Proposing an Efficient Software-Based Method for Enhancing the Reliability of Critical Application Robot

Reza solhi
Department of Electrical Engineering, Ahar Branch, Islamic Azad University, Ahar, Iran
Email:roboreza67@gmail.com

Abstract

Robots play such remarkable roles in humans' modern lives that performing many tasks without them is impossible. Using robotic systems is gradually increasing the tasks allocated to them and they are becoming more complex and critical. Software reliability is one of the most significant requirements of robots. For enhancing reliability, systems should be inherently designed to be tolerable of soft errors. In this study, via software, a method was proposed to enhance the reliability of the software embedded within robots against soft errors with minimum efficiency overhead. In as much as the research method was based on experiment, a set of programs was used as benchmarks. Indeed, errors were injected at the execution time of programs for evaluating them. The data related to the execution behavior of the programs were accumulated and then were analyzed. Simplescaller simulation software was used for investigating program behavior in the presence of the injected error. [2]

Keywords: soft error, inherent error toleration, software reliability, benchmark, error injection

1- Introduction

As a result of the development of robotic systems in industry and their application in human life, it is essential that the reliability of such systems in terms of hardware and software be investigated. As a case in point, consider the reliability of the operations of an ATM which is regarded as one of the obvious requirements of modern life. Using inherent error tolerance techniques, a robot, as an advanced machine, can detect and discover different types of errors. The software of a robot can use inherent error tolerant features to enhance the performance of the system.[3].Indeed, an inherent error tolerant system is a system which can perform its tasks properly even at the presence of software errors and hardware faults. Due to the ever-increasing use of robots in different aspects of modern human life, error tolerance is considered to be a significant issue. For enhancing the reliability of robotic systems in safety-critical applications, numerous software and hardware methods have been proposed. Using hardware and software redundancy is one of the typical methods for enhancing reliability. It should be noted that using hardware methods for enhancing the reliability of robotic systems can lead to cost increases, hardware changes and complexity. In contrast, software methods can enhance reliability and reduce hardware costs.

The main shortcoming of software methods used for enhancing the reliability of robotic systems are software complexity and efficiency overhead.

The method proposed in this study was aimed at solving the problems of hardware methods and other software methods.

The purpose of this study was to optimize software reliability by utilizing inherent features without using hardware and software redundancy. In this method, programming structures and memory classes of the program data were investigated. Indeed, it should be maintained that using programming structures with low error vulnerability can enhance the inherent reliability of the program. Redundancy was not used in the method proposed in this study. [4, 5]

2- Related works

In [21], the public tool for evaluating and detecting errors in programs was used. An executable claim is a command which examines whether specific conditions are maintained between the different variables of a program? Since these commands are not hidden from the programmer's view and their effect depend on the nature of program and the programmer's ability, they can result in several problems.

Procedure duplication method was introduced in [24]. In this method, the programmer decides to repeat the majority of the critical procedures and compares the obtained results with one another. Code is changed manually which leads to the production of error [22]. The automatic conversions method was introduced in [23] which is based on data production and redundancy. This method is done according to conversions at the level of source code. In this method, the first aim is realized by duplicating each variable and adding compatibility investigations after each reading action. However, conversions focusing on code repeat the commands of the program. Then, after the main operations are done and the program is repeated, the compatibility of the executed commands is investigated [6]. In [28], researchers proposed a method for enhancing the automatic reliability by revealing errors through statistical analysis. Time overhead of this method was more than 33%. In [30], another software-based method was proposed in 2000 which was aimed at enhancing the reliability of software for application programs; it had less than 20% time overhead. In [31], researchers conducted experiments on sensitive data in application software and managed to increase reliability with less than 20% time overhead.

A significant feature of software is that about % of soft errors at the level of software are covered without any impact on the output of the program. The results of different experiments in [22] revealed that different methods of implementing a program have significantly different impacts on the amount of covering inherent errors by the software. Hence, it can be maintained that by investigating the structures and different methods of implementing software, we can identify methods which can inherently cover more errors. Thus, the inherent reliability of a program is a function of the implementation method of it. As a case in point, by changing data structures, algorithm and the programming method, we can modify the inherent reliability of a program with regard to soft errors. In this paper, we focused on the programming methods and studied their

impact on the inherent reliability of the program [7].

