Journal of Advances in Computer Research
Quarterly pISSN: 2345-606x eISSN: 2345-6078
Sari Branch, Islamic Azad University, Sari, I.R.Iran
(Vol. 8, No. 2, May 2017), Pages: 1-11
www.jacr.iausari.ac.ir

A New Control Flow Checking Method to Improve Reliability of Embedded Systems

Hamed Nikookar✉[1] and Ahmad Patooghy[2]
1) Department of Computer Engineering, Boroujerd Science and Research Branch, Islamic Azad University, Boroujerd, Iran
2) Department of Computer Engineering, Boroujerd Branch, Islamic Azad University, Boroujerd, Iran
hamednik4@gmail.com; patooghy@gmail.com

Received: 2016/06/14; Accepted: 2016/08/10

Abstract

Due to employ of embedded systems for safety-critical applications and high probability of occurrence transient faults in them as well as increasing popularity of COTS (commercial off the shelf) components, techniques of control flow checking to improve reliability processors are of particular importance. Among all the problems of software-based technique for control flow error detection can be pointed to the performance overhead because of software redundancy and lack of proper solution for detecting Intra-block control-flow jump errors. In this article we have proposed a generic software-based technique for control flow error detection that can add instructions redundant on a basic block to detect a large number of errors as well as reduced overhead by identifying S-NODE in control flow graph and placed check instruction in these nodes. Overall, combining CFCSS (Control Flow Checking by Software Signatures) with our proposed technique has an average of 96% fault coverage in comparison to 92% fault coverage of previously proposed signature based techniques while maintaining the performance overhead has nearly SCFC.

Keywords: *COTS (Commercial-Off-The-Shelf), Control-Flow Checking, Embedded Systems, Transient Fault*

1. Introduction

Embedded systems that are employed in critical applications must be reliable because of occurrence faults in these systems causing serious damage [1, 2]. Utilizing robust equipment in embedded systems makes these systems handle faults but COTS hardware because of low cost and other characteristics has become more popular in this system. Using COTS components makes this system are faced with serious risks. In between all the faults that threaten these systems, transient Fault than all more serious. These errors are caused by temporary conditions on the chip or by external noise, causing disturb in the program control flow or data corruption. Previous studies have reported that as much as 70% of the transient faults disturb program control flow [3]. Among all the fault tolerant methods control flow checking has the best error detection over the others. For detecting transient faults and control flow checking errors, some techniques are presented that can be categorized in two general classes of hardware [4] and software [5-7] redundancy. Hardware based methods for CFC[1] inserting redundant hardware

[1] control flow checking

against software CFC can be implemented by adding only a few instructions per basic block. Because software based methods is cost free in this article we use this method to control flow checking. Most software-based techniques for control flow error detection are based on signature. In these techniques for control flow error detection, the source code is divided into several basic blocks and specifies a signature for each block. This technique Control flow checking involves two steps in each BB: 1- Compute signature at runtime 2- Compare with an expected correct value. Also these techniques there are two critical aspects: 1- performance overhead because of software redundancy 2- lack of proper solution for detecting Intra-block control-flow jump errors (Methods with high instruction overhead or low fault coverage). In this article, we have proposed a generic method for software-based control flow checking for soft error detection to improve microprocessor reliability. Our proposed technique has high fault coverage in comparison of previously proposed signature based techniques while maintaining the performance overhead has nearly SCFC. The steps in this method, as following:

• program is divided into basic blocks and control flow graph is constructed then specifies a signature for each block

•R (a unique number) will be assigned to some basic blocks

•Identifying S-NODE in control flow graph and placed check instruction in these nodes

2. Literature Review

Over the years, many Signature based methods have been proposed for protection of systems against Transient Fault. In these methods, the first program code is divided into several basic blocks and specifies a signature for each block. Two types of control flow errors that should be analyzed in this method: 1- Intra-block control-flow jump errors.

Inter two blocks control-flow jump errors. Most of these methods to detect one of the two types or if both detect they have low fault coverage. For example, as shown in Figure 1, the software-based CFC technique, CFCSS [5], for each BB assigned a unique number as the name of the compilation-time assigned reference signature (S) and three additional checking instructions are inserted. G is the run-time signature and a global register holds the run-time signature G. $d_j = S_s \oplus S_d$, d_j which is calculated during the course of pre-compilation, is the signature difference between the source node signature and the destination node signature. D is the justifying signature that is used in the branch-fan-in node. Intra-block control-flow jump errors cannot be checked by the CFCSS.

