3}}{c_{a0}c_{a2}}} \,. \tag{51}$$

The expression (26) can be also expressed as

$$T = \frac{M}{N_a N_s}(c_{s0}N_s + c_{s1})(c_{s2}N_s + c_{s3}) \tag{52}$$

where

$$c_{s0} = r_2 m_s,$$

$$c_{s1} = r_0 + r_1\frac{M+1}{2} + r_3 m_a N_a,$$

$$c_{s2} = \left(\frac{1}{m_a} + \frac{1}{m_s}\right) t_1 N_a + \frac{t_2}{m_s},$$

and

$$c_{s3} = \tau + \left(\frac{1}{m_a} + \frac{1}{m_s}\right)(t_0 + t_1 N_a) + \frac{t_2}{m_a} N_a \,. \tag{53}$$

Partially differentiating T by N_s, we have

$$\frac{\partial T}{\partial N_s} = \frac{M}{N_a}\left(c_{s0}c_{s2} - \frac{c_{s1}c_{s3}}{N_s^2}\right). \tag{54}$$

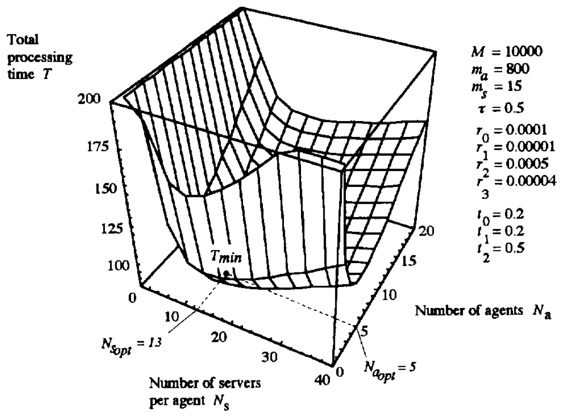
Then, we have the minimum of T,

$$T_{s\min} = \frac{M}{N_a}\left(\sqrt{c_{s0}c_{s3}} + \sqrt{c_{s1}c_{s2}}\right)^2 \tag{55}$$

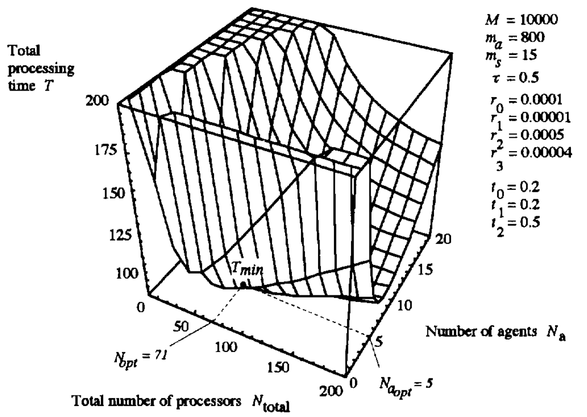
when

$$N_{s\mathrm{opt}} = \sqrt{\frac{c_{s1}c_{s3}}{c_{s0}c_{s2}}} \,. \tag{56}$$

To see the characteristic of (47), (50), and (55), consider Fig. 6 (a) which shows the 3D graph of the total processing time, T, as a function of the number of servers per agent, N_s, and the number of agents, N_a, i.e., $T = T(N_s, N_a)$, for the case of M = 10,000, m_a = 800, m_s = 15, τ = 0.5, r_0 = 0.0001, r_1 = 0.00001, r_2 = 0.0005, r_3 = 0.00004, t_0 = 0.2, t_1 = 0.2, and t_2 = 0.5. In this figure, there exists an optimal point (N_s = 13, N_a = 5) which minimizes the total processing time among all schemes obtained by varying the numbers of N_s and N_a. The CAS model is an extension of the CS model, i.e., the CS model can be regarded as a special case of the CAS model that has only *one dummy* agent processor. Hence, in Fig. 6(a) the CS model corresponds to the curve when N_a = 1. Obviously, the curve of N_a = 1 is worse than others and hence the CS model is worse than the CAS model.



(a)



(b)

Fig. 6. Total processing time versus number of processors: analysis. (a) $T(N_s, N_a)$. (b) $T(N_{total}, N_a)$.

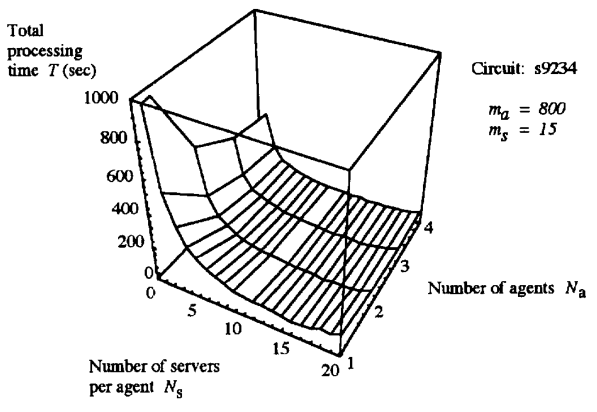
The total number of processors on the CAS model is