3- Method

As argued above, different methods of programming, different data definitions, memory classes and the way of using data are of high significance in the amount of covering errors in a program and the inherent reliability of program. In this study, the researchers intended to enhance the inherent reliability of program by only following simple principles. It should be pointed out that redundancy was not used in this study. Along this purpose, we firstly investigated the degree of the impact of different implementation methods by carrying out an extensive set of experiments. The method of defining data and memory classes were examined. Indeed, the chief objective was to enhance the inherent reliability of program by properly defining data and memory classes. The experiments were conducted according to the following stages:

Stage one: a program was implemented as A benchmark on four different implementation methods. In each method, program data are defined based on a specific method by using a memory class. Each method describes the memory class. In the first implementation, program data were implemented by automatic memory class. In the second method, the global memory class was used for implementing program data. In the third method, static memory class was used. In the fourth method, register memory class was used for implementing program data. Indeed, five different programs were implemented by means of four implementation methods. A total of twenty programs were used as benchmarks in the experiments.

Stage two: the degree of the vulnerability of each memory class was investigated. In fact, by carrying out certain experiments, we examined the impact of each memory class on error coverage. In this way, we investigated the reliability of each program. It should be noted that the vulnerability of a program to error refers to the impact of an error on the output and behavior of the program. In a program with high vulnerability, errors are more likely to result in program failure. Hence, reducing program vulnerability to error can produce a condition where more percentage of errors are covered in the program. Consequently, it can be argued that a program with low vulnerability can have high reliability.

In this study, extensive experiments were conducted for determining the vulnerability of memory classes. It can be argued that in case a program uses more hardware blocks at the execution time which are vulnerable to error, it will have low reliability. For instance, registers and the queue of commands within the processor are highly vulnerable to soft errors. The executed commands were used as benchmarks. Five programs were used in the experimental environment and each program included about 45% of the errors. This set of experiments was defined from distributed programs for vulnerable structure. In each of these programs, memory classes were designed via four methods and each of them was produced by different behavior of the parameters. These memory classes were designed and implemented by vulnerable

classes. One of them is the memory class for defining variable locally, globally, statically and register. Variable definition is compared in different methods for reducing vulnerable structures in each program with different parameters. The structure with the lowest vulnerability can be used as the benchmark. The first method was defined as the memory class; the memory class of variables was automatically allocated to the program function. As the program exits the memory filed, the respective variable is automatically freed and its space is given back to the system. In this class, for the sake of comparison, the intended parameters are used. In the global memory class which is used in all functions, by comparing this memory class with other memory classes, parameters with different values are produced. Then, the produced commands in each program are selected. After that, the commands are compared with one another and their degrees of vulnerability are compared with one another. [8, 9]

Static memory class is defined in an accessible function and is used in the same function. In this program, this memory class was used without taking initial values. This type of class is defined for local memory class. Moreover, register memory class is located within CPU registers and the speed of doing operations is high which results in the enhancement of the operation of program speed. Also, this memory class is implemented for local variables. For comparing these programs, vulnerable structure at the program level is reduced with the lowest amount of memory class. The behavior of each program with different memory classes is different in the simple-scalar environment. In this study, the designed commands were compared with each type of memory class at the program output and diagrams were created for the produced commands [23].

3.1. Benchmarks

The programs which were introduced as benchmarks included the programs which were written in C programming language. These programs were designed for benchmarking in the simple-scalar environment with specific commands. Desirable output in these programs included five benchmarks which are described in the following table.

Table .1. The specification of the benchmarks used in the present study

Benchmark	Specification	Input variables	
		description	range
Bubble sort	Bubble sorting of a list of numbers	Bubble sorting	N=100
Factorial	Factorial calculation of a number	Factorial of a number	N=100
Tower of Hanoi	Doing tower of Hanoi game	Calculating Hanoi Tower	N=50
Fibonacci	Calculating Fibonacci series	The sum of two numbers produces the next number	N=100
Matrix	Matrix multiplication	The size of each row and column of each element	N=10

3.2. Experimental instrument

The experiments were carried out in the Simple-Scalar software for analyzing the software reliability of the benchmarks. Simple-Scalar is considered to be a functional

and practical instrument for simulating programs. It was designed and established by Todd Austin in 1994 which was expanded later. Simple-Scalar consists of a set of robust computer systems for simulation. Indeed, Simple-Scalar software can simulate functional programs with significantly high precision and efficiency which can be used for investigating the systemic structure and architecture of modern processors.

