The Quest for Efficient and Trustworthy Systems

Baris Kasikci
Every 2 years, we create 2x more data than what we have created in all of human history\(^1\)

Efficiency of computer systems needs to catch up

Total cost of poor software quality > $2 Trillion in the US\(^2\)

Trustworthiness (Reliability + Security) needs to improve

---

\(^1\) Kirk Bresniker, World Economic Forum, 2018

\(^2\) Consortium for Information & Software Quality, 2021 Report
My Approach
Designing efficient and trustworthy systems based on a systematic understanding of program behavior
Efficiency

Datacenter Efficiency
- Whisper [MICRO’22]
- Thermometer [ISCA’22]
- I4 [SOSP’19]
- Snorlax [SOSP’17]

Heterogeneous Systems Support
- Persistent Memory Indexing [FAST’21]
- Optimus [ASPLOS’20]

My work appears in major venues in all these areas

Trustworthiness

Failure Reproduction and Analysis
- OmniTable [OSDI’22]
- Debugging in the Brave New World [ASPLOS’22]
- ER [PLDI’21]
- REPT [OSDI’18]
- Snorlax [SOSP’17]
- Hippocrates [ASPLOS’21]
- Agamotto [OSDI’20]

Verified Distributed Systems
- Sift [ATC’22]
- IGOR [RTAS’21]
- I4 [SOSP’19]

Hardware Security
- MOESI-prime [ISCA’22]
- Dolma [SEC’21]
- NDA [MICRO’19]
- Foreshadow [SEC’18]
- Morpheus [ASPLOS’19]
<table>
<thead>
<tr>
<th></th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datacenter Efficiency</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Failure Reproduction and Analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Security</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Navigating the efficiency-trustworthiness tension: A careful balance between offline and online techniques
## Outline

<table>
<thead>
<tr>
<th>Datacenter Efficiency</th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data-driven optimizations</td>
<td>Lightweight profiling</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Failure Reproduction and Analysis</th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Hardware Security</th>
<th></th>
</tr>
</thead>
</table>
Datacenters consume massive energy
  • 3% of the global energy, large carbon footprint\(^1\)
  • $35 million/year savings from 1% less work\(^2\)

Responsiveness impacts revenue
  • 400ms delay decreases Google Search users by 0.4%\(^2\)
  • Two second delay on search responses reduces Microsoft Bing’s revenue by 4.3%\(^2\)

Profile-Guided Optimizations, PGO
e.g., use a profile of branch traces for reordering code to make it cache-friendly
Limitations:

Significant hardware modifications
Impractical on-chip space overhead
Limited gains due to on-chip space limits
Profile-Guided Software Optimizations

On-Chip Analysis and Optimizations
Profile-Guided Software and Hardware Optimizations

Online Lightweight Profiling
Profile-Guided Software and Hardware Optimizations

Offline Analysis of Profiling Data

Online Lightweight Profiling

Performance improvement of up to 90% of the theoretical limit
Little-to-No hardware modifications (Intel & ARM technology transfer)
I-SPY: Context-Driven Conditional Instruction Prefetching with Coalescing

Tanvir Ahmed Khan*  Akshitha Sriraman*  Joseph Devietti†  Gilles Pokam‡  Heiner Litz§  Baris Kasikci*
*University of Michigan  †University of Pennsylvania  ‡Intel Corporation  §University of California, Santa Cruz
*{takh, akshitha, barisk}@umich.edu  †devietti@cis.upenn.edu  ‡gilles.a.pokam@intel.com  §hlitz@ucsc.edu

MICRO’20
Performance Impact of Instruction Cache Misses

20-70% performance lost due to large instruction footprint (14-45x I-cache)
Why Does Prior Work Fall Short?

[1] Grant Ayers et al., AsmDB: understanding and mitigating front-end stalls in warehouse-scale computers, ISCA 2019
Why Does Prior Work Fall Short?

