CASTED: Core-Adaptive Software Transient Error Detection for Tightly-Coupled Cores

Konstantina Mitropoulou, Vasileios Porpodas, Marcelo Cintra

University of Edinburgh

IPDPS 2013

Outline

- Transient Errors
- Software-based Error Detection
- Challenges/Existing approaches
- CASTED
- Performance & Fault Coverage Evaluation
- Conclusions

Transient Errors

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

Main sources of hardware errors :

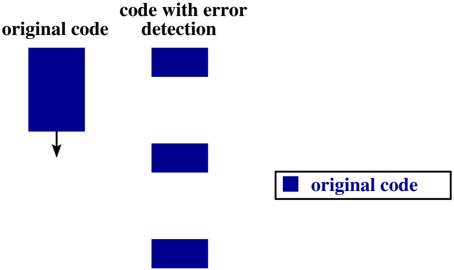
- small transistor technologies
- voltage scaling

Transient errors are:

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

Compiler-based Error Detection (1)

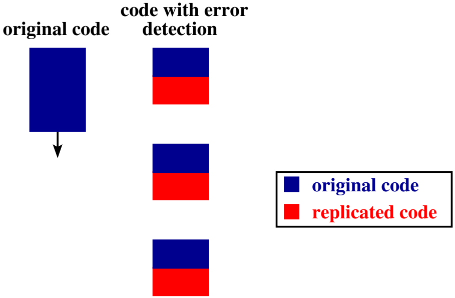
Dual-modular error detection:



Compiler-based Error Detection (1)

Dual-modular error detection:

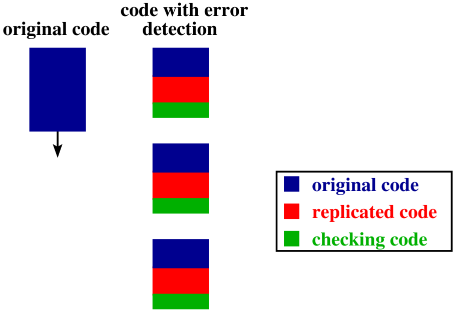
- *replicates* the computation



Compiler-based Error Detection (1)

Dual-modular error detection:

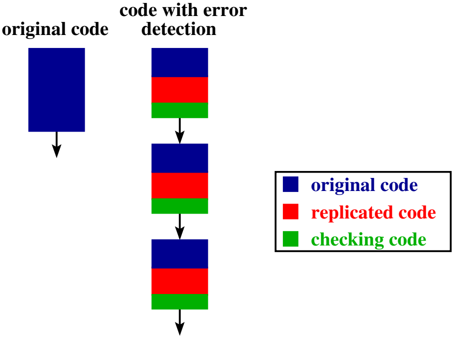
- *replicates* the computation
- *compares* the two outputs



Compiler-based Error Detection (1)

Dual-modular error detection:

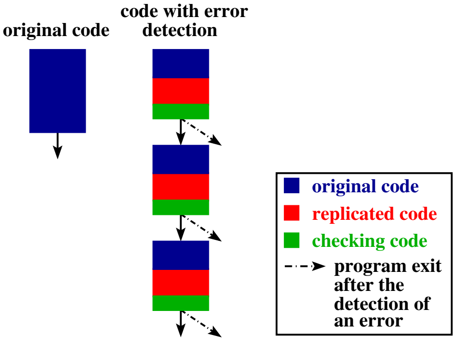
- *replicates* the computation
- *compares* the two outputs
- if the outputs are identical, then the execution *continues* normally



Compiler-based Error Detection (1)

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


original code
code with error detection
original code
replicated code
checking code
program exit after the detection of an error

Compiler-based Error Detection (2)

Challenge:

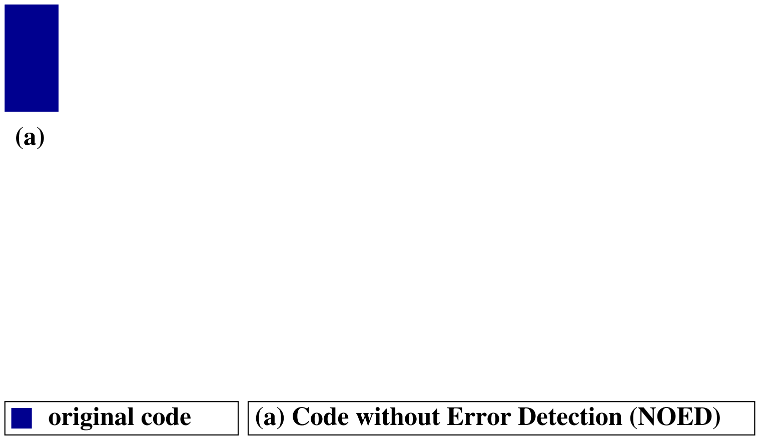
- decrease performance overhead without sacrificing system's reliability

