International Journal of Future Generation Communication and Networking
Vol. 13, No. 1, (2020), pp. 1250-1257

RELIABLE FAULT TOLERANT MULTI-CORE SYSTEM: A SURVEY

Usha Jadhav[1], Dr. P. Malathi[2]

[1]*Assistant Professsor, Dept. of Electronics and Telecommunication Engg. DYPCOE, Akurdi Pune, India*
[2]*Professor, Dept. of Electronics and Telecommunication Engg. DYPCOE, Akurdi Pune, India*

Abstract

ARM Processor is a core used in 100's of billion of real-time Embedded System which contain Electronics with less area overhead, low power consumption, reduced cost and portable with high computation. To carry the user experience in the home, and office over the car such embedded solution based on multi-core platforms are used in diverse application domain like aerospace, industrial automotive, Smartphone, Tablets, Medical, Audio Video player to provide fail safe operation with consumer comfortability. This paper focuses on survey of various faults tolerant techniques for ensuring low overhead to protect processor and program execution flow which improves the reliability of the system. Hardware implemented fault tolerant techniques inevitably add resource overhead while Software implemented fault tolerant technique add runtime overhead. The future scope of this paper provides a trade-off between resource overhead and runtime overhead to improve fault coverage.

Keywords: real-time Embedded System, Fault, Fault tolerant techniques, multi core

I. INTRODUCTION

Intel have turned to multi core processor based on 10nm process node [2] which covers strong commercial demands in market for high performance computing and safety critical application using Graphics Processing Units [1]. Reliability is a major concern for such real time high performance computing devices due to the use of submicron technologies. It has increased the sensitivity to radiation induced transient faults [3]. These transient on chip faults [6] arises from various sources like high energy particle impact, radiation intensive space environment, total ionizing dose accumulation and aging of chip die. This leads to generation of Single Event Upsets (SEUs).

In order to satisfy real time constraints with safety critical system, one approach is to address hardware faults and other is to tackle with software faults. Hardware faults [7] can occur due to smaller size of functional units, aggressive lowering of operating voltages, SEUs caused by radiation intensive operating environment. Many techniques used to achieve fault tolerance in hardware by replicating or adding hardware units like by adding redundancy in sensors and in processors used in system and applies voting algorithms. But it incurs increase in area, power consumption, performance degradation and high design and manufacturing cost. Software faults can occur due to occasional deadline misses while communicating with register and memory. Techniques implemented in software are able to increase the reliability by insertion of extra instructions using Dual Modular Redundancy (DMR) and Triple Modular Redundancy (TMR) [4] in the program code to detect errors in both data flow and control flow. This leads to increase in runtime overhead.

Organization of the Paper: The rest of this paper is organized as follows. Section II shows Multi-core System background with necessary system requirements for fail safe operation to achieve higher reliability. In Section III, the Fault Tolerant techniques in multi-core system are discussed. In Section IV, we present Reliability Estimation of Fault tolerance Techniques in Software. Comparative assessments of Fault Tolerant Techniques in Software are discussed in Section V. Finally, we conclude the paper in Section VI by providing conclusion and future scope of research.

ISSN: 2233-7857 IJFGCN

II. MULTI CORE SYSTEM BACKGROUND WITH IT'S REQUIREMENT

Multicore systems are used with diverse range of applications like aerospace, Automotive, Network, Digital Signal processing (DSP), High performance Computing (HPC) and Graphics processing Unit (GPU). Increasing hardware and software complexity of such applications have challenges in reliability with increasing soft errors through reading data either from main memory or register file. Main memory is accessible for the content of register in high level programming. But the register file is critical resource for knowing the chance of error in register to propagate to output and it is the parameter used to measure the reliability of the system. Thus the use of GPU over traditional CPU supports for active threads to fetch the input data [1]. GPU register file is the fastest type of memory on GPU and Selective fault tolerant techniques using Single error correction double error detection (SECDED)[1] helps in getting probability for multi bit upset in register file which is higher than in cache and main memory.

