Environnement de développement d'applications multipériodiques sur plateforme multicœur

Mikel Cordovilla Mesonero

Membres du jury

Frédéric Boniol	ONERA Toulouse	Directeur de thèse
Liliana Cucu-Grosjean	INRIA Nancy-Grand Est	Examinateur
Emmanuel Grolleau	ENSMA Poitiers	Rapporteur
Pierre-Emmanuel Hladik	INSA Toulouse	Examinateur
Eric Noulard	ONERA Toulouse	Encadrant de thèse
Claire Pagetti	ONERA Toulouse	Encadrant de thèse
Olivier H. Roux	IRCCyN/École Centrale de Nantes	Rapporteur
Jean-Pierre Talpin	INRIA-Rennes	Examinateur

2 Avril 2012

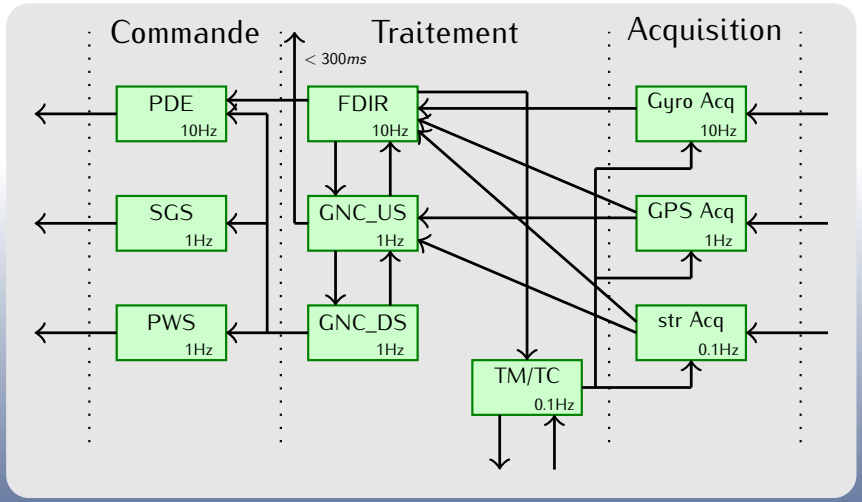
isae Institut Supérieur de l'Aéronautique et de l'Espace

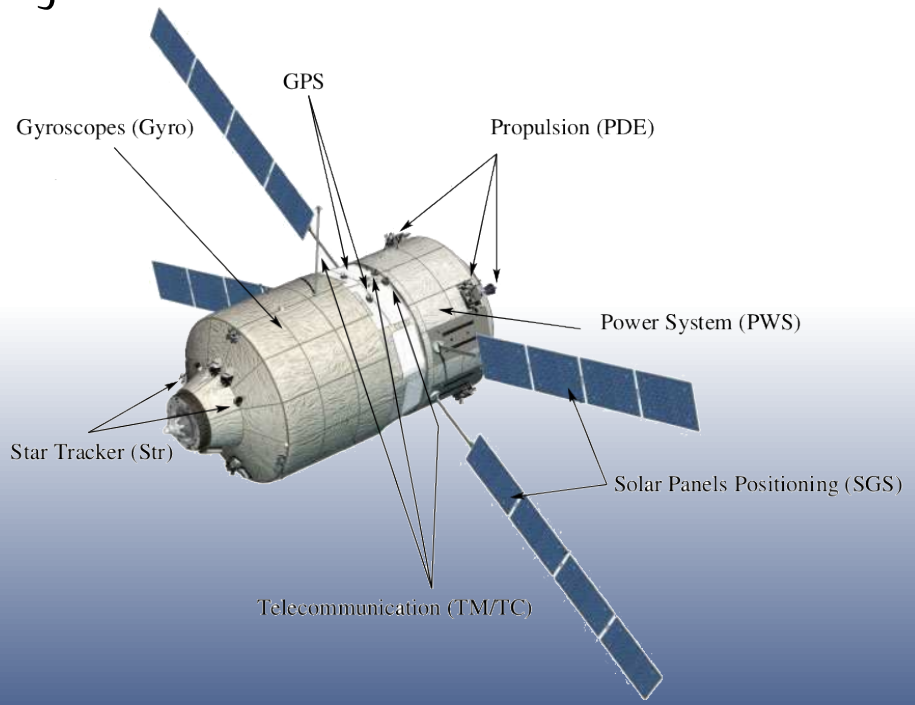
ONERA THE FRENCH AEROSPACE LAB

Université de Toulouse

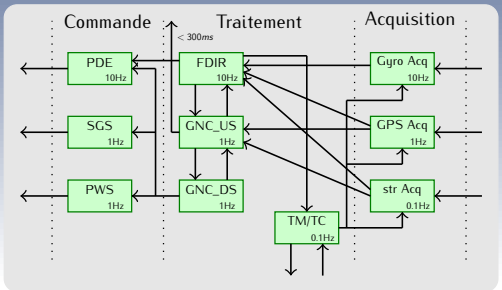
Contexte : programmation temps réel de systèmes embarqués critiques

- **Exemple** : Flight Application Software (FAS)
- **Particularités** :
  - Multipériodique
  - Patron de communication explicite
  - Contraintes de précédences généralisées
- **Objectif** : exécution "sûre" du système sur architecture multicœur





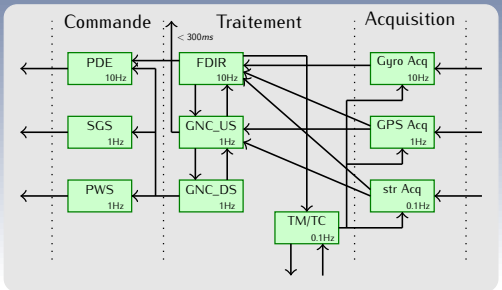
Application : de l'assemblage à l'exécution



Monoprocesseur

- Comment modélise-t-on le système ?

