FAULT SIMULATION IN A DISTRIBUTED ENVIRONMENT*

Patrick A. Duba, Rabindra K. Roy, Jacob A. Abraham, and *William A. Rogers*

Computer Systems Group
Coordinated Science Laboratory
University of Illinois
Urbana, IL 61801

Department of Electrical & Computer Engineering
University of Texas at Austin
Austin, TX 78712

Abstract

Fault simulation of VLSI circuits takes considerable computing resources and there have been significant efforts to speed up the fault simulation process. This paper describes a distributed fault simulator implemented on a loosely-coupled network of general purpose computers. The techniques used result in a close to linear speedup and can be used effectively in most industrial VLSI CAD environments.

I. Introduction

Test generation is an important phase in the successful design and validation of a circuit. It has been determined that test generation is an NP-complete problem [1,2]; thus, the computational requirements grow exponentially as the circuit size increases. The problem is much worse for sequential circuits, where the search space must also consider one more dimension, that of time. Efficient automatic test pattern generators for sequential circuits are not yet available, so the burden is laid on the test engineers, who have to generate test sequences from their experience. Fault simulation involves the process of applying such test patterns to a circuit model in order to validate their effectiveness. For a given fault model, the quality of the test set is expressed as the *fault coverage*, which is the percentage of faults detected out of all the possible faults. Fault simulators are thus indispensable tools in the test generation process. Considering the fact that a significant portion of the cost of producing an IC is due to testing, it is essential to perform the fault simulation in as short a time as possible, to reduce both the design-test-redesign cycle and the time the produced chip spends on the test bed.

In its first form, fault simulation was performed serially, injecting one fault at a time into the circuit. The inadequacy of this approach became evident with the development of MSI and LSI circuits which prompted researchers to find new methods of fault simulation. At present, the three most prevailing techniques used for fault simulation are parallel, deductive and concurrent [3]. For simulating very large circuits, concurrent fault simulation is considered to be the best approach [4]. The costs of parallel and deductive fault simulation are $O(G^3)$ and at least $O(G^2)$, respectively [5], where G is the number of gates in the circuit. A formal model shows that the cost of concurrent fault simulation is worse than linear for realistic circuits [4]; based on observations, concurrent fault simulation seems to behave as an $O(G^2)$ algorithm [6]. As a result, when these algorithms are implemented on a general purpose processor they take a prohibitive amount of time for current VLSI circuits. This has led to research into making the fault simulation process faster. There are three ways to achieve this: by making a simpler model, which suffers from lack of accuracy, or by using a more efficient algorithm, such as one which exploits hierarchy in the circuit [7], or by using special purpose hardware accelerators, e.g. IBM's Yorktown Simulation Engine (YSE) [8], NEC's Hardware Accelerator (HAL) [9], Zycad, Silicon Solutions, Daisy, etc. [10,11].

The hardware accelerators no doubt provide the fastest simulation, but they have some severe drawbacks. They are very expensive in initial cost as well as maintenance, they are very inflexible because of their fixed architectures, and it is not possible to incorporate new features into them. Moreover, they have fixed fault libraries, which are applicable to only one or a few technologies, and any major change in the prevailing technology may lead to their obsolescence. Hence, there is an interest in software simulators, which are flexible and can be run on general purpose computers.

An alternative to hardware accelerators is the use of distributed and parallel processing for fault simulation. The distributed processing approach is applicable to general purpose processors and the implementation can be made flexible enough to incorporate changes in technology. In almost all of the cases, distributed processing is less expensive when compared to hardware accelerators. In the late 1980s, there has been a strong trend towards networking. Almost all the IC design houses have a network of engineering workstations. Most of the workstations in a network are idle during nonwork hours. If all these workstations are used as a distributed processing system to work on the same problem, it is possible to solve problems that are too complex for most mainframes. This led to the investigation and development of the distributed fault simulator described in this paper.

Implementing an algorithm in a distributed environment is not a simple task. The problem domain must be decomposed among the processors [12]; this is also known as *task partitioning*. Stated in simple words, task partitioning is nothing but breaking the original problem into several smaller subproblems. Another important issue is *task allocation*, i.e., mapping the subproblems to the processors. If task partitioning and task allocation are not performed properly, an increase in the number of processors in a system may actually result in a decrease of total throughput [13]. This decrease can occur if a key factor called *interprocessor communication (IPC)*, which consists of all messages exchanged among processors and shared memories, is not taken into consideration [14].

