Distributed Genetic Algorithm of Test Generation For Digital Circuits

Skobtsov Y.A., El-Khatib A.I., Ivanov D.E.

Abstract – **The distributed genetic algorithms are considered for problem of test generation. The different forms of parallelization of genetic algorithms are investigated for test generation.**

Keywords – **genenic algorithms, distributed calculations, test generation, fault simulation, digital circuits**.

I .Introduction

Genetic algorithms (GA) are successfully used for the test generation of digital circuits [1] (from 90th) along with the deterministic structural methods. Experience shows that genetic algorithms give the best results for the circuits, oriented to the data processing, while the deterministic methods more successfully work for sequential circuits with difficult control logic. In this paper represented parallel GA, which allow extending an effective application GA for this problem.

II. Distributed GA.

Inherent GA "internal" parallelism and possibility of the distributed calculations promote to development of parallel GA (PGA). The first papers in this direction appeared in 60-th years, but only in 80-th years, when accessible facilities of parallel realization were developed, the PGA researches adopted systematic mass character and practical orientation. The great number of models and realizations are developed in this direction, some of which are represented below [2].

Parallelism of GA gives the following advantages:

1) Search of alternative decisions of the same problem;
2) Parallel search from different points in decision space;
3) Good realization is assumed as islands or cellular structure;
4) Large efficiency of search even in the case of realization not on parallel calculable structures;
5) Good compatibility with other evolutional and classic procedures of search;
6) Substantial increase of speed execution on the multi-possessor systems.

Further we shall consider the modern main methods of the PGA realization. Most known is global parallelism which represented on Fig.1.a).

This model is based on simple (classic) GA in which the calculations are performed in parallel.

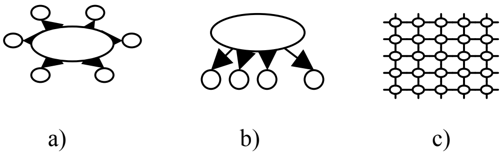


Fig.1 Different realization of parallel GA

This approach is faster, than classic GA, which can be executed sequentially, and does not usually require balance on the load as on different processors more frequent than all the values of fitness-functions for different individuals (strings) are calculated (having about equal computation complexity). The exception makes the genetic programming, where different individuals can strongly differ on the complication (treelike or graph- structures).

This model often named "**master-slave**". Many researchers use the pool of processors for the increase of speed execution of algorithm. At the same time the independent program passages of algorithm at different processors are executed essentially quick than at one processor. It must be noted, that in this case there is no co-operation between different passes of algorithm. It is extraordinarily simple method of implementation of simultaneous work (if it is possible) and it can be very useful. For example, it can be used for the decision of the same task with different initial conditions. By virtue of the probabilistic nature GA allow effectively using this method. At the same time we have minimum program changes, but advantages can be considerable.

In Fig.1.b) also represented an extraordinarily popular "**model of islands**" (coarse grain), where great number of sub algorithms simultaneously work in parallel, exchanging in the search process by some individuals. This model assumes direct realization on the computing systems with MIMD- architecture. Thus every “island” corresponds to its processor.

In cellular GA (fine grain), shown on Fig.1.c, parallelism usually will be realized on the computer systems with SIMD-architecture, where every processor represents subpopulation (from one individual). Although another papers are known, where authors use single possessor computers and systems with MIMD-architecture.

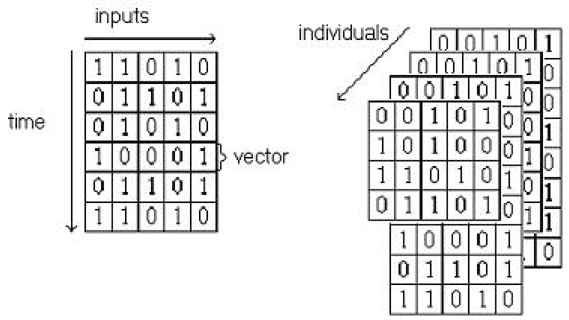
III. Genetic Algorithm of Test Generation

The purpose of automatic test generation is construction of input sequences of **binary** sets, which check up any physical defect possible in the process of production (or)