Application : de l'assemblage à l'exécution

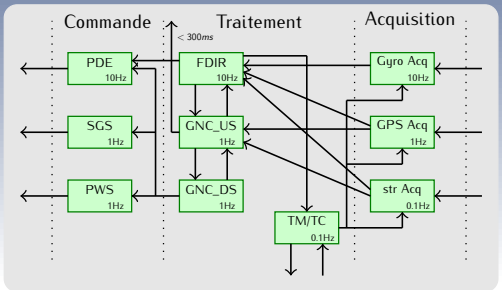


spécification PRELUDE

Monoprocesseur

- Comment modélise-t-on le système ? PRELUDE

Application : de l'assemblage à l'exécution

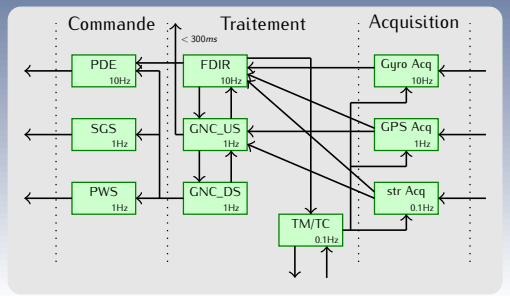


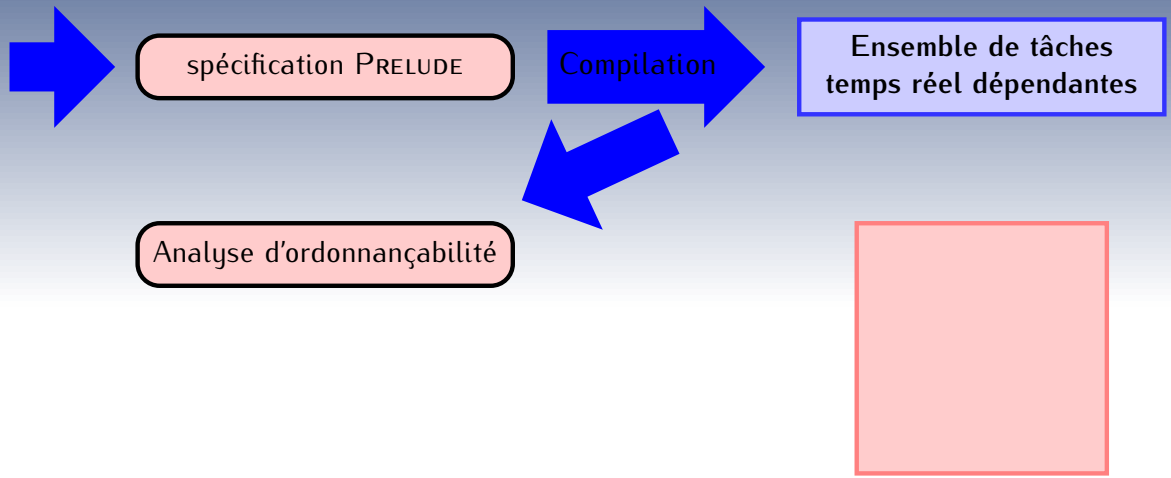
spécification PRELUDE → Compilation → **Ensemble de tâches temps réel dépendantes**

Monoprocesseur

- Comment modélise-t-on le système ? PRELUDE
- L'ensemble de tâches respecte-t-il les contraintes temporelles ?

Application : de l'assemblage à l'exécution

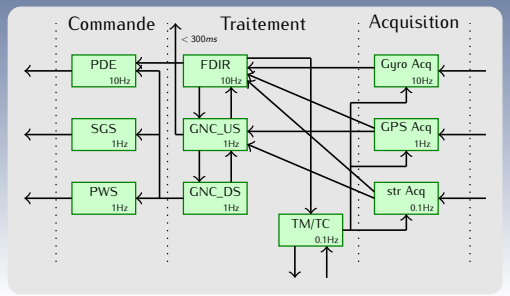


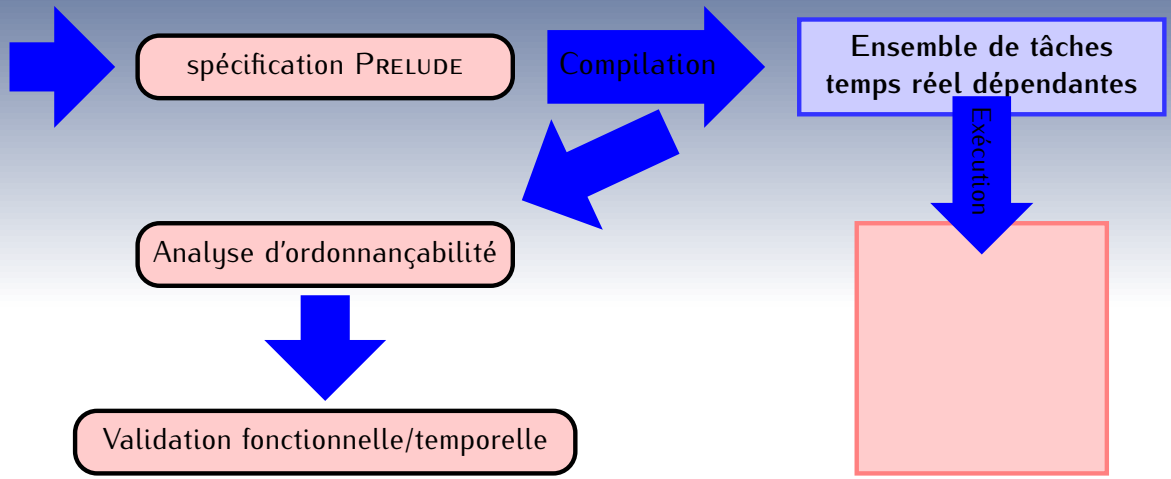


Monoprocesseur

- Comment modélise-t-on le système ? Prelude
- L'ensemble de tâches respecte-t-il les contraintes temporelles ? CNS pour RM et EDF
- L'exécution est-elle correcte ?

Application : de l'assemblage à l'exécution

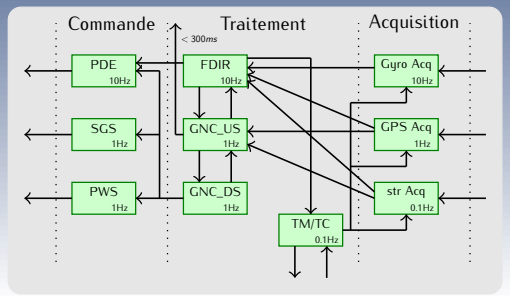


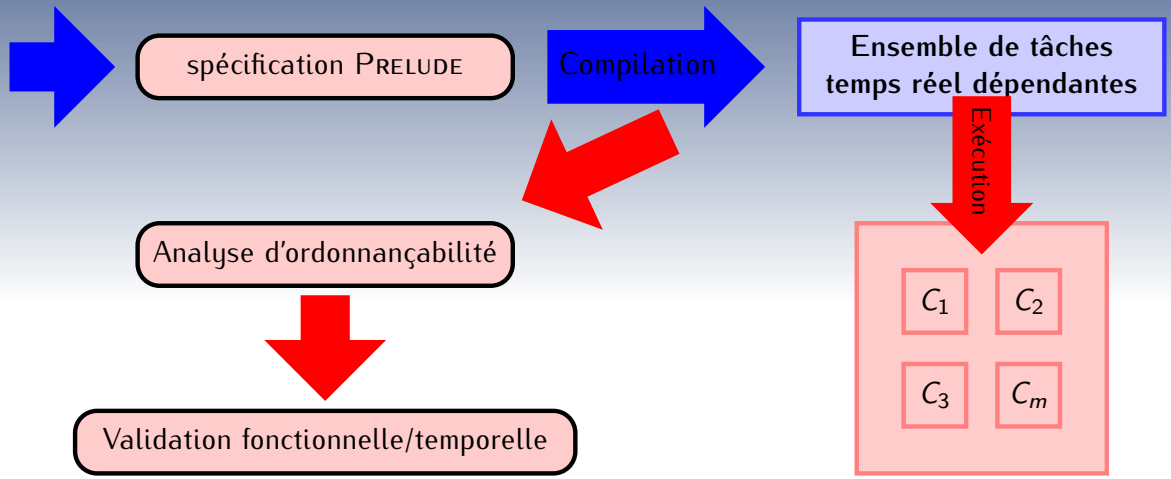


Monoprocesseur

- Comment modélise-t-on le système ? Prelude
- L'ensemble de tâches respecte-t-il les contraintes temporelles ? CNS pour RM et EDF
- L'exécution est-elle correcte ? MarteOS