Solutions/Existing approaches:

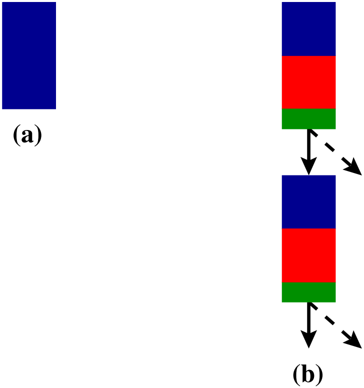
- optimize the code:
  - minimize checking points
  - reduce replicated code
- use more resources:
  - execute original and redundant code on separate cores

Overview of Existing Techniques

Overview of Existing Techniques



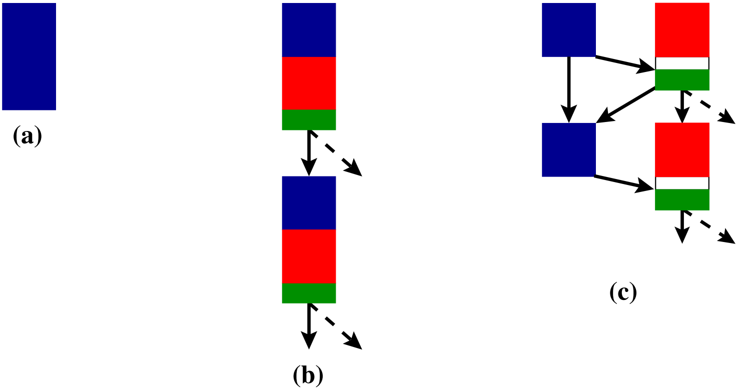
Overview of Existing Techniques



🟦 original code
🟥 replicated code
🟩 checks
- -➤ program exit

(a) Code without Error Detection (NOED)
(b) Single–Core Error Detection (SCED)

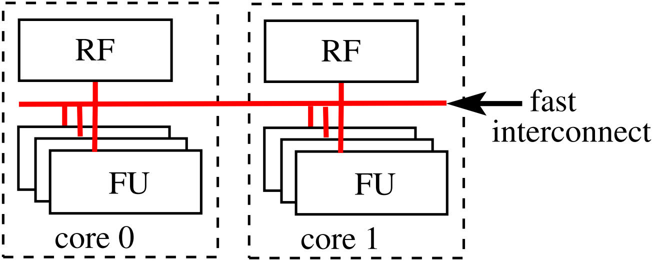
Overview of Existing Techniques



🟦 original code
🟥 replicated code
🟩 checks
- -➤ program exit

(a) Code without Error Detection (NOED)
(b) Single–Core Error Detection (SCED)
(c) Dual–Core Error Detection (DCED)

Limitations of Existing Techniques



Performance degradation factors:

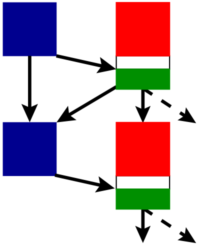
- communication latency
- sub-optimal placement of the code
- lack of adaptivity

CASTED

CASTED solves this problem by introducing adaptation.
CASTED schedules the instructions taking into consideration:

- available resources
- communication latency

CASTED



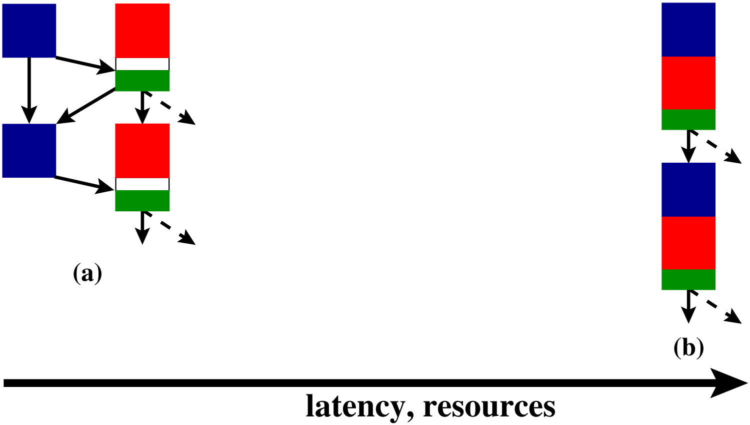
(a)

latency, resources

🟦 original code
🟥 replicated code
🟩 checks
- -➤ program exit

(a) Dual–Core Error Detection (DCED)

