Parallel Simulation Algorithm of VLSI for Multicore Workstations with Dynamic Faults Grouping

D. E. Ivanov

***Abstract*—A new algorithm for parallel fault simulation of VLSI on multicore workstations with common memory is proposed. To speeding up the algorithm two-level parallelization is used. First, main schema of the algorithm is based on the concurrent many-threaded simulation of the groups of faults for each input vector. Second, each group of faults is simulated in bit-parallel way. The dynamic fault grouping is adopted. The results of computational experiments on ISCAS-89 benchmarks circuits are reported, which are obtained on the 12-core workstation.**

***Index Terms*—Fault simulation, multicore workstation, multithreaded programming, parallel calculation, VLSI circuit.**

I. Introduction

Fault simulation algorithms of VLSI are important in the design of digital circuits. The results of these algorithms are lists of tested/untested faults or the fraction of detected faults to the total number, named as fault coverage. The disadvantage of these algorithms is that for large circuit the simulation process may continue for a very long time due the large list of faults. So, today is the actual task of constructing of fast fault simulation algorithms of VLSI.

One way to speed-up the fault simulation algorithm is use of parallel calculation. To date, there are three main schemes for parallelizing of fault simulation of sequential circuits.

1) *Fault list partitioning* [1], [2]. In this approach, a complete list of faults F is partitioned into several sub lists $F_1, F_2, \ldots, F_n$, each of which is transferred to a separate processor in system, where fault simulation on a given test sequence is performed. In this scheme each processor must have its own copy of the circuit description and test sequence. Currently, this method is used most widely and considered that he has a good scalability with increasing the number of processors.

2) *Circuit partitioning* [3]. In this approach, the circuit is partitioned into several sub circuits, each of which is simulated on a separate processor. The main advantage of this approach is that it significantly reduces the memory requirements, which stores the description of the circuit. The main disadvantage is the necessity of developing of processors interaction protocols.

3) *Test sequence partitioning* [4] is that the input sequence is divided into a number of sequences, which are formed subtasks for computer nodes. Processes also need to communicate to inform each other about what faults detected to the current time.

These schemes are the base to construct new fault simulation algorithms. Methods with fault list partitioning also are divided into synchronous and asynchronous, which try to improve processor utilization by reducing their idle, which occurs during synchronization [5]. Conceptually they are often based on the synchronous methods. Therefore, we will talk about synchronous methods with fault set partitioning. It should be noted the work [6], in which proposed one of the basic methods of parallel fault simulation. It is used both static and dynamic fault grouping. The authors also suggested algorithms of this kind for cluster systems [7] and for multi-core workstations with shared memory [8].

One of the trends is the development of such methods for multicore systems with shared-memory [9]. Obviously, the performance of the same method to the different parallel systems varies greatly. Our experiments in [8] were shown that blind transfer of methods with the fault list partition to multicore systems does not give expected growth of productivity. In this paper we propose a new parallel algorithm for fault simulation of VLSI circuits, which works on such systems. In contrast to previous methods, it does not use the static partitioning of faults. Our algorithm utilizes new scheme in which the parallel simulation of dynamically grouped faults for each input test vector is performed. In addition, it uses bit-parallel simulation for those faults, which are simulated on the same processor.

II. Bit-Parallel Fault Simulation Algorithm for Single-Processor System

In this section it is briefly described a fault simulation algorithm of digital circuits, which is the basis for the parallel version. The algorithm is ideologically based on the PROOFS algorithm, and its details are given in [10]. The main advantage of this algorithm is the bit-parallel fault simulation. The algorithm works with synchronous sequential circuits, in which the combinational part and elements of the states are marked out. Simulation of behavior of the circuit is performed by iterative simulation of its combinational part. Algorithm deals with single stuck-at-0 and stuck-at-1 faults. Below is considered the pseudo code of the basic algorithm, since it is essential to understanding the principle of building of parallel version.

```
FaultSimulation1(Circuit, FaultList, Test, Length){
    for( int i=0 ; i<Length ; i++ ){
        Vector=Test[i];
        SVI=SimulateGoodCircuit(Circuit,Vector);
        while( there are unconsidered faults in FaultList ){
```

Manuscript received October 5, 2015; revised May 2, 2016.

