Δ -encoding: Practical Encoded Processing

Dmitrii Kuvaiskii
Christof Fetzer
Technische Universität Dresden
Cyber-Physical Systems: new challenges

– increased computational power demands → commodity HW?
– very high requirements for software reliability
– mixed criticality
Hardware Errors in the Wild
only one fault...
Hardware Errors in the Wild
Landscape of Solutions

fault coverage

cost-effectiveness
Landscape of Solutions

- Lockstep [DSN’05]
- AN-encoding [SAFECOMP’10]
- Redundant Multithreading [ISCA’02]
Landscape of Solutions

Lockstep [DSN'05]
AN-encoding [SAFECOMP'10]
Redundant Multithreading [ISCA'02]

SWIFT [CGO'05]
Shoestring [ASPLOS'10]
ReStore [DSN'05]
Cost-effective solutions

Common assumptions:

- RAM
- Cache
- CPU
Common assumptions:
– memory is ECC-protected
Cost-effective solutions

Common assumptions:
- memory is ECC-protected
- single transient fault
Cost-effective solutions: SWIFT

Common assumptions:
- memory is ECC-protected
- single transient fault

SWIFT [CGO'05]:
- duplicate instructions
- store only once
- insert checks before stores
Cost-effective solutions: SWIFT

Common assumptions:
- memory is ECC-protected
- single transient fault

**SWIFT [CGO’05]:**
- duplicate instructions
- store only once
- insert checks before stores
- overhead 1.4 – 2x
Cost-effective solutions: problems?

But what if:

– multi-bit error in memory
– high error rate
– design faults in HW
But what if:
- multi-bit error in memory
- high error rate
- design faults in HW

**AN encoding** [SAFECOMP'10]:
- detects transient and permanent faults
- protects CPU and memory
Dependable Solutions: AN encoding

But what if:
- multi-bit error in memory
- high error rate
- design faults in HW

**AN encoding** [SAFECOMP'10]:
- detects transient and permanent faults
- protects CPU and memory
- overhead of >10x
ECC is not enough

„Flipping Bits in Memory Without Accessing Them“, ISCA'2014
Sad Reality

ECC is not enough

„Flipping Bits in Memory Without Accessing Them“, ISCA'2014

intermittent and permanent errors do happen...

„Cycles, Cells and Platters“ from Microsoft, EuroSys'2011
Sad Reality

ECC is not enough

„Flipping Bits in Memory Without Accessing Them“, ISCA'2014

intermittent and permanent errors do happen...

„Cycles, Cells and Platters“ from Microsoft, EuroSys'2011

...and they propagate to software

„Understanding the Propagation of Hard Errors“, ASPLOS'2008
Landscape of Solutions

Delta-encoding

- AN-encoding [SAFECOMP'10]
- Redundant Multithreading [ISCA'02]
- Lockstep [DSN'05]

- SWIFT [CGO'05]
- Shoestring [ASPLOS'10]
- ReStore [DSN'05]

fault coverage

cost-effectiveness
Landscape of Solutions

- Lockstep [DSN’05]
- AN-encoding [SAFECOMP’10]
- Redundant Multithreading [ISCA’02]
- ∆-encoding
- AN codes
- duplicated instructions
- SWIFT [CGO’05]
- Shoestring [ASPLOS’10]
- ReStore [DSN’05]
**Δ-encoding: teaser**

- Performance overhead (normalized execution time)
  - SWIFT: 1.6x
  - Δ-encoding: 2.4x
  - AN encoding: 16.0x

- Fault coverage
  - 99.997% of faults detected
Outline

– Concepts
  – AN encoding: overview
  – Δ-encoding: our take on AN

– Implementation

– Performance & fault coverage
Concepts
AN encoding: overview
AN encoding: overview

\[ n = 3 \quad \text{AN-encode} \quad n \times A = 27 \]
AN encoding: overview

\[ n = 3 \]

\[ n \times A = 27 \]

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 ...

