Proposing an Efficient Software-based Method to Enhance Reliability of Computer Systems against Soft Errors

Nahid Saadati[1], Bahman Arasteh[2]
[1] Department of Mechatronic Engineering,Ahar Branch,Islamic Azad University,Ahar,Iran
Email:nahidsaadati13@gmail.com
[2] Department of computer Engineering,Tabriz Branch,Islamic Azad University,
Tabriz,IranEmail:b_arasteh2001@yahoo.com

Abstract

In recent years, along with rapid developments in technology, computer systems have increasingly become more integrated and more modular. Indeed, the reliability and efficiency of computer systems are of high significance. Hence, the quantitative evaluation of the optimization of reliability indexes in computer systems is considered to be a crucial issue. Reliability enhancement of computer systems against electromagnet and radiation interferences is a critical requirement in industries. Enhancing reliability can prevent system failure and fault and avoid financial and humanistic losses. Accordingly, software can play outstanding roles in reducing the number of errors in software programs. Consequently, software can enhance the reliability of computer systems. In this paper, an efficient software-based method was proposed to enhance the reliability of computer systems against soft errors. The results of the experiments revealed that the proposed method had lower overhead, higher efficiency and higher error coverage than the earlier methods.

Keywords: reliability, fault tolerance, error, slicing, redundancy

1- Introduction

Nowadays,we live in a world where computers are ubiquitous. Sometimes, a computer system such as a digital machine can be easily identified, but sometimes a computer system such as the electromechanical part of machines cannot be identified easily. In line with the extensive use of computer systems in practical and functional and security applications, there is a critical need to develop products which users can rely on. Different methods can be used to meet these needs which have been developed based on the intensive use of a machine or the faulty module. Such methods have been proposed to sort out the reliability issue of the practical and security applications. Nevertheless, it should be pointed out that, in recent years, achieving high reliability and efficiency of programs is considered to be a remarkable challenge.

Indeed, enhancing the reliability of computer systems helps prevent the fault occurrence in computer systems; consequently, reliability enhancement can obstruct and prevent human and financial and other kinds of losses. It can be argued that software plays a remarkable role in the reliability of computer systems. For increasing reliability, computer systems should

be designed in such a way that they are tolerable against errors and this is one of the approaches for preventing system failures. Error tolerability can be designed and implemented with respect to the software or hardware of the system. Error can occur in the following two ways: persistent errors are the first type of errors; in case this error type is not corrected or removed and if it remains in the system for a long time, it will lead to the system failure. On the other hand, soft error or transient error is the other error type which appears within a short time and results in irreparable and irremediable losses in the system.

To achieve reliability in computer systems, one can use the approach of error detection which is regarded as one of the most effective techniques. In general, not detecting or identifying errors of computer systems at an early time can result in the destruction and loss of data and facilities. Furthermore, it should be noted that, in some situation, human losses can happen if such errors cannot be detected and identified. Thus, it should be noted that there is an increasing emphasis and focus on error detection in scientific and industrial communities. In recent years, due to not being able to detect errors, different problems have been created in computer systems. As a case in point, The race 25 which is a machine developed for radiation therapy has erroneously used more radiation for six patients and it is attributed to error occurrence in the system.

The objective and rationale behind this study is to make computer systems resistant to electromagnet interferences and radiations which are due to Alpha particles and cosmic radiations. Also, another objective of this study is to enhance the reliability and resistance of computer systems considered to be an essential requirement in different industries, namely medical industry, military and aerospace industries. Hence, it can be maintained that the contribution of the present study regarding reliability enhancement is of high significance for modern community and the related industries. In fact, the purpose of this study is to propose a software-based method with minimum overhead so as to enhance the reliability of computer systems with safety-critical applications against soft errors.

