Towards Principled Error-Efficient Systems

Sarita Adve
University of Illinois at Urbana-Champaign
sadve@illinois.edu

IOLTS 2020 Keynote

Collaborators:

Abdulrahman Mahmoud, Radha Venkatagiri, Vikram Adve, Khalique Ahmed, Christopher Fletcher, Siva Hari, Maria Kotsifakou, Darko Marinov, Sasa Misailovic, Hashim Sharif, Yifan Zhao, and others

This work is supported in part by DARPA, NSF, a Google Faculty Research Award, and by the Applications Driving Architecture (ADA) Research center (JUMP center co-sponsored by SRC and DARPA)
Errors are becoming ubiquitous
Errors (Must Prevent)

A problem has been detected and Windows has been shut down to prevent damage to your computer.
The problem seems to be caused by the following file: kbdhid.sys

MANUALLY_INITIATED_CRASH
If this is the first time you’ve seen this stop error screen, restart your computer. If this screen appears again, follow these steps:
Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.
- If you have a memory module, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use safe mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.

Technical Information:
*** STOP: 0x00000002 (0x00000000, 0x00000000, 0x00000000, 0x00000000)
*** kbdhid.sys - Address 0x94efd12a base at 0x94efb000 DateStamp 0x4a5bc705

Cosmic particles can change elections and cause planes to fall through the sky, scientists warn
Tiny invisible particles can cause bits of information held by computers to ‘flip’ with potentially serious ramifications

Ian Johnston  Science Correspondent in Boston | @montaukan | Friday 17 February 2017 15:45 |
Errors (Must Prevent)

Designs that aim to prevent all errors

Too expensive for many systems

Key Facilitator: Moore’s Law + Dennard Scaling

Power, Performance, Area are now limiting factors

*Pictures from publicly available sources
Applications Provide Opportunities

Algorithm / Application
Applications Provide Opportunities

Algorithm / Application

Execution

Output

Perfect

Perfect
Applications Provide Opportunities

Algorithm / Application

Execution

Output

→ Perfect
→ Can tolerate errors

→ Perfect
→ Sufficient Quality
Errors 🤡 (Tolerable)

Low-Cost Resiliency

Expensive

Cheap(er)

User Acceptable Output

Prevent all

Tolerate some
Approximate Computing

Expensive

Precise computation

Cheap(er)

Approximate computation

User Acceptable Output
Error-Efficient Systems

Error-Efficient
Only prevent as many (HW or SW) errors as absolutely needed (allow others)

Conserve resources across the system stack
Error-Efficient Systems

Adoption Challenges:

Lack of principled and unified methodologies

Excessive programmer burden
Research Vision

1. Enable error-efficiency as a first-class metric for novice and expert users

2. Principled and unified error-efficiency workflow across the system stack
• Software-centric error analysis and error efficiency: Approxilyzer, Winnow

• Software testing for hardware errors: Minotaur

• Domain-specific error efficiency: HarDNN

• Compiler and runtime for hardware and software error efficiency: ApproxTuner

• Putting it Together: Towards a Discipline for Error-Efficient Systems
Objective of Error Analysis

How do (hardware) errors affect program output?
Error Outcome of Single Error

Sobel

Single error injection

APPLICATION

APPLICATION

Output Corruption!
Quantifying Output Quality

Sobel

Quality Metric (domain specific)

Image difference (rmse)

7% Quality degradation

Single error injection

Output Corruption!

APPLICATION

APPLICATION
Is Output Quality Acceptable?

Quality Metric (domain specific)

Quality Threshold = 10%

Image difference (rmse)

Sobel

7%

Single error injection

Quality degradation

User-Acceptable Output Corruption!
Error Outcome of Single Error

Sobel

APPLICATION
Error Outcome of All Errors

Sobel

APPLICATION
Challenges of Automated Error Analysis

- **Accurate**: Precisely calculate output quality
- **Comprehensive**: All errors (for given error model)
- **Automatic**: Minimal programmer burden
- **Cheap**: Many error injections = expensive!
- **General Methodology**: Applications + Error Models

Meeting ALL of the above requirements is hard
Tools Suite for Automated Error Analysis

• **Accurate**: Precisely calculate output quality

• **Comprehensive**: All errors (for given error model)

• **Automatic**: Minimal Programmer Burden

• **Cheap**: Many error injections = expensive!