Application : de l'assemblage à l'exécution

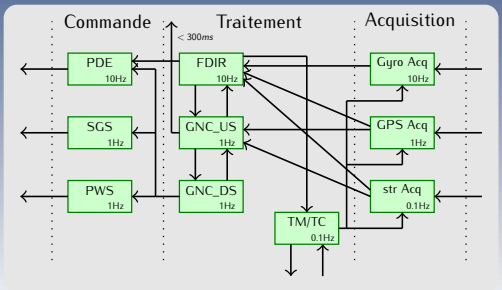




Multicœur

- Comment modélise-t-on le système ? Prelude
- L'ensemble de tâches respecte-t-il les contraintes temporelles ? SchedMCore Converter
- L'exécution est-elle correcte ? SchedMCore Runner

Description formelle du système



spécification PRELUDE → Compilation → **Ensemble de tâches temps réel dépendantes**

- PRELUDE est un langage formel conçu pour la spécification de l'architecture logicielle de systèmes multipériodiques
- Langage synchrone flot de données
- Hypothèse synchrone relâchée

Forget, J. (2009).
A Synchronous Language for Critical Embedded Systems with Multiple Real-Time Constraints.
PhD thesis, Université de Toulouse - ISAE/ONERA, Toulouse, France.

Exemple Prelude

```
imported node tau_1 (i0 :int, i1: int) returns (o1 :int, o2 :int) wcet 5;
imported node tau_2 (i0 :int) returns (o1 :int) wcet 10;
imported node tau_3 (i0 :int, i1: int) returns (o1 :int) wcet 20;
node sampling (i : rate (10,0)) returns (o1, o2)
var vf, vs;
let
  (o1,vf)=tau_1(i, (0 fby vs)*^3);
  vs=tau_2(vf/^3);
  o2 = tau_3((vf~>1/10)/^6,(vs~>1/30)/^2);
tel
```

date	0	1	10	20	30	40	50	60	61	...

Exemple Prelude

```
imported node tau_1 (i0 :int, i1: int) returns (o1 :int, o2 :int) wcet 5;
imported node tau_2 (i0 :int) returns (o1 :int) wcet 10;
imported node tau_3 (i0 :int, i1: int) returns (o1 :int) wcet 20;
node sampling (i : rate (10,0)) returns (o1, o2)
var vf, vs;
let
  (o1,vf)=tau_1(i, (0 fby vs)*^3);
  vs=tau_2(vf/^3);
  o2 = tau_3((vf~>1/10)/^6,(vs~>1/30)/^2);
tel
```

date	0	1	10	20	30	40	50	60	61	...
i	i^0		i^1	i^2	i^3	i^4	i^5	i^6		...

Exemple Prelude

```
imported node tau_1 (i0 :int, i1: int) returns (o1 :int, o2 :int) wcet 5;
imported node tau_2 (i0 :int) returns (o1 :int) wcet 10;
imported node tau_3 (i0 :int, i1: int) returns (o1 :int) wcet 20;
node sampling (i : rate (10,0)) returns (o1, o2)
var vf, vs;
let
  (o1,vf)=tau_1(i, (0 fby vs)*^3);
  vs=tau_2(vf/^3);
  o2 = tau_3((vf~>1/10)/^6,(vs~>1/30)/^2);
tel
```

date	0	1	10	20	30	40	50	60	61	...
i	i^0		i^1	i^2	i^3	i^4	i^5	i^6		...

v_f	v_f^0		v_f^1	v_f^2	v_f^3	v_f^4	v_f^5	v_f^6		...

Exemple Prelude

```
imported node tau_1 (i0 :int, i1: int) returns (o1 :int, o2 :int) wcet 5;
imported node tau_2 (i0 :int) returns (o1 :int) wcet 10;
imported node tau_3 (i0 :int, i1: int) returns (o1 :int) wcet 20;
node sampling (i : rate (10,0)) returns (o1, o2)
var vf, vs;
let
  (o1,vf)=tau_1(i, (0 fby vs)*^3);
  vs=tau_2(vf/^3);
  o2 = tau_3((vf~>1/10)/^6,(vs~>1/30)/^2);
tel
```

date	0	1	10	20	30	40	50	60	61	...
i	i^0		i^1	i^2	i^3	i^4	i^5	i^6		...

v_f	v_f^0		v_f^1	v_f^2	v_f^3	v_f^4	v_f^5	v_f^6		...
$v_f/\hat{}3$	v_f^0				v_f^3			v_f^6		...
v_s	v_s^0				v_s^1			v_s^2		...

Exemple Prelude

```
imported node tau_1 (i0 :int, i1: int) returns (o1 :int, o2 :int) wcet 5;
imported node tau_2 (i0 :int) returns (o1 :int) wcet 10;
imported node tau_3 (i0 :int, i1: int) returns (o1 :int) wcet 20;
node sampling (i : rate (10,0)) returns (o1, o2)
var vf, vs;
let
  (o1,vf)=tau_1(i, (0 fby vs)*^3);
  vs=tau_2(vf/^3);
  o2 = tau_3((vf~>1/10)/^6,(vs~>1/30)/^2);
tel
```

date	0	1	10	20	30	40	50	60	61	...
i	i^0		i^1	i^2	i^3	i^4	i^5	i^6		...
0 fby v_s	0				v_s^0			v_s^1		...
(0 fby v_s)*^3	0		0	0	v_s^0	v_s^0	v_s^0	v_s^1		...
v_f	v_f^0		v_f^1	v_f^2	v_f^3	v_f^4	v_f^5	v_f^6		...
v_f/^3	v_f^0				v_f^3			v_f^6		...
v_s	v_s^0				v_s^1			v_s^2		...

Exemple Prelude

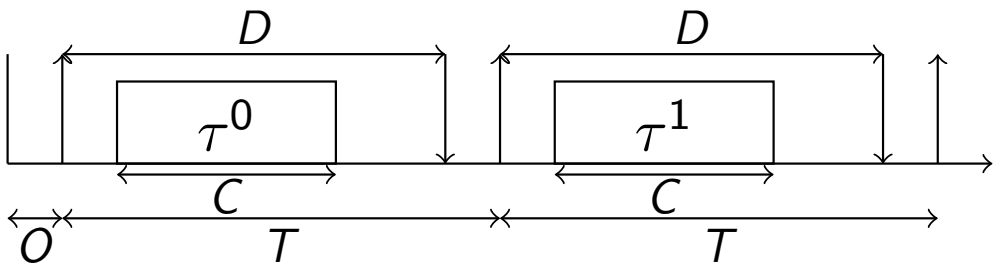
```
imported node tau_1 (i0 :int, i1: int) returns (o1 :int, o2 :int) wcet 5;
imported node tau_2 (i0 :int) returns (o1 :int) wcet 10;
imported node tau_3 (i0 :int, i1: int) returns (o1 :int) wcet 20;
node sampling (i : rate (10,0)) returns (o1, o2)
var vf, vs;
let
  (o1,vf)=tau_1(i, (0 fby vs)*^3);
  vs=tau_2(vf/^3);
  o2 = tau_3((vf~>1/10)/^6,(vs~>1/30)/^2);
tel
```

date	0	1	10	20	30	40	50	60	61	...
i	i^0		i^1	i^2	i^3	i^4	i^5	i^6		...
0 fby v_s	0				v_s^0			v_s^1		...
(0 fby v_s)*^3	0		0	0	v_s^0	v_s^0	v_s^0	v_s^1		...
v_f	v_f^0		v_f^1	v_f^2	v_f^3	v_f^4	v_f^5	v_f^6		...
v_f/^3	v_f^0				v_f^3			v_f^6		...
v_s	v_s^0				v_s^1			v_s^2		...
(v_f/~>1/10)/^6		v_f^0							v_f^6	...
(v_s/~>1/10)/^2		v_s^0							v_s^2	...

