An off-line multiprocessor real-time scheduling algorithm to reduce static energy consumption

First Workshop on Highly-Reliable Power-Efficient Embedded Designs
Shenzhen, China

Vincent Legout, Mathieu Jan, Laurent Pautet

CEA LIST

February 24, 2013

Introduction

- Context: embedded hard real-time systems (e.g. automotive, ...)
  - Hard real-time: time constraints cannot be violated
- Reducing energy consumption has several advantages:
  - Increase the autonomy of battery-powered systems
  - Provide energy efficient or “green” solutions
- Target multiprocessor systems
  - Uniprocessor are now superseded by multiprocessor systems
  - Almost no multiprocessor solution exists to reduce consumption

Objective

Schedule multiprocessor real-time systems to reduce consumption

Modeling the consumption of processors

Focusing on static consumption

- Dynamic and static consumption ($P = P_{dynamic} + P_{static}$)
  - $P_{static} = constant$ ($leakage\ current$)
- Static consumption now dominates dynamic consumption
  - Technology evolves: higher density, smaller supply voltage
- 2 software solutions to reduce consumption
  - Dynamic consumption: Dynamic Voltage & Frequency Scaling
  - Static consumption: Dynamic Power Management

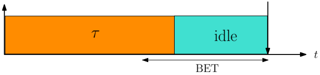
DPM: Dynamic Power Management

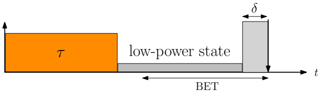
Reduce static consumption by using the low-power states of processors

- In a low-power state:
  - Processor inactive and energy consumption reduced
  - Transition delay δ required to get back to the active state

- Several low-power states available
  - The lower the consumption, the higher the transition delay

- Break-Even Time (BET): smallest idle period for which activating a low-power state saves more energy than letting the processor idle





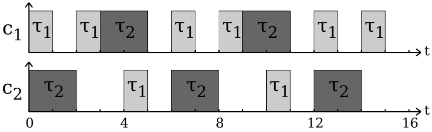
Objective: activate the most efficient low-power states without increasing the number of preemptions and migrations

DPM multiprocessor scheduling algorithms

The objective being to create large idle periods

- Partitioned scheduling (no migration)
  - Use uniprocessor DPM solutions on each processor
  - Cannot merge idle periods from different processors
- Global scheduling (migrations allowed)
  - Minimize the number of processors [Bhatti09]
    - Non-optimal scheduling algorithm
- Existing DPM scheduling algorithms are not suitable
  - Schedule decisions should be taken according to the workload
  - Instead, they use task characteristics (e.g. earliest deadline)

	τ_1	τ_2
WCET	1	2
Period	2	3



Approach

- Solve the problem off-line. Several advantages:
  - Compute a schedule in a hyperperiod
    - Without taking scheduling decisions for each task individually
    - But for the task set as a whole in a hyperperiod
  - Use linear programming to express constraints and objectives:
    - Real-time constraint: no deadline miss
    - Consumption objective: large idle periods
  - Ensure a maximal consumption in the worst case scenario

- Computation using the Worst Case Execution Time (WCET)
- Larger idle periods means minimizing the number of idle periods

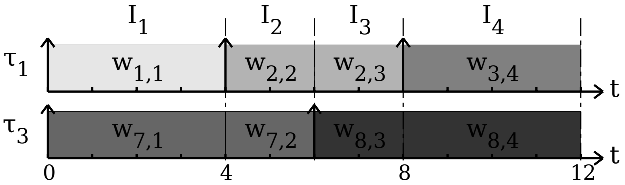
- LPDPM: Linear Programming DPM

Task model

- m processors, n tasks (WCET C, period T), hyperperiod H
- Global utilization $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$ $(m - 1 < U < m)$
- Division in intervals, for example :

	τ_1	τ_2	τ_3
WCET	0.8	2.4	4
Period	4	4	6
Jobs	3	3	2

- Objective to compute all weights of all tasks on all intervals
  - $w_{j,k}$ weight of job j on interval k



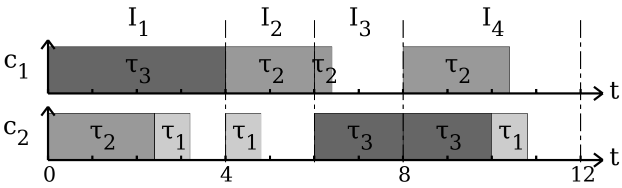
Initial linear system

- Set of constraints to keep the schedulability [Lemerre08]:
  - $|I_k|$: length of interval k
  - J_k: set of jobs in interval k
  - E_j: set of intervals where j is present

$\forall k, \sum_{j \in J_k} w_{j,k} \leq m$ — $U \leq m$ inside each interval

$\forall k, \forall j, 0 \leq w_{j,k} \leq 1$ — $0 \leq u \leq 1$ for each task inside each interval