2- Identifying the Vulnerable Parts

The main responsibility of the proposed method is for examine and investigate the control flow, and to segment a program in the form of basic blocks or segments which are free of code branches. A specific instruction is attributed to each block and an error is detected by comparing the run time instruction with the measurement instruction. The most important techniques are related to the control flow examination, i.e. RSCFC, ECCA and CFCSS. In ECCA method, a unique prime number greater than two is allocated to each block which is referred to as the block identity. Two lines of code are allocated for each block. CFCSS method utilizes a GSR (global signature register) which includes the run time instruction for node V_n. This method examines the control flow. The register compares the run time instruction with the instruction produced at the time of compilation and the branch accuracy is determined based on this comparison (i). At the compilation time, the unique prime number S_i is allocated to each

basic block. At the run time, if G register includes a different amount of S_i which is allocated to the current node, an error will occur in the program. The time G for transmitting from one basic block to another is produced and the instruction function produces a new G in the target branch node. Using the instruction of the previous node and the new node, F function measures G through the following equation:

$$F=f(G,di) =G \text{ xor } di \quad (1)$$

RSCFC is considered to be a novel method in which the program is divided into a number of basic blocks. In the first stage, the dependencies among blocks are extracted; then, based on the type of dependency, a sign is allocated to each block in which the available dependencies are coded. For detecting the available faults in the control flow of a program, using the available data at the beginning and the end of blocks, one can use AND logical operation among the run time signs. It should be highlighted that RSCFC method has a better fault coverage and efficiency than previous methods; moreover, it consumes less memory.

For examining the efficiency and memory consumption in the error injection method, three programs, i.e. test program, insertion sorter, rapid sorter and matrix multiplication were selected.

3- The Proposed Method

The major aim of slicing was to best remove program faults and errors by limiting and restricting the search range so that the error location can be detected. As the resulting set is smaller and more accurate, the obtained slice will be better. The issue of limiting and restricting the search space is aimed at discovering error cause in the programs. Error elimination includes the two stages of error detection and error correction. Detecting error location is easily accomplished in small programs. However, in large programs, manual error detection is considered to be a boring and time-consuming task. Hence, using methods to automate and minimize the search domain is essential. For example, statistical methods and delta error elimination are deemed to be methods which can significantly contribute to error detection. Nevertheless, it can be maintained that one of the methods which has been extensively studied in recent years is slicing. It is a method used by programmers to summarize a program. Hence, the reduced and executable summarized program is referred to as a slice. This method is used to break a program into smaller parts; consequently, such smaller parts will be easily perceived, tested and kept. Perceiving and understanding the entire program in order to make a change only in a small part of it is highly time-consuming. Hence, using techniques to minimize the program is inevitable. With respect to slicing, two issues should be taken into consideration: Firstly, a slice from a program is obtained by removing a set of program sentences and phrases. Secondly, it includes the features of the main program. An algorithm is considered to be appropriate which not only produces slices including the mentioned features but are also small as much as possible. Indeed, different concepts for

