Joint Symposium on Fault-Tolerant Computing, pp.128- 133, in China, July 1989.

ANALYSIS OF PARALLEL PROCESSING FOR TEST GENERATION IN A DISTRIBUTED SYSTEM

Hideo Fujiwara and Tomoo Inoue

Department of Computer Science
Meiji University
1-1-1 Higashi-mita, Tama-ku
Kawasaki 214, Japan

Abstract

The problem of test generation for logic circuits is known to be NP-hard, and hence it is very hard to speed up 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 analyze the effects of the allocation of subproblems to processors, the grain size of subproblems and the speedup ratio of the multiple-processor system to a single processor system, in order to see the performance of the multiple-processor system.

1. Introduction

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 these fault simulation is less than $O(G^3)$ and seems to behave as an $O(G^2)$ algorithm [6,7,8]. 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] is to use the MIT Connection Machine as a test-pattern generation engine. Until now, the results of [14,15] were that speed-up 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. El-ziq 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].

In Motohara et al. [18] and Fujiwara [19], two approaches were presented to parallel processing for test generation using a multiple processor system called LINKS-1; (1) parallelism to deal with a large number of faults simultaneously and (2) parallelism to search a large number of nodes in a decision tree simultaneously. In this paper, we shall extend the first approach and apply it to a loosely-coupled distributed network of general purpose computers, where a cluster of faults will be allocated to each processor instead of allocating one fault each time. We shall analyze the effect of *granularity* of computation (the number of faults allocated to a processor each time) to find the optimal granularity and the speedup ratio of the multiple-processor system to a single processor system, in order to see the performance of the multiple-processor system.

2. Architecture of the Distributed System

The architecture of our loosely-coupled multiple-processor system is illustrated in Figure 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 a set of faults. Here, we shall consider to distribute 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 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.

If we decrease the granularity of computation (the number of faults given to a server) 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. Since the interaction among various performance factors is very complex, it is very difficult to predict the exact granularity size which will yield the best performance for a given problem and distributed system.

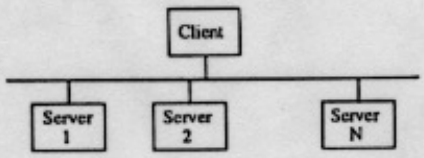

Figure 1. Architecture of the distributed system

3. Formulation of the Problem

We shall consider a distributed network consisting of a client and N servers as shown in Figure 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

$$T_j = \sum_{i=1}^{M} \delta_{ij}\left(\tau_{ij} + \lambda_{ij}\tau_c\right) \qquad (1)$$

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

$$T = \max\{T_j\} \qquad (2)$$

The problem is thus to find a task allocation schedule which minimizes T. However, this is a hard problem in general, and so we shall consider a uniform and homogeneous problem where all the servers are uniform and the same computation and communication patterns are repeated during each process.

4. Homogeneous Problem with Static Task Allocation

4.1 Optimal Granularity

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. Further we assume *homogeneous* case where the same computation and communication patterns are repeated during each process for all faults; i.e., $\tau_{ij} = \tau$, $\lambda_{ij} = \lambda$, and $T_j = T$ for all faults f_i and servers j, the expressions mentioned above are simplified as follows.

Let m be the number of target faults transferred from the client to a server at each communication. Then $\lambda = 1/m$.

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 expressions (1) and (2), we have

$$T = \frac{M}{N}(\tau + \lambda\tau_c) = \frac{M}{N}\left(\tau + \frac{\tau_c}{m}\right) \quad (3)$$

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

$$T_{min} = \frac{M}{N}\tau + \tau_c \quad (4)$$

The above expression means that for the homogeneous case, where the processing and communication time are all the same for all faults, and 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 fault simulation process after test-pattern generation process, then all faults have not to be processed.

Case II (with fault simulation)

Suppose that the client requests to a server to 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. 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. 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

$$\rho = \frac{\text{number of newly processed faults per server}}{\text{number of target fauts per server}} \quad (5)$$

where $\rho \ge 1$. Note that this ratio will decrease as the number of processed faults increases. Therefore, it is expressed as ρ_i, the ratio for i-th processed fault f_i.

During each iteration of server process, m target faults are test-generated and ρm faults are found to be detectable through fault simulation. Therefore, the probability that test-generation process for i-th fault f_i is allocated to server j is

$$\delta_{ij} = \delta_i = \frac{1}{N\rho_i} \quad (6)$$

The total amount of processing and communication time T is obtained from expression (3) by substituting (6) and $\tau_i = \tau$, $\lambda_{ij} = \lambda = 1/m$, and $T_j = T$.

$$T = \sum_{i=1}^{M} \frac{1}{N\rho_i}\left(\tau + \frac{\tau_c}{m}\right) \quad (7)$$

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

$$\tau_c = t_0 + t_1 N \quad (8)$$

where t_0 and t_1 are constants.

