TM-1197

Fault Simulation on Parallel Inference Machines

by
H. Nakashima & R. Satoh (Mitsubishi)

July, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

Fault Simulation on Parallel Inference Machines

並列推論マシンにおける故障シミュレーション

Hiroshi Nakashima Reiko Satoh
(Mitsubishi Electric Corp.)

Abstract

This paper describes a method of fault simulation for parallel processors. The proposed method is based on *Time Warp* for *good* simulation and *Single Fault Propagation* for *fault* simulation. These methods, however, are modified for efficient parallel execution, removing the *barrier* between good and fault simulation, and simulating multiple faulty circuits in parallel. Preliminary performance evaluation results on parallel inference machines, Multi-PSI and PIM/m, show good efficiency of our method, about 6 fold speedup on 9 processors even for a small circuit.

本稿は，Time Warp 法と単一故障伝播法に基づく，並列マシン上の故障シミュレーションの方式について述べたものである。本方式では，効率的な並列計算のためにこれら二つの手法を改良し，論理シミュレーションと故障シミュレーションの間のバリア同期を除去するとともに，複数の故障マシンの並列シミュレーションを実現している。本方式を並列推論マシン Multi-PSI と PIM/m 上に実装して性能評価を行った結果，小規模な回路においても 9 プロセッサで約 6 倍という，良好な台数効果を得ることができた。

1 Introduction

In the flow of VLSI circuit design, one of the most time-consuming step is *fault simulation* which is performed for the evaluation and improvement of circuit test patterns. Although automatic test generation systems will eliminate try-and-errors with handmade patterns in this lengthy step, they include and spend much time on fault simulation in order to reduce patterns to be generated. The performance of fault simulation, therefore, is important for the efficiency of VLSI testing CAD, in either cases that test patterns are handmade or automatically generated.

Since a fault simulator handles huge number of gates and faults, it is natural to simulate them in parallel in order to achieve high performance. In fact, various parallel fault simulation methods have been proposed, although many of them are *bit parallel* for sequential processing. On the other hand, *Time Warp (TW)* method [Jefferson 85] is playing an important role in parallel logic simulation in which a circuit is divided into partitions simulated in parallel. Several implementations of the method on parallel machines showed significantly good performance [Chung 89, Matsumoto 92]. This method, however, has been scarcely adopted for fault simulation because it seems hard to combine the method with *Single Fault Propagation (SFP)* [Hong 78] which is the base of most of high speed fault simulators [Waicukauski 87]. That is, a straightforward parallelization of a SFP fault simulator using TW method will be inefficient because of insufficient number of faulty events evaluated in parallel and the synchronization barrier between good and fault simulation.

In this paper, we propose an efficient parallel fault simulation method in which these two methods are combined. In our method, multiple faulty circuits are simulated in parallel to generate sufficient number of faulty events. The barrier is removed performing good and fault simulations concurrently. We also show experimental results of the fault simulator implemented on parallel inference machines, Multi-PSI [Takeda 88] and PIM/m [Nakashima 92].

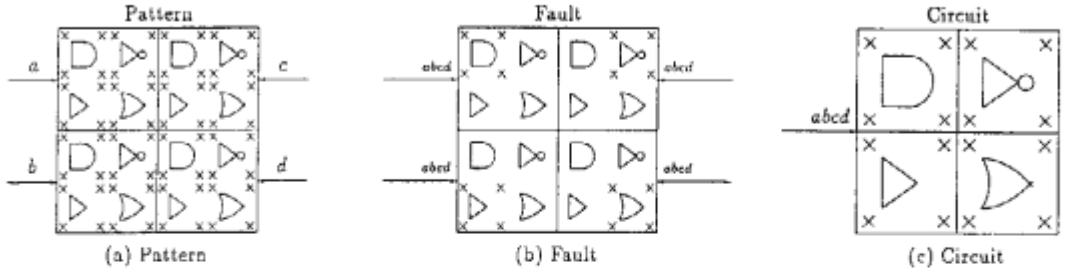
This paper is organized as follows: Section 2 briefly discusses the parallelism in fault simulation; Section 3 describes the circuit model, algorithm and implementation issues of our method; Section 4 presents experimental results and analysis on them; and Section 5 gives the conclusion.

