An Evolutionary Method for Improving the Reliability of Safety-critical Robots against Soft Errors

Mahnaz Mohammadzadeh[1], Bahman Arasteh[2]
[1]Department of Mechatronic Engineering, Ahar Branch, Islamic Azad University, Ahar, Iran
Email: mmohammadzadeh1989@gmail.com
[2]Department. of Computer Engineering, Tabriz Branch Islamic Azad University, Tabriz, Iran
Email: b_arasteh@iaut.ac.ir

ABSTRACT
Nowadays, Robots account for most part of our lives in such a way that it is impossible for us to do many of affairs without them. Increasingly, the application of robots is developing fast and their functions become more sensitive and complex. One of the important requirements of Robot use is a reliable software operation. For enhancement of reliability, it is a necessity to design the fault tolerance system. In this paper, we will present a genetic algorithm and learning automata with high reliability to evaluate the software designed into the robot against soft-error with minimum performance over-head. This method relies on experiment; hence, we use the program sets as criteria in evaluation stages. Indeed, we have used the error injection method in the execution of experimental processes. Relevant data, regarding program execution behavior were collected and then analyzed. To evaluate the behavior of program, errors entered using the simple scalar simulation software.

KEYWORDS: Soft-Error, Fault tolerance, Retrieval Blocks Technique, Evolutionary Algorithms

1. INTRODUCTION

In the industrial world in which the role of computer and mechanized systems are too sensible, appearance and occurrence of robots are the notable and impressive phenomena in the automation of the industrial systems [1]. The extent of robot´s application has reached a climax and can be exploit in the many of factories and industry products' lines, especially, the automotive and aviation industries [2,3]. Also robots play an important role in the medical field along with technological advances and the development of robotic science. Therefore, it is necessity to access reliable services. The need for safe and reliable software operations is considered as an important requirement for modern life because the software undertakes the main role in accomplishing functional systems. Hence, it has a specific importance in reliability degree of systems. In order to raise reliability, we should design the system in the form of fault tolerance. Errors can be divided into three types: Soft errors, permanent errors and intermittent errors. In this paper, we

focus on soft-errors. Soft- errors can occur in short time and lead to irrecoverable damages. External events are the cause of soft - errors/ transient – errors occurrence [4].
Generally, with application of two redundancy techniques: software and hardware, we can boast reliability of internal system in robots against soft- errors. Increase of reliability through hardware requires hardware changes. The disadvantages of this method include reduced performance, weight gain, power consumption and cost [5]. In contrast, software methods do not need hardware changes. The main advantage of software methods is low cost and it´s flexibility that draws researchers´ attention. In this paper we present a genetic algorithm and learning automata with a high reliability to evaluate the software designed into the robot against soft–error with minimum performance over head.

2. RELATED WORKS

Initially, we briefly review the pervious works. In reference [6], the authors have used the fault injection technique to measure the vulnerability of sections in the program. This technique needs special hardware and software tools and it is a time consuming process. In reference [7], the researchers have used the failure injection technique to measure the vulnerability of each instruction in the program, and using this method for the large program is very consuming. In [8], the instructions that affect global variables are considered as critical instructions. The proposed method in [9] provides a simple static analyzing technique where instructions that affect control flow and memory address are tagged critical. This method can just identify small portions of vulnerable instructions and blocks rather than all of them. In reference [10], the researchers have used the statistical analysis and manifested the error that leads to automatic enhancement reliability. In reference [11], the researchers tried to retrieve the transient error. According to [12], the researchers presented software method to enhance reliability of the application program with minimum over-head. As for [13], the researchers evaluated the critical data in application software. In this paper, we used a heuristic method to extract blocks of a program that derive inherently soft-errors and the redundancy techniques used just on the vulnerable blocks.