code word  invalid  code word  invalid  code word  invalid  code word
AN encoding: overview

\[ n = 3 \]

\[ n \times A = 27 \]

encode: \[ n \times A \]
AN encoding: overview

\[ n = 3 \quad \rightarrow \quad n \times A = 27 \]

encode: \[ n \times A \]

decode: \[ n' / A \]
AN encoding: overview

n = 3

\[ n \times A = 27 \]

encode:

\[
\begin{align*}
0 & \quad \text{invalid} \\
1 & \quad \text{invalid} \\
2 & \quad \text{invalid} \\
3 & \quad \text{invalid} \\
4 & \quad \text{invalid} \\
5 & \quad \text{invalid} \\
6 & \quad \text{invalid} \\
7 & \quad \text{invalid} \\
8 & \quad \text{invalid} \\
9 & \quad \text{code word} \\
10 & \quad \text{code word} \\
11 & \quad \text{code word} \\
12 & \quad \text{code word} \\
13 & \quad \text{code word} \\
14 & \quad \text{code word} \\
15 & \quad \text{code word} \\
16 & \quad \text{code word} \\
17 & \quad \text{code word} \\
18 & \quad \text{code word} \\
19 & \quad \text{code word} \\
20 & \quad \text{code word} \\
21 & \quad \text{code word} \\
22 & \quad \text{code word} \\
23 & \quad \text{code word} \\
24 & \quad \text{code word} \\
25 & \quad \text{code word} \\
26 & \quad \text{code word} \\
27 & \quad \text{code word}
\end{align*}
\]

decode:

\[ n' / A \]

check:

\[ n' \% A \]
AN encoding: overview

\[ n = 3 \xrightarrow{AN-encode} n \times A = 27 \]

Encode: \[ n \times A \]
Decode: \[ n'/A \]
Check: \[ n' \% A \]

invalid code word
Invalid code word
Invalid code word
Invalid code word
...
Concepts
Our take: Δ-encoding
Δ-encoding: bird's eye view

original

add

xor

sub

CPU
Δ-encoding: bird's eye view

original

Δ-encoded

CPU

add

xor

sub

A₁

add₁

xor₁

sub₁

A₂

add₂

xor₂

sub₂

CPU

add

xor

sub
Δ-encoding: bird's eye view

original

add
xor
sub

CPU

Δ-encoded

A1
add1
xor1
sub1

A2
add2
xor2
sub2

benefit from instruction-level parallelism

CPU
Δ-encoding: bird's eye view

Original:
- add
- xor
- sub

Δ-encoded:
- add
- xor
- sub

Accumulators:

- Accu1
- Accu2

Check:
- Check
Δ-encoding: bird's eye view

original

add

xor

sub

Δ-encoded

A1

add1

accu1

xor1

accu1

sub1

accu1

A2

add2

accu2

xor2

accu2

sub2

accu2

check

later in this talk
Δ-encoding: first try

- build on AN encoding with 2 copies

- choose $A_1$ and $A_2$ such as

\[
A_1 - A_2 = 1
\]

$A_1 = 9$ \hspace{1cm} $A_2 = 8$

hence the name Δ-encoding
Δ-encoding: first try

\[ \downarrow \text{build on AN encoding with 2 copies} \]

– choose \( A_1 \) and \( A_2 \) such as

\[
A_1 - A_2 = 1
\]

\[
\begin{align*}
A_1 &= 9 \\
A_2 &= 8
\end{align*}
\]

– then decoding is

\[
n = n'_1 - n'_2 = n \times A_1 - n \times A_2 = n \times (A_1 - A_2)
\]

hence the name \( \Delta\text{-encoding} \)
Δ-encoding: first try

\[ n = 3 \]

\[ Δ-encode \]

\[ n \times A_1 = 27 \]
\[ n \times A_2 = 24 \]
$\Delta$-encoding: first try

$n = 3$

$\Delta$-encode

$n \times A_1 = 27$

$n \times A_2 = 24$

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 ...

