Table of Contents Search Author Index

# A Checkpointing Technique for Rollback Error Recovery in Embedded Systems

Mohsen Bashiri, Seyed Ghassem Miremadi and Mahdi Fazeli
Dependable Systems Laboratory,
Computer Engineering Department,
Sharif University of Technology, Tehran, Iran.
E-mail: {m_bashiri, b_fazeli}@mehr.sharif.edu, Miremadi@sharif.edu

***Abstract*— In this paper, a general Checkpointing technique for rollback error recovery for embedded systems is proposed and evaluated. This technique is independent of used processor and employs the most important feature in control flow error detection mechanisms to simplify checkpoint selection and to minimize the overall code overhead. In this way, during the implementation of a control flow checking mechanism, the checkpoints are added to the program.**

**To evaluate the Checkpointing technique, a pre-processor is implemented that selects and adds the checkpoints to three workload programs running in an 8051 microcontroller –based system. The evaluation is based on 3000 experiments for each checkpoint.**

***Index Terms*— Rollback error recovery, Embedded systems, Control Flow Checking, Checkpoint**

## I. Introduction

Despite the widely use of fault-tolerant microprocessors, like ERC32 [1] & LEON [2] in safety critical applications, employment of COTS microprocessors becomes a common trend while cost and time to market [3] are most important concerns.

Since the error detection capability of COTS microprocessor is relatively low, it is essential to use some supplementary error detection mechanisms to enhance the reliability of COTS based embedded systems.

Regarding the previous researches, control flow checking technique is an efficient and viable solution to enhance the error detection capability of COTS microprocessors. In this technique the execution flow of a program is followed up and if a violation from the correct program flow occurs a control flow error (CFE) is detected. Many control flow checking techniques have been introduced in literature such as TSM (Time Signature Analysis) [4], TTA (Time-Time Address signature) [5], ECIC (Enhanced Committed Instruction Counting) [6] and ECI (Error Capturing Instructions) [7].

In a safety critical embedded system, it is fundamental to recover the system after the detection of an error as soon as possible with minimum overheads. Error recovery is the process of regaining the system integrity or its operational status after the occurrence of an error [8]. Basically, the error recovery techniques are based on three policies:

- Global reset [9]
- Rollback recovery [10]
- Roll-forward recovery [11]

In first and second techniques, whenever the error detection mechanism reports an error manifestation, the system returns to the beginning or a distinct point of the program. Since there is no need to additional hardware and also the software overhead is rather low, these two techniques are cost efficient. In such techniques, whenever an error is detected, the system must rollback to a distinct point or the beginning of the program, so the considerable time overhead is the main disadvantage of these techniques.

In the third technique, the continuity of the running program will be kept so that the recovery latency is omitted. On the other hand, the extra hardware is imposed to the system in order to implement the recovery technique and the cost efficiency would be lost. In fact, to guarantee the continuity of the running program, it is necessary to detect error as soon as possible and the system must not miss the time. To reach the aim, the system should have a correct run between at least two concurrent runs of the program and a spare processor is required to determine which run is correct In the presence of fault.

As discussed above, in each error recovery techniques the cost and performance overhead are the most important factors. Selecting an appropriate error recovery technique is a trade off between two mentioned factors and highly depends to the system requirements. For example, in some applications such as space, there is no constraint on the system cost and in some other applications such as real time systems, minimizing the performance overhead is a major concern.

The error recovery generally is based on checkpointing concept. Checkpointing is the act of capturing the current state of a system for these goals [8]:

- Rollback
- Synchronization
- Error detection

In figure 1 these three goals in a system with ability of roll-forward recovery can be seen clearly.

Top

1-4244-0765-6/06/$20.00 ©2006 IEEE

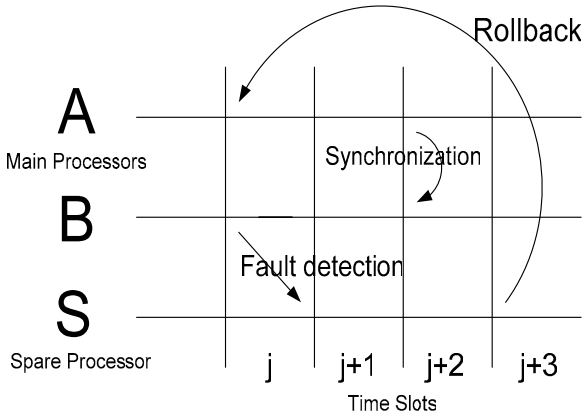


Fig 1- Checkpoint usage in roll-forward recovery

In rollback error recovery techniques, checkpointing is used to regain the program correct state. In contrast, in roll-forward recovery techniques, checkpointing is used to help the system to proceed without backtracking. In [12], [13], [14] and [15] some error recovery mechanisms are proposed and evaluated.