3. PROPOSED METHODS

In this section, we introduce a heuristic method to identify soft- error hiding and masking blocks of programs. The blocks that hide and mask soft – errors inherently with high probability are taken into consideration as invulnerable blocks. After the estimation of hiding factor for each block, we apply the redundancy technique just on the vulnerable blocks. Hence; reducing the amount of redundancy leads to the reduction of the performance overhead.

3.1 Background

The positive point is that only a fraction of soft- errors affect the final result of a program [14]. There are many points in the system hierarchy at which soft- errors can be hidden inherently without altering the program

outputs. A soft- error will be masked at application level if it alters data or flow of program and this state doesn´t affect program outcome and results. This phenomenon is due to inherent features and redundancy in the application level. There are several sources of inherent features and redundancy at the application level like algorithm and code, that improve the resiliency of program against soft-errors. Logical operations are major sources of error hiding phenomenon in the program. These features reduce the probability of soft-error propagation to the program output. In fig.1, block 3 shows the corresponding operation. In this program, block 3 is a type of non-operational need. Alterations in data and instructions of this block don´t influence final result due to soft-errors. In other words, soft-errors inherently hide and mask in the block.

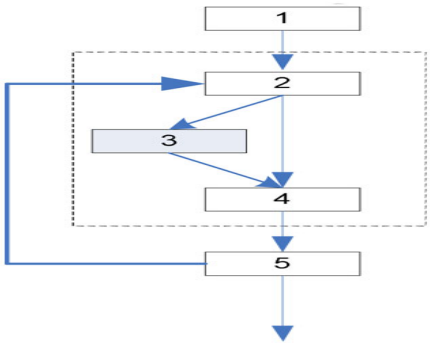

Fig.1. Function of program

It can be argued that soft-errors in some blocks segment of programs are masked and don´t propagate to the program output. Identification of these blocks by a heuristic method genetic algorithm and learning automata is the goal of this study.

3.2 Genetic Algorithms and Learner Automata

Genetic algorithms operate based on evolution idea in the nature and search the final solutions among the population of potential solutions. In each generation, it selects the best generation and generates the new population of off springs after breeding. In this process, the adapted people remain in next generation with high probability. In the initial algorithm, many of people are selected randomly and the target function evaluates each of them. If the condition is not satisfactory, the next generation would be selected based on fitness function with selection of parents and off springs undergone the mutation and crossover stages. Then, they compute fitness degree of new offspring and the new population generated by means of succession of offspring instead of parents. This process iterate until the condition is satisfied. To gain lots of information about genetic algorithms do refer to [15]. Next, we will consider the learning automata.

In learning automata, learning is the selection of optimization action among the set of allowed actions. These actions were exerted on the random environment. The environment responded to automata action by a random feedback among the set of allowed responses. The environmental reaction depended on automata action statistically. The environment terms include collection of all external conditions and their effects on automata performance. In fig.2. , we showed the association between learning automata and environment. For more information about learner automata refer to [16-19].

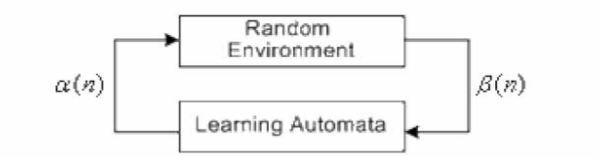

Fig. 2. Connection of LA with random environment

If learner automata selects α_i operation in nth iteration, and it receive a desired response from the environment, the probability of $p_i(n)$ increases and the probability of other operations decreases. On the contrary, if the environment responds undesirably, the probability of α_i decreases and the probability of other automata operation increases. Anyway, the transition exerted such that the sum of $p_i(n)$ s is constant and equal to one. The probability of operations changes as follows [20,21]:

- desired response from environment

$$\begin{aligned} P_i(n+1) &= p_i(n) + a(1 - P_i(n)) \\ P_j(n+1) &= (1 - a)P(n) \qquad \forall j, j \neq i \end{aligned} \tag{1}$$

