Rely: Verifying Quantitative Reliability for Programs that Execute on Unreliable Hardware

Michael Carbin, Sasa Misailovic, and Martin Rinard

MIT CSAIL
Image Scaling Kernel: Bilinear Interpolation

\[ f(\begin{array}{c} \vdots \end{array}) = \begin{array}{c} \vdots \end{array} \]
Bilinear Interpolation

```c
int bilinear_interpolation(int i, int j,
                           int src[][[]], int dest[][[]])
{
    int i_src = map_y(i, src, dest),
    j_src = map_x(j, src, dest);

    int up   = i_src - 1, down  = i_src + 1,
    left = j_src - 1, right = j_src + 1;

    int val = src[up][left] + src[up][right] +
              src[down][right] + src[down][left];

    return 0.25 * val;
}
```
Bilinear Interpolation

```c
int bilinear_interpolation(int i, int j,  
    int src[][[]], int dest[][[]])
{
    int i_src = map_y(i, src, dest),  
        j_src = map_x(j, src, dest);

    int up   = i_src - 1, down  = i_src + 1,  
        left = j_src - 1, right = j_src + 1;

    int val = src[up][left] + src[up][right] +  
              src[down][right] + src[down][left];

    return 0.25 * val;
}
```
Unreliable Hardware

Unreliable Units (ALUs and Memories)

- May produce incorrect results
- Faster, smaller, and lower power
Image Scaling with Approximate Bilinear Interpolation

Reliability

20%  40%  60%  80%  90%  99%  99.9%
Unreliable Hardware

Necessitates

- Hardware Specification: probability operations execute correctly
- Software Specification: required reliability of computations
- Analysis: verify software satisfies its specification on hardware
Rely: a Language for Quantitative Reliability

- Hardware Specification (Architect)
- Software Specification (Developer)
- Static Analysis (Language)

Reliability

- 20%
- 40%
- 60%
- 80%
- 90%
- 99%
- 99.9%
Hardware Specification

```plaintext
hardware {
  operator (+) = 1 - 10^{-7};
  operator (-) = 1 - 10^{-7};
  operator (*) = 1 - 10^{-7};
  operator (<) = 1 - 10^{-7};
  memory urel {rd = 1 - 10^{-7}, wr = 1};
}
```
Approximate Bilinear Interpolation in Rely

```c
int bilinear_interpolation(int i, int j,
    int src[][[]], int dest[][[]])
{
    int i_src = map_y(i, src, dest),
        j_src = map_x(j, src, dest);

    int up   = i_src - 1, down  = i_src + 1,
        left = j_src - 1, right = j_src + 1;

    int val = src[up][left] + . src[up][right] + .
        src[down][right] + . src[down][left];

    return 0.25 * . val;
}
```

**Unreliable Operations:** executed on unreliable ALUs
Approximate Bilinear Interpolation in Rely

```c
int bilinear_interpolation(int i, int j, int in urel src[][[]], int in urel dest[][[]]) {
    int i_src = map_y(i, src, dest),
    j_src = map_x(j, src, dest);

    int up   = i_src - 1, down  = i_src + 1,
    left = j_src - 1, right = j_src + 1;

    int in urel val = src[up][left]    +. src[up][right] +.
                    src[down][right] +. src[down][left];

    return 0.25 *. val;
}
```

**Unreliable Memories:** stored in unreliable SRAM/DRAM
What is reliability?
Semantics of Reliability

- **Reliable Hardware**
  - One Execution
- **Unreliable Hardware**
  - Multiple Executions
- **Reliability**
  - Probability unreliable execution reaches same state
  - Or, $R(\{x, y\}) = \text{probability over distribution of states that } x \text{ and } y \text{ (only) have correct values.}$
Semantics of Reliability

