INSTITUTE
OF COMMUNICATION,
INFORMATION
AND PERCEPTION
TECHNOLOGIES

Scuola Superiore
Sant'Anna

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

TeCIP Institute, Scuola Superiore Sant'Anna
Area della Ricerca CNR, Via Moruzzi, 1
56124 Pisa, ITALY

Retis
Real-Time Systems Laboratory

Outline

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

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

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: $O(1)$)

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: $O(1)$)

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, ξ: constant value due to context switch, pipeline invalidation and cache-related preemption delay

System model

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

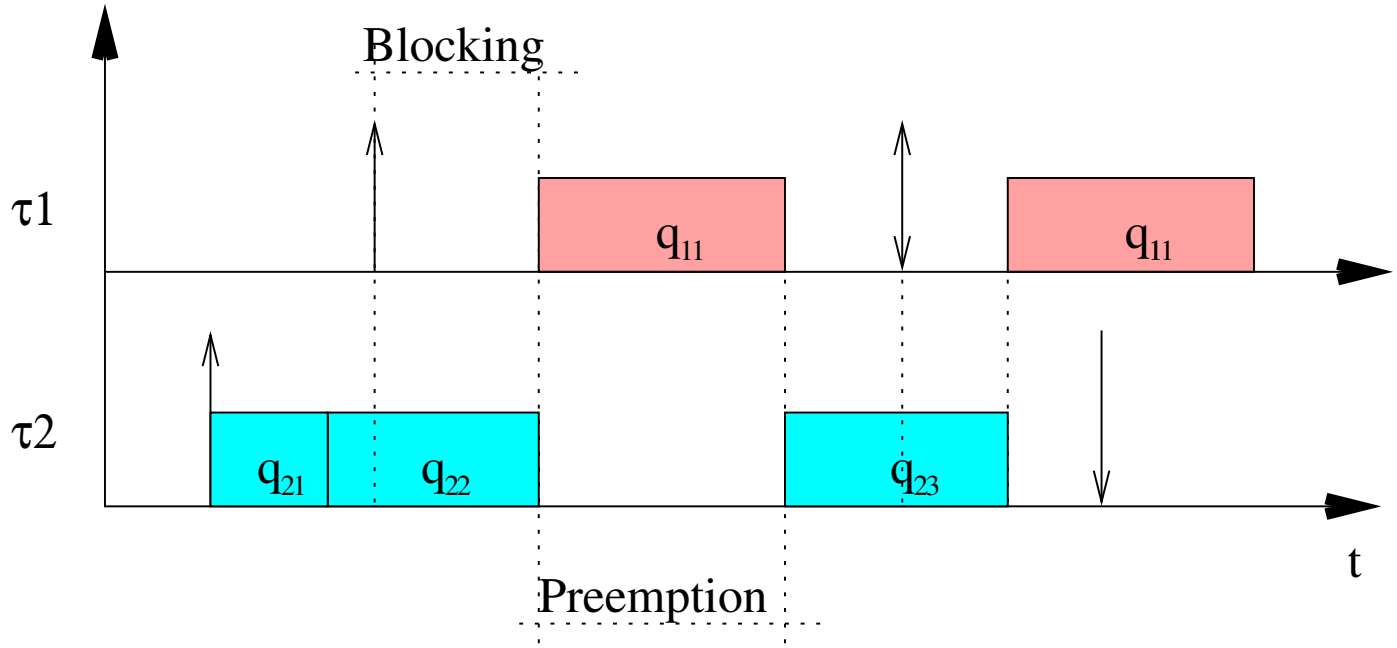
Preemption cost, ξ: constant value due to context switch, pipeline invalidation and cache-related preemption delay

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

Non preemptive WCET: $C_i^{NP}(s) = \alpha C_i^{NP} + (1 - \alpha) C_i^{NP}/s$

System model

Preemption Point Placement: Bertogna, Buttazzo and Yao [8]



$q_{i,j}(s)$: length of the j-th non preemptive region of τ_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

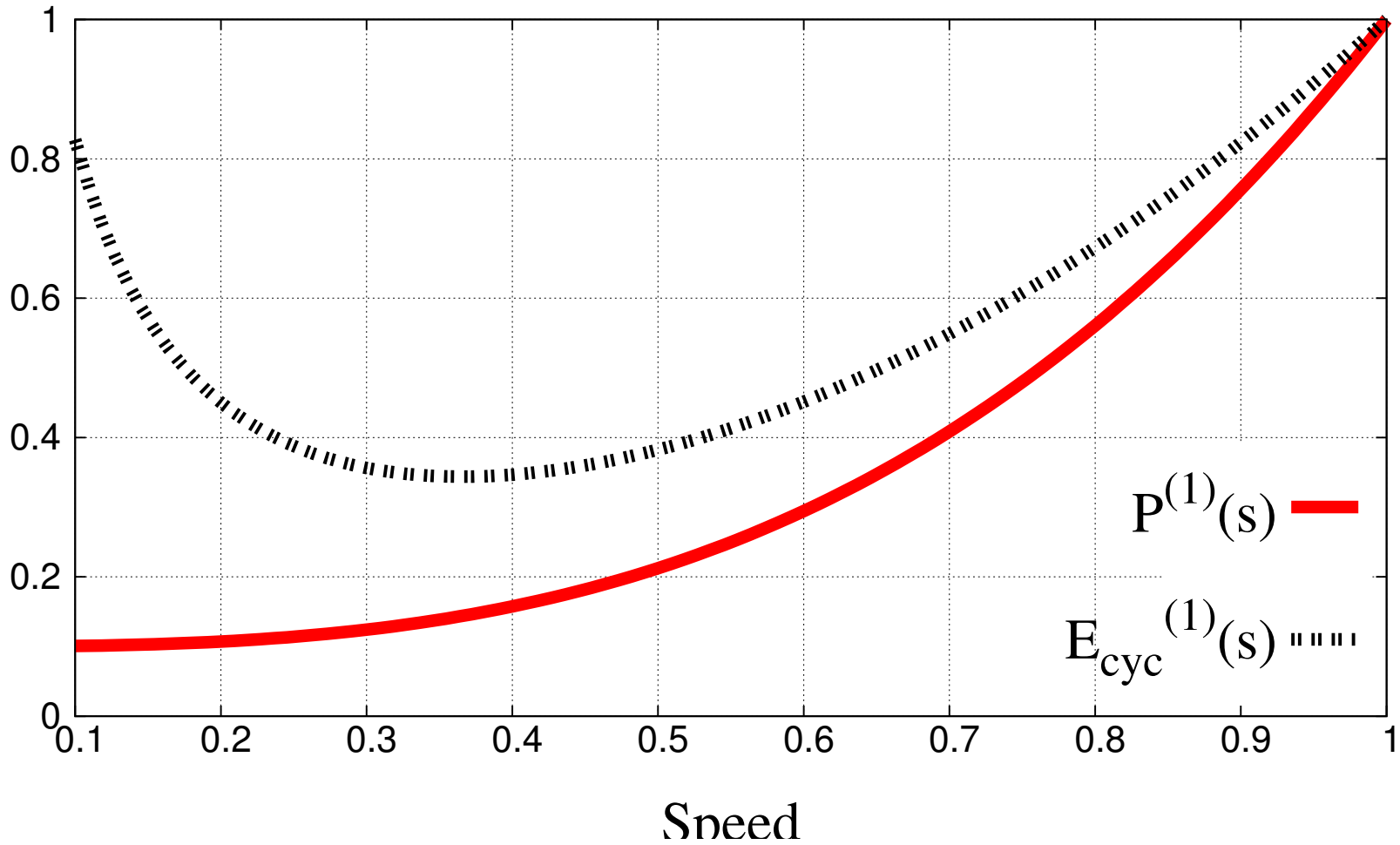
The best s (denoted as s^*) minimizes the energy per cycle