- undesired response from environment

$$\begin{aligned} P_i(n+1) &= (1-b)p_i(n) \\ P_j(n+1) &= \frac{b}{r-1} + (1-b)P_j(n) \\ P_i(n+1) &= P_i(n) + a(1-\beta_i(n)).(1-P_i(n)) - b.\beta_i(n).P(n) \\ P_j(n+1) &= P_j(n) - a.(1-\beta_i(n)).P_j(n) + b\beta_i(n)\left[\frac{1}{r-1} - P_j(n)\right] - a.(1-\beta_i(n)).P_j(n) \end{aligned} \tag{2}$$

3.3 Idetification of Error Hiding Blocks by A Genetic Algorithm and Learner Automata

Evolutionary computation is a field of machine learning which attempts to mimic the process of evolution to find solutions for a certain problem. Genetic algorithm and learner automata are commonly applied to a variety of problems involving search and optimization. These algorithms generate a sequence of populations by means of selection, crossover and mutation mechanisms. GA+LA generate the population of chromosomes via selection, crossover and mutation mechanisms.

Agenom in a genetic algorithm shows one potential solution. In each generation, all of the chromosomes are evaluated by a fitness function. The fitness function measures their suitability within their environment and the crossover operators exert on parents via learner automata.

The algorithm will iterate until the population has evolved into a solution for the problem or it will iterate till the time the maximum number of iterations have occurred. Our aim is the quantification and measurement of the impact of each block and conditional branch on the program output and consequently the identification of the error hiding blocks.

