A Multi-Core Fault Tolerance Approach Based on Coded-Processing

Lukas Osinski, Ralph Mader, Jens Harnisch, Jürgen Mottok

A Multi-Core Fault Tolerance Approach Based on Coded Processing

Lukas Osinski*, Ralph Mader†, Jens Harnisch‡ and Jürgen Mottok*
*Laboratory for Safe and Secure Systems - LaS^3
Technical University of Applied Sciences Regensburg, Germany,
{lukas.osinski, juergen.mottok}@oth-regensburg.de
†Continental AG, Regensburg, Germany,
ralph.mader@continental-corporation.com
‡Infineon AG, Munich, Germany,
jens.harnisch@infineon.com

***Abstract*—Development trends for computing platforms moved from increasing the frequency of a single processor to increasing the parallelism with multiple cores on the same die. Multiple cores have strong potential to support cost-efficient fault tolerance due to their inherent spatial redundancy. This work makes a step towards software-only fault tolerance in the presence of permanent and transient hardware faults. Our approach utilizes software-based spatial triple modular redundancy and coded processing on a shared memory multi-core controller. We evaluate our approach on an Infineon AURIX TriBoard TC277 and provide experimental evidence for error resistance by fault injection campaigns with an iSystem iC5000 On-chip Analyzer.**

***Index Terms*—Fault Detection, Triple Modular Redundancy (TMR), Fault Tolerance, Multi-Core, Coded Processing, Shared Memory, Spatial Redundancy**

I. Introduction

Development trends for computing platforms moved from increasing the frequency of a single processor to increasing the parallelism with multiple cores on the same die [1]. Although, chip-multiprocessors (CMP) present new development challenges, they have strong potential to support cost-efficient fault tolerance due to their inherent spatial redundancy. Suitable approaches could counteract the rising frequency of transient and permanent errors in memories [2] and CPUs [3]. Fault tolerance requires at least error detection and recovery [4]. In gerneal, detection and recovery from random hardware faults is realized by combinations of information redundancy (e.g. ECC), temporal redundancy (e.g. rollback) and spatial redundancy (e.g. triple modular redundancy) [5]. Classic and effective hardware-based approaches for detecting random hardware faults exploit fully replicated cycle-by-cycle synchronized hardware components (e.g. lockstepping) or specifically hardened hardware components. However, these approaches introduce higher costs and do not allow a flexible program execution environment where legacy binary code and redundant code can co-exist depending on the required level of reliability. Software-based approaches could lower costs and increase flexibility by achieving fault tolerance via software-only methods. However, in spite of experimental studies clearly indicating the occurrence of permanent and intermittent errors, most concepts assume only transient errors.

The authors gratefully acknowledge the financial funding from the Bayerische Forschungsstiftung (BayFor), research initiative FORMUS^{3}IC "Multi-Core safe and software-intensive Systems Improvement Community" under funding code AZ-1165-15.

II. Our Contribution

We present a software-based redundancy approach which provides resilience against soft errors and makes a step towards fault tolerance in case of permanent random hardware faults at application level. Our approach utilizes software-based triple modular redundancy and coded processing while taking in the advantage of the inherent spatial redundancy of CMPs. The key contributions of this paper are:

- A performance optimized-software only fault tolerance approach for shared memory multi-core controller based on coded processing
- A fault tolerance approach which attains real-time requirements by utilizing the inherent spatial redundancy of CMPs
- A conept towards detection (and recovery) of permanent hardware faults

III. Software-based fault tolerance approach

Coded processing is an encoding scheme based on the theory of arithmetic codes and is in general limited to integer arithmetic. Arithmetic codes exploit information redundancy, i.e., additional bits are required to store an encoded integer. There exist different strategies and coding rules for the implementation of arithmetic encoding which differ mainly in their supported arithmetic operations and their coverage with regard to the fault model. The simplest representative of arithmetic codes is called AN-encoding. With AN-encoding, an integer n is encoded by multiplying it by a constant A. The resultant integer $\hat{n} = A * n$ is called a code word. If a hardware error alters the code word $\hat{n}$, it becomes an invalid word with high probability. If $\hat{n}$ still represents a code word, $\hat{n}\ mod\ A$ equals 0; if the result of the operation is unequal to 0, a hardware error is detected. To decode the code word, a division $\hat{n}/A$

TABLE I
STATE OF THE ART - GAP ANALYSIS

	Source-Level	Task-Level	Proposal
Fault Model			
Transient	✓	✓	✓
Intermittent	✓	X	✓
Permanent	✓	X	✓[2]
No Protection Gaps	X	✓	✓
Realtime Requirements	X	✓[1]	✓
Negligible Overhead	X	✓[3]	✓
Fault Diagnosis	X	X	✓
Fault Tolerance			
Error Detection	✓	✓	✓
Recovery	X	✓[1]	✓[2]
Spatial redundancy	X	X	✓

[1] Increase of the Worst Case Execution Time due to rollback recovery and sequential task exeuction
[2] Conceptual work in progress
[3] In terms of execution time overhead

is performed. AN-encoding supports all relevant operations including the division. However, some operations (e.g. bit-wise operations) require more sophisticated implementations which can hamper performance and/or require intermediate decoding of operands. Furthermore, with AN-encoding control flow errors and erroneous or lost data access is not detected. To detect these types of errors, variants of AN-encoding were developed, namely ANB- and ANBD-encoding. ANB-encoding introduces the static signature B which allows the determination of the identity of the data [6]. As a result, swapped data/operations and control flow errors can be detected. As a drawback ANB-encoding loses the support for the division operation due to the associated complex signature calculation. To allow the detection of a lost update, i.e., in case a store operation was omitted, the additional dynamic signature D is introduced. ANBD-encoding provides very high fault coverage, but if applied on source-level, it incurs very high penalties regarding execution time since more computational effort is required for execution the encoded operations. Execution time penalties range form 6x (AN-encoding) [7] up to 250x (ANBD-encoding) [8] compared to unencoded execution. This fact makes encoding on source-level - in combination with further drawbacks (see Table 1) impractical for most use cases with real-time requirements.