This communication overhead, which is absent in the analyses of sequential algorithms, can be a bottleneck if not handled efficiently. Granularity of computation (the smallest size of a task given to a processor) should be based on the parameters of a particular system. If the granularity is chosen to be small, then the subproblems can be divided evenly among the processors. This ensures better utilization of the processors, but the amount of interprocessor communication becomes large. Therefore, if communication is the bottleneck of a system, granularity should be large. It is very difficult to predict the exact granularity size which will yield the best performance for a given problem and distributed system. However, some broad ranges, based on computation and communication requirements for each granule size, can be specified within which the algorithm is expected to behave reasonably.

Fault simulation algorithms, which were developed for sequential

*This work was supported by the Semiconductor Research Corporation under Contract 87-DP-109.

Patrick A. Duba is currently with Hewlett Packard EDD, Colorado Springs, Colorado.

processors, are not suitable for distributed implementation in their original forms. One of the reasons is the difficulty in properly partitioning the given task into subtasks. However, novel techniques developed recently provide a way to overcome this difficulty [4]. These techniques automatically partition the problem into several smaller and somewhat similar sized subproblems using information about the circuit hierarchy. The partitioning is done in such a way that each subproblem gets a unique subset of the set of all the possible faults in the circuit. Thus the subsets of faults associated with different subproblems are mutually disjoint. As a result of this, the task allocation problem is greatly simplied.

This paper describes the implementation of a distributed fault simulator running on a loosely coupled network of general purpose processors connected via a bus. A large number of simulations were run on circuits of varying sizes and total number of faults. For a given problem, there were two parameters to be decided upon: the partition size (how many faults are injected into each partition) and the number of processors used. When compared to a serial run, many situations yielded a near linear speedup. As the circuit size, and hence the total number of faults, increased the speedup figures were found to improve.

The paper is divided into five sections. Section I gives the introduction to the problem. Section II describes the method of hierarchical fault simulation, a hierarchical fault simulator (CHIEFS), and hierarchical fault partitioning. The distributed algorithm and implementation are described in section III. Section IV reports the results and explains various anomalies in the performance graphs and section V concludes the paper.

II. Hierarchical fault simulation

The computationally intensive task in fault simulation is evaluation of logic elements. Fault simulation speed can be increased by reducing the number of evaluations or by increasing the speed of element evaluation. If the evaluation rate is increased, the simulation will go faster, but the computational complexity of the algorithm remains unchanged. Since the complexity of fault simulation is worse than linear [4], the computational requirements for circuit evaluations will increase faster than the corresponding increase in circuit complexity. Reducing the number of evaluations decreases the complexity of the algorithm and provides increased performance from existing hardware. An event driven simulation performs evaluations only when signals change, so there is no scope of reduction in the number of evaluations in the event driven algorithm. Hence the only other alternative is to reduce the number of primitives in the circuit by altering the circuit representation. Hierarchy provides the framework to make these changes and hierarchical design practices provide the information required to construct a hierarchical circuit representation. The fault simulator, CHIEFS (Concurrent HIerarchical and Extensible Fault Simulator), has demonstrated that these changes are feasible and has provided dramatic simulation speedup under appropriate conditions [4, 7].

A. Description of the CHIEFS System

The CHIEFS system consists of several C programs, two parsers, a fault simulator, and a wrapper. These programs contain about 20,000 lines of code, which were developed under UNIX. The parsers were written with LEX [15] and YACC [16] to parse the fault and circuit description languages. These parsers produce a common data structure to be used by the fault simulator and the wrapper. The circuit description languages currently accepted are SCALD [17, 18] and TDL [19]. The fault library language was developed in-house. Figure 1 shows the relationship among various parts of the simulator.