• **General Methodology**: Applications + Error Models

Tool Suite: **Relyzer, Approxilyzer, gem5-Approxilyzer, Winnow**
Error Model

- Perturbation in program state (instructions + data)
  - Caused by underlying fault in hardware

- Error Model for Instructions
  - Single bit transient errors in operand registers of dynamic instructions

- Error Model for Data
  - Multi-bit (random 1-bit, 2-bit, 4-bit, 8-bit) transient errors in memory

Error Analysis Output:

**Application Data Error Profile + Application Instruction Error Profile**
Error Analysis User Interface: Inputs

Quality Metric

+ Unmodified Program

+ Quality Threshold (Optional)

Error Analysis
Error Analysis User Interface: Output

Quality Metric +

Unmodified Program +

Quality Threshold (Optional)

Error Analysis

Comprehensive Error Profile
Comprehensive Error Profile

Quality Metric

+ Unmodified Program

+ Quality Threshold (Optional)

Error Analysis

Comprehensive Error Profile

Error outcome (for all error sites)

0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
Error Outcome for One Error Site

Quality Metric

Unmodified Program

Quality Threshold (Optional)

Error Analysis

Comprehensive Error Profile

Error outcome (for one error site)

0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
Error Outcome for One Error Site

Quality Metric +

Unmodified Program +

Quality Threshold (Optional)

Error Analysis

Comprehensive Error Profile

Error outcome (for one error site)

Error Model: Single bit errors in operand registers of dynamic instructions

0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218

Error Site Description
Error Outcome for One Error Site

Quality Metric

Unmodified Program

Quality Threshold (Optional)

Error Analysis

Comprehensive Error Profile

Error outcome (for one error site)

Error Site: Dynamic instruction + Operand Register + Register Bit

0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218

PC + Cycle = Dynamic instruction
Error Outcome for One Error Site

Error Site: Dynamic instruction + Operand Register + Register Bit

0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
Error Outcome for One Error Site

**Quality Metric**

Unmodified Program

+ Quality Threshold (Optional)

---

**Error Analysis**

---

---

**Comprehensive Error Profile**

---

Error outcome (for one error site)

---

**Error Site: Dynamic instruction + Operand Register + Register Bit**

0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218

---

Register Type

Operand Type
Error Outcome for One Error Site

Error Outcome: Impact of an error, at this error site, on program output

0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
Error Outcome for One Error Site

Quality Metric +

Unmodified Program +

Quality Threshold (Optional)

Error Analysis

Comprehensive Error Profile

Error outcome (for one error site)

Error Outcome: Impact of an error, at this error site, on program output

0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218

Quality Degradation
Error Outcome for One Error Site

Error Analysis

Comprehensive Error Profile

Error outcome (for one error site)

Quality Metric

+ Unmodified Program

+ Quality Threshold (Optional)

Error Model: 1-bit transient error in (data bit stored) in DRAM

Address
0x40a670, 342769813038500, Read, 0x6a10a, 2, 7 :: QD - 0.0008

Bit

Quality Degradation

PC + Cycle = Dynamic instruction

Access Type

Byte Offset
Error Outcome for One Error Site

0x400995, 594769813038500, r8, 14, Integer, Source :: QD - 0.0218
Error Outcome for All Error Sites

Quality Metric

Unmodified Program

Quality Threshold (Optional)

Error Analysis

Comprehensive Error Profile

Error outcome (for all error sites)

Billions of error sites in average programs → Error injections in all expensive!
Error Pruning Using Equivalence

Errors flowing through similar control+data paths produce similar outcomes
Errors flowing through similar control+data paths produce similar outcomes.