In the present study, all the programs were compiled in the Simple-Scalar software with the benchmark of GCC. In other words, it can be maintained that this software was used for evaluating the performance of the proposed method. The experiments were conducted according to the following steps:

- At first, error-reduction blocks of each program were detected by the proposed method.
- Secondly, software-based errors were injected into the programs via error injection process. Then, the programs were compiled in the simple-scalar instrument by GCC.
- The programs were executed in the simple-scalar instrument in the simoutorder environment.
- Finally, the results of simulations were obtained and analyzed.[10]

At the execution time, the vulnerability of a program is stated based on the machine commands and commands. It should be noted that one of the main factors which reduces the reliability of a software is its complexity. By producing different software algorithms, the reliability of a program can be enhanced.

4- Results

The results of experiments indicate the impact of memory class of data on the vulnerability of commands. It should be noted that the proposed method was investigated with regard to program vulnerability in terms of soft error. Figure 1 illustrates load command in different programs at the execution time.

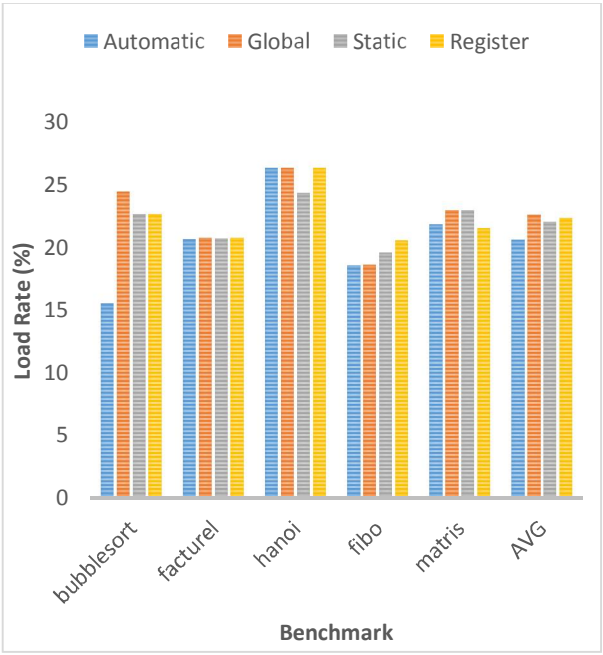

Fig. 1. The rate of load commands in different programs implemented with different memory classes

The investigation of behaviors of different executed programs in the simprofile environment revealed that the programs implemented with automatic and static memory classes have the lowest degrees of load commands, respectively. Figure 2 depicts the amount of store command in different programs at the execution time.

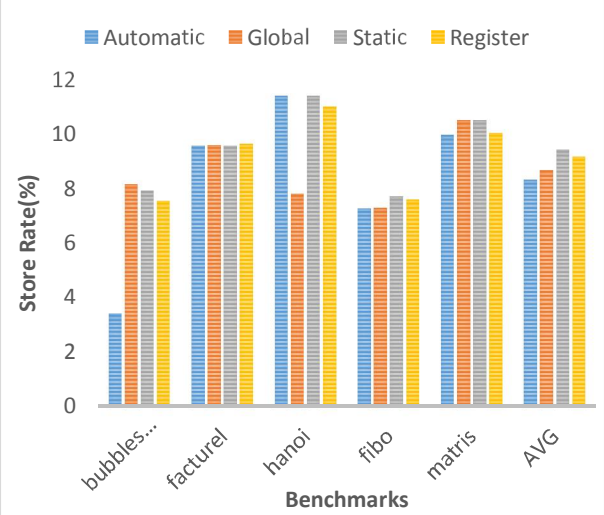

Fig.2. Store command rate in different implemented programs with different memory classes

Investigating the behavior of the implemented store command in the simprofile indicated that the degree of vulnerability of implemented programs to soft error in local and global memory classes were minimal. Hence, as the degree of vulnerability decreases, the reliability of the program to soft errors increases.