Ratio of newly processed faults to target faults: ρ

Figure 2 gives the typical curve of the number of newly processed faults for 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

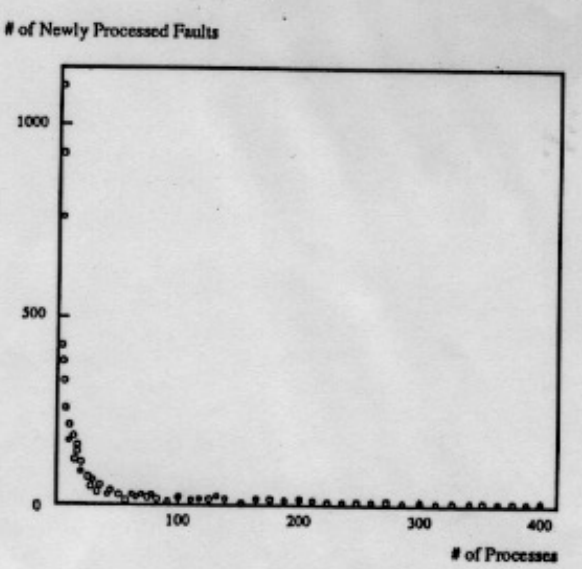

Figure 2. Number of newly processed faults for C7552

of newly processed faults to target faults, $\rho(x)$, is

$$\rho(x) = \frac{1}{r_0 + r_1 x} \tag{9}$$

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

The ratio $\rho(x)$ will also decrease monotonously 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

$$\rho(x) = \frac{1}{r_0 + r_1 x + r_2 mN} \tag{10}$$

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

Without loss of generality, we can assume that the index i of a fault f_i corresponds to the order of processing, i.e., the number of processed faults is i when fault f_i is processed. Therefore, the above expression (10) can be given by

$$\rho_i = \frac{1}{r_0 + r_1 i + r_2 mN} \tag{11}$$

Summarizing the above expressions (7), (8) and (11), we have

$$\begin{aligned} T &= \sum_{i=1}^{M} \frac{1}{N}\left(r_0 + r_1 i + r_2 mN\right)\left(\tau + \frac{t_0 + t_1 N}{m}\right) \\ &= \frac{M}{N}\left(r_0 + r_1 \frac{M+1}{2} + r_2 mN\right)\left(\tau + \frac{t_0 + t_1 N}{m}\right) \\ &= \frac{M}{N}\left(\mu_1 + \mu_2 m + \frac{\mu_3}{m}\right) \end{aligned} \tag{12}$$

where

$$\mu_1 = \left(r_0 + \frac{r_1}{2}(M+1)\right)\tau + r_2 t_0 N + r_2 t_1 N^2$$

$$\mu_2 = r_2 N \tau$$

and

$$\mu_3 = \left(r_0 + \frac{r_1}{2}(M+1)\right)(t_0 + t_1 N) \tag{13}$$

Differentiating T by m, we have

$$\frac{dT}{dm} = \frac{M}{N}\left(\mu_2 - \frac{\mu_3}{m^2}\right) \tag{14}$$

Then, we have the minimum of T

$$T_{min} = \frac{M}{N}\left(\mu_1 + 2\sqrt{\mu_2 \mu_3}\right) \tag{15}$$

when

$$m = \sqrt{\frac{\mu_3}{\mu_2}} \tag{16}$$

Figure 3 presents the total amount of processing and communication time as a function of the number of target faults per server for three cases of different parameters; (a) $t_0 = t_1 = 0.1$ and $r_2 = 0.00001$, (b) $t_0 = t_1 = 0.01$ and $r_2 = 0.00001$, and (c) $t_0 = t_1 = 0.1$ and $r_2 = 0.000001$. From the curves of (a) and (b), we can see that as the communication time decreases, both the optimal granularity and the total processing time decrease. From the curves of (a) and (c), we can see 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.

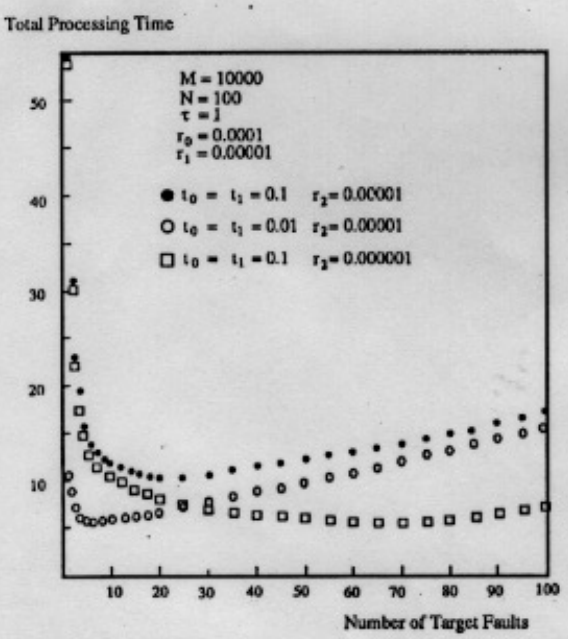

Figure 3. Computation time versus number of target faults

4.2 Speedup of multiple-processor system

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

$$S = \frac{T_{single}}{T_{multi}} \tag{17}$$

The computation time required to complete test generation for all faults on a single processor is given by

$$T_{single} = \sum_{i=1}^{M} (r_0 + r_1 i)\tau = M\left(r_0 + r_1\frac{M+1}{2}\right)\tau \tag{18}$$