Overfitting prefetches based on limited execution information hurts speedup
Context-Driven* Conditional Prefetching

If (cond1 && cond2)
call B

... prefetch G[B]

If (!cond1)
call D

*Context = Control flow tracked using efficient online hardware tracing

Prefetch only if B was executed recently

4 branches-long context information allows 90% of the speed of an ideal cache
I-SPY

Context-Driven Conditional Prefetching

• A data-driven optimization powered by offline analysis of profiling information
• Avoids unnecessary prefetches
• Can be implemented with minor hardware support

Achieves 90% of the ideal cache performance

• Outperforms prior work by Google by 22.5%

$700$ million in savings\(^1\)

**Thermometer: Profile-Guided BTB Replacement for Data Center Applications.**

Shixin Song
shixins@umich.edu
University of Michigan, USA

Tanvir Ahmed Khan
tak@umich.edu
University of Michigan, USA

Niranjan Soundararajan
niranjan.k.soundararajan@intel.com
Intel Labs, India

Sara Mahdizadeh Shahri
smahdiz@umich.edu
University of Michigan, USA

Sreenivas Subramoney
sreenivas.subramoney@intel.com
Intel Labs, India

Heiner Litz
hlitz@uucsc.edu
University of California, Santa Cruz, USA

Baris Kasikci
barisk@umich.edu
University of Michigan, USA

**ISCA’22**

**Whisper: Profile-Guided Branch Misprediction Elimination for Data Center Applications**

Tanvir Ahmed Khan∗
Mohammed Ugur†
Krishnendra Nathella‡
Dam Sunwoo§
Heiner Litz∥
Daniel A. Jimenez¶
Baris Kasikci∥

∗University of Michigan †ARM ‡University of California, Santa Cruz §Texas A&M University ¶[takah, mesur, barisk]@umich.edu †[(Krishnendra.Nathella, Dam.Sunwoo)]@arm.com ∥hlitz@uucsc.edu ¶djmenez@acm.org

**MICRO’22**

**OCOLOS: Online COde Layout OptimizationS**

Yuxuan Zhang∗
Tanvir Ahmed Khan†
Gilles Pokam‡
Baris Kasikci‡
Heiner Litz∥
Joseph Devietti∗

∗University of Pennsylvania †University of Michigan ‡Intel Corporation ¶University of California, Santa Cruz §[yuxuan, devietti]@seas.upenn.edu ¶{takah, barisk}@umich.edu ¶gilles.a.pokam@intel.com ¶hlitz@uucsc.edu

**EuroSys’22**

**APT-GET: Profile-Guided Timely Software Prefetching**

Saba Jamilian∗
Tanvir Ahmed Khan†
Grant Ayers‡
Baris Kasikci‡
Heiner Litz∗

∗University of California, Santa Cruz †Google ¶University of Michigan

**MICRO’21**

**DMon: Efficient Detection and Correction of Data Locality Problems Using Selective Profiling**

Tanvir Ahmed Khan
thak@umich.edu
University of Michigan

Ian Neal
ian@umich.edu
University of Michigan

Gilles Pokam
gil@umich.edu
Intel Corporation

Barzam Mozafari
brazm@umich.edu
University of Michigan

**ISCA’21**

**Ripple: Profile-Guided Instruction Cache Replacement for Data Center Applications**

Tanvir Ahmed Khan∗
Dexin Zhang†
Akshitha Siriramana¶
Joseph Devietti∥
Gilles Pokam∥
Heiner Litz‡
Baris Kasikci∥

∗University of Michigan †University of Science and Technology of China ¶University of Pennsylvania §Intel Corporation ¶University of California, Santa Cruz ¶[takah, akshitha, barisk]@umich.edu †devietti@cis.upenn.edu ‡gil@umich.edu ¶gilles.a.pokam@intel.com ∥hlitz@uucsc.edu

**MICRO’22**

**Tig: Profile-Guided BTB Prefetching for Data Center Applications**

Tanvir Ahmed Khan
thak@umich.edu
University of Michigan, USA

Niranjan Soundararajan
niranjan.k.soundararajan@intel.com
Intel Labs, India

Nathan Brown
nbro@umich.edu
University of Michigan, USA

Rakesh Kumar
rakesh.kumar@ntnu.no
Norwegian University of Science and Technology, Norway

Sreenivas Subramoney
sreenivas.subramoney@intel.com
Intel Labs, India

Gilles Pokam
gilles.a.pokam@intel.com
Intel Labs, USA

Baris Kasikci
barisk@umich.edu
University of Michigan, USA

Heiner Litz
hlitz@uucsc.edu
University of California, Santa Cruz, USA

**MICRO’21**

**PDede: Partitioned, Deduplicated, Delta Branch Target Buffer**

Niranjan Soundararajan
niranjan.k.soundararajan@intel.com
Processor Architecture Research Lab, Intel Labs, India

