Allocation of Real-Time Tasks onto Identical Core Platforms under Deferred fixed Preemption-Point Model

Ikram Senoussaoui[1,2], Houssam-Eddine Zahaf[1], Mohammed Kamel Benhaoua[3], Giuseppe Lipari[1], Richard Olejnik[1].

[1]CRIStAL, Lille University, France
[2]LAPECI, Oran1 University, Algeria
[3]LAPECI, Mascara University, Algeria

Paris, RTNS'20, 9/10 June 2020

Table of content

1 Context and motivation

2 Deferred preemption models for single-core processor

3 Deferred preemption task allocation onto multi-core platform

4 Experiments results

5 Conclusion and future work

Table of content

1 Context and motivation

2 Deferred preemption models for single-core processor

3 Deferred preemption task allocation onto multi-core platform

4 Experiments results

5 Conclusion and future work

Introduction : target systems

Introduction : target systems

- Real-Time constraints

Introduction : target systems

- Real-Time constraints
- Computer vision applications :

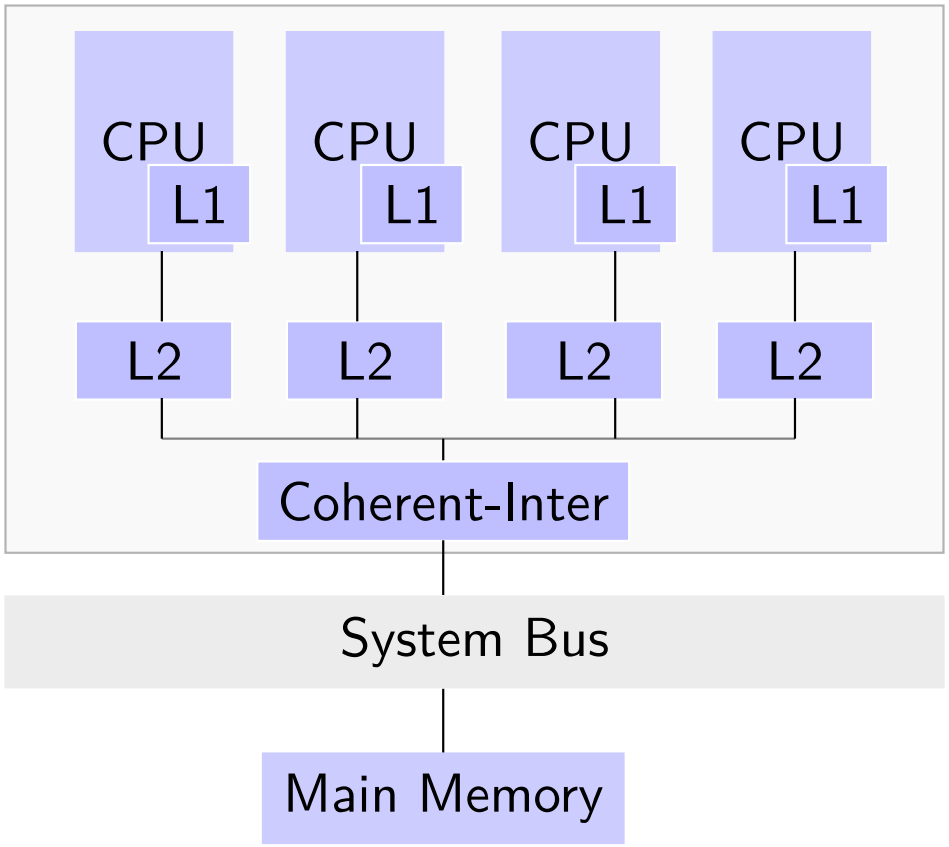
Introduction : target systems

- Real-Time constraints
- Computer vision applications : **large data sets**

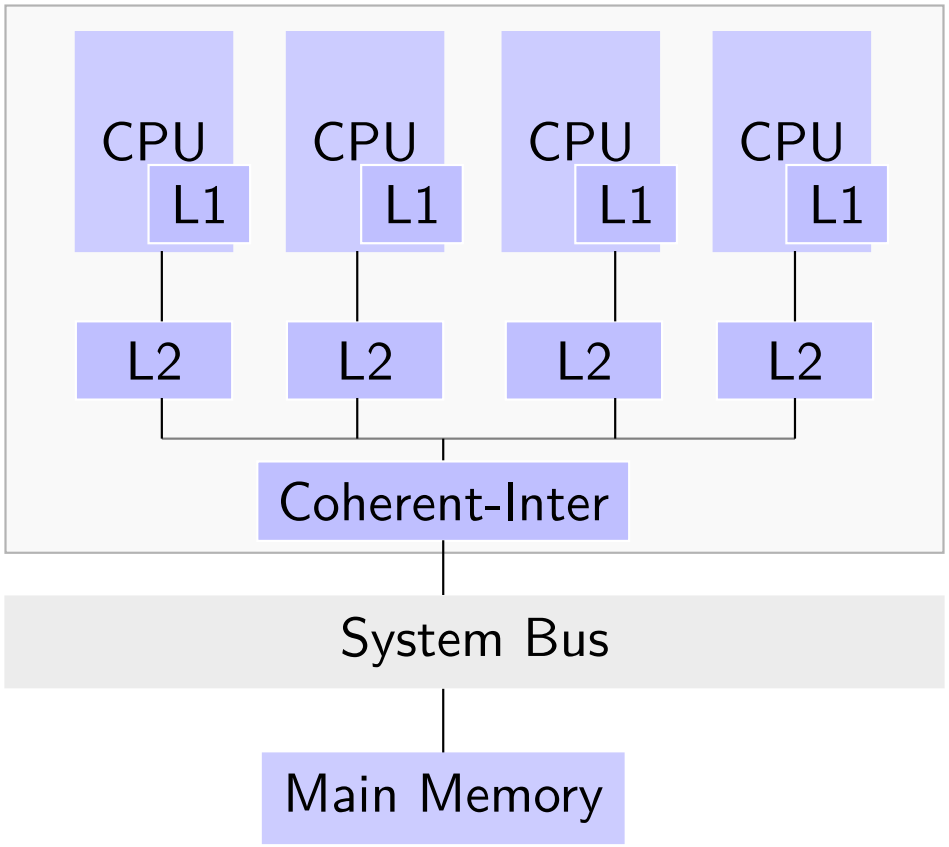
Introduction : target systems

- Real-Time constraints
- Computer vision applications : **large data sets**
- Multicore platforms