slicing the program and a number of methods have been proposed for measuring the slices. The major reason for the diversity of the methods is that there are different programs, and differing features are required in different slices. According to the definition, a slice is a set of instructions and the control of program documents which are directly or indirectly affected by the slicing criteria; however, they do not necessarily create an executable program. Slicing is accomplished in three methods of data slicing, complete slicing and related slicing. In data slicing, the slices are measured based on the data dependencies among the sentences and phrases. For slicing, definition-use chains are firstly created and the slices are measured. Depending on the type of slices, the chains are created as dynamic or static ones. In general, this method operates successfully for the errors which do not change the control flow. A significant function of this type of slicing is that it can obtain memory errors. In complete slicing, besides data dependencies, control dependencies are also taken into consideration. In case erroneous sentence has no direct impact on the output, data slicing will not cover it. Indeed, there is a remarkable distinction between static and dynamic slice which was previously measured without considering the input assumptions of the program. In fact, it should be noted that, in certain cases, dynamic slice relies on test. Furthermore, related slicing, in addition to data and control dependencies, includes probabilistic dependencies of conditional sentences which have no impact on the respective sentence (error output). However, if they were measured otherwise, they had impacts on the measurement. This type of slicing has no remarkable applications in real errors. In first stage of the proposed method, the program is sliced. The rationale behind slicing is to facilitate error elimination from the program by restricting and limiting the search space so that the error location is easily detected. As the obtained set is smaller and more precise, the obtained slice will be more desirable and better. Two issues should be considered with respect to the slice: firstly, it should be noted that a slice of the program is obtained by eliminating a set of program sentences. Secondly, it includes the features of the main program. In the second stage, the static slice of the algorithm which has no data about the input values of the program is used. At this time, the algorithm should preserve the interesting behaviors of the main programs. Consequently, a relatively small piece of the program will be produced. In the static slice, CFG flow control graph is used. It has a slice which is the same as the static slice criterion. They operate in an abstract level. It is only the data which can be used about each of the expressions; finally, they are identified as dead variables. Hence, a remarkable percentage of data and codes are eliminated. Indeed, throughout these two stages, only the instructions and data which affect output remain.

In the third stage, the redundancy technique was used; it is regarded as one of the most important tools in creating error tolerance in systems. Indeed, the data redundancy technique used in the present study adds redundant data to the main data and allows the system to detect the errors; that is, the blocks which are vulnerable to error are

identified. Then, these blocks are protected against soft errors. Repetition is used for protecting vulnerable and sensitive blocks against soft error. In other words, other data are defined as the redundant data for each sensitive one and the instructions related to the valuation and use of these instructions should be repeated. Consequently, software reliability is enhanced through minimum redundancy.

4- Experiments

In this study, a test-based method was used in the simple-scalar simulation environment. Indeed, simple-scalar is deemed to be an architecture-based simulator. It has a number of functions and tasks where the most common one is the architecture analysis. It should be mentioned that the size of all the codes used in the simulation was relatively small since simple-scalar infrastructures provide an extensive set of policies for implementing the majority of common models. Software architectures of the programs are modeled by means of driving implementation techniques in which the sample instruction set and the input/output sample module should be used. The instruction set interprets each of the instructions and directs the activities of the hardware model through the interpreting interface.

In this study, a large number of error injection experiments were carried out to investigate the impact of the proposed method on the reliability enhancement of the computer systems. In the respective experiments, a set of programs are used for test and trial. Different programs with different features were tested in the study. The error was injected on the code or data of the program. Then, the behavior of the program in the presence of the error was examined. In the process of the experiments, each program was tested and examined in two stages: in the first stage, a set of errors was injected on the main program. Then, after applying the proposed method on each program, in the second stage, a set of errors was injected again on each of the respective programs. Finally, the results of the two stages were compared with one another which revealed the impact and effectiveness of the proposed method on enhancing the reliability of programs with respect to the injected errors.

4.2. Test and trial

The error model used in this study is a single-bit error and the errors were injected at the time of running the program. While running the program, one bit of the program data or code is changed as an error. In running the program, depending on the size of the code and program, one bit and a total of 5000 data errors and 5000 code errors were injected in the memory space. Indeed, each program had 1000 errors where one error was injected in each run of the program. As mentioned above, error injection was conducted in the simple-scalar simulation environment. Moreover, it should be noted that the errors were injected randomly and uniformly in the code space of memory. After the injection of each error in the simple-scalar environment, different results might occur as given in table 2 below which show the descriptions of the programs used in the study. Also, table 3 shows the characteristics of the programs.