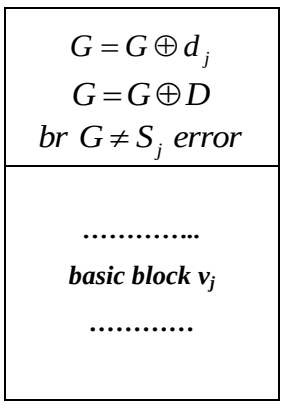

Figure 1: The basic block is equipped by cfcss.

In RSCFC [6], SCFC [7] and I2BCFC [8] Intra-/Inter-block control-flow jump errors can be checked. RSCFC imposes a high overhead to the system and a few of the Intra-block control-flow jump errors detect by SCFC and I2BCFC. For example, as shown in Figure 2, SCFC and I2BCFC, if occur a mistake jump in the upper or lower half of the block, the error will not be detected. Also if due to a fault, instruction of each block branches directly to update instruction ($s=s_i$), the execution skips check instruction (Error=S [ID] or err=S[sel] ^1) in SCFC and I2BCFC, and the code will not detect the fault.

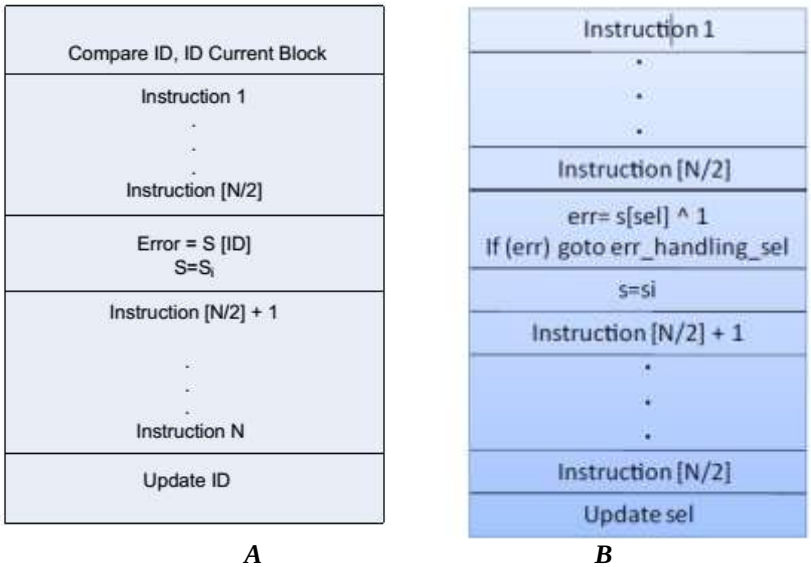

Figure 2: A) The basic block is equipped by scfc[7] and B) The basic block is equipped by I2BCFC[8].

To solve these problems we have a solution in this article. The idea of this solution is very generic and can be combining with different control flow checking method. Among the methods that have been proposed CFCSS is one of the most prospective angles [9- 11]. For this reason, the proposed method in this article is combined with CFCSS. As mentioned above there are two cases that cannot be managed by traditional CFCSS algorithm performance overhead and Intra-block control-flow jump errors which are the research focuses of this article.

3. The Proposed Method

The proposed method in this article is based on software. In this method the program is divided into basic blocks and control flow graph is constructed based on the CF[1] of the program. A node in the control flow graph represents a basic blocks. This method assigns a unique signature(S) to each node of the control flow graph and inserts redundant instructions for each block. Intra-block control-flow jump errors will be checked by this method.To check the Intra-block control-flow jump errors are assigned a number to each basic block. The number in each basic block represents the total number of instructions in each basic block. Maximum number among all the basic blocks called N. To basic blocks that number is nearly to N, R is assigned. R is a unique

[1] Control flow

number for these blocks. In this blocks R twice is EXORed with run-time Signature (Once before update Instruction and once after it in these basic blocks).

A basic overview of the implemented scheme is shown in Figure 3. G is run-time Signature and Si is compilation-time assigned reference signature. For control flow checking in these basic blocks that has n instructions, four instructions are defined. Two instructions are placed at ⌊n/4⌋ and 3⌊n/4⌋ of these basic blocks; R is EXORed with run-time Signature (G) that Eq. (1) are shown.

$$G = G \oplus R \tag{1}$$

Instructions of run-time signature updates are placed at the middle of these basic blocks; Eq. (2) is shown.

$$G\ Update \tag{2}$$

Check Instruction is placed at the end of some basic blocks; Eq. (3) is shown.

$$br\ G \neq S_i\ error \tag{3}$$

For the remaining blocks only Eq. (2) for each basic blocks and Eq. (3) for some basic blocks is added.