The structure of this paper is as follows: Following the introduction, the proposed checkpointing technique is explained in section 2. In section 3, the experimental system is introduced and the results are reported and finally, some future works are pointed in section 4.

## II. The Proposed Checkpointing Technique

The first step to develop an error recovery technique is defining the error model. Devising a technique which can support all possible errors is impractical so it is mandatory to restrict the error model to reduce the overheads and cost of the technique. The proposed technique is based on control flow error model. In the compile time, the program is portioned in to basic blocks. The basic block is a group of instruction in a program that there is no jump instruction or jump destination inside it. Then the error detection mechanism is added to the program basic blocks and the basic blocks in which the state of the system should be captured are defined. The checkpointing is also implemented as an interrupt routine. In run time, whenever a control flow error is detected, the error detection mechanism informs the error recovery routine and the recovery routine recovers the state of the system to a recently correct stored state.

In this experiment checkpoint includes the content of the registers like general purpose registers, accumulator, stack pointer and program status word, and memory locations like stack region, constants and variables.

The checkpoint is saved in memory for rolling back the system whenever an error is detected. This memory essentially must be a fault-tolerant memory. Techniques like TMR memory modules (hardware redundancy) or error correction codes (information redundancy), can be applied to the memory to make it robust.

Whenever the system wants to capture a checkpoint from the program, the block in which the checkpoint is being captured, must be determined.

## III. Experimental Implementation & Results

An 89C51 microcontroller which is widely used in industrial applications is used to implement this technique.

Three benchmarks have been used in the evaluation system.

- Bubble sort
- Matrix multiplication
- Link list copy

In each of these benchmarks, the program has been divided to basic blocks. Then a checkpoint capturing and a recovery routine are added to the main program.

In next step, the block in which the checkpoint must be captured is determined. Since it is probable that an error occurs before the first checkpoint location, it is mandatory to capture a checkpoint in the beginning of the first block.

For each used bench mark, a checkpoint capturing is inserted to each of basic blocks individually and the overheads are extracted. The results comparison is done at the end of the experiments.

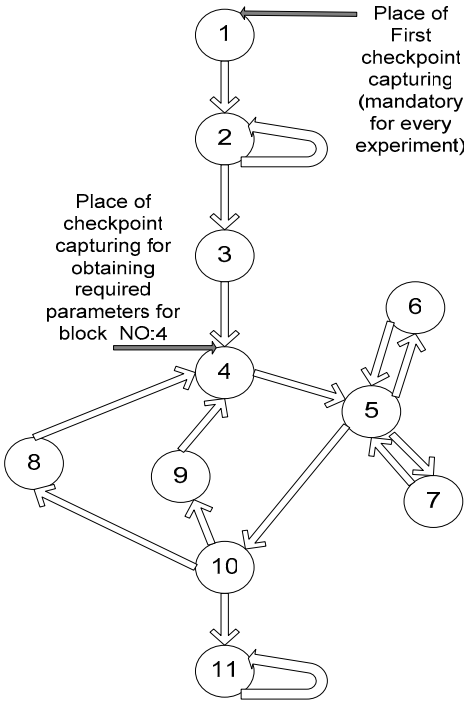


Fig 2- Control flow graph of a typical program

In order to announce the microcontroller that an error has been detected, a controllable random signal generator is used to send a virtual error detection signal to the microcontroller. This triggers the error recovery mechanism to recover the system to the previous correct captured state.

The assumptions in this technique are as follows:

- Since the main goal of this research work is the evaluation of the proposed error recovery technique, the error detection latency is supposed to be zero for the sake of simplicity. If the error detection mechanism is specified, the

overheads can easily be extracted regarding the error detection latency of the mechanism.

- There are no interrupt or I/O actions during the runtime of the main program. However it is possible to removing this restriction for the scheme, by some modification.
- It assumes that all the errors are transient. This is a true assumption based on previous researches [16]. The permanent errors might never been repaired.

