HARP: Practically and Effectively Identifying Uncorrectable Errors in Memory Chips That Use On-Die Error-Correcting Codes

Minesh Patel, Geraldo F. Oliveira, Onur Mutlu

*Presented at MICRO on 20 October 2021*

https://arxiv.org/abs/2109.12697

https://github.com/CMU-SAFARI/HARP
**HARP Summary**

**Motivation:** state-of-the-art memory error mitigations often require the processor to identify which bits are at risk of error (i.e., profiling)  

**Problem:** on-die ECC complicates error profiling by altering how errors appear outside of the memory chip

**Goal:** understand and address the challenges on-die ECC introduces

**Contributions:**

1. **Analytically study** on-die ECC’s effects and identify three key challenges
   - i. Exponentially increases the number of at-risk bits
   - ii. Makes individual at-risk bits harder to identify
   - iii. Interferes with commonly-used memory data patterns

2. **Hybrid Active-Reactive Profiling (HARP):**
   - i. Separately identifies (1) raw bit errors and (2) errors introduced by on-die ECC
   - ii. Effectively reduces profiling with on-die ECC into profiling without on-die ECC

**Evaluation:** demonstrate that HARP overcomes the three challenges

- HARP identifies all errors faster than two baselines, which sometimes fail to achieve full coverage of at-risk bits
- Case study showing that HARP identifies all errors faster than the best-performing baseline (e.g., by 3.7x for a raw per-bit error probability of 0.75)

[SAFARI](https://github.com/CMU-SAFAI/HARP)
HARP Outline

1. Memory Error Mitigation and Profiling
2. On-Die ECC’s Impact on Error Profiling
3. HARP: Practical and Effective Profiling
4. Evaluations
5. Conclusion and Takeaways
Scaling-Related Memory Errors

• Density scaling increases memory error rates

Uncorrelated single-bit errors are the primary challenge with continued DRAM process scaling.
Increasing Single-Bit Error Rates

• Higher error rates require more sophisticated solutions

Past
No Mitigation Necessary
Row/Column Sparing

Present
On-Die ECC
Single-Error Correction

Near Future
Advanced Solutions

Reproduced from prior work [Kline+, HPCA’20], [Nair+, ISCA’13]
Typical On-Die ECC Design Today

- 128-bit single-error correcting (SEC) Hamming code
Error Mitigation at High Error Rates

Large research space

Software

In-Processor

In-Memory

Page retirement
Task replication
Application fault tolerance

... Bit-Repair Mechanisms
state-of-the-art
for addressing
scaling-related errors

Row/Column Sparing
On-Die ECC

... Cost and efficiency depend on error characteristics
Memory Repair Mechanisms

- **Identify** and **repair** any bits that are **at-risk of error**

**Memory Controller**

- **Repair Mechanism**
  - **Error Profile**

**Unreliable Memory**

- **Error-Prone Data Store**

To/from CPU

- **write data**
- **read data**
- **repaired**

Must **know** which bits are at-risk

Determined through **“Error Profiling”**
Error Profiling Algorithms

1. **Profiler Type?**
   - **Active**
     - Induce **worst-case conditions**
   - **Reactive**
     - Passively **monitor the memory chip**

2. **Check for errors**
3. **Mark erroneous bits as at-risk**

**Repeat until all at-risk bits identified**
Error Profiling

1. **Profiler Type?**
   - **Active**: Induce **worst-case conditions**
   - **Reactive**: Passively **monitor the memory chip**

2. **Check for errors**

3. **Mark erroneous bits as at-risk**

4. **Repeat until all at-risk bits identified**

- **Requires observing at-risk bits fail**
Profiling a Memory Chip with On-Die ECC

- On-die ECC changes how errors appear to the profiler

**Q: How does on-die ECC affect error profiling?**

**Unreliable Memory**

```
Profiler
write data
read data
```

```
On-Die ECC
```

```
Error-Prone Data Store
```

**Goal:** understand and address the challenges that on-die ECC introduces for error profiling
HARP Outline

1. Memory Error Mitigation and Profiling
2. On-Die ECC’s Impact on Error Profiling
3. HARP: Practical and Effective Profiling
4. Evaluations
5. Conclusion and Takeaways
Profiling a Memory Chip

- Profiler’s goal: identify all bits that are **at risk of error**

Profiler **cannot see** into the memory

**Memory Chip (Without On-Die ECC)**

- Raw data

**Memory Chip (With On-Die ECC)**

- Post-correction data
- On-Die ECC

Data Store

Profiler marks the **bits** that are observed to fail.
Error Profiling Without On-die ECC