System model

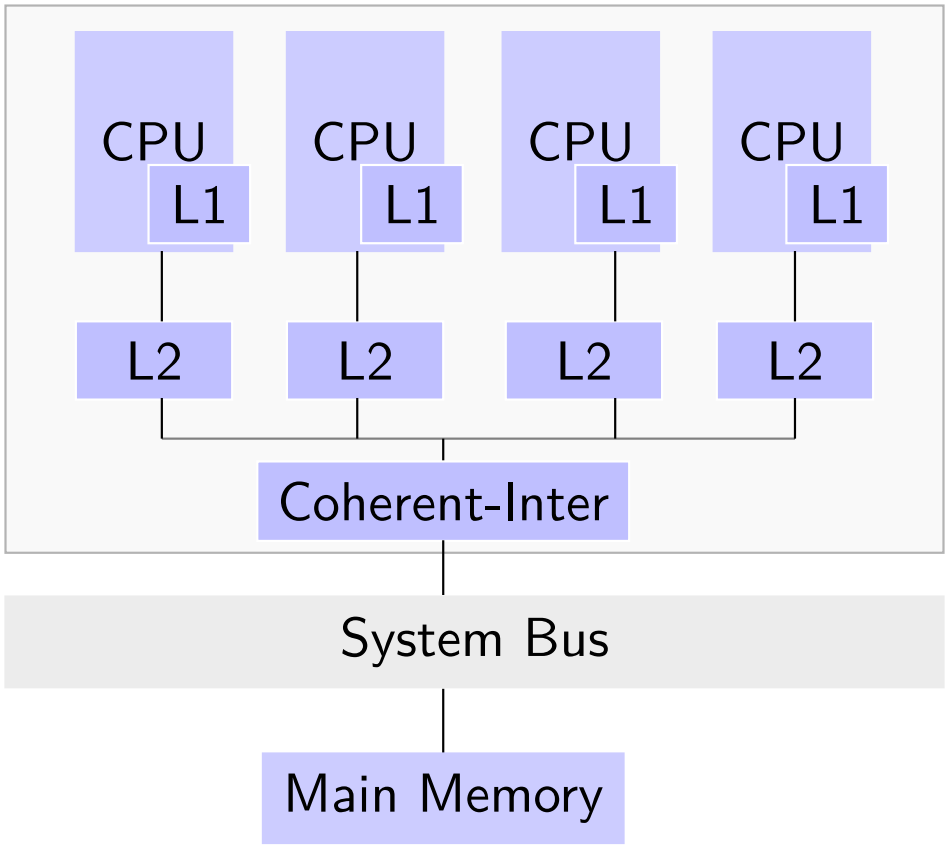


System model



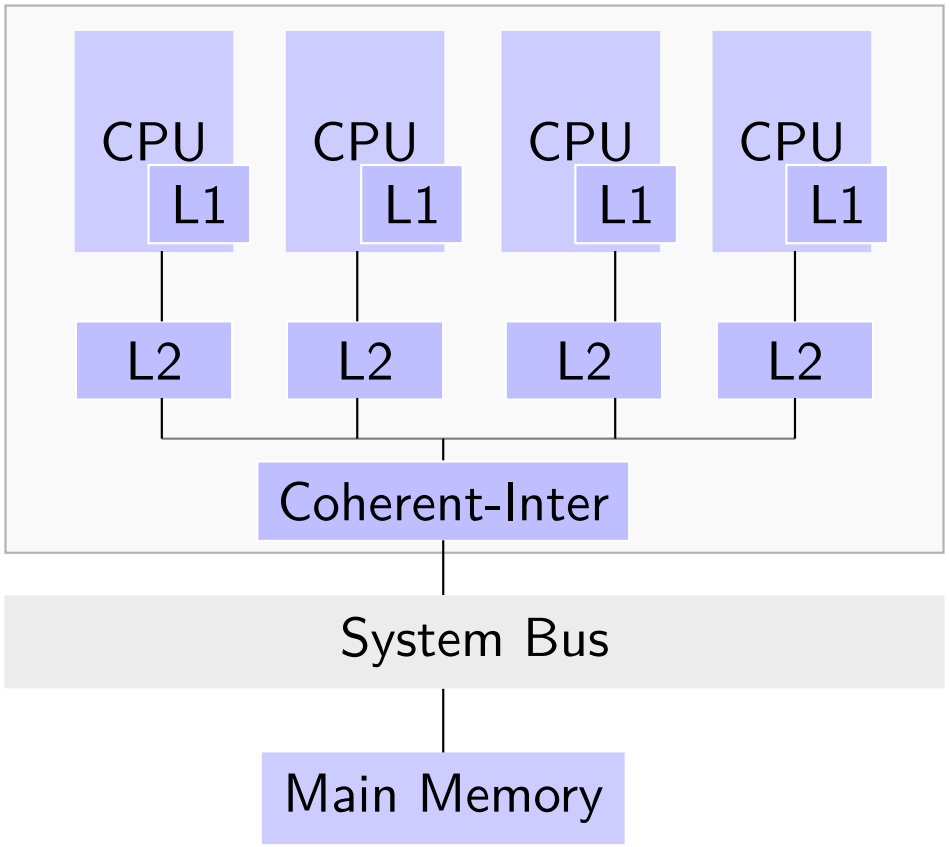
- Tasks are allocated to different cores

System model



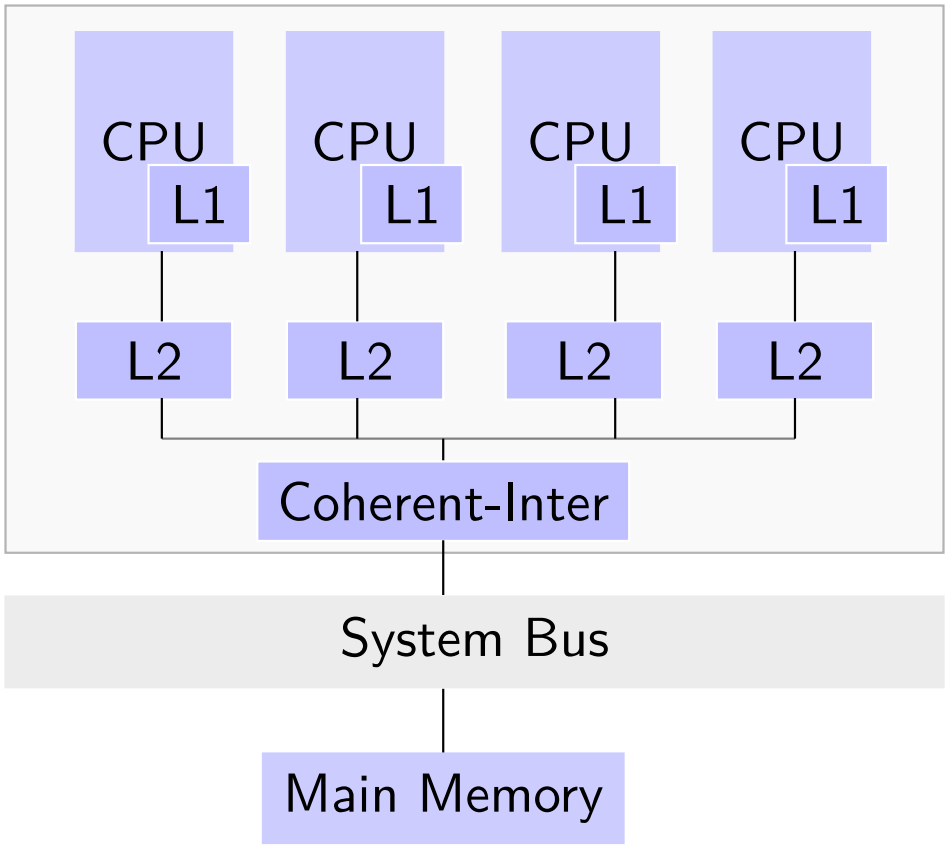
- Tasks are allocated to different cores
- Preemption is allowed between tasks in each core

System model



- Tasks are allocated to different cores
- Preemption is allowed between tasks in each core
- Large runtime-overhead of all tasks :

System model



- Tasks are allocated to different cores
- Preemption is allowed between tasks in each core
- Large runtime-overhead of all tasks : **Preemption**

Preemption in fully-preemptive scheduling

- Traditionally, the cost of preemption has been neglected in real applications

Preemption in fully-preemptive scheduling

- Traditionally, the cost of preemption has been neglected in real applications
- The preemption-related overheads are of significant importance in real-time scheduling : even up to **more than 30%** of the task's worst case execution time (Rodolfo Pellizoni et al, RTSS'08)

Preemption in fully-preemptive scheduling

- Traditionally, the cost of preemption has been neglected in real applications
- The preemption-related overheads are of significant importance in real-time scheduling : even up to **more than 30%** of the task's worst case execution time (Rodolfo Pellizoni et al, RTSS'08)
- Preemption includes :

Preemption in fully-preemptive scheduling

- Traditionally, the cost of preemption has been neglected in real applications
- The preemption-related overheads are of significant importance in real-time scheduling : even up to **more than 30%** of the task's worst case execution time (Rodolfo Pellizoni et al, RTSS'08)
- Preemption includes :
  - A large cache-related preemption delays (CRPD) :

Preemption in fully-preemptive scheduling

- Traditionally, the cost of preemption has been neglected in real applications
- The preemption-related overheads are of significant importance in real-time scheduling : even up to **more than 30%** of the task's worst case execution time (Rodolfo Pellizoni et al, RTSS'08)
- Preemption includes :
  - A large cache-related preemption delays (CRPD) :
    - The preemption point (PP) where the preemption occurs,

Preemption in fully-preemptive scheduling

- Traditionally, the cost of preemption has been neglected in real applications
- The preemption-related overheads are of significant importance in real-time scheduling : even up to **more than 30%** of the task's worst case execution time (Rodolfo Pellizoni et al, RTSS'08)
- Preemption includes :
  - A large cache-related preemption delays (CRPD) :
    - The preemption point (PP) where the preemption occurs,
    - The cache blocks used until PP, that may be reused by the preempted task,

Preemption in fully-preemptive scheduling

- Traditionally, the cost of preemption has been neglected in real applications
- The preemption-related overheads are of significant importance in real-time scheduling : even up to **more than 30%** of the task's worst case execution time (Rodolfo Pellizoni et al, RTSS'08)
- Preemption includes :
  - A large cache-related preemption delays (CRPD) :
    - The preemption point (PP) where the preemption occurs,
    - The cache blocks used until PP, that may be reused by the preempted task,
    - The evicting cache blocks of the preempting task (Sebastian Altmeyer et al, JSA, 2011)

Preemption in fully-preemptive scheduling

- Traditionally, the cost of preemption has been neglected in real applications
- The preemption-related overheads are of significant importance in real-time scheduling : even up to **more than 30%** of the task's worst case execution time (Rodolfo Pellizoni et al, RTSS'08)
- Preemption includes :
  - A large cache-related preemption delays (CRPD) :
    - The preemption point (PP) where the preemption occurs,
    - The cache blocks used until PP, that may be reused by the preempted task,
    - The evicting cache blocks of the preempting task (Sebastian Altmeyer et al, JSA, 2011)
  - Other run-time costs related to all possible schedules :