B. Hierarchical Fault Partitioning

CHIEFS features a hierarchical fault partitioning technique which increases simulator performance by reducing the total amount of computation required to perform the simulation. This technique is based on the hierarchical circuit description and (redundant) high level functional

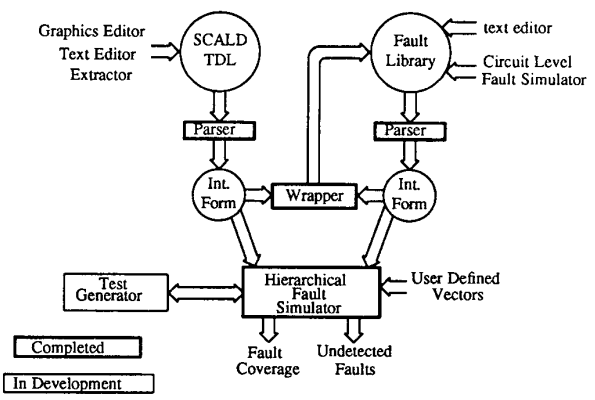

Figure 1. CHIEFS Simulation System Diagram

definitions for the higher level macromodules in the circuit description. Initially, the circuit description is *automatically* partitioned, using hierarchical information, into subcircuits each of which contains a small subset of all the possible faults, as shown in Figure 2. During simulation, the circuit description is reconfigured so that the current fault partition is simulated at the primitive level and all other (fault-free) parts of the circuit are simulated at the highest possible levels. This is achieved by substituting the high-level functional definition for subcircuits wherever possible in the fault-free areas. Fault simulation is performed and the circuit is reconfigured; then the next partition is faulted and the previous partition is now fault-free. This reconfiguration process is repeated until all partitions have been fault simulated. A performance model shows that the total time for simulating many passes is indeed much less than simulating everything in one pass for very large circuits [4]. It is instructive to note that this technique partitions the given problem into subproblems that do not require information from each other during evaluation. If, in a distributed environment, each processor simulates a disjoint set of fault partitions, many fault partitions can be evaluated in parallel. This was the key idea behind the distributed implementation of this fault simulator.

C. Partitioning Algorithm

The hierarchical partitioning algorithm creates a series of circuit representations (partitions), each of which is a complete representation of the circuit. Each partition has a unique partition number associated with it and considers a unique set of modules for fault injection. In other words, each partition has a a unique set of faults for simulation. The partitioning algorithm uses a user-specified number of faults (partition size) as a guide in creating a partition. This recommended size has a -25% to +50% tolerance to allow the algorithm to partition along the higher-level module boundaries whenever possible, which helps in minimizing the number of modules in the partition. Fault behavior is modeled only in the simulator primitives so each partition must contain a portion of the circuit which is

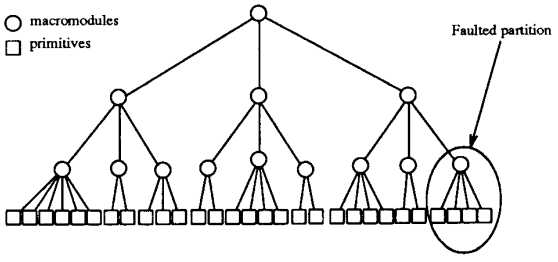

Figure 2. Hierarchical Fault Partitioning

represented at the primitive level. The remaining part of the circuit is represented functionally (where possible) by a higher-level functional representation. A configuration is created by marking the nodes of a tree expansion of the circuit hierarchy. This tree data structure is also used to record the faults that have been simulated, the faults that are in the current partition, the faults that have been detected, and the faults that remain undetected. Each module in the circuit hierarchy also contains the sum of the number of faults in all of its lower-level constituent modules or in itself if the module is a primitive. This number is used in the partitioning algorithm to decide if a module is an acceptable object for partitioning or if the module contains too many faults and therefore must be broken into its constituent lower-level modules.

The partitioning algorithm starts at the highest level definition in the circuit hierarchy and searches the tree data structure from left to right to find modules which have not been incorporated into a partition, (see Figure 2). The number of faults in the first available module is checked against the recommended partition size. If it is within the tolerances, the module, all its descendents, and all its direct ancestors are marked as *fault enabled* in the current partitioning. All siblings of the module and all siblings of the direct ancestors of the module may be simulated with functional models if they exist. These modules are checked for the presence of a functional model and if present, the module is marked for functional evaluation. If a functional model does not exist then the descendents of the module are checked. This procedure continues until a complete circuit representation consisting of fault-enabled and functional modules is created. If the number of faults in the module under consideration is too large, the partitioning algorithm descends into the descendents of the module and continues. If the number of faults was too small, the module is marked as fault-enabled and the next sibling is considered for incorporation into the current partition. Subsequent siblings are incorporated until the minimum number of faults have been enabled for the current partition. Once the simulation for a partition has finished all the fault-enabled modules are marked as previously simulated to prevent them from being incorporated in another partition. The simulation is complete when all the modules have been marked as previously simulated.

