An Energy-Aware Algorithm Exploiting Limited Preemptive Scheduling under Fixed Priorities

M. Bambagini, M. Bertogna, M. Marinoni and G. Buttazzo

8th IEEE International Symposium on Industrial Embedded Systems
Porto, Portugal, June 19, 2013
Outline

1. Introduction
2. System model
3. Algorithm
4. Experimental results
5. Conclusions
Introduction

This paper presents an energy saving scheduling algorithm for periodic real-time tasks on single core fixed-priority systems.
Introduction

This paper presents an energy saving scheduling algorithm for periodic real-time tasks on single core fixed-priority systems

Main contributions:
- mix of DVFS and DPM approaches

Our algorithm, with preemption overhead, consumes down to 17% less than the actual state of art on DPM/DVFS sensitive systems
Introduction

This paper presents an energy saving scheduling algorithm for periodic real-time tasks on single core fixed-priority systems.

Main contributions:

- mix of DVFS and DPM approaches
- platform-aware algorithm

Our algorithm, with preemption overhead, consumes down to 17% less than the actual state of art on DPM/DVFS sensitive systems.
Introduction

This paper presents an energy saving scheduling algorithm for periodic real-time tasks on single core fixed-priority systems

Main contributions:

▶ mix of DVFS and DPM approaches
▶ platform-aware algorithm
▶ preemption overhead is considered
Introduction

This paper presents an energy saving scheduling algorithm for periodic real-time tasks on single core fixed-priority systems.

Main contributions:

- mix of DVFS and DPM approaches
- platform-aware algorithm
- preemption overhead is considered
- exploiting advantages of the limited preemptive task model
Introduction

This paper presents an energy saving scheduling algorithm for periodic real-time tasks on single core fixed-priority systems.

Main contributions:
- mix of DVFS and DPM approaches
- platform-aware algorithm
- preemption overhead is considered
- exploiting advantages of the limited preemptive task model
- low complexity (offline: pseudo polynomial, online: \(O(1)\))
This paper presents an energy saving scheduling algorithm for periodic real-time tasks on single core fixed-priority systems

Main contributions:

▶ mix of DVFS and DPM approaches
▶ platform-aware algorithm
▶ preemption overhead is considered
▶ exploiting advantages of the limited preemptive task model
▶ low complexity (offline: pseudo polynomial, online: $O(1)$)

Our algorithm, with preemption overhead, consumes down to 17% less than the actual state of art on DPM/DVFS sensitive systems
System model

Set of $m$ different speeds ($s = f / f_{max}$), sorted in ascending order

Preemption cost, $\xi$: constant value due to context switch, pipeline invalidation and cache-related preemption delay
Set of $m$ different speeds ($s = f/f_{\text{max}}$), sorted in ascending order

Preemption cost, $\xi$: constant value due to context switch, pipeline invalidation and cache-related preemption delay

$n$ periodic fixed-priority tasks, $\tau_1, \ldots, \tau_n$ (descending priority order)

Non preemptive WCET: $C_{i,\text{NP}}(s) = \alpha C_{i,\text{NP}} + (1 - \alpha) C_{i,\text{NP}} / s$
Preemption Point Placement: Bertogna, Buttazzo and Yao [8]

$q_{i,j}(s)$: length of the $j$-th non preemptive region of $\tau_i$
Power model

Power consumption in active state (Martin et al. [31]):

\[ P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0 \]
Power model

Power consumption in active state (Martin et al. [31]):

\[ P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0 \]

What is the best speed \( s \)?

Trade-off between required execution time and consumed power:

- scaling \( s \) up: shorter execution, higher power consumption
- scaling \( s \) down: longer execution, lower power consumption
Power model

Power consumption in active state (Martin et al. [31]):

\[ P(s) = K_3 s^3 + K_2 s^2 + K_1 s + K_0 \]

What is the best speed \( s \)?

Trade-off between required execution time and consumed power:

- scaling \( s \) up: shorter execution, higher power consumption
- scaling \( s \) down: longer execution, lower power consumption

The best \( s \) (denoted as \( s^* \)) minimizes the energy per cycle

\[ E_{cyc}(s) = \alpha \cdot P(s) + (1 - \alpha) \cdot \frac{P(s)}{s} \]

\( s^* \) does not consider what happens during idle intervals
Consider: $\alpha = 0.2$ and $P^{(1)}(s) = 0.9s^3 + 0.1$
Consider: $\alpha = 0.2$ and $P^{(2)}(s) = 0.278s + 0.722$
Power model

Additional energy saving feature: low-power states

- low power consumption
- no execution
- no negligible time overhead to handle a complete transition

Break-even time: \( \delta = \delta_{a \rightarrow \sigma} + \delta_{\sigma \rightarrow a} \)
Algorithm

The approach exploits the limited preemptive advantages:

1. lower speeds than in the fully-preemptive model
2. bounded preemption number (\#non preemptive regions)
3. longer blocking tolerance
Algorithm

The approach exploits the limited preemptive advantages:

1. lower speeds than in the fully-preemptive model
2. bounded preemption number (#non preemptive regions)
3. longer blocking tolerance

The algorithm is divided in two stages:

- offline DVFS step selects the most energy-effective speed
- online DPM step prolongs the time spent in low power state
For each task, the PPP algorithm finds the maximum chunk length as a function of higher priority tasks’ blocking tolerance.
Algorithm - Offline DVFS step

For each task, the PPP algorithm finds the maximum chunk length as a function of higher priority tasks’ blocking tolerance.

