IJSTE - International Journal of Science Technology & Engineering | Volume 4 | Issue 10 | April 2018
ISSN (online): 2349-784X

The Modern Approach to Exploit Multiple Cores using Process-Level Redundancy for Transient Fault Tolerance

Merlin Asha M
PG Student
Department of Electronics and Communication Engineering
SNS College of Technology, Coimbatore – 641035

A Santhosh Kumar
Assistant Professor
Department of Electronics and Communication Engineering
SNS College of Technology, Coimbatore – 641035

Abstract

This paper describes the software approach of fault tolerance for shared memory multi core system using PLR. PLR uses a software-centric approach transient fault tolerance which ensuring a correct software execution. This scheme is used at user space level which does not necessitate changes to the original application. PLR create a set of redundant process per application process. In this scheme multithread redundant process is mainly used to detect the soft errors and recover from the fault. The main goal is to use software leverage by available using hardware parallelism for low overhead fault tolerance. Fault tolerance which allow the system perform correctly even in the presence of faults. In existed the system is appraised for fault coverage performance. This paper presents Software based multi core alternatives for design and analysis transient fault tolerance using process-level redundancy (PLR) which implements fault error detection and fault recovery. This is flexible alternative but higher overhead correctness is defined by software output overhead incurred by our approach ranges is lower when comparable to existed. It furnishes a low overhead mechanism and render improved performance over existent software transient fault tolerance techniques.

Keywords: Fault Tolerance, Process-Level Redundancy, Soft Errors, Transient Faults

I. Introduction

Transient faults, also called as soft errors, which concern the reliability of computer systems. A transient fault occurs even in the presence of error. A fault cause error results failure. Fault cause error observed by deviation from expected behaviour results failure. It occurs event (E.g., cosmic particle strikes, power supply noise, device coupling) alleviation or removal of enough charge to invert the state of the transistor. The inverted value may cause the effect in program execution. A current trend in process technology shows that the future error rate will remain comparatively constant. The number of usable transistor per chip continuously grows in an exponential manner, which increases dramatically. These trends had shown that to ensure correct execution operation of systems. Transient fault characteristics are reliability, dependability, accessibility and availability. By the error correcting codes and parity within high performance microprocessor the memory is easily protected.

However, the same specified techniques cannot be directly adopted for general purpose computing domain. Compared to the ultra-reliable, computing environments, general purpose system is driven by a different and often conflicting set of factors. Thus, the reliability improves to meet user expectations of failure rates. While the reliability level of hardware technique is not rendered by a software technique, they provide a significantly low cost and flexible (zero hardware design cost). For checking computation and control flow process existing software transient fault tolerance approaches use the encyclopaedias to insert redundant instructions. Redundant more than is needed, desired or required to ensure correct execution. This paper presents the design and analysis of transient fault tolerance using PLR (process level redundancy). This paper exploits the Multiple Cores for transient fault tolerance using PLR (process level redundancy). To ensure correct execution, PLR compares the output and it creates a set of redundant process per application process. This is mostly used to leverage overhead mechanism.
This paper makes the following contributions:

- The system is correctly executed by introducing a software-centric transient fault tolerance
- By Sphere of Replication (SOR) concept, we differentiate between hardware-centric and software-centric commonly used
- By ignoring benign fault and to show how software-centric can be effective

II. Background

Fault can be classified by its effect on execution into the following categories
1) Benign fault
2) Silent Data Corruption (SDC)
3) Detected Uncover able Error (DUE)

A. *Benign fault:*

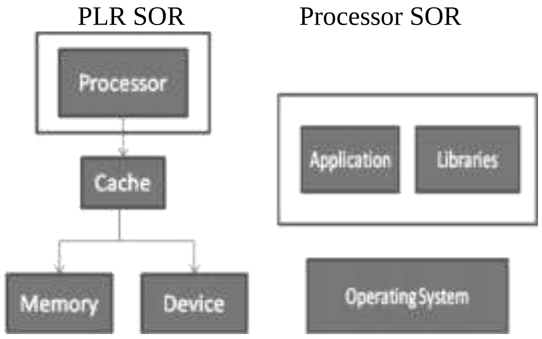
A transient fault that does not propagate to affect the correctness of an application is considered a benign fault.