$$E_{cyc}(s) = \alpha \cdot P(s) + (1 - \alpha) \cdot P(s)/s$$

(s^* does not consider what happens during idle intervals)

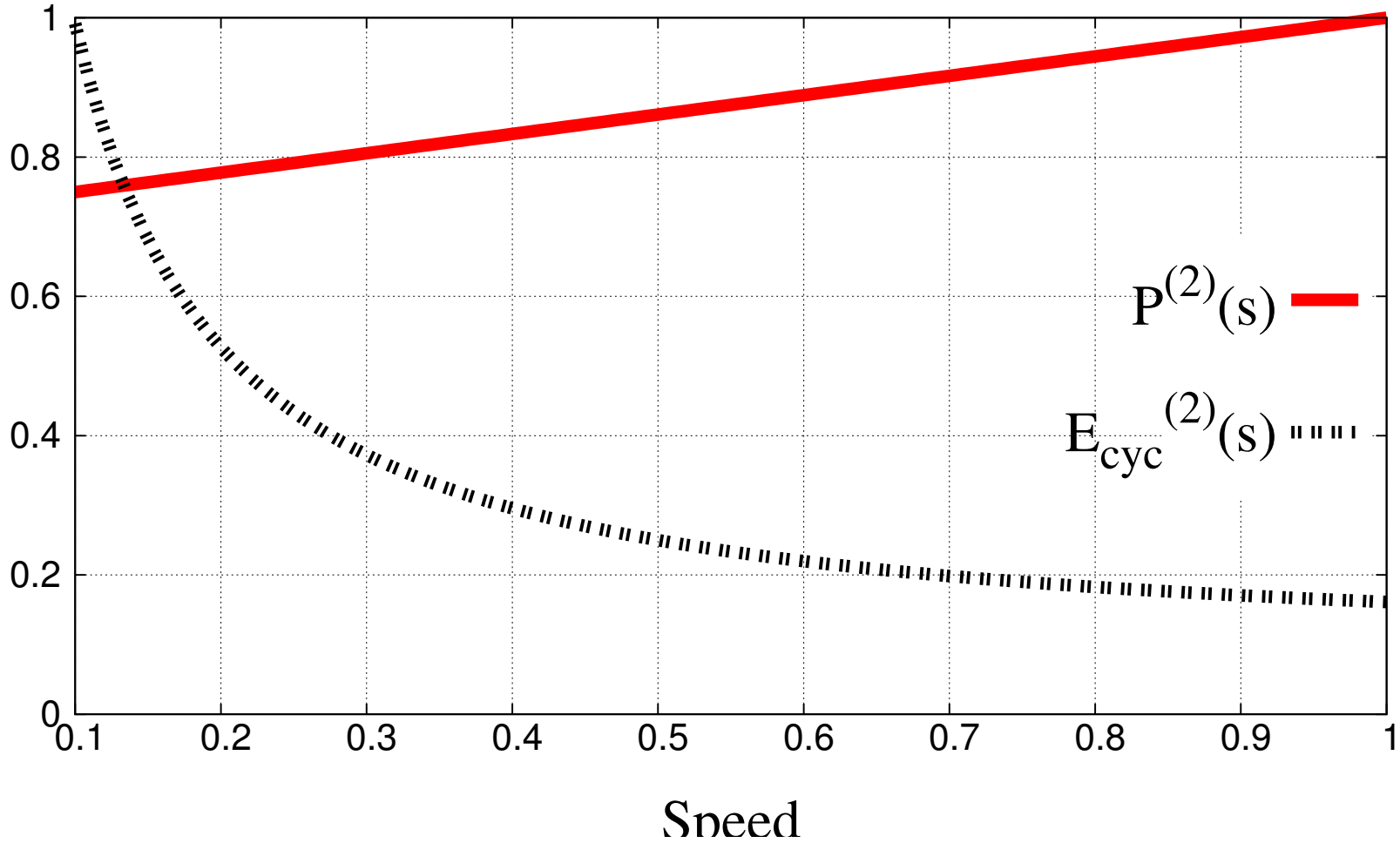
Power model

Consider: $\alpha = 0.2$ and $P^{(1)}(s) = 0.9s^3 + 0.1$



Power model

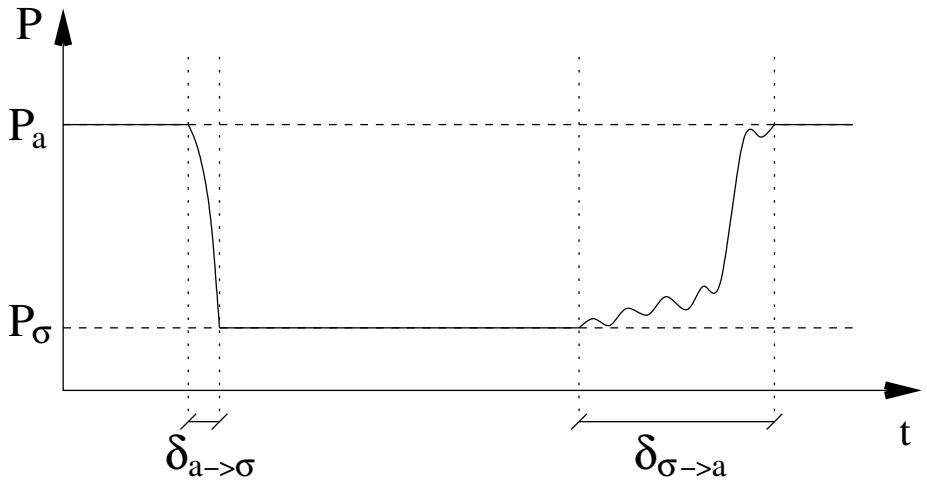
Consider: $\alpha = 0.2$ and $P^{(2)}(s) = 0.278s + 0.722$



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: $\delta = \delta_{a \to \sigma} + \delta_{\sigma \to a}$

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

Algorithm - Offline DVFS step

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

Algorithm - Offline DVFS step

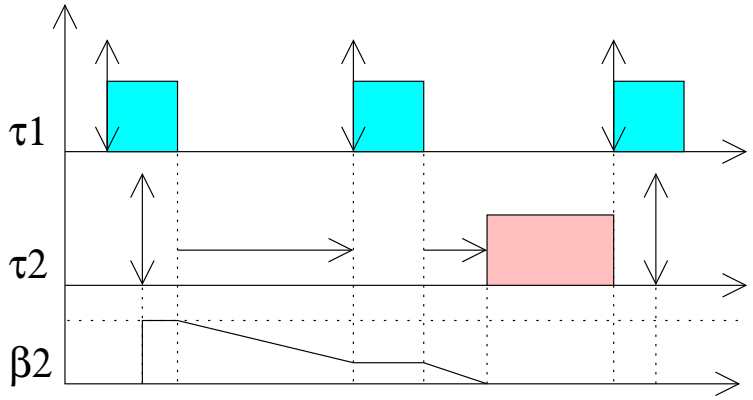
For each task, the PPP algorithm finds the maximum chunk length as a function of higher priority tasks' blocking tolerance

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

Complexity: pseudo polynomial

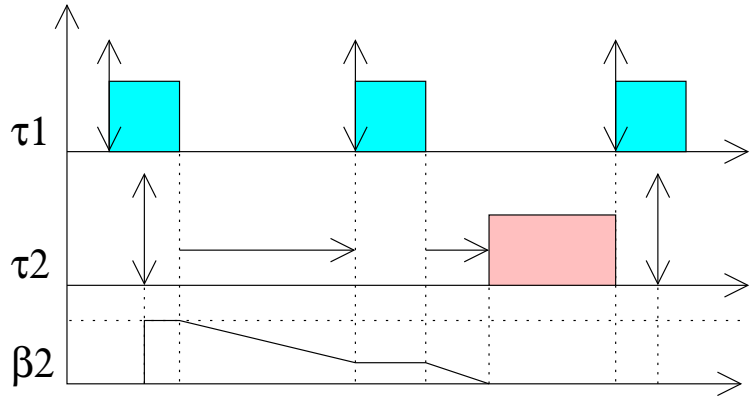
Algorithm - Online DPM step

The PPP algorithm also returns the blocking tolerances, β_i



