

Development of Parallel Distributed Computing System for ATPG Program

K. Han

Electrical and Computer Engineering
Auburn University

ABSTRACT

I present an approach about how to increase the speed-up ratio in parallel test generation based on a general system for distributed test generation. I implement a general purpose system for distributed computing using MPI. HITEC/PROOFS, an existing fault simulation and test generation package, is plugged into the system and together they form a distributed fault simulation and test generation framework. The experimental results on the ISCAS circuit demonstrate that this approach leads to considerable speed-up compared to running the package on a single computer. This approach is convenient to use since most of the work to achieve parallelization is transparent to the user.

1. INTRODUCTION

Generation of test vectors for the VLSI devices used in contemporary digital systems is becoming much more difficult as these devices increase in size and complexity. Automatic Test Pattern Generation (ATPG) techniques are commonly used to generate these tests. Since ATPG is an NP complete problem with complexity exponential to circuit size, the application of parallel processing techniques to accelerate the process of generating test vectors is an active research area.

In recent years, besides the research to increase the efficiency of the test generation system in a single processor [1], a lot of researches all over the world have been devoted to study the parallel test generation algorithms. Some approaches based on fault parallelism, heuristic parallelism, searching space parallelism, function parallelism, circuit parallelism, etc, has been proposed. Many prototype parallel systems have been implemented, such as the parallel system using grouping circuit faults according to primary input fan-out cones [2], the system using heuristic parallelism [3], the system using searching space parallelism to deal with hard-to-detect faults [4], and the system using the parallelism of faults [5], searching space [6], and circuit [7] for sequential circuits. All of these systems belong to small-scale parallelism, with 5~16 processors. Their common characteristic is to emphasize on dynamic load balance among processors. It is difficult to increase the speed-up ratio with dynamic load balance in MPP ($N \geq 128$). So a research paid much attention to static load balancing that would increase the speed-up ratio of parallel test generation and avoid the high communication overhead, and proposed an optimal parallel test algorithm based on static fault partitioning in MPP [8].

However, it is definitely more convenient and cost-effective to do distribute computing with existing general purpose software packages instead of specially designed parallel software. This project intends to transform general-purpose software to perform distributed computing. I present a distributed system implemented with MPI.

2. THE HITEC/PROOFS PACKAGE

HITEC/PROOFS [9] is a gate-level, sequential circuit fault simulation and test generation package from the University of Illinois. It runs on the UNIX platform, and it is not specifically designed for parallel computing. It targets single *stuck-at* faults in synchronous sequential circuits described in the ISCAS89 benchmark format.

The HITEC/PROOFS package consists of two main programs and four pre-processing programs. A thorough analysis of their inputs and outputs as well as the data flow is essential to constructing system.

The four pre-processing programs levelize the circuit (*level*), create a complete fault list (*faultlist*), collapse the fault list (*equiv*), and calculate the dominators in a circuit (*dominators*), respectively.

The two main programs are the test generator (*testgen*) and the fault simulator (*faultsim*). The fault simulator can be executed as a stand-alone program to fault grade a set of test vectors provided by the user, or in conjunction with the test generator to identify all faults detected by each test sequence generated by it. It also needs the levelized circuit description and the collapsed fault list (*circuit.eqf*) generated by the *equiv* program.

The test generator takes as inputs the levelized circuit description, collapsed fault list, and the output of the *dominators* program. The outputs are *circuit.grs* that contains runtime, efficiency and other statistics; *circuit.atp* that contains the result test set; and *circuit.red* that contains untestable faults.

3. MESSAGE PASSING WITH MPI

To build a distributed computing framework with general purpose software, we need to create an engine to facilitate message passing among processes as well as undertake processes management such as spawning new processes.

MPI [10] is simply a function that explicitly transmits data from one process to another. It is a powerful and very general method of expressing parallelism. It can be used to create extremely efficient parallel programs, and message passing is currently the most widely used method of programming many types of parallel computers.

I implement this project using the master/slave architecture. The master deals with task allocation and message management. The slaves deal with actual execution of the algorithm and finding the test vectors. The master keeps checking for messages being received from slaves. The communication primitives and process spawning are implemented with MPI.

4. SYSTEM ARCHITECTURE

4.1 Partitioning

In test generation, it is usually assumed that there is only one fault at a time. There is no interaction among different faults in the fault list. This single fault assumption makes it possible to partition the original test generation problem using the data parallel approach.

