Offline Equivalence:
A Non-Preemptive Scheduling Technique
for Resource-Constrained Embedded Real-Time Systems

Mitra Nasri*        Björn B. Brandenburg

MPI-SWS, Kaiserslautern, Germany
What do you do if you have a nice scheduling table that doesn’t fit into memory?

**Offline Equivalence**

allows you to store only a little “crucial” information to **rebuild** your table at **runtime** with the help of an efficient **online scheduling** algorithm.
Motivation

- Many embedded systems (still) have limited processing power and memory
- Usually no operating system
- Naturally **non-preemptive**

---

**Arm Cortex MCU family**

<table>
<thead>
<tr>
<th>Part Number</th>
<th>Package Type</th>
<th>Core</th>
<th>Frequency (MHz)</th>
<th>Flash Size (kB)</th>
<th>Internal RAM Size (kB)</th>
<th>I/Os (High Current)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32L011G4</td>
<td>UFQFN 28 4x4 x0.55</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>16</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>STM32L011K4</td>
<td>LQFP 32 7x7x1.4, ...</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>16</td>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>STM32L021D4</td>
<td>TSSOP 14</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>16</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>STM32L021F4</td>
<td>UFQFN 20 3x3 x0.3</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>16</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>STM32L021G4</td>
<td>UFQFN 28 4x4 x0.55</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>16</td>
<td>2</td>
<td>24</td>
</tr>
<tr>
<td>STM32L021K4</td>
<td>LQFP 32 7x7x1.4, ...</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>16</td>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>STM32L031F4</td>
<td>TSSOP 20</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>STM32L071C5</td>
<td>LQFP 48 7x7x1.4</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>64</td>
<td>20</td>
<td>37</td>
</tr>
<tr>
<td>STM32L071R2</td>
<td>LQFP 64 10x10x1.4</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>192</td>
<td>20</td>
<td>51</td>
</tr>
<tr>
<td>STM32L071V8</td>
<td>LQFP 100 14x1 4x1.4</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>128</td>
<td>20</td>
<td>84</td>
</tr>
</tbody>
</table>
Existing Approaches

Table-driven scheduling or cyclic executive

- Low runtime overhead
- High schedulability ratio
- Flexible: allows adding constraints during construction

Online scheduling

- No need to store a table
- Less flexibility to add complex constraints
- work-conserving (fixed-priority, EDF, etc.)
- Non-work-conserving (Precautious-RM, CW-EDF, etc.)

A power train ECU [Anssi13]:
- 6 periodic tasks with release offset
- Periods \{1, 5, 10, 10, 40, 100\}
- 500 jobs in a hyperperiod
- Offline table is at least 2 KiB

An automotive benchmark from Bosch [Kramer15]:
- Periods \{1, 2, 5, 10, 20, 50, 100, 200, 1000\}
- 1886 jobs in a hyperperiod
- Adding a functionality with 30 frames per second leads to 63,238 jobs in a hyperperiod
Existing Approaches

Table-driven scheduling or cyclic executive

- Low runtime overhead
- High schedulability ratio
- Flexible: allows adding constraints during construction

Offline Equivalence

Online scheduling

- No need to store a table
- Stores less information

- Less flexibility to add complex constraints
- work-conserving (fixed-priority, EDF, etc.)
- Non-work-conserving (Precautious-RM, CW-EDF, etc.)

Table must be stored in memory
This Paper: Offline Equivalence

- Offline table generator
- Scheduling table
- Online policy
- Online scheduling algorithm
- Scan the table and store differences
- Modify online scheduler to use differential data
- Types of irregularities:
  - Priority inversion
  - Idle interval
- Differential data (irregularities)
- Modified online scheduling algorithm

Our online Scheduler (OE)
Contributions

- Offline equivalence technique
- An efficient offline table generation algorithm (for a non-preemptive set of jobs)
Agenda

- Offline equivalence
  - Efficient table generation
  - Evaluation
  - Conclusion
Two Key Components of Offline Equivalence

- Offline table generator
  - Schedule table
  - Scan the table and store differences

- Online scheduling algorithm
  - Differential data (irregularities)
  - Our online scheduler (OE)
  - Modified online scheduling algorithm

- Modify online scheduler to use differential data
Scan Phase

- **Scan** the table to identify irregularities w.r.t. the online policy and **store** them
  - Priority inversion irregularity
  - Idle interval irregularity

### Online policy:
rate monotonic

#### Idle interval irregularity table (IIT)

<table>
<thead>
<tr>
<th>τ3 = (8, 60)</th>
<th>τ2 = (6, 12)</th>
<th>τ1 = (3, 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Time line" /></td>
<td><img src="image2" alt="Time line" /></td>
<td><img src="image3" alt="Time line" /></td>
</tr>
</tbody>
</table>

#### Priority inversion table (PIT)

- From time 9, for 1 time unit
- The 3rd Job of τ2 starts at 30

Only two entries were needed
Modifying Baseline Online Scheduler to Use Differential Data

**Priority inversion table (PIT)** (sorted by Task# and Job#)

<table>
<thead>
<tr>
<th>Task #</th>
<th>Job #</th>
<th>Start time</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Idle-time irregularity table (IIT)</th>
<th>(sorted by start time)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Start time</th>
<th>Duration</th>
</tr>
</thead>
</table>

**This loop runs for ever**

- **start**
  - Should schedule an idle interval now?
    - yes: Busy-wait until the end of idle interval
    - no: Is there an irregular job that must start now?
      - yes: Find the highest priority pending job
      - no: Execute the job
  - Busy-wait until the end of its WCET
  - If one hyperperiod has passed, reset all time variables and local data

- WCETs are already padded to include scheduler overhead

- If one hyperperiod has passed, reset all time variables and local data
Implementation

- **Baseline online scheduling policy**: non-preemptive RM

- **Implementation platform**: Arduino
  - Entire implementation of OE scheduler is just 179 lines of simple C++ code (according to SLOCCount)
  - Possibility to store extra tables:
    - in flash memory
    - in RAM

- Available online at
Agenda

- Offline equivalence approach
- Efficient table generation
- Evaluation
- Conclusion

Task model
- Periodic Tasks
- Constrained deadline
- No release offset
Why Non-preemptive Scheduling is Hard?