D. E. Ivanov is with Institute of Mathematics of National Academy of Sciences of Ukraine, Tereschenkivska st., 3, Ukraine (e-mail: dmitry.ivanov.iamm@gmail.com).

DOI: 10.7763/IJMO.2016.V6.522

```
Group=FormGroupOfFaults(FaultList);
SimulateGroupOfFaults(Circuit,Group,Vector);
CheckFaultDetectionInGroup(SVI,Group);
SaveStatesForUndetectedFaults(SVI,Group);
} // end loop on faults
} // end loop on test vectors
} // end of simulation
```

Here: *Circuit* – net list of circuit description; *Fault List* – list of faults, *Test* – test sequence; *Length* – its length.

The main loop of simulation is performed by the input test vectors and includes the following steps. First, the simulation of fault-free circuit is executed. Its result is the values of signals on all lines of the circuits, which are stored in the array SVI. This values then used in three next cases: a) to determine the activity of faults: inactive fault is not simulated for current input vector; b) to determine the testability of faults after their simulation; c) to determine what elements of the state of faulty circuits should be saved for the simulation in the next time frame. The internal loop is performed while in the list there are faults, which were not considered for the current test vector. First, it is formed the group of faults which will be simulated in parallel in different bits of the machine word. Each group contains the number of faults which corresponds to the number of bits in the computer word. This follows by the fault detectability analysis. Detected faults are deleted from the list. For all undetected faults the values of states are saved to perform the simulation in the next time frame. Only signal values that differ from those in the fault-free circuit are stored. Then both cycles are continued.

III. Parallel by Groups of Faults Simulation Algorithm for Multiprocessor System with Common Memory

In this section is described the proposed algorithm, which is the multi-threaded version of the algorithm described in the previous section.

Parallelization of the greatest possible fragment of code should lead to the better scalability by reducing the proportion of sequential code. Therefore, is better to start from the top cycle. However, in this case it is not possible: the simulation on the next input vector *t+1* cannot start when is not finished the simulation on the current test vector *t*, because the signal values in the circuit in the time of transition from vector *t* to vector *t+1* are not defined.

The internal loop by the faults includes several functions. The greatest computational load falls on the function *Simulate Group Of Faults()*. Simulation of some group of faults on the current input vector does not affect on the simulation of another group on the same vector. Therefore, simulation of the one group is an independent branch of the program and several of these groups can be simulated in parallel. To organize the parallel execution, the function *Simulate Group Of Faults()* should be implemented as a thread class. Because such threads are independent and do not interact, there is no need to synchronize their execution. It is clear that the procedures *Check Fault Detection In Group()* and *Save States Fo rUndetected Faults()* can be easily included in our thread. Otherwise, the function *Form Group Of Faults()* is a preparatory. It must be executed before the parallel threads start and therefore cannot be included into this thread. But this should not greatly affect the performance, because the function performs the minimal computational steps with the pointers on the faults in the group. Pseudo code of this implementation in terms of threads is shown below.

```
FaultSimulation2(Circuit, FaultList, Test, Length, hreadsNumber){
CreateSimulationThreads();
for( int i=0 ; i<Length i++ ){
Vector=Test[i];
SVI=SimulateGoodCircuit(Circuit,Vector);
while( there are unconsidered faults in FaultList ){
for( int Number=0;Number<ThreadsNumber;Number++ ){
Group=FormGroupOfFaults(FaultList);
ThreadsArray[Number]->Data=(Vector,SVI,Group);
ThreadsArray[Number]->Resume();
}
for( int Number=0;Number<ThreadsNumber;Number++ ){
ThreadsArray[Number]->WaitFor();
}
} // end loop on faults
} // end loop on test vectors
DeleteSimulationThreads();
} // end of simulation
```

Here: *Threads Number* indicates the number of simultaneously running simulation threads. Its value must be determined experimentally. *Threads Array* contains the corresponding number of pointers to the object of our thread class. Simulation thread includes the code of functions *Simulate Group Of Faults()*, *Check Fault Detection In Group()* and *Save States For Undetected Faults()* from the algorithm *FaultSimulation1*. Function *Wait For()* waits the suspension of corresponding thread.

