Statistics on Distributed Concurrent Fault and Design Error Simulation

Brian Grayson, Saghir A. Shaikh, and Stephen A. Szygenda

TR TR-PDS-1996-006 May 1996

Parallel & Distributed Systems group
Department of Electrical & Computer Engineering
University of Texas at Austin
Austin, Texas 78712

Statistics on Distributed Concurrent Fault and Design Error Simulation

Brian Grayson* Saghir A. Shaikh Stephen A. Szygenda
bgrayson@pine.ece.utexas.edu saghir@mail.utexas.edu szygenda@uts.cc.utexas.edu

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712-1084

May 13, 1996

Abstract

We present useful and unique experimental data and its analysis on load distribution and performance comparisons of three methods of distributed concurrent fault and design error simulations. These statistics will significantly help a simulator designer in evaluating various trade-offs involved in hardware and software acceleration of the simulation process.

Keywords

concurrent fault simulation, concurrent design error simulation, distributed simulation, load balancing, performance, statistics.

1 Introduction

Design verification of digital circuits includes logic, fault and design error simulations [1]. Logic simulation is performed to verify the functionality of the design, and fault and error simulations are done to test the design for design errors and anticipated manufacturing defects. The complexity of fault and error simulations is between G and G^3, (depending on the topology of the circuit) where G is the number of primitives in the circuit. The cost of simulation for typical circuits of thousands of gates is thus extremely large.

Fault simulation is a tool that defines how well the test vectors used to verify a finished product will detect a manufacturing defect. Fault simulation usually provides coverage for a limited set of faults. In general, it includes logical faults modeled by simplistic fault models, such as the stuck-at model, where signals get stuck at a fixed logical value of 0, 1, or X. Fault simulation also assumes that the components are fault free and only their interconnections are affected. Thus, simulation of classical stuck-at faults does not provide a true measure of coverage of all possible design faults and errors. A new measure of design verification coverage, based on the number of design errors detected, was introduced by Kang and Szygenda [1]. A design error is defined as a "design change that causes a different output value from the desired value for some input pattern."[1]. Some techniques to model design errors and use them for design validation have been presented in [1, 2, 3]. The design error models defined by Kang and Szygenda are: *signal*, *gate*, *local*, and *include*. Signal and gate errors model design changes in signals and gates respectively. Local errors represent a double-error situation: both a gate error and a signal error on one of the gate's signals. Include errors consider a design error involving a change in the number of inputs to a gate (i.e., adding an unwanted signal, or removing a desired signal). Basic algorithms of error simulation are similar to those of fault simulation – only the underlying "fault" model has changed. Nevertheless, both fault simulation and design error simulation are vital but expensive tasks of the design cycle.

Concurrent fault simulation is one of several popular techniques which exploit parallelism. Some other techniques are Parallel fault simulation, Deductive fault

*Brian Grayson is supported by a National Science Foundation Graduate Fellowship.

simulation, and Parallel-Pattern Single Fault simulation. Concurrent fault simulation is popular because of its efficiency, generality and accuracy. In the various existing implementations of concurrent fault simulation [4, 5, 6, 7], the basic algorithm is the same; they usually incorporate various heuristics to increase efficiency. The first concurrent design error simulator, named *c_sim*, was developed by Redlien [8] at the University of Texas at Austin. This simulator is capable of gate-level logic, fault, and error simulations. A modified form of *c_sim* was used to collect the data presented here.

Fault or error simulation is a computation-intensive process. It can be accelerated by execution on general-purpose parallel machines or on a network of independent (distributed) workstations [11, 12, 14, 15, 16, 17, 18]. A distributed environment offers a simple, economical and efficient method for acceleration. The simulation task is divided into subtasks, and each subtask is assigned to a workstation in a network. Various efforts for acceleration of fault simulation with this approach have been reported [11, 12, 13]. Banerjee [14] describes five possible methods for partitioning fault simulation for parallel processing: fault-partitioning, pattern-partitioning, circuit-partitioning, pipelining, and a hybrid method. Most of these methods can be applied to various fault simulation algorithms, both for general purpose machines as well as for a distributed environment. Here we have focused on the parallelization of a concurrent algorithm, in a distributed environment, because the concurrent algorithm exploits the inherent parallelism of the fault simulation process at least as efficiently as other techniques [6, 7].

In the fault-partitioning method the modeled faults are partitioned into n groups, where n is the number of processors available. Since the detection of a fault, or the failure to detect that fault, is independent of the detection ability of another fault, no inter-processor communication is needed during the simulation. Each processor executes the simulation algorithm for the entire circuit with a subset of the faults. After all the n processors have finished the assigned task, the individual results need to be merged to reflect the overall fault detection. The main disadvantage of this approach is that each processor simulates the good circuit for all input patterns – thus the good circuit gets simulated n times for each input pattern.

In the pattern-partitioning method the input patterns are partitioned among the n processors. Each processor executes the the concurrent fault simulation for the entire circuit but with a subset of the patterns. Again, no inter-processor communication is needed during simulation, and post-processing is needed to merge results. The advantage of this method over fault-partitioning is that the good circuit is evaluated only once for each input pattern, while the disadvantage is that all of the faults are simulated on all n processors. If there are few faults, this disadvantage becomes inconsequential, since the majority of the work is spent simulating the good circuit. This method, without further enhancements, is limited to combinational circuits.