Major challenge in multi core system is to conduct a large number of fault injection campaigns in reasonable time, to provide detailed observation in presence of faults and identify relationship between application profiling and specific platforms parameters in large data set [2]. Thus the use of Virtual platform frameworks using machine learning approach solves the problem. Soft errors may lead the processor to incorrectly execute an application or enter in the loop and never finish the execution [3]. The Software Implemented Hardware Fault Tolerant Techniques (SIHFT) is applied to assembly code executed by ARM processors. Online monitoring and checking of software control flow to detect run time deviations from control flow graph is critical in resource constrained embedded system's reliability [4]. Worst Case Execution Time (WCET)-Aware Control Flow Checking based on Super Nodes (WACFC-SN) [4] which makes program partially resilient to control flow errors while keeping the program WCET below a given upper bound.

Parallel applications using OpenMP and Pthreads running on top of Linux operating system for multicore need to be protected against soft errors for high reliability compared to sequential bare metal ones[5]. In such case operating system itself is a source of error. Traditional fault tolerance methods like Triple Modular Redundancy (TMR) and Conditional Dual Modular Redundancy (CDMR) protects only the application and do not protects Linux OS. The Fault tolerance method to the operating system to evaluate the time overheads is very critical task in multicore platform.

Today with the use of smart phones while driving a car can manage for fail safe operation with all risks in maximum reaction time of up to 500ms. Multi core support works with Linux, Mac OS and Virtual Machines. To design a highly reliable multi core system:

- Synchronization protocols and data flow dependencies [8] during parallelism and multi-threading need to be considered.
- Concurrent access to shared data must be considered explicitly when addressing system safety [8]
- Multi core soft error evaluation by using realistic Linux kernel, instruction set architectures (ISAs) and standard parallelization libraries, considering several benchmarks [2]
-Hardware-software co-design to meet hard real time constraints [7]
-Beam radiation experiment and fault injection campaigns at hardware and software level [1]

III. FAULT TOLERANT TECHNIQUES IN MULTI-CORE SYSTEM

The increasing hardware and software complexity of multi core system requirement gives two different approaches of fault detection and mitigation as fault tolerant techniques at hardware and software level. Hardware based fault tolerant techniques have direct access to hardware resources by replicating or adding hardware module. While Software based fault tolerant techniques protects the processor against soft errors by adding instruction redundancy in the register file and improves the reliability. N-Version Programming is used by adding the N-modular redundancy scheme to provide tolerance against Hardware faults while Recovery Block (RB) scheme is used to provide tolerance against software faults which minimizes the total system cost by providing reliability of the real-time optimized individual module. Hardware based fault tolerant techniques are not applicable to commercial-off-the-shelf (COTS) processor but Software based fault tolerant techniques are applicable to COTS as they are reconfigurable. Hardware based fault tolerant techniques increases area overhead, Power consumption, and performance degradation with high cost. Whereas Software implemented fault tolerant technique requires more processing time and increases energy

ISSN: 2233-7857 IJFGCN

consumption since more instructions are executed. In software implemented fault tolerant technique increases reliability compared to hardware because of program uses more memory addresses instead of registers.

There are three classes of SW implemented fault tolerant techniques: Naïve Duplication, Selective Duplication and Algorithmic based Fault Tolerance (ABFT) [1]. In Naive duplication whole program code is duplicated while in selective duplication method program is divided into basic blocks and only selected blocks are duplicated which results in performance but reduced reliability. ABFT is limited to specific group of applications like High Performance Computing.

Reliability of Multi core system: Transient software faults in multicore system are detected and mitigated either by using fault masking or fault removal. Fault masking sends no effect on program output or does not use corrupted data. While fault removal uses forward error and backward error recovery algorithms based on state condition and improve the reliability of the system. There are two ways to evaluate the reliability of a system. First approach is using beam radiation experiment which provides more realistic at the cost of limited visibility while the second approach is using fault injection campaigns which provides more visibility with realistic result. Fault Injection campaign at RTL level is used for simple circuits while software level is used for system.

