Dynamic Fault Grouping for PROOFS: A Win for Large Sequential Circuits

Charles R. Graham Elizabeth M. Rudnick Janak H. Patel

Center for Reliable & High-Performance Computing
University of Illinois, Urbana, IL

Abstract

This paper discusses the important role of fault grouping in a parallel 32-bit fault simulator such as PROOFS. Three algorithms are presented which dynamically order the fault list during fault simulation to determine how the faults get grouped together. The dynamic fault grouping algorithms were incorporated into PROOFS and tested on benchmark circuits. The algorithms showed a marked reduction in the number of faulty circuit gate evaluations (compared to a static fault grouping) for almost all of the circuits with more than 20 flip-flops. For the largest benchmark circuit, s35932, all of the algorithms showed at least a 39% reduction in the number of faulty circuit gate evaluations and at least a 55% speedup in simulation time.

I Introduction

This section will explain the importance of fault grouping in determining the fault simulation efficiency of a parallel 32-bit fault simulator. Then, some limitations of static fault grouping will be presented.

In order to take advantage of the parallelism inherent in PROOFS, the proper fault grouping is essential [1] [2]. Each fault injected into the circuit causes a certain number of circuit elements to be evaluated. Let the list of all circuit elements evaluated for a given fault and vector pair be called the *sphere of influence*. The goal of fault grouping is to group faults together whose spheres of influence have the greatest intersection.

When a group of 32 faults is injected into a circuit, all the gates in each of the 32 spheres of influence must be evaluated before a new group of faults can be injected into the circuit. In the case of ideal fault grouping (where all 32 faults grouped together cause exactly the same circuit elements to be simulated) the number of gate evaluations is reduced by a factor of 32 because each gate must be evaluated only once to simulate every fault in the fault group. On the other hand, if there is no intersection between spheres of influence, then each gate is evaluated to determine the value of one faulty circuit only. When this occurs, the major advantage of parallel fault simulation is lost. In a fault simulator that uses word level parallelism, such as PROOFS, the fault grouping that is chosen will determine the number of gate evaluations that must be performed and thus affect the fault simulation time.

PROOFS implements a static fault grouping technique based on a depth-first search of the circuit starting at the primary outputs. The ordering of the faults in the fault list is done in a preprocessing stage prior to fault simulation. Once PROOFS scans a fault list, the ordering of that fault list is not changed except to permanently drop faults that have been detected or to temporarily drop faults that are known to be inactive. Faults are taken from the fault list in sequential order for each test vector to be simulated [1].

A static fault ordering, as used in the original PROOFS, is a simple approach that attempts to group faults together that are most likely to have intersecting spheres of influence. One type of fault that this static fault ordering approach does not take into account is the hyperactive fault. A hyperactive fault is defined as a fault which is undetected by a vector and has a very large sphere of influence. Hyperactive faults pose a limitation on parallel fault simulation because they tend to dramatically increase the number of gates evaluated for their fault groups. Lee and Ha noted during the development of their PROOFS-based fault simulator called HOPE that when highly active faults were grouped together, the bits in each 32-bit word were better utilized [2][3]. In order to reduce the negative effects of hyperactive faults, they implemented dynamic fault grouping in addition to static fault grouping in their HOPE1.1 fault simulator [2]. Later Kung and Lin developed the HyHOPE fault simulator in order to further reduce the number of gate evaluations caused by hyperactive faults. The algorithm used in HyHOPE separates hyperactive faults from non-hyperactive faults and simulates them differently [4].

The goal of our research is to implement and test dynamic fault ordering algorithms that group normal faults separately from hyperactive faults. In order for a dynamic fault ordering algorithm to be successful, it must identify hyperactive faults in a simple way because using a complex algorithm would most likely defeat the purpose of reducing overall simulation time [2].

*This research was supported in part by DARPA under Contract DABT63-95-C-0069, in part by the Semiconductor Research Corporation under Contract SRC 95-DP-109, and by Hewlett-Packard under an equipment grant

II Proposed Solutions: Dynamic Fault Grouping Based on Fault Activity

This section explains the details of three different dynamic fault grouping algorithms which were developed to group hyperactive faults together. Each of the algorithms begins with the same preprocessed fault list used by PROOFS (depth-first from primary outputs). The fault list is re-ordered just before the simulation of a new vector. The new fault list is then used to inject groups of 32 faults into the circuit.

A Fault Grouping Based on Potential Detections

The first fault ordering algorithm separates the faults into two lists: (1) faults that were not potentially detected by the previous vector and (2) faults that were potentially detected by the previous vector. A potentially detected fault is defined as a fault which causes an unknown value, X, to be propagated to a primary output while the good circuit value is a known 1 or 0. As in the original PROOFS, faults which are known to be inactive are not inserted into either list. The potentially detected fault list is then appended to the end of the undetected fault list, and the resulting fault list is passed to the fault injection routine.