The third method of circuit-partitioning involves splitting the circuit into subcircuits, and assigning each subcircuit to a processor. Each processor executes the simulation for a subcircuit with all the faults and all the patterns. Although this requires greater inter-processor communication, it does not have either the good simulation overhead of fault-partitioning or the duplicated fault simulation overhead of pattern-partitioning. With a good circuit-partitioning, the inter-processor communication can be minimized. However, a significant amount of pre-processing may be required to ensure a near-optimal partition.

In the pipelined approach the circuit is divided into levels. Each level (or multiple levels) of the circuit is mapped to a processor, and then multiple processors are run in a pipelined fashion. Another way of accelerating the fault simulation process using the pipeline approach is to divide the algorithm in subtasks, and then assign each subtask to one of several processors which execute them in a pipeline fashion [10]. An advantage to this algorithm over the general circuit-partitioning one is that here the decision of where to split the circuit has already been determined. However, in order for a signal to propogate from a primary input to a primary output, it must typically be communicated from the first processor in the pipeline to the last, leading to a much larger amount of communication than the circuit-partitioning algorithm, where many signals go from a gate on a processor to other gates on the same processor.

In the hybrid method, two or more of the above stated methods are combined. Banerjee [14] describes the merits of using the first method, fault-partitioning, for the early phase of fault simulation, and using the second method, pattern-partitioning, for the later phase. As mentioned above, fault-partitioning is most efficient when the number of faults is large (and thus the cost of the good-circuit simulation is small in comparison to the cost of the fault-simulation), and pattern-partitioning is most efficient when the number of faults is small. Since the number of undetected faults drops exponentially with respect to simulation time, the fault-partitioning algorithm should perform well in the early phase of the simulation, and pattern-partitioning should perform well in the long tail of the simulation.

Table 1: Summary of the Benchmark Circuits

Name of Circuit	Function	Number of Gates	Number of Primary Inputs	Number of Primary Outputs
C95	Example	27	15	7
74ls181	ALU	63	14	8
C432	Priority Decoder	160 (18 XOR)	36	7
C499[a]	ECAT[b]	202 (104 XOR)	41	32
C880	ALU and Control	383	60	26
C1355[a]	ECAT	546	41	32
C1908	ECAT	880	33	25
C2670	ALU and Control	1193	233	140
C3540	ALU and Control	1669	50	22
C5315	ALU and Selector	2307	178	123
C6288[c]	16-bit Multiplier	2406	32	32
C7552	ALU and Control	3512	207	108

[a] Circuit *C499* and *C1355* are functionally equivalent. All XOR gates of *C499* have been expanded into their *NAND*-gate equivalents in *C1355*

[b] ECAT – Error Correction and Translation

[c] The simulation on this circuit could not be performed because it required more virtual memory than available at the time of experimentation.

We have chosen to investigate the first, second, and last methods in this research. We used nine combinational circuits, collectively known as ISCAS-85 [9], and additional circuits named *C95* and *74ls181* for the experiments. These circuits are widely accepted as standard benchmarks for comparing various design automation tools.

We present statistics related to distributed fault and design error simulations. Since these statistics are difficult and time consuming to obtain, they have rarely been published. The statistics presented here include performance comparisons and load distribution figures for 4, 8 and 16 processors. The performance comparison results include total execution time, speedup, and average and maximum times on each processor. The load distribution statistics contain the number of good, faulty and erroneous events handled by each processor.

Our primary objective for performing the experimentation was not to report the best speedup obtained as compared to other implementations [11, 12, 13, 14, 15] but to present the useful and unique statistics of the kind described above for the distributed concurrent fault and design error simulation process. Two major motivating factors for the collection of this data and its analysis are:

1. Information obtained from the data on these benchmark circuits may help a simulator designer in evaluating various trade-offs involved in hardware and software acceleration of the simulation process.
2. Increased interest in design error simulation creates a need for this kind of experimental data.

This paper is organized in four sections. This introduction is followed by Section 2, which presents a brief overview of the simulator and methodology used for the experiments. Section 3 presents the results along with their analysis. The paper concludes with a summary, followed by the bibliography.

2 Methodology

The concurrent simulator used for this research is *c_sim*, which was developed at the University of Texas at Austin [8]. This simulator uses a time based, event driven, compiled model algorithm. It is capable of functional, fault and design error simulations at gate-level using three-valued logic. It accepts the circuit description in Standard Intermediate Format (SIF) and simulation patterns in SIM format. The model library consists of C++ functions. Some fault and error collapsing is performed before the simulation is started.