• **Reliable Hardware**
  • One Execution

• **Unreliable Hardware**
  • Multiple Executions

• **Reliability**
  • Probability unreliable execution reaches same state

• Or, $R(\{x, y\}) = \text{probability over distribution of states that } x \text{ and } y \text{ (only) have correct values.}$
Approximate Bilinear Interpolation
Reliability Specification

```c
int bilinear_interpolation(int i, int j,
    int in urel src[][[]], int in urel dest[][[]]);
```
Approximate Bilinear Interpolation
Reliability Specification

```
int<.99>
bilinear_interpolation(int i, int j,
                        int in urel src[][[]], int in urel dest[][[]]);
```

- Reliability of output is a function of reliability of inputs
Approximate Bilinear Interpolation Reliability Specification

\[
\int <.99 \times R(i, j, \text{src}, \text{dest})>
\]

\[
bilinear\_interpolation(int i, int j, 
\quad int \text{in urel src[][]], int in urel dest[][]});
\]

- Reliability of output is a function of reliability of inputs
- The term \(R(i, j, \text{src}, \text{dest})\) abstracts the joint reliability of the function’s inputs on entry
Approximate Bilinear Interpolation

Reliability Specification

```
int<.99 * R(i, j, src, dest)>
bilinear_interpolation(int i, int j,
    int in urel src[][[]], int in urel dest[][[]]);
```

- Reliability of output is a function of reliability of inputs
- The term $R(i, j, src, dest)$ abstracts the joint reliability of the function’s inputs on entry
- Coefficient .99 bounds reliability degradation
How does Rely verify reliability?
Rely’s Analysis Framework

- **Precondition** generator for statements

\[ \bigwedge_i r_i \ast R(\{x_{i1}, \ldots, x_{in}\}) \leq r'_i \ast R(\{x'_{i1}, \ldots, x'_{im}\}) \]

**Specification**

\[ 0.9 \ast R(\{x\}) \leq 0.99 \ast R(\{y\}) \]
Assignment Rule

\[ \{r_1 \cdot R(\{x_1, \ldots, x_n\}) \leq r_2 \cdot \text{rel}(e) \cdot \text{wr}(x') \cdot R(\{x'_1, \ldots, x'_m\} \cup f v(e))\} \]

\[ x' = e \]

\[ \{r_1 \cdot R(\{x_1, \ldots, x_n\}) \leq r_2 \cdot R(\{x'_1, \ldots, x', \ldots, x'_m\})\} \]
Assignment Rule

\[ \{ r_1 \ast R(\{x_1 \ldots, x_n\}) \leq r_2 \ast \text{rel}(e) \ast \text{wr}(x') \ast R(\{x'_1, \ldots, x'_m\} \cup f\nu(e)) \} \]

Unmodified

\[ \{ r_1 \ast R(\{x_1 \ldots, x_n\}) \leq r_2 \ast R(\{x'_1, \ldots, x', \ldots, x'_m\}) \} \]

\[ x' = e \]
Assignment Rule

\[
\{ r_1 \ast R(\{x_1 \ldots, x_n\}) \leq r_2 \ast \text{rel}(e) \ast \text{wr}(x') \ast R(\{x'_1, \ldots, x'_m\} \cup f\nu(e))\}
\]

\[
x' = e
\]

\[
\{ r_1 \ast R(\{x_1 \ldots, x_n\}) \leq r_2 \ast R(\{x'_1, \ldots, x', \ldots, x'_m\})\}
\]
Assignment Rule

\[
\{ r_1 \ast R(\{x_1 \ldots , x_n\}) \leq r_2 \ast rel(e) \ast wr(x') \ast R(\{x'_1, \ldots , x'_m\} \cup fv(e)) \} \\
\ \\
\{ r_1 \ast R(\{x_1 \ldots , x_n\}) \leq r_2 \ast R(\{x'_1, \ldots , x', \ldots , x'_m\}) \}
\]

- \( rel(e) \ast wr(x') \) is the probability the expression and write execute correctly
Verifying the Reliability of Bilinear Interpolation

```c
int<.99 * R(i,j,src,dest)>
bilinear_interpolation(int i, int j,
    int in urel src[][[]], int in urel dest[][[]])
{
    int i_src = map_y(i, src, dest),
        j_src = map_x(j, src, dest);

    int up   = i_src - 1, down  = i_src + 1,
        left = j_src - 1, right = j_src + 1;

    int in urel val = src[up][left]    +. src[up][right] +.
                     src[down][right] +. src[down][left];

    return 0.25 * . val;
}
```
Verifying the Reliability of Bilinear Interpolation