Peter Braun
pbraun@umich.edu
University of California, Santa Cruz, USA

Baris Kasikci
barisk@umich.edu
University of California, Santa Cruz, USA

Heiner Litz
hlitz@uucsc.edu
University of California, Santa Cruz, USA

Sreenivas Subramoney
sreenivas.subramoney@intel.com
Processor Architecture Research Lab, Intel Labs, India
**Awards**
- VMware Early Career Grant
- Intel Rising Star Award
- Intel Faculty Awards
  - 2017, 2018
- Rackham Ph.D. Fellowship
  - Tanvir Ahmed Khan
- MICRO’22 Best Paper Award

**Grants**
- NSF, Intel, SRC

**Intel¹ and ARM² technology transfer Collaborations**
- ARM
- University of Pennsylvania
- UC Santa Cruz

---

## Outline

<table>
<thead>
<tr>
<th>Datacenter Efficiency</th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data-driven optimizations</td>
<td>Lightweight profiling</td>
</tr>
</tbody>
</table>

| Failure Reproduction and Analysis | | |

| Hardware Security | | |
## Outline

<table>
<thead>
<tr>
<th></th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Datacenter Efficiency</strong></td>
<td>Data-driven optimizations</td>
<td>Lightweight profiling</td>
</tr>
<tr>
<td><strong>Failure Reproduction and Analysis</strong></td>
<td>Static &amp; symbolic program analysis</td>
<td>Selective information monitoring</td>
</tr>
<tr>
<td><strong>Hardware Security</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows

Philip J. Guo* Thomas Zimmermann† Nachiappan Nagappan† Brendan Murphy†

* Stanford University
† Microsoft Research

“Developers can fix a bug if they can reproduce the associated failure”

Reproducing failures is difficult, especially for production use cases
Memory dump

Last state of data in memory

Branches monitored for profiling (profile-guided optimizations)

Control flow trace

Manual Failure Reproduction
Navigating the efficiency/trustworthiness tension: Use branch traces for both optimizations and reproducing failures.
REPT and Execution Reconstruction

REPT: Reverse Debugging of Failures in Deployed Software
  • Most-widely deployed failure reproduction and analysis system in the world
  • Used in ~ 1 billion Microsoft Windows systems

Execution Reconstruction (ER)
  • Offline symbolic program analysis
  • Online selective hardware monitoring (control and data)
  • Reproduces arbitrarily longer executions than what REPT can
Execution Reconstruction: Harnessing Failure Reoccurrences for Failure Reproduction

Gefei Zuo  
gefeizuo@umich.edu  
University of Michigan, USA

Jiacheng Ma  
jcma@umich.edu  
University of Michigan, USA

Andrew Quinn  
arquinn@umich.edu  
University of Michigan, USA

Pramod Bhatotia  
pramod.bhatotia@in.tum.de  
TU Munich, Germany

Pedro Fonseca  
pfonseca@purdue.edu  
Purdue University, USA

Baris Kasikci  
barisk@umich.edu  
University of Michigan, USA
Prior Work vs. Execution Reconstruction (ER)

**ER (our work)**

- **Record nothing**
  - (e.g., symbolic execution)
  - no runtime overhead ✔
  - poor reproducibility ✗

- **Record everything**
  - high runtime overhead ✗
  - great reproducibility ✔

---

Existing trade-offs are insufficient to reproduce complex production failures
Challenges

Record nothing (e.g., symbolic execution) → Partial trace → Full failure trace

Record everything

ER

Offline

Online
void foo(int x) {
    int v[16] = {0};
    if (v[x] > 0) {
        // Path constraint - 1
    }
    else {
        // Path constraint - 2
    }
}
Background: Symbolic Execution

```c
void foo(int x) {
    int v[16] = {0};
    if (v[x] > 0) {
        ...
    }
    ...
}
```