This code was modified to run on a workstation cluster incorporating three techniques - fault-partitioning, pattern-partitioning, and a hybrid method (a combination of both). These three techniques were chosen for

Table 2: Basic Simulation Data

Name of Circuit	No. of Modeled Faults	No. of Modeled Design Errors	No. of Input Patterns	Fault Coverage [%]	Design Error Coverage [%]	Fault Simul. Time [Sec.]	Design Error Simul. Time [Sec.]
C95	158	335	50	98.7	95.8	0.177	0.349
74ls181	344	814	1000	100	90.4	4.490	10.892
C432	818	1837	1000	100	96.6	12.050	21.097
C499	1206	2537	1000	100	100	16.047	20.819
C880	1756	4130	1000	98.7	90.2	25.627	55.876
C1355	2374	5953	1000	100	99.2	46.446	73.174
C1908	3813	8013	1000	98.8	97.3	109.788	176.510
C2670	5725	12071	1000	89.2	87.7	155.806	283.583
C3540	7136	15641	1000	97.4	92.8	164.819	339.204
C5315	10549	23228	1000	99.4	97.9	253.375	338.068
C7552	15231	33633	1000	96.9	95.5	496.643	722.482

their simplicity and ease of implementation. They exploit the coarse-grain parallelism and do not need any inter-processor (or inter-process) communication during the simulation. The two approaches not investigated here, circuit-partitioning and pipeline approach, would require major modifications to the existing simulator code, inter-processor communication, and sophisticated optimization routines for partitioning. The techniques selected for this research require only a simple synchronization barrier for merging of final results at the end of the simulation. Load balancing is currently performed statically, that is, the faults or patterns are randomly divided into n sets. The code could be easily modified to perform some form of dynamic load balancing through a limited amount of communication, but preliminary results showed (and later results confirmed) that the static load balancing did a fairly good job. The distributed simulation can be modeled by a simple client-server model: a server partitions the complete set of faults or patterns, and each client receives one subset and simulates to completion, at which time the client reports its results to the server, which merges all of the results.

Nine ISCAS benchmark circuits and two other model circuits were chosen for this research. For a given circuit, the same set of random input patterns was used for both the fault and the design error simulations. Table 1 on the preceding page shows the summary of the data circuits used and Table 2 presents basic simulation statistics. All simulations were performed on Pentium-90 machines running the NetBSD operating system.

3 Results and Analysis

This section summarizes the performance and load statistics for distributed fault and design error simulation. Figures are presented for four and eight processors for three distribution techniques - fault-partitioning, pattern-partitioning and hybrid. Aditionally, the largest circuit *C7552* was simulated on 16 processors using the hybrid technique. The performance comparison results include total execution time, speedup, and average and maximum times on each processor in the distributed environment. Load distribution statistics contain the number of good, fault and erroneous events assigned to each processor in the distributed environment. The times shown in the tables are user CPU times. Total running time for the simulation includes the partitioning and merging overhead.

3.1 Fault/Error-partitioning

The list of faults is split evenly among the processors. Each of the processors reads the SIF file, the SIM file, and a subset of the fault list. Each processor simulates its subset of faults using the concurrent algorithm. At the end of the simulation the server merges the results obtained from each individual processor. Collected statistics are presented in Table 3 and Table 4.

Figures in Table 3 on the following page show that fault-partitioning did not achieve a significant performance increase. This is due to the large overhead of the good simulations for each circuit. Table 4 shows that the number of good events are the same for the serial simulation and for each processor of the distributed simulation. This is the good-circuit overhead mentioned above that

Table 3: Performance statistics for the fault/error-partitioning

No. of Proc	Name of Circuit	Running Time on Distributed Env. [Sec.]		Speedup		Overhead [Sec.]		Maximum Time on each processor		Average Time on each processor	
		Fault	DE	Fault	DE	Fault	DE	Fault	DE	Fault	DE
	C95	0.68	0.87	0.78	0.75	0.12	0.33	0.19	0.23	0.17	0.22
	74ls181	17.99	25.69	1.01	1.62	0.21	0.41	4.54	6.54	4.50	6.42
	C432	44.63	55.19	1.03	1.48	0.47	0.41	11.31	14.49	11.16	13.80
	C499	57.70	63.23	1.14	1.32	0.41	0.47	14.47	16.08	14.42	15.81
	C880	94.05	123.82	1.13	1.73	0.65	0.78	23.78	32.80	23.51	30.96
4	*C1355*	161.77	183.26	1.15	1.50	0.53	0.54	41.09	49.59	40.44	45.81
	C1908	294.73	371.09	1.49	1.84	0.47	0.71	74.04	96.28	73.68	92.77
	C2670	457.63	597.48	1.39	1.91	1.67	1.82	115.76	150.63	114.41	149.37
	C3540	477.04	632.88	1.36	2.11	0.66	1.02	122.11	161.52	119.26	158.22
	C5315	908.50	983.22	1.14	1.38	1.50	1.88	228.80	250.15	227.13	245.80
	C7552	1746.17	1928.45	1.16	1.49	2.03	5.14	440.47	492.67	436.54	482.11
	C95	1.33	1.57	0.52	0.57	0.27	0.53	0.20	0.22	0.17	0.20
	74ls181	35.85	44.00	0.94	1.75	0.55	0.70	4.54	5.75	4.48	5.50
	C432	88.02	99.19	1.08	1.56	0.78	1.01	11.17	13.10	11.00	12.40
	C499	111.61	118.28	1.15	1.37	0.79	0.82	14.02	15.09	13.95	14.79
	C880	185.81	214.39	1.12	2.02	1.09	1.11	23.52	27.62	23.23	26.80
8	*C1355*	317.28	326.94	1.16	1.68	1.02	1.06	40.23	43.53	39.66	40.87
	C1908	546.24	611.24	1.59	2.23	0.86	1.16	69.23	78.69	68.28	76.41
	C2670	872.78	1038.86	1.44	2.16	3.32	3.34	110.23	131.65	109.10	129.86
	C3540	902.06	1031.62	1.45	2.57	1.24	1.48	114.20	131.81	112.76	128.95
	C5315	1786.36	1853.08	1.16	1.47	2.64	3.02	224.16	234.84	223.29	231.64
	C7552	3277.45	3451.13	1.21	1.69	3.45	3.77	422.15	435.28	409.68	431.39