Different programs in the uncod branch command for implementation are illustrated in figure 3. As mentioned above, the lower the degree of vulnerability to soft error, the higher the reliability of implemented programs in automatic and static memory classes which have the lowest data, respectively. For executing programs in these two parameters, provided that memory amount is low, the vulnerability decreases and reliability increases.

Different programs in simprofile environment with cond branch were simulated for testing the reduction of vulnerability to soft errors.

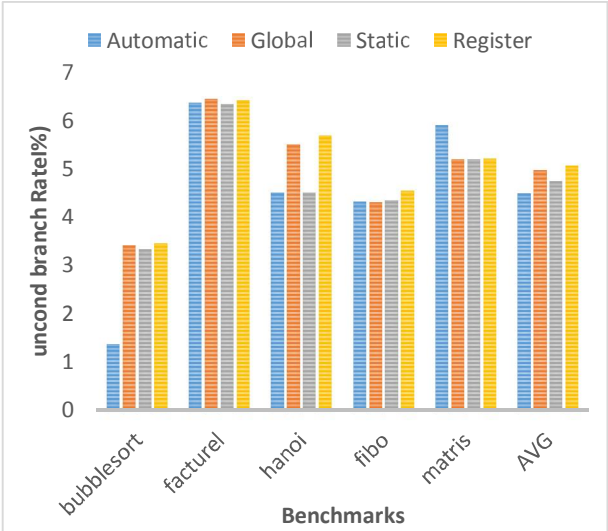

Fig. 3. Rate of uncond branch rate in different implemented programs with different memory classes

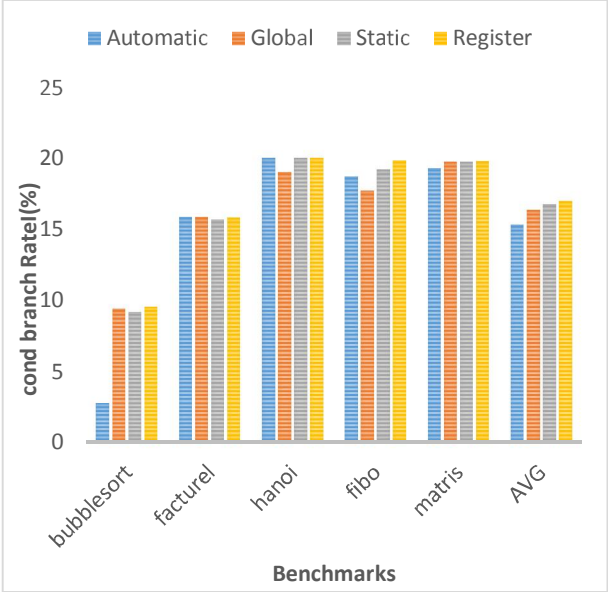

Fig.4. Cond branch command in different implemented programs with different memory classes

The vulnerability of the program in local and global memory classes was lower than other classes. Hence, it can be argued that the degree of vulnerability to soft errors was reduced and, consequently, reliability was enhanced.

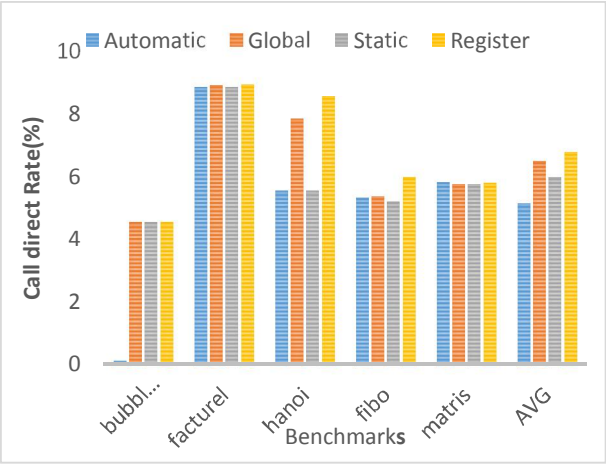

Fig.5. The rate of call diet in different programs implemented with different memory classes

The comparison of figures 4 and 5 reveals that simprofile command can reduce vulnerability of the program to soft errors more than call direct in the case of automatic and static memory classes. As reliability of a program decreases, its vulnerability to soft errors increases. In other words, high rate of call direct resulted in more vulnerability to soft errors. The following figure depicts call indirect command in different programs at the execution time.

