# 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

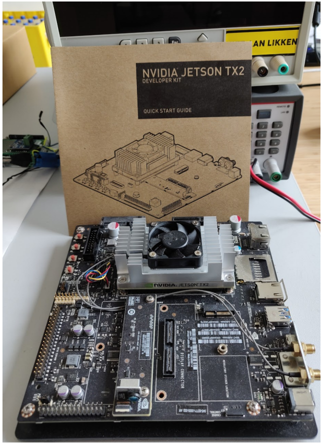
University of Amsterdam, Netherlands



ADMORPH
No. 871259

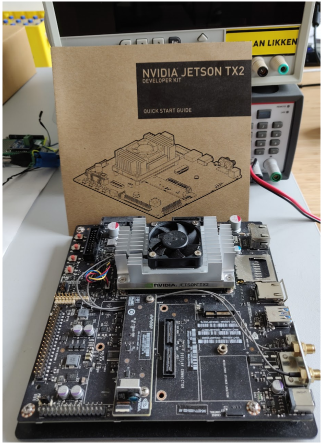
# Motivation

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



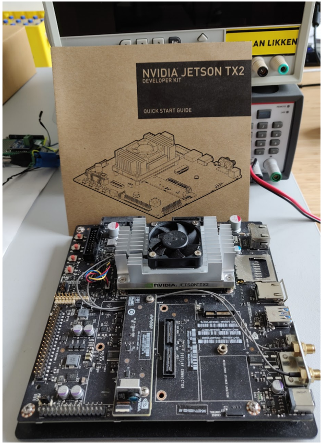
# Motivation

- 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



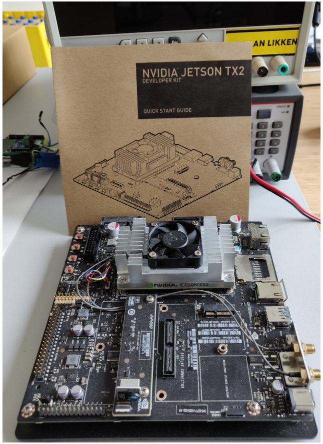
# Motivation

- 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?**



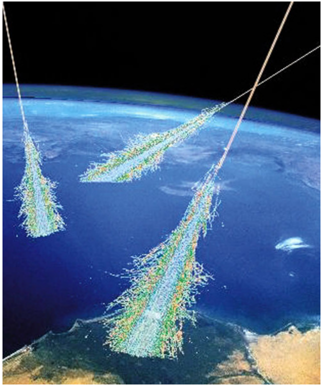
# Motivation

- 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 → instant detection
  - Task-level → parallelization



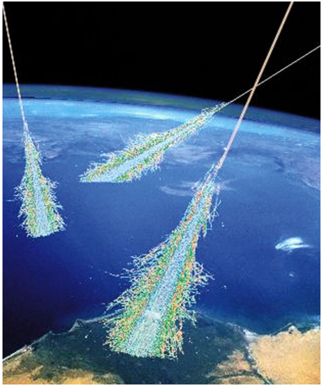
# Fault Model

- Targeting *Single Event Upsets* (SEUs)
  - Transient event in the hardware
  - Causes incorrect results/deadline overrun



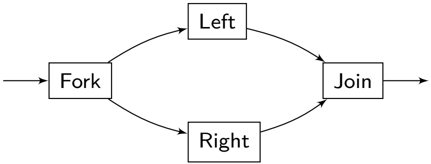
# Fault Model

- 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



# Real-time DAG scheduling

- 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 |

# Anatomy of software-based fault tolerance

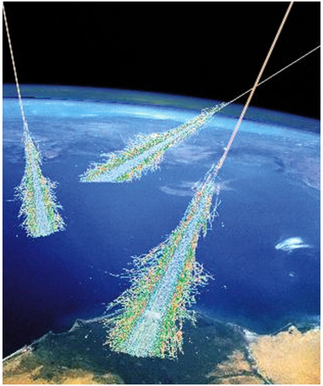
- Run some unit of code multiple times

| | |
|---|---|
| $2\times$ | 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)
  $\lambda = 10^{-3} / hour / processsor$

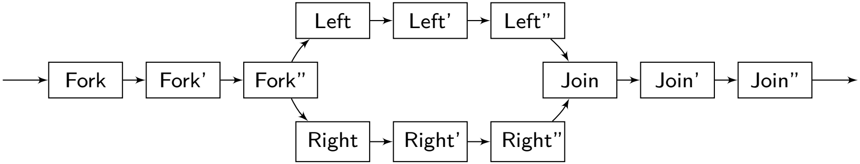


# Traditional fault mitigation

- 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

[1] Julius Roeder et al. "Towards Energy-, Time-and Security-Aware Multi-Core Coordination". In: *International Conference on Coordination Languages and Models*. Springer, 2020, pp. 57–74.