Algorithm - Online DPM step

The PPP algorithm also returns the blocking tolerances, β_i

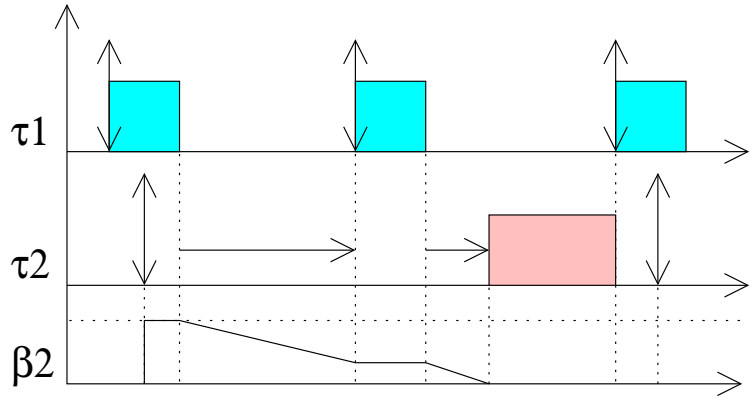


This step runs at each idle. The CPU is put in sleep (if possible) until first job arrival plus $\beta_{min} = \min_{\tau_i} \beta_i$, using only a timer

The algorithm accounts for both static and dynamic slacks

Algorithm - Online DPM step

The PPP algorithm also returns the blocking tolerances, β_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)

Algorithm

Consider a systems with four speeds:

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

Two tasks, τ_1 and τ_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 $P(s) = 0.9s^3 + 0.1$, $s^* = 0.4$

Speed	$C_1^{NP}(s)$	$C_2^{NP}(s)$	β_{min}	$U(s)$	Action
$s_1 = 0.3$	-	-	-	-	Discarded, $s_1 < s^*$

Algorithm

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

Speed	$C_1^{NP}(s)$	$C_2^{NP}(s)$	β_{min}	$U(s)$	Action
$s_1 = 0.3$	-	-	-	-	Discarded, $s_1 < s^*$
$s_2 = 0.6$	30	70	< 0	$0.9\bar{6}$	Not feasible