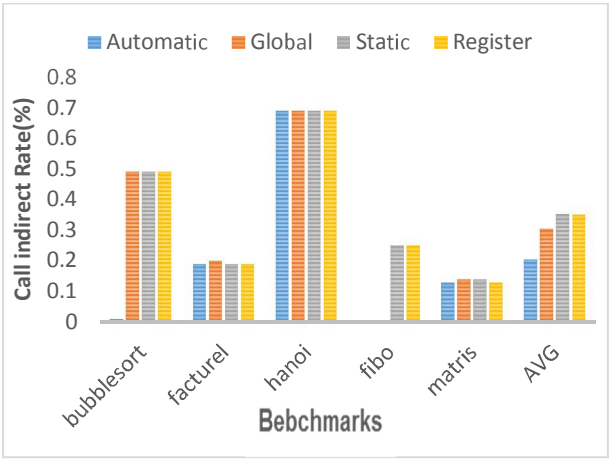

Fig.6. The rate of call indirect in different programs implemented by different memory classes

This figure shows the output of the programs. The value of parameter in these programs is low which means that reliability has increased and vulnerability has decreased. As shown in this figure, other memory classes increase vulnerability to soft errors; hence, reliability decreases. Also, the following figure depicts heap segment command in different programs.

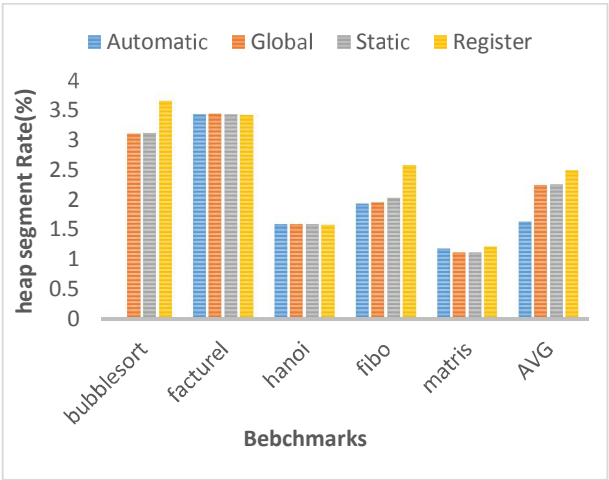

Fig.7. The rate of heap segment rate in different programs implemented by different memory classes

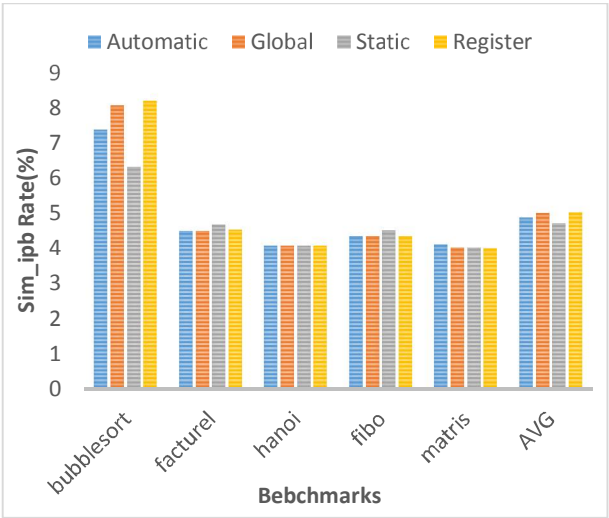

Fig.8. Sim_ipb rate in different programs implemented by different memory classes

In general, the results of the conducted experiments indicate that using the above-mentioned memory classes reduces the degree of vulnerability to soft errors. Consequently, reliability is inherently enhanced. This figure reveals that automatic and static memory classes with different parameters in sim_ipb command reduces vulnerability and inherently enhances reliability better.