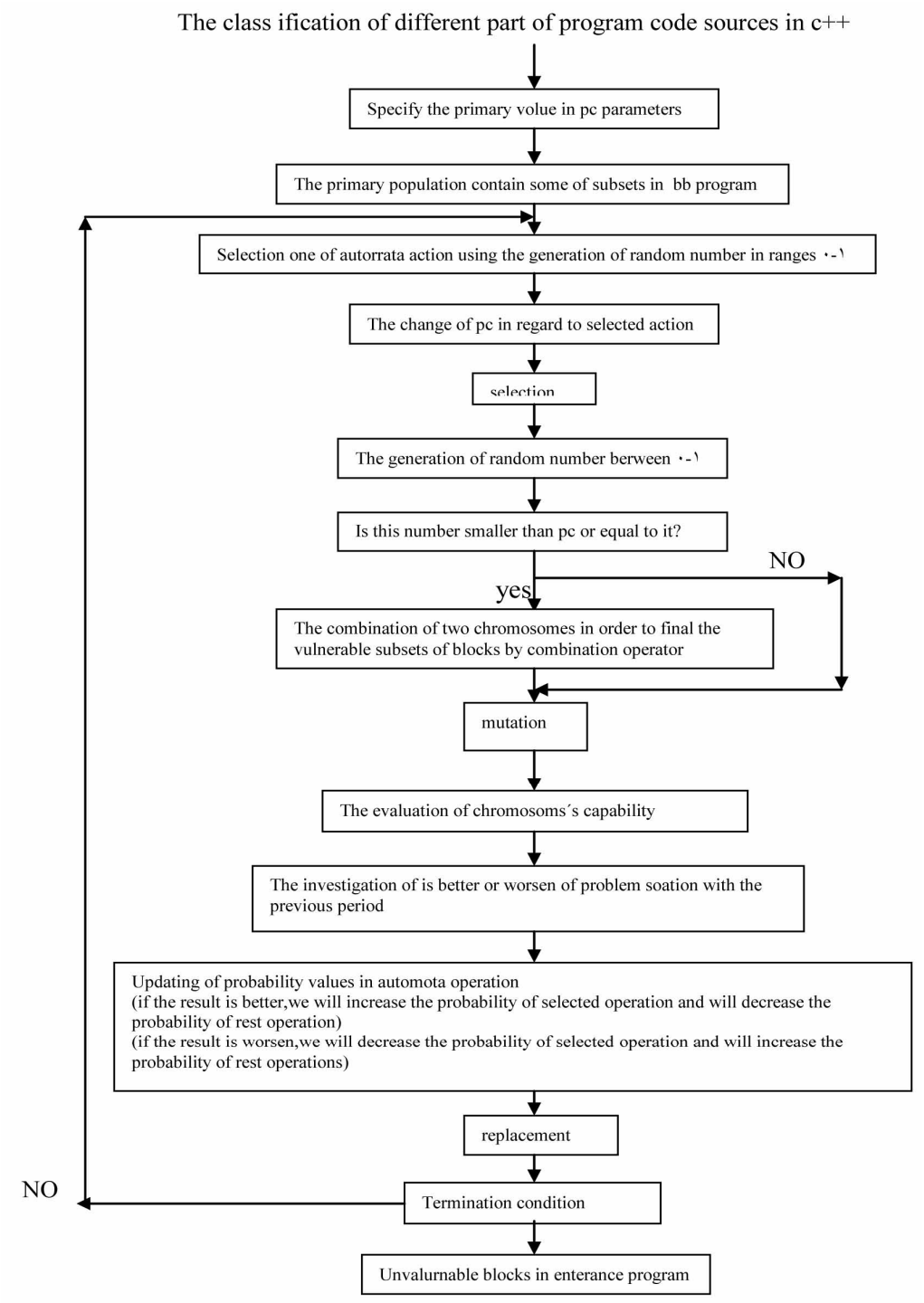

Fig.3. The flow chart of proposed method

In order to identify error hiding block, we propose GA+LA in our developed GA+LA, every consequence of block program is considered as a chromosome. In other word, each chromosome is represented by a binary string of length M. Every bit in the corresponding string of a chromosome is related to a block in the program.

The initial population is generated randomly. In each generated chromosome, the crossover and mutation operator were exerted on parents. During the evolution process, the effect of each branch in the program output is measured. Indeed, we search for the conditional branches and succeeding blocks which inherently mask the effects of soft- errors in the program.

In this paper, we used learning automata, new cross over methods to combine parents. in this method, the automata selects two chromosomes of parents and selects two genomes ،I, J randomly in one of two parents' chromosome. Then, this same two genome are selected into other parents' chromosome. The set of genome, number I, j termed transposition set. Therefore, the genomes with the same number replaced each other in two transposition sets that called the off-springs of two parents automata terminologically. After computing the fitness of each chromosome in the current population, the algorithm selects those chromosomes which don´t affect the program results. At the end of the final population, the generated quantity for each block shows its average vulnerability.

Fig.3. illustrates the flowchart of proposed method.

4. THE EXPERIMENTED EVALUATION SYSTEM

In this section, the experimental approach has been utilized in order to introduce the selected programs in proposed method. In the error injection experiment, we use seven programs as criteria. The following work load programs all written in the C programming language. Our criteria include:

- Lagrange
- N-graph
- Regression
- Tsp
- Qsort
- Matrix-mult
- compress

In table 1, we describe specifications of each criterion.

Our experiments were conducted according to following stages:

- Firstly, the error hiding blocks of each program were identified by the proposed method.
- Secondly, the software- based errors were injected into the source programs through the error- injection process and then the programs are compiled by the GCC in the simple scalar tool set.

• Thirdly, the programs are executed in simple scalar tool set with the sim-out order simulator.

• Eventually, we obtain simulation results and analyze them.

5. EXPERIMENTAL RESULTS

Error coverage arid performance over-head are the parameters for evaluating this method. We have done three experiments as the following.

