DRIFT: Decoupled compileR-based Instruction-level Fault-Tolerance

Konstantina Mitropoulou [†], Vasileios Porpodas [†] and Marcelo Cintra[†⋆]

School of Informatics, University of Edinburgh [†]
Intel Labs Braunschweig[⋆]

LCPC 2013

Outline

- Transient Errors
- Compiler-based Error Detection
- DRIFT
- Performance & Fault Coverage Evaluation
- Conclusions

Transient Errors

As hardware errors become more frequent
$\hookrightarrow$ increased need for high-reliable and low-overhead error detection methodologies

Main sources of hardware errors :

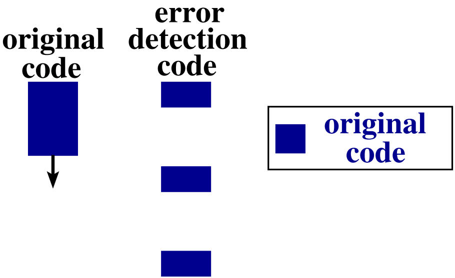
- small transistor technologies
- voltage scaling

Transient errors are:

- temporal phenomena
- the most frequent type of errors
- easy to handle at run-time

Compiler-based Error Detection

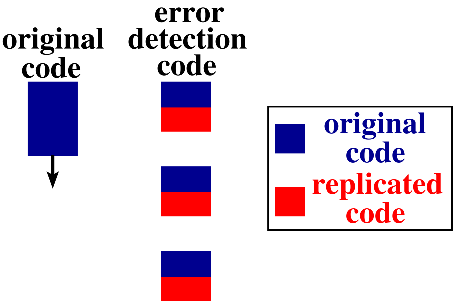
Dual-modular error detection:



Compiler-based Error Detection

Dual-modular error detection:

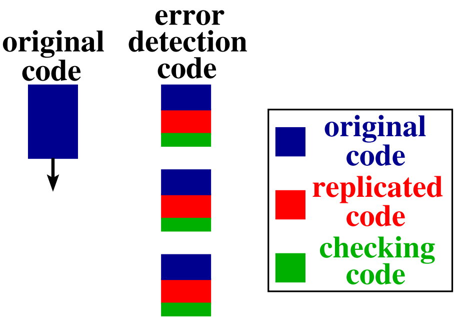
- *replicates* the computation



Compiler-based Error Detection

Dual-modular error detection:

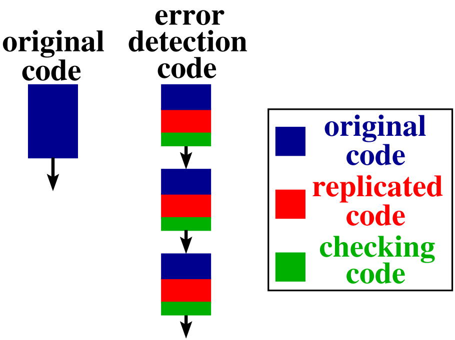
- *replicates* the computation
- *compares* the two outputs



Compiler-based Error Detection

Dual-modular error detection:

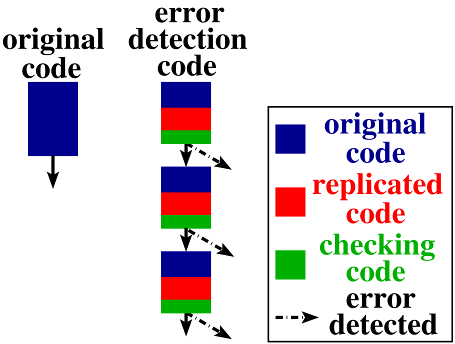
- *replicates* the computation
- *compares* the two outputs
- if the outputs are identical, then the execution *continues* normally



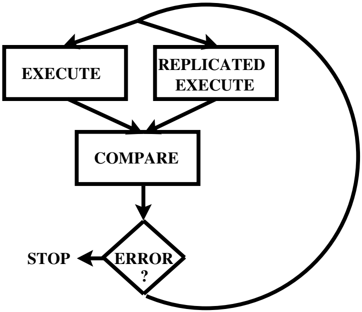
Compiler-based Error Detection

Dual-modular error detection:

- *replicates* the computation
- *compares* the two outputs
- if the outputs are identical, then the execution *continues* normally
- in case of an error, the execution *rolls back* to the last checkpoint

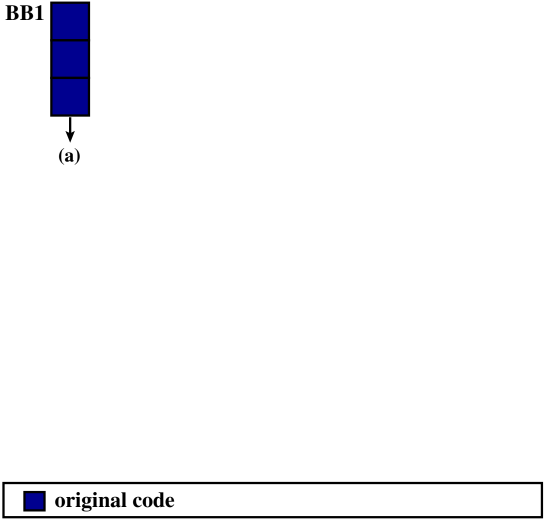


Synchronized Error Detection



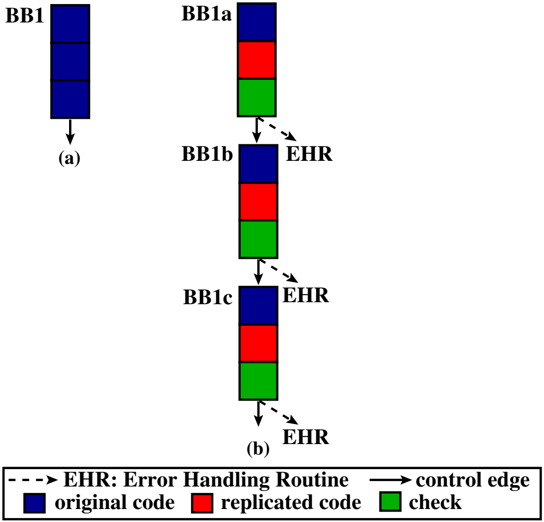
Synchronized Error Detection

Basic-block Fragmentation Problem



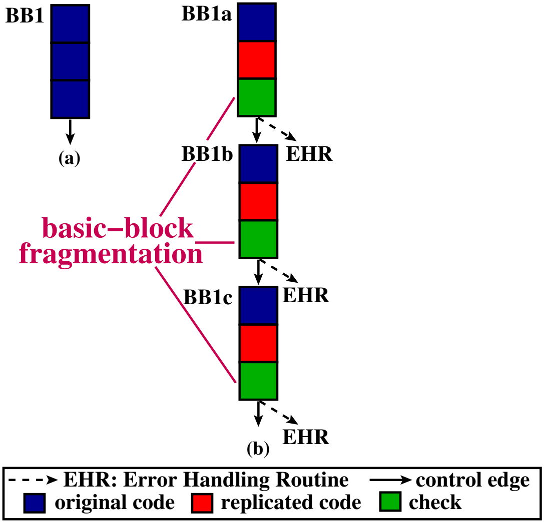
(a) No Error Detection

Basic-block Fragmentation Problem



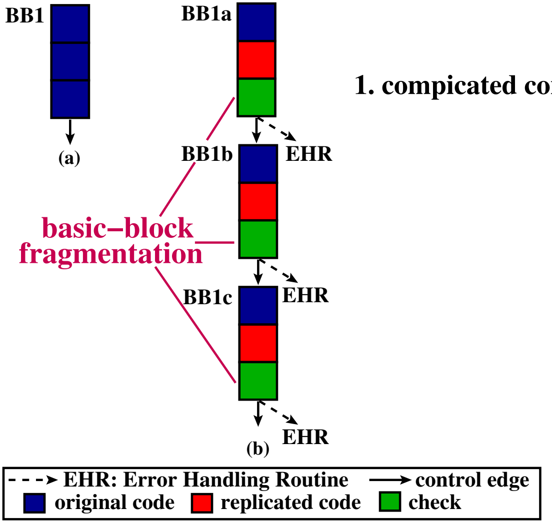
(a) No Error Detection (b) Synchronized Error Detection