III. Distributed Hierarchical Fault Simulation

Distributed processing can be defined as several loosely coupled processors working on the same problem to achieve some goal. It is different from tightly coupled parallel processing, where communication time between processors is comparable to local memory access time. In loosely coupled systems, the communication costs are relatively high. These systems are preferable if computation granularity is large and task interactions are relatively infrequent. Though they offer less efficiency than tightly coupled systems or simulation engines, loosely coupled systems have the advantage of being less expensive and more versatile. The distributed fault simulator, DCHIEFS (Distributed Concurrent HIerarchical and Extensible Fault Simulator), was developed on a loosely coupled system.

Several parameters influence the performance of a distributed system [20], some of which are:

(1) The amount of parallelism inherent in the application problem.

(2) The decomposition of a problem into subproblems.

(3) The allocation of these subproblems to processors.

(4) The communication delay of the interconnection structure relative to the speed of the processors.

The first parameter is application-specific. The next two depend on the implementation and the last one depends on the particular multiple-processor system used.

All these parameters will be examined for the fault simulation problem. The precedence relation of a problem determines the inherent sequentiality that must be observed in that problem [21]. Because only single faults are assumed to occur in a faulty circuit, there is no precedence relation in the simulation of different faults. Thus high parallelism can be achieved in the fault simulation problem. The method for decomposing the problem into smaller subproblems is based on hierarchical fault partitioning. Each faulted partition is considered as a subproblem and allocated a processor. The proper grain size of a subproblem (faulted partition) is very difficult to determine. Some general guidelines have been followed. The partition size must be small enough so that each of the available processors gets at least one partition, yet large enough so that the evaluation time of the subproblem is not negligible compared to the communication time. The allocation of the subproblems to processors can be done in any order. The interconnection network employed is a bus, which is slow relative to the processor speed and limits the maximum number of processors that can be utilized efficiently. However, it was adequate for the maximum number of processors considered in this paper.

The following conditions must be satisfied for a distributed program to be efficient:

(1) $T \simeq T_s / p$

where T is the turnaround time for p processors working on the problem, T_s is the turnaround time for a single processor implementation and p is the number of processors in the multiple-processor implementation. It should be noted that T also includes communication time. This implies that the overhead due to distributing the algorithm should be low to satisfy the condition of linear speedup.

(2) $T_i \simeq T_j$

where T_i and T_j are turnaround times of processors i and j, respectively. This condition implies a homogeneous distribution and hence a balanced load on all the processors.

An alternative way to partition the problem of fault simulation of a given circuit could be at the circuit topology level, where different processors have different portions of the circuit. However, in that case some processors will require results of computation from other processors, which would lead to large amounts of communication between the processors. In the environment under consideration, communication is the bottleneck, so minimizing communication is necessary. Hence partitioning at the circuit topology is not an efficient method. The advantage of using faulted partitions is the independence of the subproblems from each other resulting in minimized communication. Figure 2 shows the general picture of hierarchical fault partitioning. Each host processor, also referred to as a server, obtains a copy of the circuit and injects faults in a unique portion of the circuit.

A. Architecture of the distributed system

The architecture of our loosely coupled system is shown in Figure 3. The client (master) and servers are on a communication bus using the TCP protocol. The system employs the Network File System (NFS) which enables the sharing of files in a heterogeneous environment of machines, operating systems and networks [22]. The Remote Procedure Call (RPC) facility of NFS supports a mechanism where a client machine can request a remote server machine to execute a procedure and return the results to the client. These two facilities were used to implement the communications between the client and servers.

B. Control structure

To keep the performance close to the optimal level, an efficient control paradigm should be selected. The control structures may be synchronous or asynchronous. In the synchronous case, all the servers start and finish a parallel portion of the algorithm at the same time; whereas in the asynchronous case, each server may be in different phases of the algorithm simultaneously. In a synchronous control structure if imperfect load balancing (varying partition sizes in the case of DCHIEFS) occurs, then the servers that get smaller loads eventually become idle. In addition, if the client has to determine which servers are ready to process more partitions, it has to serially check the status of the servers (polling). Both of these procedures cause a performance penalty. To avoid these penalties, we chose an asynchronous control structure. With this control structure, the inherent difficulties associated with different subproblem sizes (imperfect load balancing) have a minimized effect on performance. In our algo-