Table 4: Load (Events) distribution statistics for the fault/error-partitioning approach

No. of Proc.	Name of Circuit	Good Events		Faulty Events		Erroneous Events	
		Serial Simulation	Avg. on Each Processor for Distributed Simulation	Serial Simulation	Avg. on Each Processor for Distributed Simulation	Serial Simulation	Avg. on Each Processor for Distributed Simulation
	C95	2222	2222	3612	954	10656	2779
	74ls181	1023041	102341	8586	2214	240173	63223
	C432	299911	299911	62570	15997	413042	108594
	C499	385206	385206	131050	33248	351190	90072
	C880	596494	596494	117421	30856	1376720	353351
4	*C1355*	1079664	1079664	532049	133956	1694629	425313
	C1908	1750178	1750178	2164323	550014	5461921	1381507
	C2670	2259715	2259715	2992050	758331	8668845	2181213
	C3540	2838626	2838626	2829237	725713	10949123	2760050
	C5315	5169920	5169920	1610339	409583	5376370	1360075
	C7552	7412930	7412930	4351626	1098450	14221136	3577614
	C95	2222	2222	3612	493	10656	1440
	74ls181	1023041	1023041	8586	1137	240173	33488
	C432	299911	299911	62570	8174	413042	56671
	C499	385206	385206	131050	16844	351190	45871
	C880	596494	596494	117421	16072	1376720	181997
8	*C1355*	1079664	1079664	532049	67474	1694629	213772
	C1908	1750178	1750178	2164323	278932	5461921	699657
	C2670	2259715	2259715	2992050	385716	8668845	1099382
	C3540	2838626	2838626	2829237	373793	10949123	1394841
	C5315	5169920	5169920	1610339	207581	5376370	688199
	C7552	7412930	7412930	4351626	555269	14221136	1800183

is inherent to the fault-partitioning approach. Faulty and erroneous events are reduced by the factor of n, and since the good events roughly equal the faulty events for these simulations, the speedup is maximally bounded by 2. The design error acceleration shows better speedup because the number of erroneous events are more than the number of faulty events. There is not much difference between the maximum and the average running times on a processor. This fact shows that this technique of static load balancing provides a fairly good load balance.

Table 5: Performance statistics for the pattern-partitioning

No. of Proc	Name of Circuit	Running Time on Distributed Env. [Sec.]		Speedup		Overhead [Sec.]		Maximum Time on each processor		Average Time on each processor	
		Fault	DE	Fault	DE	Fault	DE	Fault	DE	Fault	DE
	C95	0.67	1.23	0.46	0.61	0.33	0.38	0.20	0.32	0.17	0.31
	74ls181	5.99	14.65	2.51	2.74	0.31	0.35	1.59	3.76	1.50	3.66
	C432	17.77	35.83	2.31	2.14	0.53	0.87	4.71	9.43	4.44	8.96
	C499	28.02	55.62	2.12	1.35	0.48	0.78	7.54	15.35	7.00	13.90
	C880	32.02	77.54	3.03	2.72	0.78	1.26	8.29	20.01	8.00	19.39
4	*C1355*	86.12	194.85	2.08	1.43	0.78	1.05	22.25	51.53	21.53	48.71
	C1908	193.13	374.52	1.83	1.44	0.77	1.38	60.07	122.54	48.28	93.63
	C2670	226.92	447.84	2.63	2.41	2.48	4.06	59.71	116.87	56.73	111.96
	C3540	283.24	609.52	2.27	2.16	1.46	2.77	72.14	156.12	70.81	152.38
	C5315	377.58	632.85	2.61	2.03	2.02	2.95	98.62	169.32	94.40	158.21
	C7552	686.03	1154.07	2.91	2.46	3.07	5.03	173.23	296.70	171.51	288.52
	C95	1.23	2.22	0.54	0.58	0.27	0.38	0.18	0.36	0.15	0.28
	74ls181	7.54	18.11	3.31	3.94	0.36	0.49	1.08	2.37	0.94	2.26
	C432	24.85	53.78	3.31	2.82	0.65	0.82	3.26	7.00	3.11	6.72
	C499	41.33	93.58	2.66	1.61	0.67	0.62	5.71	12.95	5.17	11.70
	C880	41.19	107.38	4.41	3.79	0.91	1.42	5.32	13.87	5.15	13.42
8	*C1355*	133.27	341.20	2.60	1.57	0.83	1.10	17.60	46.60	16.66	42.65
	C1908	278.72	552.88	2.21	1.68	0.88	1.62	49.35	104.33	34.84	69.11
	C2670	275.78	595.25	4.04	3.51	4.42	4.55	36.08	78.50	34.47	74.41
	C3540	429.65	946.27	2.86	2.65	1.85	3.33	56.75	126.01	53.71	118.28
	C5315	550.35	1043.72	3.04	2.06	2.15	3.38	84.18	165.64	68.79	130.46
	C7552	890.78	1637.27	4.29	3.45	5.22	5.53	114.42	209.49	111.35	204.66