1. Generate postcondition from return statement

```
return 0.25 *. val;  \rightarrow  .99 * R(src, i, j, dest) \leq rd(val) * op(*) * R(val)
```

2. Work backwards to produce verification condition

```
.99 * R(src, i, j, dest) \leq rd(val) * op(*) * rd(src)^4 * op(+)^3 * wr(val) * R(src, i, j, dest)
```

3. Use hardware specification to replace reliabilities

Reliability of return
Reliability of sum of neighbors

```
.99 * R(src, i, j, dest) \leq (1 - 10^{-7}) * (1 - 10^{-7}) * (1 - 10^{-7})^4 * (1 - 10^{-7})^3 * 1.0 * R(src, i, j, dest)
```
Verifying the Reliability of Bilinear Interpolation

1. Generate postcondition from return statement

\[
\text{return } 0.25 \times \text{val}; \\
0.99 \times R(\text{src}, i, j, \text{dest}) \leq \text{rd}(\text{val}) \times \text{op}(\times) \times R(\text{val})
\]

2. Work backwards to produce verification condition

\[
0.99 \times R(\text{src}, i, j, \text{dest}) \leq \text{rd}(\text{val}) \times \text{op}(\times) \times \text{rd}(\text{src})^4 \times \text{op}(+)^3 \times \text{wr}(\text{val}) \times R(\text{src}, i, j, \text{dest})
\]

3. Use hardware specification to replace reliabilities

\[
0.99 \times R(\text{src}, i, j, \text{dest}) \leq 0.999999 \times R(\text{src}, i, j, \text{dest})
\]

4. Discharge Verification Condition
Verification Condition
Checking Insight

$$\bigwedge_i r_i \cdot R\{x_{i1}, \ldots, x_{in}\} \leq r'_i \cdot R\{x'_{i1}, \ldots, x'_{im}\}$$

Computing full **joint distributions** is intractable and input distribution dependent

$$\{x'_{1}, \ldots, x'_{m}\} \subseteq \{x_{1}, \ldots, x_{n}\} \rightarrow R(\{x_{1}, \ldots, x_{n}\}) \leq R(\{x'_{1}, \ldots, x'_{m}\})$$
Conjunct Checking

• A conjunct is implied by a pair of constraints

\[ r_1 \leq r_2 \quad \{x'_1, \ldots, x'_m\} \subseteq \{x_1, \ldots, x_n\} \]

\[ r_1 \cdot R(\{x_1, \ldots, x_n\}) \leq r_2 \cdot R(\{x'_1, \ldots, x'_m\}) \]

• Decidable, efficiently checkable, and input distribution agnostic
Verification Condition Checking for Approximate Bilinear Interpolation

\[ .99 \leq .999999 \quad \{\text{src, i, j, dest}\} \subseteq \{\text{src, i, j, dest}\} \]

\[ .99 \times R(\{\text{src, i, j, dest}\}) \leq .999999 \times R(\{\text{src, i, j, dest}\}) \]
What about...programs?
(conditionals, loops, and functions)
Conditionals

if (y > .0) 

x = x + .1 

x = 2 * . x + .1
Conditionals

\[ \ell = y > \theta \]

if (\ell)

\[ x_1 = x_0 + 1 \]

\[ x_2 = 2 \times x_0 + 1 \]
Conditionals

\[ \ell = y > . \theta \]
\[ \text{if } (\ell) \]

\[ x_1 = x_0 + . 1 \]

\[ x_2 = 2 * . x_0 + . 1 \]

\[ x = \phi (\ell, x_1, x_2) \]
\(\ell = y > 0\) 
\[\text{if } (\ell)\]

\[x_1 = x_0 + 1\]

\[x_2 = 2 \cdot x_0 + 1\]

\[\text{Spec} \leq R(\ell, x_1)\]

\[\text{Spec} \leq R(\ell, x_2)\]