Basic-block Fragmentation Problem



(a) No Error Detection (b) Synchronized Error Detection

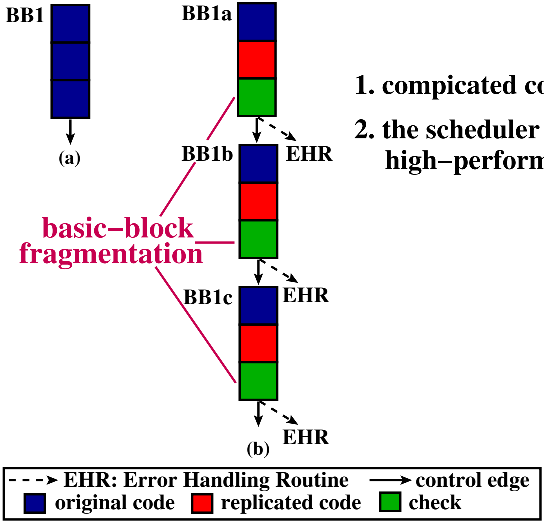
Basic-block Fragmentation Problem



(a) No Error Detection (b) Synchronized Error Detection

1. compicated control–flow

Basic-block Fragmentation Problem

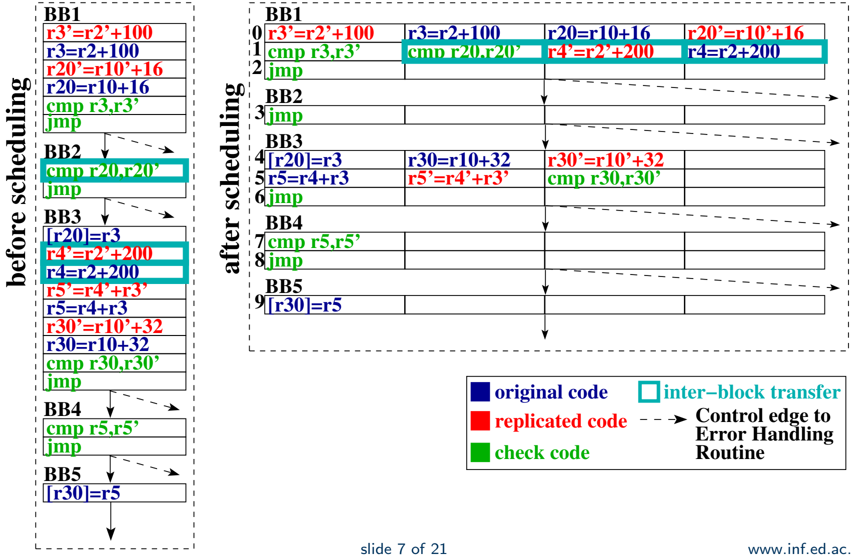


(a) No Error Detection (b) Synchronized Error Detection

1. compicated control–flow
2. the scheduler fails to produce high–performing code