Modèle de tâches - Contraintes temps réel

Un programme PRELUDE génère un ensemble de tâches périodiques communicantes $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$ entrée de SCHEDMCORE :

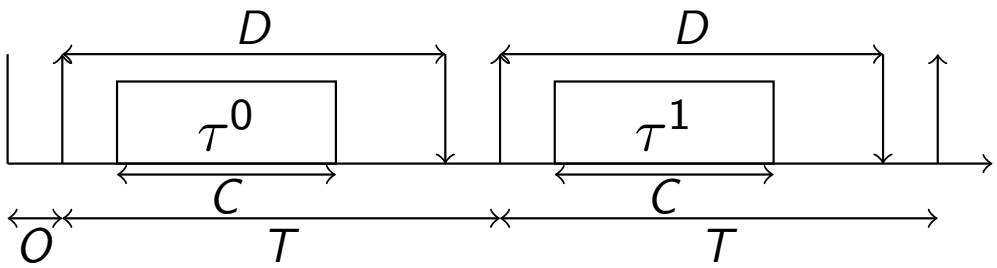
$\mathcal{S} = \{\tau_j = (O_j, T_j, D_j, C_j)\}_{j=1,\dots n}$ est un ensemble fini de tâches.
τ_j^i est le $i^{\text{ème}}$ job de τ_j ;



Modèle de tâches - Contraintes temps réel

Un programme PRELUDE génère un ensemble de tâches périodiques communicantes $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$ entrée de SCHEDMCORE :

$\mathcal{S} = \{\tau_j = (O_j, T_j, D_j, C_j)\}_{j=1,...n}$ est un ensemble fini de tâches.
τ_j^i est le $i^{\text{ème}}$ job de τ_j ;



Exemple **sampling** :

τ_i	O_i	T_i	D_i	C_i
τ_1	0	10	10	5
τ_2	0	30	30	10
τ_3	1	60	60	20

Modèle de tâches - Contraintes de précédence

Un programme PRELUDE génère un ensemble de tâches périodiques communicantes $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$:

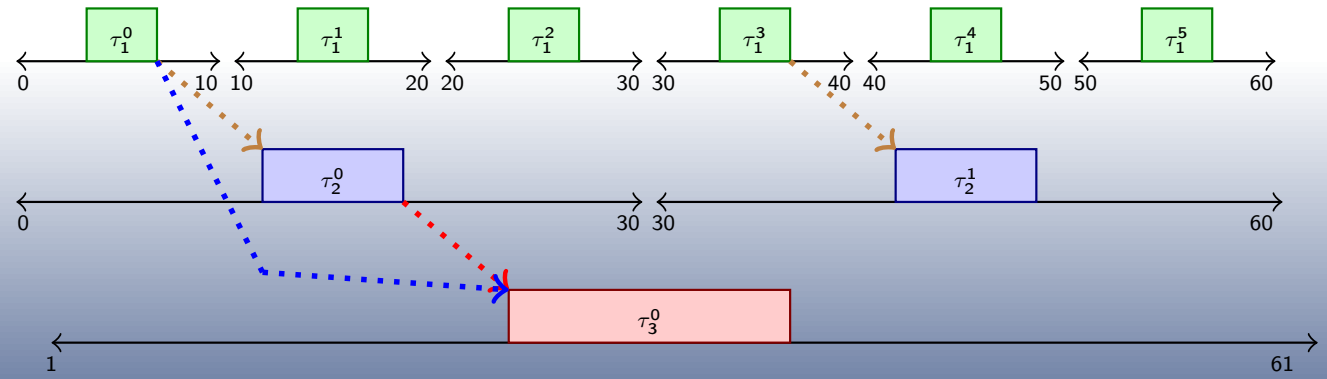
$\mathcal{R}$ est une relation de précédence, définie comme un ensemble de précédence répétitives entre **jobs** selon un patron.

Modèle de tâches - Contraintes de précédence

Un programme PRELUDE génère un ensemble de tâches périodiques communicantes $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$:

$\mathcal{R}$ est une relation de précédence, définie comme un ensemble de précédence répétitives entre **jobs** selon un patron.
Exemple **sampling** :

$$\tau_1 \xrightarrow{(0,0)} \tau_2 \; ; \tau_2 \xrightarrow{(0,0)} \tau_3 \; ; \; \tau_1 \xrightarrow{(0,0)} \tau_3$$



Modèle de tâches - Fonction de communication

Un programme PRELUDE génère un ensemble de tâches périodiques communicantes $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$:

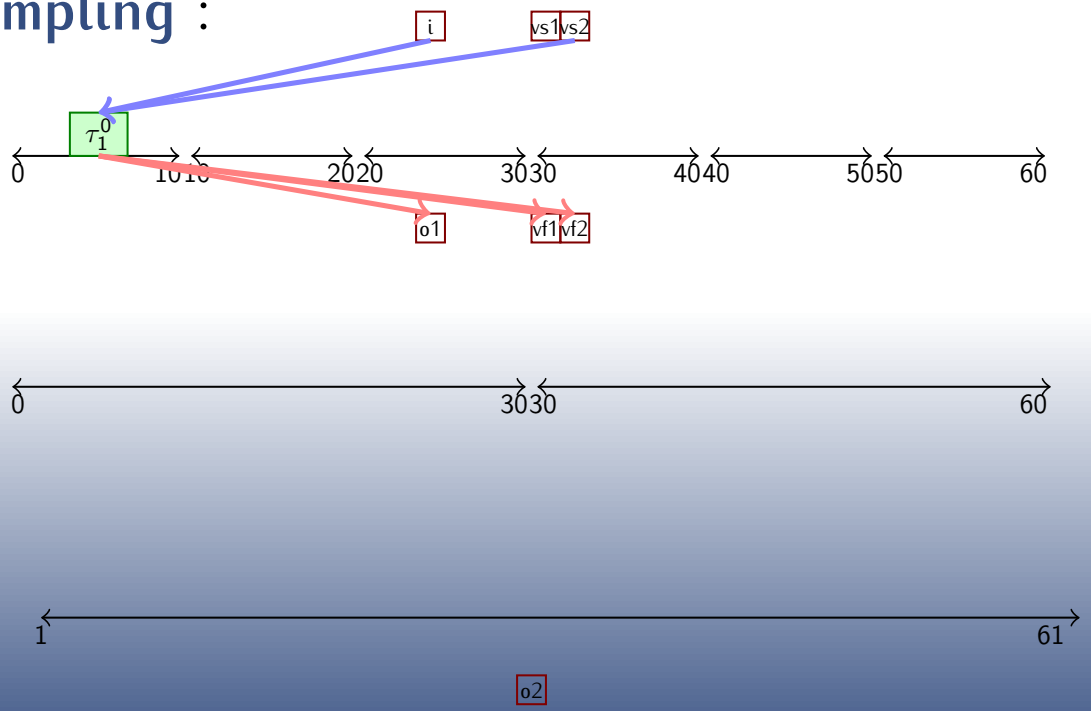
$\mathcal{C}$ est la fonction de communication implantée par un protocole sur buffers partagés. Indique pour chaque job dans quels buffers lire ou écrire ses données (modèle à mémoire partagée).

