Compiling for Resilience:  
The Performance Gap

Norman A. Rink, Dmitrii Kuvaiskii, Jeronimo Castrillon, and Christof Fetzer
Technische Universität Dresden

Symposium on ERPP, ParCo 2015
4 September 2015
Edinburgh, Scotland
Motivation

- Frequency of transient HW faults (aka. soft errors) is likely to increase.
  - Smaller feature sizes, denser packing.
  - Cosmic rays.
  - Dark/dim silicon: operating at near-threshold voltage.

- Protecting against faults in HW is expensive and inflexible.
  - Requires additional area and power.
  - Extends development cycles and verification.

- SW-based protection is …
  - more flexible, adaptable to changing faulting scenarios.
  - expensive in the sense of performance degradation.

Source: http://wipac.wisc.edu/deco

Norman Rink, norman.rink@tu-dresden.de
Motivation: Encoding

- SW-based approach: Encoding.
- Transform programs so that ... 
  - data is represented by valid code words.
  - operations are carried out on code words.
- Previously: encoding implemented as source-to-source transformation.
  - Slow-down of 3x-250x.
- Goal:
  - Implement encoding at LLVM IR level.
  - Apply compiler techniques to optimize encoded programs.
- Here: assess scope for an optimizing resilience-aware compiler.
Choose constant $A$.
Integer variable $n \rightarrow n' = n \times A$.

```c
int64_t encode(int64_t x) {
    return x * A;
}

int64_t decode(int64_t x) {
    return x / A;
}

void check(int64_t x) {
    if (x % A)
        exit(AN_ERROR);
}
```

Replace operations with encoded versions.

```c
int64_t add_enc(int64_t x, int64_t y) {
    return x + y;
}

int64_t mul_enc(int64_t x, int64_t y) {
    return ((__int128_t)x * ((__int128_t)y) / A;
}

int64_t and_enc(int64_t x, int64_t y) {
    int64_t xp = decode(x);
    int64_t yp = decode(y);
    return encode(xp & yp);
}
```

Arithmetic and bitwise operations only.
AN Encoding: example

(1) original data-flow graph

(2) after value encoding, A=3

(3) after operation encoding

(4) after check insertion

Norman Rink, norman.rink@tu-dresden.de
Main ideas:
- Checking is expensive (modulo operation).
- Optimize by “strength reduction” and omission.

Different checking strategies:
- COMPCHK: “comprehensive”, i.e. perform modulo check on every argument.
- ACCUCHK: accumulate arguments and perform a modulo check of the accumulator only once per function.
- OMIACCU: omit every other accumulation.
Fault model

- Transient faults, only one fault is injected per program run.

- Bit-flips in the following places:
  - r/w of registers
  - r/w of memory locations
  - write to address bus
Results I: Fault Coverage

- Fault injection campaign:
  - 5 test cases
  - 3 encoding strategies
  - 36,000 random faults
  - \( \rightarrow 540,000 \) experiments

- Classify results in five categories:
  - CORRECT, ANCRASH, OSCRASH, HANG
  - Silent Data Corruption (SDC)

- Variation in fault coverage: < 5%.
- Shift in proportions.
Results I: Performance Degradation

- Speed-up of up to 2x.
- Matrix-Vector Mult. bound by expensive multiplication.

Norman Rink, norman.rink@tu-dresden.de
Results II

- Speed up of 1.2x-1.3x.
- Unexpected peak at ACCUCHK, cf. Matrix-Vector Mult.

<table>
<thead>
<tr>
<th>slow-down</th>
<th>COMPCHK</th>
<th>ACCUCHK</th>
<th>OMIACCU</th>
<th>input</th>
</tr>
</thead>
<tbody>
<tr>
<td>DES</td>
<td>6.15</td>
<td>4.94</td>
<td>4.64</td>
<td>256KB</td>
</tr>
<tr>
<td>CRC</td>
<td>2.19</td>
<td>1.77</td>
<td>1.95</td>
<td>1.4MB</td>
</tr>
</tbody>
</table>

Norman Rink, norman.rink@tu-dresden.de
Conclusion and Outlook

- Programs react differently to strength reduction and omission of checks.
  - Reduction algorithms are particularly robust.
  - Placement of encoding/checking instructions can “artificially” improve coverage.
  - Need approach for programs where performance does not benefit from our strategies.

**Resilience-aware Compiler**

- close the gap: minimize the number of additional instructions
- exploit the trade-off between performance degradation and fault coverage
- apply different protections mechanisms to different program sections as appropriate

Norman Rink, norman.rink@tu-dresden.de
Compiling for Resilience: The Performance Gap

Norman A. Rink, Dmitrii Kuvaiskii, Jeronimo Castrillon, and Christof Fetzer
Technische Universität Dresden

Thank you.
Compiling for Resilience: The Performance Gap

Norman A. Rink, Dmitrii Kuvaiskii, Jeronimo Castrillon, and Christof Fetzer
Technische Universität Dresden

Back up slides
All Faults vs. Registers Only
All Faults vs. Registers Only II