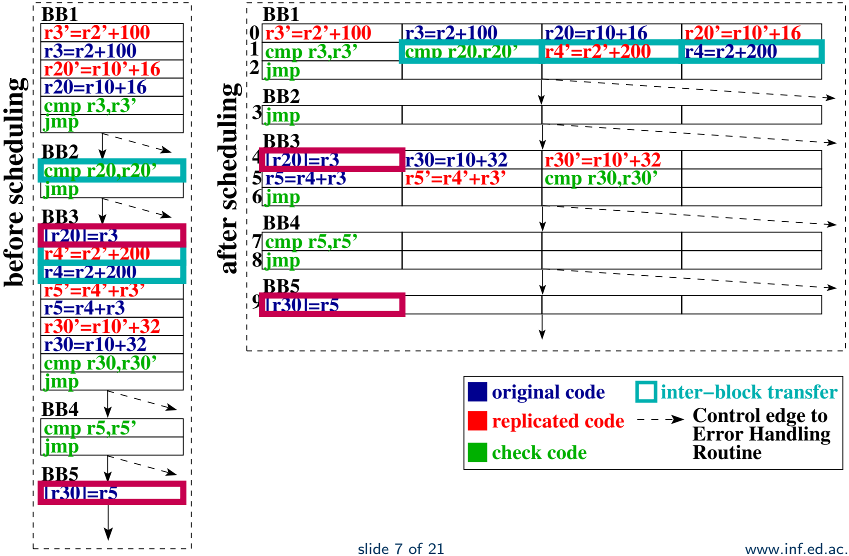
Basic-block Fragmentation Example

Synchronized Error Detetction

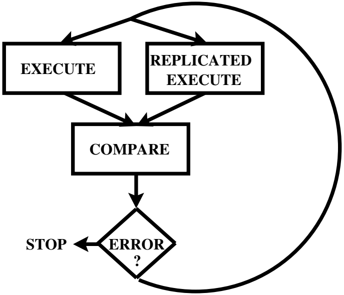


Basic-block Fragmentation Example

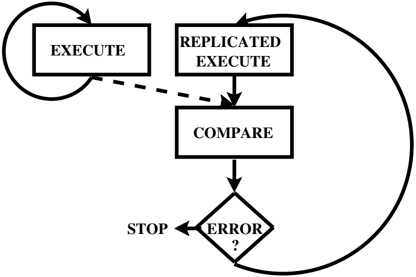
Synchronized Error Detetction



Synchronized VS Decoupled Error Detection

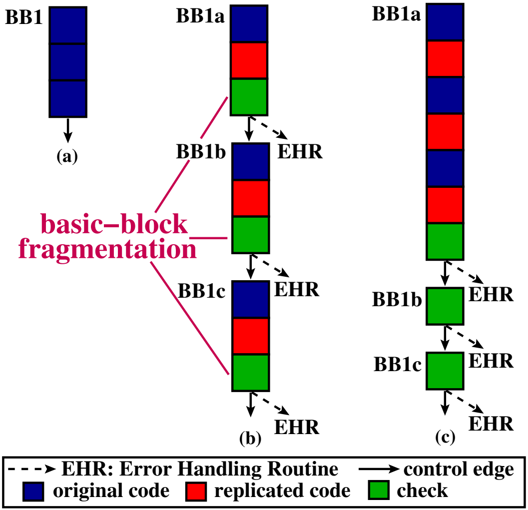


Synchronized Error Detection



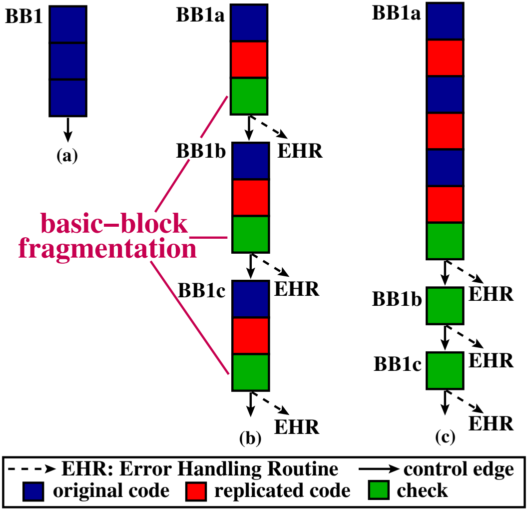
Decoupled Error Detection

DRIFT



(a) No Error Detection (b) Synchronized Error Detection
(c) Decoupled Error Detection

