Software-based Fault Tolerance – Mission (Im)possible?

Peter Ulbrich

The 29th CREST Open Workshop on Software Redundancy
November 18, 2013
Soft Errors – A Growing Problem

- **Soft-Errors (Transient hardware faults)**
  - Induced by e.g., radiation, glitches, insufficient signal integrity
  - Affecting microcontroller logic
Soft Errors – A Growing Problem

- **Soft-Errors (Transient hardware faults)**
  - Induced by e.g., radiation, glitches, insufficient signal integrity
  - Affecting microcontroller logic
Soft Errors – A Growing Problem

- Soft-Errors (Transient hardware faults)
  - Induced by e.g., radiation, glitches, insufficient signal integrity
  - Affecting microcontroller logic

- Future hardware designs: more performance and parallelism
  → On the price of being less and less reliable

Peter Ulbrich – ulbrich@cs.fau.de
Soft Errors – A Growing Problem

- **Soft-Errors (Transient hardware faults)**
  - Induced by e.g., radiation, glitches, insufficient signal integrity
  - Affecting microcontroller logic

- Future hardware designs: more **performance** and **parallelism** → **On the price of being less and less reliable**

---

**Toyota Acceleration Case**

- **Electronic throttle control system** (2005 Camry)

  "Toyota claimed the 2005 Camry's main CPU had error detecting and correcting RAM. *It didn't.*"  
  1. **Unintended acceleration potentially involving 261 deaths**
  2. **Experts identified soft errors as possible cause**

  1. US News, Mar 17, 2010
Software-Based Fault Tolerance

- Software-based redundancy
  - Triple Modular Redundancy (e.g., recommended by ISO 26262)
  - Selective and adaptive
  - Resource efficient
Software-Based Fault Tolerance

Software-based redundancy
- **Triple Modular Redundancy** (e.g., recommended by ISO 26262)
  - Selective and adaptive
  - Resource efficient

Single points of failure
- Interface and Majority Voter
- Allowing for **Silent Data Corruptions** (SDC)
  - Replication is impossible!
威胁适用性 – 任务失败了？

- 三重模冗余可靠性

\[ R_{TMR} = R_{Voter} \cdot R_{2-of-3} \]
Threats to Applicability – Mission failed?

- **Triple modular redundancy reliability**
  
  \[ R_{TMR} = R_{Voter} \cdot R_{2-of-3} \]

- **Voting on unreliable hardware?**
  - Very small → residual error probability?
  - Risk analysis → inherently complex (no random error distribution! [4])
Triple modular redundancy reliability

\[ R_{TMR} = R_{Voter} \cdot R_{2-of-3} \]

Voting on unreliable hardware?
- Very small → residual error probability?
- Risk analysis → inherently complex (no random error distribution! [4])

→ Dealbreaker for software-based TMR
Research Aims

Safety-Critical System

- Sensors
- Interface
- Replica 1
- Replica 2
- Replica 3
- Majority Voter
- Actuators

- Isolation domain
- Sphere of redundancy (SOR)
Research Aims

- Eliminate single points of failure
Eliminate single points of failure
Constrain residual error probability
Research Aims

- Eliminate single points of failure
- Constrain residual error probability
- Dependability as a resource efficient option
Agenda

- Introduction
  - The Combined Redundancy approach (*CoRed*)
    - Holistic protection – eliminating single points of failure
    - Arithmetic coding
    - Dependable voting
  - Constraining residual error probability
    - From coding theory to application – lessons learned
    - Finding appropriate parameters
    - Circumvent implementation pitfalls
- Evaluation
  - Use case
  - Experimental setup
  - Fault-injection results
- Conclusion
The **Combined Redundancy Approach (CoRed)**

\[
\text{TMR} + \left\{ \right. \} 
\]
CoRed Overview – Holistic Protection Approach

- **The Combined Redundancy Approach (CoRed)**
  
  TMR + \{ Data-flow encoding \}

Peter Ulbrich – ulbrich@cs.fau.de
CoRed Overview – Holistic Protection Approach

- The **Combined Redundancy Approach (CoRed)**
  - TMR + \{ Data-flow encoding \\
    Dependable voters \}

Peter Ulbrich – ulbrich@cs.fau.de
**CoRed Overview – Holistic Protection Approach**

- **The Combined Redundancy Approach (CoRed)**
  - TMR + \{ Data-flow encoding, Dependable voters \}

- **Holistic protection approach for control applications**
  - Input to output protection
    - 1 Reading inputs → 2 Processing → 3 Distributing outputs
Eliminating Input and Output Vulnerabilities