$$N_{total} = 1 + N_a + N_a N_s = 1 + N_a(N_s + 1). \tag{57}$$

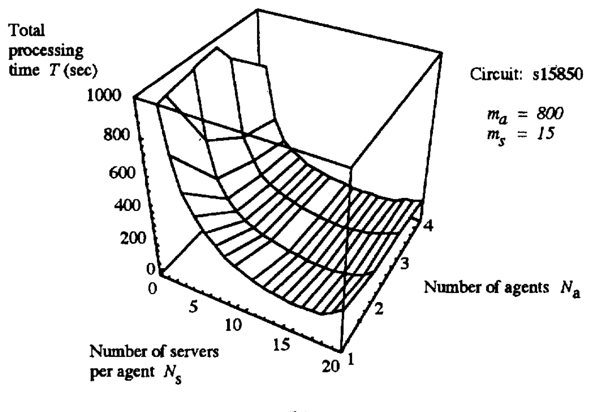
Fig. 6 (b) shows the 3D graph of the total processing time, T, as a function of the total number of processors, N_{total}, and the number of agents, N_a, i.e., $T = T(N_{total}, Na)$ for the above case. In this figure, the optimal point is (N_{total} = 71, N_a = 5). Similarly as mentioned above, the CS model corresponds to the curve when N_a = 1 which is worse than others as seen in Fig. 6 (b). The curves go down as N_a increases, and it becomes minimum when N_a = 5. Hence, we can easily see that the CS model which is the CAS model with N_a = 1 is generally worse than the CAS model, i.e., the new CAS model can reduce the overall processing time over the previous CS model.

A. Experimental Results

The experimental results for ISCAS'89 benchmark circuits are given in Figs. 7 and 8. Fig. 7 shows the 3D graphs of the total processing time, T, as a function of the number of servers per agent, N_s, and the number of agents, N_a, for ISCAS'89 benchmark circuits s9234 and s15850. Circuits s9234 and

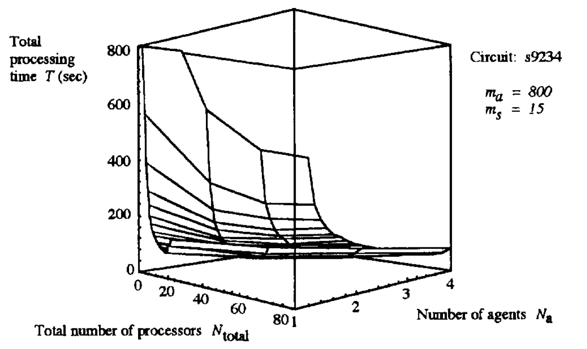


(a)

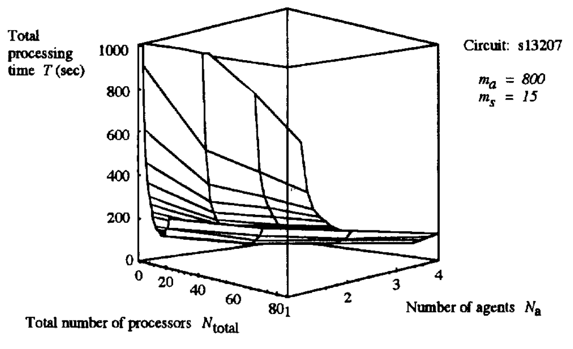


(b)

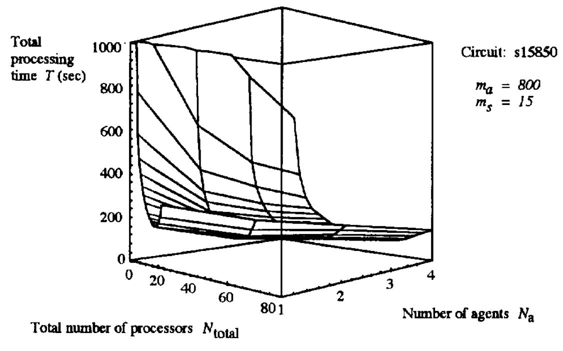
Fig. 7. Total processing time versus number of processors: experimental results. (a) Circuit s9234. (b) Circuit s15850.



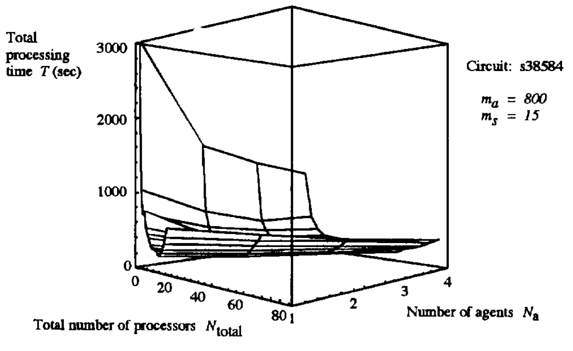
(a)



(b)



(c)



(d)

Fig. 8. Total processing time versus number of processors: experimental results. (a) Circuit s9234. (b) Circuit s13207. (c) Circuit s15850. (d) Circuit s38584.

s15850 have 9234 and 15850 signal lines, respectively. As seen in Fig. 7, for the direction of the axis of N_s, the curves go down and then go up slightly, as the number of servers per agent, N_s, increases. For the direction of the axis of N_a, the curves go down. However, due to the lack of the number of processors, we cannot show here that the curves go up as the number of agents, N_a, increases.