For each block, about 3000 experiments have been done. The average runtime of the program is then extracted. The other important parameter which is measured in this experiment is the error recovery delay by comparing the obtained results with the result of golden run.

The number of required cycles of checkpoint and error recovery routine for each used benchmarks are reported in table I.

TABLE I
The time overhead of the checkpoint and recovery routine

| Benchmark | Main program [cycles] | Checkpoint routine [cycles] | Recovery routine [cycles] |
|---|---|---|---|
| Bubble sort | 937 | 50 | 61 |
| Link list copy | 408 | 85 | 96 |
| Matrix multiplication | 580 | 102 | 114 |

TABLE II
The experimental results of matrix multiplication benchmark

| Blocks label | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| Blocks iterations in each run | 18 | 1 | 3 | 9 | 3 | 3 | 1 | 1 | 3 |
| Total execution time with checkpoint capturing without fault injection [cycles] | 2642 | 806 | 1022 | 1670 | 1022 | 1022 | 806 | 806 | 1022 |
| Total execution time with checkpoint capturing with fault injection [cycles] | 2938 | 1060 | 1127 | 2273 | 1161 | 1202 | 1046 | 1099 | 1235 |
| Performance overhead [%] | 355 | 38.9 | 76.2 | 187.9 | 76.2 | 76.2 | 38.9 | 38.9 | 76.2 |
| Error recovery latency to normal time ratio [%] | 11.2 | 31.5 | 10.3 | 36.1 | 13.6 | 17.6 | 29. 8 | 36.3 | 20.8 |

TABLE III
The experimental results of link list copy benchmark

| Blocks label | A | B | C |
|---|---|---|---|
| Blocks iterations in each run | 8 | 1 | 8 |
| Total execution time with checkpoint capturing without fault injection [cycles] | 1245 | 601 | 1245 |
| Total execution time with checkpoint capturing with fault injection [cycles] | 1405 | 791 | 1418 |
| Performance overhead [%] | 205 | 47.3 | 205 |
| Error recovery latency to normal time ratio [%] | 12.9 | 31.5 | 13.9 |

TABLE IV
The experimental results of bubble sort benchmark

| Blocks label | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| Blocks iterations in each run | 10 | 1 | 9 | 45 | 12 | 45 | 45 | 9 | 8 |
| Total execution time with checkpoint capturing without fault injection [cycles] | 1562 | 1058 | 1506 | 3522 | 1674 | 3522 | 3522 | 1506 | 1450 |
| Total execution time with checkpoint capturing with fault injection [cycles] | 1872 | 1389 | 1589 | 3616 | 1765 | 3580 | 3674 | 1705 | 1690 |
| Performance overhead [%] | 66.7 | 12.9 | 60.7 | 276 | 78.6 | 276 | 276 | 60.7 | 54.7 |
| Error recovery latency to normal time ratio [%] | 19.8 | 31.2 | 5.5 | 2.7 | 5.4 | 1.6 | 4.3 | 13.2 | 16.6 |

In tables II, III and IV the obtained experimental results of the proposed checkpoint technique are reported. The performance overhead is calculated by dividing the deference of total execution time of the program with and without checkpoint capturing by the total execution time without checkpoint capturing. The error recovery latency also is extracted by dividing the deference of total

execution time of the program with and without fault injection by the total execution time without fault injection.

As it is shown in the tables, there is an inverse relation between performance overhead and error recovery latency. It means that if the checkpoint capturing is performed in the most used blocks the maximum performance overhead is imposed to the system. On the other hand, the error recovery latency is not necessarily minimum but significantly reduced regarding the location of the blocks. For example in our experiments, if checkpoint capturing is performed in a frequently used block which is not located at the beginning or the end of the program, the minimum error recovery latency is obtained in most cases. Therefore, to obtain the minimum error recovery delay, it is not only sufficient to select the most used blocks for checkpoint capturing but also the location of the block should be considered.

## IV. Conclusion & Future Works

In this paper, an error recovery mechanism for embedded systems based on checkpoint capturing principal was presented and evaluated. The experimental results showed that the proposed technique has very low memory overhead as well as the acceptable program execution time overhead. The error recovery latency can be also significantly reduced by inserting the checkpoints in appropriate locations.

Regarding the error recovery methods, it is possible to apply the roll-forward recovery to embedded system by using control flow error model.