The proposed method in this article has solved the problems previous technique (I2BCFC and SCFC) and has created a generic software-based method. As can be seen in Figure 3, if the fault skips each of the redundant instructions, in the next basic block with execution of check instructions will detect the fault.

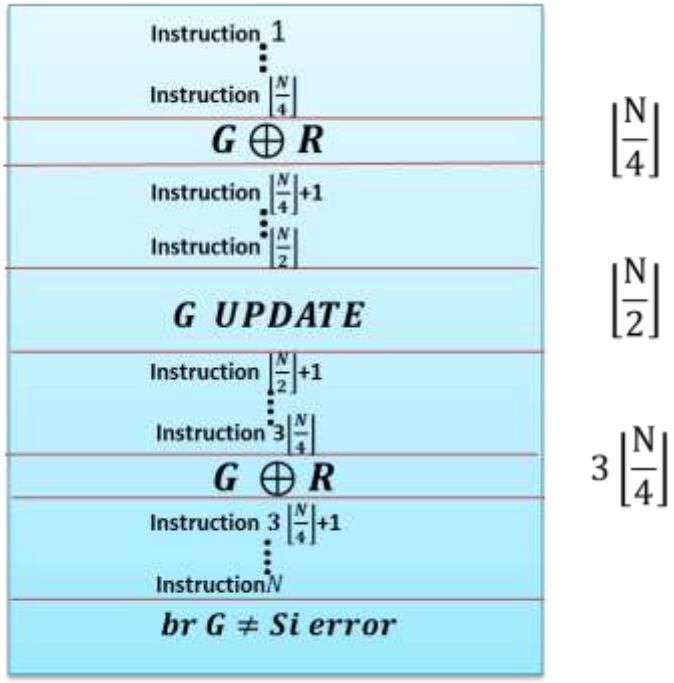

Figure 3: The basic block is equipped by proposed method.

To reduce overhead, Check Instruction is placed at the end of some blocks. To reduce performance overhead, after creating the program control flow graph sensitive nodes are identified. The nodes according to the algorithm [12] include nodes which are used more than once as a head for edges (junction nodes), nodes which are used more than

once as a tail for edges (decision nodes), the entry node, and the exit node. *Si* would only be checked in these nodes. Figure 4, shows s-node for a control flow graph.

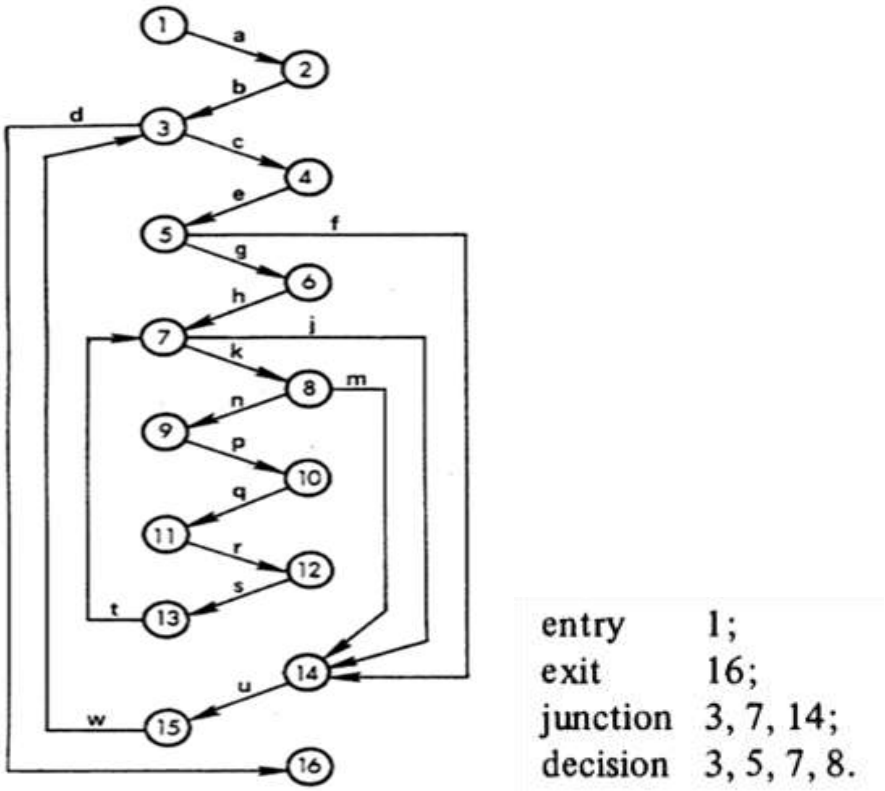

Figure 4: This Figure shows s-node for a control flow graph [11].