- Symbolic (unknown) input
- Program state
- Control flow (branches)
Background: Symbolic Execution

```c
void foo(int x) {
    int v[16] = {0};
    if (v[x] > 0) {
        ...
        if{
            ...
            ...
            ...
            // FAILURE
    }
    {v[x] > 0} ∧ ... }
Background: Symbolic Execution

Challenge #1: Path explosion
Shepherded Symbolic Execution

- Avoids path-explosion by following a control flow trace recorded in production

Online ➔ Offline

Record nothing ➔ Shepherded Symbolic Execution ➔ Record everything

Partial trace (control-flow)
Shepherded Symbolic Execution

Challenge #1: Path explosion

- Symbolic (unknown) input
- Program state
- Control flow (branches)

FAILURE
Shepherded Symbolic Execution

- Symbolic (unknown) input
- Control flow (branches)
- Program state

 FAILURE
SMT Solving is Difficult

Challenge #2: Constraint solving is difficult (NP complete)

Symbolic (unknown) input

Program state

Control flow (branches)

\{{v[x] > 0} \land \ldots\}
Shepherded Symbolic Execution

- Avoids path-explosion by following a control flow trace recorded in production
- Reduces (simplifies) constraints using key data values recorded in production

Key question: What data values best simplify constraint solving?
Constraint Simplification: Intuition

• SMT solving is a hard problem (NP-Complete)
• **Observation**: reasoning about memory aliasing takes the most time

Memory Addresses: \( x \) may-alias \( y \)

Symbolic Memory
Constraint Simplification: Intuition

• SMT solving is a hard problem (NP-Complete)
• Observation: reasoning about memory aliasing takes the most time

Hypothesis: Recording addresses can simplify constraint solving
Constraint Simplification: Example

Symbolic inputs