The pseudo-code of the algorithm *FaultSimulation2* shows that the simulation is performed in parallel for groups of faulty circuits. Execution of the simulation for one input vector *t* is illustrated on Fig. 1.

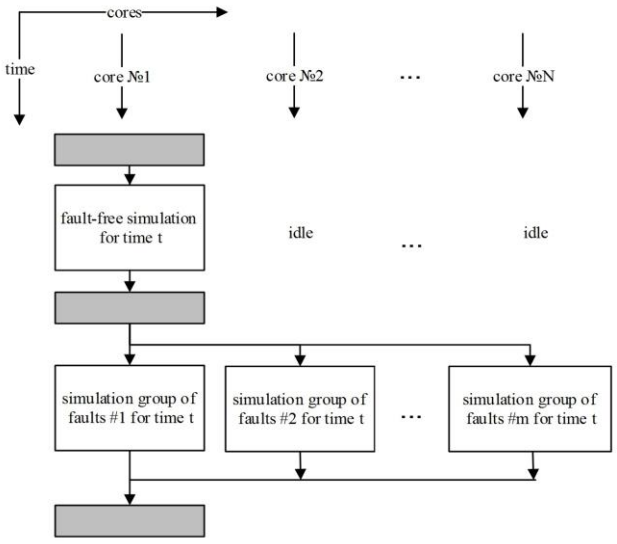

Fig. 1. Simulation of fault-free VLSI and further parallel simulation of groups of faults.

Section of parallel simulation must be repeated required

number of times because the number of groups of faults may not match to the number of cores in the system. Also, before the parallel section there is a big part of sequential code, which simulates the fault-free circuit. It is also desirable to perform its parallel simulation with other threads. However, the thread with simulation of fault-free circuit cannot be created and run directly, because it calculated the signal values in the circuit, which required for subsequent procedures of fault simulation. However, parallelization in this case is still possible. To do this, it is necessary to create an additional thread that will contain the code of function *Simulate Good Circuit()*. But on the step *t* fault-free circuit is simulated for the input vector *t+1*, which results are needed for simulation of faulty circuits in the next time frame. This will require an additional simulation of fault-free circuit for the first time frame, which must be done before the opening cycle by the input vectors. Also, it is necessary to create two copies of the array of signal values in the fault-free circuit: *SVI* - for fault simulation at current time frame, *SVINEXT* - for pre-calculated values for the next time frame. After the end of the simulation of the fault-free circuit for the next time frame it is necessary to rewrite the value of *SVINEXT* in *SVI*.

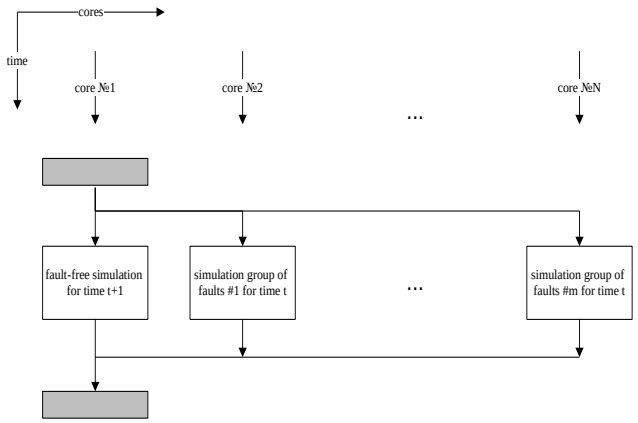

Fig. 2. Parallel simulation of fault-freeVLSI for time t+1 and groups of faults for time t.

Pseudo code of such implementation is shown below, and schematic representation of the threads execution for time frame *t* is illustrated on Fig. 2.

```
FaultSimulation3(Circuit, FaultList, Test, Length, hreadsNumber){
CreateSimulationThreads();
SimulateGoodCircuit->Data(Test[0]);
SimulateGoodCircuit->Resume();
SVI=SimulateGoodCircuit->WaitFor();
for( int i=0 ; i<Length i++ ){
Vector=Test[i];
NextVector=Test[i+1];
SimulateGoodCircuit->Data(NextVector);
SimulateGoodCircuit->Resume();
while( there are unconsidered faults in FaultList ){
for( int Number=0;Number<ThreadsNumber;Number++ ){
Group=FormGroupOfFaults(FaultList);
ThreadsArray[Number]->Data=(Vector,SVI,Group);
ThreadsArray[Number]->Resume();
}
for( int Number=0;Number<ThreadsNumber;Number++ )
ThreadsArray[Number]->WaitFor();
} // end loop on faults
SVINEXT=SimulateGoodCircuit->WaitFor();
SVI=SVINEXT;
} // end loop on test vectors
DeleteSimulationThreads();
} // end of simulation
```

In such implementation the number of concurrent threads is *ThreadsNumber+1*.