From expressions (15), (17), and (18), we have the maximum speedup S_{max} as follows:

$$S_{max} = \frac{T_{single}}{T_{min}} = \frac{N\left(r_0 + r_1\frac{M+1}{2}\right)\tau}{\mu_1 + 2\sqrt{\mu_2\mu_3}} = \frac{N}{\left(1 + \sqrt{\left(\frac{r_2 N}{r_0 + r_1\frac{M+1}{2}}\right)\left(\frac{t_0+t_2N}{\tau}\right)}\right)^2} < N \tag{19}$$

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

$$r_2 mN << r_0 + r_1\frac{M+1}{2} \tag{20}$$

and

$$\frac{t_0+t_2N}{m} << \tau \tag{21}$$

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.

Figure 4 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.

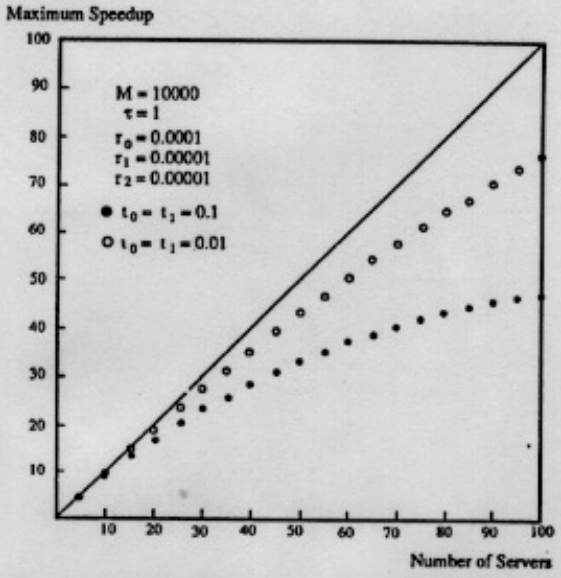

Figure 4. Maximum speedup ratio vs number of servers

5. Conclusion

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 subproblems to processors, the grain size of subproblems and the speedup ratio of the multiple-processor system to a single processor system, in order to see the performance of the multiple-processor system.

For the homogeneous case where the processing and communication time are assumed to be the same for all faults, if test-pattern generation process is applied to each fault, 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 a fault can also be a test-pattern for other faults. Hence, if we apply fault simulation process after test-pattern generation process, then all faults have not to be processed. To analyze this case, we have introduced a ratio of newly processed faults to target faults. For this model, the function T has its minimum between $1 < m < M/N$.

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 to 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.

References

[1] P.Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," IEEE Trans. Comput., Vol. C-30, No.3, pp.215-222, March 1981.

[2] H.Fujiwara and T. Shimono, "On the acceleration of test pattern generation algorithms," IEEE Trans. Comput., Vol. C-32, No.12, pp.1137-1144, Dec. 1983.

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

[4] O.H.Ibarra and S.K.Sahni, "Polynomially complete fault detection problems," IEEE Trans. Comput., Vol. C-24, No.3, pp.242-249, March 1975.

[5] H.Fujiwara and S.Toida, "The complexity of fault detection problems for combinational logic circuits," IEEE Trans. Comput., Vol. C-31, No.6, pp.555-560, June 1982.

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

[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, No.9, pp.848-862, Sept. 1987.

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

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

[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," Electronic 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," Proc. Int. Test Conf., pp.224-228, 1984.

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

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

[16] Y.M.Elziq, H.H.Butt and A.K.Bhatt, "An automatic test pattern generation machine," Proc. IEEE Int. Conf.on Computer-Aided Design, pp.257-259, 1984.

[17] P.A.Duba, R.K.Roy, J.A.Abraham, and W.A.Rogers, "Fault simulation in a distributed environment," Proc. 25th Design Automation Conf., pp.686-691, 1988.

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

[19] H.Fujiwara and A.Motohara, "Fast test pattern generation using a multi-processor system," Trans. of IEICE, Vol. E-71, No.4, pp.441-447, April 1988.

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

[21] S. Shimojo, H.Miyahara, and K.Takashima, "Process assignment on multi-processor with communication contentions," Trans. of IECE, Vol. J68-D, No.5, pp.1049-1056, March 1985 (in Japanese).

[22] Y-T. Wang and R.J.T. Morris, "Load Sharing in Distributed Systems," IEEE Trans. Comput., Vol. C-34, No.3, pp.204-217, March 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, No.4, pp.421-432, April 1987.

[24] F. Brglez and H. Fujiwara, "A neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN," Proc. IEEE Int. Symp. on Circuits and Systems, Kyoto, Japan, June 5-7 1985.