\[x = \phi(\ell, x_1, x_2)\]

\[\text{Spec} \leq R(x)\]
\[
\ell = y > \theta \\
\text{if } (\ell)
\]

\[
x_1 = x_0 + 1
\]

\[
\text{Spec} \leq \text{op}(+) \cdot R(\ell, x_0)
\]

\[
x_2 = 2 \cdot x_0 + 1
\]

\[
\text{Spec} \leq \text{op}(+) \cdot \text{op}(\cdot) \cdot R(\ell, x_0)
\]

\[
x = \phi (\ell, x_1, x_2)
\]

\[
\text{Spec} \leq R(x)
\]
\[ ℓ = y > . \theta \]

\[ if (ℓ) \]

\[ x_1 = x_0 + . 1 \]

\[ Spec \leq op(+.) \cdot R(ℓ, x_0) \]

\[ Spec \leq R(ℓ, x_1) \]

\[ x_2 = 2 \cdot x_0 + . 1 \]

\[ Spec \leq op(+.) \cdot op(*) \cdot R(ℓ, x_0) \]

\[ Spec \leq R(ℓ, x_2) \]

\[ x = \phi (ℓ, x_1, x_2) \]

\[ Spec \leq R(x) \]
\[ \ell = y >. 0 \]
if (\ell)

\[ x_1 = x_0 +. 1 \]
\[ x_2 = 2 * . x_0 +. 1 \]

\[ \text{Spec} \leq \text{op}(+) \cdot \text{op}(\cdot) \cdot \text{R}(x_0, y) \land \text{Spec} \leq \text{op}(+) \cdot \text{op}(\cdot) \cdot \text{R}(x_0, y) \]

\[ x_1 = x_0 +. 1 \]
\[ x_2 = 2 * . x_0 +. 1 \]

\[ \text{Spec} \leq \text{op}(+) \cdot \text{op}(\cdot) \cdot \text{op}(\cdot) \cdot \text{R}(x_0, y) \land \text{Spec} \leq \text{op}(+) \cdot \text{op}(\cdot) \cdot \text{op}(\cdot) \cdot \text{R}(x_0, y) \]

\[ x = \phi(\ell, x_1, x_2) \]

\[ \text{Spec} \leq \text{R}(x) \]
Simplification

\[ \text{Spec} \leq \text{op}(+) \cdot \text{op}(>) \cdot R(x_0, y) \quad \land \quad \text{Spec} \leq \text{op}(+) \cdot \text{op}(\cdot) \cdot \text{op}(>) \cdot R(x_0, y) \]

\[ \ell = y > \cdot 0 \quad \text{if } (\ell) \]

\[ x_1 = x_0 + .1 \]

\[ x_2 = 2 \cdot x_0 + .1 \]

\[ x = \phi (\ell, x_1, x_2) \]

\[ \text{Spec} \leq R(x) \]
Simplification

\[ \ell = y > \theta \quad \text{if } (\ell) \]

\[ \text{Spec} \leq \text{op}(+) \cdot \text{op}(\ast) \cdot \text{op}(>) \cdot R(x_0, y) \land \]

\[ \text{Spec} \leq \text{op}(+) \cdot \text{op}(\ast) \cdot \text{op}(>) \cdot R(x_0, y) \]

\[ x_1 = x_0 + 1 \]

\[ \text{Spec} \leq R(\ell, x_1) \]

\[ x_2 = 2 \ast x_0 + 1 \]

\[ \text{Spec} \leq R(\ell, x_2) \]

\[ x = \phi (\ell, x_1, x_2) \]

\[ \text{Spec} \leq R(x) \]
Loops

- Reliability of loop-carried, unreliably updated variables decreases monotonically.

```c
int sum = 0;
for (int i = 0; i < n; i = i + 1) {
    sum = sum + a[i];
}
```

- Finitely Bounded Loops: bounded decrease.
- Unbounded loops: conservative result is 0.
Functions