The offline phase finds the minimum $s$ in $[s^*, s_m]$ which guarantees the task set feasibility, also considering the preemption overhead.
For each task, the PPP algorithm finds the maximum chunk length as a function of higher priority tasks’ blocking tolerance.

The offline phase finds the minimum \( s \) in \([s^*, s_m]\) which guarantees the task set feasibility, also considering the preemption overhead.

Complexity: pseudo polynomial.
The PPP algorithm also returns the blocking tolerances, $\beta_i$.
The PPP algorithm also returns the blocking tolerances, $\beta_i$

This step runs at each idle. The CPU is put in sleep (if possible) until first job arrival plus $\beta_{min} = \min_{\tau_i} \beta_i$, using only a timer

The algorithm accounts for both static and dynamic slacks
The PPP algorithm also returns the blocking tolerances, $\beta_i$.

This step runs at each idle. The CPU is put in sleep (if possible) until first job arrival plus $\beta_{min} = \min_{\tau_i} \beta_i$, using only a timer.

The algorithm accounts for both static and dynamic slacks.

Complexity: $O(1)$
Consider a system with four speeds:

- $s_1 = 0.3$, $s_2 = 0.6$, $s_3 = 0.7$ and $s_4 = 1.0$

Two tasks, $\tau_1$ and $\tau_2$:

- $T_1 = 60$, $T_2 = 150$,
- $C_{1}^{NP}(s_4) = 18$, $C_{2}^{NP}(s_4) = 42$
- $\alpha_1 = 0.0$, $\alpha_2 = 0.0 \rightarrow C_i^{NP}(s) = C_i^{NP}/s$

Although the analysis uses preemption cost, here it is assumed null
Algorithm

Assuming $P(s) = 0.9s^3 + 0.1$, $s^* = 0.4$

<table>
<thead>
<tr>
<th>Speed</th>
<th>$C_1^{NP}(s)$</th>
<th>$C_2^{NP}(s)$</th>
<th>$\beta_{\text{min}}$</th>
<th>$U(s)$</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 = 0.3$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Discarded, $s_1 &lt; s^*$</td>
</tr>
</tbody>
</table>
**Algorithm**

Assuming \( P(s) = 0.9s^3 + 0.1, \ s^* = 0.4 \)

<table>
<thead>
<tr>
<th>Speed</th>
<th>( C_1^{NP}(s) )</th>
<th>( C_2^{NP}(s) )</th>
<th>( \beta_{min} )</th>
<th>( U(s) )</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>( s_1 = 0.3 )</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Discarded, ( s_1 &lt; s^* )</td>
</tr>
<tr>
<td>( s_2 = 0.6 )</td>
<td>30</td>
<td>70</td>
<td>&lt; 0</td>
<td>0.96</td>
<td>Not feasible</td>
</tr>
</tbody>
</table>
Algorithm

Assuming $P(s) = 0.9s^3 + 0.1$, $s^* = 0.4$

<table>
<thead>
<tr>
<th>Speed</th>
<th>$C_1^{NP}(s)$</th>
<th>$C_2^{NP}(s)$</th>
<th>$\beta_{min}$</th>
<th>$U(s)$</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 = 0.3$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Discarded, $s_1 &lt; s^*$</td>
</tr>
<tr>
<td>$s_2 = 0.6$</td>
<td>30</td>
<td>70</td>
<td>$&lt; 0$</td>
<td>0.96</td>
<td>Not feasible</td>
</tr>
<tr>
<td>$s_3 = 0.7$</td>
<td>26</td>
<td>60</td>
<td>34</td>
<td>0.83</td>
<td>Feasible, then exit</td>
</tr>
</tbody>
</table>

$\tau_1$ is executed non-preemptively

$\tau_2$ is divided in two chunks lasting 26 and 34
Algorithm

Execution without the online step: fragmented idle times

\[
\begin{align*}
\tau_1 & \quad \tau_2 \\
P(t) & \quad P(t)
\end{align*}
\]
Algorithm

