gem5-Approxilyzer: An Open-Source Tool for Application-Level Soft Error Analysis

Radha Venkatagiri*, Khalique Ahmed*, Abdulrahman Mahmoud, Sasa Misailovic, Darko Marinov, Christopher Fletcher, Sarita Adve

University of Illinois at Urbana-Champaign

* Equal contribution first authors

Supported by NSF and by the Applications Driving Architectures (ADA) Research Center
Motivation

• Modern systems increasingly susceptible to soft errors
  – Too expensive to protect against all errors

• Systems that allow controlled errors becoming popular
  – Approximate computing, low-cost less-than-perfect resiliency solutions

Fundamental question: How do soft errors affect program output?
Error Outcome (of Single Error)

Sobel

Single error injection

Output Corruption!
Quantifying Output Quality

Sobel

Quality Metric
(domain specific)

APPLICATION

Image difference
(rmse)

7%

Quality degradation

APPLICATION

Single error injection

Output Corruption!
Is Output Quality Acceptable?

Sobel

Quality Metric (domain specific)

Quality Threshold = 10%

Image difference (rmse)

7%

Quality degradation

Single error injection

User-Acceptable Output Corruption!
Error Outcome (of Single Error)

Sobel

APPLICATION
Error Outcome (of All Errors)

Sobel Filter

APPLICATION
Application-Level Error Analysis (Holy Grail)

Sobel Filter

Ideal Application Error Analysis:

- Comprehensive
  - All errors (for given error model)
- Accurate
  - Precisely calculate output quality
- Automatic
  - Minimal programmer burden
- Cheap
  - Many error injections = expensive!
• **Approxilyzer**: Tool to determine impact of error on program output
  - Minimal programmer burden, general-purpose, accurate, comprehensive
  - Program analysis + (relatively few) error injections
  - Error Model: Single bit transient errors in operand register + dynamic instructions

Approxilyzer

- Approxilyzer: Tool to determine impact of error on program output
  - Minimal programmer burden, general-purpose, accurate, comprehensive
  - Program analysis + (relatively few) error injections

Prior implementation has limitations
- Proprietary simulator (Wind River Simics)
- Restricted to SPARC ISA

Contributions: gem5-Approxilyzer

- **gem5-Approxilyzer**: Fully open-source implementation of Approxilyzer
  - [https://github.com/rsimgroup/gem5-Approxilyzer](https://github.com/rsimgroup/gem5-Approxilyzer)
  - Built using open-source gem5 simulator
  - Support for x86; non-trivial engineering effort
  - Can be extended to other ISAs

- Show effectiveness/accuracy of Approxilyzer technique for x86
  - Two orders of magnitude reduction in error injections (over naïve techniques)
  - Determines error outcomes with high accuracy (97% on average)

- Error Analysis of application under different ISAs (x86 and SPARC)
  - Error profile of same application significantly different under x86 & SPARC
  - Static instructions that are approximable/resilient vary significantly by ISA
• Introduction

• (gem5-) Approxilyzer interface & techniques
  • gem5-Approxilyzer Use Cases

• Results

• Conclusion
gem5-Approxilyzer: Inputs

End-to-end Quality Metric (domain-specific) +

Unmodified Program

+ Quality Threshold (Optional)

gem5-Approxilyzer
gem5-Approxilyzer: Output

End-to-end Quality Metric (domain-specific) + Unmodified Program + Quality Threshold (Optional)

gem5-Approxilyzer

Comprehensive error outcome profile
Comprehensive Error Outcome Profile

End-to-end Quality Metric (domain-specific)

+ 

Unmodified Program

+ 

Quality Threshold (Optional)

gem5-Approxilyzer

Error outcome (for all error sites)

Comprehensive error outcome profile

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

End-to-end Quality Metric (domain-specific)

+  

Unmodified Program

+  

Quality Threshold (Optional)

gem5-Approxilyzer

Comprehensive error outcome profile

Error outcome (for one error site)

0x400995, 594769813038500, r8, 14, Integer, Source: SDC-Maybe:0.0218

Error Site Description

Error Outcome
Error Outcome for One Error Site

End-to-end Quality Metric (domain-specific)

Unmodified Program

Quality Threshold (Optional)

gem5-Approxilyzer

Comprehensive error outcome profile

Error outcome (for one error site)