**Arithmetic Codes → ANBD Code**
- **Based on VCP [5]**
- **Data integrity:** Key
- **Address integrity:** Per variable signature
- **Outdated data:** Timestamp

\[ v' = A \cdot v + B + D \]
Eliminating Input and Output Vulnerabilities

- **Arithmetic Codes → ANBD Code**
  - Based on VCP [5]
  - **Data integrity:** Key
  - **Address integrity:** Per variable signature
  - **Outdated data:** Timestamp

\[ v' = A \cdot v + B + D \]

- **Set of arithmetic operators** (+, -, *, =, …)
  - Checksum vs. Arithmetic code (AN code)
  - AN Code → Encoded data operations
  - Enabler for dependable voter

\[ Z = X \odot Y \]
**CoRed Dependable Voter – Basics**

- **CoRed Dependable Voter**
  - **Input**: variants ($X'$, $Y'$, $Z'$)
  - **Output**: Equality set ($E$) and encoded winner ($W$)
  - **No decoding necessary**

- **Control-flow signatures**
  - **Static signature** (expected value): Compile-time
    - Used as return value $E$
  - **Dynamic signature** (actual value): Runtime, computed from variants
    - Applied to winner $W$
  - **Validation**: Subsequent check (decode)
Agenda

- Introduction
- The Combined Redundancy approach (CoRed)
  - Holistic protection – eliminating single points of failure
  - Arithmetic coding
  - Dependable voting
- Constraining residual error probability
  - From coding theory to application – lessons learned
  - Finding appropriate parameters
  - Circumvent implementation pitfalls
- Evaluation
  - Use case
  - Experimental setup
  - Fault-injection results
- Conclusion
From Coding Theory to Application

Safety-Critical System

- $R_I = 1$
- $R_V = 1$

CoRed Interface

Isolation domain

Sphere of redundancy (SOR)

Decoded_Static() {
  TAssert(_B > 0);
  assert(check());
  return (vc-_B-D)/_A;
};

Arithmetic coding operations

Think binary

Mathematics

Know your compiler & architecture

C/C++

Assembler

Peter Ulbrich – ulbrich@cs.fau.de
Constraining residual error probability

- Coding theory
  - Data word + redundant information = code word
  - Fault detection → distance between code words

\[ v' = A \cdot v + B + D \]
Constraining residual error probability

- **Coding theory**
  - Data word + redundant information = code word
  - Fault detection → distance between code words

- **Residual error probability**
  - Chance for code-to-code word mutation
  - Fundamental property for fault tolerance mathematics

\[ v' = A \cdot v + B + D \]

\[ P_{sdc} = \frac{\text{valid code words}}{\text{possible code words}} \approx \frac{1}{A} \]
Constraining residual error probability

- **Coding theory**
  - Data word + redundant information = code word
  - Fault detection → distance between code words

- **Residual error probability**
  - Chance for code-to-code word mutation
  - Fundamental property for fault tolerance mathematics

\[
\nu' = A \cdot \nu + B + D
\]

\[
P_{sdc} = \frac{\text{valid code words}}{\text{possible code words}} \approx \frac{1}{A}
\]

- Pitfall 1: Mapping Code to Binary
  - Resulting in the following patch for the

\[
\text{CoRed} : 58\,659, 59\,665, 63\,157, 63\,859, \text{and } 63\,877.
\]

\[
\nu' = A \cdot \nu + B + D
\]

\[
P_{sdc} = \frac{\text{valid code words}}{\text{possible code words}} \approx \frac{1}{A}
\]

\[
\text{v} = A \cdot \text{v} + B + D
\]
Choosing Keys and Signatures

- **Mathematics:** prime numbers
  - Intuitively plausible
  - Literature: little help to find suitable $A$s

- **Practitioner’s approach:** min. Hamming distance
  - Distance ($d$) between code words (# unequal bits)
  - $d-1$ bit error detection capabilities

- **Brute force**
  - $1.4 \times 10^{14}$ experiments for all 16 bit $A$s
    
    | $A$   | $d_{min}$ | #errors detectable |
    |-------|----------|---------------------|
    | 58,368| 2        | 1                   |
    | 58,831| 3        | 2                   |
    | 58,659| 6        | 5                   |
  - The bigger the better is misleading!
Fault-simulation → entire fault-space
  - Each and every $A$, $\nu$ and fault pattern
  - $6.5 \times 10^{16}$ experiments for 16 bit $A$s and 1-8 bit soft errors

→ Excess of predicted residual error probability

→ Violation of predicted fault-detection capabilities
- Binary representation of code words
  - Coding theory is unaware of machine word sizes
    → Dangerous over- and underflow conditions
  - Extended AN code (EAN) implementation

→ Compliance with coding theory!