IV. RELIABILITY ESTIMATION OF FAULT TOLERANCE TECHNIQUES IN SOFTWARE

Software only fault tolerant techniques are more effective and efficient as they have been used at different abstraction level from high level C++ code to low level assembly code. Also it provides flexibility in program execution environment for real time application where legacy binary code and redundant code can co-exist depending on the required level of reliability imposed by instruction-, thread-, process- and virtual machine. In safety critical systems such as automotive, aerospace and industrial control, it is important to guarantee a system's real time constraints which provide strong demand for reliability and fault tolerance. The need for reliability and fault tolerance arises to protect the processors against soft errors. It may affect the data flow in register or memory and program flow execution. Thus data flow in register and memory is protected through Data flow techniques and Execution flow is controlled through control flow technique. The comparison between Data flow and control flow techniques is given in Table I.

There are various data flow and control flow fault tolerant techniques used to provide reliability in an Embedded system like Error Detection by Duplicated Instructions (EDDI) [3] called Variables (VAR) technique as data flow technique and Control Flow Checking (CFC) technique [4],
[7], Software Implemented Hardware Fault Tolerant (SIHFT) [3] technique, Triple Modular Redundancy (TMR) technique, Conditional Double Modular Redundancy (CDMR) [5] technique, Software-only Error Detection Technique using Assertions (SETA)[3], Selective Fault Tolerant technique and Machine learning techniques as control flow technique to detect soft errors.

TABLE I

COMPARISON BETWEEN DATA FLOW AND CONTROL FLOW TECHNIQUE

Data flow technique	**Control flow technique**
Data flow is exercised where data is passed through program and what transformations are carried out in a system on data.	Control flow is exercised about the workflow of the task to be executed in a particular order.
This technique is represented using data flow graph.	This technique is represented using control flow graph.
Protects the data flow register and memory.	Ensures the correctness of execution flow.
Data flow technique uses Registers and are replicated.	Control flow technique divides the code in to basic blocks.
Checkers are inserted	A unique signature is assigned to each basic block.

All these techniques aim to achieve a trade-off between performance and/or resource overheads in terms of runtime overhead/Execution time overhead and fault detection coverage and memory

ISSN: 2233-7857 IJFGCN

footprint. The rum time overhead is measured using Worst Case Execution Time (WCET), Architectural Vulnerability Factor (AVF) and Mean Work to Failure (MWTF) metric.

Worst Case Execution Time is the maximum length of the time taken by the computational unit to execute a task, thread or process on specific hardware platform. WCET addresses system level multitasking issues and real time mixed criticality schedulability analysis. In multi core platform, other tasks in the system will impact the WCET of a given task if they share cache, memory lines and other hardware features.

Architectural Vulnerability Factor allows evaluating the probability of a low level corruption to propagate to the output. It is a measure of micro architectural structure's susceptibility to transient faults.

The MWTF is defined by equation (1) [3].

$$MWTF = \frac{\text{Amount of work completed}}{\text{Number of errors encountered}} \quad (1)$$

$$MWTF = (\text{raw error rate X AVF X execution time})^{-1}$$

Faults can originate from hardware or software. Majority of software faults are transient in nature. Error detection is possible using various fault tolerant techniques. A fault can be masked or cause an error. If error is generated, it can be detected or undetected. Fault Coverage is calculated by the equation (2) [3].

$$F_{coverage} = \frac{E_{detected} + F_{masked}}{F_{total}} = 1 - \frac{E_{undetected}}{F_{total}} \quad (2)$$

Where: $F_{coverage}$ is the Fault Coverage.

$E_{detected}$ is the number of errors detected.

F_{masked} is the number of correct executions.

$E_{undetected}$ is the number of undetected errors.

F_{total} is the number of executions.

In the data flow technique, if the aim is to detect the errors, registers are duplicated and when correction is included, registers are triplicate. Thus, error correction presents higher overheads than error detection. Various data flow techniques uses more than one checking rule as checkers. If more checkers are included in one technique, more reliability is achieved.

Algorithm 1 shows the steps for software-implemented fault tolerant Data flow technique.

Algorithm 1

Software-implemented fault tolerant Data flow technique

1. Assign every register used by the program a spare register as replica.
2. Duplicate the instructions which perform write operations on registers or memory excluding branch instruction.
3. Replicate all instructions that operate on the replicated data.
4. Perform consistency check using checkers between the original register data and their replicas data using compare instruction.
5. If mismatch is detected the program branches to an error detection subroutine which flags an error to the host.

