Comprehensive Analysis of Software-Based Fault Tolerance with Arithmetic Coding for Performant Encoding of Integer Calculations

COMS 415 Presentation

Group 17: Ben Smith, Gurumanie Singh Dhiman, Jaret Van Zee, Jonathan Tan

04/15/2025

IOWA STATE UNIVERSITY

Problem Statement

- Hardware unreliability is an increasing problem
  - Transient Faults : Faults that occur for a short time
  - Permanent Faults : Faults that continuously affect the system
  - Intermittent Faults : Faults that occur for a short time but periodically
- Traditionally, fault tolerance is handled by specialized hardware.
- Fault tolerance is required by standards like IEC-61508.
- This paper analyzed different methods for Arithmetic Coding for running fault tolerance on commercial off-the-shelf (COTS) hardware.
  - Performance (detection capability)
  - Overhead

Approaches to Redundancy

- Redundancy Types:
  - Hardware : Extra physical components (expensive & inflexible)
  - Software : Duplicate code versions (costly to develop)
  - Time : Repeating operations (misses permanent faults)
  - Information : Arithmetic coding (potentially costly computations)
- Why Information Redundancy?
  - More dynamic
  - Adds error detection to the data itself (through encoding)

What is Arithmetic Coding? A simple example

- Let there be a coding constant $A = 3$
- Let there be two values $v = 5$ and $u = 7$
- Arithmetic Coding encodes them like so:

$$v_c = Av = 3 \times 5 = 15$$
$$u_c = Au = 3 \times 7 = 21$$

- If we want to calculate the addition of v and u, we will take

$$v_c + u_c = 15 + 21 = 36$$

  which is divisible by A, hence **no error**
- If there was an error (like a bit flip) that caused v' to be 14,

$$v_c + u_c = 35$$

  which is not divisible by A, hence an **error has occurred**
- This is also how AN encoding works

Different Types of Arithmetic Coding

- AN : Multiply integer values by a constant A, i.e., $\mathrm{v_c} = Av_0$
- Residue : Forms a residuum of a value, i.e., $\mathrm{v_c} = A - (v \% A)$
- Complement : Use the signed representation, e.g., 1's and 2's complement

Different Types of Arithmetic Coding's Capability

	1's Complement		2's Complement		AN		Residual	
	Unsign.	Sign.	Unsign.	Sign.	Unsign.	Sign.	Unsign.	Sign.
Arith.								
+	Adapt.	Adapt.	Direct	Direct	Direct	Direct	OF corr.	OF corr.
-	Adapt.	Adapt.	Direct	Direct	UF corr.	Direct	UF corr.	UF corr.
×	No	No	Adapt.	Adapt.	Adapt.	Adapt.	OF corr.	OF corr.
/	No	No	No	Adapt.	Adapt.	Adapt.	No	No
mod	No	No	No	Direct	Direct	Direct	No	No
Comp.								
==	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.
!=	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.
<	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	No	No
>	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	No	No
<=	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	No	No
>=	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	Adapt.	No	No

Problems with Existing methods

- AN codes are widely used but inefficient for 64-bit operations
  - They double bit-width ($64 \rightarrow 128$ bits)
  - Less optimized for 64-bit processors
- Residue codes can't uniquely decode values.
- Complement-based codes (like one's complement) are overlooked in literature but are promising.
- This paper proposes:
  - A comprehensive strategy to evaluate arithmetic codes
  - Identify the best performer for 64-bit datatype (Ones' complement)