Algorithm

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

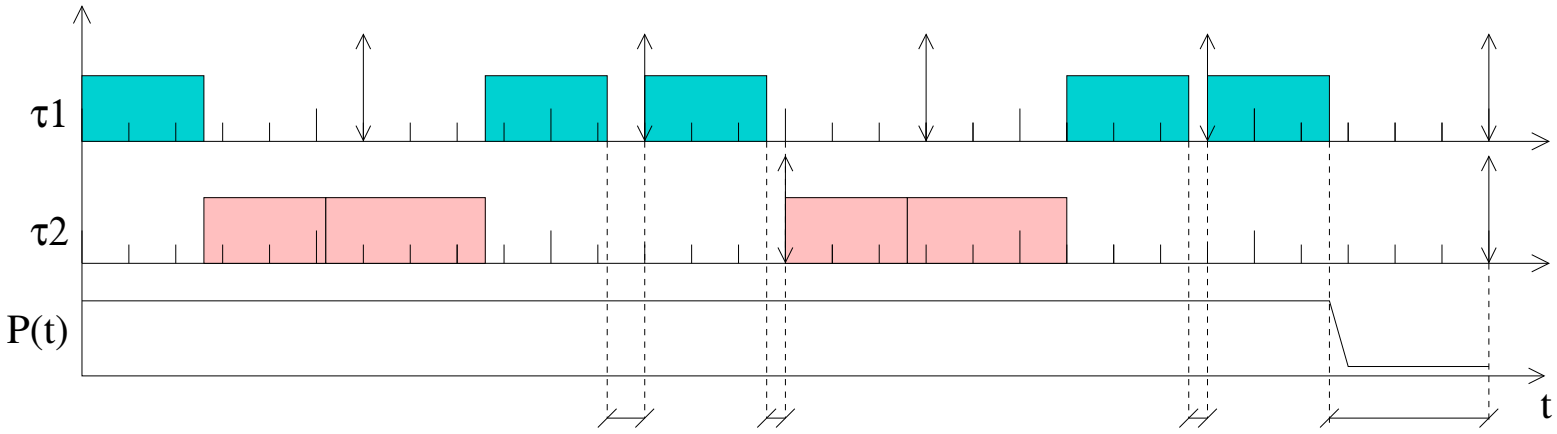
Speed	$C_1^{NP}(s)$	$C_2^{NP}(s)$	β_{min}	$U(s)$	Action
$s_1 = 0.3$	-	-	-	-	Discarded, $s_1 < s^*$
$s_2 = 0.6$	30	70	< 0	$0.9\bar{6}$	Not feasible
$s_3 = 0.7$	26	60	34	$0.8\bar{3}$	Feasible, then exit