This technique can be also implemented in distributed microcontroller systems to improving the dependability of entire system ore some selected nodes.

In this paper, it was assumed that the checkpoint capturing is done in only one of the blocks except the first blocks whereas the checkpoint capturing can be performed in more than one block and the results may be better in reducing the error recovery latency.

Also it is possible to extracting the analytical model of the scheme, for comparing it to the experimental results.

## References


[1] J. Gaisler, "Concurrent Error-Detection and Modular Fault Tolerance in a 32-bit Processing Core for Embedded Space Flight Applications," *Proc. of Fault-Tolerant Computing Symposium*, Vol. 24, IEEE Computer Society, 1994, pp. 128-130.

[2] J. Gaisler, "A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture," *Proc. of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'02)*, Jun. 2002, pp. 409 - 415.

[3] A. Mahmood, and E. J. McCluskey, "Concurrent Error Detection Using Watchdog Processors – A Survey," *IEEE Trans. on Computers*, Feb. 1988, pp. 160 -174.

[4] H. Madeira, M. Rela, P. Furtado and J. G. Silva, "Time Behaviour Monitoring as an Error Detection Mechanism," *Proc. of the 3rd IFIP Working Conference on Dependable Computing for Critical Applications (DCCA-3)*, Sep. 1992, pp. 121-132.

[5] G. Miremadi, J. Ohlsson, M. Rimen, and J. Karlsson, "Use of Time, Location and Instruction Signatures for Control Flow Checking," *Proc. of the DCCA-6 International Conference, IEEE Computer Society Press*, 1998, pp. 201-221.

[6] A. Rajabzadeh, and G. Miremadi, "Enhanced Committed Instruction Counting (ECIC): A Scheme for Error Detection Enhancement in COTS Processors," *Proc. of the IEEE TTTC International Conference on Automation, Quality and Testing, Robotics (AQTR2004 THETA 14)*, Tome I, Cluj-Napoca, Romania, May 2004, pp. 291-296.

[7] G. Miremadi, J. Karlsson, U. Gunneflo, and J. Torin, "Two Software Techniques for On-Line Error Detection," *Proc. of the 22nd International Symposium on Fault-Tolerant Computing (FTCS-22)*, Jul 1992, pp. 328-335.

[8] D. K. Pradhan, *Fault-Tolerant Computer System Design*, Prentice-Hall, ISBN: 0-13-057887-8, 1996.

[9] D. A. Rennels, and R. Hwang, "Recovery in Fault-Tolerant Distributed Microcontrollers," *Proc. of the International Conference on Dependable Systems and Networks (DSN 2001)*, IEEE Computer Society Press, Göteborg, Sweden, Jul. 2001, pp. 475-480.

[10] D. K. Pradhan, and N. H. Vaidya, "Brief Contributions Roll-Forward and Rollback Recovery: Performance-Reliability Trade-Off," *IEEE Trans. on Computer*, Vol. 46, No. 3, Mar. 1997, pp. 372-378.

[11] J. Long, W. K. Fuchs, and J. A. Abraham, "A Forward Recovery Using Checkpointing in Parallel Systems," *Proc. of the International Conference on Parallel Processing*, Aug. 1990, pp. 272-275.

[12] D. Szentivanyi, S. Nadjm-Tehrani, and J. M. Noble, "Optimal Choice of Checkpointing Interval for High Availability," *Proc. of the 11th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'05)*, Hunan, China, Dec. 2005, pp. 159-166.

[13] G. L. Park, H. Y. Youn, and H. S. Choo, "Optimal Checkpoint Interval Analysis Using Stochastic Petri Net," *Proc. of the 7th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'01)*, Seoul, South Korea, Dec. 2001, pp. 57-60.

[14] N. Bowen, and D. K. Pradhan, "Processor and Memory Based Checkpoint and Rollback Recovery," *IEEE Trans. on Computers*, Feb. 1993, pp. 22-31.

[15] N. Bowen, and D. K. Pradhan, "Virtual Checkpoints: Architecture and Performance," *IEEE Trans. on Computers*, May 1992, pp. 516-525.

[16] M. A. Schuette, and J. P. Shen, "Processor Control Flow Monitoring Using Signatured Instruction Streams," *IEEE Trans. on Computers*, Vol. C-36, No. 3, Mar. 1987, pp. 264-276.