A fault that causes potential detections is likely to be very active because it has caused an X value to propagate from a flip-flop to a primary output. In this case there is a very high probability that the X value was also propagated to a flip-flop. This means that the fault may remain as a potentially detected fault in the next time frame [2].

B Fault Grouping Based on Fault Effects at Flip-Flops

The second algorithm groups faults based on the number of flip-flops that each fault causes to be erroneous in the circuit. If a fault effect is propagated to several flip-flops in a circuit but the fault is not detected, there is a high probability that the fault will be hyperactive in the next time frame. This algorithm separates the fault list into three groups using a threshold, t. Faults are placed into the groups as defined below and then joined into one list before being sent to the fault injection routine.

Group 1: # of flip-flops having fault effects = 0

Group 2: 0 < # of flip-flops having fault effects < t

Group 3: # of flip-flops having fault effects $\geq$ t

C Fault Grouping Based on Potential Detections and Fault Effects at Flip-Flops

The third algorithm combines the two previous algorithms. The fault list is separated into four groups. First, the potentially detected faults are removed from the original list; then the remaining faults are ordered into three groups as described in the previous section. These four groups are then concatenated before being sent to the fault injection routine.

III Results

The PROOFS fault simulator was modified to dynamically order the faults according to the three algorithms detailed previously. The simulation results were gathered and compared to the results of the original version of PROOFS. The metrics used for comparing the original PROOFS to the PROOFS with dynamic fault ordering were (1) the overall execution time and (2) the number of faulty circuit gate evaluations.

Results were gathered using the GATEST vectors [5], the ISCAS89 benchmark circuits, and several synthesized circuits. Table 1 contains simulation details for the circuits and vectors used, including the execution time and number of faulty circuit gate evaluations (Flt Eval). We are particularly interested in the simulation results of the large sequential circuits because these are the circuits that have the longest fault simulation times. Table 1 contains the simulation results for all the circuits containing more than 20 flip-flops. The circuits requiring more than 9 seconds of execution using the original PROOFS are of particular interest and have been placed at the bottom of the table for easy comparison. All simulations were run on an HP 9000 J200 workstation with 256 MB of RAM.

A Fault Grouping Based on Potential Detections

Table 1 shows the results of the circuit simulations when faults are grouped dynamically according to potential detection status (listed under POT). For the last six circuits listed in the table, the average reduction in the number of faulty circuit gate evaluations was 27.5%. The average speedup for the same circuits was 30%.

Figure 1 shows a graph of the number of faulty circuit gate evaluations as a function of the test vector number for the circuit s35932. The graph shows that dynamic fault grouping consistently outperforms static fault grouping at each stage of fault simulation.

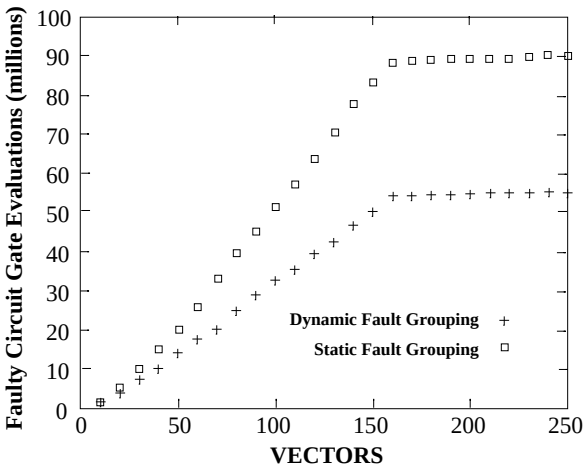

Figure 1: Dynamic vs. Static Fault Grouping: Comparison of Faulty Circuit Gate Evaluations for s35932

