Optimal Granularity of Parallel Test Generation on the
Client-Agent-Server Model

TOMOO INOUE,† TOMONORI YONEZAWA ‡ and HIDEO FUJIWARA †

This paper proposes 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 and presents an approach to parallel test generation for logic circuits on the CAS model. In this paper, we consider the fault parallelism in which a cluster of faults will be allocated from the client processor to an agent processor and from an agent processor to a server processor for the CAS model. Hence, we have to consider two granularities: one is the size of the cluster between the client and agents, and the other is the size of the cluster between agents and servers. We formulate the problem of test generation for the CAS model and analyze the optimal pair of granularities in both cases of static and dynamic task allocation. Finally, we present experimental results based on an implementation of our CAS model on a network of workstations using the ISCAS'89 benchmark circuits. The experimental results are very close to the analytical results which confirms the existence of an optimal pair of granularities that minimizes the total processing time for benchmark circuits as well as analysis.

1. Introduction

Theoretically, it is 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[3-5] 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[14]. 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[13] proposed their methods of parallel processing for test genera-

† Nara Institute of Science and Technology
‡‡ Matsushita Electric Industrial Co., Ltd.
the client processor to an agent processor and from an agent processor to a server processor for the CAS model. Hence, we have to consider two granularities; one is the size of the cluster between the client and agents, and the other is the size of the cluster between agents and servers. We formulate the problem of test generation for the CAS model and analyze the optimal pair of granularities in both cases of static and dynamic task allocation. Finally, we present experimental results based on an implementation of our CAS model on a network of workstations using the ISCAS’89 benchmark circuits. The experimental results are very close to the analytical results which confirms the existence of an optimal pair of granularities that minimizes the total processing time for benchmark circuits as well as analysis.

2. Architecture of the Client-Agent-Server Model

The architecture of our loosely-coupled multiple processor systems is illustrated in Fig. 2. This system is derived by inserting agent processors between a client and servers of the CS model. We call it a Client-Agent-Server model (CAS model). In this CAS model, $N_a$ agents are connected to the client, and $N_s$ servers are connected to each agent, where all processors are 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 sub-tasks and distributes each sub-task 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 for faults in a given circuit, the above process can be redescribed as follows: The client first generates a fault table of the faults. 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 received 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 for all the target faults, 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.

3. Formulation of the Problem

We formulate the test generation problem for the CAS model. It consists of one client, \( N_e \) agents and \( N_s \) servers per agent. Let the \( k \)-th server connected to the \( j \)-th 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 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.

The parameters used here are defined as follows:

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

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

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

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

\[ T = \max \{ T_{jk} \}. \]

4. 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.

4.1 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 is follows:

1. All servers are uniform, i.e., \( \tau_{ijk} = \tau_i \) 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., \( \delta_{ijk} = \delta_i \) for all faults \( f_i \) and servers \( S_{jk} \).

4.2 Communication Probability: \( \lambda_{aij}, \lambda_{sijk} \)

Let \( m_d \) 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 \( m_d \) 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

\[ \lambda_{aij} = \frac{1}{m_d} \]

since such a communication occurs only once for those \( m_d \) 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

\[ \lambda_{sijk} = \frac{1}{m_s} \]

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

4.3 Probability of Process Allocation: \( \delta_{ijk} \)

Suppose that the client requests an agent to process \( m_s \) target faults. The agent extract \( m_s \) faults from the \( m_s \) target faults, and requests a server to process the \( m_s \) target faults. Note that \( m_s \leq m_e \). The server generates a test-pattern for one of the \( m_s \) faults, and find out all the faults detected by the test-pattern by performing fault simulation for all faults, 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 \( \rho \) or \( \rho_{m_s} \) be the number of faults that are newly detected or found to be redundant at completion 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.
number of newly processed faults per server \( \rho \) is given by:

\[
\rho = \frac{m_s}{\rho \cdot m_s} = \frac{1}{\rho_i}. \tag{8}
\]