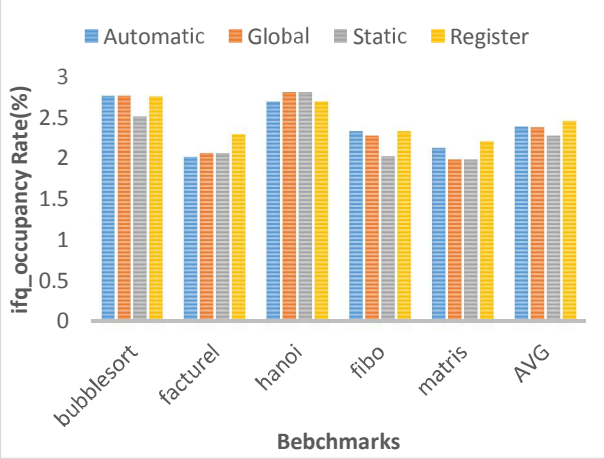

Fig.9. The rate of ifq-occupancy command in different programs implemented by different memory classes

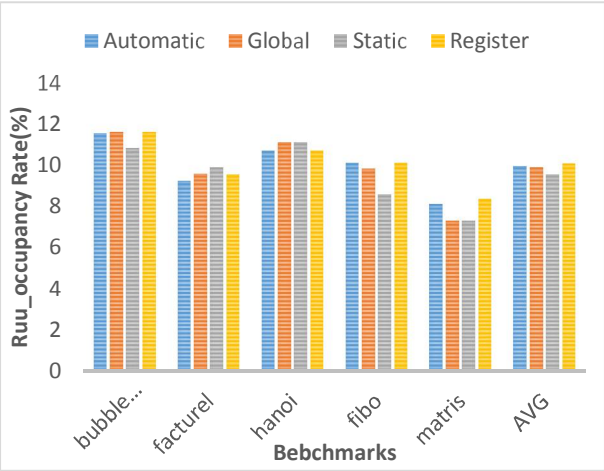

Fig.10. The rate of Ruu_occupancy in different programs implemented by different memory classes

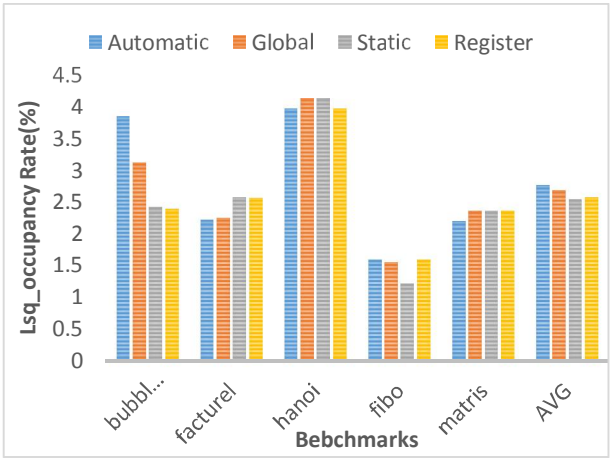

Fig.11. The rate of Lsq_occupancy in different programs implemented by different memory classes

In sum, the obtained results indicated that using the automatic memory class can inherently reduce vulnerability more than other memory classes. As a result, reliability is inherently enhanced. In contrast, global memory class has more vulnerability; hence, more use of this class in memory leads to vulnerability enhancement which results in the inherent reduction of reliability.

5- Discussion and Conclusion

In this study, different experiments and simulations were carried out for the four different versions of benchmarks. The differences among benchmark versions were related to the memory class applied for the data. The present study analyzed the impact of different memory classes on vulnerability to soft errors. Indeed, the researchers found that the programs implemented with the automatic memory class have less vulnerability and the programs implemented with register memory class and global memory class have more vulnerability. Thus,

it can be maintained that the programs executed with register and global memory classes have less inherent reliability. It should be noticed that defining memory classes regardless of reliability redundancy can enhance program reliability against soft errors.

As a direction for further research, the impact of different compilers on vulnerable commands and reliability can be investigated by interested researchers in future. Furthermore, future studies can investigate the effect of different implementation methods, designing and software architecture on the rate of vulnerable commands and program reliability.