- A total of 10000 software- based errors are injected into the code and the data of the workload programs in the experiments and the results were collected.
- 40% of random instructions in the program are protected against soft-errors and those of modified programs have been used in error injection experiments. Similar to pervious state, 10000 errors are injected into the code and the data of each program were coded.
- After identifying vulnerable blocks by proposed method and protecting them, the error is injected

Table 1: Specifications of criterion

Criterion	Description	Problem type	Entrance variables	
			description	Amplitude domain
lag	Insert of poly-nominal with lagranges	Numerical analysis problem	The number of point for incorporation in any situation of each point(RND)	N=20
n-graph	The computation of function root	Numerical analysis problem	Replication multitime for finding function root	N=100
regression	The anticipation value of numerical variable by linear regression	Statistical analysis problem	The number of dependent and independent variables	N=50
Tsp	The solution of problem TSP	Graph combination problem	The number of graph nodes and graph length	N=100
Qsort	The sort of numerical list	Combination problem	The number of random numbers	N=100
Matrix-mult	The multiplication of two matrices	Numerical problem	Row and column length in any element	N=5
compress	The compression of one line randomly	Compressed text problem	The production of line randomly and lin. length	N=20

Table 2.The results of program after error injection

Result classes	Description	Detection mechanism
Correct	Production of correct output by the program	Checking result of the erroneous program with golden run
Fail-Silent Data Corruption (SDC)	Production of incorrect output by the program	Checking result of the erroneous program with golden run
System Exception	Hang: program timeout	Detected by simulator time-out and the incomplete program will be killed
	Crash: Abnormal program termination (invalid instruction, invalid memory address, overflow, segmentation error)	Detected by simulator exceptions

Table 3: The results of error injection

original	lag	n-raph	regressi	TSP	Qsort	Matrix-m	compres	AVG
	14%	23%	8.19%	11.60%	13.80%	19.54%	6%	14%
random	9.30%	15.30%	5.80%	8.62%	10.20%	14%	2.60%	9.36%
prposed	0.40%	6.90%	1.72%	2.90%	2.30%	0.90%	1.80%	2.42%

Table 4: The comparison of performance overhead in proposed method 8th other method

	Method presented in [10, 11]	Method presented in [12, 13]	Proposed Method
AVG. execution time-overhead	> 20%	> 33%	< 27%

In any execution, one bit of data or code in the program has changed as error. In the execution of any program, a total of 5000 data based errors and 5000 code- based errors and one bit are injected in memory space considering the volume and code of program. Indeed, any program executed 10000 times and one error was injected in any execution. Error injection in the programs are executed in simple scalar tool sets. After an error is injected, the following program outcomes are possible according to Table 2.Based on the result of the experiments, table 3 demonstrates the result of error injection experiments in the various blocks of work load programs. Failure rate in the proposed method was approximately decreased three times regarding random redundancy. So as to evaluate the performance overhead, we

exploit the sim-out order simulation tool set. In comparison with other methods, the proposed method has higher time overhead in some aspects and had lower time overhead in other aspects. In table 4, we compare the performance overhead in the proposed method with other methods. We obtain the number of jump instructions, rate of memory consumption and compute the performance over head and analyze the results and compute the performance overhead of proposed method.

6. CONCLUSIONS

Due to the inherent features and redundancy, only a fraction of soft- errors may lead to system failure. We analyze the sources of soft- errors hiding and masking at the program level. This paper presents a heuristic method to identify and categorize the inherently error hiding blocks of a program. Our goal is the identification of the program blocks and protecting vulnerable blocks against soft-errors the error-injection based experiments are used in order to evaluate the proposed method. The experimental results show that the effectiveness of the proposed method is higher than the previous methods. The proposed method can be used soft-error tolerance techniques and had highly reliability.

REFERENCES

[1] Rajabzadeh, G. Miremadi and M. Mohandespour (1999). Error detection enhancement in COTS superscalar processors with performance monitoring features,Journal of Electronic Testing: Theory Application(JETTA), 20(5), pp. 553–67, 2004

[2] Profeta, N. Andrianos, Yu. Bing, B. Johnson, T.DeLong and D.Guaspart, (1996). Safety-critical systems built with COTS, Computer, 29(11), pp.54–60.

[3] P. Tso and P. Galaviz, (1999). Improved aircraft readiness through COTS, In IEEE systems readiness technology conference (AUTOTESTCON_99), pp. 451–6.