Table 1. Comparing memory overhead and efficiency

	Memory overhead (%)			Performance overhead (%)		
Program	CFCSS	ECCA	RSCFC	CFCSS	ECCA	RSCFC
BS	1.32	1.54	1.45	1.86	6.78	1.74
QS	1.33	1.32	1.38	1.33	1.86	1.21
MM	1.12	1.16	1.13	1.56	3.02	1.43
FFT	1.23	1.6	1.34	1.34	2.86	1.43

Table 2. Different occurred results

Class result	Description	Detection mechanism
Right	The production of right output by the program	Examining error results of the program by golden running of the program
Silent data corruption (SDC)	The production of wrong output by the program	Examining the result of program errors by golden running of the program
Detecting and removing the interruption time by the simulator	interruption: the program fails	Exceptional system
Detecting exception by the simulator	Failure: abnormal finishing of the programs (invalid instruction, invalid memory address, overflow, error classification)	

Table 3. Characteristics of the tested programs

Tested program	Description	Input variables	
		Description	range
Heap sort 9max)	Sorting a list of numbers	Random production of a thread	N=100
Radix sort	Sorting a list of numbers	Random production of a thread	N=100
N. queens	Predicting the value of a numerical variable	Some random numbers	N=50
Binary search tree (BST)	Conducting search operation	Some random numbers	N=100
Linked list	Sorting a list of numbers	Some random numbers	N=100
Travelling salesman problem (TSP)	Solving TSP	The number of graph nodes and the length of the node	N=100
Knapsack problem	Sorting a list of numbers	Some random numbers	N=20

- An example before and after slicing the program: heap sort (max) sorting algorithm

Heap sort (max) is considered to be one of the data sorting methods which have been implemented based on the features of the heap tree. According to the heap tree definition, in a heap-max or min-heap, the biggest or smallest value among the data is located in the root of the tree. Finding the largest or the smallest element among the elements has the Θ (1) fixed cost. By eliminating this element from the tree, the next largest or smallest element is again located in the root. Thus, by consecutive elimination of the elements of the heap tree and their insertion in the new place, a sorted descending or ascending array will be obtained.