τ_1 is executed non-preemptively

τ_2 is divided in two chunks lasting 26 and 34

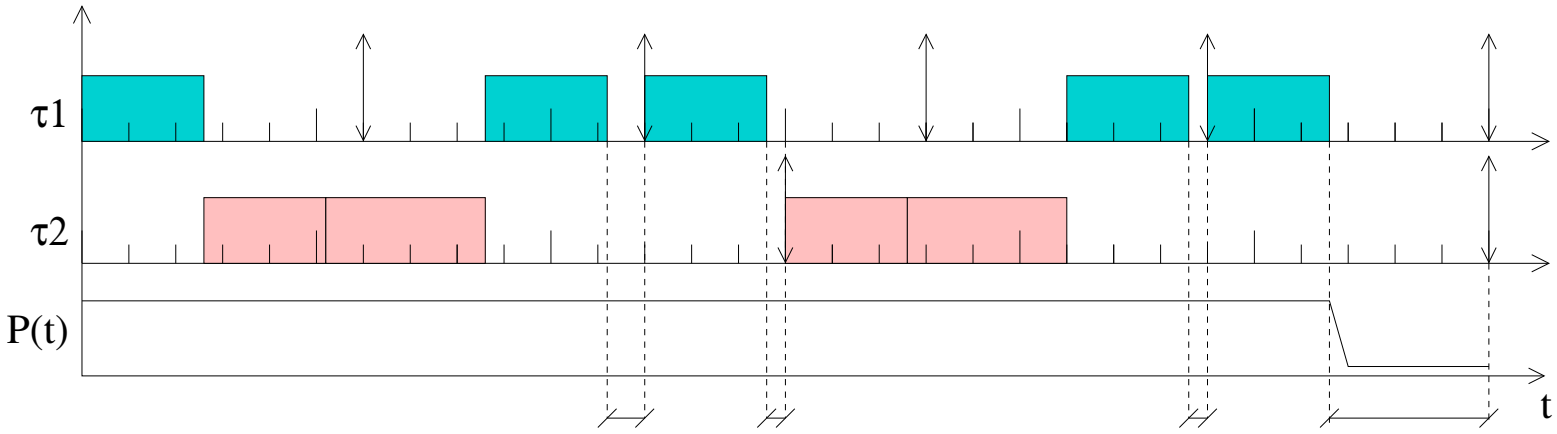
Algorithm

Execution without the online step: fragmented idle times

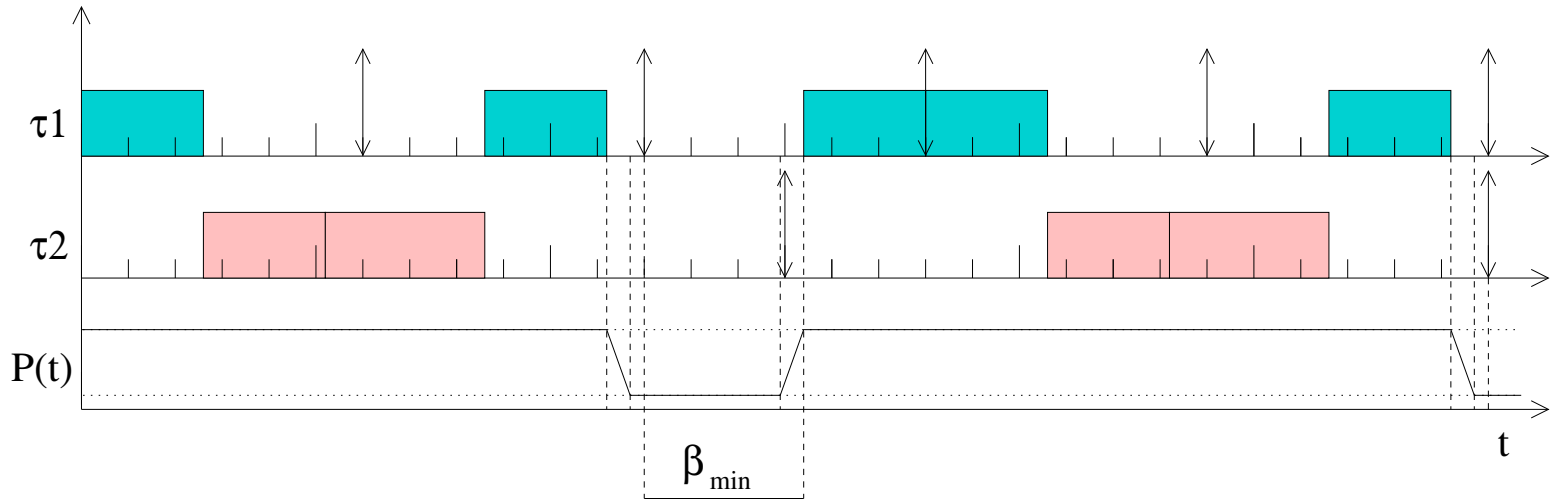


Algorithm

Execution without the online step: fragmented idle times



Execution with the online step



Algorithm

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

Speed	$C_1^{NP}(s)$	$C_2^{NP}(s)$	β_{min}	$U(s)$	Action
$s_1 = 0.3$	-	-	-	-	Discarded, $s_1 < s^*$
$s_2 = 0.6$	-	-	-	-	Discarded, $s_2 < s^*$
$s_3 = 0.7$	-	-	-	-	Discarded, $s_3 < s^*$

Algorithm

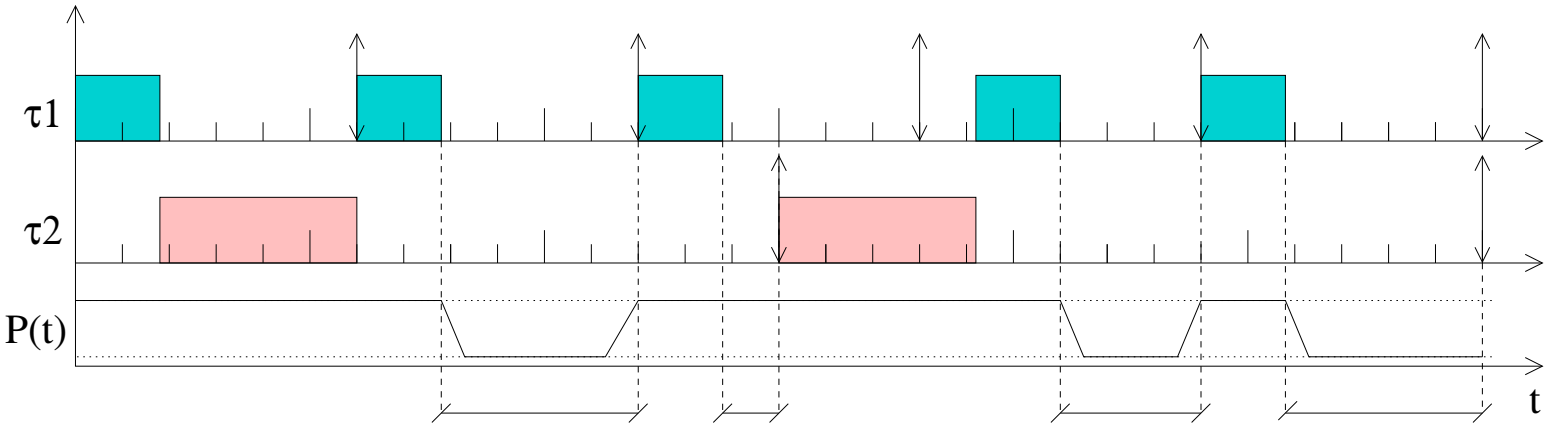
Assuming $P(s) = 0.278s + 0.722$, $s^* = 1.0$

Speed	$C_1^{NP}(s)$	$C_2^{NP}(s)$	β_{min}	$U(s)$	Action
$s_1 = 0.3$	-	-	-	-	Discarded, $s_1 < s^*$
$s_2 = 0.6$	-	-	-	-	Discarded, $s_2 < s^*$
$s_3 = 0.7$	-	-	-	-	Discarded, $s_3 < s^*$
$s_4 = 1.0$	18	42	42	0.58	Feasible, then exit