2 Parallelism in Fault Simulation

It is well known that there are three types of parallelism in fault simulation, *pattern*, *fault* and *circuit* parallelism [Levendel 83] (Figure 1). While the first two types are often exploited in *bit parallel* execution on serial machines, the third type mainly concerns *real parallel* execution on parallel machines.

The *pattern* parallel simulation combined with SFP, so-called *PPSFP*, is regarded as one of the most promising method for combinational circuits on serial machines [Waicukauski 87]. In this method, multiple input patterns, usually 32, are injected in the circuit to be simulated, and multiple states of each gate corresponding to

中島 浩（三菱電機，hiroshi@isl.melco.co.jp），佐藤 令子（三菱電機）



(a) Pattern (b) Fault (c) Circuit

Figure 1: Parallelism in Fault Simulation

the patterns are evaluated simultaneously using bitwise logic operations. It is also attempted to apply the method to sequential circuits which require iterative evaluation to converge the states of flip-flops in them [Gouders 91]. The pattern parallelism, however, is hardly exploited in real parallel execution because the potential parallelism in a parallel machine is, say 32 bit times 64 processors, much more what can be utilized in this method.

The *fault* parallelism is also often exploited in bit parallel execution using bitwise operations to evaluate the states of multiple faulty circuits [Cheng 91]. Since a large circuit has many potential faults and multiple faulty circuits can be evaluated independently of each other, this approach seems fit for real parallel execution [Duba 88]. However, a straightforward extension of the fault parallel simulation for parallel machines will compel each processor element to perform the good simulation of the entire circuit. This concludes that the speedup, which is at most the ratio of the execution time for the fault simulation to that for the good simulation, will be insufficient for large scale parallel machines.

The *circuit* parallel simulation has been widely used for logic (good) simulation. Especially, several simulators with TW method achieved good speedup on large scale parallel machines [Chung 89, Matsumoto 92]. As for fault simulation, however, the straightforward application of the method will also fail, because the number of faulty events generated by the injection of a fault will be too small to provide enough works to large amount of processor elements. Although *concurrent* method will generate sufficient events and achieve fairly high speedup [Mueller-Thuns 89], its low performance on single processor will cancel the good feature.

Therefore, it will be required to combine two or more parallelization methods in order to achieve both high single processor performance and good speedup. Combinations of the pattern parallelism with others were proposed in several previous works [Ishiura 87, Narayanan 88, Narayanan 89], but pattern parallel method will be hardly applied to sequential circuits especially in the case of parallel processing.

Thus, we combined the circuit parallel and fault parallel in order to accelerate both good and fault simulations. Good simulation is performed using TW method which will exploits the circuit parallelism. On the other hand, fault simulation will be accelerated by the simultaneous injection of multiple faults which will provide sufficient and widely diffuse faulty events.

3 Algorithm and Implementation

3.1 Circuit Model

Our fault simulator is capable to handle combinational circuits and *synchronous* sequential circuits. That is, a circuit to be simulated should allow zero delay evaluation for its logic gates and unit delay for flip-flops.

The logic value representing the state of a signal is 0, 1, or X. A fault is stuck-at 0 or 1, and is injected into one of the input terminals of a gate which has two or more inputs, or the output terminal if it has two or more fanout branches.

3.2 Algorithm Overview

The SPF method is represented as follows:

```
for t := 0 to last_cycle do
  begin
    good_simulation(t)
    for f := 1 to N_of_faults do
      begin
        inject_fault(f)
        propagate_fault(f)
        if is_detected(f) then drop_fault(f)
      end
  end
```

In order to exploit the circuit parallelism, the circuit is divided into partitions each of which is allocated to a processor element. The functions good_simulation and propagate_fault are executed by a levelized event driven simulator in a processor element, and will exchange events input to or output from gates on other processor elements. Moreover, the inner and outer loop are parallelized in order