Control flow checking uses a signature to detect control flow errors in processors or program. There are basically two types control flow techniques as Hardware-only control flow technique and Software-only control flow techniques. Hardware-only control flow technique uses extra signature and makes use of the watchdog helps in the error detection with extra power as watchdog do not access the cache memories available on chip to processor. This technique concerned with error detection rate, but not about the overheads they cause.

Control flow techniques are designed to protect the program flow against incorrect jumps using signature. The signature is concerned with a global register at the beginning of basic block. A basic block consists of instructions executed in sequence. Verify the signature register contains the expected

ISSN: 2233-7857 IJFGCN

value or not by inserting checkers in the code. If it does not, it means program flow was incorrectly followed and an error is reported.

Algorithm 2 shows the steps for software-implemented fault tolerant control flow technique.

Algorithm 2

Software-implemented fault tolerant Control flow technique

1. Divide the code into Basic Blocks (BB).
2. Assign entry and exit signature at the beginning and end of each Basic block.
3. Perform some transformation or insert additional checking code as Instrumentation code at the beginning and end of each basic block.
4. At runtime, as control flow passes through basic blocks, the instrumentation code with bitwise AND/XOR operation computes the signature.
5. Compare run time signature with the pre-computed expected signature.
6. If mismatch is detected, the error handler is invoked.

In order to automatically apply the software implemented technique to various case study application, various papers used a tool like HPTC [1], OVPsim-FIM [2], [3], [5] gem5-FIM [2] and QEMU etc. as fault injection frameworks. Methodology flow for Software-implemented Fault tolerance technique's flow is shown in Fig. 1.

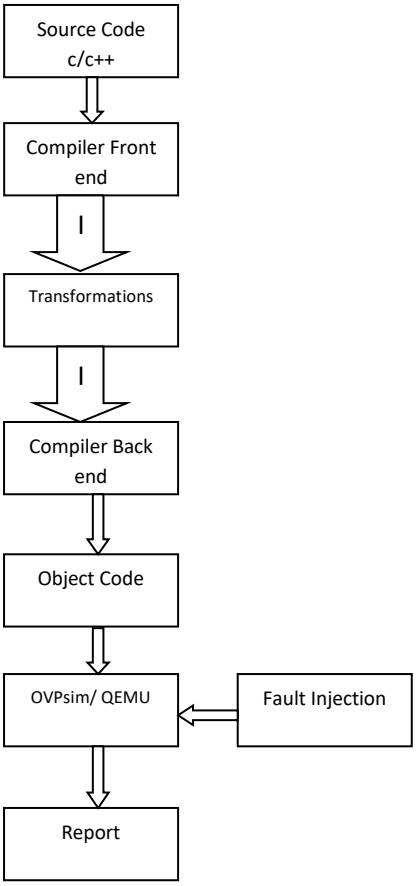

Fig. 1 Software-implemented Fault tolerance technique's flow

In the first step, the source code for the target architecture is compiled using front end in the absence of fault and generates Intermediate Representation (IR). Apply some transformations based on redundancy requirements and generate hardened IR using appropriate fault tolerance technique. Hardened IR is given to the compiler back end to generate object code. Generate a fault list based on fault injection campaign. Simulate the targeted application using OVPsim or QEMU and inject faults at run time based on fault list. This evaluates fault tolerant techniques' ability to protect the processor against soft error with low overhead and program execution flow which improves reliability of the system.

ISSN: 2233-7857 IJFGCN

V. COMPARATIVE ASSESSMENT OF FAULT TOLERANCE TECHNIQUES IN SOFTWARE

Various authors experienced the different types of faults as listed in table II-column 3 with case studies like visual odometry which is used to determine vehicle position and orientation by analyzing series of images using machine learning algorithm, use of selective fault tolerance technique to detect faults in GPU register file at both Hardware and software level and tested ARM processor using radiation and fault injection to estimate the reliability using data flow and control flow techniques. Omission faults are originated when action not performed by human but when it should be. Silent Data Corruption (SDC) [1] is when the program output is incorrect. This SDC is harmful as it is undetected. Detected Unrecoverable Error (DUE) [1] occurs when the application crashes or the system hangs. DUE gives performance degradation and data loss. Vanished (V) fault as no fault traces are left means faults are masked at processor level and do not propagate to memory with application produces correct output and internal memory state's is correct. While Output Not Affected (ONA) [2] in which faults propagate to application memory state's and final result of computation will be within the acceptable error margin and the application ends within the number of iterations executed in golden run. Output Mismatch (OMM) [2] fault in which the application aborts without any error indication and resulting memory is affected. An Unexpected Termination (UT) [2] faults indicate the error with application terminates abruptly. Hang in which application does not finish needs preemptive scheduling. UNACE [5] means Unnecessary for Architecturally Correct Execution in which the program completes its execution and produces an expected output. Exceptions are not applicable for bare metal applications because of the absence of an operating system to catch them. Some of the simulators see the exceptions as segmentation faults, halting the simulation and reporting the problem.