Preemption in fully-preemptive scheduling

- Traditionally, the cost of preemption has been neglected in real applications
- The preemption-related overheads are of significant importance in real-time scheduling : even up to **more than 30%** of the task's worst case execution time (Rodolfo Pellizoni et al, RTSS'08)
- Preemption includes :
  - A large cache-related preemption delays (CRPD) :
    - The preemption point (PP) where the preemption occurs,
    - The cache blocks used until PP, that may be reused by the preempted task,
    - The evicting cache blocks of the preempting task (Sebastian Altmeyer et al, JSA, 2011)
  - Other run-time costs related to all possible schedules :
    - Scheduler cost, Pipeline cost

Reducing the preemption-overhead

- Disabling preemption (non-preemptive scheduling) :

  (–) CRPD + scheduling overhead ☺
  (+) Large blocking time ☹

Reducing the preemption-overhead

- Disabling preemption (non-preemptive scheduling) :

  (–) CRPD + scheduling overhead ☺
  (+) Large blocking time ☹

- Hybrid preemption models : Deferred preemption approach 😉

The problem to solve

We have :

- A set of deferred preemption tasks
- An homogeneous multi-core platform

The problem to solve

We have :

- A set of deferred preemption tasks
- An homogeneous multi-core platform

Objectives

- Allocate the tasks to different cores
- Reduce the preemption overhead across all cores in the platform

The problem to solve

We have :

- A set of deferred preemption tasks
- An homogeneous multi-core platform

Objectives

- Allocate the tasks to different cores
- Reduce the preemption overhead across all cores in the platform

Constraints

- All deadlines must be respected

Table of content

1 Context and motivation

2 Deferred preemption models for single-core processor

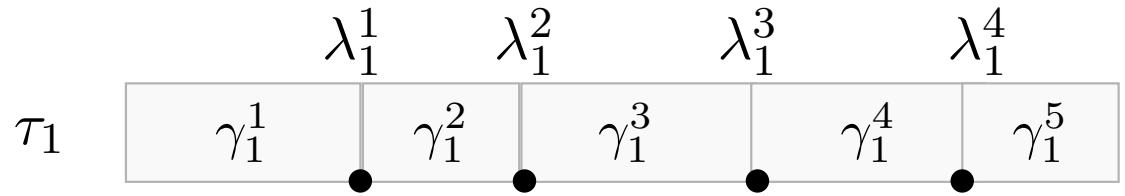
3 Deferred preemption task allocation onto multi-core platform

4 Experiments results

5 Conclusion and future work

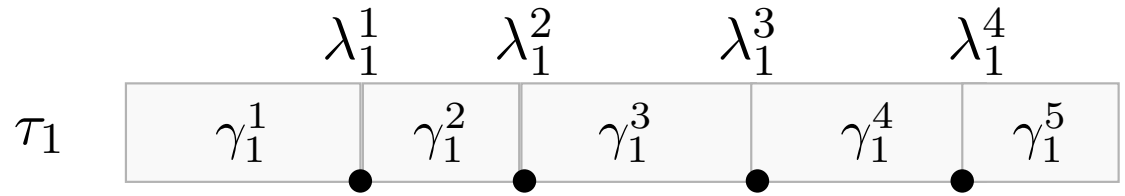
Deferred preemption task model

- Each task is divided into a sequence of statically defined non-preemptive chunks separated by np_i pre-defined preemption points



Deferred preemption task model

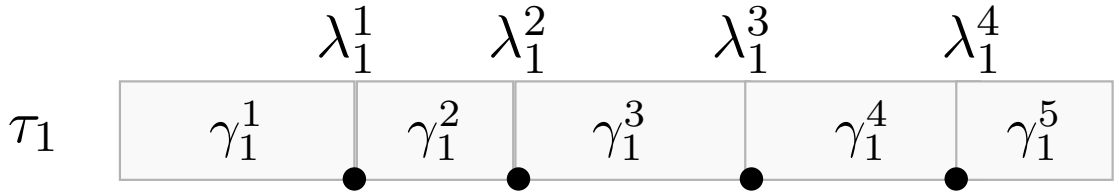
- Each task is divided into a sequence of statically defined non-preemptive chunks separated by np_i pre-defined preemption points



- The task may only be preempted at the pre-defined preemption points λ_i^j

Deferred preemption task model

- Each task is divided into a sequence of statically defined non-preemptive chunks separated by np_i pre-defined preemption points



- The task may only be preempted at the pre-defined preemption points λ_i^j
- Advantages :
  - Predict the exact number of preemption
  - Compute tight upper bounds on the preemption cost
  - Obtain less pessimistic WCET bounds for each task

Maximum non-preemptive execution-time

- Deferred preemption versions :
  - **Floating preemption-point model**
  - **Fixed preemption-point model**

Maximum non-preemptive execution-time

- Deferred preemption versions :
  - **Floating preemption-point model**
  - **Fixed preemption-point model**
- Compute the maximum length of the non-preemptive regions $\mathbf{Q_i}$ for each task in the system :
  - Sanjoy Baruah (ECRTS'05) for EDF case

Maximum non-preemptive execution-time

- Deferred preemption versions :
  - **Floating preemption-point model**
  - **Fixed preemption-point model**
- Compute the maximum length of the non-preemptive regions $\mathbf{Q_i}$ for each task in the system :
  - Sanjoy Baruah (ECRTS'05) for EDF case
- Tasks ordered for increasing relative deadlines :

$$SI_i^{EDF} = \min(t - \sum_{j=1}^{n} dbf(\mathcal{T}, t)), \qquad \text{with : } D_i < t \leq D_{i+1}$$

Selection of Effective Preemption points

- Select a subset $\overline{\Lambda}_i$ of preemption points from the pre-defined preemption point set (Marko Bertogna et al, ECRTS'11) :

$$\forall \tau_i \in \mathcal{T}, C(NPR_i^{max}) \leq Q_i$$

Selection of Effective Preemption points

- Select a subset $\overline{\Lambda}_i$ of preemption points from the pre-defined preemption point set (Marko Bertogna et al, ECRTS'11) :

$$\forall \tau_i \in \mathcal{T}, C(NPR_i^{max}) \leq Q_i$$

- Include the preemption cost of any **Effective preemption point (EPP)** into *WCET* :

$$C(\tau_i, \overline{\Lambda}_i) = \sum_{j=1}^{np_i+1} C(\gamma_i^j) + \sum_{\lambda \in \overline{\Lambda}_i} C(\tau_i, \lambda) \qquad (1)$$

Table of content

1 Context and motivation

2 Deferred preemption models for single-core processor

3 Deferred preemption task allocation onto multi-core platform

4 Experiments results

5 Conclusion and future work

Global vs Partitioned ??

Partitioned :

- Each core has its own ready-queue
- Final allocation of tasks onto cores

- Easy to implement : No task migration
- A good exploitation of the platform

Global :

- One ready-queue shared between all cores
- The m highest priority tasks are scheduled

- Hard to implement
- A large runtime overhead due to the high number of migrations

Global vs Partitioned ??

Partitioned :

- Each core has its own ready-queue
- Final allocation of tasks onto cores

- Easy to implement : No task migration
- A good exploitation of the platform

Global :

- One ready-queue shared between all cores
- The m highest priority tasks are scheduled

- Hard to implement
- A large runtime overhead due to the high number of migrations

Enumerative algorithm

- Generate all possible allocations for each task to different cores in the platform

Enumerative algorithm

- Generate all possible allocations for each task to different cores in the platform

Sort by deadline : $\tau_3 > \tau_2 > \tau_1$

$Allocation_8$	$\{\emptyset\}\|\{\tau_1, \tau_3, \tau_2\}$
$Allocation_7$	$\{\tau_1, \tau_3, \tau_2\}\|\{\emptyset\}$
$Allocation_6$	$\{\tau_1, \tau_2\}\|\{\tau_3\}$
$Allocation_5$	$\{\tau_1, \tau_3\}\|\{\tau_2\}$
$Allocation_4$	$\{\tau_3, \tau_2\}\|\{\tau_1\}$
$Allocation_3$	$\{\tau_3\}\|\{\tau_1, \tau_2\}$
$Allocation_2$	$\{\tau_2\}\|\{\tau_1, \tau_3\}$
$Allocation_1$	$\{\tau_1\}\|\{\tau_3, \tau_2\}$

Enumerative algorithm

- Generate all possible allocations for each task to different cores in the platform

Sort by deadline : $\tau_3 > \tau_2 > \tau_1$

- For each allocation :
  - Compute the value of $\mathbf{Q_i}$ and select the **EPP only for the studied task** τ_1

