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</th>
<th>Core</th>
<th>Operating Frequency (MHz) (Processor speed)</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>STM32L071C8</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 10x10 x1.4</td>
<td>ARM Cortex-M...</td>
<td>32</td>
<td>192</td>
<td>20</td>
<td>51</td>
</tr>
<tr>
<td>STM32L071VB</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

Offline Equivalence

- No need to store a table
- Stores less information

Table-driven scheduling or cyclic executive

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

Off-line scheduling

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 are {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
This Paper: Offline Equivalence

- Offline table generator
- Scheduling table
- Online policy
- Online scheduling algorithm
- Scan the table and store differences
- Differential data (irregularities)
- Types of irregularities:
  - Priority inversion
  - Idle interval
- Modify online scheduler to use differential data
- Our online Scheduler (OE)
- Modified online scheduling algorithm
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**
- **Scan the table and Store differences**
- **Online scheduling algorithm**
- **Schedule table**
- **Our online scheduler (OE)**
- **Differential data (irregularities)**
- **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-time irregularity table (IIT)

<table>
<thead>
<tr>
<th></th>
<th>From time 9, for 1 time unit</th>
</tr>
</thead>
</table>

#### Priority inversion table (PIT)

<table>
<thead>
<tr>
<th></th>
<th>The 3rd Job of ( \tau_2 ) starts at 30</th>
</tr>
</thead>
</table>

\( \tau_3 = (8, 60) \)

\( \tau_2 = (6, 12) \)

\( \tau_1 = (3, 10) \)
Modifying Baseline Online Scheduler to Use Differential Data

This loop runs for ever

Should schedule an idle interval now?

Is there an irregular job that must start now?

Find the highest priority pending job

Execute the job

Busy-wait until the end of its WCET

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

Busy-wait until the end of idle interval

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>

Idle-time irregularity table (IIT)
(sorted by start time)

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

Is there an irregular job that must start now?

WCETs are already padded to include scheduler overhead

Should schedule an idle interval now?

This loop runs for ever

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 200 lines of simple C++ code
  - Possibility to store extra tables:
    - in flash memory
    - in RAM
  - Available online at
    - [People.mpi-sws.org/~bbb/papers/details/rtas17m/index.html](http://People.mpi-sws.org/~bbb/papers/details/rtas17m/index.html)
Agenda

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

Task model
- Periodic Tasks
- Constrained deadline
- No release offset

Strongly NP-Hard!
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.

A simpler approach: **iterative backtracking**

1. **For each possible schedule** for $J_i$
   1.1. If $J_i$ and all other scheduled jobs meet their timing constrains
      1.1.1. **Recursively try to schedule** $J_{i+1}$ (all other not scheduled jobs)
      1.1.2. If succeeded, return the schedule

This paper:
To reduce the backtracking steps and improve the search speed,
**group jobs in chained windows!**
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.
Chained Window Technique in a Nutshell

New job $J_i$

WCET $C_i$

Create a new chained windows

Merge
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.

Average Size of Stored Data (in bytes)

Total Utilization
Memory Savings Depend on the Table Generation Algorithm

![Graph showing memory savings depend on the table generation algorithm. The graph plots the average size of stored data (in bytes) against total utilization. Three algorithms are compared: CW-EDF, Chained Window, and Original Time Table. The graph indicates that CW-EDF offers significant memory savings at lower total utilizations compared to the other two algorithms. At a total utilization of 0.1, CW-EDF saves about 9.3 times the memory compared to the Original Time Table. At higher utilizations, the savings are still substantial, with CW-EDF using significantly less memory than the other two.]

Average Size of Stored Data (in bytes)

Total Utilization

<table>
<thead>
<tr>
<th>CW-EDF</th>
<th>Chained Window</th>
<th>Original Time Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,519</td>
<td>2,513</td>
<td>2,450</td>
</tr>
<tr>
<td>2,518</td>
<td>2,530</td>
<td>2,555</td>
</tr>
<tr>
<td>2,480</td>
<td>2,571</td>
<td>2,657</td>
</tr>
</tbody>
</table>

0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9

0  500  1,000  1,500  2,000  2,500  3,000
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)

450
400
350
300
250
200
150
100
50
0
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?

![Graph showing schedulability ratio vs utilization for different techniques including Chained Window (CW-EDF), NP-EDF, NP-RM, and BB-Naïve and BB-Moore.](image-url)
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

Conclusion and future work
Summary and Conclusions

**What does it do?**

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

- **Chained Window Technique**
  - Is fast and efficient in generating a schedule

**What does it not do?**

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