Execution without the online step: fragmented idle times

Execution with the online step

\[ \beta_{\text{min}} \]
Algorithm

Assuming $P(s) = 0.278s + 0.722$, $s^* = 1.0$

<table>
<thead>
<tr>
<th>Speed</th>
<th>$C_1^{NP}(s)$</th>
<th>$C_2^{NP}(s)$</th>
<th>$\beta_{min}$</th>
<th>$U(s)$</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 = 0.3$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Discarded, $s_1 &lt; s^*$</td>
</tr>
<tr>
<td>$s_2 = 0.6$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Discarded, $s_2 &lt; s^*$</td>
</tr>
<tr>
<td>$s_3 = 0.7$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Discarded, $s_3 &lt; s^*$</td>
</tr>
</tbody>
</table>
Algorithm

Assuming $P(s) = 0.278s + 0.722$, $s^* = 1.0$

<table>
<thead>
<tr>
<th>Speed</th>
<th>$C_1^{NP}(s)$</th>
<th>$C_2^{NP}(s)$</th>
<th>$\beta_{min}$</th>
<th>$U(s)$</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1 = 0.3$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Discarded, $s_1 &lt; s^*$</td>
</tr>
<tr>
<td>$s_2 = 0.6$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Discarded, $s_2 &lt; s^*$</td>
</tr>
<tr>
<td>$s_3 = 0.7$</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Discarded, $s_3 &lt; s^*$</td>
</tr>
<tr>
<td>$s_4 = 1.0$</td>
<td>18</td>
<td>42</td>
<td>42</td>
<td>0.58</td>
<td>Feasible, then exit</td>
</tr>
</tbody>
</table>

Both $\tau_1$ and $\tau_2$ execute non-preemptively
Algorithm

Execution without the online step: fragmented idle times

\[
\begin{align*}
\tau_1 & \quad \tau_2 & \quad P(t) \\
\end{align*}
\]
Algorithm

Execution without the online step: fragmented idle times

Execution with the online step
Simulations

Experimental results were obtained through simulations

Simulation parameters:

- **DVFS-sensitive architecture:**
  - \( P^{(1)}(s) = 0.9s^3 + 0.1 \) and \( P^{(1)}_\delta = 0.05 \)

- **DPM-sensitive architecture:**
  - \( P^{(2)}(s) = 0.278s + 0.722 \) and \( P^{(2)}_\delta = 0.4 \)

- 19 discrete speeds in \([0.1, 1.0]\) with step 0.05

- 10 periodic tasks with \( C_i^{NP}(1.0) \in [100, 500] \) and \( \alpha_i = 0.2 \)

- Task generation algorithm: UUniFast (Bini et al. [12])
Simulations

Test 1: Average slowest speed analysis w/o preemption overhead

![Graph showing average lowest speed analysis with different preemptive strategies]

- Non-Preemptive
- Fully-Preemptive, $\xi=10$
- Fully-Preemptive, $\xi=0$
- Limited Preemptive, $\xi=10$
- Limited Preemptive, $\xi=0$
Simulations

Test 2: Contribution of each step on DVFS-sensitive architecture

Utilization, $\xi = 10$

- Pure DPM, $\delta = 0$
- Pure DVFS
- DVFS+DPM, $\delta = 500$
- DVFS+DPM, $\delta = 0$
Simulations

The following tests uses VOSS (Chen and Kuo [3]), the actual state of art for fixed priority systems using Rate Monotonic

Offline, VOSS computes the slowest possible speed using the more precise RTA (including $\xi$) instead of Liu and Layland’s bound

Online, at every idle, VOSS postpones each task arrival (by the first abs deadline) by its blocking tolerance and the estimated idle time

VOSS complexity: $O(n \cdot \log n)$ at each idle time
Simulations

Test 3: Comparisons with VOSS on DVFS-sensitive architecture

\[ \xi = 10, \delta = 750 \]
\[ \xi = 10, \delta = 500 \]
\[ \xi = 0, \delta = 750 \]
\[ \xi = 0, \delta = 250 \]
\[ \xi = 0, \delta = 500 \]
\[ \xi = 0, \delta = 250 \]
Test 4: Comparisons with VOSS on DPM-sensitive architecture

\[ \xi = 10, \ \delta = 750 \]
\[ \xi = 0, \ \delta = 750 \]
\[ \xi = 10, \ \delta = 500 \]
\[ \xi = 0, \ \delta = 500 \]
\[ \xi = 10, \ \delta = 250 \]
\[ \xi = 0, \ \delta = 250 \]
Conclusions

The proposed algorithm integrates DVFS and DPM techniques with limited preemptive tasks on fixed priority systems.

It outperforms VOSS with $O(1)$ complexity rather than $O(n \cdot \log n)$.

As future work, we aim at:

1. supporting sporadic tasks, common in event driven systems
2. improving DPM step to consider dynamic parameters, while keeping the overall complexity low
3. implementing such algorithm on a real system
thank you

m.bambagini@sssup.it