Modèle de tâches - Fonction de communication

Un programme PRELUDE génère un ensemble de tâches périodiques communicantes $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$:

$\mathcal{C}$ est la fonction de communication implantée par un protocole sur buffers partagés. Indique pour chaque job dans quels buffers lire ou écrire ses données (modèle à mémoire partagée).

Exemple **sampling** :

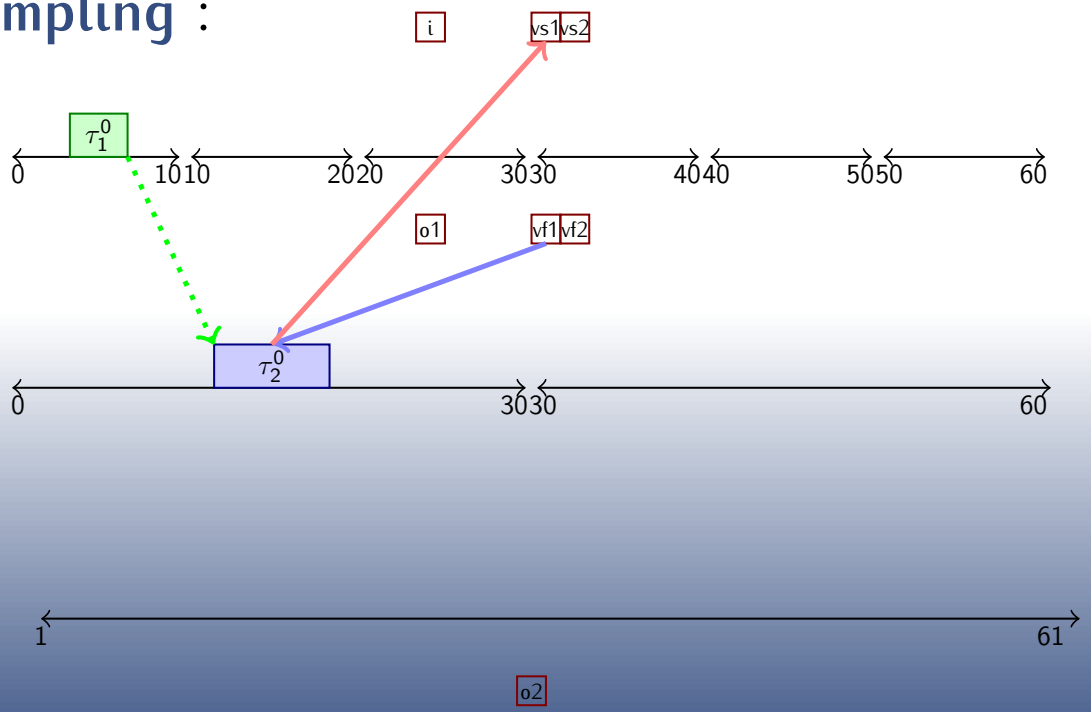


Modèle de tâches - Fonction de communication

Un programme PRELUDE génère un ensemble de tâches périodiques communicantes $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$:

$\mathcal{C}$ est la fonction de communication implantée par un protocole sur buffers partagés. Indique pour chaque job dans quels buffers lire ou écrire ses données (modèle à mémoire partagée).

Exemple **sampling** :

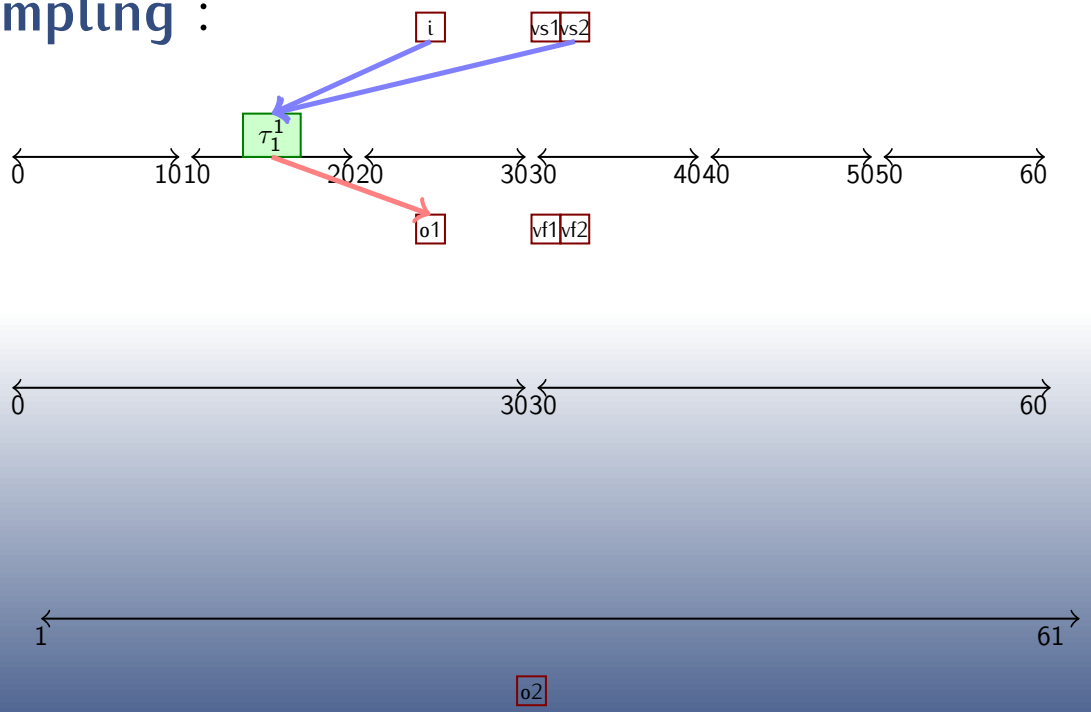


Modèle de tâches - Fonction de communication

Un programme PRELUDE génère un ensemble de tâches périodiques communicantes $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$:

$\mathcal{C}$ est la fonction de communication implantée par un protocole sur buffers partagés. Indique pour chaque job dans quels buffers lire ou écrire ses données (modèle à mémoire partagée).

