Task-level Redundancy vs Instruction-level Redundancy against Single Event Upsets in Real-time DAG scheduling

Lukas Miedema  
l.miedema@uva.nl

Benjamin Rouxel  
b.rouxel@uva.nl

Clemens Grelck  
c.grelck@uva.nl

University of Amsterdam, Netherlands
Embedded systems have become ubiquitous
  - E.g. internet of things
  - *Commercial Off the Shelf* (COTS) hardware
Motivation

- Embedded systems have become ubiquitous
  - E.g. internet of things
  - *Commercial Off the Shelf (COTS)* hardware
- Reliability is a concern for critical applications
  - Software-based fault-tolerance
  - Real-time constraints: no timing opacity
Embedded systems have become ubiquitous
  - E.g. internet of things
  - Commercial Off the Shelf (COTS) hardware
Reliability is a concern for critical applications
  - Software-based fault-tolerance
  - Real-time constraints: no timing opacity

What is the best granularity level?
Motivation

- Embedded systems have become ubiquitous
  - E.g. internet of things
  - Commercial Off the Shelf (COTS) hardware
- Reliability is a concern for critical applications
  - Software-based fault-tolerance
  - Real-time constraints: no timing opacity
- **What is the best granularity level?**
  - Instruction-level $\rightarrow$ instant detection
  - Task-level $\rightarrow$ parallelization
Targeting *Single Event Upsets* (SEUs)
- Transient event in the hardware
- Causes incorrect results/deadline overrun
Fault Model

- Targeting *Single Event Upsets* (SEUs)
  - Transient event in the hardware
  - Causes incorrect results/deadline overrun
- Caused by cosmic rays
  - Event distribution: Poisson
  - Each event is independent
Directed Acyclic Graph (DAG) scheduling
- Tasks have dependencies
- With one global deadline and period
- Up to the system to order tasks
Anatomy of software-based fault tolerance

- Run some unit of code multiple times
  - $2 \times$ detection of faults
  - $3 \times$ mitigation one fault
  - $> 3 \times$ mitigation of multiple faults

Cost matters: considerable overhead
Anatomy of software-based fault tolerance

- Run some unit of code multiple times
  - $2 \times$ detection of faults
  - $3 \times$ mitigation one fault
  - $> 3 \times$ mitigation of multiple faults
- Cost matters: considerable overhead
Single Event Upsets are rare

- Poisson distribution: $P(\text{SEU} \in \tau_i) = 1 - e^{C_i \cdot \lambda}$
- "High" rate (e.g. in space)
  $\lambda = 10^{-3}/\text{hour}/\text{processor}$
Traditional fault mitigation

- Number of replicas needed is annoyingly unbound
- Need for bound execution time to guarantee scheduling
  - Limit number of replicas to $n$
  - Increase $n$ until satisfied
- $R_{\text{fault-free}}$ is the WCRT (even under faults)
Traditional fault mitigation

- Number of replicas needed is annoyingly unbound
- Need for bound execution time to guarantee scheduling
  - Limit number of replicas to $n$
  - Increase $n$ until satisfied
- $R_{\text{fault-free}}$ is the WCRT (even under faults)
Alternative: mitigation using restarting

- Replication provides detection-only
- Mitigation by restarting
- Efficient use of hardware for DAG scheduling, but opaque impact on timing
  - Timing impact has no upper bound
Leveraging TeamPlay coordination language

- Existing coordination language for specification of real-time systems\(^1\)
- Communication between tasks is handled by middleware

---

Mitigation: checkpoint-restart at the task level
Detection: two approaches;
Mitigation: checkpoint-restart at the task level
Detection: two approaches;

Task-level redundancy
- Run every task twice
- Allows replication in space (parallelism)
- But: tasks have to run to their end to detect faults
  - Can be delayed considerably for (extremely) long-running tasks
Mitigation: checkpoint-restart at the task level

Detection: two approaches;

1. Task-level redundancy
   - Run every task twice
   - Allows replication in space (parallelism)
   - But: tasks have to run to their end to detect faults
     - Can be delayed considerably for (extremely) long-running tasks

2. Instruction-level redundancy
   - Interweave two copies of the task instructions
   - Near-instant detection of SEUs
   - But: replication in time (no parallelism)
Task-level Redundancy vs Instruction-level Redundancy

- Mitigation: checkpoint-restart at the task level
- Detection: two approaches;

1. Task-level redundancy
   - Run every task twice
   - Allows replication in space (parallelism)
   - But: tasks have to run to their end to detect faults
     - Can be delayed considerably for (extremely) long-running tasks

2. Instruction-level redundancy
   - Interweave two copies of the task instructions
   - Near-instant detection of SEUs
   - But: replication in time (no parallelism)

---

Analyzing timing behavior with UPPAAL SMC

- Using UPPAAL in SMC mode\(^3\)
  - Models hybrid automata
  - *Stochastic Model Checking* mode
  - SMC deals with infinitely large state spaces by sampling
- Automatically generate UPPAAL models from TeamPlay-encoded DAGs