Skobtsov Y.A., El-Khatib A.I. – Donetsk National Technical University, Artema Str., 58, Donetsk, 83000, UKRAINE, E-mail: skobtsov@kita.dgtu.donetsk.ua
Ivanov D.E. – Institute of Applied Mathematics and Mechanics of NAS of Ukraine, R. Luxemburg Str., 74, Donetsk, 83114, UKRAINE, E-mail: ivanov@iamm.ac.donetsk.ua

exploitations of logical circuits. However there is the enormous number of possible potential physical defects. Some class of faults, which simulate the real physical defects, is usually examined therefore. Thus, fault is the model of (one or a few) physical defects. In practice more frequent all is examined by single stack-at constant faults, test generation for which usually gives satisfactory results (fault coverage) for real faults in the circuits. In the case of necessity (low fault coverage), test can be generated for other types of faults, such as shorts, a transistor is constantly opened (short), delays of propagation of signals, etc. It is known, that the test generation is a NP-difficult task, which means that in worst case it is solved by enumeration of all possibilities [1]. Therefore, in spite of the fact, that the great number of serial structural methods of test generation are developed, which for many circuits give good results (high fault coverage), the algorithm parallelizations for the tests generation were made an attempt [3,4].

During the test generation for digital circuits with application of GA, as an individual can be used a test sequence (Fig.2a). Population consists of the fixed number of test sequences, possibly, different length (Fig.2b).



1	1	0	1	0
0	1	1	0	1
0	1	0	1	0
1	0	0	0	1
0	1	1	0	1
1	1	0	1	0

a) individuals b) populations

Fig.2 Encoding Of individuals and population in GA

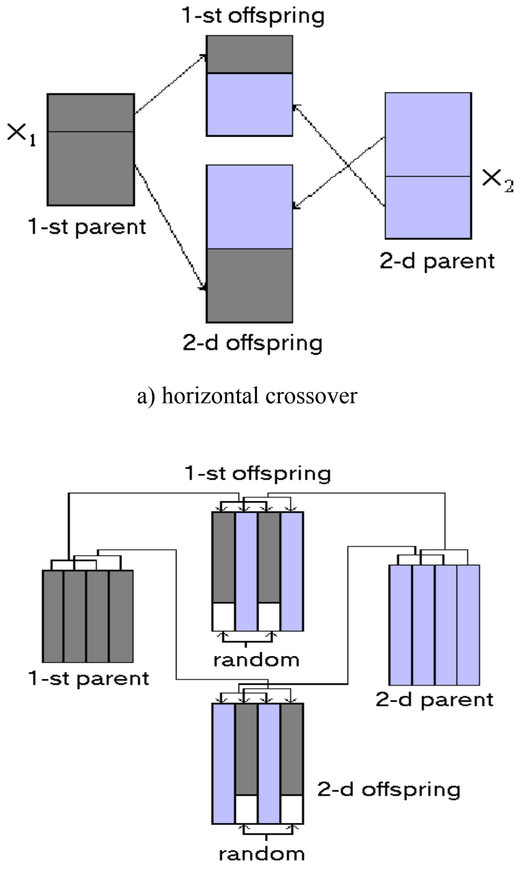
For the chosen encoding of individuals and populations the following problem oriented genetic operators can be used [1]:

1) **Crossover**. Two types of operation will be realized (Fig.3): vertical and horizontal crossover which are executed accordingly with probabilities P_v and $P_h = 1 - P_v$.

2) **Mutation.** Three types of this operator accordingly are used with probabilities P_{m_1}, P_{m_2} and P_{m_3}:

- Delete of one input vector from the by random chosen position. Application of this operation allows to reduce the length of the generated test sequence in that case, when a remote vector does not worsen test properties of sequence;
- Addition of one input vector in random position, that also allows to extend the search area of decisions;
- Random replacement of bits in a test sequence.

Because the purpose of test generation is design of input sequence, in which maximally differ the values of signals in fault and good circuits, the quality of test sequence (fitness-function) can be estimated as measure of difference of values of signals in fault and good circuits. In the simplest case the programs of logical simulation of good circuits are used for this purpose, allowing estimating the values of signals on two neighboring (in time) test patterns. On the basis dates of good simulation the following fitness-functions are developed:



a) horizontal crossover

b) vertical crossover

Fig.3 Operations of the horizontal and vertical crossing GA

$$h(v, f) = c_1 f_1(v, f) + c_2 * f_2(v, f) \qquad (2.1)$$

where - f_1 and f_2 is number of changes of signals on the outputs of logical gates and triggers accordingly ($c_2 >> c_1$).

