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:

- Original code
- Code with 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
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

(a) Code without Error Detection (NOED)

original code
Overview of Existing Techniques

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

- original code
- replicated code
- checks
- program exit
Overview of Existing Techniques

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

- original code
- replicated code
- checks
- → program exit
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
(a) Dual-Core Error Detection (DCED)
CASTED

Latency, resources

- Original code
- Replicated code
- Checks
- Program exit

(a) Dual-Core Error Detection (DCED)
(b) Single-Core Error Detection (SCED)
CASTED

(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.
Example 1 - Resource Constrained

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

original instruction
non-replicated instruction
Example 1 - Resource Constrained

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

- original instruction
- replicated instruction
- check instruction
- non-replicated instruction
Example 1 - Resource Constrained

(a) Original Data Flow
(b) Original Code without Error Detection (NOED)
(c) Data Flow with Error Detection Code
Example 1 - Resource Constrained

Larger DFG, more ILP & longer critical path.

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

- original instruction
- replicated instruction
- check instruction
- non-replicated instruction

(a) 
(b) 
(c)
Example1 - Resource Constrained

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

- Original instruction
- Replicated instruction
- Check instruction
- Non-replicated instruction
Example 1 - Resource Constrained

Dual-core outperforms the resource constrained single-core.

- (a) Original Data Flow
- (b) Original Code without Error Detection (NOED)
- (c) Data Flow with Error Detection Code
- (d) Single-Core Error Detection (SCED)
- (e) Dual-Core Error Detection (DCED)
Example 1 - Resource Constrained

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

- Original instruction
- Replicated instruction
- Check instruction
- Non-replicated instruction

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

- Original instruction
- Replicated instruction
- Check instruction
- Non-replicated instruction
Example 1 - Resource Constrained

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

- Original instruction
- Replicated instruction
- Check instruction
- Non-replicated instruction

N.R. denotes non-replicated instruction.
Example 1 - Resource Constrained

(a) Original Data Flow
(b) Original Code without Error Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single-Core Error Detection (SCED)
(e) Dual-Core Error Detection (DCED)
(f) CASTED
Example 1 - Resource Constrained

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

original instruction
replicated instruction
check instruction
non-replicated instruction
Example1 - Resource Constrained

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

- original instruction
- replicated instruction
- check instruction
- non-replicated instruction

Slide 26 of 41
Example 1 - Resource Constrained

CASTED hides the communication penalty.

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

- original instruction
- replicated instruction
- check instruction
- non-replicated instruction
Example 2 - Latency Constrained

Single-core technique benefits from the extra resources.

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

- original instruction
- replicated instruction
- check instruction
- non-replicated instruction
Example 2 - Latency Constrained

Dual-core technique benefits less from the extra resources.

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

Legend:
- Original instruction
- Replicated instruction
- Check instruction
- Non-replicated instruction
- N.R.
Example 2 - Latency Constrained

CASTED perfectly adapts to the architecture configurations.

(a) Original Data Flow
(b) Original Code without Error Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single-Core Error Detection (SCED)
(e) Dual-Core Error Detection (DCED)
(f) CASTED
Example 2 - Latency Constrained

Dual-core technique suffers from the communication latency.

(a) Original Data Flow
(b) Original Code without Error Detection (NOED)
(c) Data Flow with Error Detection Code
(d) Single-Core Error Detection (SCED)
(e) Dual-Core Error Detection (DCED)
CASTED perfectly adapts to the architecture configurations.

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

<table>
<thead>
<tr>
<th>original instruction</th>
<th>replicated instruction</th>
<th>check instruction</th>
<th>non-replicated instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>CHK</td>
<td>N.R.</td>
</tr>
<tr>
<td>A'</td>
<td>B'</td>
<td>CHK</td>
<td>N.R.</td>
</tr>
<tr>
<td>A'</td>
<td>C'</td>
<td>CHK</td>
<td>N.R.</td>
</tr>
<tr>
<td>A</td>
<td>C</td>
<td>CHK</td>
<td>N.R.</td>
</tr>
<tr>
<td>A</td>
<td>C'</td>
<td>CHK</td>
<td>N.R.</td>
</tr>
<tr>
<td>A'</td>
<td>B</td>
<td>CHK</td>
<td>N.R.</td>
</tr>
<tr>
<td>A</td>
<td>B'</td>
<td>CHK</td>
<td>N.R.</td>
</tr>
<tr>
<td>A'</td>
<td>C'</td>
<td>CHK</td>
<td>N.R.</td>
</tr>
</tbody>
</table>

NOTES:
- (a) Original Data Flow
- (b) Original Code without Error 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

- **delay 1**
  - NOED
  - SCED
  - DCED
  - CASTED

- **delay 2**
  - NOED
  - SCED
  - DCED
  - CASTED

- **delay 3**
  - NOED
  - SCED
  - DCED
  - CASTED

- **delay 4**
  - NOED
  - SCED
  - DCED
  - CASTED
Performance Evaluation

SCED improves as the resources increase.

- Delay 1
- Delay 2
- Delay 3
- Delay 4
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)

Error Distribution

- jpeg
- h263dec
- mpeg2dec
- h263enc
- 175.vpr
- 181.mcf
- parser

Error Distribution:
- NOED
- SCED
- DCED
- CASTED

Categories:
- benign
- detected
- exceptions
- data-corruption
- time-out
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