3.1 Capability of CFE[1] Detection

The proposed method in this article has solved the problems previous techniques (I2BCFC and SCFC) and has created a generic software-based method. As an example, a Part of a control flow graph that its basic blocks have been equipped by I2BCFC and SCFC is shown in Figure 5. As can be seen in Figure 5, if each of CFE's occurs, code will not be able to detect fault. In I2BCFC and SCFC, a basic block is divided into two parts and instructions redundant for control flow checking is inserted at the end or beginning of these parts. So if the total number of instructions in each of parts is high, fault occurrence is more probable. As mentioned, a basic block with high total number of instructions in our approach divided into four parts. So reducing the number of instructions in each parts makes the probability of errors lead to failure is lower.

[1] Control flow error

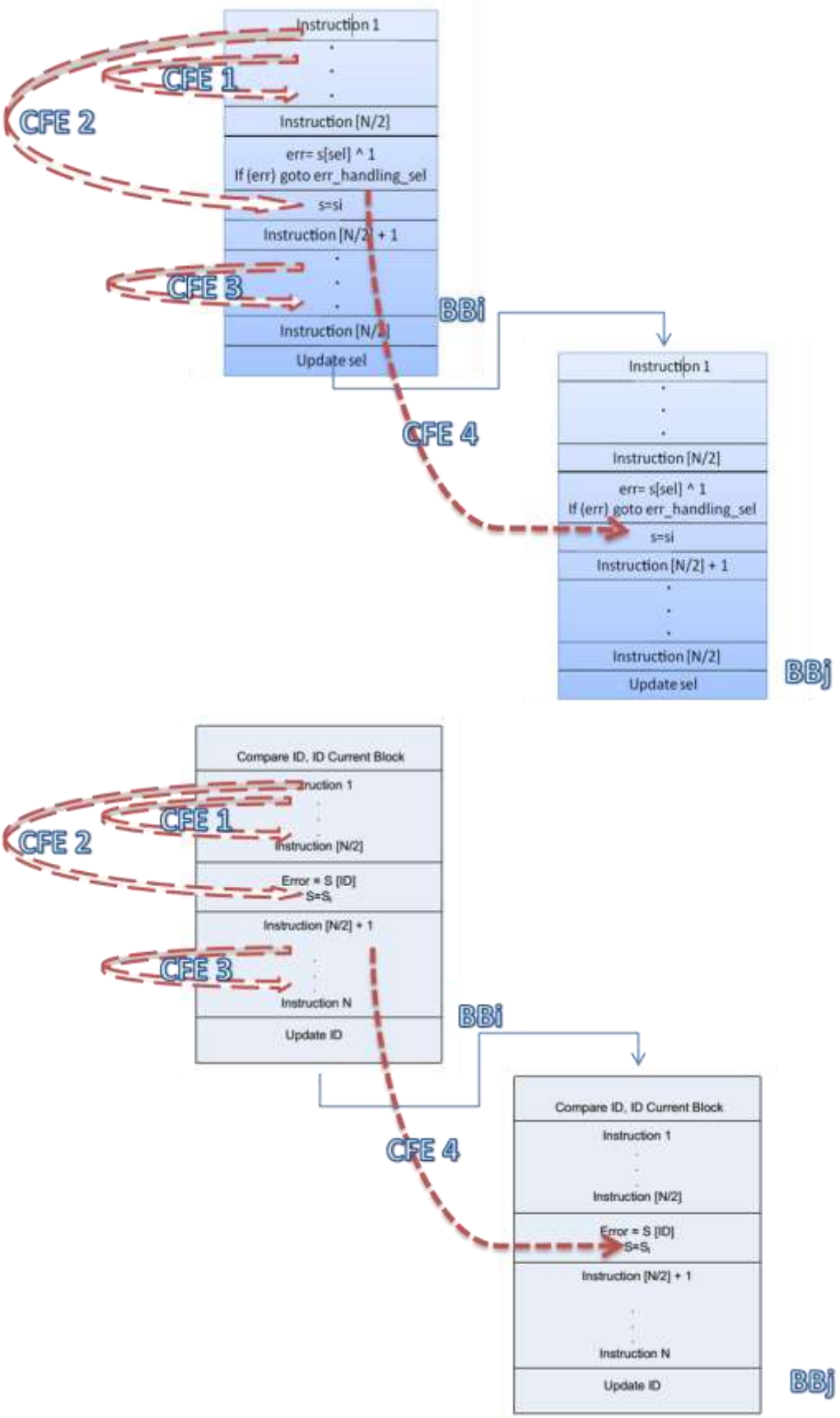

Figure 5: A Part of a control flow graph

3.2 Performance Overhead

The proposed technique achieves Proper overhead by performing checking at a few blocks. As mentioned, there are two types of basic blocks in our approach. Type1: a basic block with high total number of instructions. Type2: a basic block with low total number of instructions. Blocks of a program may be in one of four modes.

1. *if BB* $\in$ *Type*1 $\rightarrow$ *Count of instructions redundant=4*
2. *if BB* $\in$ *Type*1 *AND BB* $\in$ *S_NODEs* $\rightarrow$ *Count of instructions redundant=5*