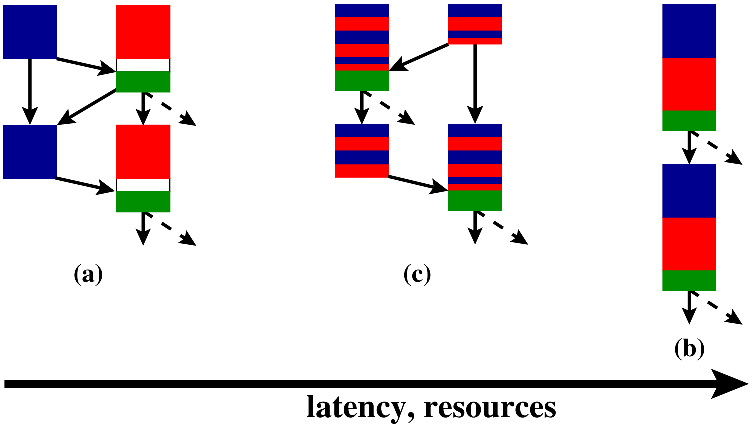
CASTED



- 🟦 original code
- 🟥 replicated code
- 🟩 checks
- - -➤ program exit

(a) Dual–Core Error Detection (DCED)
(b) Single–Core Error Detection (SCED)

CASTED



🟦 original code

🟥 replicated code

🟩 checks

- -➤ program exit

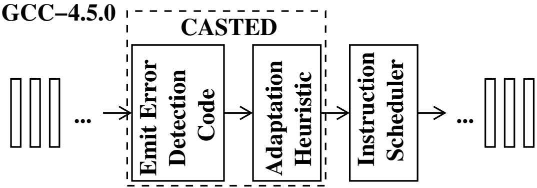
(a) Dual–Core Error Detection (DCED)

(b) Single–Core Error Detection (SCED)

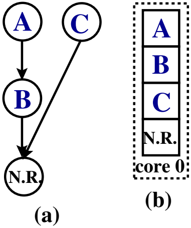
(c) CASTED

CASTED Algorithm

- Emit error detection code:
  - *replicate* all necessary instructions
  - *isolate* original code from redundant code using register renaming
  - *insert* checks
- Adaptation heuristic:
  - is a greedy heuristic that maps the code to the current architecture configuration. It schedules the instructions considering the available resources, the communication latency and the data-flow.



Example1 - Resource Constrained



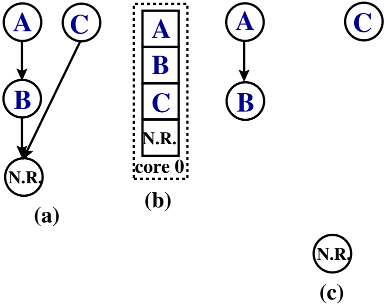
■ original instruction

N.R. non–replicated instruction

(a) Original Data Flow

(b) Original Code without Eror Detection (NOED)

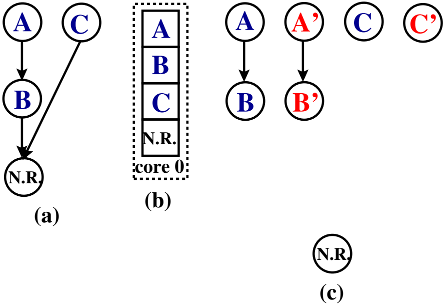
Example1 - Resource Constrained



🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code

Example1 - Resource Constrained



🟦 original instruction

🟥 replicated instruction

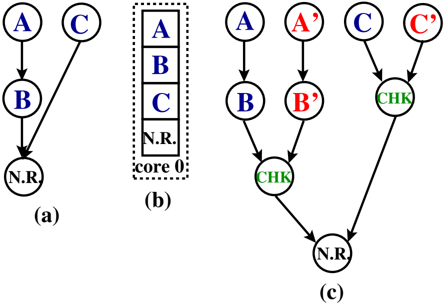
🟩 check instruction

N.R. non−replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code

Example1 - Resource Constrained

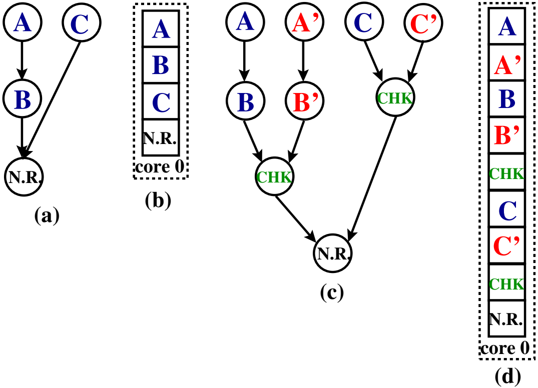
Larger DFG, more ILP & longer critical path.



🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code

Example1 - Resource Constrained



(a)

(b)

(c)

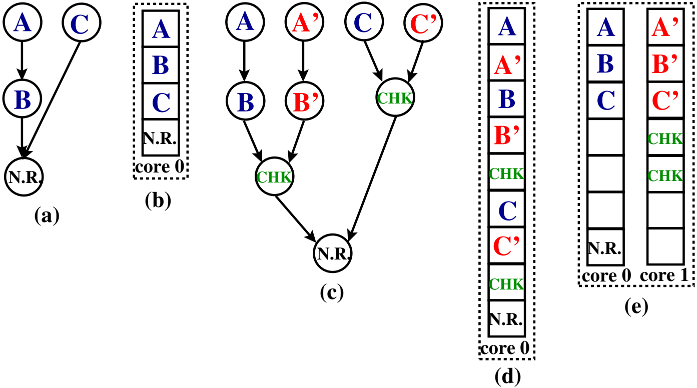
(d)

🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)

Example1 - Resource Constrained

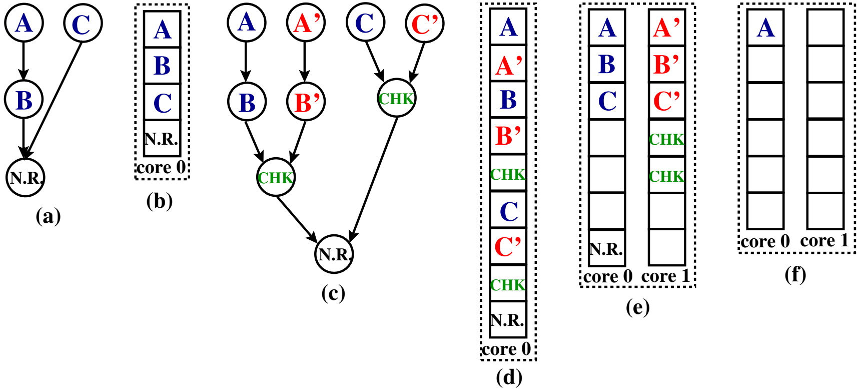
Dual–core outperforms the resource constrained single–core.



🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)
(e) Dual–Core Error Detection (DCED)

Example1 - Resource Constrained



🟦 original instruction

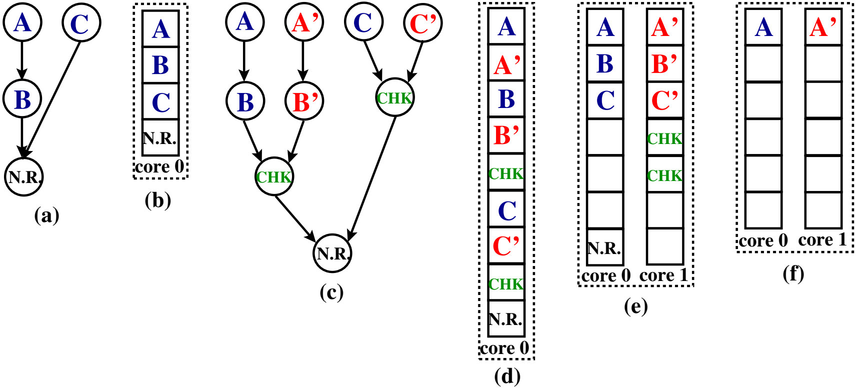
🟥 replicated instruction

🟩 check instruction

N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)
(e) Dual–Core Error Detection (DCED)
(f) CASTED