Proposed scheme of the construction of fault simulation algorithm does not refer to any of the schemes mentioned in introduction. Most closely it coincides with the fault list partitioning scheme. However, in this scheme there is no pre-partitioning of fault list and no binding of sub-lists with the nodes of system. Instead, faults' grouping for different nodes occurs dynamically for each input vector in the test sequence. Also additional simulation thread with fault-free simulation for next time frame is performed.

IV. The Experimental Results

Experiments were performed only with five big benchmarks from the catalog ISCAS-89 on the multicore system with the following characteristics: two Intel(R) Xeon CPU X5650 with frequency 2.67Ghz (each of 6 cores); the technology HyperThreading is off; the RAM 16 GB; operating system MS Windows Server 2008 R2. Two series of experiments were carried out, which must answer the following questions. 1) It is known that the memory subsystem in parallel computing systems is bottleneck. Is it necessary to use separate or common circuit description tables for each simulation thread? 2) How well the proposed algorithm is scalable?

Series 1. Two versions of the algorithm were implemented. In the first modification one copy of tables with circuit description is used. In the second modification a number of copies of the tables were created equal to the threads number. The experiments show that no matter which kind of circuit's description was used: common or separate tables. Therefore, further development of the algorithm was carried out for the modification with common tables.

Series 2. Is studying the scalability of the parallel fault simulation algorithm of digital circuits *FaultSimulation3* depending on the threads number. It was varied from 1 to 18. On practice the implementation of the basic scheme is almost always supplemented by heuristics that significantly improve the performance (often up to 2 times). In our study, we investigated the scalability only of the new basic scheme of the algorithm without the use of heuristics.

To accurately determine the extremum in the graph of acceleration, maximum number of threads was chosen greater than the cores number in the system. Graphs of the acceleration depending on the number of concurrent processing threads on average, in the best and worst cases are shown in Fig. 3. Analysis of the numerical values shows that the greatest acceleration for all circuits is achieved when running in parallel 11 threads of simulation of fault groups. In this case the actual number of parallel threads is 12, because 1 fault-free simulation thread must be added. As expected, this count corresponds to the number of cores in the system.