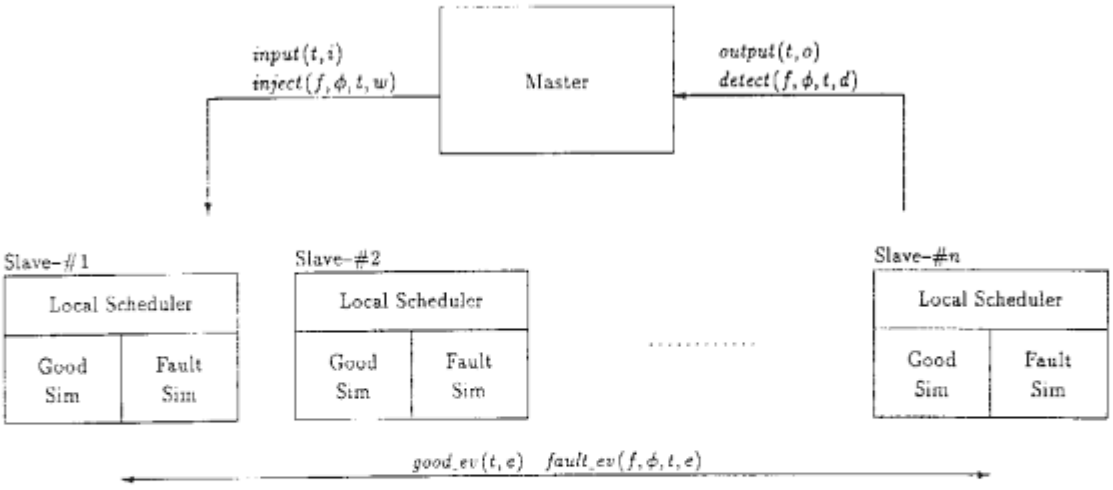


Figure 2: Configuration of Fault Simulator

to exploit the fault parallelism and to break the barrier between the good and fault simulations.

3.3 Parallelization of Inner Loop

The inner loop is parallelized in two ways, one is the bitwise fault parallel method with a group of (up to) 32 faults, and the other is *do-all* parallelization. The loop can be executed completely in parallel if each gate can hold states corresponding to all faults. This assumption, of course, is unrealistic and a certain restriction is necessary to keep the memory space for a gate at a reasonable size. We devised two techniques for exploiting as much parallelism as possible under the memory space limitation.

The first technique is similar to *Bounding Window* [Briner 91], which is introduced to limit the size of the trail for rollback of TW method. That is, the number of faulty circuit groups to be simulated concurrently is limited to a certain number Δf which is the size of *faulty trail* of each gate. Unlike the original Bounding Window, the condition to start the simulation for a fault group f is not necessary to be the completion of the group $f - \Delta f$. Instead, a pool for free entries of the faulty trail is maintained, and the simulation for f can be started if it can get an entry ϕ from the pool.

The trail the size of Δf, however, is still too large to be contained in each gate, because Δf is desirable to be larger than the number of processor elements in order to provide sufficient works to them. Therefore, we introduced the second technique, an on-demand loading/saving of gate states from/to the single faulty trail allocated on each processor element. Each entry of the trail contains a fault group identifier F, its simulation time T, and a list of gate states S.

When a processor element performs the simulation of the group f on the time t with the faulty trail entry ϕ for the first time, it sets F and T of the entry ϕ to f and t and clears S. A gate activated by the fault propagation of f may have the state of f' and t' associated to ϕ', and its state will be appended to S of ϕ', if F and T of ϕ' is equal to f' and t' (i.e., the simulation for f' on t' may not be completed).

On the other hand, the simulation of f on t with ϕ is performed twice or more, a kind of rollback occurs on and after the second occasion. That is, the states in S of ϕ is loaded to corresponding gates, saving their states if necessary. This technique promises that the state of a gate is evaluated in a constant time for the first time, and also on rollback cases under a reasonable expectation that a gate whose state is loaded will be activated again.