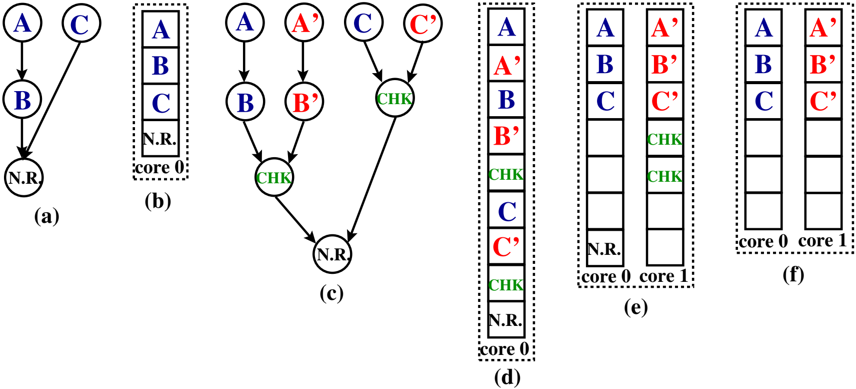
Example1 - Resource Constrained



🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)
(e) Dual–Core Error Detection (DCED)
(f) CASTED

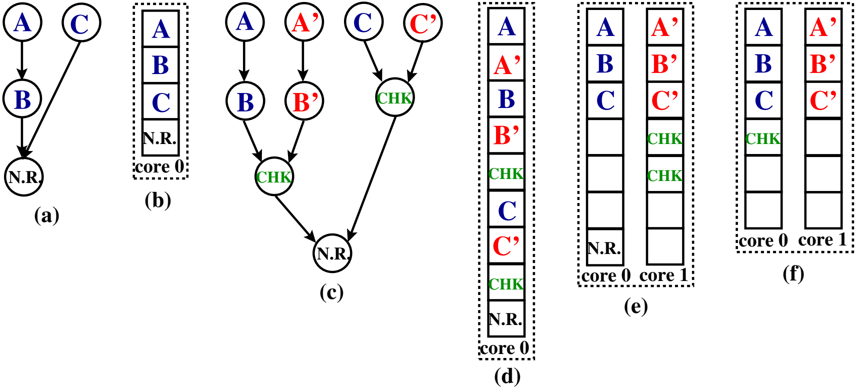
Example1 - Resource Constrained



🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)
(e) Dual–Core Error Detection (DCED)
(f) CASTED

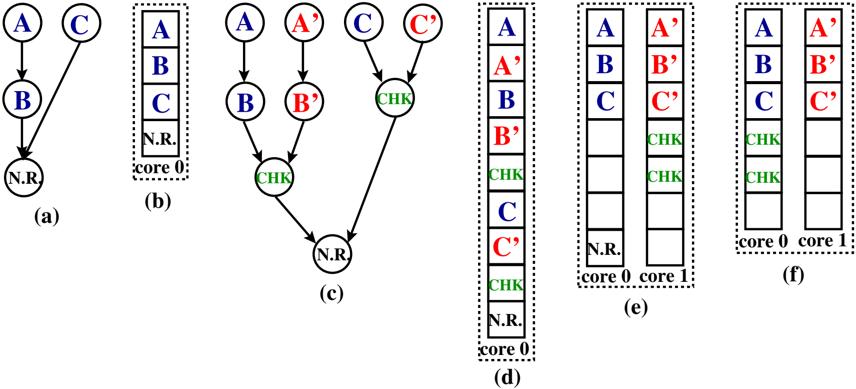
Example1 - Resource Constrained



🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non−replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single−Core Error Detection (SCED)
(e) Dual−Core Error Detection (DCED)
(f) CASTED

Example1 - Resource Constrained



🟦 original instruction

🟥 replicated instruction

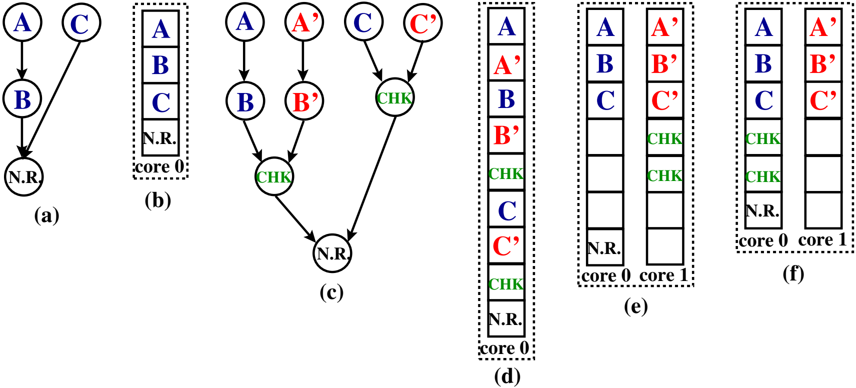
🟩 check instruction

N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)
(e) Dual–Core Error Detection (DCED)
(f) CASTED

Example1 - Resource Constrained

CASTED hides the communication penalty.