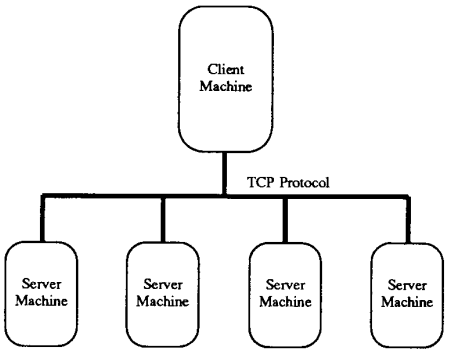

Figure 3. Architecture of the distributed system.

rithm, when a server finishes its assigned subproblem, it is assigned a new partition number, which has not already been assigned to any server. This happens irrespective of the status of simulation on other servers. To minimize polling, a *call-back* scheme is chosen, where each server process is responsible for initiating the request to the client for a new fault partition number.

C. Distributed Algorithm

Figure 4 shows the structure of the distributed process. The process proceeds as follows: the user specifies the number of servers, which the client initializes, by sending the circuit name and the user input options. The client determines if the requested number of servers is available. If so, the client initializes the desired number of servers; otherwise, the available servers are initialized. Based on the user-specified fault partition size, the servers concurrently divide the circuit into fault partitioned sections

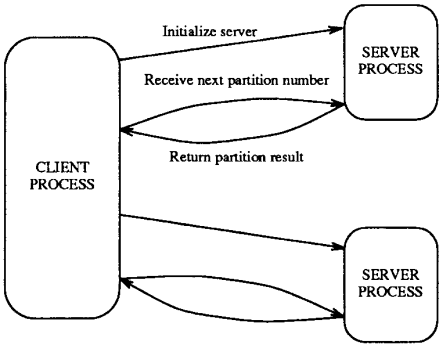

Figure 4. Servers communicating with the client through a call-back scheme.

using algorithm given in Section IIC. The algorithm is based on information about circuit hierarchy, which usually does not have a uniform or homogeneous structure. As a result, the actual sizes of fault partitions are not always equal to the specified size.

Each server gets a unique partition number from the client. Using the partition number, each server injects faults in the corresponding partition, as explained in section 2. The server simulates the faults contained in that partition. Once completed, the server sends the results to the client and requests a new partition number (call-back). The client saves the result and provides a new partition number for the server. The process continues until all the partitions of the circuit are fault simulated. Figure 5 gives an overview of the algorithm.

D. Implementation

The distributed fault simulator, DCHIEFS, has been implemented on a network of SUN workstations. The network is comprised of eight SUN 3/50's and one SUN 3/280 file-server connected by a 10 Mb/s ethernet. The file-server is also connected to three other file-servers, each with its own set of workstations. Although workstations connected to the other file-servers were not included in the experiment, their utilization is planned for future studies.

DISTRIBUTED ALGORITHM

Client process determines the number of server processes available

Client process initializes the server processes

While there are simulations to perform {

A server process sends results to the client process and requests a new partition number

The client process saves the results and provides a new partition number for the server process

The server process repartitions the circuit for the partition number it received

The server process simulates for the faults contained in that partition
}

Figure 5. The distributed algorithm.

The implementation required a server process to be running in the background on each of the server machines. A distributed simulation is initiated by starting a client process on the client machine (refer to Figures 3 and 4), then the distributed process proceeds until simulations of all the partitions are complete. It should be noted that there is no reason why the client machine can not also have a server process running on it. Either the file-server or a workstation can be chosen as the client. In our environment, however, the file-server (SUN 3/280) is 3 to 4 times faster than any of the workstations (SUN 3/50s). To simplify the analysis, the file-server was not used as a client or a server.

IV. Results

Several circuits of varying gate counts and total fault counts were simulated on DCHIEFS. Two different parameters, the size of the fault partitions and the number of servers used, were varied to study the performance of the distributed simulator. Since the communications between client processor and server processors were found to occur infrequently, the client processor (workstation) was also used to run a server process. It is well known that the runtime of any program depends on the system load. To eliminate this load dependency, no other user processes were allowed to run either on the fileserver or on the workstations during the course of the experiment. This *no-load* condition ensured repeatability of the results, and was confirmed by repeating some of the experiments.