B. *Silent Data Corruption (SDC):*

A transient fault that is undetected and propagates to corrupt program output is considered as SDC.

C. *Detected uncover able error (DUE):*

A transient fault that is detected and propagates without the possibility of recovery is considered a DUE.



(a) hardware-centric (b) software-centric
Fig. 1: Transient Fault Detection Models

A software centric model (e.g., PLR) views the system as the software layers and the sphere of influence around particular layers. Most previous access is hardware-centric-even compiler approaches (e.g., SWIFT, EDDI) Software-centric able to leverage the strength of a software approach.

III. SOFTWARE-CENTRIC FAULT DETECTION

SOR concept is used for defining the boundary of reliability in redundant hardware design.
1) All the inputs are replicated
2) Execution is redundant
3) Output is compared WHILE the hardware-centric model is appropriate for hardware-implemented techniques; it is awkward to apply the same approach to software. The reason is that software naturally operates at a different level and does not have full visibility into the hardware.

However, hardware-centric SOR is simulated by the effort of previous compiler-based approaches. For example, SWIFT places its SOR, around the processor. Each load is performed twice for input replication and all Computation is performed twice on the repeated inputs. Output comparison is fulfilled by checking the date of each store instruction anterior to executing the store instruction. This particular approach works because it is possible to emulate processor redundancy with redundant instructions. However, other hardware-centric SORs would be impossible to emulate with the software. That is, an SOR around hardware caches cannot be implemented by software alone.

IV. PROCESS-LEVEL REDUNDANCY (PLR)

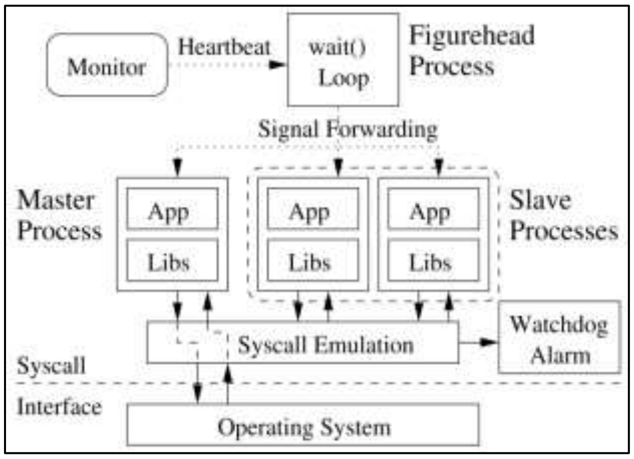


Fig. 2: Overview of the PLR System Architecture with Three Redundant Processes

– Ensures SOR with input replication and output comparison
– System calls for emulation
– For Determinism, it detects and recovers from transient fault

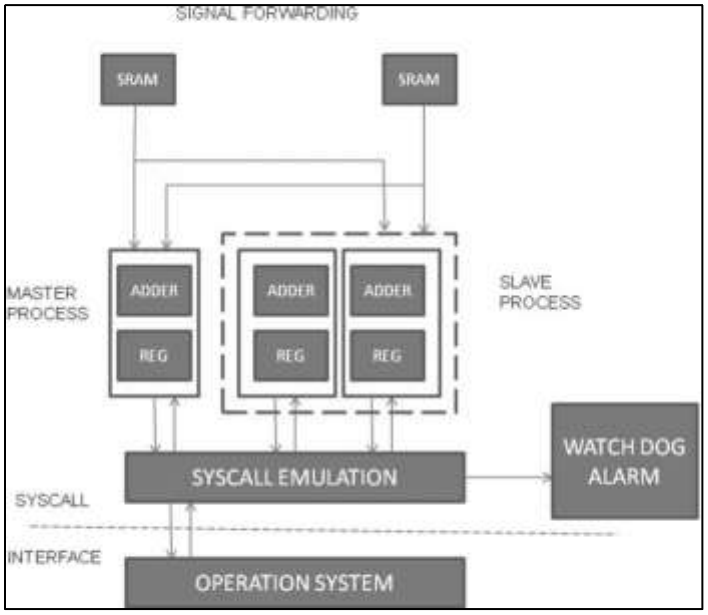


Fig. 3: Modified PLR System Architecture

Software-implemented: PLR is implemented entirely in software and runs in user space under the application. In this manner, PLR is able to provide transient fault tolerance without requiring modifications to the OS or underlying hardware. In addition, software implementation makes PLR extremely flexible. Applications that must be reliable can be run with PLR, while other applications run regularly.