🟦 original instruction

🟥 replicated instruction

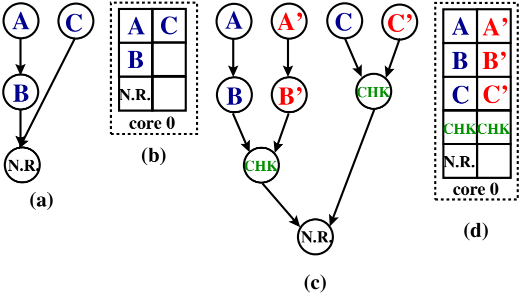
🟩 check instruction

N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)
(e) Dual–Core Error Detection (DCED)
(f) CASTED

Example2 - Latency Constrained

Single–core technique benefits from the extra resources.



(a)

A	C
B	
N.R.	

core 0

(b)

(c)

A	A'
B	B'
C	C'
CHK	CHK
N.R.	

core 0

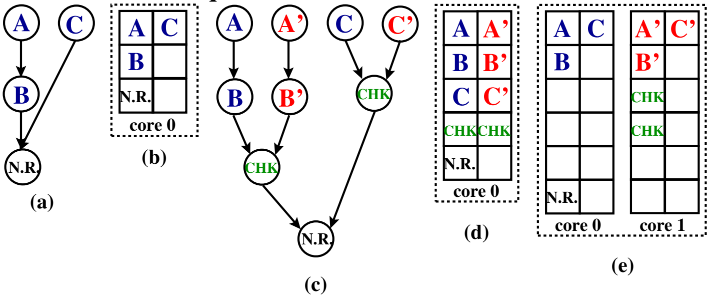
(d)

🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)

Example2 - Latency Constrained

Dual–core technique benefits less from the extra resources.

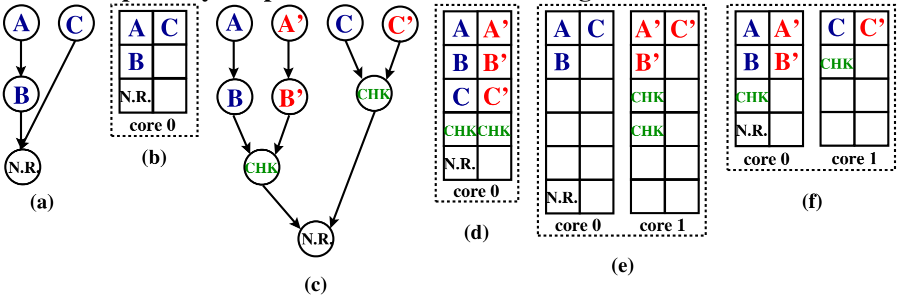


🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)
(e) Dual–Core Error Detection (DCED)

Example2 - Latency Constrained

CASTED perfecty adapts to the architecture configurations.



(a)

(b)

core 0	
A	C
B	
N.R.	

(c)

(d)

core 0	
A	A'
B	B'
C	C'
CHK	CHK
N.R.	

(e)

core 0		core 1	
A	C	A'	C'
B		B'	
		CHK	
		CHK	
N.R.			

(f)

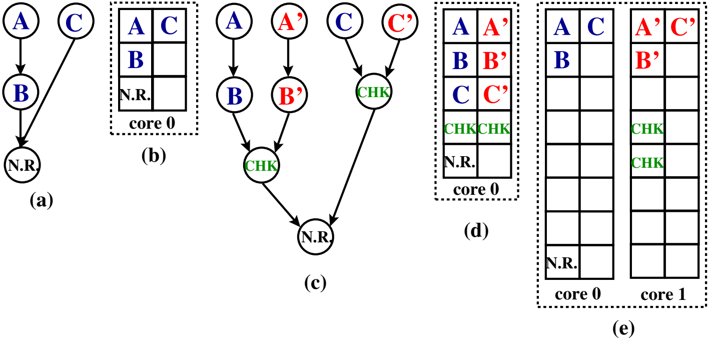
core 0		core 1	
A	A'	C	C'
B	B'	CHK	
CHK			
N.R.			

🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)
(e) Dual–Core Error Detection (DCED)
(f) CASTED

Example2 - Latency Constrained

Dual–core technique suffers from the communication latency.