Exemple **sampling** :

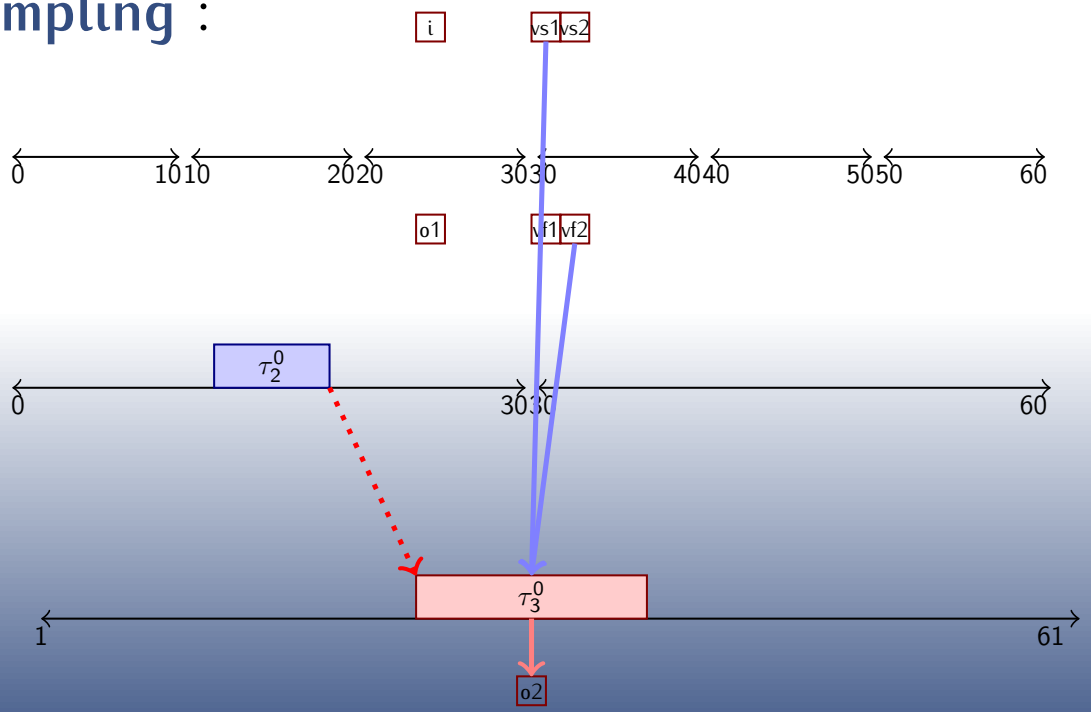


Modèle de tâches - Fonction de communication

Un programme PRELUDE génère un ensemble de tâches périodiques communicantes $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$:

$\mathcal{C}$ est la fonction de communication implantée par un protocole sur buffers partagés. Indique pour chaque job dans quels buffers lire ou écrire ses données (modèle à mémoire partagée).

Exemple **sampling** :

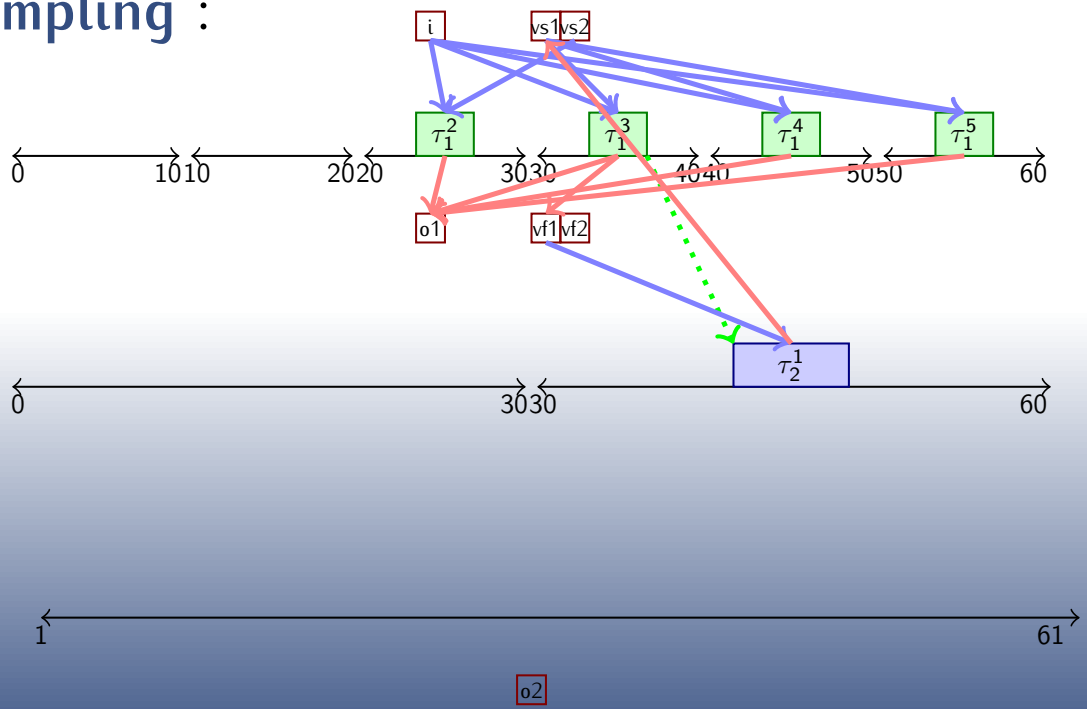


Modèle de tâches - Fonction de communication

Un programme PRELUDE génère un ensemble de tâches périodiques communicantes $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$:

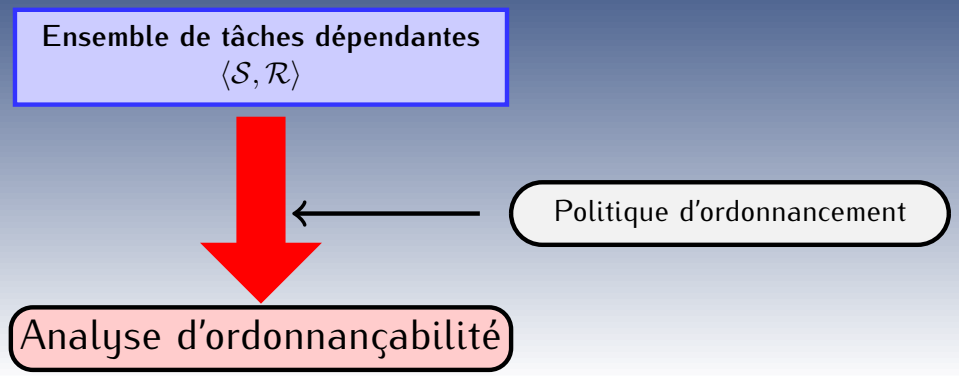
$\mathcal{C}$ est la fonction de communication implantée par un protocole sur buffers partagés. Indique pour chaque job dans quels buffers lire ou écrire ses données (modèle à mémoire partagée).

Exemple **sampling** :



Plan

Analyse d'ordonnançabilité



Objectif

Assurer que l'exécution d'un ensemble de tâches sur une architecture avec une politique donnée respecte toutes les contraintes temporelles et de précédence.