TABLE I: CHARACTERISTICS OF THE PARALLEL VERSION OF THE ALGORITHM

Number of threads	Circuit														
	s9234			*s35932*			*s38417*			*s38584*			*s38584_1*		
	«1»	«2»	«3»	«1»	«2»	«3»	«1»	«2»	«3»	«1»	«2»	«3»	«1»	«2»	«3»
1	**98**	0.08	1	**619**	0.08	1	**2441**	0.08	1	**2792**	0.08	1	**1512**	0.08	1
2	1.78	0.15	0.52	1.96	0.16	0.47	2.04	0.17	0.44	1.92	0.16	0.48	1.38	0.11	0.70
3	2.51	0.21	0.34	2.88	0.24	0.29	2.90	0.24	0.29	2.79	0.23	0.30	2.00	0.17	0.45
4	3.16	0.26	0.25	3.73	0.31	0.20	3.86	0.32	0.19	3.66	0.31	0.21	2.62	0.22	0.32
5	3.38	0.28	0.23	4.30	0.36	0.16	4.40	0.37	0.16	4.20	0.35	0.17	3.05	0.25	0.27
6	3.06	0.26	0.27	3.92	0.33	0.19	3.89	0.32	0.19	3.81	0.32	0.20	2.70	0.23	0.31
7	3.38	0.28	0.24	4.30	0.36	0.16	4.50	0.38	0.15	4.28	0.36	0.16	3.12	0.26	0.26
8	3.77	0.31	0.20	4.84	0.40	0.13	4.83	0.40	0.13	4.84	0.40	0.13	3.60	0.30	0.21
9	4.08	0.34	0.18	5.29	0.44	0.12	5.35	0.45	0.11	5.25	0.44	0.12	3.99	0.33	0.18
10	4.26	0.36	0.17	5.68	0.47	0.10	5.77	0.48	0.10	5.67	0.47	0.10	4.26	0.35	0.17
11	4.67	0.39	0.14	6.19	0.52	0.08	6.72	0.56	0.07	6.24	0.52	0.08	4.61	0.38	0.15
12	4.45	0.37	0.15	5.07	0.42	0.12	5.42	0.45	0.11	4.99	0.41	0.13	3.34	0.28	0.24
13	3.92	0.33	0.19	4.95	0.41	0.13	5.31	0.44	0.11	5.09	0.42	0.12	3.54	0.30	0.22
14	3.92	0.33	0.19	4.73	0.39	0.14	5.55	0.46	0.11	5.19	0.43	0.12	3.61	0.30	0.21
15	3.92	0.33	0.19	4.45	0.37	0.15	5.05	0.42	0.12	5.09	0.42	0.16	3.48	0.29	0.22
16	3.06	0.26	0.27	4.33	0.36	0.16	4.69	0.42	0.14	4.40	0.37	0.16	3.38	0.28	0.23
17	2.97	0.25	0.28	4.27	0.36	0.16	4.71	0.39	0.14	4.38	0.36	0.16	3.36	0.28	0.23
18	2.80	0.23	0.30	4.21	0.35	0.17	4.66	0.39	0.14	4.42	0.37	0.16	3.35	0.28	0.23

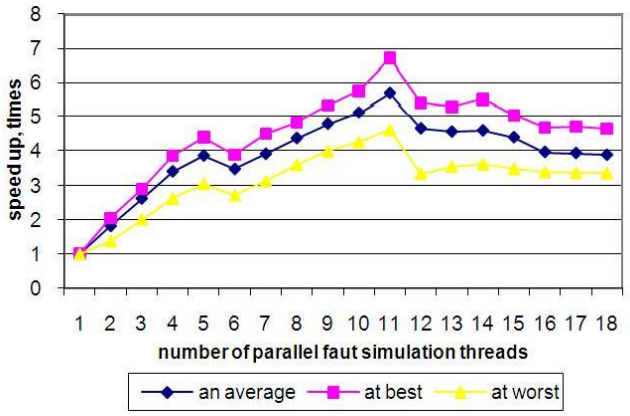

Fig. 3. The speed-up of the algorithm depending on the number of parallel fault simulation threads.

Based on the simulation time the next characteristics of parallelization are calculated: acceleration, utilization of cores and proportion of sequential code [11]. They are reported in Table I. Here, for each circuit in column «1» it is shown the speed up of the algorithm (excepting row 1, which shows the time in seconds for application with one fault simulation thread, this values are based for further calculation, bolded in table), «2» - the efficiency of use of cores, «3» - the proportion of serial code. One comment should be made for one-threaded application. From Fig.2 we can see that if we're talking about the single-threaded application, actually we have two calculation threads: one simulation thread for fault-free circuit and one simulation thread for group of faults. Therefore, the above numeric data are deteriorated. In comparison with the pure single-threaded application, numerical results should be much better. However, we did not produce a single-threaded implementation to comparison with it, because in this case it no longer corresponds to our new algorithm scheme. Since the experiments show that the number of threads equal to the number of processor cores, we can say that the comparison is made not with a single-core, but with a dual-core system.

Note that for all circuits achieved a relatively high acceleration: in the worst case it is equal to 4.61 times. The range of this parameter is relatively small: from 4.61 to 6.72. This is qualitatively differs from another results. For example, in [6] acceleration is in interval from 1.10 to 7.31 for eight-processor system iPSC/860. But that algorithm uses both static and dynamic fault list grouping and some additional heuristics. Thus, the closeness of numerical data for all circuits shows the advantage of our basic scheme of the algorithm.