• Only one source of errors: when the physical bit fails

"Direct" error
Same errors inside and outside of the chip

Profiler identifies these bits

Error profile = union(direct errors)
Error Profiling With On-Die ECC

• Two different sources of errors

Direct error
Appears before and after error correction

Indirect error
Occurs when on-die ECC mistakenly corrects a bit

Profiler identifies these bits

error profile = union(direct errors, indirect errors)
A Closer Look at Indirect Errors

- Indirect errors depend on the raw bit error pattern

In the figure:
- **Pre-correction**
  - Data: `X---XX` (At Risk)
  - Metadata: `---XX` (Not at risk)

- **On-Die ECC**
  - Direct error: `E---EE`
  - Indirect error: `E---E`

- **Post-correction**
  - Direct error (same position) corrected to `E---EE`
  - Indirect error (different positions) corrected to `E---E`

**SAFARI**
A Closer Look at Indirect Errors

Key observation: Any bit can be at-risk of indirect errors with different combinations of raw bit errors.

On-die ECC causes statistical dependence between otherwise independent bits.
Challenges Introduced by On-Die ECC

1. Exponentially increases the at-risk bits
   A small set of raw bit errors creates a combinatorially larger set of at-risk post-correction bits

2. Harder to observe each at-risk bit
   At-risk post-correction bits can only be exposed by specific raw bit error patterns

3. Interferes with data patterns
   Data patterns must consider combinations of raw bits instead of just individual bits alone
Challenge 1: Combinatorial Explosion

• N at-risk bits can fail in $2^N$ ways

$2^3$ unique error patterns

Correctable

Uncorrectable
Challenge 1: Combinatorial Explosion

Every uncorrectable pattern can cause a unique indirect error

2^3 unique error patterns

Correctable

Uncorrectable
Challenge 1: Combinatorial Explosion

- **Exponential increase** in the number of at-risk bits that the profiler must identify

### Worst-Case Explosion of At-Risk Bits

<table>
<thead>
<tr>
<th>At-Risk Pre-Correction</th>
<th>At-Risk Post-Correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>$2^n - 1$</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>255</td>
</tr>
</tbody>
</table>
Challenge 2: Identifying At-Risk Bits

• Indirect errors only appear for specific ECC-dependent combinations of pre-correction errors
• This makes identifying indirect errors slow and difficult
  • The profiler can neither see nor control pre-correction errors
  • Instead, the profiler is forced to blindly explore different pre-correction error combinations to achieve high coverage

The slow exploration can only be overcome by transparency into on-die ECC
Challenge 3: Data Patterns

• Profilers employ carefully-designed data patterns

![Data Patterns Diagram]

• Data-patterns induce worst-case circuit behavior
  • Maximizes the chance of identifying errors
  • Exercises different failure modes
Challenge 3: Data Patterns

• On-die ECC breaks these data patterns in **two ways**

1. **Carefully-Designed**

   0101010101
   1010101010

   → **On-Die ECC**

2. **Obfuscated**


   Multi-bit data patterns are **difficult to design and use**
   (discussed in our paper)

2. **Conventional data patterns induce single-bit worst case conditions**

   On-die ECC requires **multiple bits** to fail concurrently to expose errors
HARP Outline

1. Memory Error Mitigation and Profiling
2. On-Die ECC’s Impact on Error Profiling
3. HARP: Practical and Effective Profiling
4. Evaluations
5. Conclusion and Takeaways
Key Observation

• Indirect errors are an artifact of on-die ECC

- Direct error
- Indirect error

Upper-bounded by the ECC algorithm

• An N-error correcting ECC can only cause at most N indirect errors at a time
Key Observation

• Indirect errors are an artifact of on-die ECC

Key idea:
Identify **direct** and **indirect** errors separately
Hybrid Active-Reactive Profiling (HARP)

1. **Active Profiling**
   - Quickly identifies **direct errors**

2. **Reactive Profiling**
   - Safely identifies the **indirect errors**

---

**Memory Controller**
- Active Profiler on error detected
- ECC bypass
- Read data
- Write data

**Memory Chip**
- On-Die ECC
- Data Store

**Primary Components**
- Secondary ECC
- Error Profile
- Repair Mechanism

---

**SAFARI**
Active Profiling Design

Memory Controller
- Repair Mechanism
- Error Profile
- Active Profiler

Memory Chip
- On-Die ECC
- Data Store

Bypass only for reads (returns data bits, ignores ECC metadata)

- ECC bypass is a simple, low-overhead change
  - No change to data transfer granularity or ECC algorithm
  - Enables using existing profiling algorithms to identify bits at risk of direct errors as if there is no on-die ECC
Active Profiling Design