Plusieurs politiques analysées

Analyse d'ordonnançabilité des politiques d'ordonnancement existantes (FP, gEDF, gLLF, LLREF)

- Méthodes analytiques
- Méthodes de parcours exhaustif

Méthodes de parcours exhaustif

Principe

Génération de toutes les séquences d'ordonnancement afin de vérifier le respect des contraintes temporelles et de précédence

Méthodes de parcours exhaustif

Principe

Génération de toutes les séquences d'ordonnancement afin de vérifier le respect des contraintes temporelles et de précédence

Cyclicité

- Connu pour les ensembles **synchrones** : $[0, ppcm(T)]$
- Connu pour les ensembles **asynchrones** et une politique **FP** : $[S_n, S_n + ppcm(T)]$
- Inconnu pour les ensembles **asynchrones** et une politique à **priorité dynamique** : répétition à $k \times ppcm(T)$
- Pas de résultats pour les **ensembles dépendants** : répétition à $k \times ppcm(T)$

Cucu-Grosjean, L. and Goossens, J. (2011).
Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms.
Journal of Systems Architecture, 57(5) :561 – 569.
Special Issue on Multiprocessor Real-time Scheduling.

Méthodes de parcours exhaustif

Principe

Génération de toutes les séquences d'ordonnancement afin de vérifier le respect des contraintes temporelles et de précédence

Génération des séquences

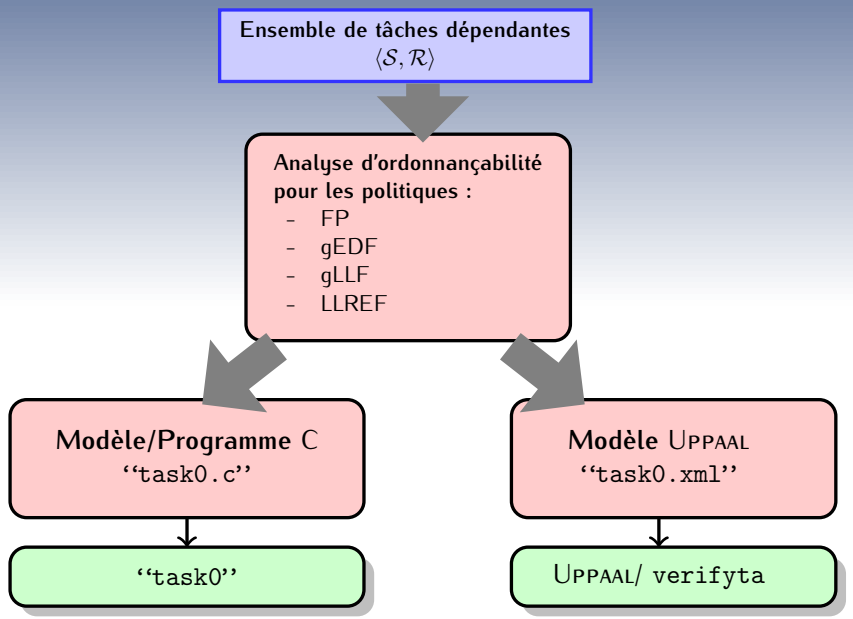
Plusieurs formalismes : automates temporisés, réseaux de Petri temporisés, automates finis, algorithmes ad-hoc...

- Explosion combinatoire
- Contraintes de précédence généralisées non modélisées

Cucu-Grosjean, L. and Goossens, J. (2011).
Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms.
Journal of Systems Architecture, 57(5) :561 – 569.
Special Issue on Multiprocessor Real-time Scheduling.

Plan

Analyse d'ordonnançabilité multiprocesseur : SchedMCore Converter



SchedMCore Converter

Un outil pour l'analyse d'ordonnançabilité de politiques préemptives globales (méthode de parcours exhaustif)

- encodage de l'analyse d'ordonnançabilité sous forme d'un problème d'accessibilité dans un **automate** fini
- génération des programmes C ou Uppaal pour l'exploration.

Encodage de configuration de tâches

Paramètres temporels

$conf(t)$ la configuration à l'instant t est le tuple $(O_\tau(t), T_\tau(t), D_\tau(t), C_\tau(t))$

Encodage de configuration de tâches

Paramètres temporels

$conf(t)$ la configuration à l'instant t est le tuple $(O_\tau(t), T_\tau(t), D_\tau(t), C_\tau(t))$

τ	(O_i, T_i, D_i, C_i)
$\tau_1(0)$	$(0, 10, 10, 5)$
$\tau_2(0)$	$(0, 30, 30, 10)$
$\tau_3(0)$	$(1, 60, 60, 20)$

Encodage de configuration de tâches

Paramètres temporels

$conf(t)$ la configuration à l'instant t est le tuple $(O_\tau(t), T_\tau(t), D_\tau(t), C_\tau(t))$

Contraintes de précédence

$M_{i,j}(t)$ est l'ensemble des mots actifs de la contrainte de précédence $\tau_i \rightarrow \tau_j$ à l'instant t. $M_{i,j}(t) = \{(m_1, m'_1), ..., (m_k, m'_k)\}$.

On a donc k contraintes $\tau_i^{m_k} \rightarrow \tau_j^{m'_k}$

τ	(O_i, T_i, D_i, C_i)
$\tau_1(0)$	$(0, 10, 10, 5)$
$\tau_2(0)$	$(0, 30, 30, 10)$
$\tau_3(0)$	$(1, 60, 60, 20)$

Encodage de configuration de tâches

Paramètres temporels

$conf(t)$ la configuration à l'instant t est le tuple $(O_\tau(t), T_\tau(t), D_\tau(t), C_\tau(t))$

Contraintes de précédence

$M_{i,j}(t)$ est l'ensemble des mots actifs de la contrainte de précédence $\tau_i \rightarrow \tau_j$ à l'instant t. $M_{i,j}(t) = \{(m_1, m'_1), ..., (m_k, m'_k)\}$.

On a donc k contraintes $\tau_i^{m_k} \rightarrow \tau_j^{m'_k}$

τ	(O_i, T_i, D_i, C_i)
$\tau_1(0)$	$(0, 10, 10, 5)$
$\tau_2(0)$	$(0, 30, 30, 10)$
$\tau_3(0)$	$(1, 60, 60, 20)$

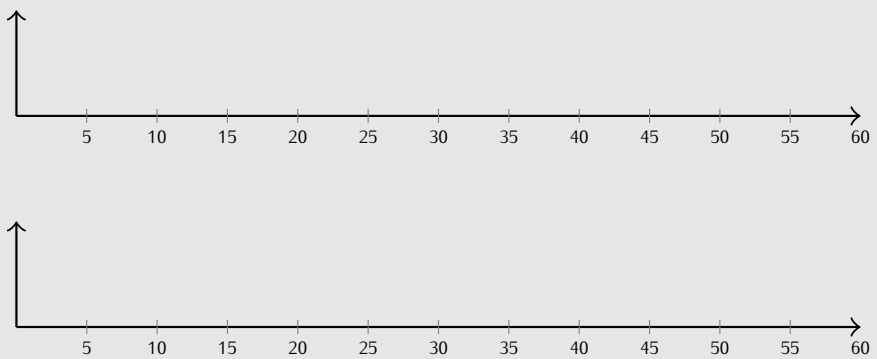
M	$(m_{i,j}, m'_{i,j})$
$M_{1,2}(0)$	$\{(0, 0)\}$
$M_{1,3}(0)$	$\{(0, 0)\}$
$M_{2,3}(0)$	$\{(0, 0)\}$