Figures 6, 7, and 8 show the speedups for different number of processors and various partition sizes for three different circuits. The first circuit, MULT44 is a 4X4 multiplier, which has 3 levels of hierarchy (which is the number of levels in the circuit hierarchy tree, as shown in Figure 2) and 416 possible single line stuck-at faults. Simulation results of

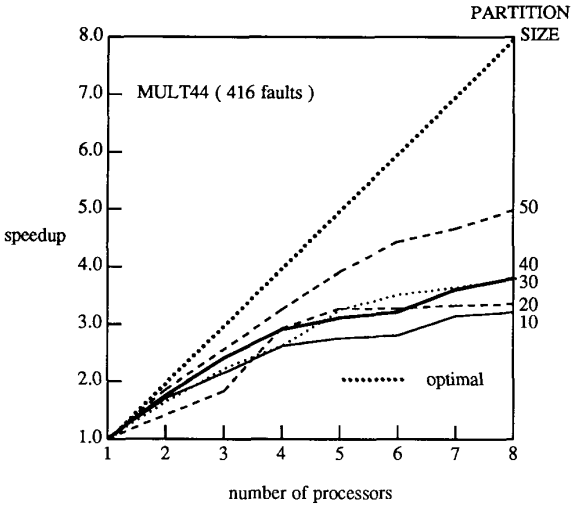

Figure 6. Speedup versus the number of processors for varying partition sizes.

MULT44 is included in Figure 6. Figure 7 shows the results for a control logic module, which contains many sequential elements with 3 level of hierarchy and 664 possible single line stuck-at faults. The last circuit simulated, FASTMULT, is a 24X24 bit mantissa multiplier with 25530 faults and 5 levels of hierarchy. Figure 8 shows the simulation results for FASTMULT.

Almost all the speedup curves are monotonically increasing with respect to the number of processors, which ensures that the use of more processors result in lower run-time in most of the cases, but the rates of increase in the curves do not follow a regular pattern with respect to either the number of processors or the partition sizes. Two factors, both of which are related to the irregularity in hierarchical structure of the circuit, are responsible for this anomalous behavior. First, the circuits do not have a uniform hierarchical structure, i.e., the sizes and the heights of the subtrees associated with different macromodules at the same level are not the same. This implies that the corresponding hierarchical tree structure of the circuit, similar to Figure 2, is not balanced. As a result of this, partitions having deeper leaves in the faulty part take longer time to be simulated. This causes an imbalance in the amount of work associated with each partition. Also, the nonfaulty parts of the circuit are different in different partitions. Some of the nonfaulty parts have a functional description and some do not. So, the partitions, even when containing identical number of faults, may take different times to be simulated, depending on the availability of a functional description of the nonfaulty part of the circuit. Secondly, the actual partition sizes may be widely varying. The spread in sizes may be as much as 75% of the user-specified partition size (50% on the higher side and 25% on the lower side). As a result, the total number of partitions π_t becomes more than π , where π is given by

$$\pi = \left\lfloor N / P_s \right\rfloor,$$

where N is the total number of faults and P_s is the partition size specified by the user. If π_t is not a perfect multiple of the number of processors, then some of the servers remain idle for some period of time when the simulation process is about to end. This penalizes the performance, as can be seen for larger number of processors for all the partition sizes in Figures 7(a) and 7(b).

The two factors mentioned above explain the anomalies in the speedup curves. A very uniform hierarchical structure of a circuit, both in terms of number of faults and the availability of functional descriptions of nonfaulty parts will ensure good uniformity in sizes of tasks associated with the subproblems. However, this is very hard to achieve for any general circuit, which will have some parts that are random as opposed to some that are regular. Also, the regular parts are usually of different sizes, so it is very hard to say which combination of partition size and number of processors will lead to the best performance. In the experiments reported in this paper, some situations occurred when the speedups were very close to optimal (partition size 50, 5 to 7 processors for the control logic in Figure 7(b) and partition size 500, 5 processors for FASTMULT in Figure 8). In these cases, near-perfect load balancing occurred. But, given a circuit, it is still unknown how to predict the best partition size.