[4] M.Jafari-Nodoushan, G.Miremadi and A.Ejlali (2008). Control-Flow Checking Using Branc Instructions, In Proceeding of the 8th International Conference on Embedded and Ubiquitous Computing, 2008.

[5] Yenier, U. (2003). Fault Tolerant Computing in Space Environment and Software Implemented Hardware Fault Tolerance Techniques, Technical Report, Department of Computer Engineering, Bosphorus University, Istanbul.

[6] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, L.Tagliaferri, (2003). “Data Criticality Estimation in SoftwareAppliction” ,INTERNATIONAL TEST CONFERENCE

[7] D. Borodin and B.H.H. Juurlink, (2010). ” Protective Redundancy Overhead Reduction Using Instruction Vulnerability Factor”, ACM ,CF’10, Italy

[8] Shuguang Feng, Shantanu Gupta, Amin Ansari and Scott Mahlke (2010). “Shoestring: Probabilistic Soft-error Resilience on the Cheap,” in ASPLOS.

[9] D. Thaker, D. Franklin, J. Oliver, S. Biswas, D. Lockhart, T. Metodi, and F. T. Chong (2006). “Characterization ofError-Tolerant Applications when Protecting Control Data,”In Proc. of the IEEE Int’l Symp. on Workload Characterization.

[10] K. pattabiraman, Z. Kalbarczyk, R. Iyer (2011). “Automated Derivation of Application Aware Error Detectors Using Static Analysis: Trusted Approach”, IEEE Transaction on Dependable and Secure Computing, Volume 8 , Issue 5.

[11] T. Vijaykumar, I. Pomeranz and K. Chen, (2002). “ Transient Fault Recovery using Simultance Multithreading” , in 29th Internationa Symosium on Computer Architecture (ISCA).

[12] Benso, S. Chiusano, P. Prinetto. L. Tagliaferri (2000). “C/C++ Source-to-Source Compiler for Dependable Applications”, in IEEE International Conference on Dependable systems and Networks (DSN

[13] A. Benso, S. Di Carlo, G. Di Natale, P. Prinetto, L. Tagliaferri, (2003). "Data Criticality Estimation in Software Application", in International Test Conference, pp. 802-810.

[14] B.Arasteh., A.Rahmani., A.Mansoor, GH.Miremadi (2012). " Using Genetic Algorithm to Identify Soft-Error Derating Blocks of an Application Program",EuromicroConference on Digital System Design,

[15] D. E. Goldberg, (1989). "Genetic Algorithms in Search,Optimization and Machine Learning", Reading, MA,Addition-Wesley.

[16] P. Mars, K. S. Narendra, and M. Chrystall (1983). "Learning Automata Control of ComputerCommunication Networks",Proceedings of Third Yale Workshop on Application of Adaptive Systems Theory, Yale University

[17] K. S. Narendra, and M. A. L. Thathachar (1989). "Learning Automata: An Introduction", Prentice-hall, Englewood cliffs.

[18] M. R. Meybodi, and S. Lakshmivarhan, (1983). "A Learning Approach to Priority Assignment in a Two Class M/M/1Queuing System with Unknown Parameters", Proceedings of Third Yale Workshop on Applications of Adaptive System Theory, Yale University, 106-109

[19] B. J. Oommen, and D. C. Y. Ma, (1988). "Deterministic Learning Automata Solution to the Keyboard Optimization Problem",IEEE Transaction on Computers, Vol. 37, No. 1, 2-3

[20] Narendra, K.S. and Thathachar, M.A.L. (1989). "Learning Automata: An Introduction", Prentice Hall, Inc

[21] M. R. Meybodi, H. Beigy. (2002). Utilizing Distributed Learning Automata to Solve Stochastic Shortest Path Problem. Technical Report, Soft Computing Laboratory, Computer Engineering Department, Amirkabir University of Technology.