Memory Controller

- Repair Mechanism
- Error Profile
- Active Profiler
- on error detected

Memory Chip

- On-Die ECC
- Data Store

ECC bypass

Reduces the task of profiling with on-die ECC into a task of profiling without on-die ECC with minimal modifications to on-die ECC
Reactive Profiling Design

- System designer must choose a suitable secondary ECC
- Large ECC design space – only one requirement:
  - Secondary ECC must correct $N$ errors per on-die ECC word given an $N$-error-correcting on-die ECC
  - Requires **aligning** the two ECC words (details in our paper)

SAFARI
Improving Reactive Profiling

- Reactive profiling slowly identifies indirect errors one-at-a-time as they occur during runtime.
- We can shorten this process by anticipating indirect error locations from the already-observed direct errors.

Can predict a subset of indirect errors by knowing the on-die ECC implementation (i.e., its parity-check matrix).

- We introduce two HARP variants:
  - HARP-A(ware) – knows the parity-check matrix
  - HARP-U(naware) – does not know the parity-check matrix
HARP Outline

1. Memory Error Mitigation and Profiling
2. On-Die ECC’s Impact on Error Profiling
3. HARP: Practical and Effective Profiling
4. Evaluations
5. Conclusion and Takeaways
Evaluation Methodology

• We evaluate HARP using **Monte-Carlo simulation**
  • Enables **accurately measuring** coverage (using a SAT solver)
  • 1,036,980 total ECC words
    • Across 2769 randomly-generated (71, 64) and (136, 128) ECC codes
    • ≈14 CPU-years (20 days on 256 cores) of simulation time

• Artifacts are **open-sourced**

DOI: 10.5281/zenodo.5148592

https://github.com/CMU-SAFARI/HARP
Baseline Profiling Algorithms

- We compare HARP with two baseline algorithms:

1. **Naive**: round-based profiling that ignores on-die ECC
   - Each round uses different data patterns (e.g., random data)
   - Profiler marks observed errors as at-risk bits

2. **BEEP** [Patel+, MICRO'20]: knows the exact on-die ECC implementation (i.e., its parity-check matrix)
   - Same overall round-based strategy as Naive
   - Data patterns designed using the known parity-check matrix
1. HARP achieves **full coverage** in all cases, **outperforming** both baseline algorithms
   - BEEP **fails** to achieve full coverage because it **does not explore** different pre-correction error patterns
2. HARP is **independent** of the number of pre-correction errors because it directly reads raw data bit values
HARP **overcomes** all three profiling challenges by **separating** direct and indirect errors.

Outperforming both baseline algorithms:
- **BEEP fails** to achieve full coverage because it **does not explore** different pre-correction error patterns.
- **HARP is independent** of the number of pre-correction errors because it directly reads raw data bit values.
**Profiling Speed Evaluation**

- **Goal:** determine how many profiling rounds are necessary to prevent* N-bit error patterns
- We inject 2 raw bit errors per ECC word
  - Per-bit error probability = 1.0 (fails in every profiling round)

*for 99th percentile coverage

![Diagram showing profiling speed evaluation]
Profiling Speed Evaluation

- HARP achieves high coverage of at-risk bits much faster than the baseline algorithms
Profiling Speed Evaluation

HARP achieves high coverage of at-risk bits much faster than the baseline algorithms. HARP performs 20.6- to 62.1% faster than the best-performing baseline.
Case Study: DRAM Data-Retention

- We consider a system that uses an ideal repair mechanism to safely reduce the DRAM refresh rate.
- We study how the end-to-end bit error rate (BER) changes when using different profilers.

**BEEP** fails to reach zero BER.

**HARP** always reaches zero BER.

**SAFARI**
Case Study: DRAM Data-Retention

HARP reaches zero BER 3.7x faster than the best-performing baseline.
HARP Outline

1. Memory Error Mitigation and Profiling
2. On-Die ECC’s Impact on Error Profiling
3. HARP: Practical and Effective Profiling
4. Evaluations
5. Conclusion and Takeaways
Other Information in the Paper

• Detailed analysis of on-die ECC
  • How on-die ECC introduces statistical dependence between post-correction errors
  • Differences between direct and indirect errors

• Discussion about HARP’s design decisions

• More evaluation results
  • Coverage of direct and indirect errors
  • Analysis of profiler bootstrapping
  • Case study on the end-to-end memory bit error rate (BER)

• Detailed artifact description
HARP: Practically and Effectively Identifying Uncorrectable Errors in Memory Chips That Use On-Die Error-Correcting Codes

Minesh Patel
ETH Zürich

Geraldo F. Oliveira
ETH Zürich

Onur Mutlu
ETH Zürich

ABSTRACT
Aggressive storage density scaling in modern main memories causes increasing error rates that are addressed using error-mitigation techniques. State-of-the-art techniques for addressing high error rates identify and repair bits that are at risk of error from within the memory controller. Unfortunately, modern main memory chips internally use on-die error correcting codes (on-die ECC) that obfuscate the memory controller’s view of errors, complicating the process of identifying at-risk bits (i.e., error profiling).