Fault Detection Capability of Different Arithmetic Coding

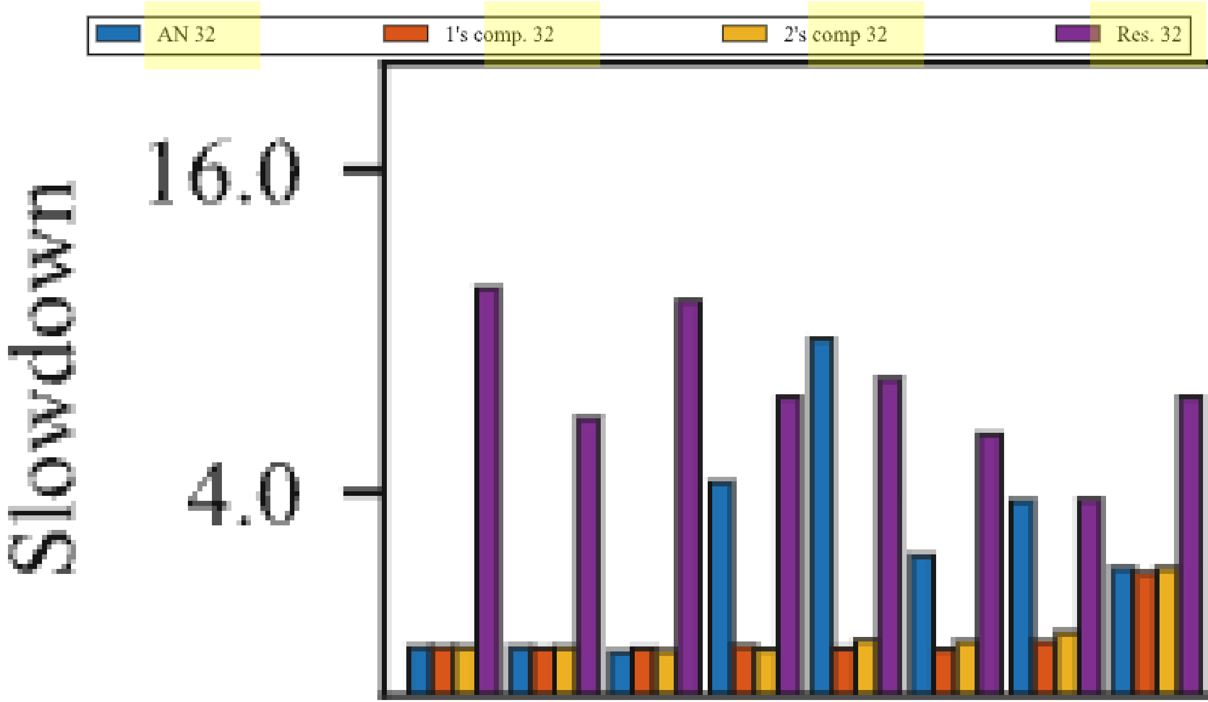
- Best constant (A)
- Distance for separate codes (C_d)
- Average Hamming distance ($\overline{H_d}$)
- Normalized Hamming distance ($\frac{\overline{H_d}}{\max(H_d)}$)

	Coding	1's comp.	2's comp.	AN sep.	Residue	Inv. residue
	A	–	–	255	1	255
(Higher the better)	C_d	8	0	7	0	0
(Higher the better)	$\overline{H_d}$	8.00	6.01	10.02	4.00	7.97
(Closer to 1 the better)	$\frac{\overline{H_d}}{\max H_d}$	1	0.75	0.62	0.5	0.996