3.4 Parallelization of Outer Loop

The parallelization of the inner loop will be insufficient to get good performance, because the barrier between the good and fault simulations brings synchronization overhead. The barrier before the fault simulation is reasonable because it will be worthless to perform the fault simulation with unstable good circuit states as the reference. The barrier after the fault simulation, however, can be broken if each gate has a *good trail* for its good states for the reference of the fault simulation. Moreover, this breaking naturally makes it possible to break the barrier between consecutive good simulations as TW simulators do.

For breaking these barriers, we adopted the concept of Bounding Window again. Each gate has the good trail size of $\Delta t + \Delta t'$, Δt for the reference and $\Delta t'$ for rollback in the TW simulation. Therefore, the good simulation on time t can be started, if the good simulation on $t - \Delta t'$ and the fault simulation on $t - \Delta t - \Delta t'$ are completed.

3.5 Implementation

We implemented the fault simulator on parallel inference machines Multi-PSI [Takeda 88] and PIM/m [Nakashima 92], which have up to 64 and 256 processor elements connected by two-dimensional mesh networks. The whole program is written in a committed choice logic programming language KL1 [Ueda 90].

As shown in Figure 2, the simulator has the single-master/multiple-slave configuration. The master processor distributed input patterns and gathers good output values to/from slave processors. As for the fault simulation, it maintains a pool of undetected faults, makes up and injects fault groups, and collects detected faults to removed them from the pool. The master is also responsible to the synchronization which is performed by gathering local completions of good/fault simulation on slave processors.

Each slave processor has a local scheduler and good and fault simulators. The local scheduler receives messages from the master and other slaves. If the processor is idle, a message immediately activates the good or fault simulator. Otherwise and usually, the message is put in one of pools each of which is corresponding to a simulation time t or a fault group f. When the simulation for a time or a fault group is completed, a pool is picked up and its contents are injected into one of the simulator. The order of picking is faulty event pools first in FIFO manner, then the oldest good event pool. On the invocation of a simulator, the local scheduler indicates whether rollback may occur, in order to keep the simulator from performing unnecessary operations in usual non-rollback cases.

This master-slave configuration simplifies the implementation of the synchronization and the maintenance of faults. However, this design might not be excellent, because the master with many slaves could be a *hot spot*, as discussed later.

Table 2: Statistics of Circuits

level	# of gates	# of faults	coverage
1	339	1430	95.7 %
2	1017	4196	95.6
3	2373	8987	93.3

4 Experimental Results

4.1 Test Circuits

For the performance evaluation, we used binary trees of 32 bit adders. A tree n level deep has 2^n input vectors, $v_1, v_2, \ldots, v_{2^n}$, 32 bit wide for each, and $2^n - 1$ adders to calculate $\sum v_i$. Table 2 shows the number of gates and faults in circuits with level 1 to 3 which are used for the evaluation. The table also shows fault coverage of pseudo random 256 input patterns. Most of undetected faults due to the redundancy of carry lookahead logic.

Since our circuit partitioning routine is under development, these circuits are partitioned by hand using a strategy in which each 1 bit adder is the partitioning unit and adjacent bits are merged if necessary. The result is quite similar to what is gotten by depth first partitioning from primary outputs, and our routine will also perform similar partition.

4.2 Single Processor Performance

Table 1 shows single processor performances of Multi-PSI and PIM/m with 256 input patterns. The good simulation performance of Multi PSI is about 2.5 K-event/sec and is significantly higher than 1.4 K-event/sec of Matsumoto's logic simulator on the same machine [Matsumoto 92], probably owing to the zero-delay evaluation. Moreover, since levelization greatly reduces the number of events about 30 %, the difference is greater than it looks. We also evaluated parallelization overhead caused by unnecessary trail operations. The overhead is small, about 7 %, because of the elimination of operations for rollback described in Section 3.5.