Table 6: Load (Events) distribution statistics for the pattern-partitioning approach

No. of Proc.	Name of Circuit	Good Events		Faulty Events		Erroneous Events	
		Serial Simulation	Avg. on Each Processor for Distributed Simulation	Serial Simulation	Avg. on Each Processor for Distributed Simulation	Serial Simulation	Avg. on Each Processor for Distributed Simulation
	C95	2222	622	3612	3234	10656	8574
	74ls181	123041	30897	8586	8616	240173	78233
	C432	299911	75267	62570	71376	413042	238344
	C499	385206	96686	131050	156232	351190	407253
	C880	596494	149754	117421	94336	1376720	523906
4	*C1355*	1079664	270792	532049	558963	1694629	1622102
	C1908	1750178	438730	2164323	1443603	5461921	3418306
	C2670	2259715	566870	2992050	1434608	8668845	3890837
	C3540	2838626	711931	2829237	1985938	10949123	5698294
	C5315	5169920	1296087	1610339	1583887	5376370	4240632
	C7552	7412930	1858157	4351626	2876898	14221136	8107377
	C95	2222	355	3612	2849	10656	7244
	74ls181	123041	15540	8586	8323	240173	50665
	C432	299911	37826	62570	68362	413042	200469
	C499	385206	48599	131050	151797	351190	386658
	C880	596494	75297	117421	84720	1376720	364723
8	*C1355*	1079664	135980	532049	545983	1694629	1542482
	C1908	1750178	220155	2164323	1168651	5461921	2704285
	C2670	2259715	284729	2992050	958325	8668845	2533843
	C3540	2838626	357482	2829237	1746824	10949123	4495211
	C5315	5169920	650448	1610339	1645821	5376370	4176285
	C7552	7412930	932362	4351626	2474385	14221136	6554099

3.2 Pattern-partitioning

In this approach the pattern file is partitioned into n subsets and distributed onto n processors. Each processor reads the SIF file and simulates all the faults concurrently with the assigned subset of patterns. The results obtained from all the processors are merged at the end of the simulation. Statistics related to the pattern-partitioning approach are presented in Table 5 and Table 6.

Again, the pattern-partitioning approach did not provide an ideal, nearly-linear in n acceleration for most

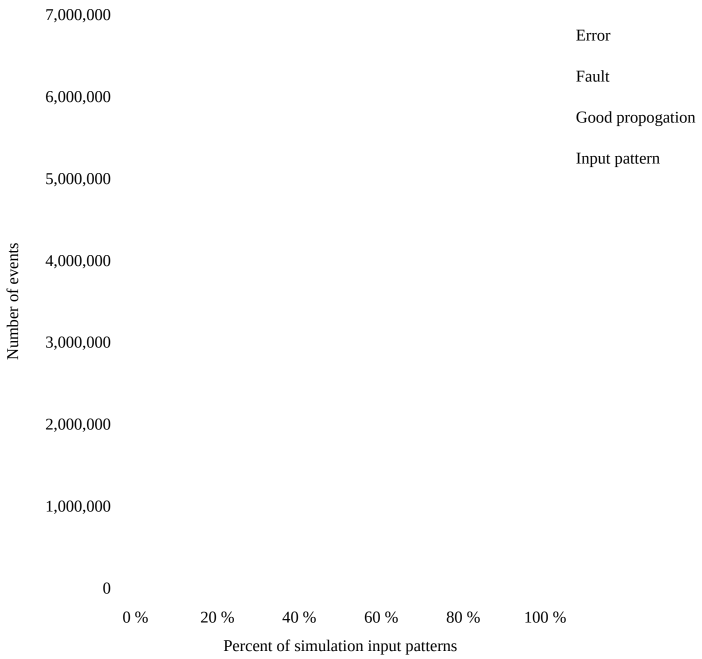

Figure 1: Good, Faulty and Error Event distribution through the course of simulation for C7752 circuit.

circuits. Typically, the speedup stayed in the vicinity of 3. Only in one case of simulation on eight processors did it reach a figure near 4. While the number of good events was reduced by a factor of n for each processor, the number of faulty and erroneous events in most cases was not reduced nearly as much. Due to the absence of inter-processor communication in this approach, some faults (errors) are redundantly detected. Thus the distributed simulation has more effective number of faulty or erroneous events (i.e. total of all the faulty/erroneous events on all processors) than the serial simulation. However, in some cases fault and error events were reduced as much as 30% compared to the serial simulation. This is due to the fact that every processor has a reduced set of input patterns.

