Modeling Soft-Error Propagation in Programs

Guanpeng (Justin) Li
Karthik Pattabiraman

Siva Hari
Michael Sullivan
Timothy Tsai
Motivation: Soft Errors

Soft errors becoming more common in processors

Silent Data Corruption (SDC)

Normal Execution → Fault → Error Propagation

- Silent Data Corruption (SDC)
- Crash
- Benign

Exceptions, No Output → Correct Output

Amazon S3 Incident
Software Solutions

Software protection techniques are more flexible and cost-effective!

Increasing Protection Overhead

Soft Error

Impactful Errors
Selective Instruction Duplication

Instruction Sequence

Instruction Duplication

A Knapsack Problem

Selected Instructions for Given Target SDC Coverage

Instruction:
SDC Rate = X%
Overhead = Y%

"The Golden Curve"

Application Specific!

Protection Overhead

*Measured in Libquantum, SPEC
Developing Fault-Tolerant Applications

1. Thousands of fault injections need to be done
2. Repeat every time code is modified
Our Goal

Estimating SDC Rate

Our Goal

SymPLFIED/ Relyzer/ GangES
[DSN’08, ASPLOS’12, ISCA’14]

AVF/ PVF/ ePVF
[MICRO’03, HPCA’10, DSN’16]

No existing technique models error propagation in both fast and accurate way!

Fast prediction of SDC without fault injection!
Challenges

- Tracking SDC propagation is hard
  - Over billions of executed instructions
  - Every instruction may propagate errors with different probabilities
- Dynamic nature of program execution
  - Control-flow divergence

Corrupting subsequent states
Trident: Key Insight

• Error propagations can be decomposed into modules, which can be abstracted into probabilistic events
  - Decomposition
  - Abstraction
Trident: Workflow

Source Code
Program Input
Output Insn.

Profiling
Prediction

Insn. for Prediction
Insn. SDC Rates
Overall SDC Rate
Trident: Our Approach

• Three-level modeling
  • Register-communication
  • Control-flow
  • Memory dependency

\[
\begin{align*}
BB4: & \quad \text{} \quad $2 = \text{LOAD} 0x04 \\
& \quad \text{} \quad $3 = \text{ADD} $2, 4 \\
& \quad \text{} \quad \text{CMP} $4, $3, 4 \\
& \quad \text{} \quad \text{BR} $4, \text{BB5, BB10} \\
BB5: & \quad \text{} \quad $5 = \text{MUL} $6, 16 \\
& \quad \text{} \quad \text{...} \\
BB10: & \quad \text{} \quad \text{...} \\
BB11: & \quad \text{} \quad \text{STORE}, 0x08 \\
BB12: & \quad \text{} \quad \text{...} \\
BB102: & \quad \text{} \quad \text{...} \quad = \text{LOAD} 0x08
\end{align*}
\]
Trident: Register Commn.

\[ f_s = 100\% \times 100\% \times 25\% \times 100\% = 25\% \]
Trident: Control-Flow

\[
f_C = \frac{P_e}{P_d}
\]

*For non-loop-terminating branches

Corruption probability of STORE?

Reg. \( f_S \)

Contl. \( f_C \)

Mem. \( f_M \)
Trident: Memory-Dependency

\[ P(I_n) = f_S(I_n) \times f_C(I_{n2}) \times f_S(I_{n3}) \times f_C(I_{n4}) \ldots \ldots \]

* n corresponds to the index of dynamic instructions
Experimental Setup

• **Comparison with fault injection**
  - Accuracy
  - Speed (wall clock time)

• **Fault Model**
  - Single bit-flip injections – accurate [DSN’17]
  - Random insn. – one per program execution

• **Benchmarks**
  - 11 open-source benchmarks from various domains
Experimental Methodology

- Created two simpler models
  - Accuracy of each sub-model
  - As proxy to prior work
- Baseline: Fault injection derived by LLFI [1]
  - The closer SDC rate to fault injection, the better prediction

Goal is to predict SDC rate as per fault injection

Reminder:

- Baseline: Fault injection derived by LLFI [1]
  - The closer SDC rate to fault injection, the better prediction

Two Simpler Models for Comparison

[1] LLVM Fault Injector [DSN’14]
Evaluation: Accuracy

Program SDC Rate; 3,000 Sampled Instructions; Error Bar: +/-0.07% ~ +/-1.76% at 95% Confidence Interval

**Trident** is close to fault injection results, and significantly better than the simpler models!

- **Mean Absolute Error**
  - Trident: 4.75%
  - Simpler Models: 15.13% and 19.13%

- **t-Test on Individual Instructions**
  - Trident: 8 out of 11 are statistically indistinguishable
  - Simpler Models \(f_S\) and \(f_S + f_C\): Only 2 and 4

3,000 randomly sampled instructions for fault injection and the models
Evaluation: Speed

- Program’s Overall SDC Rate:
  - 6.7x faster at 3,000 samples

- Per-Instruction SDC Rate:
  - On average, 380x faster at 100 samples per instruction
  - Benchmarks: FI takes nearly 100 hours whereas Trident takes <20 mins

**Trident** is faster than fault injection by 2 orders of magnitude!
Use Case: Selective Instruction Duplication

Recap:

Selective Instruction Duplication

="The Golden Curve"

By Fault Injections

By Trident

By $f_s f_c$

By $f_s$

SDC Coverage

Protection Overhead

*Measured in Libquantum, SPEC
Extension

• Understand how error propagation is affected by multiple inputs
• Extension for bounding SDC rate with multiple inputs
Summary

• Fault injections are too slow to integrate into software development cycle

• Trident is both accurate and fast in predicting SDC rates

• Can guide selective protection of instructions in programs – comparable to fault injection in accuracy for fraction of cost

• Open Source: https://github.com/DependableSystemsLab/Trident

Guanpeng (Justin) Li
University of British Columbia (UBC)
gpli@ece.ubc.ca