Table 1: Single Processor Performance

machine	level	good sim.		fault sim.		
		exec. time (sec)	event/sec	exec. time (sec)	event/sec	time/fault (msec)
Multi-PSI	1	27.8	2459	38.8	575	1.36
	2	70.5	2681	173.6	653	1.51
	3	154.0	2786	456.0	664	1.44
PIM/m	1	12.5	5470	26.6	1267	0.62
	2	36.2	5228	83.3	1429	0.69
	3	83.2	5156	227.0	1393	0.69

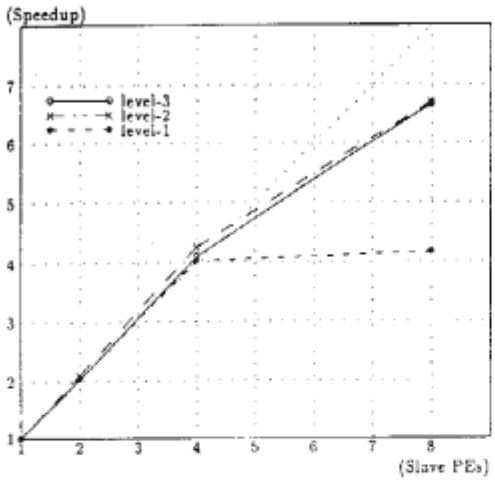


Figure 3: Speedup of Good Simulation

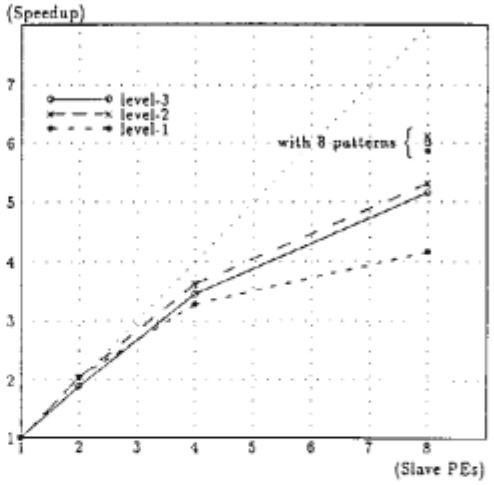


Figure 4: Speedup of Fault Simulation

The fault simulation performance measured by event/sec scale, however, is not excellent. This is partly caused by the parallelization overhead, about 30 %, for the unnecessary gate state saving. On the other hand, the execution time per fault injection is almost constant regardless of the size of circuit. This backs up our observation that the straightforward extension of fault parallel simulation for real parallel processing should fail, as discussed in Section 2.

4.3 Multiprocessor Performance

Figure 3 and Figure 4 show speedups of good and fault simulators on up to 8 slave processors on PIM/m (Multi-PSI's data is quite similar) with 256 input patterns. The parameters Δt, $\Delta t'$ and Δf are 4, 4 and 64, respectively. As for the fault simulation, speedups on 8 slaves with 8 patterns are also shown.

Speedups of good simulations are resemble to the results of similar size circuits in [Matsumoto 92], and are better than those of larger ones in [Mueller-Thuns 89]. This shows that our synchronization mechanism with small Bounding Window works as well as other global synchronization strategies. Speedups for larger circuits are better, and they are improved further, 7.5 for 8 slaves, by enlarging t' as discussed later.

As for the fault simulation, speedups are smaller than good simulations. For the first 8 patterns, however, significantly better results are gotten. A part of the reason of these results is that the number of undetected faults rapidly decreases. That is, local fault simulators become starved and the speedup drops as the simulation progresses. This relationship in the case of the level-1 circuit and 8 slaves is shown in Figure 5.

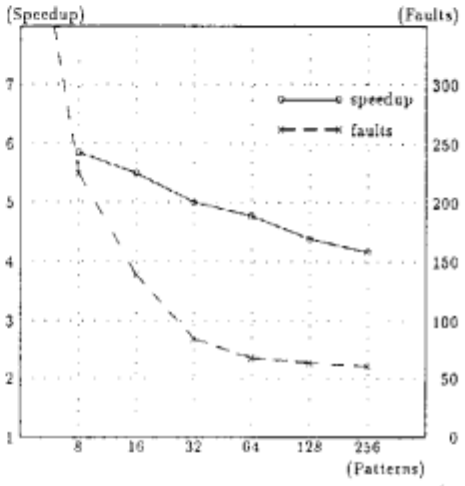


Figure 5: Speedup and Undetected Faults

The other reason is the hot spot at the master processor. The load average of the master processor is nearly 100 % in the case of the fault simulation, while it is approximately 50 % in the good simulation. This is mainly caused by the maintenance of faults such as fault dropping and grouping. These operation can be distributed to slaves to cool the hot spot, if we give up immediate dropping and optimal grouping. This issue is left as a future work.