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 ...
\( \Delta \)-encoding: first try

\[
\begin{align*}
n & = 3 \\
\Delta \text{-encode} & \\
n \times A_1 & = 27 \\
n \times A_2 & = 24
\end{align*}
\]

encode: \( n \times A_1 \quad n \times A_2 \)
Δ-encoding: first try

\[ n = 3 \]

\[ \Delta\text{-encode} \]

\[ n \times A_1 = 27 \]
\[ n \times A_2 = 24 \]

encode:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 ...

decode:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 ...

\[ \text{encode: } n \times A_1 \quad n \times A_2 \]

\[ \text{decode: } n'_1 - n'_2 \quad n'_1 - n'_2 \]
Δ-encoding: first try

\[ n = 3 \]

\[ \Delta\text{-encode} \]

\[ n \times A_1 = 27 \]
\[ n \times A_2 = 24 \]

\[
\begin{array}{cccccccccccccc}
\end{array}
\]

encode:
\[ n \times A_1 \]
\[ n \times A_2 \]

decode:
\[ n'_1 - n'_2 \]
\[ n'_1 - n'_2 \]

check:
\[ (n'_1 \% A_1) \&\& (n'_2 \% A_2) \&\& (n'_1 / A_1 = n'_2 / A_2) \]
Δ-encoding: first try

\[ n = 3 \]

\[ \Delta - \text{encode} \]

\[ n \times A_1 = 27 \]
\[ n \times A_2 = 24 \]

encode:
\[ n \times A_1 \quad n \times A_2 \]

decode:
\[ n'_1 - n'_2 \quad n'_1 - n'_2 \]

check:
\[ (n'_1 \% A_1) \&\& (n'_2 \% A_2) \&\& (n'_1 / A_1 = n'_2 / A_2) \]
\[ \Delta \text{-encoding: second try} \]

- carefully choose \( A_1 \) and \( A_2 \)

- choose \( A_1 \) and \( A_2 \) such as

<table>
<thead>
<tr>
<th>( A_1 = 2^k + 2^i )</th>
<th>( A_2 = 2^k - 2^i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( k = 3, i = 0: )</td>
<td>( A_1 = 8 + 1 = 9 )</td>
</tr>
</tbody>
</table>
Δ-encoding: second try

- carefully choose $A_1$ and $A_2$

- choose $A_1$ and $A_2$ such as

\[
A_1 = 2^k + 2^i \quad \text{and} \quad A_2 = 2^k - 2^i
\]

$k = 3, i = 0$:

\[
A_1 = 8 + 1 = 9 \quad \text{and} \quad A_2 = 8 - 1 = 7
\]

- we notice that

\[
A_1 - A_2 = 2^k + 2^i - 2^k + 2^i = 2^{i+1}
\]

\[
A_1 + A_2 = 2^k + 2^i + 2^k - 2^i = 2^{k+1}
\]
\( \Delta \)-encoding: second try

\( \uparrow \) carefully choose \( A_1 \) and \( A_2 \)

– choose \( A_1 \) and \( A_2 \) such as

\[
\begin{align*}
A_1 &= 2^k + 2^i \\
A_2 &= 2^k - 2^i
\end{align*}
\]

\( k = 3, i = 0: \quad A_1 = 8 + 1 = 9 \quad A_2 = 8 - 1 = 7 \)

– we notice that

\[
\begin{align*}
A_1 - A_2 &= 2^k + 2^i - 2^k + 2^i = 2^{i+1} \\
A_1 + A_2 &= 2^k + 2^i + 2^k - 2^i = 2^{k+1}
\end{align*}
\]

– two ways to decode

\[
\begin{align*}
n &= (n'_1 - n'_2)/2^{i+1} = n \times (A_1 - A_2)/2^{i+1} \\
n &= (n'_1 + n'_2)/2^{k+1} = n \times (A_1 + A_2)/2^{k+1}
\end{align*}
\]
\(\Delta\)-encoding: second try