Both τ_1 and τ_2 execute non-preemptively

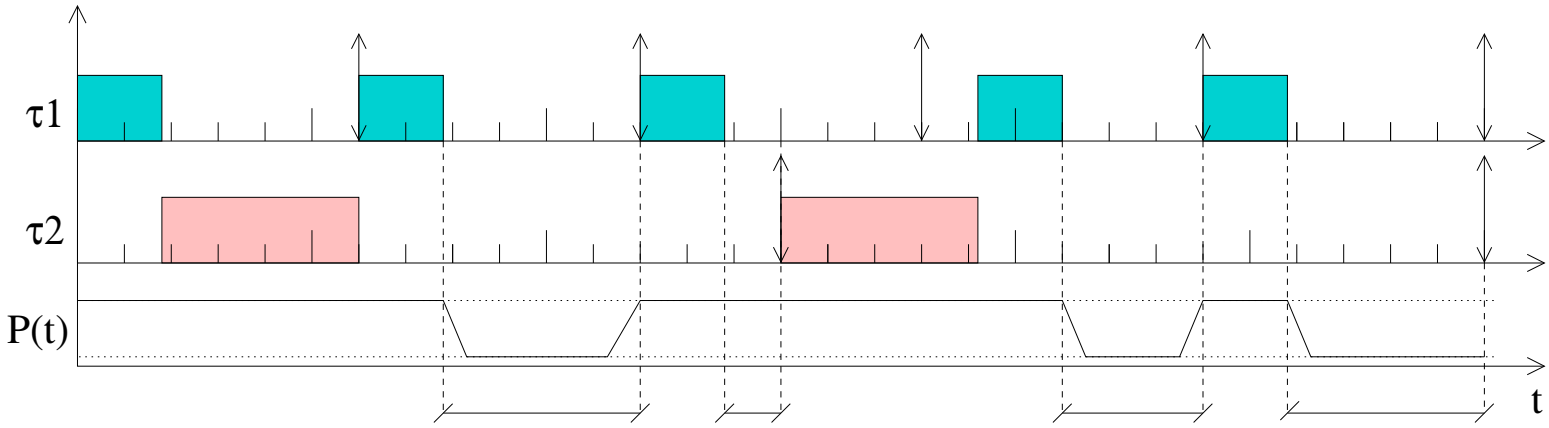
Algorithm

Execution without the online step: fragmented idle times

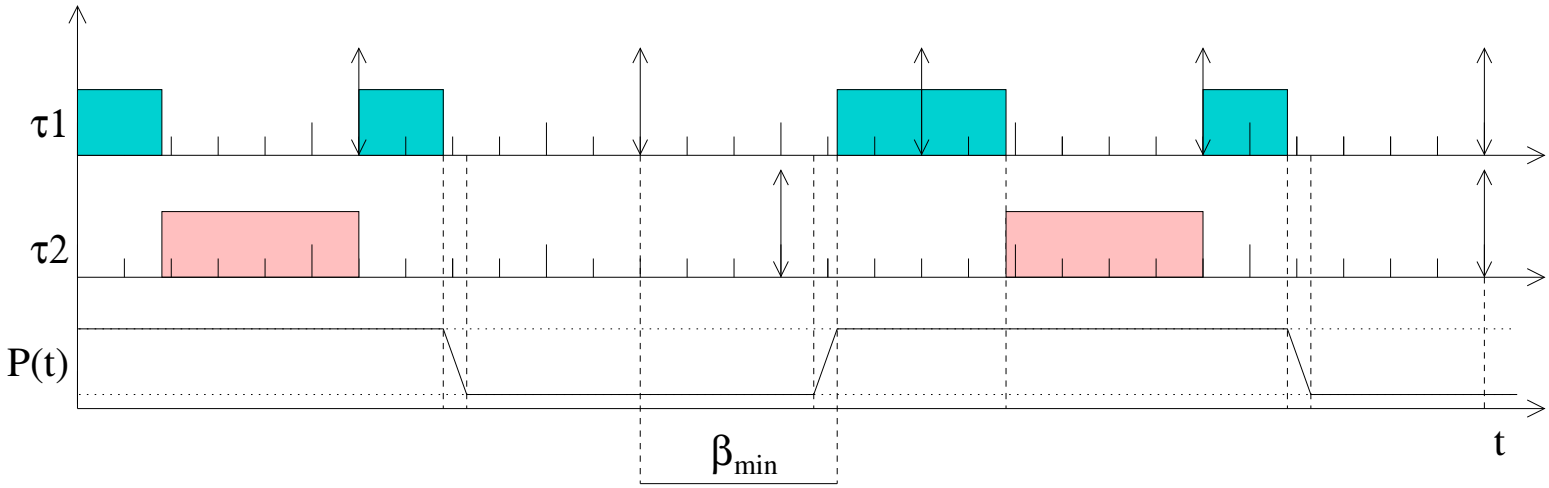


Algorithm

Execution without the online step: fragmented idle times



Execution with the online step



Simulations

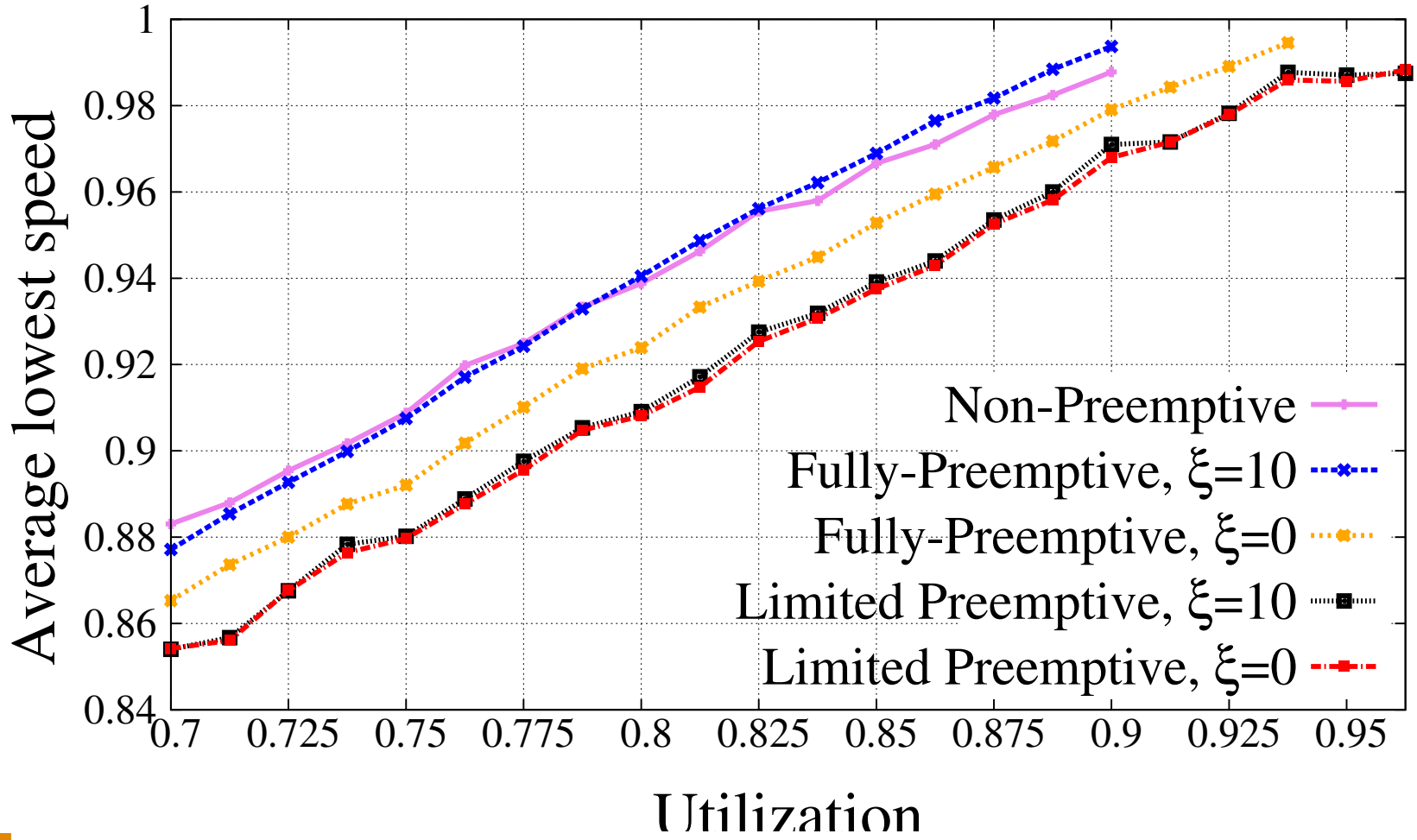
Experimental results were obtained through simulations