To understand the problems that on-die ECC causes for error profiling, we analytically study how on-die ECC changes the way that memory errors appear outside of the memory chip (e.g., to the memory controller). We show that on-die ECC introduces statistical dependence between errors in different bit positions, raising three key challenges for practical and effective error profiling: on-die ECC (1) exponentially increases the number of at-risk bits the profiler must identify; (2) makes individual at-risk bits more difficult to identify; and (3) interferes with commonly-used memory data patterns that are designed to make at-risk bits easier to identify.

profiler impacts the system’s overall bit error rate (BER) when using a repair mechanism to tolerate DRAM data-retention errors. We show that HARP identifies all errors faster than the best-performing baseline algorithm (e.g., by 3.7× for a raw per-bit error probability of 0.75). We conclude that HARP effectively overcomes the three error profiling challenges introduced by on-die ECC.

CCS CONCEPTS
• Computer systems organization → Dependable and fault-tolerant systems and networks; • Hardware → Memory test and repair.

KEYWORDS
On-Die ECC, DRAM, Memory Test, Repair, Error Profiling, Error Modeling, Memory Scaling, Reliability, Fault Tolerance

ACM Reference Format:

https://arxiv.org/abs/2109.12697
Artifacts are Open-Sourced

DOI: 10.5281/zenodo.5148592

https://github.com/CMU-SAFARI/HARP
HARP Summary

Motivation: state-of-the-art memory error mitigations often require the processor to identify which bits are at risk of error (i.e., profiling)

Problem: on-die ECC complicates error profiling by altering how errors appear outside of the memory chip

Goal: understand and address the challenges on-die ECC introduces

Contributions:
1. Analytically study on-die ECC’s effects and identify three key challenges
   i. Exponentially increases the number of at-risk bits
   ii. Makes individual at-risk bits harder to identify
   iii. Interferes with commonly-used memory data patterns
2. Hybrid Active-Reactive Profiling (HARP):
   i. Separately identifies (1) raw bit errors and (2) errors introduced by on-die ECC
   ii. Effectively reduces profiling with on-die ECC into profiling without on-die ECC

Evaluation: demonstrate that HARP overcomes the three challenges
• HARP identifies all errors faster than two baselines, which sometimes fail to achieve full coverage of at-risk bits
• Case study showing that HARP identifies all errors faster than the best-performing baseline (e.g., by 3.7x for a raw per-bit error probability of 0.75)

SAFARI https://github.com/CMU-SAFARI/HARP
HARP: Practically and Effectively Identifying Uncorrectable Errors in Memory Chips That Use On-Die Error-Correcting Codes

Minesh Patel, Geraldo F. Oliveira, Onur Mutlu

Presented at MICRO on 20 October 2021

https://arxiv.org/abs/2109.12697
https://github.com/CMU-SAFARI/HARP
Backup Slides
Addressing High Error Rates

• Unfortunately, coarse-grained mitigation is typically **impractical** at high error rates (e.g., $>10^{-4}$)

**High BERs** demand **fine-grained** mitigation
Challenge 1: Combinatorial Explosion

Pre-Correction

At-Risk Bit

Normal Bit

Direct and indirect error coincide

Direct error

Indirect error

Post-Correction
Challenge 2: Bootstrapping (2/2)

• The profiler **cannot draw conclusions** from having observed a bit **not fail**

**Memory Chip**

- Physical bit did not fail (i.e., no direct error)

**Memory Chip with on-die ECC**

- **Unknown!**

---

- **REAPER** (ISCA’17)
- **EIN** (DSN’19)
- **BEER** (MICRO’20)
- **HARP** (MICRO’21)
# Combinatorial Explosion of Errors

<table>
<thead>
<tr>
<th>Bits at risk of pre-correction errors</th>
<th>$n$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique pre-correction error patterns</td>
<td>$2^n - 1$</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>255</td>
</tr>
<tr>
<td>Uncorrectable pre-correction error patterns</td>
<td>$2^n - n - 1$</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>11</td>
<td>247</td>
</tr>
<tr>
<td>Bits at risk of post-correction errors</td>
<td>$2^n - 1$</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td>15</td>
<td>255</td>
</tr>
</tbody>
</table>
Example Granularity Matching