Fitness function for the sequence s determined as the weighed sum of fitness-functions of separate input patterns:

$$H(s, f) = \sum_{i=1}^{i=\text{length}} L^i * h(v_i, f) \qquad (2.2)$$

where s is the analyzed sequence; v_i - vector from the examined sequence, I - position of vector in a sequence, f - given fault, $0 < L < 1$.

If test generation with mentioned fitness function is not success, then the fault simulation programs are used. Here fitness functions are based on the count of signal difference in good and fault circuits and have approximately the same kind as well as in the previous case.

IV. Parallel Genetic Algorithm of Test Generation

In this paper for pararallelism of GA we use a model «master - slave», because it requires the small changes in the existent version of software realizing GA of test generations and gives quite good results.

In this approach every processor has its own copy of population. The cost of calculation of values of fitness-functions (witch use the logical simulation) are evenly distributed on all processors. For all processors the same list of faults is used. Therefore for n individuals and P processors we take to every processor the P/n individuals. The values of fitness-functions are calculated by the slave processor and are sent to one selected processor (master), which collects all information and passes it to all processors. Every processor has information about the values of fitness-function for all individuals and can design next generation on this basis.

So the processor - master executes central part (kernel) of test generation, while the logical simulation (good and fault) of digital circuits will be realized on processors – slaves. With point of view of cost calculation the fault simulation is most critical. Different methods of organization of the distributed fault simulation, which are known, mainly based on breaking-up: 1) circuits on sub circuits; 2) test sequence on a subsequence. We will take combined approach combining these two methods.

On the first and second stages the generated input sequences are distributed between working processors. On the first stage every working processor is loaded by the generation of one subsequence. For balance the list of undetected faults is broken up on approximately identical subgroups.

At the end of each of three stages the points of synchronization are placed. When a processor - master arrives at these points, he passes to the wait mode, while all working processors will not make off the tasks, that guarantees global correctness of algorithm. Thus work between a processor-muster and workers is distributed as follows.

Processor - master :

- Performs all input-output operations with an user and file system: it reads circuit description and fault list, then, it writes the generated input test sequence;
- Initially spans «slave» processes on available procedures;
- Distributes the copies (internal form) of circuits and fault lists to every working processor;
- Organizes the process control of test generation: as soon as input sequence has to be fault simulated, it sends the proper message fop activating of working processors; when working processors finish their work, processor-master receives results and accordingly changes global data structures (general fault list, values of fitness-functions for individuals and, etc.).

A working processor keeps the local copy o of circuit (in internal format) and fault list. Every «worker» takes an input sequence from the «master» and determines the faults are detected by this sequence, by the logical simulation and calculates the values of fitness-function for individuals. It sends the got results to the master and wait next job. As the population size is much larger than the number of processors, good balance in the load of processors is achieved. For every working processor the change of local fault list with the detected and undetected faults from other working processors requires enough a lot of resources and it is critical.

Final results (test input sequences and fault coverage) are near to those, that is got on the single possessor computer system with the use of a similar algorithm. Quality of decision (fault coverage test) is not here lost and is in most cases got better, and time of test generation grows short substantially.

V. Distributed Fault Simulation

Described above distributed genetic algorithm of test generation mainly is based on the distributed fault simulation one. Now we will shortly describe also this approach.

Distributed fault simulation is organized in similar way and also is based on the «master-slave» approach. One processor here is selected as master and residuary processor – as slaves. Exist several approaches to perform distributed fault simulation: partitioning of circuit [5] and partitioning of fault list [6]. Our algorithm is based on the fault list partitioning.

Data flow chart for this scheme of computational process is showed on fig.4.