On the other hand, the probability that the process for fault \( f_i \) is allocated to some server is defined by:

\[
\sum_{j=1}^{N_s} \sum_{k=1}^{N_s} \delta_{ijk} = \frac{1}{\rho_i}. \tag{9}
\]

From the assumption that \( \delta_{ijk} = \delta_i \), we have:

\[
\sum_{j=1}^{N_s} \sum_{k=1}^{N_s} \delta_{ijk} = \sum_{j=1}^{N_s} \sum_{k=1}^{N_s} \delta_i = N_a N_s \delta_i. \tag{10}
\]

Hence, we have:

\[
\delta_{ijk} = \delta_i = \frac{1}{N_a N_s \rho_i}. \tag{11}
\]

### 4.4 Ratio of Newly Processed Faults to Target Faults: \( \rho \)

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 Ref. 10, we assumed the ratio of newly processed faults to target faults for the CS model to be:

\[
\rho(x) = \frac{1}{r_0 + r_1 x + r_2 m N} \tag{12}
\]

where \( m \) is the number of target faults, \( N \) 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 (see Ref. 10).

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 \( \rho \) 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 the expression (13), we have:

\[
\rho(x) = \frac{1}{r_0 + r_1 x + r_2 m_a N_a + r_3 m_a N_a} \tag{13}
\]

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

### 4.5 Communication Time: \( \tau_{ca}, \tau_{cs} \)

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 can not 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., \( N_a + N_s N_s \).

3. After receiving the results 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 \( N_a \). On the other hand, 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 \( N_s \).

From the above assumptions, we have:

\[
\tau_{ca} = \tau_{a0} + \tau_{a1} N_a (N_a + 1) + \tau_{a2} N_a \tag{14}
\]
where \( t_{00}, t_{a1} \) and \( t_{a2} \) are constants. And we have
\[
\tau_{cs} = t_{00} + t_{a1} N_a (N_a + 1) + t_{a2} N_a
\]
where \( t_{00}, t_{a1} \) and \( t_{a2} \) are constants.

Here we assume \( t_{00} = t_{a0}, t_{a1} = t_i \) and \( t_{a2} = t_2 \). Then we have
\[
\tau_{cs} = t_0 + t_1 N_a (N_a + 1) + t_2 N_a
\]
and
\[
\tau_{cs} = t_0 + t_1 N_a (N_a + 1) + t_2 N_a
\]
where \( t_0, t_1 \) and \( t_2 \) are constants.

4.6 Total Processing Time: \( T \)

Suppose that the number of processed faults is \( i \) when fault \( f_{\pi(i)} \) is processed where \( \pi \) is a permutation of \( I_M = \{1, 2, \ldots, M\} \). Then, from the expression (14), the ratio of newly processed faults when fault \( f_{\pi(i)} \) is processed can be expressed as
\[
\rho_{\pi(i)} = \frac{1}{r_0 + r_1 i + r_2 m_a N_a + r_3 m_a N_a}.
\]
(19)

Let \( P \) be the set of all permutations of \( I_M \).

There is a one-to-one correspondence between permutations of \( I_a \) and sequences of faults. The total number of sequences is \( M! \).

From the expressions (1), (12), (17), (18) and (19), we can derive the average of total processing time for all permutations:
\[
T = \frac{1}{M!} \sum_{\pi \in P} \sum_{i=1}^{M} \frac{1}{\tau_{\pi(i)}} (M - 1) \sum_{i=1}^{M} \tau_{\pi(i)}.
\]
(21)

On the other hand we have
\[
\sum_{\pi \in P} \frac{1}{\tau_{\pi(i)}} = \sum_{i=1}^{M} \frac{1}{\tau_{i}} (M - 1) \sum_{i=1}^{M} \tau_{i}.
\]
Substituting the mean processing time for each fault:
\[
\tau = \frac{1}{M} \sum_{i=1}^{M} \tau_{i}.
\]
(22)

into the right side of the above equation (21), we have
\[
\sum_{\pi \in P} \frac{1}{\tau_{\pi(i)}} = \sum_{i=1}^{M} \frac{1}{\tau_{i}} (M! \tau).
\]
(23)