Allocation$_8$	$\{\emptyset\}\|\{\tau_1, \tau_3, \tau_2\}$
Allocation$_7$	$\{\tau_1, \tau_3, \tau_2\}\|\{\emptyset\}$
Allocation$_6$	$\{\tau_1, \tau_2\}\|\{\tau_3\}$
Allocation$_5$	$\{\tau_1, \tau_3\}\|\{\tau_2\}$
Allocation$_4$	$\{\tau_3, \tau_2\}\|\{\tau_1\}$
Allocation$_3$	$\{\tau_3\}\|\{\tau_1, \tau_2\}$
Allocation$_2$	$\{\tau_2\}\|\{\tau_1, \tau_3\}$
Allocation$_1$	$\{\tau_1\}\|\{\tau_3, \tau_2\}$

Enumerative algorithm

- Generate all possible allocations for each task to different cores in the platform

Sort by deadline : $\tau_3 > \tau_2 > \tau_1$

- For each allocation :
  - Compute the value of $\mathbf{Q_i}$ and select the **EPP only for the studied task** τ_1
  - Eliminate all **non schedulable** branches/allocations

Allocation$_8$	$\{\emptyset\}\|\{\tau_1, \tau_3, \tau_2\}$
Allocation$_7$	$\{\tau_1, \tau_3, \tau_2\}\|\{\emptyset\}$
Allocation$_6$	$\{\tau_1, \tau_2\}\|\{\tau_3\}$
Allocation$_5$	$\{\tau_1, \tau_3\}\|\{\tau_2\}$
Allocation$_4$	$\{\tau_3, \tau_2\}\|\{\tau_1\}$
Allocation$_3$	$\{\tau_3\}\|\{\tau_1, \tau_2\}$
Allocation$_2$	$\{\tau_2\}\|\{\tau_1, \tau_3\}$
Allocation$_1$	$\{\tau_1\}\|\{\tau_3, \tau_2\}$

Enumerative algorithm

- Generate all possible allocations for each task to different cores in the platform

Sort by deadline : $\tau_3 > \tau_2 > \tau_1$

- For each allocation :
  - Compute the value of $\mathbf{Q_i}$ and select the **EPP only for the studied task** τ_1
  - Eliminate all **non schedulable** branches/allocations
- Explore the space solution by solution

Allocation$_8$	$\{\emptyset\}\|\{\tau_1, \tau_3, \tau_2\}$
Allocation$_7$	$\{\tau_1, \tau_3, \tau_2\}\|\{\emptyset\}$
Allocation$_6$	$\{\tau_1, \tau_2\}\|\{\tau_3\}$
Allocation$_5$	$\{\tau_1, \tau_3\}\|\{\tau_2\}$
Allocation$_4$	$\{\tau_3, \tau_2\}\|\{\tau_1\}$
Allocation$_3$	$\{\tau_3\}\|\{\tau_1, \tau_2\}$
Allocation$_2$	$\{\tau_2\}\|\{\tau_1, \tau_3\}$
Allocation$_1$	$\{\tau_1\}\|\{\tau_3, \tau_2\}$

Enumerative algorithm

- Generate all possible allocations for each task to different cores in the platform

Sort by deadline : $\tau_3 > \tau_2 > \tau_1$

- For each allocation :
  - Compute the value of $\mathbf{Q_i}$ and select the **EPP only for the studied task** τ_1
  - Eliminate all **non schedulable** branches/allocations
- Explore the space solution by solution $\Rightarrow$ **exponential runtime**

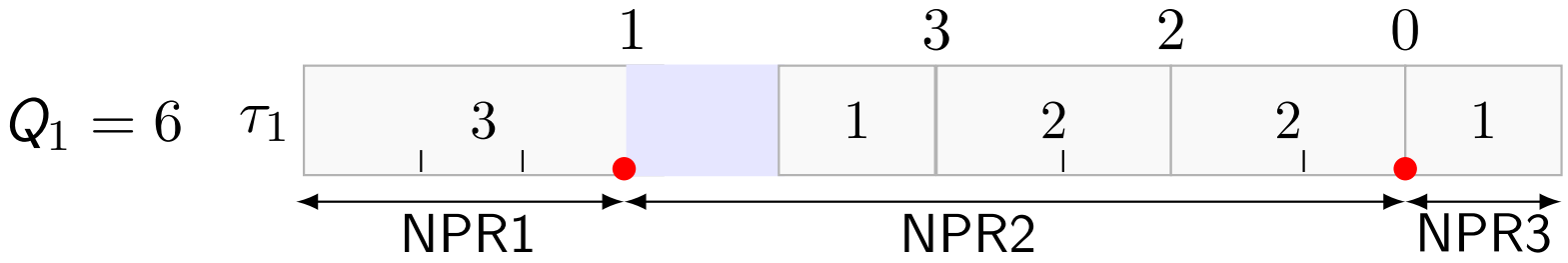
Allocation$_8$	$\{\emptyset\}\|\{\tau_1, \tau_3, \tau_2\}$
Allocation$_7$	$\{\tau_1, \tau_3, \tau_2\}\|\{\emptyset\}$
Allocation$_6$	$\{\tau_1, \tau_2\}\|\{\tau_3\}$
Allocation$_5$	$\{\tau_1, \tau_3\}\|\{\tau_2\}$
Allocation$_4$	$\{\tau_3, \tau_2\}\|\{\tau_1\}$
Allocation$_3$	$\{\tau_3\}\|\{\tau_1, \tau_2\}$
Allocation$_2$	$\{\tau_2\}\|\{\tau_1, \tau_3\}$
Allocation$_1$	$\{\tau_1\}\|\{\tau_3, \tau_2\}$

Observations

- The value of the non-preemptive parameter Q_i depends upon the temporal caracteristics of other tasks

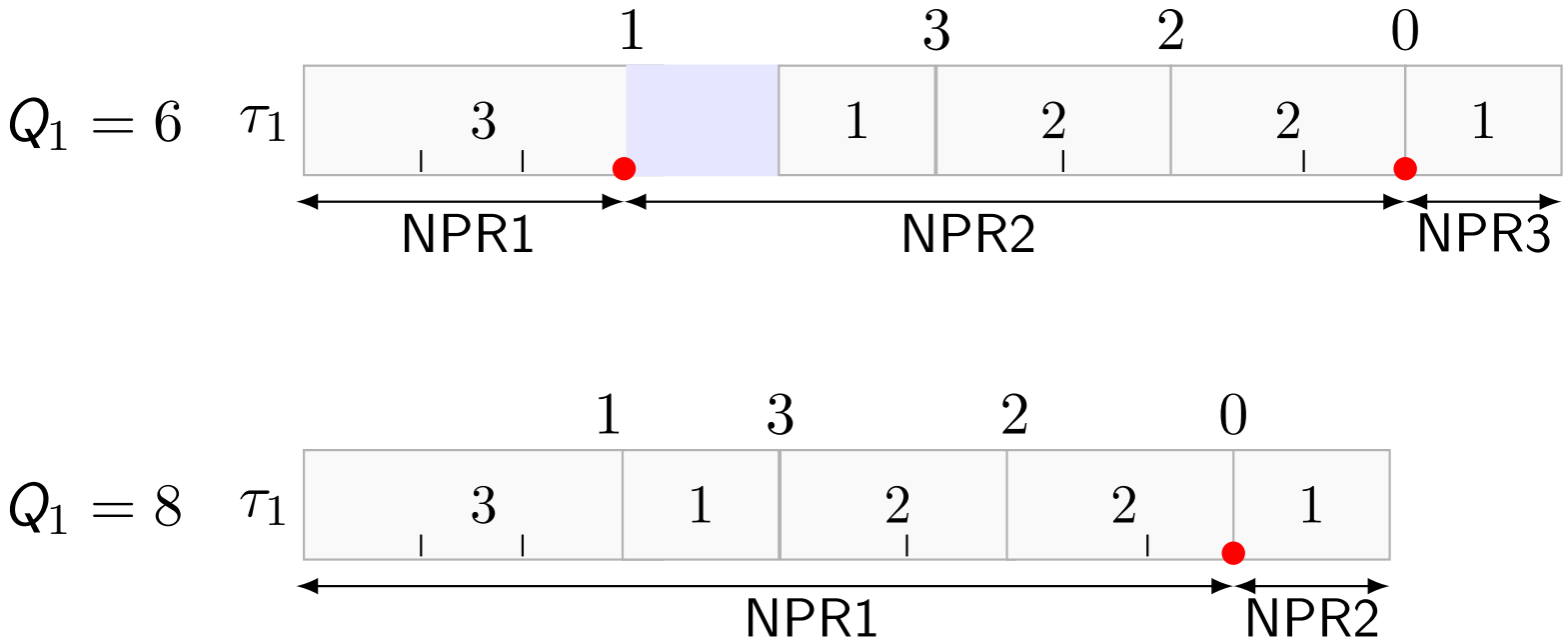
Observations

- The value of the non-preemptive parameter Q_i depends upon the temporal caracteristics of other tasks