# Task-level Redundancy vs Instruction-level Redundancy

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

# 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

# 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)

# 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[2]
    - Interweave two copies of the task instructions
    - Near-instant detection of SEUs
    - But: replication in time (no parallelism)

[2] René Rydhof Hansen et al. "Formal Methods for Modelling and Analysis of Single-Event Upsets". In: *2015 IEEE International Conference on Information Reuse and Integration*. IEEE, 2015, pp. 287–294.

# Analyzing timing behavior with UPPAAL SMC

- Using UPPAAL in SMC mode[3]
  - Models hybrid automatons
  - *Stochastic Model Checking* mode
  - SMC deals with infinitely large state spaces by sampling
- Automatically generate UPPAAL models from TeamPlay-encoded DAGs

[3] P. Bulychev et al. "UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata". In: *arXiv e-prints* (2012).

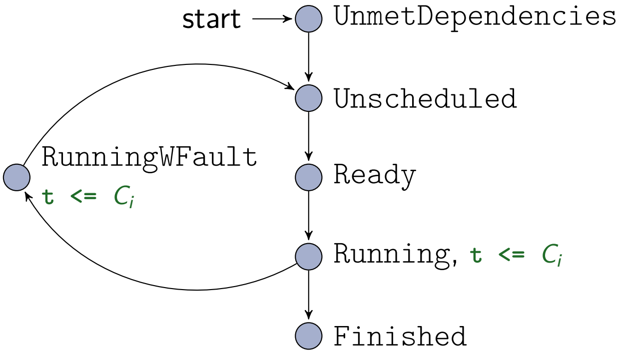
# Structure of the UPPAAL models

- 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

[4] Manar Qamhieh et al. "Global EDF Scheduling of Directed Acyclic Graphs on Multiprocessor Systems". In: *Proceedings of the 21st International Conference on Real-Time Networks and Systems*. 2013, pp. 287–296.

# UPPAAL SMC model of a task




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 $\lambda = 10^{-3} / hour / processsor$

[5] Benjamin Rouxel and Isabelle Puaut. "STR2RTS: Refactored StreamIT Benchmarks into Statically Analyzable Parallel Benchmarks for WCET Estimation & Real-Time Scheduling". In: *17th International Workshop on Worst-Case Execution Time Analysis (WCET 2017)*. Vol. 57. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 1:1–1:12.

# STR2RTS benchmarks

## Overview of used STR2RTS benchmarks

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

# Results: 802.11a

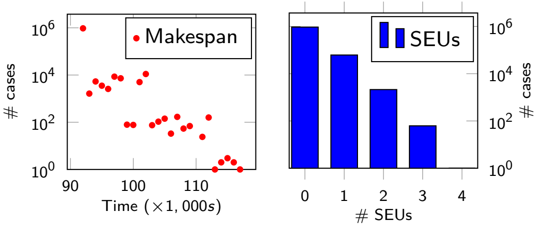
## 802.11a wireless LAN protocol transmitter