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

0x400995, 594769813038500, r8, 14, Integer, Source: SDC-Maybe:0.0218

Error Site Description
End-to-end Quality Metric (domain-specific) +

Unmodified Program

+ Quality Threshold (Optional)

---

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

0x400995, 594769813038500, r8, 14, Integer, Source :: SDC-Maybe:0.0218

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

End-to-end Quality Metric (domain-specific) + Unmodified Program + Quality Threshold (Optional)

#### gem5-Approxilyzer

- Comprehensive error outcome profile
- Error outcome (for one error site)

---

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

0x400995, 594769813038500, r8, 14, Integer, Source :: SDC-Maybe:0.0218

- Register Name
- Register Bit

---

Register Name: r8
Register Bit: 14
Error Outcome for One Error Site

End-to-end Quality Metric (domain-specific)
  +
  Unmodified Program
  +
  Quality Threshold (Optional)

gem5-Approxilyzer

Comprehensive error outcome profile

Error outcome (for one error site)

Error Site: Dynamic instruction + Operand Register + Register Bit

0x400995, 594769813038500, r8, 14, **Integer, Source :: SDC-Maybe:0.0218**

Register Type    Operand Type
Error Outcome for One Error Site

End-to-end Quality Metric (domain-specific)

+ Unmodified Program

+ Quality Threshold (Optional)

gem5-Approxilyzer

Comprehensive error outcome 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 :: SDC-Maybe:0.0218

Error Outcome
Error Outcome Categories for Single Bit Errors

- **Error-free execution**
- **Masked**
- **Detection**
- **Silent Data Corruption (SDC)**
Quality Aware Error Categorization

- Not all output corruptions are bad
  - SDCs can be differentiated based on the quality of data corruptions produced

Silent Data Corruption (SDC)

- SDC-Good: Highly Tolerable
  - Error in non-significant output
  - Quality loss < 0.0001% etc.
- SDC-Maybe: Potentially tolerable
  - Quality > Threshold?
- SDC-Bad: Not Tolerable
  - Quality loss > 100%
- Detectable Data Corruptions (DDC): NaN, infinity etc.
Error Outcome for One Error Site

End-to-end Quality Metric (domain-specific)

+ Unmodified Program

+ Quality Threshold (Optional)

gem5-Approxilyzer

Comprehensive error outcome 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 :: SDC-Maybe:0.0218

Error Outcome
Error Outcome for One Error Site

End-to-end Quality Metric (domain-specific) + Quality Threshold (Optional)

Unmodified Program

gem5-Approxilyzer

Comprehensive error outcome profile

Error outcome (for one error site)

Error Outcome category

0x400995, 594769813038500, r8, 14, Integer, Source :: SDC-Maybe:0.0218

Output quality degradation
Error Injections in All Error Sites Expensive

End-to-end Quality Metric (domain-specific)

- Unmodified Program
- Quality Threshold (Optional)

gem5-Approxilyzer

Comprehensive error outcome profile

Error outcome (for one error site)

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

Do we need to do so many error injections?
Error Injection Pruning

• Error injection pruning
  – Removes redundant error injections

• Error injection pruning technique 1: **Known outcome pruning**
  – No injection required when error outcome is known apriori
Known Outcome Pruning: Address Bound

- Error injection pruning
  - Removes redundant error injection

- Error injection pruning technique 1: **Known outcome pruning**
  - No injection required when error outcome is known apriori

```assembly
mov (%ebx), %eax
```

Address in ebx = 0x4ac8
Known Outcome Pruning: Address Bound

- Error injection pruning
  - Removes redundant error injection

- Error injection pruning technique 1: Known outcome pruning
  - No injection required when error outcome is known apriori
  - Address bound pruning
    * Errors outside address range result in Detected outcomes

```asm
mov (%ebx), %eax
```

Address in ebx = 0x10004ac8

Detection

Outside addressable range => Segmentation Fault
Known Outcome Pruning: Def-Use

- Error injection pruning
  - Removes redundant error injection

- Error injection pruning technique 1: Known outcome pruning
  - No injection required when error outcome is known a priori
  - Address bound pruning
    - Errors outside address range result in Detected outcomes
  - Def-use pruning
    - Injection in a def $\Leftrightarrow$ injection at first use
    - Only one needs to be explored by error injection

```assembly
mov (%ebx), %eax  // Definition of register eax

add %eax, %ecx    // First use of register eax
```
Known Outcome Pruning: Def-Use