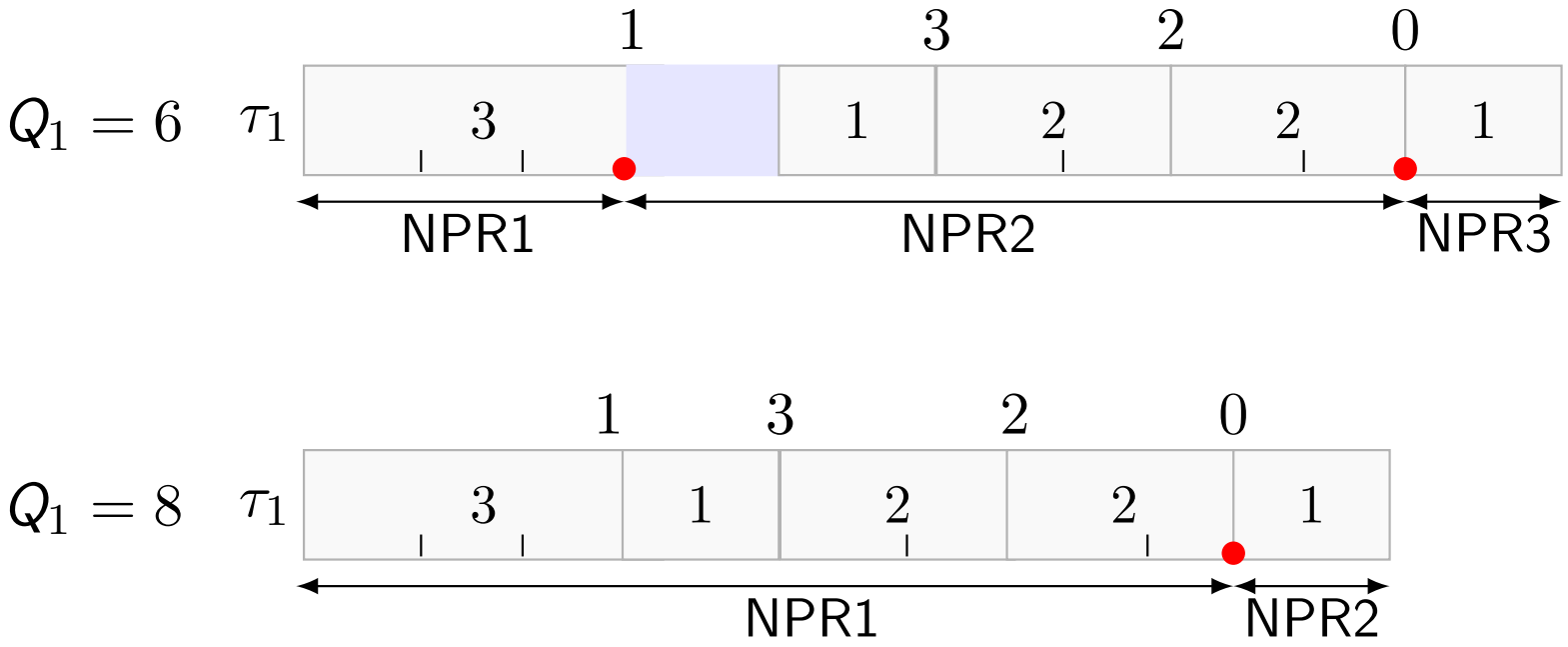
Observations

- The value of the non-preemptive parameter Q_i depends upon the temporal caracteristics of other tasks



Observations

- The value of the non-preemptive parameter Q_i depends upon the temporal caracteristics of other tasks



- A more relaxed non-preemptive interval leads to a lower preemption cost (optimality of EPP selection algorithm)

Branch and bound algorithm

- Allocate **the shortest relative deadline task** τ_i to every core in the platform (depth-first, breadth-first)

Branch and bound algorithm

- Allocate **the shortest relative deadline task** τ_i to every core in the platform (depth-first, breadth-first)
- All allocations having at least one free core are dominated ($n > m$)

Branch and bound algorithm

- Allocate **the shortest relative deadline task** τ_i to every core in the platform (depth-first, breadth-first)
- All allocations having at least one free core are dominated ($n > m$)
- For each allocation :

Branch and bound algorithm

- Allocate **the shortest relative deadline task** τ_i to every core in the platform (depth-first, breadth-first)
- All allocations having at least one free core are dominated ($n > m$)
- For each allocation :
  - Compute the maximum lenght of the non-preemptive region Q_i for the studied task τ_i

Branch and bound algorithm

- Allocate **the shortest relative deadline task** τ_i to every core in the platform (depth-first, breadth-first)
- All allocations having at least one free core are dominated ($n > m$)
- For each allocation :
  - Compute the maximum lenght of the non-preemptive region Q_i for the studied task τ_i
  - Use the dominance relation between allocations :

$$\forall i, Q(\tau_i^A) \leq Q(\tau_i^{A'}) \;\; then, \;\; cost_A > cost_{A'}$$

Branch and bound algorithm

- Allocate **the shortest relative deadline task** τ_i to every core in the platform (depth-first, breadth-first)
- All allocations having at least one free core are dominated ($n > m$)
- For each allocation :
  - Compute the maximum lenght of the non-preemptive region Q_i for the studied task τ_i
  - Use the dominance relation between allocations :

$$\forall i, Q(\tau_i^A) \leq Q(\tau_i^{A'}) \;\; then, \;\; cost_A > cost_{A'}$$

  - Otherwise,
    - Select the **EPP** only for the studied task τ_i
    - Eliminate all **non schedulable/not optimal** branches/allocations

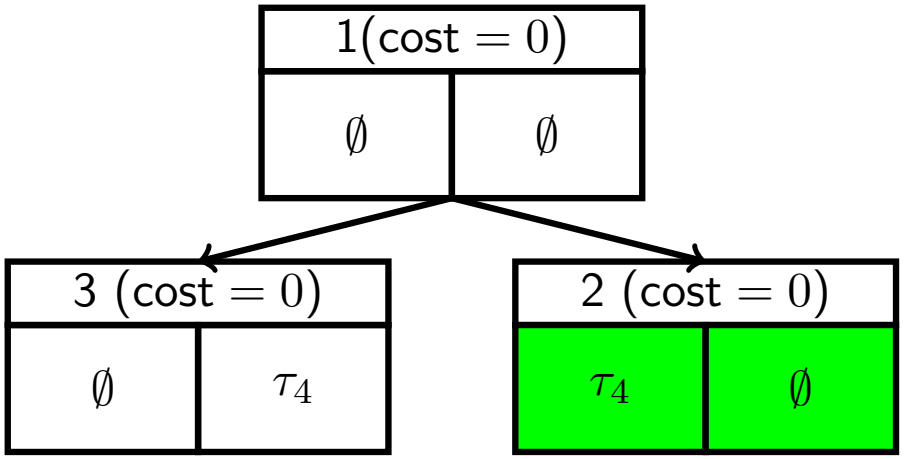
Example

- Platform with 2 cores and 4 real-time tasks

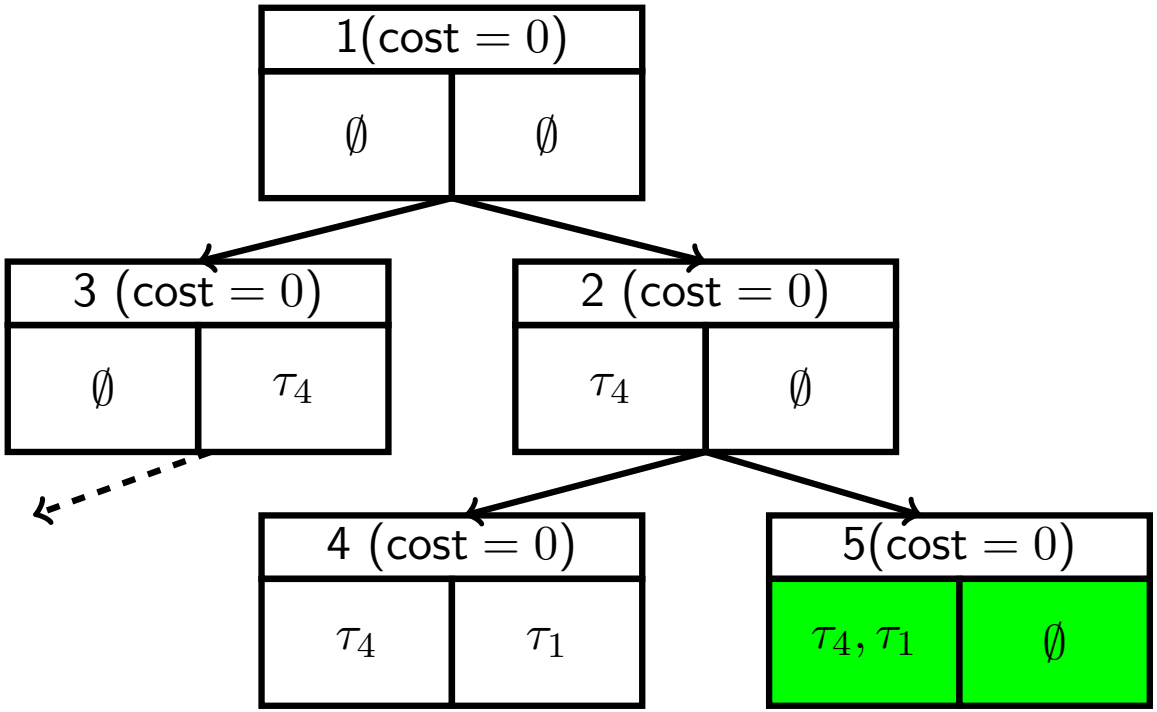
task	D_i	T_i	Γ_i	Λ_i
τ_1	1413	1500	{212,171,344,66,249}	{ 0,46,78,14,47 }
τ_2	5673	6000	{17,54,490,101,418,74}	{0,14,94,21,74,13}
τ_3	1498	1500	{146,347,136,37,121}	{0,90,32,7,19}
τ_4	1277	1500	{17,31,43,3,24,6}	{ 0,6,8,0,3,0 }