```c
1 void foo(int x, int y) {
2    int V[256] = {0};
3    V[x] = 1;
4    ...V[y]...
5    ...
6 }
```
Constraint Simplification: Example

1 void foo(int x, int y) {
2  int V[256] = {0};
3  V[x] = 1;
4  ...V[y]...
5  ...
6 }

Value Dependency

Address Dependency

Expr depends on Operand

Constant
Symbolic Input

<table>
<thead>
<tr>
<th>Expr</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>V[256]</td>
</tr>
</tbody>
</table>

Symbolic inputs
Constraint Simplification: Example

int V[256];

Write

Read

Write

Read

may-alias
Record the runtime value of $x$ as $x'$.
Constraint Simplification: Example

Record the runtime value of $\mathbf{x}$ as $\mathbf{x}'$
Constraint Simplification: Example

Record the runtime value of $x$ as $x'$
Constraint Simplification: Heuristics

Record “key” symbolic memory addresses, which are used in:

- The longest symbolic write chain
- The write chain that accesses the largest symbolic memory object
Constraint Simplification: Heuristics

Record “key” symbolic memory addresses, which are used in:

- The longest symbolic write chain
- The write chain that accesses the largest symbolic memory object

V1[4] ⊸ Write ⊸ Write ⊸ Write ⊸ Write ⊸ Read

V2[4] ⊸ Write ⊸ Read

V3[512] ⊸ Write ⊸ Read

Leave to solver

Record!

Record!
Execution Reconstruction Summary

In-production Tracing Engine (Online)  Analysis Engine (Offline)

Application + Hardware Tracing  Shepherded Symbolic Execution

Send control-flow trace

SUCCESSFULLY reproduces the failure?

Yes

Done!
Execution Reconstruction Summary

In-production Tracing Engine (Online)

- Application + Hardware Tracing
- Send control-flow trace (+“key” addresses)
- FAILURE

Analysis Engine (Offline)

- Shepherded Symbolic Execution
- Successfully reproduces the failure?
  - Yes
  - No
  - Done!

Constraint Simplification

Ask for more key data (addresses) to be recorded
REPT

vs

ER

Records

Control Flow

+ 

Key Data Values (Addresses)

Reproduces $O(10^4)$ instructions

Reproduces $O(10^7)$ instructions
Execution Reconstruction – Results

Eliminates path explosion & simplifies constraint solving
  • By recording control flow and key data values

Can reproduce failures in complex, long-running executions
  • 1000x longer than REPT (deployed in Windows), without recording a longer trace
  • Requires only 3.5 reoccurrences on average per failure

0.3% runtime performance overhead
Awards

- NSF CAREER Award
- Microsoft Research Faculty Fellowship
- Google Faculty Award
  • 2019, 2021
- Microsoft Research PhD Fellowship
  • Andrew Quinn
- NSF Graduate Research Fellowship
  • Andrew Loveless, Andrew Quinn
- Towner Prize
  • Ian Neal
- OSDI Best Paper Award
- IEEE MICRO Top Pick Honorable Mention

Grants

- NSF, SRC, Google

Collaborations

- UT Austin, KAIST, Intel

Real-world deployment in Windows

~1 billion systems

Adoption at Meta¹, Intel

---

Failure Reproduction and Analysis

OmniTable [OSDI’22]

Debugging in the Brave New World [ASPLOS’22]


Hippocrates [ASPLOS’21] Agamotto [OSDI’20]

---

Snorlax [SOSP’17]

Hippocrates [ASPLOS’21]

Agamotto [OSDI’20]

---

## Outline

<table>
<thead>
<tr>
<th>Datacenter Efficiency</th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data-driven optimizations</td>
<td>Lightweight profiling</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Failure Reproduction and Analysis</th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Static &amp; symbolic program analysis</td>
<td>Selective information monitoring</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hardware Security</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Outline

<table>
<thead>
<tr>
<th></th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datacenter Efficiency</td>
<td>Data-driven optimizations</td>
<td>Lightweight profiling</td>
</tr>
<tr>
<td>Failure Reproduction and Analysis</td>
<td>Static &amp; symbolic program analysis</td>
<td>Selective information monitoring</td>
</tr>
<tr>
<td>Hardware Security</td>
<td>Classification of attacks</td>
<td>Threat model-specific defenses</td>
</tr>
</tbody>
</table>

**Threat model-specific defenses**

**Selective information monitoring**

**Lightweight profiling**

**Static & symbolic program analysis**
Foreshadow bypasses the virtual memory abstraction:
A VM in the Cloud can leak secrets from someone else's VM
Prior work: Eliminating access to a specific channel

We reported the first non-cache microarchitectural attack to Intel

Prior defenses were channel-specific
Principled and Comprehensive Defenses

Our approach: Eliminate attacks at their source by reasoning about information flow and stopping secret propagation

Prior work: Eliminating access to a specific channel

Navigating the efficiency/trustworthiness tension: defenses tailored to threats
NDA: Preventing Speculative Execution Attacks at Their Source

Ofir Weisse  
University of Michigan

Ian Neal  
University of Michigan

Kevin Loughlin  
University of Michigan

Thomas F. Wenisch  
University of Michigan

Baris Kasikci  
University of Michigan
NDA’s Key Insight

Speculative execution attacks require a chain of dependent instructions to access and transmit secrets.

By controlling data propagation, NDA can break these dependency chains, thwarting the code sequences required to mount attacks.
Analysis of Attacks: A Chain of Dependent Instructions

I) Access Phase