3.3 Hybrid Method

The hybrid method combines the advantages of both the fault-partitioning and pattern-partitioning approaches. This method uses fault-partitioning when the number of faults is high (i.e. during the initial phase of the simulation) and pattern-partitioning when the number of undetected faults is much lower (i.e. during the later phase of the simulation). We chose to run 5% of the input patterns with the fault-partitioning phase, and the remaining 95% of the input patterns for the pattern-partitioning phase. This choice was made by observing the event queue distribution of some sample circuits throughout the course of the simulation, and choosing a point at which the majority of faults had been detected. In one of the experiments on circuit *C7552*, we collected the numbers of good, faulty and erroneous events generated for several segments of the simulation. These results are presented in Fig. 1.

The time distribution of dequeued events in Fig. 1 shows that in the early phase of fault (error) simulation, faulty (erroneous) events greatly outnumber good events (because all the faults are being simulated concurrently – none have yet been detected), whereas toward the end of the simulation, the ratio of faulty (error) events to good events has dropped significantly (because only a few undetected faults remain to be simulated concurrently). Statistics related to the hybrid method are summarized in Tables 7 through 9. For this technique we also present the results obtained from the simulations of the largest circuit, *C7552*, on sixteen processors.

The hybrid approach produced significantly better performance than the fault/error-partitioning and pattern-partitioning approaches.

In the first phase only 5% of the patterns are used for the simulation, so load distribution statistics show that

Table 7: Performance statistics for fault simulation for hybrid approach

No. of Proc.	Name of Circuit	Running Time on Dist. Env. [Sec.]	Speedup	Overhead [Sec.]	Max. Time on Each Proc. in F-P Phase [Sec.]	Avg. Time on Each Proc. in F-P Phase [Sec.]	Max. Time on Each Proc. in P-P Phase [Sec.]	Avg. Time on Each Proc. in P-P Phase [Sec.]
4	*C95*	0.77	0.31	0.54	0.10	0.09	0.14	0.13
	74ls181	2.46	1.94	0.75	0.42	0.38	1.29	1.24
	C432	5.20	2.33	1.07	1.03	1.00	3.09	2.98
	C499	6.63	2.56	1.33	1.52	1.49	3.78	3.70
	C880	9.89	2.78	1.61	1.87	1.80	6.41	6.30
	C1355	16.38	2.93	1.50	5.12	4.87	9.76	9.64
	C1908	47.23	2.35	1.85	12.96	12.62	32.42	28.02
	C2670	55.53	2.95	6.03	9.08	8.89	40.42	39.93
	C3540	57.49	2.91	3.25	15.76	14.52	38.49	37.52
	C5315	85.34	3.08	3.71	20.96	20.74	60.67	59.88
	C7552	170.79	3.00	6.33	38.27	36.49	126.19	122.08
8	*C95*	1.55	0.16	1.30	0.11	0.08	0.14	0.12
	74ls181	2.32	2.06	1.19	0.35	0.33	0.79	0.72
	C432	4.64	2.79	1.99	0.90	0.84	1.75	1.66
	C499	5.32	3.19	1.91	1.17	1.14	2.24	2.11
	C880	7.83	3.51	2.65	1.62	1.59	3.56	3.47
	C1355	11.37	4.22	2.10	3.64	3.42	5.62	5.31
	C1908	34.85	3.19	3.38	8.33	8.06	23.15	18.65
	C2670	38.15	4.29	8.93	7.57	7.41	21.65	20.88
	C3540	37.98	4.41	4.42	10.99	10.39	22.56	21.54
	C5315	56.31	4.66	5.48	17.24	16.96	33.59	32.33
	C7552	108.06	4.75	10.04	30.79	30.10	67.22	65.19
16	*C7552*	83.90	6.12	17.77	27.33	26.67	38.79	36.07

good events are reduced to 5% of the serial simulation, and faulty/erroneous items are reduced by a factor of 4 to 8. In the second phase, the good work has been reduced by a factor of n, and since each processor is not doing as much redundant work (many faults were eliminated in the first phase), the fault/error work has been reduced by a factor of 5 to 10 or more.

Again, the total number of good and faulty/erroneous events were roughly reduced by a factor of 3. Although the other two methods achieved an n-fold reduction in one type of events (faulty/erroneous events for the fault-partitioning method, good events for the pattern-partitioning method), both these methods had little or no reduction in the other type of events. Thus, the hybrid method resulted in an overall decrease in the number of events required during a simulation, and thus a decrease in the total work done by the CPU.

For each phase of the simulation, the average time roughly equals the maximum time of each processor. This provides further evidence that executing 5% of the simulation for the first phase is appropriate for good load balancing for this set of circuits. However, this does not imply that the best possible partition of the work between the fault-partitioning and pattern-partitioning phases in all cases was achieved (e.g. it is possible, although highly unlikely, that if the fault-partitioning phase had used one or a few more patterns, it might have detected all the remaining undetected faults with only a small increase in CPU time). On the other hand, since the running times for the two phases are usually within a factor of 2 of each other, the results do prove that this choice takes no more than two or three times as much CPU time as the optimal choice for the partitioning algorithm crossover point. Any transition point that occurs after a significant number of faults/errors have been detected should be sufficient to achieve speedup.