Every slave processors performs fault simulation on the data received from the master: circuit description and short fault list. The pseudocode of this process is given below.

```
slave_process_fault_simulation()
{
  search_of_master_process();
  if( master_was_found )
  {
    receive_circuit_description();
    receive_short_fault_list();
    parallel_fault_simulation()
    send_list_of_undetected_faults();
  }
}
```

Fig.4 Slave process algorithm

The kernel of this process is a procedure «*parallel_fault_simulation*», which is a regular fault simulator that used in one machine realization. In our case we used home built PROOFS-based fault simulator, described in [7]. Mark the main advantages of this algorithm that makes it very successful: 1) dynamic fault-list processing: detected fault is eliminated from fault list in the same time it detected, no simulation performs for this fault further; 2) fault sorting which allows include in one group the faults that cause the same simulation events; 3)

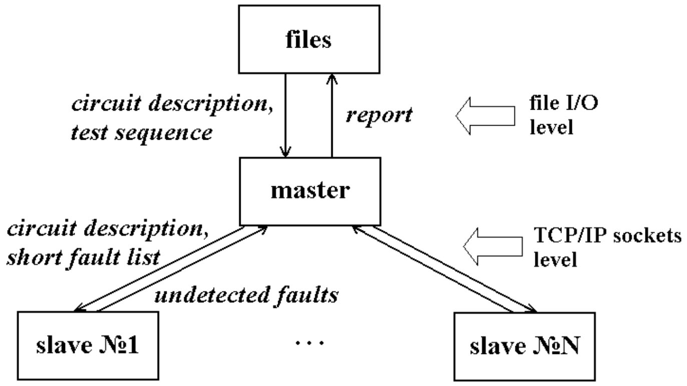


Fig.5 Data flow diagram for distributed fault simulation

the technique of functional fault injection.

Common data flow chart diagram that describes interaction among master and slaves processes is shown on fig.5.

It is necessary to notice that master process performs two types of exchange operation. File input/output needs to obtain circuit description and test sequence to be simulated. In contrast all data interchange among master and slave process is performed via TCP/IP sockets. This fact enables to construct calculation cluster on the common used computers. Authors used as such cluster 100Mbit local intranet.

As can see from data flow chart diagram master processor don't performs any simulation but organizes the calculation:

- Reads the circuit description to be simulated and input test sequence;
- Sends this description and test sequence for all client processors;
- Receives from slaves fault simulation results and makes common report.

Algorithm for master process for distributed simulation is given below.

```
distributed_simulation(circuit,test)
{
  number_of_slaves = search_of_slaves();
  if( number_of_slaves != 0 )
  {
    input_circuit_description();
    input_test();
    make_full_fault_list();
    partiting_the_fault_list(number_of_slaves);
    for( i=0 ; i< number_of_slaves ; i++ )
    {
      send_to_client_i_circuit_description ();
      send_to_client_i_part_of_fault_list();
      send_to_client_i_test_sequence();
    }
    for( i=0 ; i< number_of_slaves ; i++ )
    {
      receive_list_of_undetected_faults();
    }
    make_report();
  }
}
```

Fig.6 Master process algorithm for distributed simulation

Master process starts with search procedure of calculation clients. Done this it partition full fault list into several short one pro rata found clients. Further in cycle it makes the same work with all clients: sends circuit description in internal format; sends test sequence and corresponding short fault list. After this master passed in state of waiting data from clients. At the next step master receives the results of fault simulation from each client and makes general reports: fault coverage, common simulation time, time of simulation on every clients.

Proposed algorithm for distributed fault simulation was implemented on Windows platform into C++ Builder environment. As a core algorithm for client processors was brought slightly modified home-developed PROOFS-based algorithm. The modification is concern only TCP/IP sockets interaction with development environment: circuit input, receiving/sending fault list, obtaining test input sequence. Master process algorithm was implemented anew. All experiments were passed on the 100 Mbit/sec local intranet. As an input sequence was given randomly generated sequences, which consist of 1000 patterns.

At first we study the speed-up of simulation process depending on the number of the client processors. We give these experimental data for one of the large ISCAS-89 circuit – s35938 (fig.7). Secondly, we obtain experimental data for six largest circuits with eight clients (fig.8).

The distributed fault simulation algorithm constructed in described way enable a high parallelization of simulation process. But it is necessary to notice for both experiments it is impossibly to achieve linear speed-up of simulation time. This fact relates with the necessity for master make some irredundant job for all client processors: sending circuit description, input sequence and fault list.

CONCLUSIONS

In this paper a problem of distributed genetic test generation and fault simulation is studied. The possible way of organization of this process is described. Proposed by authors for solving these tasks algorithms that based on the scheme «master-slave» are described. Experimental data for distributed simulation algorithm for large ISCAS-89 circuits are given.