Software-centric: PLR uses a software-centric approach to fault detection with an SOR around the user space application and its accompanying shared libraries. All user-space executions are redundant and faults are only detected if they result in incorrect data exiting user space. This offers the checking boundaries for fault detection compared to most other transient fault tolerance techniques. It permits a PLR to ignore many benign faults.

Replica-based: PLR uses process replicas to provide redundancy. PLR replicates the entire process virtual address space as well as the process metadata such as file descriptors. PLR automatically creates and coordinates among the redundant processes to maintain determinism among the processes, detect transient faults, and recover from detecting faults, as well. Operators at the process level have a distinct advantage in that processes are also a basic abstraction of the OS.

Therefore, PLR can leverage multiple hardware resources such as extra hardware threads or cores by simply allowing the OS to schedule the replicas across all available hardware resources.

A. PLR Overview

An overview of the PLR system is shown in PLR gain control of an application before it begins to execute and begins its initialization phase. First, PLR makes a monitor process and then initializes metadata, including a shared memory segment exploited for inter process communication.

Then, PLR forks the application, N times with N ¼ 2 as the minimum for fault detection and N ¼ 3 as the minimum for fault detection and fault recovery. These processes are the redundant processes, which actually perform the execution of the application. One of The redundant processes labelled the master process and the other one labelled as the slave processes. During execution of the redundant processes, the systems call emulation unit co-ordinates system I/O among the redundant processes.

In general, the master process is to perform system I/O while the slave processes emulate system I/O. The system call emulation unit also implements the software-centric fault detection model and implements transient fault detection and recovery. A watchdog timer is attached to the system call emulation unit, which is used to detect error in which a fault causes one of the redundant processes to hang indefinitely

After initialization and the redundant processes are created, the original process becomes a figurehead process. The figurehead process does not do whatever work. It only holds for the redundant processes to finish execution and forwards signals to the redundant processes. In the modified diagram PLR this paper approach three redundant processes, this blog consist of SRAM, static random access memory, shift register, master and slave process. SRAM is a semiconductor memory that uses a bi stable latching circuitry (flip flop) to store each bit.

SRAM shows data reminisce, but it is still volatile in the conventional sense when the memory is not powered data is eventually lost. Static memory which is mainly used for a storage device it is a permanent storage. In this 6T SRAM it performs word line and bit line operation. The shift register is a type of sequential logic circuit mainly used for storage. They are a group of flip-flops connected in a chain. So, that the output from one flip-flop becomes the input of next flip-flop.

All the flip-flops are driven by common clock and all are set or reset simultaneously. One of the redundant processes labelled the master process and the other one labelled as the slave processes. Here we used an adder application here two half adder as designed as a full adder which propagates the sum and carry output. The system call emulation unit also implements the software-centric fault detection model and implements transient fault detection and recovery. A watchdog timer is attached to the system call emulation unit, which is used to detect error in which a fault causes one of the redundant processes to hang indefinitely.

B. Transient Fault Detection and Fault Recovery

– Output mismatch
– Watchdog timeout
– Program failure Fault detection

Fault tolerance, is which allows the system to operate correctly even in the presence of faults. A technique used during service to detect fault using the operating system.
A technique used during service to detect fault using the operating system.
Detection mechanism Detect as a mismatch of comparing buffer on an output comparison.
Recovery mechanism Use majority vote ensure correct data exist, it kills incorrect process.

C. Windows of Vulnerability

– Faults during PLR execution
– Fault during execution of operating system.

This paper describes a low overhead software based approach. The design consists of SRAM block, shift register, add, and master and slave process. A top module consists of SRAM which is used to store the data. Shift register are used, totally three shift registers and group of flip flops are connected in a chain. All the flip flops are driven by a common clock. The next stage is master and slave process, we are designing an adder circuit. Watchdog timer which is used to detect the error here designed as a NOR and inverter.