Simulation parameters:

- DVFS-sensitive architecture:
  - $P^{(1)}(s) = 0.9s^3 + 0.1$ and $P_\delta^{(1)} = 0.05$
- DPM-sensitive architecture:
  - $P^{(2)}(s) = 0.278s + 0.722$ and $P_\delta^{(2)} = 0.4$
- 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])

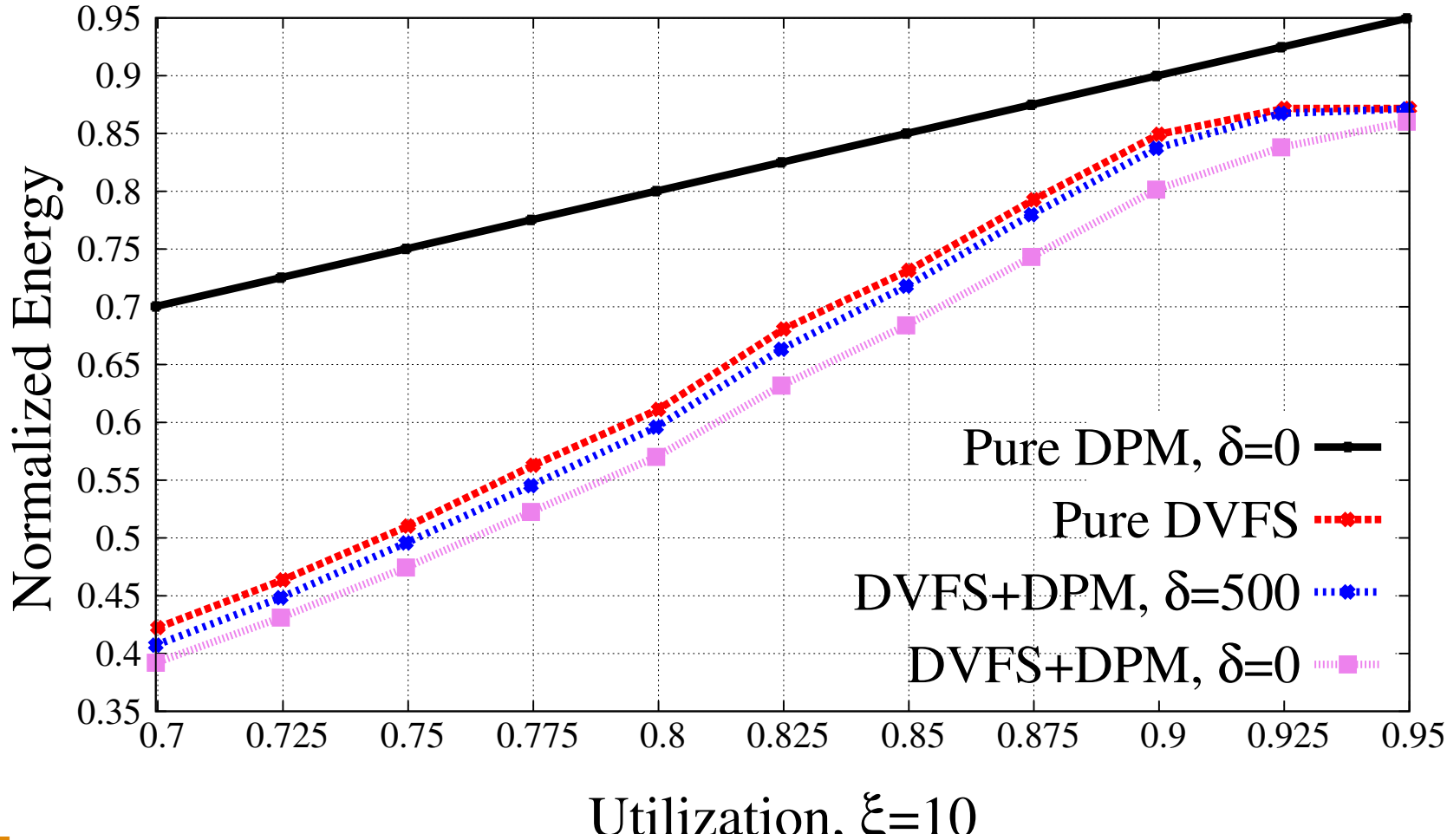
Simulations

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



Simulations

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



Simulations

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