(a)

A	C
B	
N.R.	

core 0

(b)

(c)

A	A'
B	B'
C	C'
CHK	CHK
N.R.	

core 0

(d)

core 0		core 1	
A	C	A'	C'
B		B'	
		CHK	
		CHK	
N.R.			

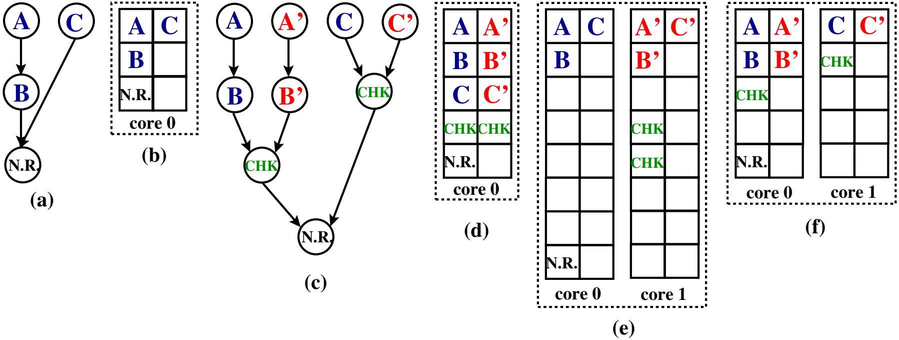
(e)

🟦 original instruction
🟥 replicated instruction
🟩 check instruction
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)
(e) Dual–Core Error Detection (DCED)

Example2 - Latency Constrained

CASTED perfecty adapts to the architecture configurations.



🟦 original instruction

🟥 replicated instruction

🟩 check instruction

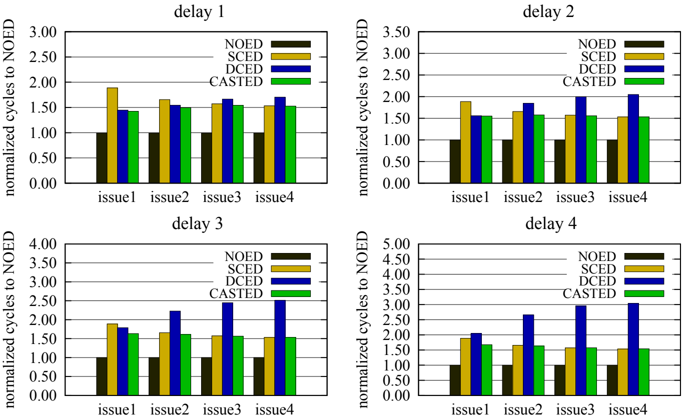
N.R. non–replicated instruction

(a) Original Data Flow
(b) Original Code without Eror Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single–Core Error Detection (SCED)
(e) Dual–Core Error Detection (DCED)
(f) CASTED

Experimental Set-up

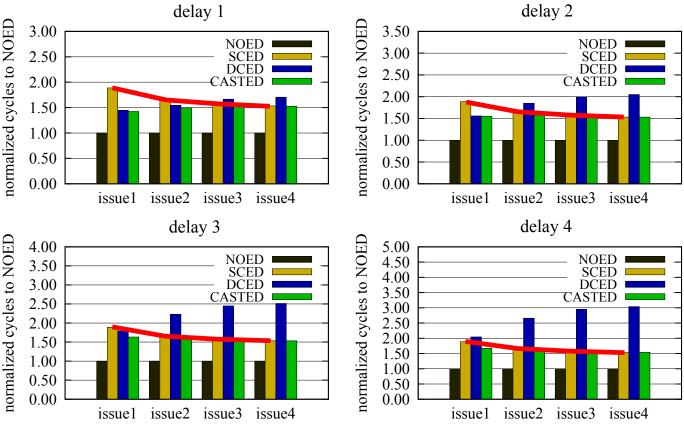
- Compiler
  - error detection code and adaptation passes added in the back-end of GCC-4.5.0
  - the *last* stage of CSE and DCE are turned-off
- Architecture
  - 2 IA64-based clusters whose issue-width takes values in the range of 1 to 4 and the communication latency varies from 1 to 4 cycles
  - SKI simulator
- Benchmarks
  - MediabenchII Video Benchmark suite
  - SPEC CINT2000
- Compare
  - NOED: No Error Detection (original code)
  - SCED: Single Core Error Detection
  - DCED: Dual Core Error Detection
  - CASTED: proposed technique