Overhead of Different Arithmetic Coding

- Y-axis: Slowdown= $\frac{t_{encoded}}{t_{native}}$

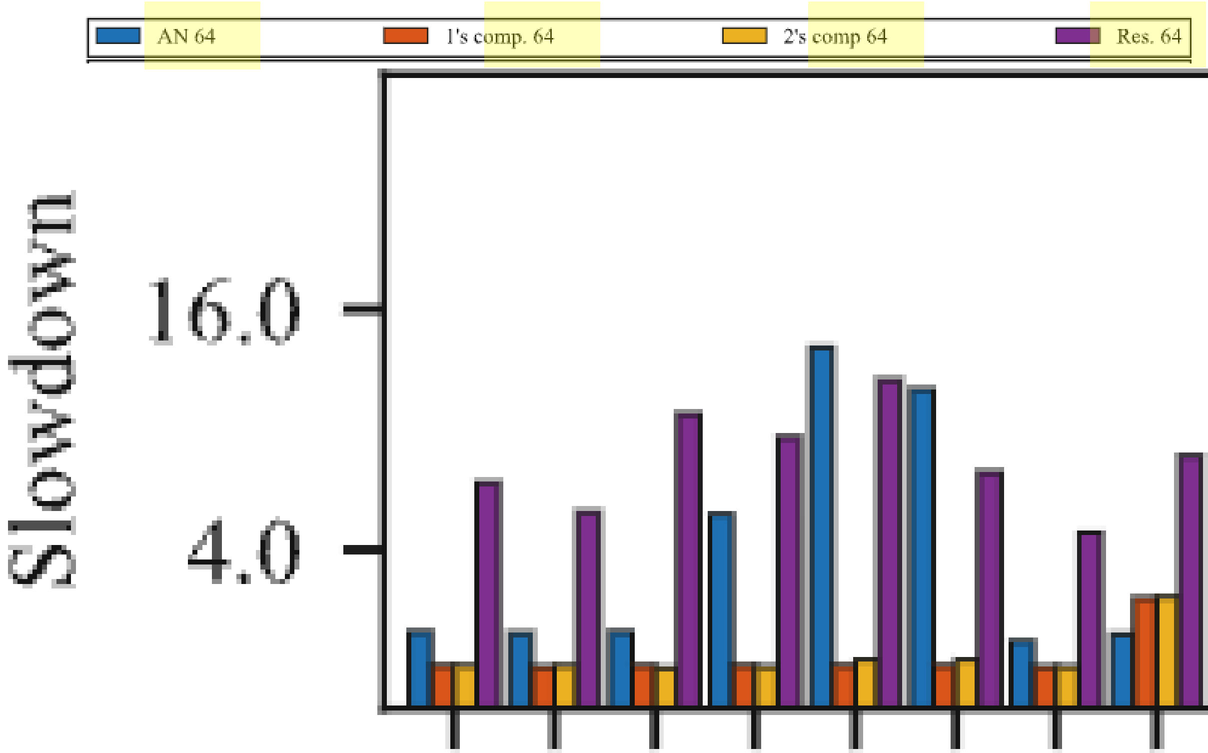
32-bit encoding:



Overhead of Different Arithmetic Coding

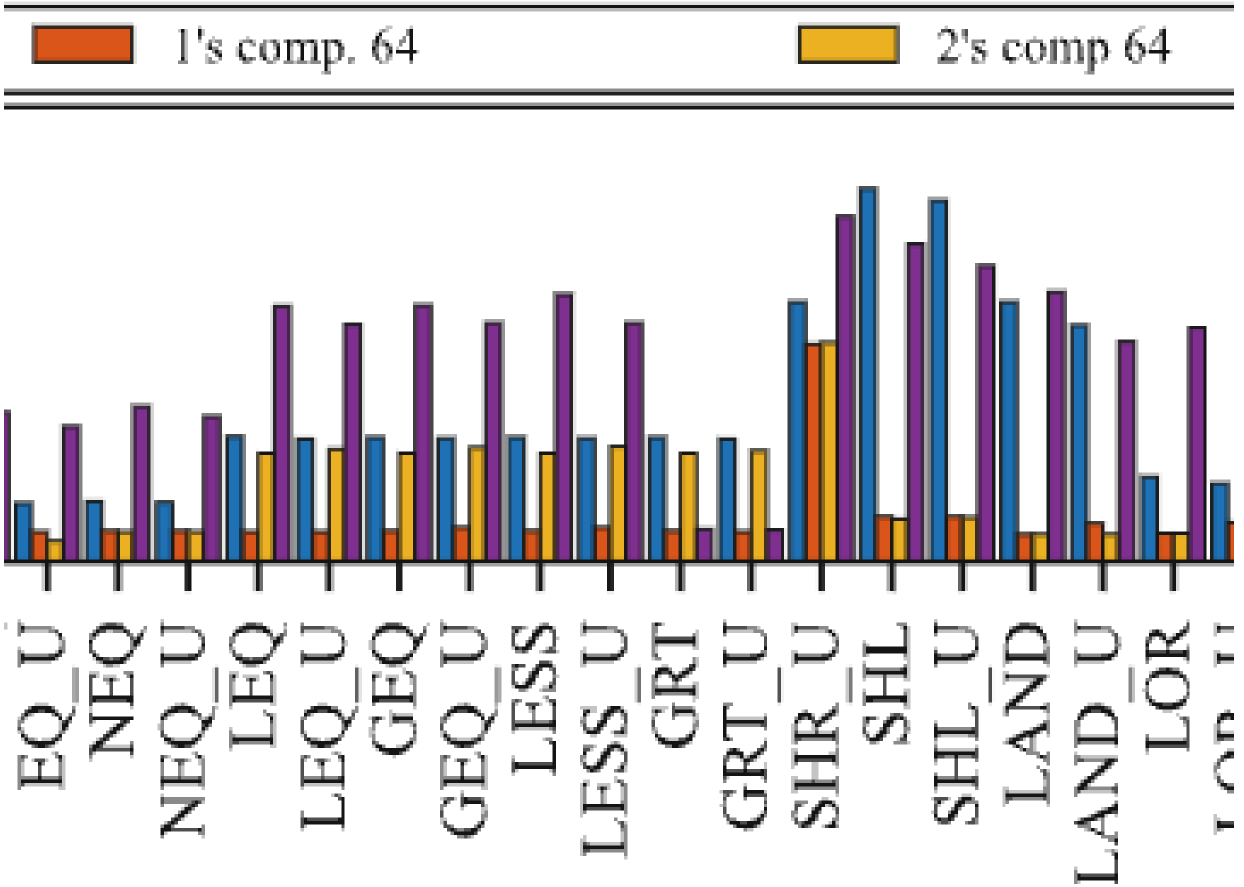
- Y-axis: Slowdown= $\frac{t_{encoded}}{t_{native}}$
- 64-bit AN incurs higher overhead due to the doubling of bit width

64-bit encoding:



Overhead of Different Arithmetic Coding

- Analysis:
  - Res-coding in general incurs high overhead
  - 1's and 2's are similar in general
  - For some operations, 1's incurs less overhead than 2's



One’s Complement

- The best option
- Detects 100% of injected faults when tested
- Outperforms AN and Residue codes in detecting permanent and transient faults
- Achieved using COTS hardware
- Only a 2.2x (for 32-bit) and a 2.3x (for 64 bit) mean slowdown

Choosing the Right Arithmetic Encoding

- The 7 Steps:
  1. Do you need arithmetic coding? (Only if math operations are involved)
  2. Identify fault types. (Permanent, transient, intermittent)
  3. Model software-level errors. (Exchanged operands, lost updates, etc.)
  4. Measure detection strength. (Use code distance, fault injection)
  5. Decide where to apply coding. (Prefer compile-time, program-level)
  6. Build a check mechanism. (Accumulators, output validation)
  7. Evaluate performance. (Real algorithms, not just isolated ops)

Conclusion

- This paper analyzes different arithmetic coding schemes for fault-tolerance
- Analyzed AN, Residue, and Complement coding schemes
- Identified that ones' complement is superior due to:
  - Fault detection capabilities
  - Low overhead
- Strengths:
  - The paper applies to big industries -> Small changes in hardware/software helps with reducing the big issues?
- Weaknesses:
  - Hard to integrate into already up and running systems

References

- Fischer, Marc, et al. "Comprehensive Analysis of Software-Based Fault Tolerance With Arithmetic Coding for Performant Encoding of Integer Calculations." Lecture notes in computer science, 2022, pp. 144–57. https://doi.org/10.1007/978-3-031-14835-4_10.

Thank you for your attention!

Questions?