References

[1] Baumann, R., "Soft Errors in Commercial Semiconductor Technology: Overview and Scaling Trends," Proceedings of the IEEE Reliability Physics Tutorial Notes, Reliability Fundamentals, 2002, pp. 121-01.1–121-04.

[2] Shirvani, P.P., Oh, N., McCluskey, E.J., and Wood,D.L.,"SoftwareImplemented Hardware Fault Tolerance Experiments COTS in Space," Proceedings of the International Conference on Dependable Systems and Network, New York, NY, 2000, 25-28.

[3] 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.

[4] Zhu and H. Aydin, Reliability Effects of Process and Thread Redundancy on Chip Multiprocessors,In Proceeding of the 36th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, 2006.

[5] P.Kongetira, K.Aingaran and K.Olukotun, Niagara: A 32-Way Multithreaded Sparc Processor, IEEE Micro, 2005.

[6] Baumann, R., "Soft Errors in CommercialSemiconductorTechnology:Overview and Scaling Trends," Proceedings of the IEEE Reliability Physics Tutorial Notes, Reliability Fundamentals, 2002, pp. 121-01.1–121-04.

[7] J.H.and Harper, R.E Lala,"Architectural principles for safety-critical real-tim,"in proceedings of the IEEE,82(1),1994,PP.25-40.

[8] 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.

[9] Rebaudengo, M., SonzaReorda, M. and Violante, M., "A Source-to-Source Compiler for Generating Dependable Software," Proceedings of the First IEEE International Workshop on Source Code Analysis and Manipulation, Florence, Italy, 2001, pp. 33-42.

[10] G. Miremadi, J. Karlsson, U. Gunneflo, J. Torin, "Two SoftwareTechniquesfor Online Error Detection", the Twenty-Second International Symposium on Fault-Tolerant Computing, July 1992, pp. 328 – 335.

[11] Araste,b., Rahmani,a., Mansoor,a., Miremadi,gh,." Using Genetic Algorithm to Identify Soft-Error Derating Blocks of an Application Program",EuromicroConference on Digital System Design, 2012. 15.

[12] Lisboa, C.A.L,Carro,L., Reorda, M., Violante, M., "OnlineHardeningofPrograms Against SEUs and SETs," Proceedings of the 21stInternational Symposium on Defect and Fault Tolerance in VLSI Systems, 2006.

[13] Jing, Y., Garzaran, M. J., and Snir, M., "Efficient Software Checking for Fault Tolerance," Proceedings of the IEEE International Symposium on Parallel and Distributed Processing,Miami,FL,April 14-18,2008, pp.1-5.

[14] Rebaudengo, M., SonzaReorda, M., Torchiano, M., and Violante, M., "Soft-error Detection Through Software Fault-Tolerance Techniques", Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Albuquerque, NM, USA, Nov 1999, pp. 210-218.

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

[16] Alkhalifa, Z., Nair, V.S.S., Krishnamurthy, N. and Abraham, J. A., “Design and Evaluation of System-Level Checks for on-Line Control Flow Error Detection,” IEEE Transactionon Parallel and Distributed Systems, Vol. 10, No.6, 1999, pp. 627-641.

[17] N. Oh, P.P. Shirvani,E.J. McCluskey,"Error Detection by Duplicated Instructions, In Super-scalar Processors", IEEE Transactions on Reliability, Vol. 51, No. 1, 2002, pp. 63-75.

[18] J.-S. Lu, F. Li, V. Degalahal, M. Kandemir, N. Vijaykrishnan, M.J. Irwin, "Compiler-directed instruction duplication for soft error detection". Proceedings of Design, Automation and Test in Europe, 2005, pp. 1056-1057.

[19] Oh, N., Shirvani, P. P. and McCluskey, E. J., “Control- Flow Checking by Software Signatures,” IEEE Transactions on Reliability, Vol. 51, No. 1, 2002, pp. 111-122.

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

[21] Li, Aiguo , and bingrong Hong. “Software Implemented Transient Fault Detection in Space computer.” Aerospace science and technology 11.2-3(2007):245-255