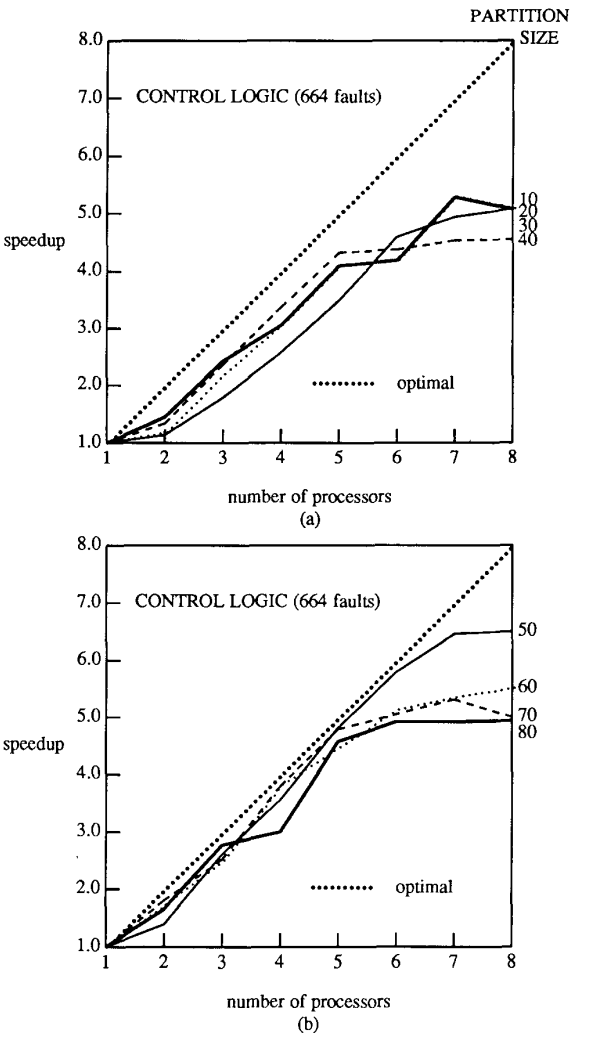

Figure 7. Speedup versus the number of processors for varying partition sizes.

The three example circuits presented here show very good speedups for the distributed implementation. Several experiments were performed in an attempt to find a formula for the optimal partition size for a given circuit, but it became evident that the optimal partition size depends heavily on the circuit topology, thus it can vary vastly even for circuits of similar size and functionality. However, even if the optimal partition size is not used, distributed fault simulation still shows very good speedup, allowing this technique to be used to simulate very large circuits that may currently require prohibitive simulation times on large mainframes.

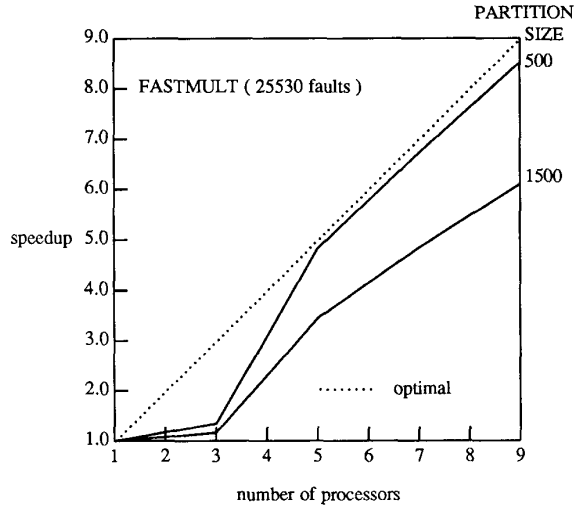

Figure 8. Speedup versus the number of processors for varying partition sizes.

V. Conclusion

The purpose behind this research was to find a low-cost method to speed up fault simulation by providing more general purpose processors in a network environment. The results show that distributed fault simulation is a promising alternative to the use of hardware accelerators. Most IC design houses have networks of engineering workstations and distributed fault simulation over that network can be an attractive solution to their fault simulation needs. Distributed fault simulation as a method is not restricted to CHIEFS and it can be applied to any hierarchical fault simulator which allows hierarchical fault partitioning. The speedups obtained for the circuits are very close to optimal in those situations where the partitioning size and the number of processors used are chosen correctly. Unfortunately, the optimal fault partition size depends heavily on the circuit topology and is difficult to predict. A rule of thumb is to use a partition size less than the ratio of total number of faults to the total number of servers. If a larger partition size is used, some server(s) will certainly remain idle, thus reducing efficiency. Even if the optimal partition size is not used, the speedup figures are still very attractive. So, this technique can be effectively used to fault simulate very large circuits. Future work will try to find a model which would predict the optimal or near optimal values of partition size and number of processors based on circuit topology.