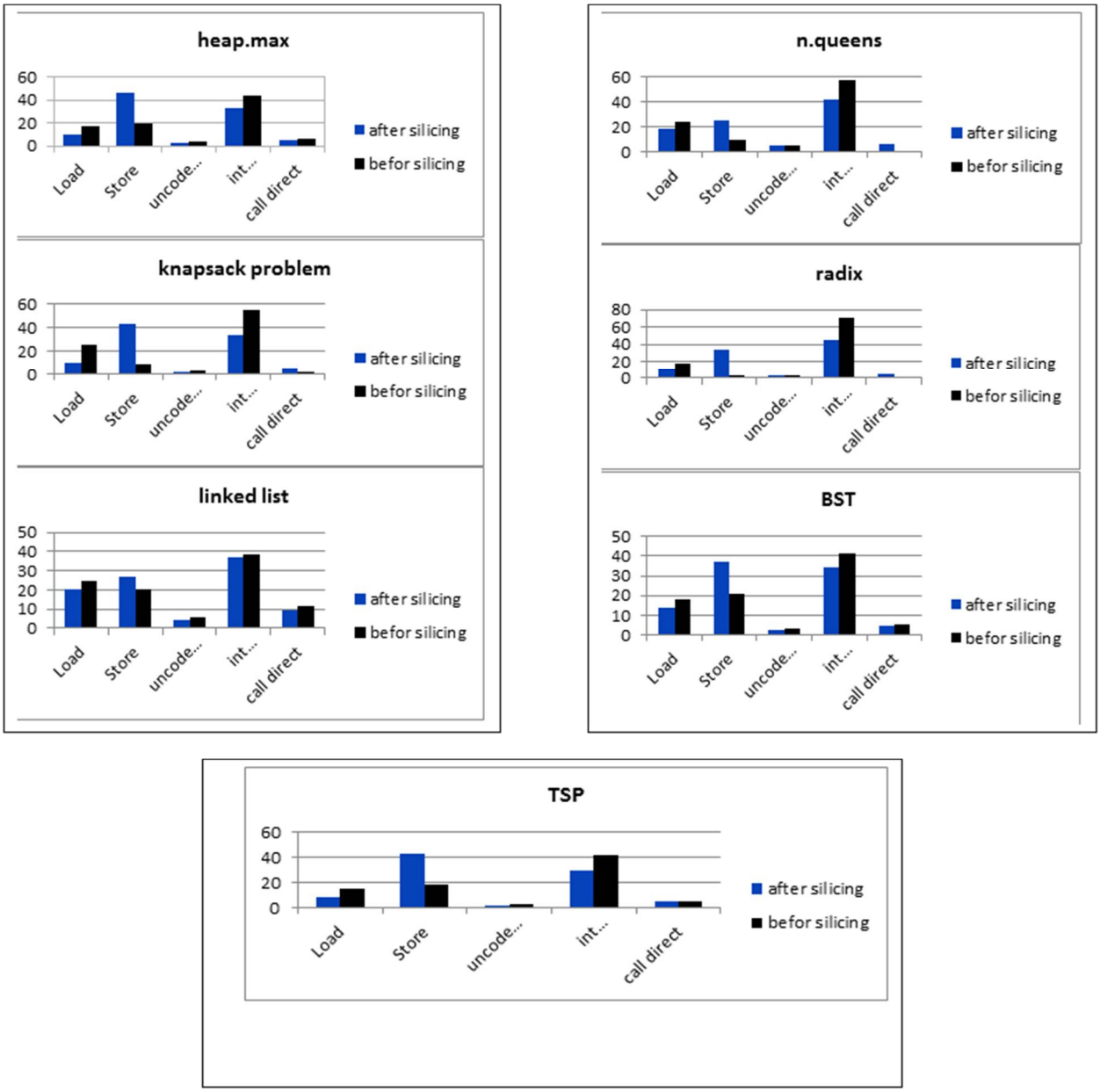

Fig. 1. The profiles of the tested programs (A, B, C)

4.3. Results

In this study, three sets of error injection experiments were conducted. In the first set of experiments, 10000 errors were injected on each of the tested programs and the respective results were obtained. In the second set, 40% of the selected instructions were randomly repeated. Indeed, 40% of the random instructions of the program were protected against the soft errors. Then, the changed programs were used in the error injection experiments. Like the procedure adopted in earlier studies, 10000 errors were injected in the code space of the program.

Table 4. Error detection coverage in the main programs and the slice programs

Error detection Coverage							
	TSP	Heap	Knap	List	N-Queen	BST	AVG
All the programs	86.3%	75.00%	89.20%	93.70%	71.00%	94.30%	84.47%
Sliced programs	83.10	73.00	78%	96.10%	87.12%	98.33%	85.96%