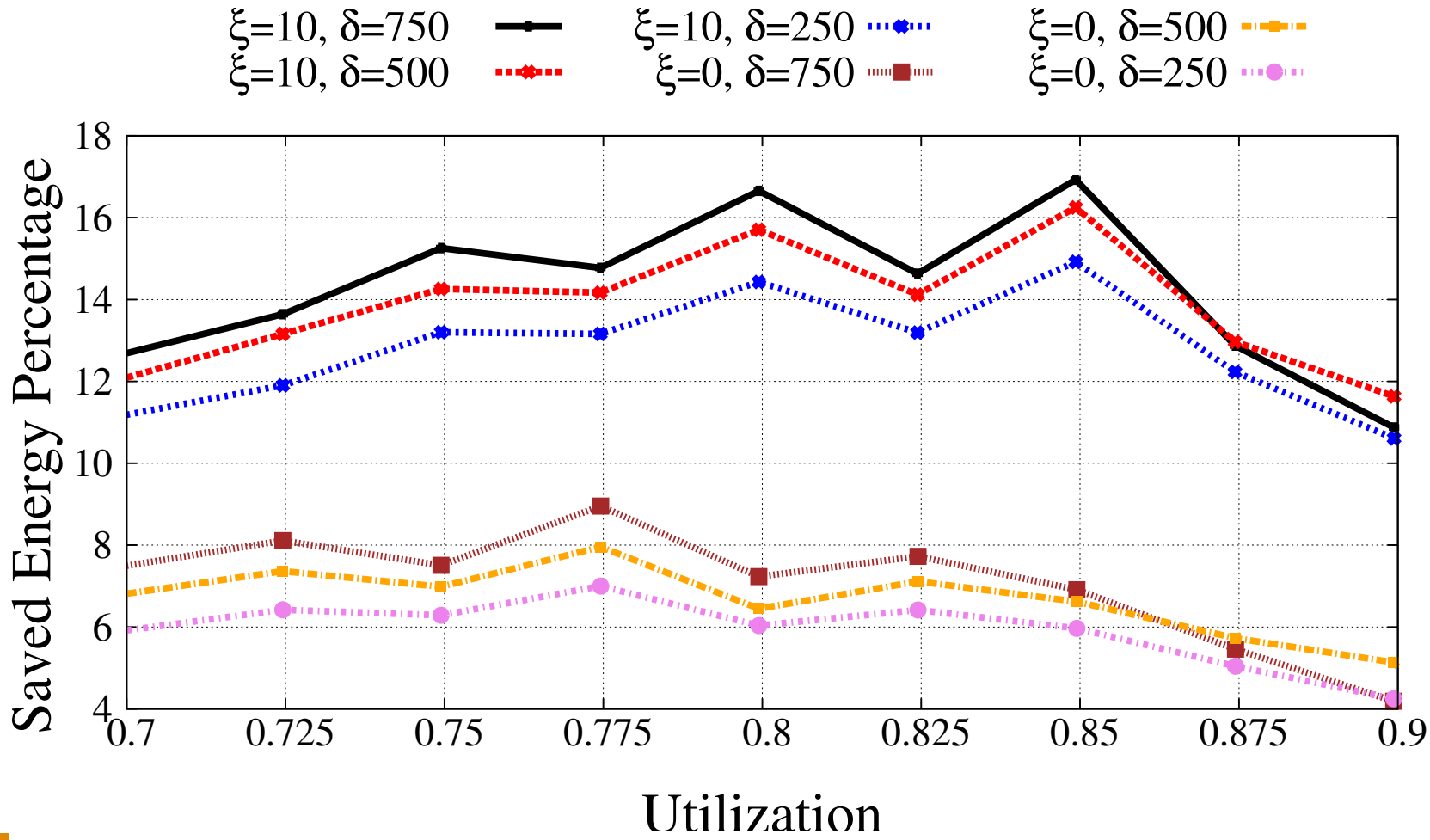
Offline, VOSS computes the slowest possible speed using the more precise RTA (including ξ) instead of Liu and Layland's bound

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

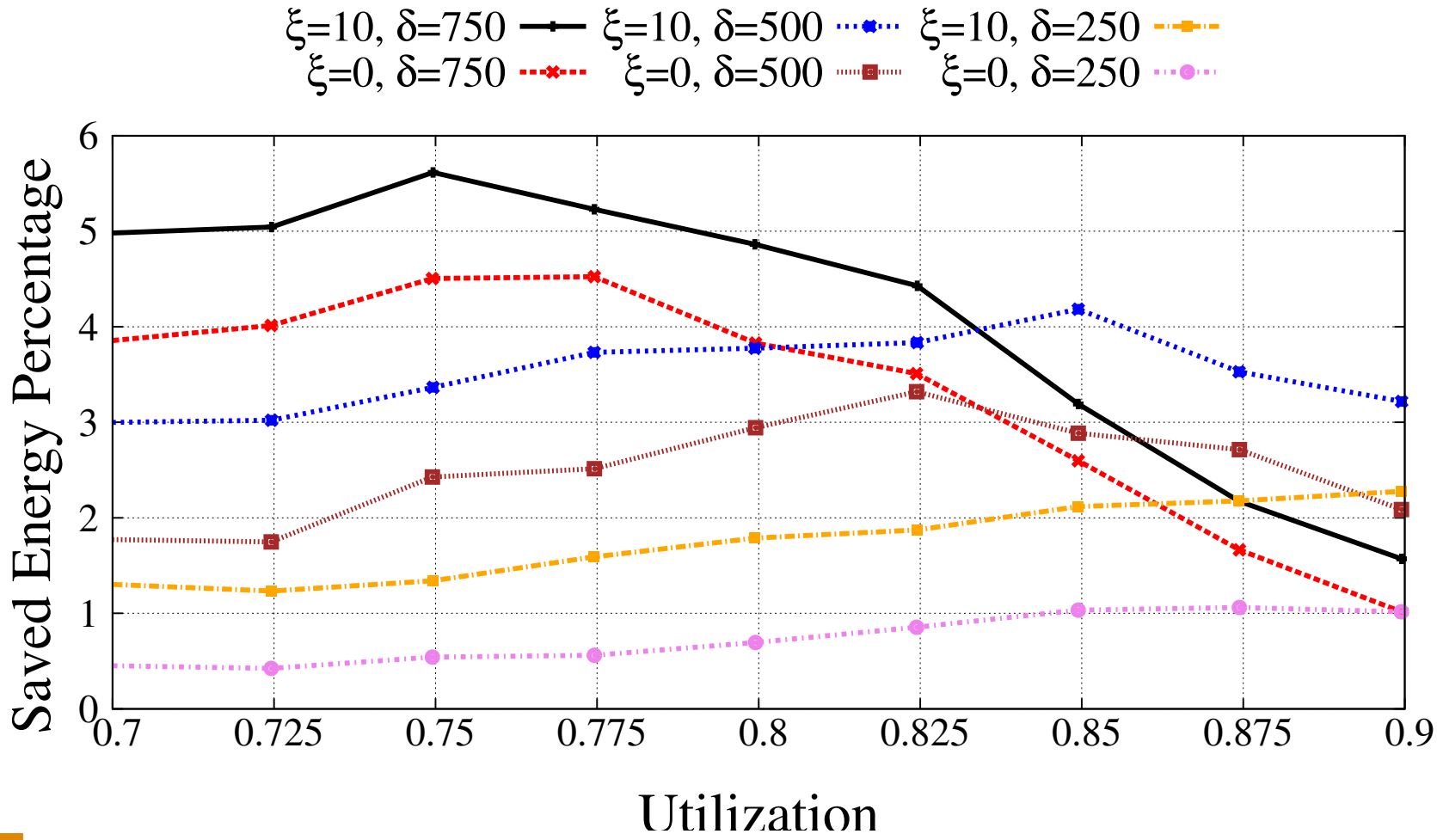
Simulations

Test 3: Comparisons with VOSS on DVFS-sensitive architecture



Simulations

Test 4: Comparisons with VOSS on DPM-sensitive architecture



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

Retis
Real-Time Systems Laboratory

thank you

m.bambagini@sssup.it