- $\tau_4 > \tau_1 > \tau_3 > \tau_2$

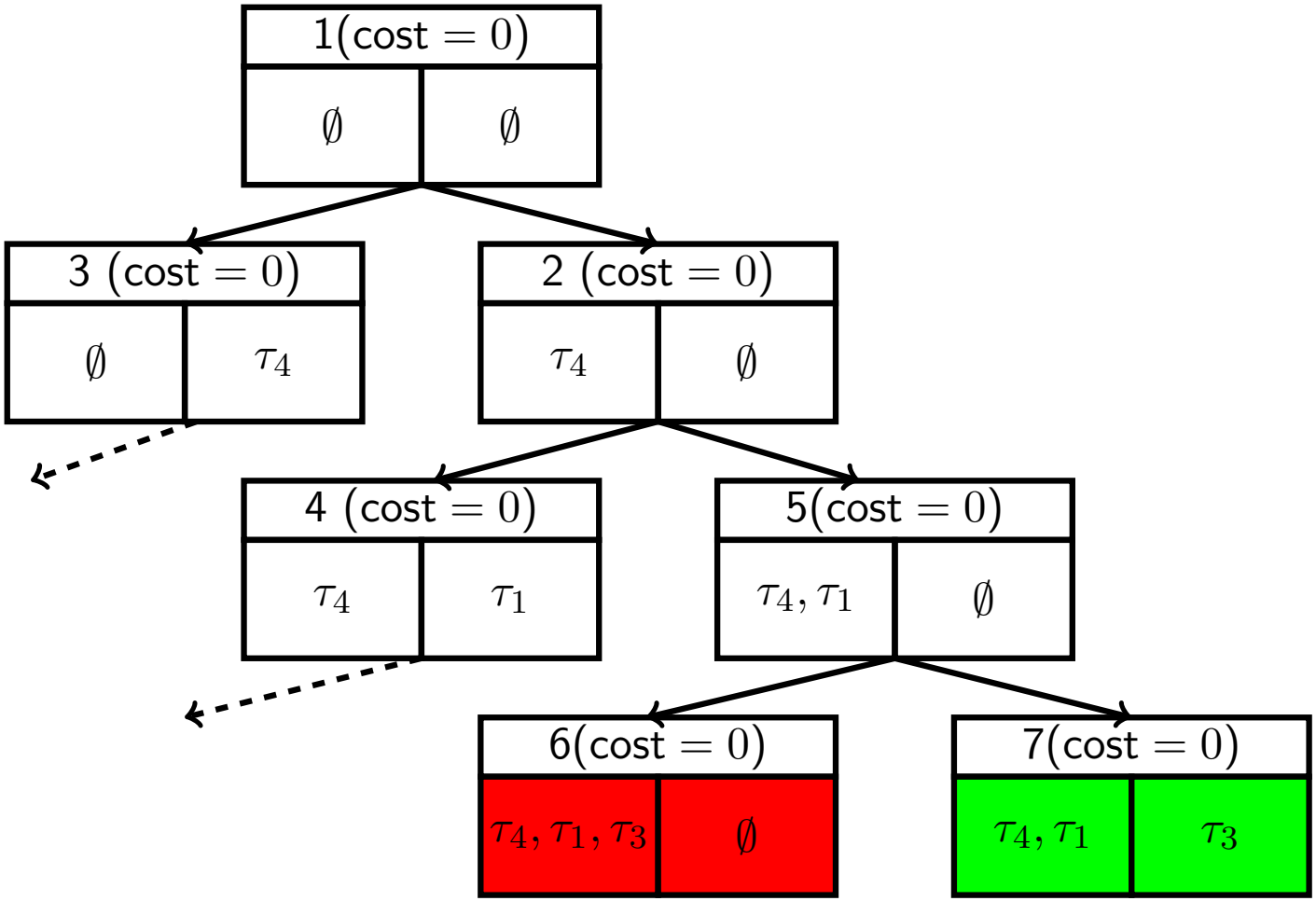
Example



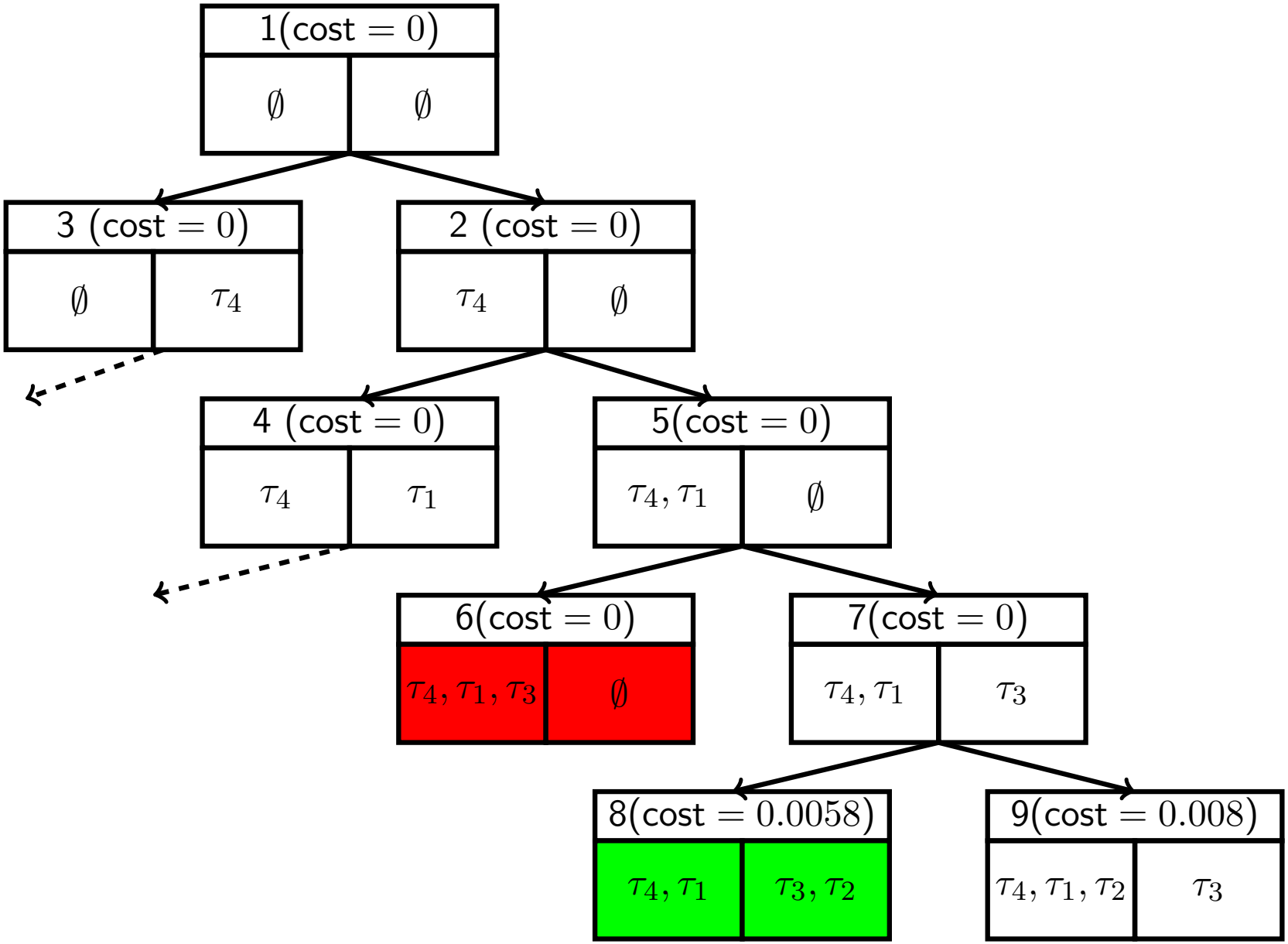
Example


1(cost = 0)
∅
∅
3 (cost = 0)
∅
τ_4
2 (cost = 0)
τ_4
∅
4 (cost = 0)
τ_4
τ_1
5(cost = 0)
τ_4, τ_1
∅