In order to find out the soft error assessment of various fault tolerant techniques, various researchers used different benchmarks like NASA Advanced Supercomputing (NAS) parallel benchmark to render the performance evaluation including various applications implemented in OpenMP and MPI. OpenMP stands for Open Multiprocessing and MPI stands for Message Passing Interface. OpenMP uses shared memory parallelism while MPI is a way to program on distributed memory devices. OpenMP relies on loop parallelization using for-while loop and it is a thread based parallelism. While MPI Application Programming Interfaces (API)'s are both process and thread based approach. OpenMP and MPI provides Linux OS and parallelization libraries impacts various applications like bit count, matrix multiplication, Quicksort, vector sum given in table II according to the types of faults occur. Table II shows the summary of various fault tolerant techniques with types of faults, Benchmarks used by researchers, performance parameter measured and platform used for fault detection.

The various simulation platforms are used to reduce human analysis time required to understand the effect of fault injection as OVPsim-FIM, gem5-FIM and analysis tool such as pandas dataframe with different libraries to assess soft error realibility.

TABLE II

COMPARATIVE ASSESSMENT OF VARIOUS FAULT TOLERANT TECHNIQUES

Ref No.	Fault Tolerant Techniques	Types of Faults	Benchmark	Performance Parameter	Platform Used
[1]	Beam Radiation Experiment(Hardware FTT)	SDC, DUE	HOTSPOT,NW, QuickSort	FIT	Fault Injection Compaign: SASSIFI, HPTC
[1]	Fault Injection Compaign (Software FTT)	SDC, DUE	HOTSPOT,NW, QuickSort	SDC AVF/DUE AVF Vs Register	Fault Injection Compaign: SASSIFI, HPTC

ISSN: 2233-7857 IJFGCN

[1]	HARDWARE-IMPLEMENTED SELECTIVE HARDENING	SDC, DUE	HOTSPOT,NW, QuickSort	Fault coverage Vs Overhead (register Qty-area)	Fault Injection Compaign: SASSIFI, HPTC
[1]	SOFTWARE-IMPLEMENTED SELECTIVE HARDENING EDDI/VAR (register duplication and replication of Instruction)	SDC, DUE	HOTSPOT,NW, QuickSort	Fault coverage Vs Overhead (register Qty-runtime execution)	Fault Injection Compaign: SASSIFI, HPTC
[2]	Supervised and Unsupervised machine learning techniques	Vanish, ONA, OMM, UT, Hang	NAS benchmark implemented in openMP and MPI KITTI Benchmark for Case study on Visual Odometry Algorithm	-Soft error score Vs Branches/ memory Insruction/Conditional Reg. Writes/Cache Hits	Fault Injection Compaign: OVPsim-FIM, gem5-FIM, Analysis tool: Python, Pandas Dataframe Libraries: matplotlib, numpy,scipy
[3]	SIHFT 1. VAR-Data Flow Technique 2. SETA-Control Flow Technique 3. Combined SETA with VAR	SEU test, bit flip in reg	matrix multiplication (MM), quicksort (QS) and Tower of Hanoi (TH)	execution time, memory footprint, fault coverage, and Mean Work to Failure	Fault Injection Compaign: OVPSim-FIM ARM-Cortex A9 Processor
[4]	ILP formulation is used to select WCEP in WACFC-SN algorithm	bit flip in reg	Mälardalen WCET benchmark suite: FIR, insertsort, matmult	fault detection ratios for the benchmark programs WACFC-SN and WACFC-BB	Freescale PowerPC 5554 processor, LLVM compiler
[5]	TMR and CDMR	bit flip in reg, SDC, HANG,UNACE, Exceptions	OpenMP, Pthreads Bit count, matrix multiplication, vector sum	% of Error[SDC, HANG,UNACE, Exceptions(segmentation fault, unidentified)] Vs sequential (Bare Metal, Linux), Parallel (OpenMP, Pthread), speed Up % of UNACE on three application Performance overhead of Bare metal, Linux sequential, Linux Parallel	OVPSim, ARM Cortex-A9
[6]	TMR, DMR	dynamic on-chip component relocation regardless of fault type	Video application	Reduction in resource overhead using Xilinx Microblaze.	Xilinx Kintex-7 FPGA platform