Equivalence Classes (control + data heuristics)

Inject error in Pilot
Pilot outcome = Outcome of all errors in class

Load-Order-Chain = <PC-B, PC-C>

Few error injections to predict the outcome of all errors
Comprehensive Error Profile with Few Injections

Quality Metric

+ Unmodified Program

+ Quality Threshold (Optional)

Error Analysis

Error outcome (for all error sites)

Up to 5 orders of magnitude reduction is error injections
Validation of Equivalence Heuristics

- Heuristics used to build equivalence classes need validation
  - Does the pilot accurately represent its equivalence class?

Equivalence class (EC)

- Pilot (Representative error-site from EC)
Validation of Equivalence Heuristics

- Heuristics used to build equivalence classes need validation
  - Does the pilot accurately represent its equivalence class?

Equivalence class (EC)

- Pilot (Representative error-site from EC)

Population
Validation of Equivalence Heuristics

- Heuristics used to build equivalence classes need validation
  - Does the pilot accurately represent its equivalence class?

Equivalence class (EC)

Pilot = Error Outcome E

100% Validation Accuracy

All in Population = Error Outcome E

~ 7 million error injections to validate this technique
Validation Accuracy: Data Errors

On average, >97% (up to 99%) validation accuracy
Validation Accuracy: Data Errors

On average, >97% (up to 99%) validation accuracy
Customized Error Efficiency: Use Case 1

Instruction Error Profile → Customized Ultra Low-Cost Resiliency

- Selectively protect instructions
  - End-to-end output quality is not acceptable to user/application
  - Protection Scheme: Instruction duplication
  - Less instructions protected → Reduced resiliency overhead

- Optimal (custom) resiliency solution
  - Quality vs. resiliency coverage vs. overhead
Ultra-Low Cost Resiliency (Water)

Protect All Output Corruptions

99% Resiliency Coverage
Ultra-Low Cost Resiliency (Water)

Significant resiliency overhead savings for small loss of quality

- Protect All Output Corruptions
- Protect All Output Corruptions with Quality Degradation > 1%

99% Resiliency Coverage →

55%
Customized Error Efficiency: Use Case 2

Data Error Profile → Approximate Computing

Identify first-order approximable data in a program
Customized Approximate Computing (FFT)

![Graph showing the data bytes in application in relation to the approximation target. The graph includes a line labeled '1-Bit' and indicates that 90% approximate corresponds to a specific point.]
Customized Approximate Computing (FFT)

77% of data bytes are approximable 90% of the time when corrupted with a single-bit error.
Customized Approximate Computing (FFT)

Data Bytes in Application (%) vs. Approximation Target (%)

- 1-Bit
- 2-Bit
- 4-Bit
- 8-Bit

90% approximate →
Customized Approximate Computing (Swaptions)
Mapping Data to Approximate Memory

Approximate Memory Technique $\rightarrow$ Lower DRAM refresh rate to save power*

*Flikker [ASPLOS’11]
Mapping Data to Approximate Memory

Approximate Memory Technique → Lower DRAM refresh rate to save power*

*Flikker [ASPLOS’11]
Mapping Data to Approximate Memory

Approximate Memory Technique $\rightarrow$ Lower DRAM refresh rate to save power*

*Flikker [ASPLOS’11]

Automatic identification of Critical data

Swaptions

Quality Threshold = $0.001$

Mapping Accuracy = 99.9%

Power Savings = 23%
Outline

- Software-centric **error analysis** and error efficiency: Approxilyzer, Winnow

- **Software testing** for hardware errors: Minotaur

- Domain-specific error efficiency: HarDNN

- Compiler and runtime for hardware and software error efficiency: ApproxTuner

- Putting it Together: Towards a Discipline for Error-Efficient Systems
Minotaur: Key Idea

Analyzing software for...