I partition the original fault list (*circiuf.fault*) and into pieces and distribute each sub-list to an instance of test generator, which then work on each partitioned fault lists separately and simultaneously. Other input files like the circuit description don't need to be partitioned, because they don't represent workload. The results from these processes can then be collected and combined to produce the final result.

For test generation, I combine the results by combining all *circuit.ref* files into one file, and all *circuit.atp* files into one file then eliminate all the duplicate entries from this file.

Load balancing is also an important issue in distributed computing. In this project static load balance is not used. I use a EDPFS (Equal Distance Partitioning of Fault Sequence) [8], which partitions the fault list into equal pieces, and assigns each piece to a processor whenever it is available. This simple strategy yields satisfactory results for this project. In the future I should experiment with more sophisticated load balancing strategies.

The correctness of the final result is guaranteed provided that we stitch together all the test sequences generated by multiple test generators. The combined test sequences will be able to detect the combined faults from all separate test sequences. Therefore the partitioning of fault list does not affect the correctness of the result.

The more processors are available, thus the smaller a sub-list is, and the less time it takes to generate test vectors for the smaller fault list. On the other hand, there is less chance the test generator can figure out that a test vector can test multiple faults.

4.2 Inter-process Communication

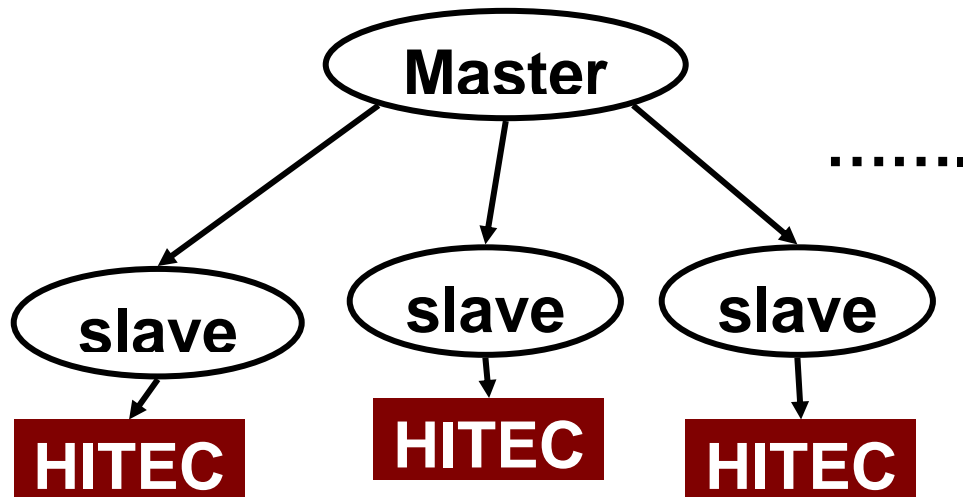
HITEC/PROOFS, like most fault simulation and test generation packages, do not have built-in support for message passing. Therefore we create slaves, with MPI and make them reside on each computer in our system. Each slave communicates with the HITEC/PROOFS process on its computer through file exchange, and communicates with other agents to accomplish the tasks of message passing and process management.

4.3 Implementation of system

The system is written in C, using MPI to implement its communication primitives. As shown in the following figure, the system consists of two programs, one is called the master and the other is the slave.

The master program gets the complete fault list, splits it into several small sub-lists, and then distributes those sub-lists to the slaves. The slave programs then start the test generator, get the results for those sub-lists, and send results back to the master program. Finally, the master program constructs the final results.

The master program executes only on one computer, and it starts the slave programs automatically when it is necessary. Each slave program runs on one computer in the system, and communicates with the HITEC program running on that computer.



The master program has three tasks. The first task is to initialize the master program itself, and to find out the number of computers available in the system. It then starts one slave program on each available computer and passes necessary information to those slave programs in order to run the test engine.

The second task is to partition the fault list. It gets the entire original fault list from a bench file, partition the fault list into several smaller sub-lists, then distributes each sub-list to an slave program.

The third task is to merge results from all agents. It collects all individual results from the slaves to construct a correct final result.

The slave program does the following: first, it receives arguments for spawning the test engine. These arguments include the fault list file. Then the slave program generates the feedback file (test vector file). These feedback files are generated by the test engine and need to be passed back to the master program to form the final result.

After that, the slave program receives the input files and a fault list from the master, and dumps that information to the local file system.