4.4 Parameters for Synchronization

In order to evaluate the effect of the parallelization described in Section 3, we measured speedups on 8 slaves with 256 patterns varying the parameters Δf, Δt and $\Delta t'$.

Figure 6 shows results for good simulation in which $\Delta t'$, the size of the good trail, varies from 1 to 16. In simulations for all circuits, the optimal value of $\Delta t'$ is 8,

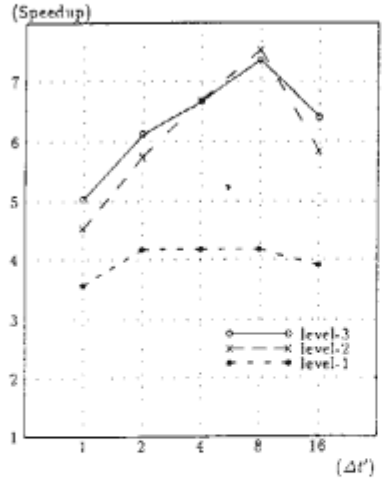


Figure 6: Good Simulation Varying $\Delta t'$

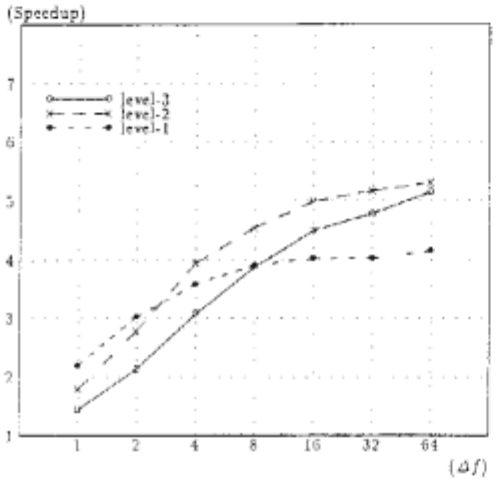


Figure 7: Fault Simulation Varying Δf

with which 7.5 fold speedup is gotten for level-2 circuit. In the fault simulation, however, the values of Δt and $\Delta t'$ can be 2 to get good results, while making them 1 will degrade the performance 10 to 20 %.

On the other hand, the performance of fault simulation is very sensitive to the value of Δf which is the size of the faulty trail, as shown in Figure 7. These figures also show that the growth of circuit size makes the performance more sensitive to parameters, especially to Δf. Therefore, it might be necessary to adjust parameter values according to circuit size. This issue is also left as a future work.

5 Conclusions

In this paper, we presented an efficient fault simulation method for parallel processors which exploits both the fault and circuit parallelism, combining Single Fault Propagation and Time Warp methods. In our fault simulator, multiple fault groups are injected and evaluated concurrently using the faulty trail allocated on each processor element. Faulty circuit states of gates are loaded and saved from/to the faulty trail on-demand. The barrier between the good and fault simulations are broken by modified Bounding Window method in which the good trail contains states in the past not only for rollback but also for the reference from the fault simulator. Owing to these techniques, we got good performance results, about 6 fold fault simulation speedup on 8 slave processors of parallel inference machines.

Although we successfully implemented the simulator and achieved good performance, there is much work to be pursued in future. First of all, we must complete the circuit partitioning routine which will enable us to evaluate simulator's performance with various and large scale circuits. It will be also necessary to modify our method in order to cool the hot spot at the master processor and to adjust the synchronization parameters.

Acknowledgment

We would like to thank Katsuto Nakajima and Masakazu Furuich for their valuable suggestions.