Fig. 8 shows the 3D graphs of the total processing time, T, as a function of the total number of processors, N_{total}, and the number of agents, N_a, for four circuits s9234, s13207, s15850, and s388584. In Fig. 8, we can see that as the total number of processors, N_{total}, increases, the curves go down and then go up slightly. However, due to the lack of the number of processors, we cannot also show here that the curves go up as the number of agents, N_a, increases.

In these figures, we can see that, for the interval with a small number of agents, all the shapes of the 3D graphs in Figs. 7 and 8 generally coincide with those of Fig. 6 obtained from the above analysis. Hence we can expect that there exists an optimal number of processors or an optimal scheme which minimizes the total processing time.

VII. Conclusions

In this paper we presented an approach to parallel processing based on fault parallelism for test generation in a loosely coupled distributed network of general-purpose processors. In order to get a more efficient scheme than the CS model, we proposed another model called a CAS model which can decrease the work load of the client by adding agent processors to the CS model.

We considered two problems for the CAS model: optimal granularity problem and optimal scheme problem. We formulated the problem of test generation for the CAS model and analyzed 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. We analyzed the relationship between the number of processors and the total processing time and showed that there exists an optimal scheme for the CAS model which minimizes the total processing time for a given number of processors. We showed the new model (CAS model) can reduce the overall processing time over the previous model (CS model).

We presented experimental results based on an implementation of our CAS model on a network of more than 100 workstations using the ISCAS'89 benchmark circuits. The experimental results are very close to the theoretical results which confirms the existence of optimal granularity and optimal scheme that minimizes the total processing time for benchmark circuits.

Acknowledgment

The authors would like to thank Tomonori Yonezawa for his technical help and discussions on the experimental results.

References

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

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

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

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

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

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

[7] A. Motohara, K. Nishimura, H. Fujiwara, and I. Shirakawa, "A parallel scheme for test pattern generation," *Proc. IEEE Int'l Conf. Computer-Aided Design*, pp.156-159, 1986.

[8] S.J. Chandra and J.H. Patel, "Test generation in a parallel processing environment," *Proc. IEEE Int'l Conf. Computer Design*, pp. 11-14, 1988.

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

[10] H. Fujiwara and T. Inoue, "Optimal granularity of test generation in a distributed system," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 8, pp. 885-892, Aug. 1990.

[11] S. Patil and P. Banerjee, "A parallel branch-and-bound algorithm for test generation," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 3, pp. 313-322, Mar. 1990.

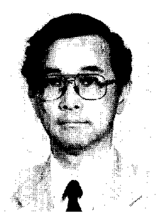
[12] S. Patil and P. Banerjee, "Performance trade-offs in a parallel test generation/fault simulation environment," *IEEE Trans. Computer-Aided Design*, vol. 10, no. 12, pp. 1542-1558, Dec. 1991.

[13] S. Patil, P. Banerjee, and J.H. Patel, "Parallel test generation for sequential circuits on general-purpose multiprocessors," *Proc. 28th ACM/IEEE Design Automation Conf.*, pp.155-159, 1991.

[14] 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.

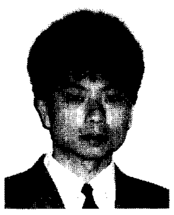
[15] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," *Proc. Int'l Symp. Circuits and Systems*, pp. 1929-1934, 1989.

[16] R.H. Klenke, R.D. Williams, and J.H. Aylor, "Parallel-processing techniques for automatic test pattern generation," *IEEE Computer*, pp. 71-84, Jan. 1992.



Hideo Fujiwara was born in Nara, Japan. He received his BE, ME, and PhD degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively. He was with Osaka University from 1974 to 1985, Meiji University from 1985 to 1993, and joined Nara Institute of Science and Technology in 1993. In 1982 he was a Visiting Research Assistant Professor at the University of Waterloo, Canada, and in 1984 he was a Visiting Associate Professor at McGill University, Canada. Presently he is a professor at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan.

Prof. Fujiwara's research interests are in the design and test of computers, including design for testability, built-in self-test, test pattern generation, fault simulation, computational complexity, parallel processing, neural networks, and expert systems for design and test. He is the author of *Logic Testing and Design for Testability* (MIT Press, 1985). He is a fellow of the IEEE as well as 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 and the IEEE Computer Society Certificate of Appreciation Award in 1991.



Tomoo Inoue was born in Tokyo, Japan. He received his BE degree in electric and communication engineering and his ME degree in electrical engineering from Meiji University, Kawasaki, Japan, in 1988 and 1990, respectively. From 1990 to 1992, he was engaged in the research and development of microprocessors at Matsushita Electric Industrial Co. Ltd., Osaka, Japan. Since 1993 he has been a research associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan.

His research interests include test generation, design for testability, and parallel processing. Mr. Inoue is a member of IEEE, the Information Processing Society of Japan, and the Institute of Electronics, Information and Communication Engineers of Japan.