Acknowledgement

This work was supported by the Semiconductor Research Corporation under Contract 87-DP-109. Special thanks to Ms. Carol Gura, Mr. Jeff Baxter and Mr. Ralph Kling, without whose help this work would not have been possible.

References

[1] O. H. Ibarra and S. K. Sahni, "Polynomially Complete Fault Detection Problems," *IEEE Transactions on Computers*, vol. C-24, pp. 242-249, March 1975.

[2] H. Fujiwara and S. Toida, "The Complexity of Fault Detection Problems for Combinational Logic Circuits," *IEEE Transactions on Computers*, vol. C-31, pp. 555-560, June 1982.

[3] Y. H. Levendel and P. R. Menon, "Fault-Simulation Methods - Extensions and Comparison," *Bell System Technical Journal*, vol. 60, pp. 2235-2259, November 1981.

[4] W. A. Rogers, J. F. Guzolek, and J. A. Abraham, "Concurrent Hierarchical Fault Simulation: A Performance Model and Two Optimizations," *IEEE Transactions on Computer-Aided Design*, vol. CAD-6, pp. 848-862, September 1987.

[5] P. Goel, "Test Generation Costs Analysis and Projections," *Proceedings of the 17th Annual Design Automation Conference*, pp. 77-84, 1980.

[6] T. W. Williams and K. P. Parker, "Design for Testability - A Survey," *Proceedings of the IEEE*, vol. 71, pp. 98-112, January 1983.

[7] W. A. Rogers and J. A. Abraham, "CHIEFS: A Concurrent Hierarchical and Extensible Fault Simulator," *Proceedings of the IEEE International Test Conference*, pp. 710-716, 1985.

[8] G. Pfister, "The Yorktown Simulation Engine : Introduction," *Proceedings of the Annual 19th Design Automation Conference*, pp. 51-54, 1982.

[9] T. Sasaki, N. Koike, K. Ohmori, and K. Tomita, "HAL : A Block Level Hardware Logic Simulator," *Proceedings of the Annual 20th Design Automation Conference*, pp. 150-156, 1983.

[10] T. Blank, "A Survey of Hardware Accelerators Used in Computer-Aided Design," *IEEE Design and Test*, pp. 21-39, August 1984.

[11] B. Milne, "Put the Pedal to the Metal with Simulation Accelerators," *Electronic Design*, pp. 39-52, September 1987.

[12] D. A. Reed, L. A. Adams, and M. L. Patrick, "Stencils and Problem Partitioning: their Influence on the Performance of Multiple Processor Systems," *IEEE Transactions on Computers*, vol. C-36, pp. 845-858, July 1987.

[13] W. W. Chu, D. Lee, and B. Iffla, "A Distributed Processing System for Naval Data Communication Networks," *Proc. AFIPS National Computer Conference*, vol. 47, pp. 783-793, 1978.

[14] W. W. Chu, M.-T. Lan, and J. Hellerstein, "Estimation of Intermodule Communication (IMC) and Its Applications in Distributed Processing Systems," *IEEE Transactions on Computers*, vol. C-33, pp. 691-699, August 1984.

[15] M. E. Lesk and E. Schmidt, "Lex - A Lexical Analyzer Generator," in *UNIX Programmer's Manual*. Murray Hill, New Jersey: Bell Laboratories, 1979.

[16] S. C. Johnson, "Yacc: Yet Another Compiler-Compiler," in *UNIX Programmer's Manual*. Murray Hill, New Jersey: Bell Laboratories, 1979.

[17] T. M. McWilliams, J. B. Rubin, L. C. Widdoes, and S. Correl, *SCALD II User's Manual*. Lawrence Livermore Laboratory, 1979 Annual Report, The S-1 Project.

[18] J. M. Acken and J. D. Stauffer, "Logic Circuit Simulation," *IEEE Circuits and Systems Magazine*, vol. 1, pp. 3-12, June 1979.

[19] *TEGAS Design Language User and Reference Manual*. Austin, Texas: Calma Company, 1984.

[20] Z. Cvetanovic, "The Effects of Problem Partitioning, Allocation, and Granularity on the Performance of Multiple-Processor Sysetms," *IEEE Transactions on Computers*, vol. C-36, pp. 421-432, April 1987.

[21] K. Hwang and F. A. Briggs, *Computer Architecture and Parrallel Processing*, 1984.

[22] Sun Microsystems Inc., in *Networking on the Sun Workstation*, Mountain View, CA, May 1985.