→ carefully choose \(A_1\) and \(A_2\)

– choose \(A_1\) and \(A_2\) such as

\[
\begin{align*}
A_1 &= 2^k + 2^i \\
A_2 &= 2^k - 2^i
\end{align*}
\]

\(k = 3, \ i = 0:\)

\[
\begin{align*}
A_1 &= 8 + 1 = 9 \\
A_2 &= 8 - 1 = 7
\end{align*}
\]

– we notice that

\[
\begin{align*}
A_1 - A_2 &= 2^k + 2^i - 2^k + 2^i = 2^{i+1} \\
A_1 + A_2 &= 2^k + 2^i + 2^k - 2^i = 2^{k+1}
\end{align*}
\]

– two ways to decode

\[
\begin{align*}
n &= (n'_1 - n'_2) \gg i+1 = n \times (A_1 - A_2) \gg i+1 \\
n &= (n'_1 + n'_2) \gg k+1 = n \times (A_1 + A_2) \gg k+1
\end{align*}
\]
\( \Delta \)-encoding: second try

\[
\begin{align*}
n = 3 & \quad \Delta \text{-encode} \\
n \times A_1 &= 27 \\
n \times A_2 &= 21
\end{align*}
\]
Δ-encoding: second try

\[ n = 3 \]

\[ \Delta \text{-encode} \]

\[ n \times A_1 = 27 \]
\[ n \times A_2 = 21 \]
Δ-encoding: second try

\[ n = 3 \]

\[ \Delta\text{-encode} \quad n \times A_1 = 27 \quad n \times A_2 = 21 \]

encode: \hspace{1cm} n \times A_1 \hspace{1cm} n \times A_2

decode: \hspace{1cm} (n'_1 - n'_2) \gg i+1 \hspace{1cm} (n'_1 + n'_2) \gg k+1

check: \hspace{1cm} (n'_1 \% A_1) \&\& (n'_2 \% A_2) \&\& (n'_1 / A_1 = n'_2 / A_2)
Probabilistic Guarantees

– given faults happening in both copies of data: \( P(\text{fault undetected}) \)
– assume 8-bit \( n_1 \) and \( n_2 \)
– choose \( A_1 = 9 \) and \( A_2 = 7 \)

**SWIFT:**

– single bit-flip \( P = 1/8 \)
– multiple bit-flips \( P = 1/256 \)

**Δ-encoding:**

– single bit-flip \( P = 0 \)
– multiple bit-flips \( P = 1/(9 \times 256) \)
Probabilistic Guarantees

- given faults happening in both copies of data: \( P(\text{fault undetected}) \)
- assume 8-bit \( n_1 \) and \( n_2 \)
- choose \( A_1 = 9 \) and \( A_2 = 7 \)

**SWIFT:**
- single bit-flip \( P = 1/8 \)
- multiple bit-flips \( P = 1/256 \)

**\( \Delta \)-encoding:**
- single bit-flip \( P = 0 \)
- multiple bit-flips \( P = 1/(9\times256) \)

- one bit flipped in \( n_1 \)
- the same one bit (out of 8 possible) must flip in \( n_2 \)
## Probabilistic Guarantees

- given faults happening in both copies of data: $P(\text{fault undetected})$
- assume 8-bit $n_1$ and $n_2$
- choose $A_1 = 9$ and $A_2 = 7$

### SWIFT:

<table>
<thead>
<tr>
<th>Event</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>single bit-flip</td>
<td>$P = 1/8$</td>
</tr>
<tr>
<td>multiple bit-flips</td>
<td>$P = 1/256$</td>
</tr>
</tbody>
</table>

### Δ-encoding:

<table>
<thead>
<tr>
<th>Event</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>single bit-flip</td>
<td>$P = 0$</td>
</tr>
<tr>
<td>multiple bit-flips</td>
<td>$P = 1/(9 \times 256)$</td>
</tr>
</tbody>
</table>