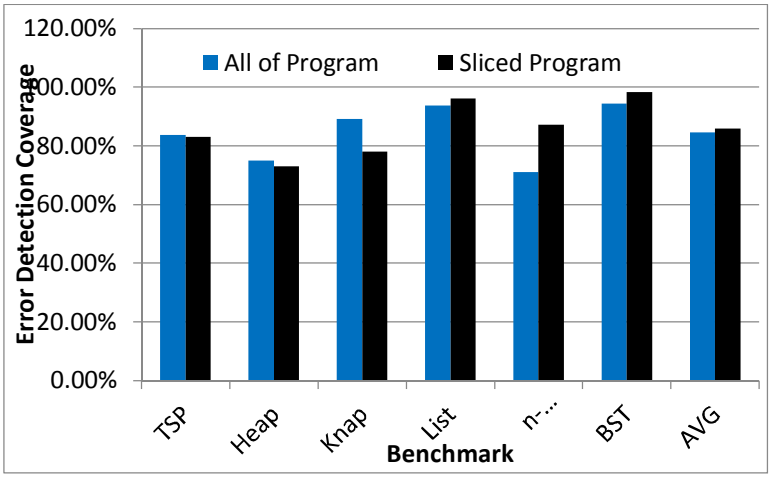

Fig.2. Error detection coverage in the main and sliced programs

As mentioned earlier, different kinds of error injection including jump instructions and changing the jump instructions were used in the present study to evaluate the proposed method. The error injection operations were conducted on six tested programs on which 5000 errors were injected. The memory and time of the manipulated programs were evaluated according to the proposed algorithm in the simulator. Then, depending on the number of jump instructions, the amount of memory consumption and the running time of the program were extracted. Then, the obtained results were acknowledged. Memory and time overhead resulting from the proposed method were measured. The following tables and figures depict the results of running the manipulated programs by means of the proposed method.

Table. 5. The results of the time overhead in the manipulated programs using slice

Time overhead							Full duplication overhead	The proposed method overhead
TSP	Heap	Knap	List	N-Queen	BST	AVG	>100%	~ 68%
71.00%	62.00%	76.00%	79%	61.00%	59.00%	68.00%		

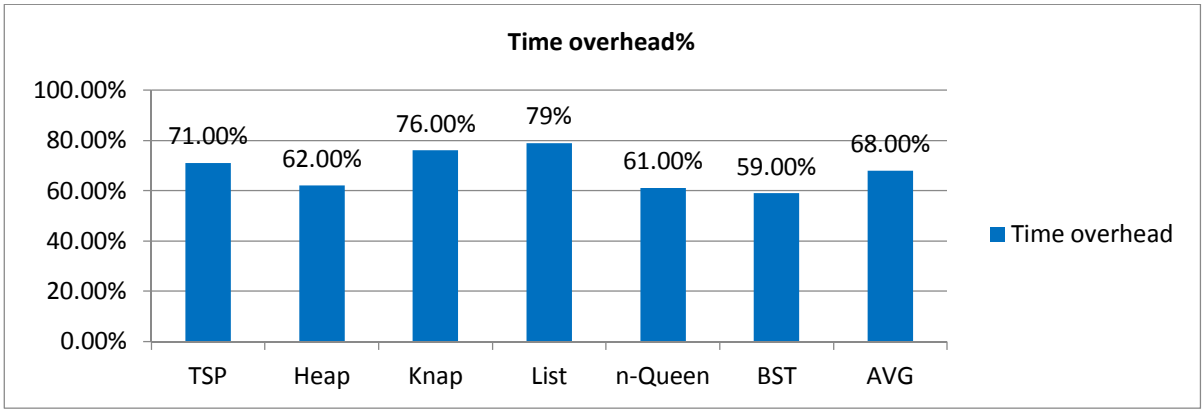

Fig.3. The results of time overhead in the manipulated programs

Table. 6. The results of the memory overhead in the manipulated programs using slice

Time overhead							Full duplication overhead	The proposed method overhead
TSP	Heap	Knap	List	N-Queen	BST	AVG	>100%	~ 66.50%
70.00%	60.00%	74.00%	77%	60.00%	58.00%	66.50%		