After receiving all necessary information from the master, the slave program starts the test engine. When the test engine finishes, the slave collects the result in the feedback files and sends it back to the master.

5. EXPERIMENTAL RESULTS

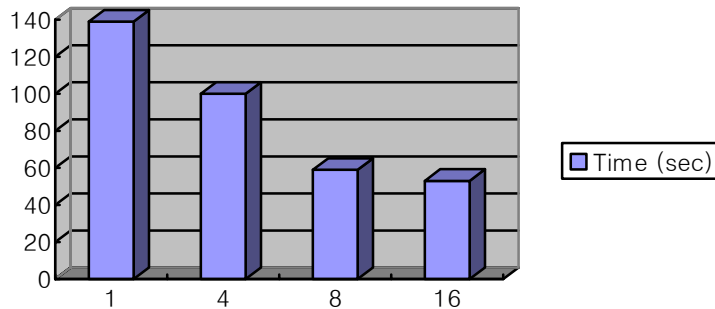
I run this system on a clustering network of one to 16 UNIX machine. The test case is the c7552 circuit which is available in an ISCAS85 format file.

Inputs: 382

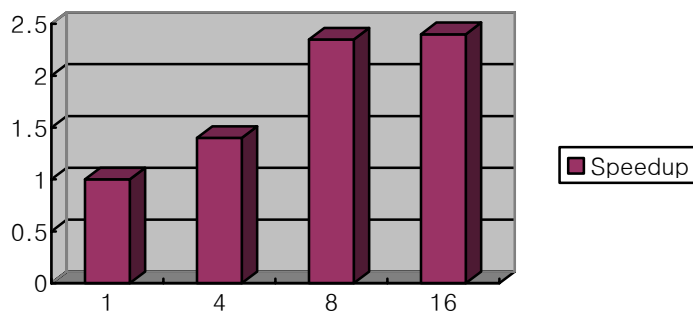
Gates: 876 inverters / 2636 gates (1310 ANDs + 1904 NANDs + 244 Ors + 54 NORs + 534 buffers)

Total signal lines: 11342

The run-time is measured in terms of elapse time (i.e. execution time or wall-clock time). The following chart shows the time spent by the system to run test generation with one to 16 machines, respectively.



The following chart shows the speed-up of the number of processors employed.



Load balancing is also an important factor. Evidently, the slowest machine determines the speedup bottleneck. With better load balancing strategy, I should be able to get better speed-up using the same set of machines. I can even try to dynamically rebalance the load by re-partitioning the fault list.

7. CONCLUSION

I designed and developed a system for distributed test generation. The system architecture is outlined, and some important issues such as partitioning are discussed. I also proposed potential improvements for the system.

I tested the system by doing test generation for the circuit. The experimental results are presented and analyzed, which clearly proves that this approach is promising.

REFERENCES

- [1] M. L. Bushnell, V.D. Agrawal. Essential of Electronic Testing. Kluwer Academic Publishers, 2000

- [2] S. Patil and P. Banerjee. Performance Trade-off in a Parallel Test Generation Fault Simulation Environment. *IEEE Trans. Computer-aided Design Integrated Circuits Systems*, 10(12): 1542-1558, Dec. 1991.
- [3] S. Chandra and J. H. Patel. Test Generation in a Parallel Processing Environment. *Proc. Int. Conf. Computer Design(ICCD-88)*, Oct. 1988: 11~14
- [4] S. Patil and P. Banerjee. A parallel Branch and Bound Approach to Test Generation. *IEEE Trans. Computer Design Integrated Circuits Systems*, 9(3), Mar. 1990: 313~322.
- [5] P. Agrawal, V. D. Agrawal, and J. Villodo. Sequential Circuit Test Generation on a Distributed System. *Proc. 30th Design Automation Conf. (DAC-93)*, June 1993.
- [6] B. Ramkumar and P. Banerjee. A Portable Parallel Algorithm for Test Generation. *Proc. Int. Conf. Computer-aided Design (ICCAD-92)*, Nov. 1992.
- [7] T. Chen. Distributed Automatic Test Pattern Generation. *Proc. Application Specific Integrated Circuits Conf. (ASIC-92)*, Sept. 1992.
- [8] Z. Zeng, J. Chen, and P. Liu. A Fault Partitioning Method in Parallel Test Generation. *Eighth Asian Test Symposium*. 1999.
- [9] Center for Reliable and High Performance Computing. University of Illinois. *HITEC/PROOFS User's Manual*.
- [10] P. S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers. 1997.