Hence, from (20) and (23) we have
\[
T = \frac{1}{M!} \sum_{i=1}^{M} \frac{1}{N_a N_a} (r_0 + r_1 i + r_2 m_a N_a + r_3 m_a N_a) \cdot \left( \frac{\tau + \tau_{cs}}{m_a + \tau_{cs}} \right)
\]
\[
= \frac{M}{N_a N_a} \left( r_0 + r_1 \frac{M + 1}{2} + r_2 m_a N_a + r_3 m_a N_a \right) \left( \frac{\tau + \tau_{cs}}{m_a + \tau_{cs}} \right)
\]
(24)

Partially differentiating \( T \) by \( m_a \), we have
\[
\frac{\partial T}{\partial m_a} = \frac{M}{N_a N_a} \left( r_0 N_a (\tau + \tau_{cs}) \right) \left( \frac{r_0 + r_1 M + 1 + r_2 m_a N_a \tau_{cs}}{m_a^2} \right)
\]
\[
= \left( r_0 + r_1 \frac{M + 1}{2} + r_2 m_a N_a \tau_{cs} \right) \left( \frac{\tau + \tau_{cs}}{m_a + \tau_{cs}} \right)
\]
(26)

Then, we have
\[
T_{\text{min}} = \frac{M}{N_a N_a} \left( \sqrt{r_0 N_a \tau_{cs}} \right)
\]
\[
+ \sqrt{\left( r_0 + r_1 \frac{M + 1}{2} + r_2 m_a N_a \tau_{cs} \right) \left( \frac{\tau + \tau_{cs}}{m_a + \tau_{cs}} \right)^2}
\]
(27)

when
\[
m_{\text{opt}} = \sqrt{\left( r_0 + r_1 \frac{M + 1}{2} + r_2 m_a N_a \tau_{cs} \right) \left( \frac{\tau + \tau_{cs}}{m_a + \tau_{cs}} \right)}
\]
(28)

Partially differentiating \( T_{\text{min}} \) by \( m_a \), we have
\[
\frac{\partial T_{\text{min}}}{\partial m_a} = \frac{M}{N_a N_a} \left( \sqrt{r_2 m_a \tau_{cs}} \right)
\]
\[
\times \left( 1 + \sqrt{\left( r_0 + r_1 \frac{M + 1}{2} + r_2 m_a N_a \tau_{cs} \right) \left( \frac{\tau + \tau_{cs}}{m_a + \tau_{cs}} \right)^2} \right)
\]
\[
\times \left( r_0 + r_1 \frac{M + 1}{2} + r_2 m_a N_a \tau_{cs} \right) \left( \frac{\tau + \tau_{cs}}{m_a + \tau_{cs}} \right)
\]
(29)

Then, we have the minimum of \( T \)
\[
T_{\text{min}} = \frac{M}{N_a N_a} \left( \sqrt{r_2 m_a \tau_{cs}} \right)
\]
\[
+ \sqrt{\left( r_0 + r_1 \frac{M + 1}{2} \right) \tau \left( \frac{\tau + \tau_{cs}}{m_a + \tau_{cs}} \right)^2}
\]
(30)

when
\[
m_{\text{opt}} = \sqrt{\left( r_0 + r_1 \frac{M + 1}{2} \right) \tau \tau_{cs} \left( \frac{\tau + \tau_{cs}}{m_a + \tau_{cs}} \right)}
\]
(31)

and
\[
m_{\text{opt}} = \sqrt{\left( r_0 + r_1 \frac{M + 1}{2} \right) \tau_{cs} \left( \frac{\tau + \tau_{cs}}{m_a + \tau_{cs}} \right)}
\]
(32)

which is derived from the expression (28) by substituting \( m_a \) in the expression for \( m_{\text{opt}} \).