3. *if BB* $\in$ *Type*2 $\rightarrow$ *Count of instructions redundant=2*
4. *if BB* $\in$ *Type*2 *AND BB* $\in$ *S_NODEs* $\rightarrow$ *Count of instructions redundant=3*

In a program because number of blocks of case 3 is the most, resulting in lower total overhead. Figure 6 shows case 2 of bb which is equipped by the combination of our proposed method with CFCSS.

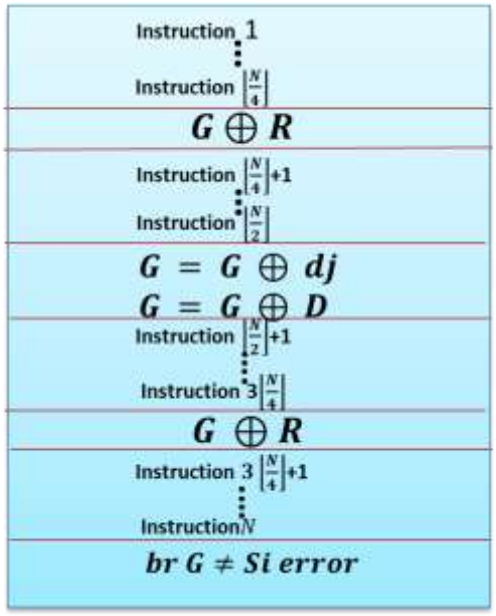

Figure 6: A bb is equipped by the combination of our proposed method with CFCSS.

4. Simulation Results

In order to evaluate the proposed method, three benchmarks Quick Sort (QS), Matrix Multiplication (MM), Bubble Sort (BS) is used. First, the source files were compiled and assembly codes were generated and a total of 500 transient faults have been injected into several executable points of each program. Branch deletion, branch insertion and branch target modification used as considered fault models. For the second part of the experiment, combination of proposed method with CFCSS is included in the assembly source code and a total of 500 transient faults are injected into the code. Table 1 and the graph in Fig. 7 illustrate percentage of faults that are detected in both the original program and the program with combination of proposed method with CFCSS.

Table 1 - Percentage of faults not detected

Code modulation	Original Code	proposed method+ CFCSS
Quick Sort	33.60	2.20
Matrix Multiplication	38.40	4.20
Bubble Sort	32.80	5.80