...hardware errors \approx ...software bugs

Leverage software testing techniques to improve hardware error analysis
Adapts four software testing techniques to hardware error analysis
Input Quality for Error Analysis → PC coverage
High quality (fast) minimized inputs from (slow) standard inputs
Prioritize analyzing specific program locations based on analysis objectives

Terminate analysis (early) when objective is met
Prioritize analysis over fast, (potentially) inaccurate inputs first
Minotaur

Input Quality Checker

Minimizer

Error Analysis

Input Selector

Minimization Objective

Minimizer

Error Analysis Output

Input Prioritization Objective

1. Input quality
2. Input minimization
3. Error injection prioritization
4. Input prioritization

4X average speedup in error analysis
10x average speedup (upto 39x) for analysis targeting low-cost resiliency
18x average speedup (up to 55x) for analysis targeting approximate computing
Outline

• Software-centric error analysis and error efficiency: Approxilyzer, Winnow

• Software testing for hardware errors: Minotaur

• Domain-specific error efficiency: HarDNN

• Compiler and runtime for hardware and software error efficiency: ApproxTuner

• Putting it Together: Towards a Discipline for Error-Efficient Systems
Deep Neural Networks (DNNs)

- Deep Neural Networks (DNNs) used in many application domains
  - Entertainment/personal devices to safety-critical autonomous cars
  - DNN software accuracy is < 100%: ResNet50 on ImageNet is ~76% accurate
  - But must execute “reliably” in the face of hardware errors

- Traditional reliability solution:

  ![Tesla’s Full Self-Driving Chip (FSD), 2019](https://www.extremetech.com/extreme/290029-tesla-will-have-full-self-driving-by-2020-robo-taxis-too)

  ~2X Overhead in area, power

- Can we use domain knowledge to reduce overheads of DNN resilience?
**HarDNN: Approach**

- **Software-directed** approach for hardening CNNs for inference

**Target Granularity**
- Identify DNN component granularity for analysis

**Vulnerability Estimation**
- Efficiently estimate DNN component vulnerability

**Selective Protection**
- Selectively protect to meet coverage and overhead targets

*High, tunable resiliency with low overhead*

**HarDNN**
- **SW**
  - GPU
  - Embedded
  - Future Accelerator

**HW**

*SARA’20, arXiv’20, DSML’20*
HarDNN Challenges

• **What granularity** components to protect?
  – Challenge: Identify granularity for selective protection

• **Which** components to protect?
  – Challenge: Accurately estimate vulnerability of each component

• **How** to protect?
  – Challenge: Low-cost protection mechanism
What Granularity Components to Target?

CNN computation hierarchy

Key computation: Convolution on feature maps
What Granularity Components to Target?

• Full network
  – Rerun inference in its entirety

• Layer
  – Estimate vulnerability of layer, duplicate vulnerable layers by running it twice

• Feature Map
  – Estimate vulnerability of feature map, duplicate vulnerable fmaps by duplicating filters

• Neuron
  – Estimate vulnerability of neuron, duplicate vulnerable neurons

• Instruction
Feature Map (Fmap) Granularity

- Robustness to translational effects of inputs

- Granularity “sweet spot”
  - Fine-grained + composable to layers

<table>
<thead>
<tr>
<th>Network-Dataset</th>
<th>Conv Layers</th>
<th>Fmaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet-ImageNet</td>
<td>5</td>
<td>1,152</td>
</tr>
<tr>
<td>VGG19-ImageNet</td>
<td>16</td>
<td>5,504</td>
</tr>
<tr>
<td>SqueezeNet-ImageNet</td>
<td>26</td>
<td>3,944</td>
</tr>
<tr>
<td>ShuffleNet-ImageNet</td>
<td>56</td>
<td>8,090</td>
</tr>
<tr>
<td>GoogleNet-ImageNet</td>
<td>57</td>
<td>7,280</td>
</tr>
<tr>
<td>MobileNet-ImageNet</td>
<td>52</td>
<td>17,056</td>
</tr>
<tr>
<td>ResNet50-ImageNet</td>
<td>53</td>
<td>26,560</td>
</tr>
</tbody>
</table>
How to Estimate Feature Map Vulnerability

- \( P_{\text{prop}} \) = Probability an error in fmap causes a Top-1 misclassification
- Use statistical injection for neurons within feature map

\[ P_{\text{prop}} = \frac{\# \text{Yes}}{(\text{Total error injections})} \]