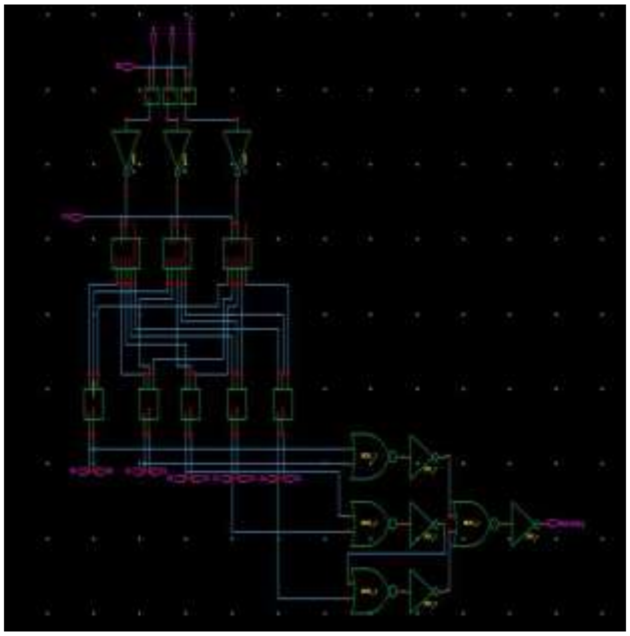


Fig. 4: Design and Analysis of the Transient Fault Tolerance Using PLR

This circuit consists of four stages 1) Memory block 2) shift register 3) master and slave process 4) watchdog timer Now we are designing memory in that memory design input signal are given in memory to store the memory value with respect to word line WL. If WL high mean the data will be positive in shift register module or else it will be stored as it is. Shift register consists of D flip flop with five bits. Based on input memory will produce the output with respect to a clock signal. From the output of the shift register is given to master and slave module it will function the full adder. Full adder process designed by using a two half adder.

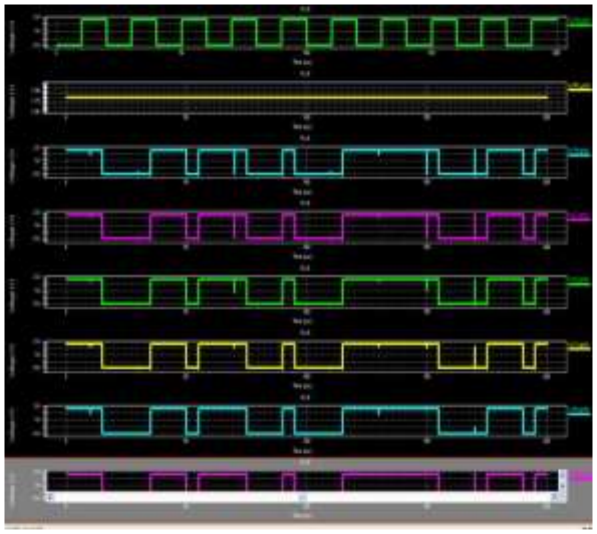


Fig. 5: Results of the Fault Outcome Fault- Free Output

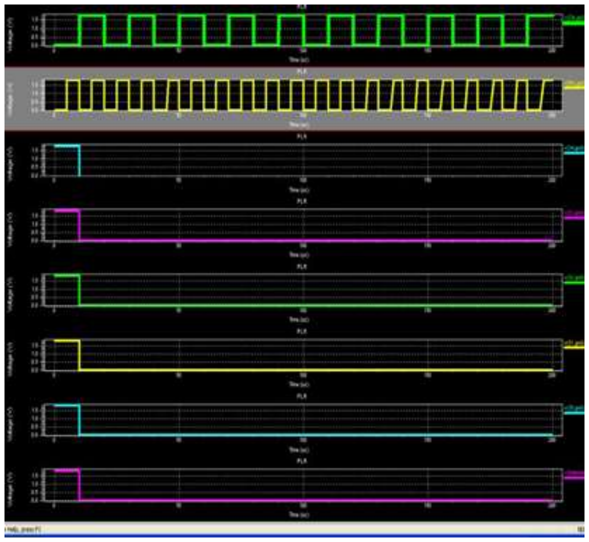


Fig. 6: Result of the Faulty Output PLR Detects the Faults

Fault free from the output of the signal if it error means the watch dog timer indicates the error hang indefinitely. If we introduce fault in the design and analysis a transient fault occurs if we are given, the input WL=0 in memory, SRAM there is no input signal is passed through shift register because In SRAM if the word line WL high means, then only we get a fault free output due to that we will receive a memory output through the view of shift register get a faulty output.