$\forall j, \sum_{k \in E_j} w_{j,k} \times |I_k| = j.c$ — All jobs are fully executed



Add an additional idle task τ'

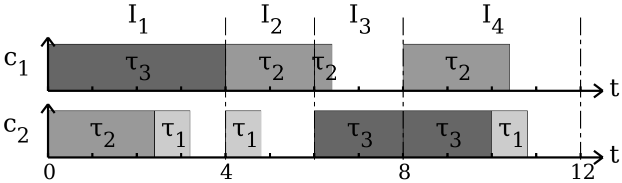
Accounts for the time where processors are supposed to be idle

- Utilization of $\tau' = m - U$, $period = H$
- As a consequence: global utilization $U = m$

- Only one processor can be idle simultaneously
- w_k weight of τ' on interval k

- Objective: reduce the number of preemptions of τ'

Interval	1	2	3	4
w_k	0.2	0.6	0.8	0.7



Add an additional idle task τ'

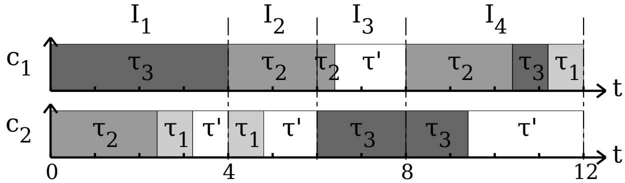
Accounts for the time where processors are supposed to be idle

- Utilization of $\tau' = m - U$, $period = H$
- As a consequence: global utilization $U = m$

- Only one processor can be idle simultaneously
- w_k weight of τ' on interval k

- Objective: reduce the number of preemptions of τ'

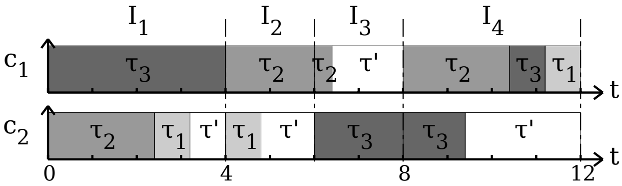
Interval	1	2	3	4
w_k	0.2	0.6	0.8	0.7



Reducing the number of preemptions of τ' (1)

- Add a first objective function in the linear system
- Avoid intervals where $0 < w_k < 1$
  - Because a preemption occurs
- Promote full intervals ($w_k = 1$) and empty intervals ($w_k = 0$)

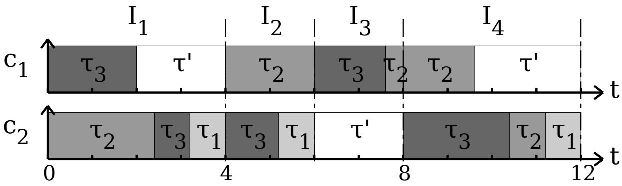
Interval	1	2	3	4
w_k	0.5	0	1	0.6



Reducing the number of preemptions of τ' (1)

- Add a first objective function in the linear system
- Avoid intervals where $0 < w_k < 1$
  - Because a preemption occurs
- Promote full intervals ($w_k = 1$) and empty intervals ($w_k = 0$)

Interval	1	2	3	4
w_k	0.5	0	1	0.6

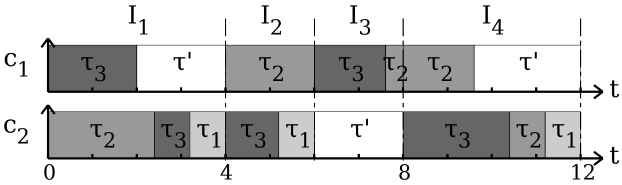


Reducing the number of preemptions of τ' (2)

- Add a second objective function in the linear system

- Avoid preemptions between intervals
- Group full intervals and group empty intervals

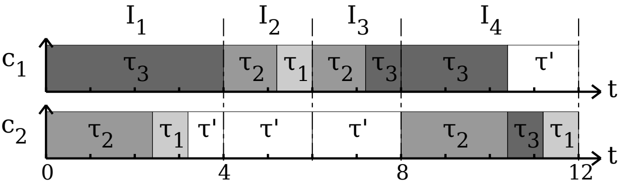
Interval	1	2	3	4
w_k	0.2	1	1	0.4



Reducing the number of preemptions of τ' (2)

- Add a second objective function in the linear system
- Avoid preemptions between intervals
- Group full intervals and group empty intervals