- Error injection pruning
  - Removes redundant error injection

- Error injection pruning technique 1: Known outcome pruning
  - No injection required when error outcome is known apriori
  - Address bound pruning
    * Errors outside address range result in Detected outcomes
  - Def-use pruning
    * Injection in a def $\Leftrightarrow$ injection at first use
    * Only one needs to be explored by error injection

\[
\text{mov (\%ebx), \%eax} \quad \text{Definition of register eax}
\]
\[
\text{add \%eax, \%ecx} \quad \text{First use of register eax}
\]
known outcome pruning

- No injection required when error outcome is known a priori

- Address bound pruning
  * Errors outside address range result in detected outcomes

- Def-use pruning
  * Injection in a def ⇔ injection at first use
  * Only one needs to be explored by error injection

```
mov (%ebx), %eax          # Definition of register eax

add %eax, %ecx           # First use of register eax
```
Error injection Pruning: Equivalence

• Error injection pruning
  – Removes redundant error injection

• Pruning technique 1: Known outcome pruning
  – No injection required when error outcome is known apriori
  – Address bound pruning
    * Errors outside address range result in Detected outcomes
  – Def-use pruning
    * Injection in a def ⇔ injection at first use
    * Only one needs to be explored by error injection

• Pruning technique 2: **Equivalence based pruning**
  – Errors are equalized based on heuristics (control + data flow)
  – Only single error injection per equalized set
Error injection pruning: Equivalence

### Insight:
Errors flowing through similar control+data paths produce similar outcomes.

**Equivalence Classes** (using data + control flow heuristics)

- **Inject error in Pilot**
- **Pilot outcome = Outcome of all errors in class**
Comprehensive Error Profile with Few Injections

End-to-end Quality Metric (domain-specific)

+ Unmodified Program

+ Quality Threshold (Optional)

Comprehensive error outcome profile

Few error injections to predict the outcome of virtually all errors
Outline

• Introduction

• (gem5-) Approxilyzer interface & techniques

• gem5-Approxilyzer Use Cases

• Results

• Conclusion
Use Cases of gem5-Approxilyzer

- **End-to-end Quality Metric**
  - Unmodified Program
  - (Optional) Quality Threshold

- **Identify instructions that are approximable**
  - Comprehensive error outcome profile

- **Identify instructions that need resiliency protection**

**Dynamic Instruction A**

<table>
<thead>
<tr>
<th>Reg 1</th>
<th>Reg 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Individual error site information can be composed to higher abstraction
Error Outcome Categorization by Use Case

- Masked
- SDC-Good
- SDC-Maybe
- SDC-Bad
- DDC
- Detected

Error
Which class of errors need resiliency protection?
Which class of errors may be approximable?
• Introduction

• (gem5-) Approxilyzer interface & techniques

• gem5-Approxilyzer Use Cases

• Results

• Conclusion
Effectiveness of Error Pruning Techniques

How effective are the error pruning techniques in reducing error injections?

<table>
<thead>
<tr>
<th>Application</th>
<th>Error Sites</th>
<th>Pruned Error Sites (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total (Before Pruning)</td>
<td>Remaining (After Pruning)</td>
</tr>
<tr>
<td>Blackscholes</td>
<td>232K</td>
<td>100K</td>
</tr>
<tr>
<td>Swaptions</td>
<td>10,300K</td>
<td>720K</td>
</tr>
<tr>
<td>LU</td>
<td>1,200K</td>
<td>268K</td>
</tr>
<tr>
<td>FFT</td>
<td>4,400K</td>
<td>215K</td>
</tr>
<tr>
<td>Sobel</td>
<td>85,300K</td>
<td>300K</td>
</tr>
</tbody>
</table>

Up to two orders of magnitude reduction in error injections
Heuristics used to build equivalence classes need validation
- Does the pilot accurately represent its equivalence class?
Heuristics used to build equivalence classes need validation

- Does the pilot accurately represent its equivalence class?