II) Transmit Phase
e.g., via the cache

III) Recover Phase
using timing measurements

During speculation

After speculation
Analysis of Attacks: A Chain of Dependent Instructions

I) Access Phase

II) Transmit Phase
   e.g., via the cache

III) Recover Phase
     using timing measurements

Only “unsafe” instructions are not allowed to speculatively transmit secrets

NDA can break the chain of dependent instructions
Unsafe Instructions: A Threat-Model-Centric View

Spectre-like attacks

1. Branch (e.g., jmp, call)
2. Steer control
3. Recover

Speculation

Access Transmit
Unsafe Instructions: A Threat-Model-Centric View

Spectre-like attacks

1. Branch (e.g., jmp, call)
2. Speculation
   - Access
   - Transmit
3. Steer control

Meltdown-like attacks

1. Access (Speculative Load)
2. Transmit
3. Fault handler

In Spectre-like attacks, instructions after a branch are potentially unsafe.

In Meltdown-like attacks, all loads are potentially unsafe.
NDA - Summary

During speculation, NDA:
  • Allows the execution of unsafe instructions (access)
  • Disallows broadcasting the effects of unsafe instructions (transmission)

Much lower overhead than in-order execution (4.8x)
  • 10.7% overhead against Spectre-like attacks
  • 36.1% overhead against Meltdown-like attacks

More comprehensive security than channel-specific defenses
  • Protection against existing and future side-channels
Hardware Security
   MOESI-prime [ISCA’22]
Dolma [SEC’21] NDA [MICRO’19]
Foreshadow [SEC’18] Morpheus [ASPLOS’19]

Awards
   Facebook Fellowship
     • Marina Minkin
   NSF Graduate Research Fellowship
     • Kevin Loughlin
   Google Fellowship
     • Kevin Loughlin
   IEEE MICRO Top Pick
   IEEE MICRO Top Pick Honorable Mention

Grants
   DARPA, ONR

Collaborations
   Microsoft, KU Leuven, Technion, University of Adelaide

Improved cloud security
Processor upgrades and patches
Future Work

Data trends will continue driving software complexity up.

Increased heterogeneity, new interconnects, and more edge devices will bring entirely new efficiency and trustworthiness challenges.

Exploring emerging computer systems problems through the lens of computer architecture, programming languages, and security.
Data Center Efficiency

Trends
Increasing memory, compute, and interconnect heterogeneity
• Multi-tier memory, specialized hardware, diverse interconnects

Future Work
#1: Rethinking profile-guided optimizations (via HW/SW co-design) for heterogeneous systems
Data Center Efficiency

**Trends**

Increasing memory, compute, and interconnect heterogeneity
  • Multi-tier memory, specialized hardware, diverse interconnects

**Future Work**

#1: Rethinking profile-guided optimizations (via HW/SW co-design) for heterogeneous systems

#2: Designing new systems abstractions for resource management in a disaggregated environment
Reliability

**Trends**

Increased heterogeneity and hardware specialization

Always-on profiling/monitoring on the edge and datacenter
  • Primarily used for performance optimizations

**Future Work**

#1: Using production data to rethink our approach to trustworthiness
Reliability

**Trends**

Increased heterogeneity and hardware specialization

Always-on profiling/monitoring on the edge and datacenter

- Primarily used for performance optimizations

**Future Work**

#1: Using production data to rethink our approach to trustworthiness

#2: Techniques for building more reliable heterogeneous systems

[1] Debugging in the Brave New World of Reconfigurable Hardware. Jiacheng Ma, Gefei Zuo, Kevin Loughlin, Andrew Quinn, Baris Kasikci. ASPLOS 2022
Hardware Security

Trends

Microarchitectural isolation is a recurring problem
Increased intermittent and silent hardware errors\textsuperscript{1,2}

Future Work

#1: Microarchitectural isolation as a foundational security primitive

\textsuperscript{1} Cores That Don’t Count, Peter H. Hochschild Paul Jack Turner Jeffrey C. Mogul Rama Krishna Govindaraju Parthasarathy Ranganathan David E Culler Amin Vahdat Proc. HotOS 2021

\textsuperscript{2} Silent Data Corruptions at Scale, Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, Sriram Sankar, Arxiv, 2021
Hardware Security

**Trends**

Microarchitectural isolation is a recurring problem
Increased intermittent and silent hardware errors\(^1,2\)

**Future Work**

#1: Microarchitectural isolation as a foundational security primitive

#2: Techniques for eliminating/reducing hardware errors\(^3\)

---


[2] Silent Data Corruptions at Scale, Harish Dattatraya Dixit, Sneha Pendharkar, Matt Beadon, Chris Mason, Tejasvi Chakravarthy, Bharath Muthiah, Sriram Sankar, Arxiv, 2021

Efficiency

**Datacenter Efficiency**
- Whisper [MICRO’22]
- Thermometer [ISCA’22]
- Twig [MICRO’21]
- PDede [MICRO’21]
- DMon [OSDI’21]
- I-SPY [MICRO’20]
- Ripple [ISCA’21]
- Huron [PLDI’19]
- Cntr [ATC’18]

**Heterogeneous Systems Support**
- Persistent Memory Indexing [FAST’21]
- Optimus [ASPLOS’20]

Trustworthiness

**Failure Reproduction and Analysis**
- OmniTable [OSDI’22]
- Debugging in the Brave New World [ASPLOS’22]
- ER [PLDI’21]
- REPT [OSDI’18]
- Snorlax [SOSP’17]
- Hippocrates [ASPLOS’21]
- Agamotto [OSDI’20]

**Verified Distributed Systems**
- Sift [ATC’22]
- IGOR [RTAS’21]
- I4 [SOSP’19]

**Hardware Security**
- MOESI-prime [ISCA’22]
- Dolma [SEC’21]
- NDA [MICRO’19]
- Foreshadow [SEC’18]
- Morpheus [ASPLOS’19]