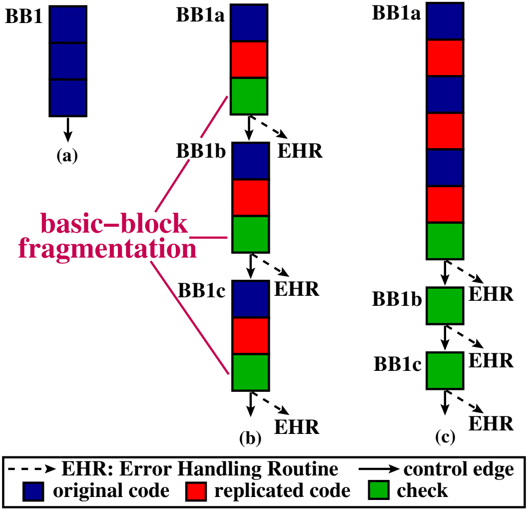
DRIFT



(a) No Error Detection (b) Synchronized Error Detection
(c) Decoupled Error Detection

1. not constrained by the limitations of code motion

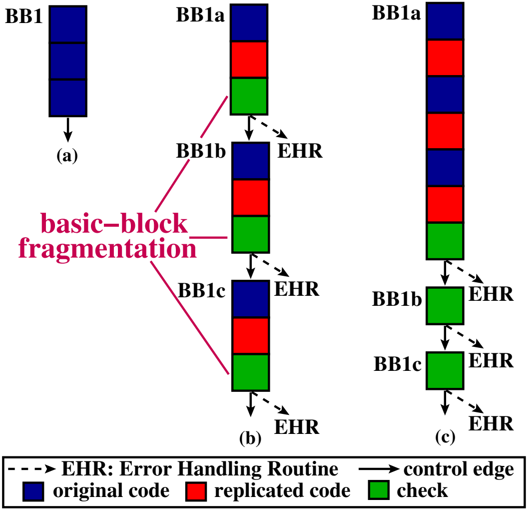
DRIFT



(a) No Error Detection (b) Synchronized Error Detection
(c) Decoupled Error Detection

1. not constrained by the limitations of code motion
2. it is OK to break the program semantics of the error detection code

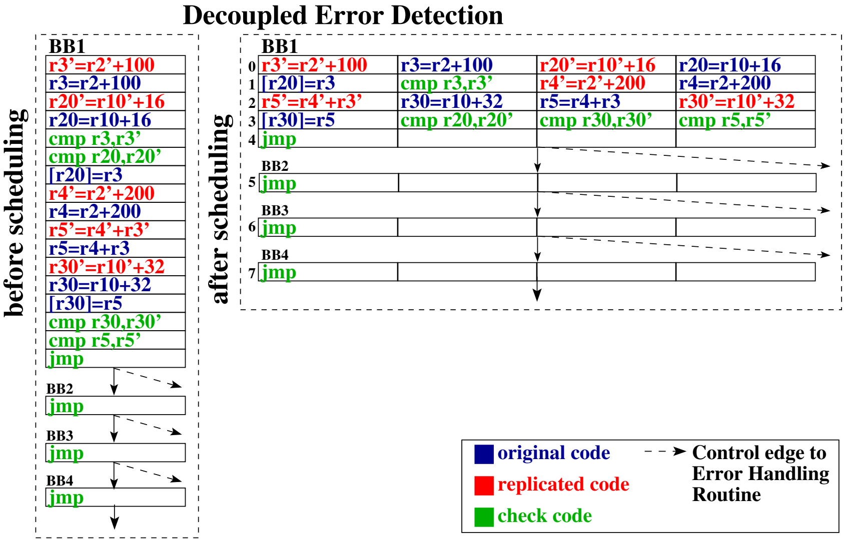
DRIFT



(a) No Error Detection (b) Synchronized Error Detection
(c) Decoupled Error Detection

1. not constrained by the limitations of code motion
2. it is OK to break the program semantics of the error detection code
3. no effect on fault–coverage

DRIFT Example



Decouple Factor

Decouple factor is a metric that describes the number of checks that are clustered together.