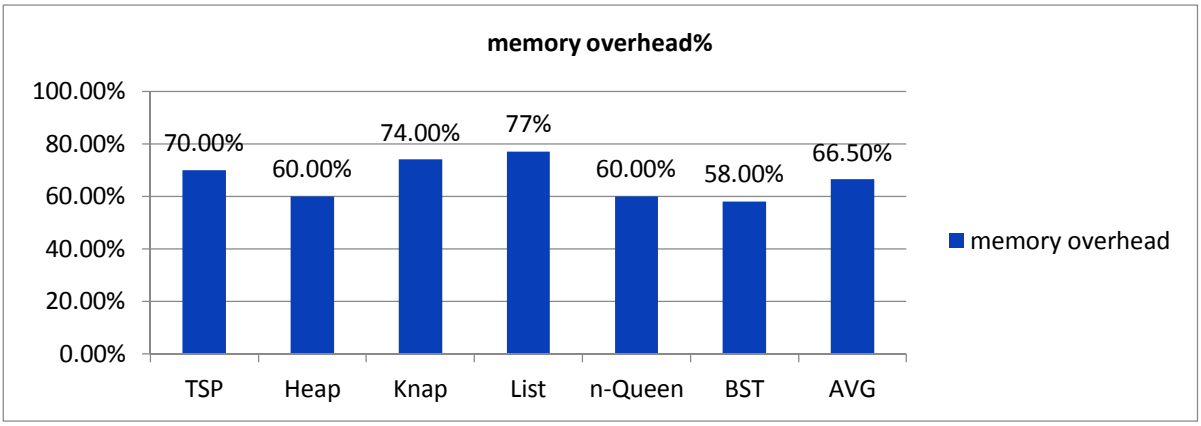

Fig.4. The results of memory overhead in the manipulated programs

5- Conclusion

The results from the present study indicated that the elimination of the instructions and data by means of the program slicing led to a reduction of the vulnerable instructions coding instructions. That is, slicing had a significant impact on their vulnerability. Moreover, the results of the study revealed that the proposed method had higher efficiency and less time and memory overhead. As discussed earlier in the paper, there are four types of program slicing; however, only the static slicing was used in this study. As a direction for further research, future researchers are recommended to investigate and examine the effectiveness of dynamic slicing, conditional slicing, and approximately static slicing on the efficiency and time and memory overhead of programs.

References

[1] O.Goloubeva, M. Rebaudengo, M. Sonze Reorda, and M. Violante, SOFTWARE-IMPLEMENTED HARDWARE FAULT TOLERANCE. 2006.

[2] Croll,P.and Nixon, P.” Developing Safety within a CASE Environment,” Proceedings of the TEEE Colloquium on Computer Aided Software Engineering Tools for Real-Time Control,1991,p.8.

[3] Ignat, N., Nicolescu, B., savaria, Y. and Nicolescu, G., ”Soft-Error Classification and Impact Analysis on Real-Time Operating System,” Proceedings of the Conference on Design, Automation and Test in Europe (DATE ’06), Germany, 2006, pp.182-187.

[4] Reis, G. Chang, J., Vachharajani N., Rangan R., August I., “SWIFT: Software Implemented Fault Tolerance”, Proceedings of the CGO’ 05,2005, pp.243-254.

[5] A ,aprie, j. c, Randell , B. and Lanwehr Avizienis ,”Basic Concepts and Taxonmy of Dependable and Secure Computing,” IEEE Transactions on Dependable and Secure Computing, vol.1.no.1,, 2004, pp.11-33.

[6] Shirvani ,P.P., oh,N., McCluskey, E.J., and Wood,D.L., “Software-Implemented Hardware Fault ataoleranceExperiments COTS in Space,” Proceedings of the International conference on Dependable Systems and Network, New york,NY, 2000, pp. 25-28.

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

[8] Rebaudengo, M., Sonza Reorda, M., Torchiano, M., and Violante, M., “Soft-error Detection Through SoftwareFault-Tolerance Techniques”, Proceedings of the IEEE International Symposium on Defect and Fault Tolerancein VLSI Systems, Albuquerque, NM, USA, Nov 1999, pp. 210-218.

[9] Randell, B., “System Structure for Software Fault Tolerant,” IEEE Transaction on Software Engineering, Vol. 1, No. 2, 1975, pp. 220-232.

[10] Avizienis A., “The N-Version Approach to Fault-Tolerant Software,” IEEE Transaction on Software Engineering, Vol. 11, No. 12, 1985, pp. 1491-1501

[11] Stefanidis, V. K., and Margarits, K. J., “Algorithm Based Fault Tolerance: Review and Study,” Proceedings of the2004 International Conference of Numerical Analysis and Applied Mathematics (ICNAAM;04), 2004, pp. 1-8.