- $n_1$ is some fixed number 0..255
- $n_2$ must be set to this number
Probabilistic Guarantees

– given faults happening in both copies of data: $P(\text{fault undetected})$
– assume 8-bit $n_1$ and $n_2$
– choose $A_1 = 9$ and $A_2 = 7$

**SWIFT:**

– single bit-flip $P = \frac{1}{8}$
– multiple bit-flips $P = \frac{1}{256}$

**Δ-encoding:**

– single bit-flip $P = 0$
– multiple bit-flips $P = \frac{1}{(9 \times 256)}$

– single bit-flip results in invalid word
Probabilistic Guarantees

– given faults happening in both copies of data: \( P(\text{fault undetected}) \)
– assume 8-bit \( n_1 \) and \( n_2 \)
– choose \( A_1 = 9 \) and \( A_2 = 7 \)

**SWIFT:**

– single bit-flip \( P = \frac{1}{8} \)
– multiple bit-flips \( P = \frac{1}{256} \)

**\( \Delta \)-encoding:**

– single bit-flip \( P = 0 \)
– multiple bit-flips \( P = \frac{1}{(9 \times 256)} \)

\( \text{lower probability with greater } A \)

– \( n_1 \) must change to some code word
– \( n_2 \) must be set to the corresponding code word
– independent events
Implementation

Hardening applications with Ω-encoding
Encoded Operations

A1
add1
accu1
xor1
accu1
sub1
accu1
CPU

A2
add2
accu2
xor2
accu2
sub2
accu2
check
Encoded Operations

– fully encoded

add, sub, modulo, comparisons
Encoded Operations

- **fully encoded**
  - add, sub, modulo, comparisons

- **partially encoded**
  - mul, shifts
Encoded Operations

- **fully encoded**
  - add, sub, modulo, comparisons

- **partially encoded**
  - mul, shifts

- **fully decoded**
  - div, and, or, xor

optimizations for speed & fault tolerance *(refer to paper)*
Accumulations and Checks

\[ \text{CPU A1} \quad \text{CPU A2} \]

- add
- accu1
- xor1
- accu1
- sub1
- accu1
- check

- add
- accu2
- xor2
- accu2
- sub2
- accu2

DSN'2015
Accumulations and Checks

\[ a_1 \% A_1 \land a_2 \% A_2 \land a_1 / A_1 = a_2 / A_2 \]

- very slow
- who watches the watchmen?
Accumulations and Checks

\[
a_1 \% A_1 \land a_2 \% A_2 \land \frac{a_1}{A_1} = \frac{a_2}{A_2}
\]
Accumulations and Checks

\[ a_1 \% A_1 \land a_2 \% A_2 \land \frac{a_1}{A_1} = \frac{a_2}{A_2} \]
Accumulations and Checks

\[ a_1 = 0 \quad \text{and} \quad a_2 = 0 \]

\[ a_1 += add_1 \quad \text{and} \quad a_2 += add_2 \]

\[ a_1 \% A_1 \quad \&\& \quad a_2 \% A_2 \quad \&\& \quad a_1/A_1=a_2/A_2 \]
Accumulations and Checks

\[a_1 = 0\]
\[a_2 = 0\]

\[a_1 += \text{add}_1\]
\[a_2 += \text{add}_2\]

\[a_1 += \text{xor}_1\]
\[a_2 += \text{xor}_2\]

\[a_1 \% A_1 \&\& a_2 \% A_2 \&\& a_1 / A_1 = a_2 / A_2\]
Accumulations and Checks

\[
\begin{align*}
    a_1 &= 0 \\
    a_2 &= 0 \\
    a_1 &= a_1 + \text{add}_1 \\
    a_2 &= a_2 + \text{add}_2 \\
    a_1 &= a_1 + \text{xor}_1 \\
    a_2 &= a_2 + \text{xor}_2 \\
    a_1 &= a_1 + \text{sub}_1 \\
    a_2 &= a_2 + \text{sub}_2 \\
    a_1 \% A_1 &\land a_2 \% A_2 &\land a_1 / A_1 = a_2 / A_2
\end{align*}
\]
Evaluation
Performance & Fault Coverage
Evaluation: Performance

![Bar chart showing performance evaluation for different benchmarks and techniques.]