Figure 3 shows the 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, \( m_s \). From this figure we can see that there exists an optimal number of target faults for an agent, \( m_{\text{opt}} \), and an optimal number of target faults for a server, \( m_{\text{opt}} \), which minimize the total processing time.
The parallel test generation system of the CAS model was implemented on a network (Ethernet) of workstations (SUN4/1 C’s). The FAN algorithm was used as a test-pattern generator. Figures 4, 5 and 6 give the 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 modified into combinational circuits by assuming full-scan design. In these figures, we can see that the shape of the graphs coincides closely with that of Fig. 3 obtained from the above analysis and hence there exists an optimal granularity pair which minimizes the total processing time.

5. 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 respectively 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 $S_{jk}$. Suppose that the number of processed faults is $i$ when fault $f_{\pi(i)}$ is processed where $\pi$ is a permutation of $I_M = \{1, 2, \cdots, M\}$. Let $m_{ai}$ and $m_{ai}$ be the numbers of target faults allocated to an agent and a server, respectively, when $f_i$ 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 (20) as follows:

$$T = \frac{1}{M!} \sum_{\pi} \sum_{i=1}^{M} \frac{1}{N_A N_S} \left( r_0 + r_l + r_m m_{ai} + r_i m_{ai} N_S \right) \left( \tau + \frac{\tau_{ca}}{m_{ai}} + \frac{\tau_{ca}}{m_{si}} \right)$$

$$= \sum_{i=1}^{M} \frac{1}{N_A N_S} \left( r_0 + r_l + r_m m_{ai} + r_i m_{ai} N_S \right) \left( \tau + \frac{\tau_{ca}}{m_{ai}} + \frac{\tau_{ca}}{m_{si}} \right)$$

(33)

(34)

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

$$\frac{\partial T}{\partial m_{si}} = \frac{M}{N_A N_S} \left( r_2 N_S \frac{\tau + \frac{\tau_{ca}}{m_{ai}}}{m_{ai}} \right) \left( r_0 + r_l + r_m m_{ai} N_S \right)$$

$$= \frac{M}{N_A N_S} \left( r_2 N_S \frac{\tau + \frac{\tau_{ca}}{m_{ai}}}{m_{ai}} \right)$$

(35)

Then, we have
Fig. 4  Total processing time versus granularity: Experimental result for circuit s9234.

Fig. 5  Total processing time versus granularity: Experimental result for circuit s13207.
Fig. 6 Total processing time versus granularity: Experimental result for circuit s15850.

\[ T_{\text{min}} = \sum_{i=1}^{M} \frac{1}{N_a N_s} \left( \frac{\sqrt{r_0 N_s \tau_{cs}}}{r_0 N_s} \right) \left( 1 + \frac{r_0 N_s \tau_{cs}}{\sqrt{(r_0 + r_0 i + r_0 m_a N_s)(\tau + \tau_{cs}) (\frac{r_0 N_s \tau_{cs}}{m_a})}} \right) \]

when

\[ m_a = \frac{\sqrt{(r_0 + r_0 i + r_0 m_a N_s)\tau_{cs}}}{r_0 N_s (\tau + \frac{\tau_{cs}}{m_a})} \]

Partially differentiating \( T_{\text{min}} \) by \( m_a \), we have

\[ \frac{\partial T_{\text{min}}}{\partial m_a} = \frac{M}{N_a N_s} \cdot \left( \frac{r_0 N_s \tau_{cs}}{1 + \sqrt{(r_0 + r_0 i + r_0 m_a N_s)(\tau + \frac{\tau_{cs}}{m_a})}} \right) \]

\[ \cdot \left( r_0 N_s \tau_{cs} - \frac{(r_0 + r_0 i) \tau_{cs}}{m_a} \right) \]

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

\[ T_{\text{dynamic}} = \sum_{i=1}^{M} \frac{1}{N_a N_s} \left( \sqrt{r_0 N_s \tau_{cs}} + \sqrt{r_0 N_s \tau_{cs}} \right) + \sqrt{(r_0 + r_0 i) \tau} \]

when

\[ m_a = \frac{\sqrt{(r_0 + r_0 i) \tau_{cs}}}{r_0 N_s \tau} \]