Performance Evaluation



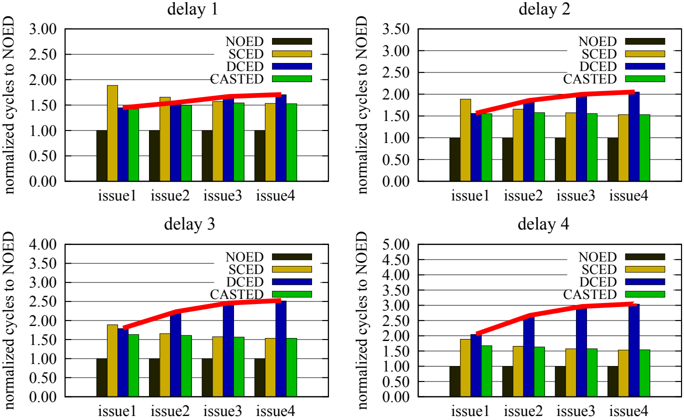
Performance Evaluation

SCED improves as the resources increase.



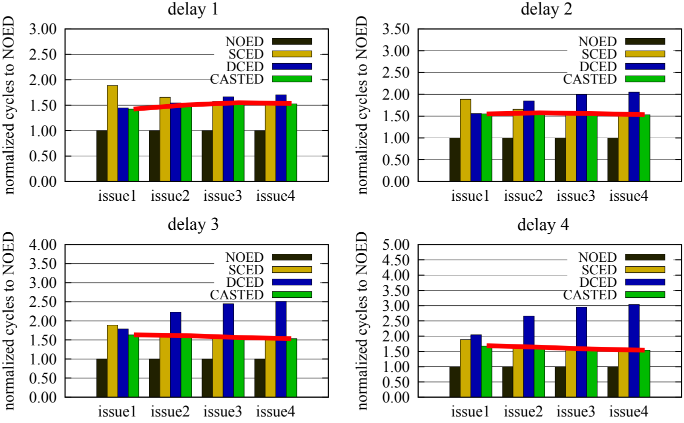
Performance Evaluation

DCED suffers communication latency and benefits less from the increase of issue-width.



Performance Evaluation

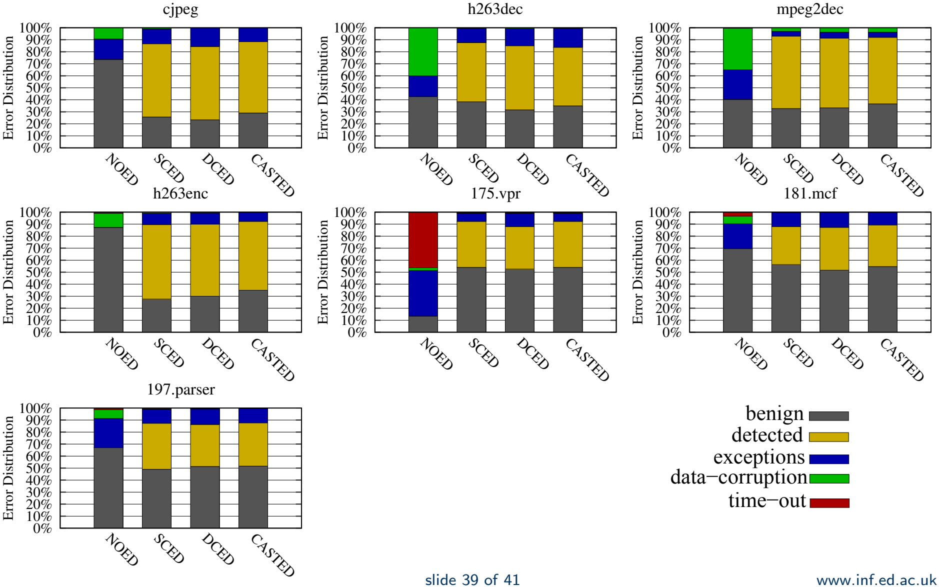
CASTED performs closely to the best technique for every configuration.



Fault Coverage Evaluation (1)

- 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 (2)



Conclusions

- The overhead of the state-of-the-art techniques varies with the architecture configurations. More resources do not guarantee better performance.
- CASTED has a fixed overhead by optimally distributing the error detection overhead to the available resources.
- Performance tracks the best policy and sometimes outperforms it.
- No degradation in fault coverage.

CASTED: Core-Adaptive Software Transient Error Detection for Tightly-Coupled Cores

Konstantina Mitropoulou, Vasileios Porpodas, Marcelo Cintra

University of Edinburgh

IPDPS 2013