Table 1: Simulation Results for PROOFS with Dynamic Fault Grouping

Circuit	Gates	Flip-Flops	Faults	Vec	Fault Cov	Time (sec)	Speedup			Flt Eval (x 1000)	Gate Eval Red. Ratio		
							POT	FEFF	COMB		POT	FEFF	COMB
s382	158	21	399	331	0.87	0.75	1.02	1.03	1.05	85	0.96	0.96	0.96
s400	164	21	426	324	0.86	0.75	1.05	1.05	1.05	85	0.94	0.94	0.94
s444	181	21	474	254	0.86	0.75	1.07	1.07	1.07	89	0.93	0.95	0.93
s526	193	21	555	371	0.76	1.06	1.01	1.02	1.03	125	0.98	0.99	0.98
s3330	1789	132	2870	863	0.74	7.09	1.00	0.90	0.90	835	1.04	1.40	1.39
s3384	1702	183	3380	675	0.93	7.53	0.98	0.91	0.93	1005	1.12	1.18	1.10
s4863	2342	104	4764	502	0.96	4.75	1.06	1.07	1.10	445	0.88	0.89	0.87
s6669	3123	239	6684	542	1.00	7.91	1.05	1.06	1.00	730	0.94	0.95	0.93
div16	856	50	2147	725	0.80	4.11	0.99	0.95	0.94	667	1.01	1.04	1.01
mult16	645	55	1708	204	0.22	3.61	1.11	1.12	1.11	655	0.86	0.86	0.85
am2910	930	87	2391	745	0.90	5.01	1.02	1.02	1.03	790	0.96	0.95	0.95
s1423	657	74	1515	699	0.85	9.51	1.28	1.29	1.28	1688	0.67	0.66	0.67
s3271	1573	116	3270	1305	1.00	9.38	1.04	1.01	1.01	1049	1.00	1.00	1.00
s5378	2779	179	4603	586	0.69	14.7	1.17	1.22	1.18	2543	0.84	0.81	0.85
s35932	16065	1728	39094	256	0.89	1294	1.55	1.56	1.56	90843	0.60	0.61	0.60
pcont2	4010	24	11272	272	0.59	17.2	1.57	1.58	1.61	3205	0.50	0.49	0.49
piir8	10712	56	29689	531	0.57	57.4	1.21	1.22	1.22	8760	0.74	0.72	0.70

B Fault Grouping Based on Fault Effects at Flip-Flops

Results in Table 1 for fault grouping based on fault effects propagated to flip-flops (listed under FEFF) show that this algorithm provides a slight improvement over the previous algorithm for the six circuits with the largest number of faulty events using the original PROOFS. The average speedup in simulation time for these circuits was 31%, and the average reduction in faulty circuit gate evaluations was 28.5%. The results in Table 1 were measured using a threshold value of 5. Several different threshold values were simulated without any appreciable performance increase.

In addition, a similar algorithm was tested which divided the fault list into two groups: (1) all the faults causing fewer fault effects than threshold t, and (2) all of the remaining faults. This algorithm was also tested with several different threshold values. The algorithm and results were similar enough to the above mentioned algorithm that more details have not been included.

C Fault Grouping Based on Potential Detections and Fault Effects at Flip-Flops

The results for the combined algorithm are shown in Table 1 (listed under COMB). The average speedup in simulation time for the last six circuits was 31%, and the average reduction in faulty circuit gate evaluations was 28%. Several different threshold values were simulated without any appreciable performance increase.

IV Conclusion

The research presented in this paper clearly shows that it is a design win to use some form of dynamic fault grouping in a parallel fault simulator such as PROOFS. Although the idea of dynamic fault grouping for such simulators is not new [2] [3] [4], several of the dynamic fault ordering algorithms presented in this paper are unique.

Three algorithms for dynamically ordering faults were described. The goal of each of these algorithms was to explore methods for improving the fault grouping in PROOFS by simulating the highly active faults together instead of letting them remain interspersed among the less active faults.

All of the algorithms improved the performance of the original PROOFS when used to simulate six of the larger circuits studied. Especially notable is the fact that for the two largest ISCAS89 benchmark circuits, s5378 and s35932, dynamic fault ordering based on fault effects at the flip-flops resulted in reductions in the number of faulty circuit gate evaluations by 19% and 39%, respectively, and speedups of 21% and 56%.

References

[1] T. M. Niermann, W.-T. Cheng, and J. H. Patel, "PROOFS: A Fast, Memory-Efficient Sequential Circuit Fault Simulator," *Proc. ACM/IEEE Design Automation Conf.*, pp. 535-540, June 1990.

[2] H. K. Lee and D. S. Ha, "New Methods of Improving Parallel Fault Simulation in Synchronous Sequential Circuits," *Proc. Int. Conf. Computer-Aided Design*, pp. 10-17, June 1993.

[3] H. K. Lee and D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *Proc. ACM/IEEE Design Automation Conf.*, pp. 336-340, June 1992.

[4] C.-P. Kung and C.-S. Lin, "HyHOPE: A Fast Fault Simulator with Efficient Simulation of Hypertrophic Faults," *Proc. Int. Conf. Computer-Aided Design*, pp. 714-718, 1994.

[5] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "Sequential Circuit Test Generation in a Genetic Algorithm Framework," *Proc. ACM/IEEE Design Automation Conf.*, pp. 698–704, June 1994.