ISSN: 2233-7857 IJFGCN

[7]	PCFC based on ILP formulation	bit flip in reg, Exit from end to main, abort, bus error, seg. fault, exit from error handler	Malardalen WCET Benchmark: Cnt, FIR, FDCT, matmult	Trade off between WECT and fault-detection coverage	LLVM

VI. CONCLUSION AND FUTURE SCOPE

This paper examined that running real time application in multi core platform are susceptible to errors and more challenging for fail safe operation especially in automotive, driverless car, aerospace, smart phones and many more. This paper surveyed the various faults tolerant techniques with types of faults, benchmarks available, performance parameter and platform used for ensuring low overhead to protect processor and program execution flow which improves the reliability of the system. Also given the software based assessment flow which reduces the cost of the system and improves the efficiency in terms of fault coverage per overhead than hardware based fault tolerant techniques. In future, to demonstrate the applicability in realistic environment use of reinforced learning algorithm, combine data flow with control flow technique to improve fault coverage, fault tolerant technique can apply to low level assembly language and extend fault tolerant method to OS and evaluate time overhead in multi-core platform.

REFERENCES

[1] M. Goncalves, F. Fernandes, I. Lamb, P. Rech and J. R. Azambuja, "Selective Fault Tolerance for Register Files of Graphics Processing Units," in IEEE Transactions on Nuclear Science, vol. 66, no. 7, pp. 1449-1456, July 2019.
[2] F. R. da Rosa, R. Garibotti, L. Ost and R. Reis, "Using Machine Learning Techniques to Evaluate Multicore Soft Error Reliability," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 66, no. 6, pp. 2151-2164, June 2019.
[3] E. Chielle et al., "Reliability on ARM Processors Against Soft Errors Through SIHFT Techniques," in IEEE Transactions on Nuclear Science, vol.63, no.4, pp.2208-2216, Aug.2016.
[4] M. Zhang, Z. Gu, H. Li and N. Zheng, "WCET-Aware Control Flow Checking With Super-Nodes for Resource-Constrained Embedded Systems," in IEEE Access, vol. 6, pp. 42394-42406, 2018.
[5] G. S. Rodrigues, F. Rosa, Á. B. de Oliveira, F. L. Kastensmidt, L. Ost and R. Reis, "Analyzing the Impact of Fault-Tolerance Methods in ARM Processors Under Soft Errors Running Linux and Parallelization APIs," in IEEE Transactions on Nuclear Science, vol. 64, no. 8, pp. 2196-2203, Aug. 2017.
[6] V. Dumitriu, L. Kirischian and V. Kirischian, "Run-Time Recovery Mechanism for Transient and Permanent Hardware Faults Based on Distributed, Self-Organized Dynamic Partially Reconfigurable Systems," in IEEE Transactions on Computers, vol. 65, no. 9, pp. 2835-2847, 1 Sept. 2016.
[7] Z. Gu, C. Wang, M. Zhang and Z. Wu, "WCET-Aware Partial Control-Flow Checking for Resource-Constrained Real-Time Embedded Systems," in IEEE Transactions on Industrial Electronics, vol. 61, no. 10, pp. 5652-5661, Oct. 2014.
[8] Lukas Osinski, Tobias Langer, Ralph Mader, Jürgen Mottok. Challenges and Opportunities with Multi-Core Embedded Platform - A Spotlight on Real-Time and Dependability Concepts. ERTS 2018, Jan 2018, Toulouse, France.

ISSN: 2233-7857 IJFGCN