Increasing the decouple factor has three side-effects:

- decrease the impact of basic-block fragmentation
- increase the probability to reduce the fault-coverage
- increase the probability to create congestion on the hardware (e.g. predicate register pressure, fully occupied functional units)

DRIFT Algorithm

code replication

r2 = r1 + 1
r2 = r1 + 1

DRIFT Algorithm

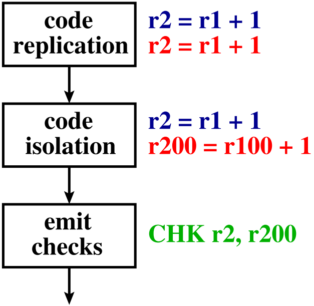
code replication

r2 = r1 + 1
r2 = r1 + 1

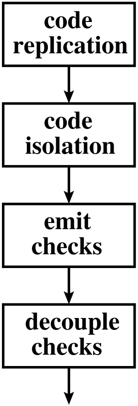
code isolation

r2 = r1 + 1
r200 = r100 + 1

DRIFT Algorithm



DRIFT Algorithm



r2 = r1 + 1
r2 = r1 + 1

r2 = r1 + 1
r200 = r100 + 1

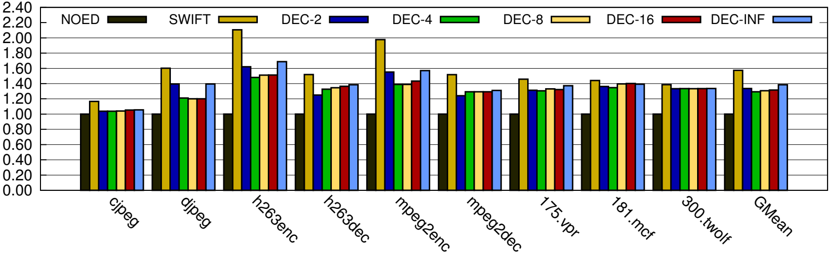
CHK r2, r200

clustering the checks considering the decouple factor

Experimental Setup

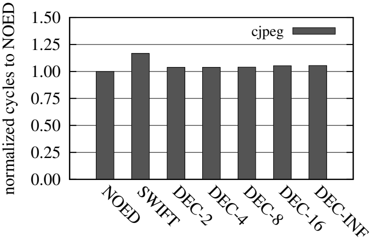
- Compiler
  - GCC-4.5.0
- Performance Evaluation
  - DELL PowerEdge 3250 server with 2x1.4GHz Intel Itanium 2 processors
- Fault-coverage Evaluation
  - SKI simulator
- Benchmarks
  - MediabenchII
  - SPEC CINT2000
- Compare
  - NOED: No Error Detection
  - SWIFT: Synchronized technique
  - DEC-x: different values of decouple factor

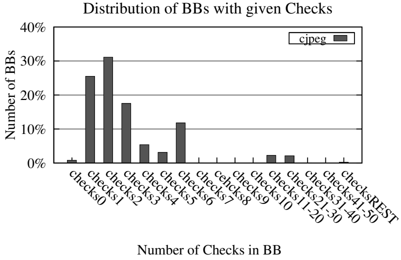
Performance Evaluation



Performance Evaluation

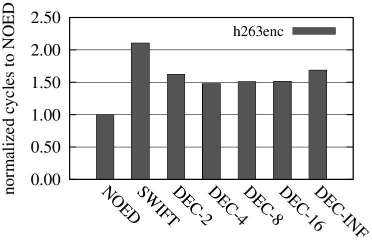
Example of benchmark with few checks per basic-block:

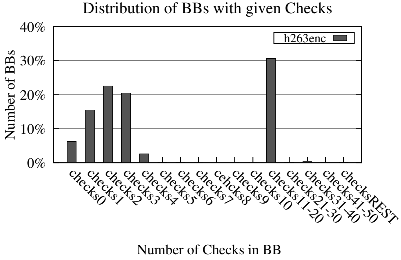




Performance Evaluation

Example of benchmark with many checks per basic-block:

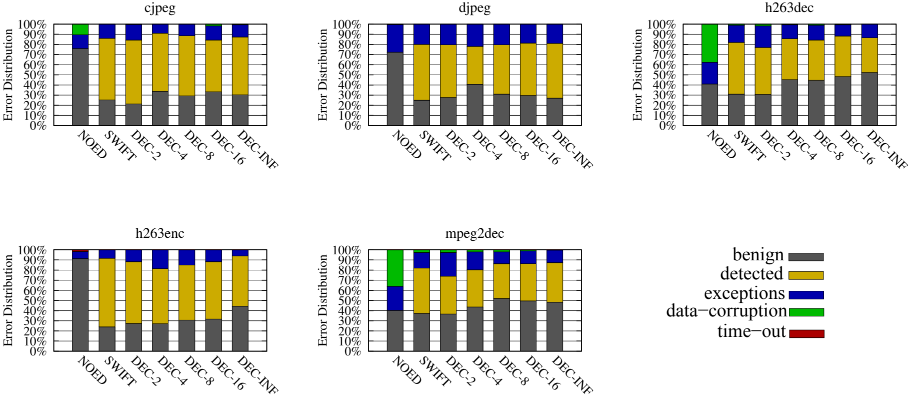




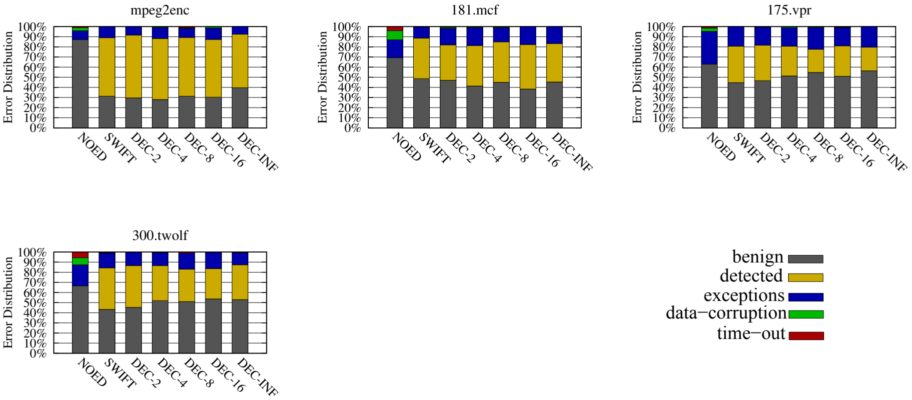
Fault Coverage Evaluation

- Single-Event Upset (SEU) fault model
- Monte Carlo simulations:
  1. count dynamic instructions
  2. randomly pick one instruction
  3. randomly flip one bit of the instruction's output
  4. execute the program
  5. repeat steps 2-4 for 300 times for each implementation of each benchmark
- Errors taxonomy:
  - *benign errors*: result in correct output
  - *detected errors*: are the errors that a technique detects
  - *exceptions*: are the errors that raise exceptions
  - *data corrupt errors*: change program's output
  - *time-out errors*: result in infinite execution of the program.

Fault Coverage Evaluation (1)



Fault Coverage Evaluation (2)



Conclusions

- Basic-block fragmentation restricts scheduler from performing aggressive code motion optimisations.
- DRIFT breaks the execute-check-confirm-execute cycle and produces scheduler-friendly code.
- DRIFT outperforms state-of-the-art up to 29.7% and reduces the error detection overhead to x1.29 (on average).
- DRIFT's performance gains have no impact on fault-coverage.

DRIFT: Decoupled compileR-based Instruction-level Fault-Tolerance

Konstantina Mitropoulou [†], Vasileios Porpodas [†]
and Marcelo Cintra[†⋆]

School of Informatics, University of Edinburgh [†]
Intel Labs Braunschweig[⋆]

LCPC 2013