• **Goal:** The system designer wants to protect each on-die ECC word with at least as strong an ECC as on-die ECC.
Per-Bit Error Probability of Each At-Risk Bit

- 70K ECC words per 1600 (71, 64) SEC Hamming codes

Figure 4: Distribution of each at-risk bit’s error probability before and after application of on-die ECC.
Challenge 1: Combinatorial Explosion

Pre-Correction

\[
\begin{array}{cccccc}
\text{X} & - & - & - & - & \text{X} \text{ X} \\
0 & 1 & 2
\end{array}
\]

At-Risk Bit

Normal Bit

\[2^3 \text{ Possible pre-correction error combinations}\]

Correctable

\[
\begin{array}{c}
\{0\} \\
\{1\} \\
\{2\}
\end{array}
\]

Uncorrectable

\[
\begin{array}{c}
\{0, 1\} \\
\{0, 2\} \\
\{0, 1, 2\} \\
\{1, 2\}
\end{array}
\]
Challenge 1: Combinatorial Explosion

Every combination can potentially cause a unique indirect error

Possible pre-correction error combinations:

Correctable:
- {}  
- {0}  
- {1}  
- {2}

Uncorrectable:
- {0, 1}  
- {0, 2}  
- {0, 1, 2}  
- {1, 2}
Challenge 1: Combinatorial Explosion

Memory Chip

- Data
- Metadata

At-Risk Bit
Normal Bit

At Risk of Direct Errors
At Risk of Indirect Errors
Key Idea (1/2)

If we “somehow identify” **all direct errors**...

... we can **safely rely** on a secondary ECC to identify remaining **indirect errors**
Key Idea (2/2)

If we can **read the raw data** (but not metadata)...

... we can **use existing profiling techniques** to quickly identify all **direct errors**
Hybrid Active-Reactive Profiling (HARP)

**1. Active Profiling**
Quickly identifies **direct errors**

- **Memory Controller**
  - **Secondary ECC**
    - write data
    - repaired read data
  - **Repair Mechanism**
  - **Error Profile**
  - **Active Profiler**

- **Memory Chip**
  - **On-Die ECC**
  - **Data Store**

- **To/from CPU**

- **on error detected**

- **ECC bypass**

**2. Reactive Profiling**
Safely identifies the **indirect errors**

SAFARI
Active Profiling Design

- Able to use existing profiling algorithms as if there is no on-die ECC to identify bits at risk of direct errors.
- Does not identify bits at risk of indirect errors.

Low hardware cost: skipping ECC decoding

- Bypass read (returns raw data, ignores metadata)
- Normal read

On-Die ECC

SAFARI
Active Profiling Design

- Able to use existing profiling algorithms as if there is no on-die ECC to identify bits at risk of direct errors
- Does not identify bits at risk of indirect errors

Reduces the task of profiling with on-die ECC into a task of profiling without on-die ECC with minimal modifications to on-die ECC

Low hardware cost:

- Bypass read (returns raw data, ignores metadata)
- Normal read

Low hardware cost:

- Reduced task of profiling with on-die ECC into a task of profiling without on-die ECC
- Minimal modifications to on-die ECC

SAFARI
ECC Bypass Costs

- Simply skips on-die ECC decoding
- Details

Low hardware cost: skipping ECC decoding

Bypass read (returns raw data, ignores metadata)

Normal read

On-Die ECC
• System designer must choose a suitable secondary ECC
• Huge ECC design space – only one requirement:
  • For $N$-error-correcting on-die ECC, secondary ECC can correct $N$ errors per on-die ECC word
  • Requires aligning on-die ECC and secondary ECC words (details in our paper)
Hybrid Active-Reactive Profiling (HARP)

1. **Active Profiling**
   - Quickly identifies **all direct errors**
   - Bypasses on-die ECC on reads
   - Uses **existing** profiling techniques

2. **Reactive Profiling**
   - Safely identifies **indirect errors**
   - Uses a **secondary ECC** at least as strong as on-die ECC

- **Memory Controller**
  - Active Profiler
  - On-Die ECC
  - ECC bypass

- **Memory Chip**
  - Data Store

- **Secondary ECC**

**SAFARI**
Challenge 2: Bootstrapping

• Identifying a bit **at risk of indirect errors** is **hard**!
  • Requires a specific **pre-correction error pattern** to occur
  • However, the profiler **cannot see** pre-correction errors

• Leads to a vicious cycle:
  
  Profiler must test a bit

  Profiler does not know if the bit has been tested

  The profiler can only identify **indirect errors** one-at-a-time in a **guess-and-check** process