V. CONCLUSION

In this paper a new parallel algorithm for fault simulation of VLSI was proposed. Main idea consists in parallelizing of simulation of fault groups for each input vector. This is achieved by the concurrent many-threaded simulation of the groups of faults. In addition, it is organized a parallel thread of fault-free simulation of circuit for the next input vector. Such organization of calculation threads produces a new basic schema for the fault simulation algorithms. It allows eliminate the scheme in which the behavior of fault-free circuit is performed before simulation of groups of faults for current input vector, when only one core is loaded and all the other are idle.

The reported numerical results of computer experiments also show good scalability of the parallel version of our algorithm in terms of parallel calculation.

As a further research it is may be noted the study of the effectiveness of the new base algorithm scheme with various heuristics.

ACKNOWLEDGMENT

The author thanks the company Intel®, as well as its division Intel® Developer Zone for the opportunity to access the 12-core workstation of the Manycore Testing Lab. We express our personal thanks to Mike Pearce and Peter Hinsbeeck for providing technical support to access during the session and after it.

REFERENCES

[1] R. B. Mueller-Thuns, D. G. Saab, R. F. Damiano, and J. A. Abraham, "VLSI logic and fault simulation on general-purpose parallel computers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 3, pp. 446-460, 1993.

[2] J. Han and S.Y. Lee, "A parallel implementation of fault simulation on a cluster of workstations," in *Proc. IEEE International Symposium Parallel and Distributed Processing IPDPS*, 2008, pp. 1–8.
[3] P. Subbaraj, P. Sivakumar, and S. Nandhanam, "Circuit partitioning problem using graphical processing units," *Journal of Computer Science*, vol. 8, no. 5, pp. 705-710, 2012.
[4] C. P. Ravikumar, V. Jain, and A. Dod, "Distributed fault simulation algorithms on parallel virtual machine," *VLSI Design*, vol. 12, issue 1, pp. 81-99, 2001.
[5] E. Ivask, S. Devadze, and R. Ubar, "Distributed fault simulation with collaborative load balancing for VLSI circuits," *Scalable Computing: Practice and Experience*, vol. 12, no. 1, pp. 153–163, 2011.
[6] S. Parker, P. Banerjee and J. Patel, "A parallel algorithm for fault simulation based on PROOFS," in *Proc. IEEE Int. Conf. Computer Design*, 1995, pp. 616-621.
[7] D. E. Ivanov, Yu. A Skobtsov, and El-Khatib, "Distributed fault simulation of digital circuits," in *Proc. the Donetsk National Technical University. Series: "Computers and Automation",* vol. 107, pp. 128-134, 2006.
[8] D. E. Ivanov, "Parallel fault simulation on multi-core processors," *Electronic and Computer Systems*, vol. 6, no. 40, pp.109-112, 2009.
[9] M. A. Kochte, M. Schaal, H.-J. Wunderlich, and C. G. Zoellin, "Efficient fault simulation on many-core processors," in *Proc. the 47th Design Automation Conference ACM*, NY, USA, 2010, pp. 380-385.
[10] D. E. Ivanov and Y. A. Skobtsov, "Parallel fault simulation for sequential circuits," *Artificial Intelligence*, vol. 1, pp. 44-50. 1999.
[11] V. Gergel and R. Strongin, *Introduction to Parallel Programming for Multiprocessor Systems, Textbook*, Nizhny Novgorod: Press of NNSU, 2003.

Dmytro E. Ivanov received the M.S. degree in specialty "mathematician" from Donetsk National University, Ukraine in 1995; the Ph.D. in "computers, components and nets" from Donetsk National Technical University, Ukraine in 2000; an associate professor diploma on control system department in Donetsk National Technical University in 2004; Dr. of Technical Sciences (Dr. Habilitation) in "computer systems and components" from Donetsk National Technical University, 2013.

From December 2014 to present time he is senior scientific researcher in the Department of Applied Problems of Modern Analysis, Institute of Mathematics of National Academy of Sciences of Ukraine. From 1998 to 2014 he has been junior scientific researcher, scientific researcher and senior scientific researcher in the Department of Theory of Control Systems, Institute of Applied Mathematics & Mechanics of National Academy of Sciences of Ukraine. From 2000 to 2013 he has been an associate professor and since 2013 he is a full professor on Control System Department in Donetsk National Technical University. His research interests include technical diagnostic of digital devices, evolutional computation and parallel calculation.