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

# Results: 802.11a

## 802.11a wireless LAN protocol transmitter

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

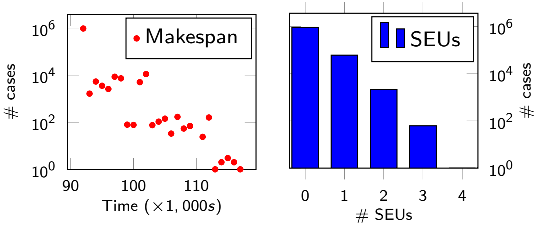



(a) Task-level redundancy, $R_{\text{fault-free}} = 92,057s$

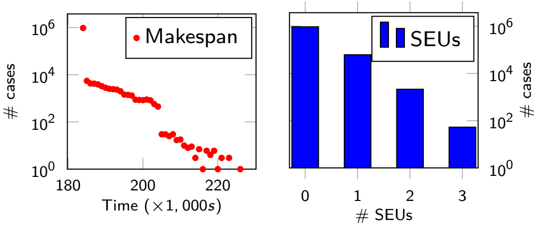
# Results: 802.11a

## 802.11a wireless LAN protocol transmitter

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




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




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

# 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 $seconds = 39\frac{1}{4}\ days$

# Results: Homogenized 802.11a

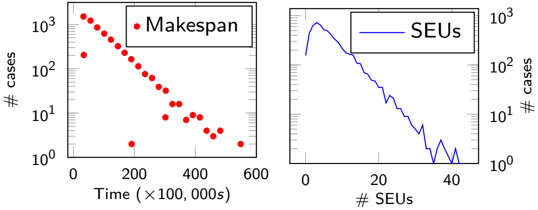
## 802.11a wireless LAN protocol transmitter

- 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 $10^5 s$

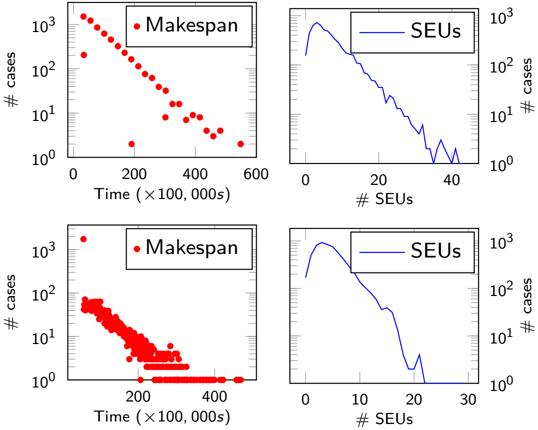



(a) Task-level redundancy

# Results: Homogenized 802.11a

## 802.11a wireless LAN protocol transmitter

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



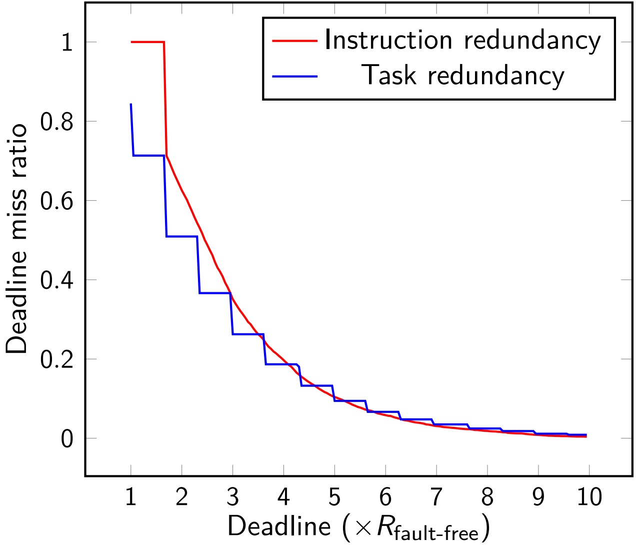

(a) Task-level redundancy

(b) Instruction-level redundancy

# Results: Homogenized 802.11a

## 802.11a wireless LAN protocol transmitter

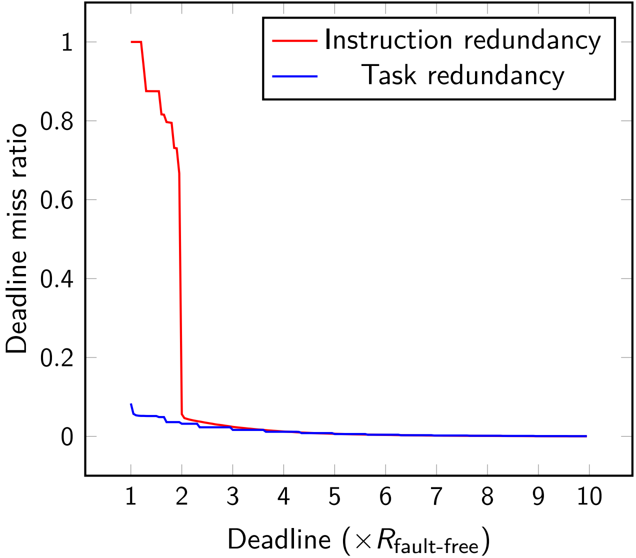
- Sweep deadline across range of values
  - Expressed in terms of the minimal $R_{\text{fault-free}}$ for the two types



# 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