- BUT mismatches are relatively rare, takes too many injections to converge
- Insight: Replace binary view of error propagation with continuous view
- Cross-entropy loss: Used to train DNNs to determine/enhance goodness of network
Loss: Continuous Metric for Error Propagation

Insight: Replace binary view of propagation with continuous view
Use cross-entropy loss

![Diagram of Convolutional Neural Network](image)
**Loss: Continuous Metric for Error Propagation**

Insight: Replace binary view of propagation with continuous view
Use cross-entropy loss
Loss: Continuous Metric for Error Propagation

Insight: Replace binary view of propagation with continuous view
Use cross-entropy loss

Input

Convolutional Neural Network

Classification

Feature Maps

Softmax

Loss: 0.54

CAR ✓
TRUCK ✗
BICYCLE ✗
Loss: Continuous Metric for Error Propagation

Our metric: average delta cross entropy loss:

\[ \Delta L_{Fmap} = \frac{\sum_{i}^{N}|L_{golden} - L_{i}|}{N} \]

\[ P_{prop} \text{ for Fmap} = \frac{\Delta L_{Fmap}}{\sum_{i}^{N} \Delta L_{Fmap}} \]
Mismatch vs. Loss: Which Converges Faster?

- How many injections per feature map? Sweep from 64 to 12,288
  - Use Manhattan distance from 12,288 injections to quantify “similarity” of vulnerability estimates

Mismatch and Loss vulnerability estimates converge with increasing injections

Loss converges faster
How to Protect?

• Objective: Duplicate computations (MACs) of vulnerable feature maps

• Duplication Strategy: Filter Duplication
  – Software directed approach: portable across different HW backends
  – Duplicates the corresponding *filter* to recomputes output *fmap*
  – Validate computations off the critical path
Overhead vs. Coverage

Overhead (MACs) sub-linear to coverage
SqueezeNet: 10X reduction in errors for 30% additional computation
Next step: combination with other granularities, prune injection space
Outline

• Software-centric error analysis and error efficiency: Approxilyzer, Winnow

• Software testing for hardware errors: Minotaur

• Domain-specific error efficiency: HarDNN

• Compiler and runtime for hardware and software error efficiency: ApproxTuner

• Putting it Together: Towards a Discipline for Error-Efficient Systems
ApproxTuner: Hardware + Software Approx

- Unified compiler+runtime framework for software and hardware approximations

- Goal:
  For each operation in the application
  - select hardware and/or software approximation with
  - acceptable end-to-end accuracy and maximum speedup (minimum energy)

- Currently for applications with tensor operations; e.g., DNNs

- Example approximations studied
  - Software: Perforated convolutions, filter sampling, reduction sampling
  - Hardware: lower precision, PROMISE analog accelerator [ISCA18]

OOPSLA’19, in review
ApproxTuner Innovations

- Combines multiple software and hardware approximations

- Uses predictive models to compose accuracy impact of multiple approximations

- 3-phase approximation tuning
  - Development-time preserves hardware portability via ApproxHPVM IR
  - Install-time allows hardware-specific approximations
  - Run-time allows dynamic approximation tuning

- Federated Tuning for efficiency at install-time
  - Install-time tuning is expensive under resource constraints
GPU Speedup and Energy Reduction

Approximations: Sampling, Perforation, FP16

2.1x mean speedup and 2x mean energy reduction with 1% QoS loss
Federated vs Empirical: Energy Reduction

Approximations: PROMISE accelerator, Sampling, Perforation, FP16

Federated-p1 gives 4.5x energy reduction, comparable to empirical tuning
Runtime tuning helps maintain responsiveness in face of frequency changes
Outline

- Software-centric error analysis and error efficiency: Approxilyzer, Winnow
- Software testing for hardware errors: Minotaur
- Domain-specific error efficiency: HarDNN
- Compiler and runtime for hardware and software error efficiency: ApproxTuner
- Putting it Together: Towards a Discipline for Error-Efficient Systems
Towards a Discipline for Error-Efficient Systems

End of Moore’s law and Dennard scaling motivate error efficient systems
- Integrate hardware errors in software engineering workflow
- Integrate hardware and software error optimization for error efficient system workflows