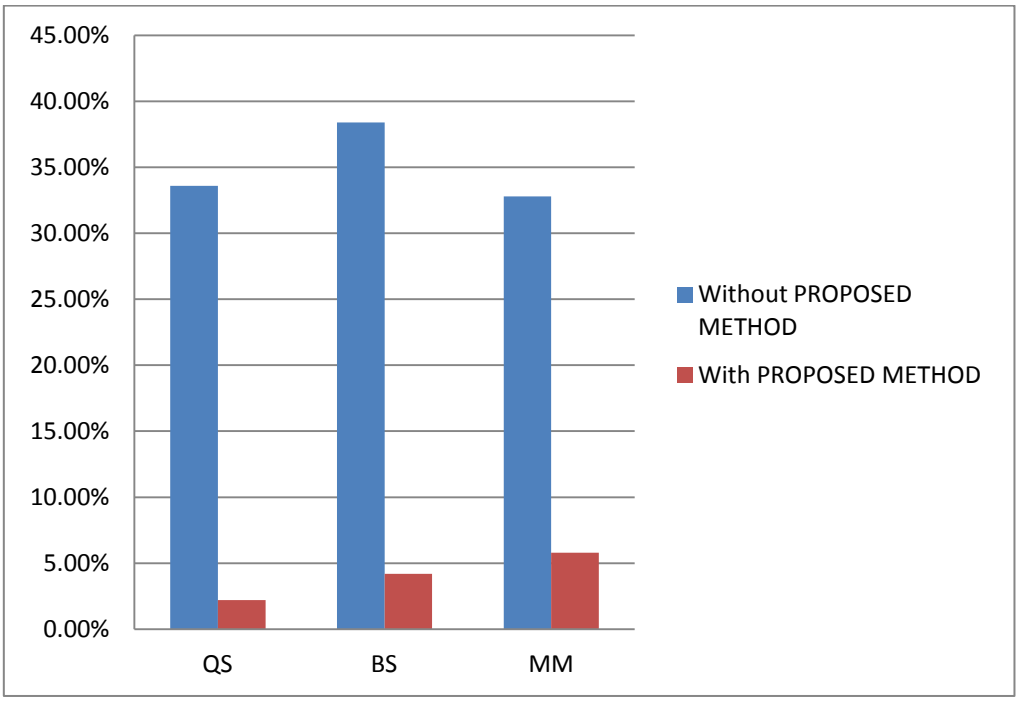

Figure 7: Percentage of faults not detected

Table 2 shows the fault coverage of the combination of our proposed method with CFCSS and other methods in selected sample programs. As it can be seen in Fig. 8, our proposed technique in comparison with CFCSS, RSCFC, and SCFC methods increases the fault coverage.

Table 2 - Total fault coverage comparison

Code modulation	CFCSS	RSFCF	SCFC	proposed method+ CFCSS
Quick Sort	96.88	93.4	96.7	97.8
Matrix Multiplication	86.86	90.3	87	94.2
Bubble Sort	92.8	93.24	95	95.8