V. Conclusion

This paper motivates the necessity for software transient fault tolerance for general purpose microprocessors and proposed PLR as an attractive alternative in emerging multi core processors. By providing redundancy at the process level, PLR leverages to freely schedule the processes to all available hardware resources. In addition, PLR can be deployed without modifications to the application, OS, underlying hardware

A real PLR supports single-threaded applications are presented and evaluated for fault coverage and performance. Fault injection experiments prove that PLR's software-centric fault detection model effectively detects faults that safely ignoring benign faults. A software implemented transient fault tolerance techniques. Nowadays we utilize general purpose hardware with multi cores. To upgrade PLR performs upon existing software transient fault tolerance techniques and takes a step toward enabling software fault tolerant solutions with comparable performance to hardware techniques.

References

[1] Alex Shye, Student Member, IEEE, Joseph Blomstedt "PLR: A Software Approach to Transient Fault Tolerance for Multi core Architectures" IEEE transaction on dependable and secure computing.
[2] S.E. Michalak et al., "Predicting the Number of Fatal Soft Errors in Los Alamos National Laboratory's ASC Q Supercomputer," IEEE Trans. Device and Materials Reliability, vol. 5, no. 3, pp. 329-335, Sept. 2005.
[3] A. Shye, T. Moseley, V.J. Reddi, J. Blomstedt, and D.A. Connors, "Using Process-Level Redundancy to Exploit Multiple Cores for Transient Fault Tolerance," Proc. 37th Int'l Conf. Dependable Systems and Networks (DSN '07), June 2007.
[4] Iwagaki , T ,;nakaso , t ohkubo , r ; Ichihara ,h " scheduling algorithm in data path synthesis for long duration transient fault tolerance"IEEE transaction on defect and fault tolerance in VLSI technolgy volume oct 2014.
[5] G.A. Reis et al., "SWIFT: Software Implemented Fault Tolerance, Proc. Int'l Symp. Code Generation and Optimization (CGO), 2005.
[6] C. Weaver et al., "Techniques to Reduce the Soft Error Rate o a High-Performance Microprocessor," Proc. 31st Int'l Symp. Computer Architecture (ISCA), 2004.
[7] S.K. Reinhardt and S.S. Mukherjee, "Transient Fault Detection via Simultaneous Multithreading," Proc. 27th Ann. Int'l Symp. Computer Architecture (ISCA), 2000.
[8] A. Shye, T. Moseley, V.J. Reddi, J. Blomstedt, and D.A. Connors," Using Process-Level Redundancy to Exploit Multiple Cores for Transient Fault Tolerance," Proc. 37th Int'l Conf. Dependable Systems and Networks (DSN '07), June 2007.
[9] N. Oh et al., "Error Detection by Duplicated Instructions in Super- Scalar Processors," IEEE Trans. Reliability, vol. 51, no. 1, Mar. 2002.
[10] K. Sundaramoorthy, Z. Purser, and E. Roten burg, "Slipstream Processors: Improving Both Performance and Fault Tolerance"Proc. Ninth Int'l Conf.Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2000.
[11] R.C. Baumann, "Soft Errors in CommercialmSemiconductor Technology: Overview and Scaling Trends," IEEE 2002 Reliability Physics Tutorial Notes, Reliability Fundamentals, pp. 121_01.1-121_01.14, Apr. 2002
[12] C. Weaver et al., "Techniques to Reduce the Soft Error Rate of a High-Performance Microprocessor," Proc. 31st Int'l Symp. Computer Architecture (ISCA), 2004.
[13] Diego Montezanti1, Enzo Rucci1, Dolores Rexachs2, Emilio Luque2, Marcelo Naiouf1 and Armando De Giusti1, 3" A tool for detecting transient faults in execution of parallel scientific applications on multi core clusters"
[14] S. Hareland et al., "Impact of CMOS Scaling and SOI on Software Error Rates of Logic Processes," VLSI Technology Digest of Technical Papers, 2001.
[15] T. Karnik et al., "Scaling Trends of Cosmic Rays Induced Soft Induced Soft Errors in Static Latches beyond 0.18," VLSI Circuit Digest of Technical Papers, 2001.