A key mechanism to achieve fault tolerance, i.e., error detection and recovery within a system, is redundancy respectively the replication of components in e.g. hardware: processors, memory; or software: entire programs or parts of it [5]. A widely used paradigm for fault tolerance is represented by the triple modular redundancy (TMR) pattern. TMR uses three identical elements which perform the same operation. After completing the operation, a voting element compares the three results and selects the correct one by majority [5]. TMR is - besides the higher costs in terms of resources - a powerful approach because it not only detects that a fault occurred,

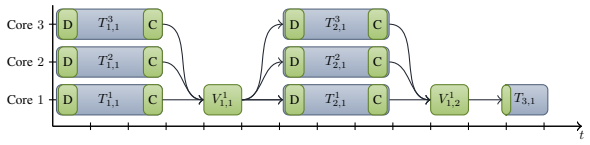

Fig. 1. Spatial Triple Modular Redundancy in Combination with Coded Processing

but is also capable of identifying the faulty component [5]. Furthermore, it contributes to a higher system availability, since the system can continue the execution by masking the faulty element. The weak spot of TMR, if realized completely in software, is the majority voter. It depicts a single point of failure (SPOF) and therefore has to meet high reliability requirements. To eliminate this SPOF, Ulbrich et al. describe a Combined Redundancy (CoRed) approach for single-core systems [9]. CoRed protects an application by software-based triple modular redundancy on task-level and applying ANBD-encoding to increase the reliability of the software voting. Computational tasks are executed sequentially on a single core, followed by a voting task. On task entrance, the previously encoded values are decoded to reduce execution time penalties - caused by encoded operations - to a negligible level. On task exit the values are encoded and the voting is performed. During the time-span where values are processed unencoded, the execution is protected by redundant task execution. The voting is performed on encoded values to enable error detection during the voting process and fault tolerance by subsequent rollback recovery. Experimental evaluation by fault injection showed full single and dual bit-flip coverage [9]. Drawbacks of this approach are e.g. increased Worst Case Execution Time due to temporal redundancy (sequential task execution) and the lack of permanent fault coverage due to a single-core approach (see Table 1).

In our approach (see Figure 1), we extend the CoRed scheme, to exploit the inherent spatial redundancy of multi-core systems to meet realtime requirements and make a step towards permanent fault coverage. Similar to CoRed, task values are decoded on task entrance, encoded on task exit and voting is performed on encoded values. We distinguish ourselves from CoRed by executing the computational tasks on a multi-core controller in parallel. This minor adaption leads to extensive changes in the encoding and execution scheme:

- Static and dynamic signature handling in tasks and voter
- Memory partitioning for storage of encoded values
- Introduction of fault diagnosis for tasks and voters
- Recovery mechanism for errors in the voting procedure

In the full paper we will describe our concepts as well as the necessary adaptions and mechanisms in detail. Furthermore, extensive evaluation results will be presented.

IV. EVALUATION

To evaluate the effectiveness of our approach we realized it on an Infineon AURIX TriBoard TC277 with Erika Enterprise 2.7.0 running. In the full paper we provide experimental

evidence for the fault detection and recovery capabilities by presenting extensive results of fault injection campaigns performed with an iSystem iC5000 On-chip Analyzer. The On-chip Analyzer injects single bit fault in CPU registers and memory cells which trigger the software-level symptoms of our fault model. For demonstration purposes our approach will be applied to an automotive architecture.

REFERENCES

[1] G. Macher, A. Höller, E. Armengaud, and C. Kreiner, "Automotive embedded software: Migration challenges to multi-core computing platforms," in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, Jul. 2015, pp. 1386–1393.

[2] E. B. Nightingale, J. R. Douceur, and V. Orgovan, "Cycles, Cells and Platters: An Empirical Analysisof Hardware Failures on a Million Consumer PCs," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: ACM, 2011, pp. 343–356.

[3] M.-L. Li, P. Ramachandran, S. K. Sahoo, S. V. Adve, V. S. Adve, and Y. Zhou, "Understanding the Propagation of Hard Errors to Software and Implications for Resilient System Design," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XIII. New York, NY, USA: ACM, 2008, pp. 265–276.

[4] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[5] K. Echtle, *Fehlertoleranzverfahren*, 1990.

[6] U. Wappler and M. Muller, "Software Protection Mechanisms for Dependable Systems," in *2008 Design, Automation and Test in Europe*, Mar. 2008, pp. 947–952.

[7] D. Kuvaiskii and C. Fetzer, "Delta-Encoding: Practical Encoded Processing," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Jun. 2015, pp. 13–24.

[8] U. Schiffel, "Hardware error detection using AN-codes," 2010.

[9] P. Ulbrich, "Ganzheitliche Fehlertoleranz in eingebetteten Softwaresystemen," Ph.D. dissertation, 2014.