Example


1(cost = 0)
∅
∅
3 (cost = 0)
∅
τ_4
2 (cost = 0)
τ_4
∅
4 (cost = 0)
τ_4
τ_1
5(cost = 0)
τ_4, τ_1
∅
6(cost = 0)
τ_4, τ_1, τ_3
∅
7(cost = 0)
τ_4, τ_1
τ_3

Example


1(cost = 0)
∅
∅
3 (cost = 0)
∅
τ_4
2 (cost = 0)
τ_4
∅
4 (cost = 0)
τ_4
τ_1
5(cost = 0)
τ_4, τ_1
∅
6(cost = 0)
τ_4, τ_1, τ_3
∅
7(cost = 0)
τ_4, τ_1
τ_3
8(cost = 0.0058)
τ_4, τ_1
τ_3, τ_2
9(cost = 0.008)
τ_4, τ_1, τ_2
τ_3

The optimality of the branch and bound algorithm

- The enumeration algorithm explores all schedulable allocations and selects the optimal one

The optimality of the branch and bound algorithm

- The enumeration algorithm explores all schedulable allocations and selects the optimal one
- The branch and bound algorithm is optimal :

The optimality of the branch and bound algorithm

- The enumeration algorithm explores all schedulable allocations and selects the optimal one
- The branch and bound algorithm is optimal :
  - Tasks can only be added to cores

The optimality of the branch and bound algorithm

- The enumeration algorithm explores all schedulable allocations and selects the optimal one
- The branch and bound algorithm is optimal :
  - Tasks can only be added to cores $\Rightarrow$ **the preemption cost can only increase**

The optimality of the branch and bound algorithm

- The enumeration algorithm explores all schedulable allocations and selects the optimal one
- The branch and bound algorithm is optimal :
  - Tasks can only be added to cores $\Rightarrow$ **the preemption cost can only increase**
  - All branches $\mathbf{S}(\cdot)$ generated by the algorithm having a preemption cost **greater** than the **lower** bound are **dominated** by the best known solution

The optimality of the branch and bound algorithm

- The enumeration algorithm explores all schedulable allocations and selects the optimal one
- The branch and bound algorithm is optimal :
  - Tasks can only be added to cores $\Rightarrow$ **the preemption cost can only increase**
  - All branches $\mathbf{S}(\cdot)$ generated by the algorithm having a preemption cost **greater** than the **lower** bound are **dominated** by the best known solution
  - The algorithm preserves the optimal solution at each branch level

The optimality of the branch and bound algorithm

- The enumeration algorithm explores all schedulable allocations and selects the optimal one
- The branch and bound algorithm is optimal :
  - Tasks can only be added to cores $\Rightarrow$ **the preemption cost can only increase**
  - All branches $\mathbf{S}(\cdot)$ generated by the algorithm having a preemption cost **greater** than the **lower** bound are **dominated** by the best known solution
  - The algorithm preserves the optimal solution at each branch level
  - The algorithm has an exponential execution time in the worst case

Allocation heuristics

- Classical bin-packing heuristics : First-Fit (FF), Best-Fit (BF), Worst-Fit (WF)

Allocation heuristics

- Classical bin-packing heuristics : First-Fit (FF), Best-Fit (BF), Worst-Fit (WF)
- BF and WF sort the cores by capacity

Allocation heuristics

- Classical bin-packing heuristics : First-Fit (FF), Best-Fit (BF), Worst-Fit (WF)
- BF and WF sort the cores by capacity
- Sort tasks by deadline, density or laxity :

Allocation heuristics

- Classical bin-packing heuristics : First-Fit (FF), Best-Fit (BF), Worst-Fit (WF)
- BF and WF sort the cores by capacity
- Sort tasks by deadline, density or laxity :
  - **Deadline** then, compute the value of $\mathbf{Q_i}$ and select the **EPP** for the shortest relative deadline task

Allocation heuristics

- Classical bin-packing heuristics : First-Fit (FF), Best-Fit (BF), Worst-Fit (WF)
- BF and WF sort the cores by capacity
- Sort tasks by deadline, density or laxity :
  - **Deadline** then, compute the value of $\mathbf{Q_i}$ and select the **EPP** for the shortest relative deadline task
  - **Density, or laxity** then, compute the value of $\mathbf{Q_i}$ and select the **EPP** for all tasks

Table of content

1 Context and motivation

2 Deferred preemption models for single-core processor

3 Deferred preemption task allocation onto multi-core platform

4 Experiments results

5 Conclusion and future work

Protocol

In all the graphs :

- 3 identical processing units

Protocol

In all the graphs :

- 3 identical processing units
- 24 tasks :
  - 8 to 15 basic blocs
  - The block preemption cost is calculated using a random percentage between 0.1 and 0.2 of the block utilization
  - Minimum period (T_i) is 120 and the maximum is 120,000 by step of 500
  - Deadline (D_i) is generated randomly between $[0.75 \cdot T_i , T_i]$

Protocol

In all the graphs :

- 3 identical processing units
- 24 tasks :
  - 8 to 15 basic blocs
  - The block preemption cost is calculated using a random percentage between 0.1 and 0.2 of the block utilization
  - Minimum period (T_i) is 120 and the maximum is 120,000 by step of 500
  - Deadline (D_i) is generated randomly between $[0.75 \cdot T_i , T_i]$
- Utilization : 0.25 to 3.75 with a step of 0.25

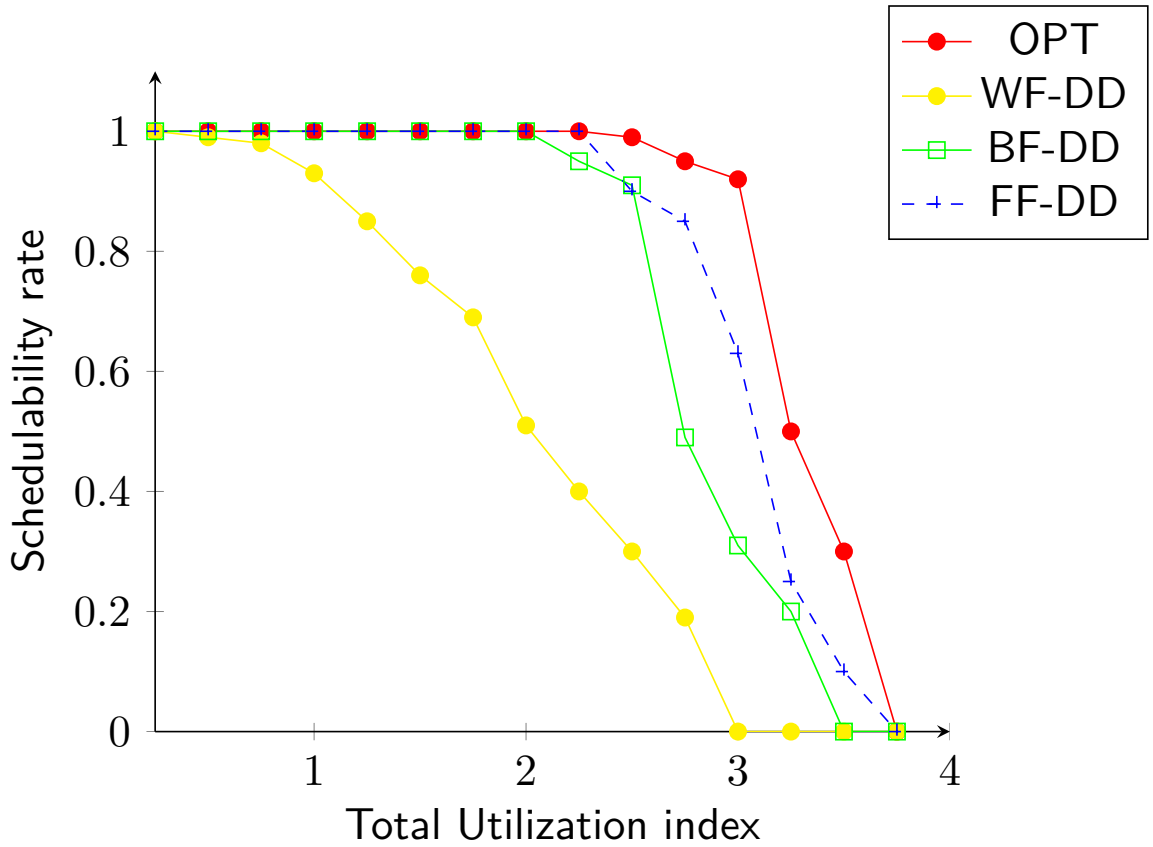
Protocol

In all the graphs :