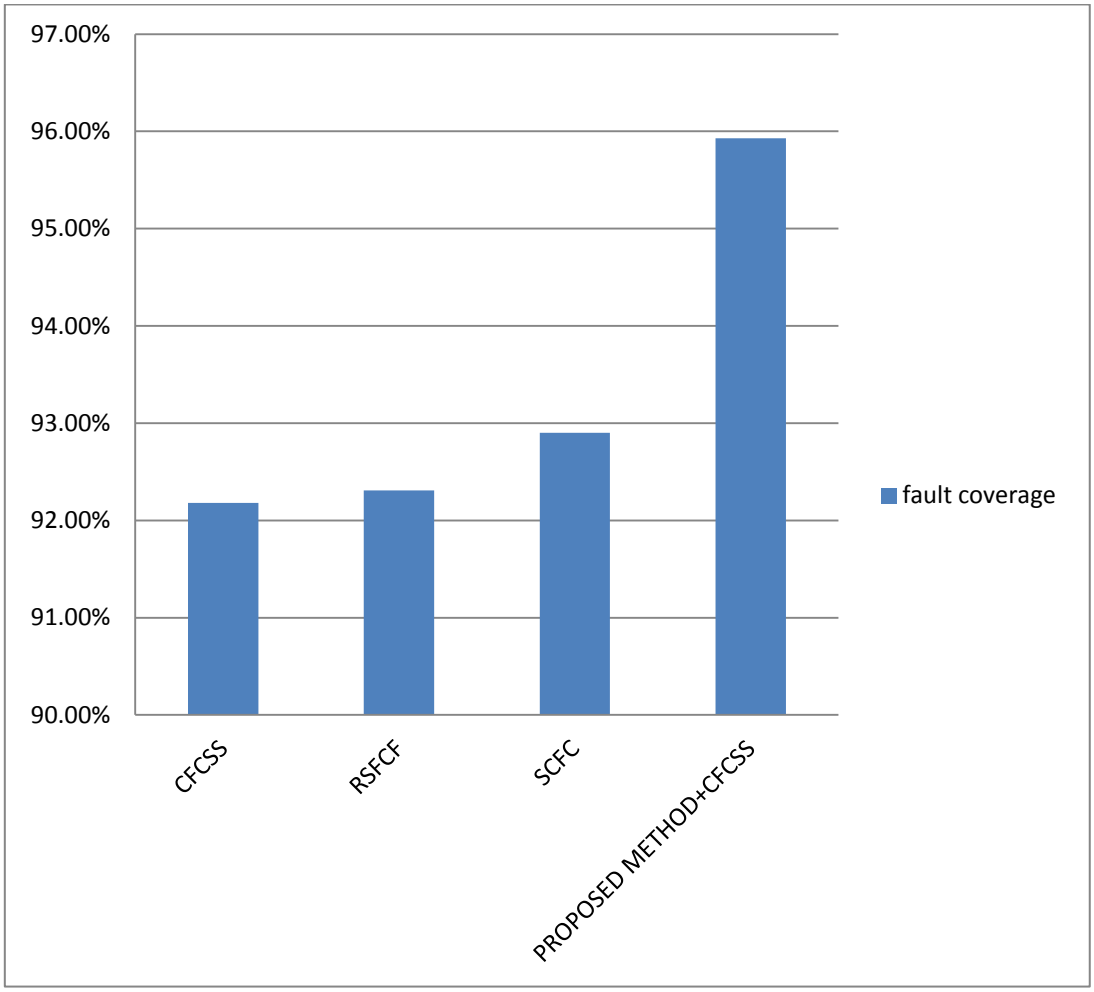

Figure 8: Total fault coverage comparison

Table 3 shows the performance overhead of the combination of our proposed method with CFCSS and other methods in selected sample programs. As it can be seen in Fig. 9, our proposed technique in comparison with CFCSS and RSCFC methods decreases the performance overhead.

Table 3 - performance overhead comparison

Code modulation	CFCSS	RSFCF	SCFC	proposed method+ CFCSS
Quick Sort	1.59	1.91	1.53	1.52
Matrix Multiplication	1.21	1.53	1.30	1.36
Bubble Sort	1.48	1.42	1.26	1.24