- **Bubblesort**: Performance comparison for native, full, parallel, and stripped versions.
- **Quicksort**: Similar performance trend as bubblesort.
- **Linkedlist**: Noticeable differences among the native, full, parallel, and stripped versions.
- **CRC32**: Performance metrics for native, full, parallel, and stripped versions.
- **Dijkstra**: Performance evaluation showing distinct impacts of native, full, parallel, and stripped techniques.
- **Patricia**: Performance analysis for native, full, parallel, and stripped variants.
- **Hardcore**: Performance comparison for native, full, parallel, and stripped versions.
- **Industrial**: Performance evaluation for native, full, parallel, and stripped versions.

Legend:
- Native
- Full
- Parallel
- Stripped
Evaluation: Performance

accumulation and checks
done in parallel

- native
- $\Delta$-full
- $\Delta$-parallel
- $\Delta$-stripped

slowdown

bubblesort
quicksort
linkedlist
crc32
dijkstra
patricia
hardcore
industrial
Evaluation: Performance

- Accumulation and checks done in parallel
- No accumulation; checks only on final results

![Bar chart showing performance slowdown for different benchmarks and configurations]

- **Native**
- **$\Delta$-full**
- **$\Delta$-parallel**
- **$\Delta$-stripped**
Evaluation: Fault Coverage

\number of Silent Data Corruptions (SDC)

- transient: 68527
- intermittent: 7282
- permanent: 6505

- bubblesort: our worst case
Evaluation: Fault Coverage

\( \downarrow \) number of Silent Data Corruptions (SDC)

- transient: 4
- intermittent: 3
- permanent: 1

- bubblesort: our worst case
- other benchmarks: 3 SDC in total
- 99.997% fault coverage
Conclusion

**Δ-encoding:** software-based fault detection

– detect transient, intermittent & permanent faults
  – by information redundancy (AN encoding)
  – and data diversity (choice of As)
– very high fault coverage
– moderate performance slowdown
– suited for small safety-critical parts of programs
Conclusion

**Δ-encoding**: software-based fault detection

- detect transient, intermittent & permanent faults
  - by information redundancy (AN encoding)
  - and data diversity (choice of As)
- very high fault coverage
- moderate performance slowdown
- suited for small safety-critical parts of programs

Thank you! Questions?
Backup Slides
## Evaluation: Performance Characteristics

### Instructions per Cycle (IPC)

<table>
<thead>
<tr>
<th></th>
<th>native</th>
<th>Δ-full</th>
<th>Δ-parallel</th>
<th>Δ-stripped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bubblesort</td>
<td>1.25</td>
<td>2.27</td>
<td>2.34</td>
<td>2.26</td>
</tr>
<tr>
<td>HardCore</td>
<td>1.78</td>
<td>2.73</td>
<td>2.61</td>
<td>2.70</td>
</tr>
<tr>
<td>Industrial</td>
<td>1.46</td>
<td>2.70</td>
<td>2.75</td>
<td>2.82</td>
</tr>
</tbody>
</table>
Dependable Solutions: hardware-based

- expensive
- energy-inefficient
- not suited for mixed-criticality systems
Evaluation: Fault Coverage (full picture)
Future Plans

- Δ-encoding specific **compiler optimizations**
- **Recovery** based on a „good“ copy
- Accumulations outsourced to **hardware watchdog**
- **Security-related** applications
AN encoding: overview

Increasing fault coverage of AN encoding:

- ANB
  
  **encode:** \( n \times A + B_n \)

- ANBD
  
  **encode:** \( n \times A + B_n + D_n \)

unique signature for each variable

unique timestamp for each variable