REFERENCES

1. Y.A.Skobtsov, V.Y.Skobtsov, D.E.Ivanov Evolutionary approach to the test pattern generation for the sequential circuits // Radioelectronics and informatics.- 2003, №3.- pages .46-51.
2. Skobtsov Yu.A., El-Khatib A.I. Parallel genetic algorithm // Proc. of Donetsk State Technical University, Series «Computers and automatic», vol.90.- Donetsk, DSTU Press.- 2005.- pp.137-144
3. D.Krishnaswamy, M.Hsiao, V.Saxena, E.M.Rudnick, J.P.Patel. Parallel genetic algorithms for simulation-based sequential circuit test generation // IEEE VLSI

Design Conference, 1997.- pp.475-481.

4. F.Corno, P.Prinetto, M.Rebaudengo, M.Sonza Reorda, E.Veiluva. Aportable ATPG tool for parallel and distributed systems // Proc VLSI Test Symp 1996.-pp.29-34.
5. S. Patil, P. Banerjee and J. Patel, “Parallel test generation for sequential circuits on general purpose multiprocessors”, in Proceedings of the 28th ACM/IEEE Design Automation Conference, (San Francisco, CA), June 1991.
6. T. Marcas, M. Royals and N. Kanopoulos, “On distributed fault simulation”, IEEE Computer, vol. 7, pp. 40-52, Jan. 1990.
7. Ivanov D.E., Skobtsov Yu.A. Parallel fault simulation for sequential circuits // Artificial intelligence.- Donetsk, DIAI Press.- 1991, №1.- pp.44-50.

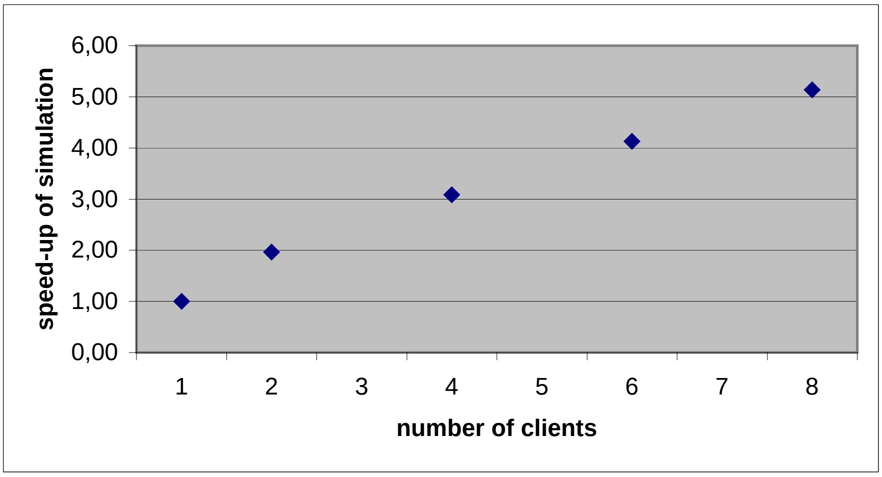


Fig.7 Speed-up of fault simulation for s35938 benchmark circuit depending on the number of the client processors

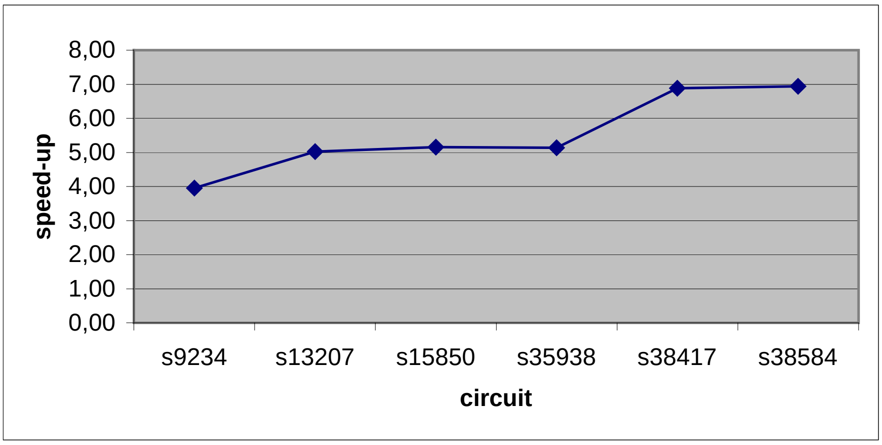


Fig.8 Speed-up of fault simulation for larges ISCAS-89 circuits with 8 clients realization.