Equivalence class (EC)

Pilot = SDC-Maybe with quality Q
Validation of Equivalence Heuristics

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

![Equivalence class (EC)](image)

- Pilot = SDC-Maybe with quality Q
- Population (Random sample of error-sites from EC)
- Statistical sample - 99% confidence
Validation of Equivalence Heuristics

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

Equivalence class (EC)

Pilot = SDC-Maybe with quality Q

100% validation accuracy

All in Population = SDC-Maybe with quality Q
Validation of Equivalence Heuristics

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

Equivalence class (EC)

Flexible quality parameter $\rightarrow \delta = 2\%$

Pilot = SDC-Maybe with quality $Q$

100% validation accuracy

All in Population = SDC-Maybe with quality $Q \pm \delta$
Heuristics used to build equivalence classes need validation

- Does the pilot accurately represent its equivalence class?

Equivalence class (EC)

- Pilot = **Approximable**
- 100% validation accuracy
- All in Population = **Approximable**

Flexible quality parameter $\delta = 2\%$

Quality threshold = 5\%
Validation of Equivalence Heuristics

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

  Equivalence class (EC)

  Pilot = Approximable

  100% validation accuracy

  All in Population = Approximable

  Flexible quality parameter → δ = 2%

  Quality threshold = 5%

  Equivalence classes selected at random (99% confidence) for validation

  ~ 1.6 million error injections
High validation accuracy (97% on average) across workloads.
High validation accuracy (97% on average) across workloads
Error Outcomes profiles for different ISAs

Error profiles for the same workloads vary significantly for x86 vs SPARC
Comparison of static PCs by use case

PCs that need resiliency protection

PCs that may be approximable

Instructions for resiliency protection and approximation vary based on ISA
Conclusions

• gem5-Approxilyzer: Open-source tool for comprehensive error analysis
  – https://github.com/rsimgroup/gem5-Approxilyzer
  – Built using open-source gem5 simulator
  – Support for x86; can be extended to other ISAs

• Show effectiveness/accuracy of Approxilyzer technique for x86
  – Two orders of magnitude reduction in error injections
  – Determines error outcomes with high accuracy (97% on average)

• Error Analysis of application under different ISAs (x86 and SPARC)
  – Error profile of same application significantly different under different ISAs
  – Static instructions that are approximable/resilient vary significantly by ISA
https://github.com/rsimgroup/gem5-Approxilyzer

gem5-Approxilyzer

End-to-end Quality Metric (domain-specific)

+ Unmodified Program

+ Quality Threshold (Optional)

gem5-Approxilyzer

Comprehensive error outcome profile

Approxilyzer is an open-source framework for instruction level approximation and resiliency software. Approxilyzer provides a systematic way to identify instructions that exhibit first-order approximation potential. It can also identify silent data corruption (SDC) causing instructions in the presence of single-bit errors. Approxilyzer employs static and dynamic analysis, in addition to heuristics, to reduce the run-time of finding Approximate instructions and SDC-causing instructions by 3-6x orders of magnitude.

Project overview can be viewed at: https://cs.illinois.edu/approxilyzer
Backup Slides
gem5-Approxilyzer end-to-end flow

1. Application compiled to x86 --> Generate simplified dynamic gem5 trace

2. Error site pruning --> Generate list of pilots for error injection

3. Perform error injections --> Calculate error outcome category and quality

4. Assign outcome to population --> Generate error outcome profile
Gem5-Approxilyzer Pruning by Technique

TABLE I
BENCHMARKS, INPUTS, AND ERROR-SITE PRUNING BY TECHNIQUE (C: CONTROL-EQUIVALENCE, S: STORE-EQUIVALENCE, C+S+K: TOTAL PRUNING USING CONTROL, STORE, AND KNOWN-OUTCOME TECHNIQUES)

<table>
<thead>
<tr>
<th>Application</th>
<th>Input</th>
<th>Error Sites</th>
<th>Pruned Error Sites (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Total</td>
<td>Remaining</td>
</tr>
<tr>
<td>Black-scholes [37]</td>
<td>21 options</td>
<td>232K</td>
<td>100K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Swaptions [37]</td>
<td>1 option 1 simulation</td>
<td>10.3M</td>
<td>720K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LU [38]</td>
<td>16x16 matrix 8x8 blocks</td>
<td>1.2M</td>
<td>268K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FFT [38]</td>
<td>$2^{20}$ data points</td>
<td>4.4M</td>
<td>215K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sobel [3]</td>
<td>81x121 pixels</td>
<td>85.3M</td>
<td>300K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>