4 Summary

Our experimentation on acceleration of concurrent fault and design error simulations on a network of workstations provided us with two kinds of statistics: performance comparisons and load distribution. The performance comparison results include total execution time, speedup, and average and maximum times on each processor. The load distribution statistics contain the number of good, faulty and erroneous events assigned to each processor. The results are presented for four and eight processors for three distribution techniques - fault-partitioning, pattern-partitioning and a hybrid method (which is a combination of both). Additionally, the largest circuit *C7552* was simulated on 16 processors using the hybrid technique. The analysis of data presented leads to the following conclusions.

Table 8: Performance statistics for design error simulation for hybrid approach

No. of Proc.	Name of Circuit	Running Time on Dist. Env. [Sec.]	Speedup	Overhead [Sec.]	Max. Time on Each Proc. in F-P Phase [Sec.]	Avg. Time on Each Proc. in F-P Phase [Sec.]	Max. Time on Each Proc. in P-P Phase [Sec.]	Avg. Time on Each Proc. in P-P Phase) [Sec.]
4	*C95*	1.19	0.36	0.83	0.13	0.12	0.23	0.20
	74ls181	4.24	2.66	0.86	0.55	0.53	2.83	2.73
	C432	7.91	2.79	1.31	1.60	1.53	5.00	4.97
	C499	8.54	2.55	1.27	2.38	2.27	4.89	4.74
	C880	18.80	3.08	2.26	2.99	2.95	13.55	13.34
	C1355	24.68	3.04	1.71	10.80	10.29	12.17	11.85
	C1908	78.41	2.27	2.09	23.47	22.56	52.85	44.54
	C2670	90.79	3.21	4.63	13.98	13.91	72.18	71.69
	C3540	110.56	3.10	3.81	27.02	26.59	79.74	77.98
	C5315	110.87	3.15	4.27	32.31	31.51	74.29	73.45
	C7552	245.25	3.02	6.66	56.66	56.16	181.93	177.45
8	*C95*	1.63	0.26	1.31	0.12	0.09	0.20	0.16
	74ls181	3.77	2.98	1.74	0.42	0.40	1.60	1.49
	C432	5.93	3.72	1.96	1.18	1.09	2.78	2.71
	C499	6.84	3.19	2.15	1.59	1.53	3.10	2.90
	C880	13.32	4.35	3.36	2.19	2.13	7.77	7.40
	C1355	16.07	4.67	2.49	6.35	6.07	7.23	6.89
	C1908	58.42	3.05	3.08	13.59	13.05	41.76	32.12
	C2670	60.63	4.81	8.24	10.51	10.28	41.88	40.20
	C3540	68.24	5.02	6.25	15.97	15.69	46.02	43.81
	C5315	71.44	4.88	5.84	22.80	22.23	42.79	41.49
	C7552	151.29	4.90	9.47	41.24	40.07	100.57	96.96
16	*C7552*	108.93	6.81	16.92	32.58	31.86	59.44	55.33

- Our primary objective for performing the experimentation was not to report the best speedup, but to achieve a fairly good speedup with simple parallelizing techniques. The speedup for the largest circuit, *C7552*, was through the hybrid method. The values for the design error simulation are 3.02, 4.90 and 6.81 for 4, 8 and 16 processors respectively.

- The hybrid technique offers the best performance (speedup) as compared to fault-partitioning and pattern-partitioning. The speedup is higher for larger circuits.

- The average and maximum running times for the distributed simulations are usually relatively close, across all three methods. These simple partitioning algorithms provide good load-balancing among the processors.

- In most of the cases just 1000 random patterns were sufficient to provide good fault and error coverage. For higher fault coverage more input patterns would be required.

- Banerjee [14] has presented some results of acceleration of parallel fault simulation algorithms using these techniques. This research has shown that the same parallelism methods can be used for accelerating concurrent fault and design error simulations.

- Design error simulations show better speedup than the fault simulations, since the ratio of design error events to good events is higher than the ratio of fault events to good events.

- For relatively small circuits like these, the speedup for the hybrid method starts to level off at 8 or 16 processors. This is due mostly to the fact that the number of faults or input patterns is small enough that the overhead of duplicated effort in the pattern-partitioning phase becomes significant. An improvement to the hybrid method is being investigated that adds a small amount of communication to the pattern-partitioning phase to reduce the occurrence of multiple detection of the same fault. In addition, experiments with larger circuits will show whether or not the hybrid algorithm is scalable up to dozens or hundreds of processors.