- Verification is **modular** (assume/guarantee)

\[
\text{int}<r_f * R(x)> f(x);
\]

\[
\begin{align*}
 r_1 * R(X) & \leq r_2 * r_f * R(Y \setminus \{y\} \cup \{x\}) \\
 y &= f(x); \\
 r_1 * R(X) & \leq r_2 * R(Y)
\end{align*}
\]

- Recursion similar to loops: unreliably updated variables naturally have 0 reliability
Rely: a Language for Quantitative Reliability

- Hardware Specification (Architect)
- Software Specification (Developer)
- Static Analysis (Language)

Reliability:
- 20%
- 40%
- 60%
- 80%
- 90%
- 99%
- 99.9%
Evaluation

• **Experiment #1**: verify specifications
  • How does the analysis behave?
Benchmarks

- **newton**: zero-finding using Newton’s method
- **secant**: zero-finding using Secant Method
- **coord**: Cartesian to polar coordinate converter
- **search_ref**: motion estimation
- **mat_vec**: matrix-vector multiply
- **hadamard**: frequency-domain pixel-block difference metric
Experiment #1: Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LOC</th>
<th>Time (ms)</th>
<th>Conjuncts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>w/o</td>
</tr>
<tr>
<td>newton</td>
<td>21</td>
<td>8</td>
<td>82</td>
</tr>
<tr>
<td>secant</td>
<td>30</td>
<td>7</td>
<td>16356</td>
</tr>
<tr>
<td>coord</td>
<td>36</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>search_ref</td>
<td>37</td>
<td>348</td>
<td>36205</td>
</tr>
<tr>
<td>matvec</td>
<td>32</td>
<td>110</td>
<td>1061</td>
</tr>
<tr>
<td>hadamard</td>
<td>87</td>
<td>18</td>
<td>3</td>
</tr>
</tbody>
</table>

Observation: small number of conjuncts with simplification
Evaluation

• **Experiment #2:** application scenarios
  • How to use reliabilities?
Checkable Computations

• A **simple checker** can validate whether the program produced a correct result

• Execution time optimization:

\[ T_{\text{reliable}} \text{ vs. } T_{\text{unreliable}} + T_{\text{checker}} + (1 - r) \cdot T_{\text{reliable}} \]
Approximate Computations

High Quality

Bilinear Interpolation Reliability (as Negative Log Failure Probability)

Target Reliability

High Quality
Other Concerns for Unreliable Hardware

**Safety:** does the program always produce a result?
- no failures or ill-defined behaviors
  
  [Misailovic et al. ICSE ’10; Carbin et al. ISSTA ’10; Sidiroglou et al. FSE’11; Carbin et al., PLDI ’12; Carbin et al., PEPM ’13]

**Accuracy:** is result accurate enough?
- small expected error
  
  [Rinard ICS’06; Misailovic et al.,ICSE ’10; Hoffmann at al. ASPLOS ’11; Misailovic et al. SAS ’11; Sidiroglou et al. FSE’11; Zhu et al. POPL ’12; Misailovic et al. RACES ’12]
Takeaway

• Separating approximate computation isn’t enough
• Acceptability of results depends on reliability

Rely

• Architect provides hardware specification
• Developer provides software specification
• Rely provides verified reliability guarantee
Backup Slides
Semantic Model

- Execution of e is a stochastic process
- Independent probability of failure for each operation
- Reliability is probability of fully reliable path
Semantic Formalization

- **Probabilistic transition system**
  \[ \langle s, \sigma \rangle \xrightarrow{p} \langle s', \sigma' \rangle \]

- Set of possible executions on unreliable hardware gives **distributions** of states
  \[ \phi \in \Sigma \rightarrow \mathbb{R} \quad \langle s, \phi \rangle \Rightarrow \phi \]

- **Predicates** defined over distributions
  \[ \llbracket P \rrbracket \in \wp(\Phi) \]

See paper for inference rules!
Identifying Reliably Update Variables

- Reliably updated vs. unreliably updated variables

```c
int sum = 0;
for (int i = 0; i < n; i = i + 1)
{
    sum = sum + a[i];
}
```

- Dependence graph gives classification
- Reliably updated variables have same reliability