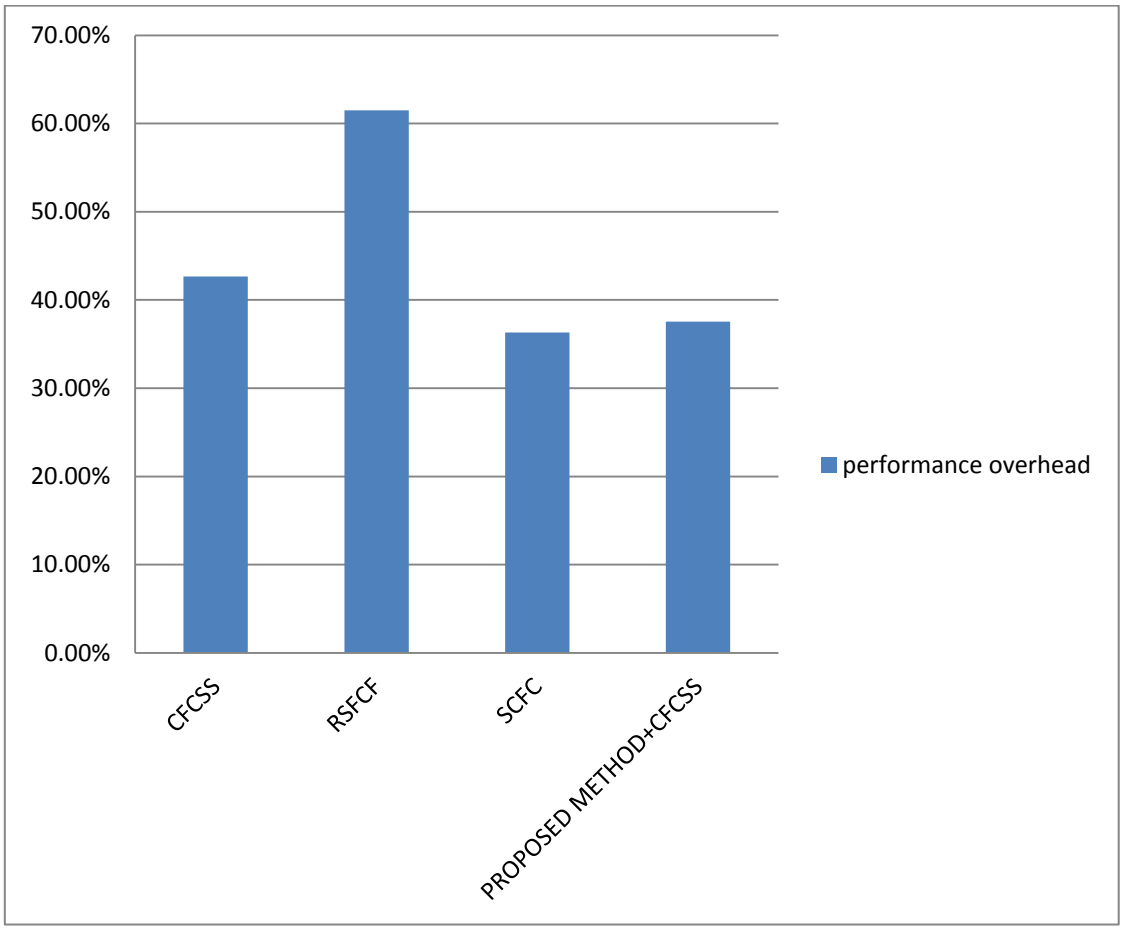

Figure 9: Performance overhead comparison

Figure 8 and 9 show that the proposed method not only has a better fault coverage, but also delivers a less performance overhead in comparison with CFCSS and RSCFC methods and; therefore, its usage is more appropriate.

5. Conclusion

In this article, we have proposed a generic software-based method for control flow error detection. The idea of proposed method is very generic and can be realized in various ways. The proposed method can be combining with different control flow checking method. The improved method effectively checks the intra-block control-flow jump errors and to reduce overhead, Check Instruction is placed at the end of some basic blocks. The improved method is relatively low in cost since it does not require any special hardware. Our fault injection experiments show that the previously proposed signature based techniques detects around 92% of all control-flow faults; however, the combination of proposed method with CFCSS detects around 96% of all control-flow faults. This method has high fault coverage in comparison of previously proposed signature based techniques while maintaining the performance overhead has nearly SCFC.

References

[1] P. Pop, "Analysis and Synthesis of Communication-Intensive Heterogeneous Real-Time Systems," PhD Dissertation, Department of Computer and Information Science, Linköping, Sweden, 2003.

[2] J.L. Hennessy and D. A. Patterson, Computer architecture: a quantitative approach: Elsevier, 2007.

[3] D. Zhu, and H. Aydin, "Reliability Effects of Process and Thread Redundancy on Chip Multiprocessors," Presented at the 2006 in Proc. Of the 36th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.

[4] Rajabzadeh, A., and Miremadi, S. CFCET: A Hardware-based Control Flow Checking Technique in COTS Processors using Execution Tracing. Microelectronics Reliability 46, 5 (2006), 959–972.

[5] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Control-flow checking by software signatures", In IEEE Transactions on Reliability, Vol. 51, pp.111-122, March 2002.

[6] A. Li, and B. Hong, "Software Implemented Transient Fault Detection in Space Computer", *Aerospace Science and Technology*, Vol. 11, No. 2, pp. 245-252, Mar. 2007.

[7] S. Asghari, H. Taheri, H. Pedram and O. Kaynak, "Software-based Control Flow Checking against transient faults in industrial environments, *IEEE Trans. on Industrial Informatics,* Vol. 10, pp. 481-490, 2014.

[8] S. A. Asghari, A. Atena, H. Taheri, H. Pedram, and S. Pourmozaffari, "I2BCFC: An Effective Intra-Inter Block Control Flow Checking Method Against Single Event Upsets", *Applied Sciences, Engineering and Technology*, Vol. 4, No. 21, pp. 4367-4379, Nov. 2012.

[9] L. Jian-ming, T. Qing-ping, X. Jian-jun, J. Cheng, "Control Flow Detection Based on Path Tracking", Computer Engineering, Vol. 35 No. 20, pp.68-70 Oct, 2009.

[10] Y.Wu, G.Gu, S.Huang, and J.Ni, Control Flow Checking Algorithm using Soft-based Intra-/Inter-block Assigned-Signature. *In Computer and Computational Sciences, 2007 IMSCCS. Second International Multi-Symposiums,* pp. 412–415, Aug. *2007.*

[11] W. Chao, F. Zhongchuan, C. Hongsong, B.Wei, L.Bin, C.Lin, Z. Zexu, W.Yuying and C.Gang, "CFCSS without Aliasing for SPARC Architecture", In International Conference on Computer and Information Technology (CIT), pp. 2094 –2100, Jul. 2010

[12] M. Paige, "On partitioning program graphs", IEEE Transactions on Software Engineering SE-3(6): 386–393, 1977.