Circuit: s15850

\[ N_a = 4 \]
\[ N_s = 8 \]

and

\[ m_{ai} = \frac{\sqrt{(r_0 + r_0 i) \tau_{cs}}}{r_0 N_s \tau}, \text{ for all } i. \] (41)

From the above expressions (40) and (41), the optimal granularity (the optimal size of target faults) of time \( t \) can be expressed as

\[ m_a(t) = \frac{\sqrt{(r_0 + r_0 x_i) \tau_{cs}}}{r_0 N_s \tau} \] (42)

and

\[ m_{ai}(t) = \frac{\sqrt{(r_0 + r_0 y_i) \tau_{cs}}}{r_0 N_s \tau} \] (43)

where \( x_i \) 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 \( x_i \) of processed faults till now (at time \( t \)), calculates the number \( m_a(t) \) of target faults from the equation (42), and then allocates \( m_a(t) \) target faults with the number \( x_i \), to an agent. The agent calculates the number \( m_{ai}(t) \) of target faults from the equation (43), picks the \( m_{ai}(t) \) target faults out of the \( m_a(t) \) target faults, and then allocates the \( m_{ai}(t) \) target faults to an idle server. Note
that although the equations (42) and (43) represent continuous functions, \( m_a(t) \) and \( m_b(t) \) 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

\[
T_{\text{static}} = \frac{M}{N_a N_b} \left( \sqrt{r_a N_a} + \sqrt{r_b N_b} \right) + \frac{\left( r_a + r_b \frac{M + 1}{2} \right)^2}{\tau} \quad (44)
\]

Hence, the difference between \( T_{\text{static}} \) and \( T_{\text{dynamic}} \) is

\[
T_{\text{static}} - T_{\text{dynamic}} = 2 \sqrt{\tau} \left( \sqrt{r_a N_a} \sqrt{r_b N_b} \right) \frac{M + 1}{2} \quad (45)
\]

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

6. Conclusions

In this paper we presented an approach to parallel processing based on fault parallelism for test generation in a loosely-coupled distributed networks of general purpose processors. In order to get a more efficient scheme than the CS model, we proposed 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.

We considered two granularities; one is the size of the cluster between the client and agents, and the other is the size of the cluster between agents and servers. We formulated the problem of test generation for the CAS model, and analyzed the optimal pair of granularities in both cases of static and dynamic task allocation. We presented experimental results based on an implementation of our CAS model on a network of workstations using the ISCAS‘89 benchmark circuits. The experimental results are very close to the analytical results which confirms the existence of an optimal pair of granularities that minimizes the total processing time for benchmark circuits.

References

Tomoo Inoue was born in Tokyo, Japan, on July 20, 1965. He received the B.E. degree in electric and communication engineering and the M.E. degree in electrical engineering from Meiji University, Kawasaki, Japan in 1988 and 1990, respectively. From 1990 to 1992, he was engaged in 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 the IEEE and the Institute of Electronics, Information and Communication Engineers of Japan and the Information Processing Society of Japan.

Tomonori Yonezawa was born in Kagoshima, Japan on March 15, 1969. He received the B.E. degree in electric and communication engineering and the M.E. degree in electrical engineering from Meiji University, Kawasaki, Japan in 1991 and 1993, respectively. Since 1993 he has been with Kyushu Research Group in Matsushita Electric Industrial Co., Ltd. and engaged in development of VLSIs for multimedia systems. His research interests include test generation, parallel processing and image processing.

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. He was with Osaka University from 1974 to 1985 and Meiji University from 1985 to 1993, and joined Nara Institute of Science and Technology in 1993. 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. Presently he is a Professor at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan.

His research interests are 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). Dr. Fujiwara 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 ICFE Young Engineer Award in 1977 and IEEE Computer Society Certificate of Appreciation Award in 1991.