The original problem is *job sequencing*:
- **Given** a set of jobs
- **Find** an ordering such that all timing constraints are met

**Branch and bound** is a common approach [Moore68, Pinedo16, ...]:
- Tries all possible combinations of the jobs in the ordering
- Even with pruning conditions it is still a combinatorial problem.
Why Non-preemptive Scheduling is Hard?

A heuristic approach: iterating backtracking

New job $J_i$
WCET $C_i$

Deadline miss
Why Non-preemptive Scheduling is Hard?

A heuristic approach: iterative backtracking

New job $J_i$
WCET $C_i$

Deadline miss

Deadline miss
Why Non-preemptive Scheduling is Hard?

A heuristic approach: iterative backtracking

New job $J_i$
WCET $C_i$

Successful

If it fails to find a schedule, a backtracking is needed to schedule the previous jobs differently.

This paper: To reduce the backtracking steps and improve the search speed, group jobs in chained windows!
What is a Chained Window?

A **chained window** is a tuple that represents a job sequence, a window of time, and a slack value and any schedule that starts and finishes the job sequence within the window, respects all timing constraints of the jobs.

![Diagram of chained window with jobs J1, J2, J3, and J4](image)
Chained Window Technique in a Nutshell

New job $J_i$
WCET $C_i$

Create a new chained windows

Merge

More details in the paper...
Agenda

- Offline equivalence approach
- Efficient offline table generation

Evaluation

- Conclusion
Main Questions

- How efficient is Offline Equivalence (OE)?
  - What is the memory requirement of OE?
  - What is the timing overhead of OE online scheduler?

  - Implementation platform:
    - Arduino Mega 5056
    - 6 KiB RAM, 256 KiB Flash memory, 16MHz processor speed

  - Measurements:
    - Required memory for OE tables (in Bytes)
    - OE online scheduler’s run time (in microseconds)

- How fast and efficient is the Chained Window technique?
  - Measurements:
    - Schedulability ratio for varying system utilization
    - Schedulability ratio for varying time budget
**Offline Equivalence Reduces Memory Requirements**

This is the **best result** among all considered table generation algorithms.
Memory Savings Depend on the Table Generation Algorithm

![Graph showing memory savings with different algorithms.](image)

- CW-EDF
- Chained Window
- Original Time Table

**Average Size of Stored Data (in bytes)**

- Total Utilization:
  - CW-EDF: 4, 10, 16, 29, 39, 49, 69, 90, 123
  - Chained Window: 104, 179, 272, 613
  - Original Time Table: 1,160, 1,923, 2,384, 2,657

**Comparison:****
- 9.3x improvement
- 2.1x improvement
What is the Runtime Overhead of OE?

Scheduler invocation overhead (in microseconds)

- Max
- Min
- Avg

3 Tasks
6 Tasks
9 Tasks
12 Tasks

- CW-EDF
- NP-EDF
- NP-RM
- OE (RAM)
- TD (RAM)

76
48
36
44
32

156
72
32
56
32

104
36
64
32

136
44
76
32

404
Main Questions

- How efficient is Offline Equivalence (OE)?
  - What is the memory requirement of OE?
  - What is the timing overhead of OE online scheduler?
  - Implementation platform:
    - Arduino Mega 5056
    - 6 KiB RAM, 256 KiB Flash memory, 16MHz processor speed
  - Measurements:
    - Required memory for OE tables (in Bytes)
    - OE online scheduler’s run time (in microseconds)

- How fast and efficient is Chained Window technique?
  - Measured outputs:
    - Schedulability ratio for varying system utilization
    - Schedulability ratio for varying time budget
How Efficient is the Chained Window Technique?

- Chained Window
- NP-EDF
- BB-Naïve
- CW-EDF
- NP-RM
- BB-Moore

Graph showing the relationship between Schedulability Ratio and Total Utilization for different scheduling techniques.
How Fast is the Chained Window Technique?

More experiments in the paper.

10 tasks per task set. Utilization 0.9.
Agenda

- Related work
- Offline equivalence approach
- Efficient offline table generation
- Evaluation

Conclusions and future work
A Framework to Construct Customized Harmonic Periods for RTS

**What does it do?**

- Schedules task according to a given schedule
- Reduces memory consumption
- Has low runtime overhead

**What does it not do?**

- Guarantees that the extra required information fits in a the memory
- Minimizes memory consumption
- Optimal, i.e., is able to find a schedule for any feasible task set

---

**Offline Equivalence**

- Chained Window Technique

**What does it do?**

- Is fast and efficient in generating a schedule

**What does it not do?**

- Optimal, i.e., is able to find a schedule for any feasible task set
A Framework to Construct Customized Harmonic Periods for RTS

Open Problems and Future Directions

- Generate a schedule with the least number of irregularities
- Find the best policy, parameters and encoding that minimizes the size of stored data
- Find a set of differential parameters such that differential data fits in a given memory size
Questions

Offline equivalence available at

http://people.mpi-sws.org/~bbb/papers/details/rtas17m/index.html

Thank you