- Improved code reliability \((A = 25I)\)
  - Predicted \(3 \times 10^{-3}\)
  - Common implementation [4] \(\approx 1.3 \times 10^{-2}\)
  - EAN implementation \(\approx 1.5 \times 10^{-5}\)

→ Improvement by orders of magnitude!
Know your Compiler and Architecture

- On target fault-injection → **entire fault space**
  - Each and every register, flag, instruction and execution path
  - FAIL* fault injection framework [6]

→ **Violation of predicted fault-detection capabilities**

- Architecture specifics
  - Absence of compound **test-and-branch** (e.g., IA32 architecture)
  - Control-flow information is stored in single bit
    → Redundancy is lost
    → Additional range checks

- Undefined Execution Environment
  - **Zombie values** → leaking from caller to voter function
  - **Compiler laziness** leaves encoded values in registers
    → Isolation assumptions violated
    → Cleaning local storage restores isolation

→ **Tight feedback loop with fault-injection experiments**
Agenda

- Introduction
- The Combined Redundancy approach (*CoRed*)
  - Holistic protection – eliminating single points of failure
  - Arithmetic coding
  - Dependable voting
- Constraining residual error probability
  - From coding theory to application – lessons learned
  - Finding appropriate parameters
  - Circumvent implementation pitfalls
- Evaluation
  - Use case
  - Experimental setup
  - Fault-injection results
- Conclusion
Categories: Fail Silent, Masked, Hardware Detected, EAN-Code, Control-Flow, Silent Data Corruption

Outcome: 401,592 experiments
Effective: 67,617 errors
- **Redundant execution campaign (Interface)**
  - Total: ~45,000 Errors
Evaluation – Experimental Results (1)

- Redundant execution campaign (Interface)
  - Total: ~45,000 Errors
  - Unprotected: Suffers from **3,622 corruptions**!
Redundant execution campaign (Interface)

- Total: ~45,000 Errors
  - Unprotected: Suffers from 3,622 corruptions!
  - TMR: Suffers from 71 corruptions!
Evaluation – Experimental Results (1)

- **Redundant execution campaign (Interface)**
  - Total: ~45,000 Errors
  - **Unprotected**: Suffers from 3,622 corruptions!
  - **TMR**: Suffers from 71 corruptions!
  - **CoRed**: Remaining corruptions are covered → 0 corruptions
Evaluation – Experimental Results (2)

- **Voter campaign**

![Diagram showing data addresses and various types of corruption detection.](image-url)
### Evaluation – Experimental Results (2)

#### Voter campaign

- **Plain voter:**
  - Total ~11,000
  - 2,465 masked
  - 7,245 retry
  - 1,223 corruptions
Voter campaign

Plain voter:
- Total ~11,000
  - 2,465 masked
  - 7,245 retry
  - 1,223 corruptions

CoRed Dependable Voter:
- Total ~26,000
  - 1,228 masked
  - 24,682 retry
  - 0 corruptions
**Evaluation – Experimental Results (2)**

**Voter campaign**
- **Plain voter:**
  - Total ~11,000
  - 2,465 masked
  - 7,245 retry
  - 1,223 corruptions
- **CoRed Voter:**
  - Total ~26,000
  - 1,228 masked
  - 24,682 retry
  - 0 corruptions

**Evaluation – Overhead**

- **Overhead Analysis**
  - I4Copter Flight-Control: 7.1% overhead (compared to plain TMR)

- **Selectivity**
  - I4Copter system CPU utilisation: 41%
    - Full replication impossible, CPU: 120%
  - Mission-critical replication of flight control
    - possible with CoRed, CPU: 60%
Eliminate single points of failure [1]
Eliminate single points of failure [1]
- TMR + Encoding: Combined Redundancy approach
- Key feature: CoRed Dependable Voter
Conclusion

Eliminate single points of failure [1]
- TMR + Encoding: Combined Redundancy approach
- Key feature: CoRed Dependable Voter

Constrain residual error probability [2]
- Parameterisation guidelines: choosing the right A
- Binary aware implementation: complying with coding theory
- Factor 1000 improvement

Dependability as a resource efficient option
- Only 7.1% overhead (flight control example)
Eliminate single points of failure [1]
- TMR + Encoding: Combined Redundancy approach
- Key feature: CoRed Dependable Voter

Constrain residual error probability [2]
- Parameterisation guidelines: choosing the right A
- Binary aware implementation: complying with coding theory
- Factor 1000 improvement

Dependability as a resource efficient option
- Only 7.1% overhead (flight control example)

→ Bullet-proof software-based fault tolerance is possible
Thank you!


(3) P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi, “Modelling the effect of technology trends on the soft error rate of combinational logic,” in DSN ’02: Proceedings of the 2002 International Conference on Dependable Systems and Networks