Table 9: Load (Events) distribution statistics for the hybrid approach

No. of Pr-oc.	Name of Circuit	Good Events			Faulty Events			Erroneous Events		
		Serial Sim.	Avg. on a Proc. in F-P phase	Avg. on a Proc. in P-P phase	Serial Sim.	Avg. on a Proc. in F-P phase	Avg. on a Proc. in P-P phase	Serial Sim.	Avg. on a Proc. in F-P phase	Avg. on a Proc. in P-P phase
4	*C95*	2222	236	585	3612	625	1123	10656	1543	3886
	74ls181	123041	12036	27934	8586	2214	0[a]	240173	11344	49391
	C432	299911	29885	67892	62570	14840	3412	413042	42735	69245
	C499	385206	37999	87314	131050	32460	2981	351190	72656	29246
	C880	596494	60286	134893	117421	21874	14833	1376720	85167	280912
	C1355	1079664	109220	243780	532049	133391	2110	1694629	368396	76662
	C1908	1750178	177512	394747	2164323	460605	236010	5461921	1033496	722104
	C2670	2259715	226707	256820	2992050	183327	350426	8668845	4878857	1060344
	C3540	2838626	286304	641113	2829237	421688	436706	10949123	1062901	2015294
	C5315	5169920	516351	580791	1610339	355441	86382	5376370	904411	388105
	C7552	7412930	738755	842057	4351626	580842	416126	14221136	1506815	1559087
8	*C95*	2222	236	337	3612	321	824	10656	783	2809
	74ls181	123041	12036	14058	8586	1137	0[1]	240173	5866	24878
	C432	299911	29885	34139	62570	7567	2220	413042	21722	36709
	C499	385206	37999	43913	131050	16420	2640	351190	36690	22127
	C880	596494	60286	67867	117421	11205	9677	1376720	43153	147429
	C1355	1079664	109220	122474	532049	67189	2053	1694629	1542482	184941
	C1908	1750178	177512	198164	2164323	231525	182107	5461921	518038	507400
	C2670	2259715	226707	129704	2992050	92479	183382	8668845	244875	553234
	C3540	2838626	286304	322073	2829237	212269	254337	10949123	532980	1105063
	C5315	5169920	516351	292800	1610339	179115	55362	5376370	453792	226994
	C7552	7412930	738755	424312	4351626	291462	238882	14221136	754786	865281
16	*C7552*	7412930	738755	219038	4351626	144404	132349	14221136	381516	471977

[a] Evaluations in pattern-partitioning phase for *74ls181* are 0 because all of the faults were detected in the fault-partitioning phase.

References

[1] Kang, S., and S. Szygenda, "Design Validation: Comparing Theoretical and Empirical Results of Design Error Modeling," *IEEE Design and Test of Computers*, March, pp. 18-26, 1994.

[2] Wang, Y., P. Chung, and I. Hajj, "Logic Design Error Diagnosis and Correction," *IEEE Transactions on Very Large Scale Integration*, Vol. 2, No. 3, pp.320-332, September 1994.

[3] Zhang, Q., and C. Trullemans, "Logic Verification of Incomplete Functions and Design Error Location," *Correct Hardware Design and Verification Methods, Lecture Notes in Computer Science 683*, Springer Verlag, pp. 68-79, May 1993.

[4] Schuster, M., and R. Bryant, "Concurrent Fault Simulation of MOS Digital Circuits," *Proc. of Conference on Advanced Research in VLSI*, M.I.T., pp.129-138 January 1984.

[5] Davidson, S., and J. Lewandowski, "ESIM/AFS-A Concurrent ArchitecturalLevel Fault Simulator," *Proc. of IEEE Intl. Test Conf.*, pp. 375-383, 1986.

[6] Rogers, W., J. Guzolek, and J. Abraham, "Concurrent Hierarchical Fault Simulation: A Performance Model and Two Optimizations," *IEEE Transactions of Computer-Aided Design*, Vol. 6, No. 5, pp. 848-862, September 1987.

[7] Gai, S., and L. Montessoro, "Creator: New Advanced Concepts in Concurrent Simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and System*, Vol. 13, No. 6, pp. 786-795, June 1994.

[8] Redlien, P. "A Technique for Concurrent Design Error and Fault Simulation," *Master's Thesis*, The University of Texas at Austin, August, 1993.

[9] Brglez, F., and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN," *Proc. of IEEE International Symposium on Circuits and Systems; Special Session on ATPG and Fault Simulation*, pp. 695-698, June 1985.

[10] Agrawal, P., V. Agrawal, K. Cheng, and R. Tutundjian, "Fault Simulation in a Pipelined Multiprocessor System," *Proc. of IEEE Int. Test Conf.*, pp. 727-734, September 1989.

[11] Duba, P., R. Roy, J. Abraham and W. Rogers, "Fault Simulation in a Distributed Environment," *Proc. of ACM/IEEE Design Automation Conference*, pp. 686-691, 1988.

[12] Markas, T., M. Royals, and N. Kanopoulos, "On Distributed Fault Simulation," *IEEE Computer*, Vol. 23, No. 1, pp. 40-52, January 1990.

[13] Muller-Thuns, R., D. Saab, R. Damiano, and J. Abraham, "VLSI Logic and Fault Simulation on General-Purpose Parallel Computers," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 3, pp.446-460, March 1993.

[14] Banerjee, P., *Parallel Algorithms in VLSI Computer Aided Design*, Englewood Cliffs, N.J., 1994.

[15] Patil, S., and P. Banerjee, "Performance Tradeoffs in a Parallel Test Generation/Fault Simulation Environment," *IEEE Trans. on Computer Aided Design*, Vol. 10, No. 12, pp.1542-1558, December 1991.

[16] Niermann, T., W. Cheng, and J. Patel, "PROOFS: A Fast, Memory-efficient Sequential Circuit Fault Simulator," *IEEE Trans. on Computer Aided Design of Integrated Cirsuit Systems*, Vol. 11, No. 2, pp. 198-207, Febuary 1992.

[17] Ostapko, D., Z. Barzilai, and G. Silberman, "Fast Fault Simulation in a Parallel Processing Environment," *Proc. of Intl. Test Conf.*, 1987.

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