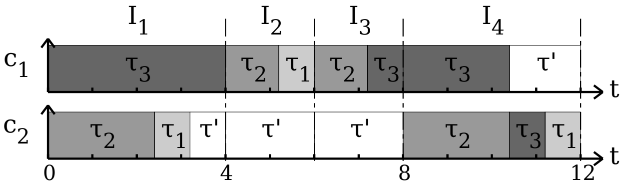
Interval	1	2	3	4
w_k	0.2	1	1	0.4



On-line schedule inside intervals

Using the weights computed off-line using the linear program

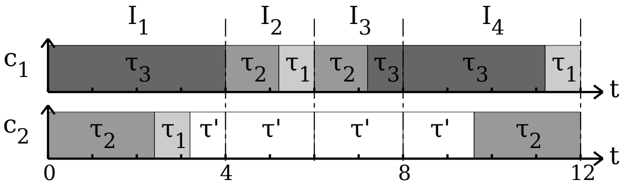
- Schedule impacts idle periods only when $0 < w_k < 1$
  - Solution: schedule τ' at the beginning or at the end the interval
- During the execution of τ', activate the deepest low-power state
  - Processor stays idle if the idle period is not large enough



On-line schedule inside intervals

Using the weights computed off-line using the linear program

- Schedule impacts idle periods only when $0 < w_k < 1$
  - Solution: schedule τ' at the beginning or at the end the interval
- During the execution of τ', activate the deepest low-power state
  - Processor stays idle if the idle period is not large enough



Experimental evaluation

- In a simulator, schedule random task sets with LPDPM and two existing optimal multiprocessor schedulers RUN & U-EDF

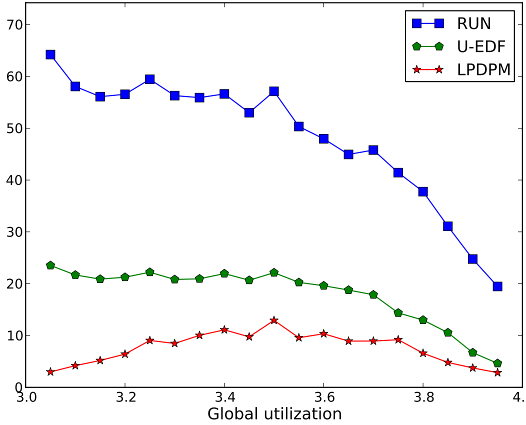
- No other DPM optimal multiprocessor scheduling algorithm
- RUN & U-EDF: schedulers reducing the number of preemptions

- 4 processors, 10 tasks
- 2000 task sets for each global utilization between 3.05 and 3.95

- Objective: evaluate the number and the length of the idle periods as well as the number of preemptions

Mean number of idle periods

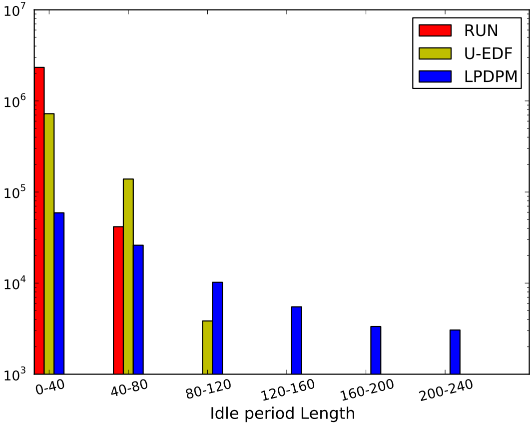
- Between 2 and 5 times less idle periods for LPDPM



Idle period lengths

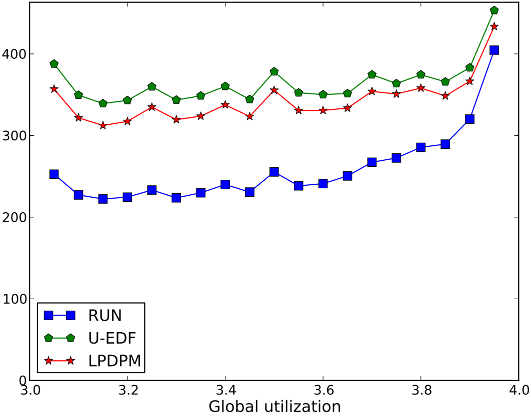
Using all idle periods generated by all task sets

- RUN & U-EDF: smaller idle periods, with a length always < 120
- LPDPM: larger idle periods, can be twice as large



Mean total number of preemptions

- Fewer preemptions than U-EDF
- Less than 1.4 more preemptions than RUN



Conclusion and perspectives

- Contribution: LPDPM, an off-line optimal multiprocessor real-time scheduling algorithm reducing static consumption
  - LPDPM generates less and larger idle periods such that deeper low-power states can be activated. The total number of preemptions does not increase.
  - Global consumption simulation using the low-power states of the STM32L board (Cortex-M3): LPDPM is up to 8% more energy efficient than existing optimal scheduling algorithms
- Perspectives
  - When tasks do not use their WCET, schedule intervals to further extend the existing idle periods
  - Temperature: impact on the energy consumption