References

[Briner 91] J. V. Briner, Jr., J. L. Ellis, and G. Kedem. Breaking the Barrier of Parallel Simulation of Digital Systems. In *Proc. 28th ACM/IEEE Design Automation Conf.*, pp. 223–226, June 1991.

[Cheng 91] W. Cheng and J. H. Patel. PROOFS. A Super Fast Fault Simulator for Sequential Circuit. In *Proc. European Design Automation Conf.*, pp. 497–502, 1991.

[Chung 89] M. J. Chung and Y. Chung. Data Parallel Simulation Using Time-Warp on the Connection Machine. In *Proc. 26th ACM/IEEE Design Automation Conf.*, pp. 98–103, June 1989.

[Duba 88] P. A. Duba, R. K. Roy, J. A. Abraham, and W. A. Rogers. Fault Simulation in a Distributed Environment. In *Proc. 25th ACM/IEEE Design Automation Conf.*, pp. 686–691, June 1988.

[Gouders 91] N. Gouders and R. Kaibel. PARIS: A Parallel Pattern Fault Simulator for Synchronous Sequential Circuits. In *Proc. IEEE Intl. Conf. on Computer-Aided Design 91*, pp. 542–545, 1991.

[Hong 78] S. J. Hong. Fault Simulation Strategy for Combinational Logic Networks. In *Proc. 8th Intl. Symp. on Fault Tolerant Computing*, pp. 96–99, June 1978.

[Ishiura 87] N. Ishiura, M. Ito, and S. Yajima. High-Speed Fault Simulation Using Vector Processing. In *Proc. IEEE Intl. Conf. on Computer-Aided Design 87*, pp. 10–13, 1987.

[Jefferson 85] D. R. Jefferson. Virtual Time. *ACM Transaction on Programming Languages and Systems*, Vol. 7, No. 3, pp. 404–425, 1985.

[Levendel 83] Y. H. Levendel, P. R. Menon, and S. H. Patel. Parallel Fault Simulation Using Distributed Processing. *Bell System Technical Journal*, Vol. 62, pp. 3107 3137, Dec. 1983.

[Matsumoto 92] 松本 幸則, 龍 和男. バーチャルタイムによる並列論理シミュレーション. 情報処理学会論文誌, Vol. 33, No. 3, Mar. 1992.

[Mueller-Thuns 89] R. B. Mueller-Thuns, D. G. Saab, R. F. Damiano, and J. A. Abraham. Portable Parallel Logic and Fault Simulation. In *Proc. IEEE Intl. Conf. on Computer-Aided Design 89*, pp. 506–509, Nov. 1989.

[Nakashima 92] H. Nakashima, K. Nakajima, S. Kondoh, Y. Takeda, Y. Inamura, S. Ohnishi, and K. Masuda Architecture and Implementation of PIM/m. In *Proc. Intl. Conf. on Fifth Generation Computer Systems 1992*, pp. 425–435, June 1992.

[Narayanan 88] V. Narayanan and V. Pitchumani. A Parallel Algorithm for Fault Simulation on the Connection Machine. In *Proc. IEEE Intl. Test Conf.*, pp. 89 93, Sept. 1988.

[Narayanan 89] V. Narayanan and V. Pitchumani. A Massively Parallel Algorithm for Fault Simulation on the Connection Machine. In *Proc. 26th ACM/IEEE Design Automation Conf.*, pp. 734-737, June 1989.

[Takeda 88] Y. Takeda, H. Nakashima, K. Masuda, T. Chikayama, and K. Taki. A Load Balancing Mechanism for Large Scale Multiprocessor Systems and Its Implementation. In *Proc. Intl. Conf. on Fifth Generation Computer Systems 1988*, pp. 978-986, Nov. 1988.

[Ueda 90] K. Ueda and T. Chikayama. Design of the Kernel Language for the Parallel Inference Machine. *The Computer Journal*, Vol. 33, No. 6, pp. 494–500, 1990.

[Waicukauski 87] J. Waicukauski, E. Eichelberger, D. Forlenza, E. Lindblom, and T. McCarthy. Fault Simulation for Structure VLSI Design. In *VLSI Systems Design*, pp. 20–32. 1987.