- 3 identical processing units
- 24 tasks :
  - 8 to 15 basic blocs
  - The block preemption cost is calculated using a random percentage between 0.1 and 0.2 of the block utilization
  - Minimum period (T_i) is 120 and the maximum is 120,000 by step of 500
  - Deadline (D_i) is generated randomly between $[0.75 \cdot T_i , T_i]$
- Utilization : 0.25 to 3.75 with a step of 0.25
- Each point is the average value of 100 executions

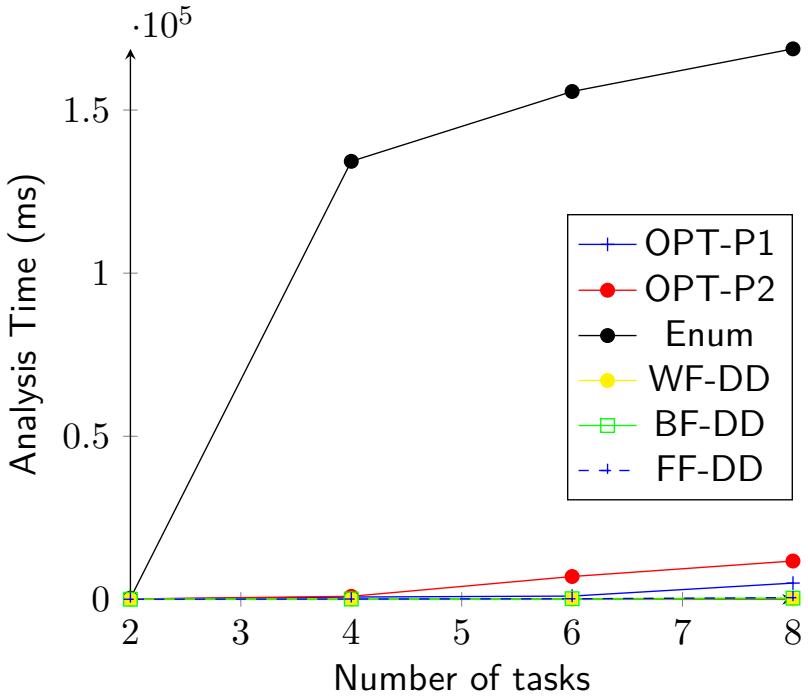
Schedulability for optimal solutions VS bin-packing heuristics

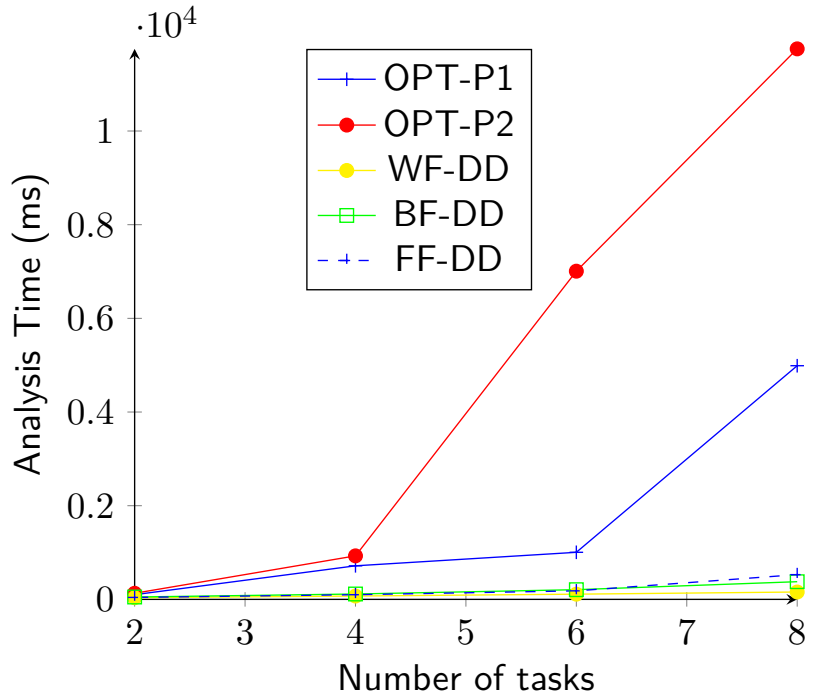
- Tasks are sorted according to their relative **deadlines** in **non-decreasing** order



The analysis time as a function of number of tasks

- Tasks are sorted according to their relative **deadlines** in **non-decreasing** order
- OPT-P1 : depth-first, OPT-P2 : breadth-first





Performance of BF, WF and FF using sorted tasks by density

- Tasks are sorted according to their **density** in **increasing (XF-DN-I)** order and **decreasing (XF-DN-D)** order.

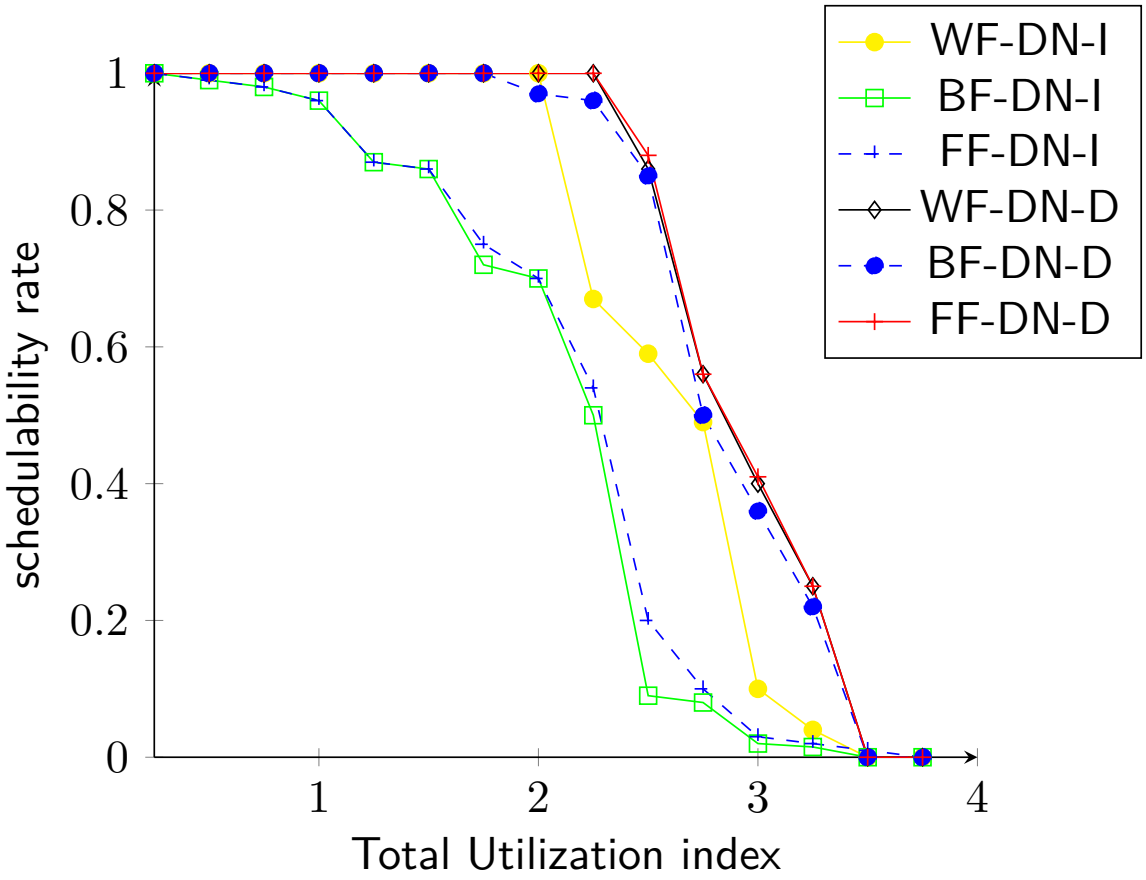


Table of content

1 Context and motivation

2 Deferred preemption models for single-core processor

3 Deferred preemption task allocation onto multi-core platform

4 Experiments results

5 Conclusion and future work

Conclusion and future work

- We proposed allocation algorithms for real-time tasks with fixed preemption points on an identical core platform :

Conclusion and future work

- We proposed allocation algorithms for real-time tasks with fixed preemption points on an identical core platform :
  - Two optimal algorithms (Enumerative and Branch and bound algorithms)
  - A set of allocation heuristics

Conclusion and future work

- We proposed allocation algorithms for real-time tasks with fixed preemption points on an identical core platform :
  - Two optimal algorithms (Enumerative and Branch and bound algorithms)
  - A set of allocation heuristics
  - Minimizing the preemption costs

Conclusion and future work

- We proposed allocation algorithms for real-time tasks with fixed preemption points on an identical core platform :
  - Two optimal algorithms (Enumerative and Branch and bound algorithms)
  - A set of allocation heuristics
  - Minimizing the preemption costs
- Extend the proposed approaches to consider dependent tasks

Thank you for your attention !