---

Structure of the UPPAAL models

- UPPAAL model consists of
  - Scheduler (singleton)
  - Task (one per task)
  - Edge (one per precedence relation)
  - Processors

- Processors experiencing SEUs @ Poisson distribution
- Online scheduler (G-EDF\(^4\)) for DAGs

---

UPPAAL SMC model of a task

- **start** → UnmetDependencies
- Unscheduled
- RunningWFault, \( t \leq C_i \)
- Ready
- Running, \( t \leq C_i \)
- Finished

UPPAAL Task Process model.
Benchmark: STR2RTS

- STR2RTS\(^5\): Conversion of StreamIT for real-time
- Used 15 streaming benchmarks
- Some exceptionally long-running
- Executed on a simulated 8-processor platform with 
  \[ \lambda = 10^{-3}/\text{hour/processor} \]

---

### Overview of used STR2RTS benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th># Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>802.11a</td>
<td>802.11a wireless LAN protocol transmitter</td>
<td>116</td>
</tr>
<tr>
<td>802.11a HSDF</td>
<td>802.11a wireless LAN protocol transmitter, homogenized</td>
<td>4753</td>
</tr>
<tr>
<td>Audiobeam</td>
<td>Real-time beamforming on a microphone input array</td>
<td>20</td>
</tr>
<tr>
<td>BeamFormer</td>
<td>Multi-channel beam former</td>
<td>56</td>
</tr>
<tr>
<td>BitonicSort</td>
<td>High performance bitonic sorting network</td>
<td>122</td>
</tr>
<tr>
<td>CFARtest</td>
<td>Constant False Alarm Rate (CFAR) Detection Benchmark</td>
<td>4</td>
</tr>
<tr>
<td>ComplexFIR</td>
<td>FIR filter with complex data types</td>
<td>3</td>
</tr>
<tr>
<td>DES</td>
<td>DES encryption algorithm</td>
<td>423</td>
</tr>
<tr>
<td>FFT2</td>
<td>Fast Fourier Transform kernel</td>
<td>26</td>
</tr>
<tr>
<td>FilterBankNew</td>
<td>Filter bank for multirate signal processing</td>
<td>53</td>
</tr>
<tr>
<td>FIRBank</td>
<td>Bank of FIR filters from PCA kernel apps</td>
<td>340</td>
</tr>
<tr>
<td>FIRcoarse</td>
<td>Bank of FIR filters from PCA kernel apps</td>
<td>3</td>
</tr>
<tr>
<td>FIR</td>
<td>Fine-grained finite impulse response kernel</td>
<td>132</td>
</tr>
<tr>
<td>MatrixMultBlock</td>
<td>Generates series of matrices, and multiplies them</td>
<td>23</td>
</tr>
<tr>
<td>Serpent</td>
<td>Implements the Serpent encryption algorithm</td>
<td>234</td>
</tr>
</tbody>
</table>
116 tasks
$10^6$ SMC simulations (each)
Histogram bin width of $10^3$ s

Results: 802.11a
802.11a wireless LAN protocol transmitter
Results: 802.11a
802.11a wireless LAN protocol transmitter

- 116 tasks
- $10^6$ SMC simulations (each)
- Histogram bin width of $10^3$ s

(a) Task-level redundancy, $R_{\text{fault-free}} = 92,057$ s
116 tasks
10^6 SMC simulations (each)
Histogram bin width of 10^3 s

(a) Task-level redundancy, \( R_{\text{fault-free}} = 92,057 \) s

(b) Instruction-level redundancy, \( R_{\text{fault-free}} = 184,094 \) s
Homogenized 802.11a

- Investigate impact of extremely long-running DAG applications
- Homogenize 802.11a
  - From 116 tasks to 4753, but leave the task WCET the same
  - Yields extremely high WCRT of 3391910 seconds $= 39 \frac{1}{4} \text{ days}$
Results: Homogenized 802.11a
802.11a wireless LAN protocol transmitter

- 4753 tasks
- $6 \cdot 10^3$ SMC simulations (each)
- Histogram bin width of $10^5$ s
Results: Homogenized 802.11a
802.11a wireless LAN protocol transmitter

- 4753 tasks
- $6 \cdot 10^3$ SMC simulations (each)
- Histogram bin width of $10^5$ s

(a) Task-level redundancy
4753 tasks
6 \cdot 10^3 \text{ SMC simulations (each)}
Histogram bin width of 10^5 \text{s}
Results: Homogenized 802.11a
802.11a wireless LAN protocol transmitter

- Sweep deadline across range of values
- Expressed in terms of the minimal $R_{\text{fault-free}}$ for the two types
Results: aggregated
Average across all benchmarks scaled to per-benchmark $R_{\text{fault-free}}$

- Sweep deadline across range of values
  - Expressed in terms of the minimal $R_{\text{fault-free}}$ for the two types
Conclusion & Further work

- Compared instruction-level redundancy against task-level redundancy
  - Task-level redundancy *almost* universally better
  - Identified cases where instruction-level is better
- Next: investigate more levels of redundancy