Séquence de configurations pour Sampling

τ_i	O_i	T_i	D_i	C_i	$\mathcal{R}$
τ_1	0	10	10	5	$\tau_1 \xrightarrow{(0,0)} \tau_2$
τ_2	0	30	30	10	$\tau_2 \xrightarrow{(0,0)} \tau_3$
τ_3	1	60	60	20	$\tau_1 \xrightarrow{(0,0)} \tau_3$

politique : FP, priorité $\tau_1 < \tau_2 < \tau_3$
architecture : 2 processeurs

time	0	1	5	10	30	61
τ_1	$(0, 10, 10, 5)$					
τ_2	$(0, 30, 30, 10)$					
$M_{1,2}$	$\{(0, 0)\}$					
τ_3	$(1, 60, 60, 20)$					
$M_{1,3}$	$\{(0, 0)\}$					
$M_{2,3}$	$\{(0, 0)\}$					

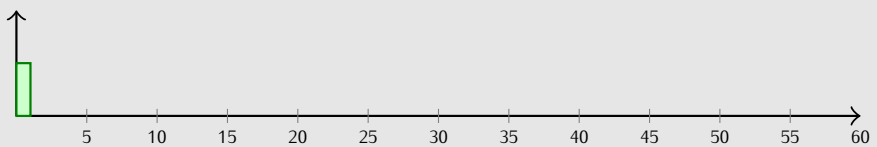


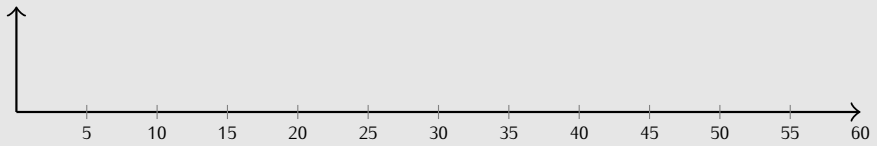
Séquence de configurations pour Sampling

τ_i	O_i	T_i	D_i	C_i	$\mathcal{R}$
τ_1	0	10	10	5	$\tau_1 \xrightarrow{(0,0)} \tau_2$
τ_2	0	30	30	10	$\tau_2 \xrightarrow{(0,0)} \tau_3$
τ_3	1	60	60	20	$\tau_1 \xrightarrow{(0,0)} \tau_3$

politique : FP, priorité $\tau_1 < \tau_2 < \tau_3$
architecture : 2 processeurs

time	0	1	5	10	30	61
τ_1	$(0, 10, 10, 5)$	$(0, 9, 9, 4)$				
τ_2	$(0, 30, 30, 10)$	$(0, 29, 29, 10)$				
$M_{1,2}$	$\{(0, 0)\}$	$\{(0, 0)\}$				
τ_3	$(1, 60, 60, 20)$	$(0, 60, 60, 20)$				
$M_{1,3}$	$\{(0, 0)\}$	$\{(0, 0)\}$				
$M_{2,3}$	$\{(0, 0)\}$	$\{(0, 0)\}$				



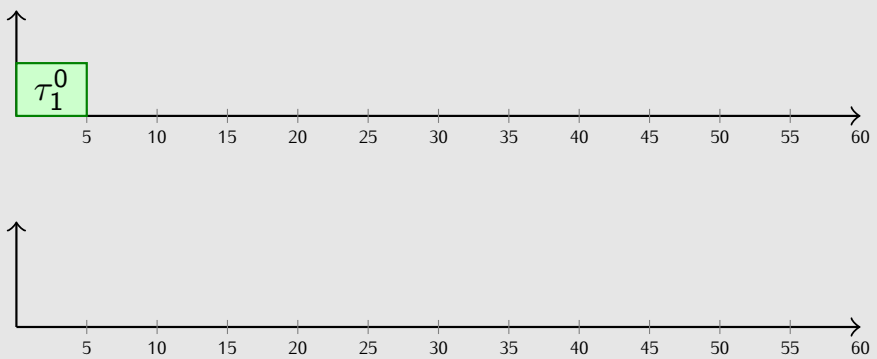


Séquence de configurations pour Sampling

τ_i	O_i	T_i	D_i	C_i	$\mathcal{R}$
τ_1	0	10	10	5	$\tau_1 \xrightarrow{(0,0)} \tau_2$
τ_2	0	30	30	10	$\tau_2 \xrightarrow{(0,0)} \tau_3$
τ_3	1	60	60	20	$\tau_1 \xrightarrow{(0,0)} \tau_3$

politique : FP, priorité $\tau_1 < \tau_2 < \tau_3$
architecture : 2 processeurs

time	0	1	5	10	30	61
τ_1	$(0, 10, 10, 5)$	$(0, 9, 9, 4)$	$(0, 5, 5, 0)$			
τ_2	$(0, 30, 30, 10)$	$(0, 29, 29, 10)$	$(0, 25, 25, 10)$			
$M_{1,2}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{\}$			
τ_3	$(1, 60, 60, 20)$	$(0, 60, 60, 20)$	$(0, 56, 56, 20)$			
$M_{1,3}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{\}$			
$M_{2,3}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{(0, 0)\}$			

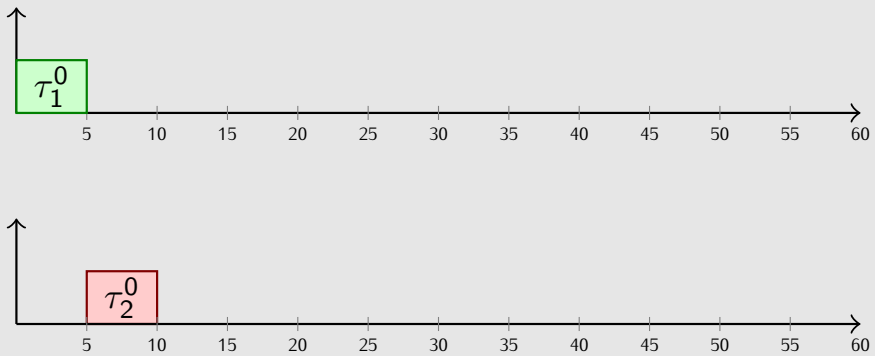


Séquence de configurations pour Sampling

τ_i	O_i	T_i	D_i	C_i	$\mathcal{R}$
τ_1	0	10	10	5	$\tau_1 \xrightarrow{(0,0)} \tau_2$
τ_2	0	30	30	10	$\tau_2 \xrightarrow{(0,0)} \tau_3$
τ_3	1	60	60	20	$\tau_1 \xrightarrow{(0,0)} \tau_3$

politique : FP, priorité $\tau_1 < \tau_2 < \tau_3$
architecture : 2 processeurs

time	0	1	5	10	30	61
τ_1	$(0, 10, 10, 5)$	$(0, 9, 9, 4)$	$(0, 5, 5, 0)$	$(0, 0, 0, 0) \rightarrow (0, 10, 10, 5)$		
τ_2	$(0, 30, 30, 10)$	$(0, 29, 29, 10)$	$(0, 25, 25, 10)$	$(0, 20, 20, 5)$		
$M_{1,2}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{\}$	$\{\}$		
τ_3	$(1, 60, 60, 20)$	$(0, 60, 60, 20)$	$(0, 56, 56, 20)$	$(0, 51, 51, 20)$		
$M_{1,3}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{\}$	$\{\}$		
$M_{2,3}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{(0, 0)\}$		

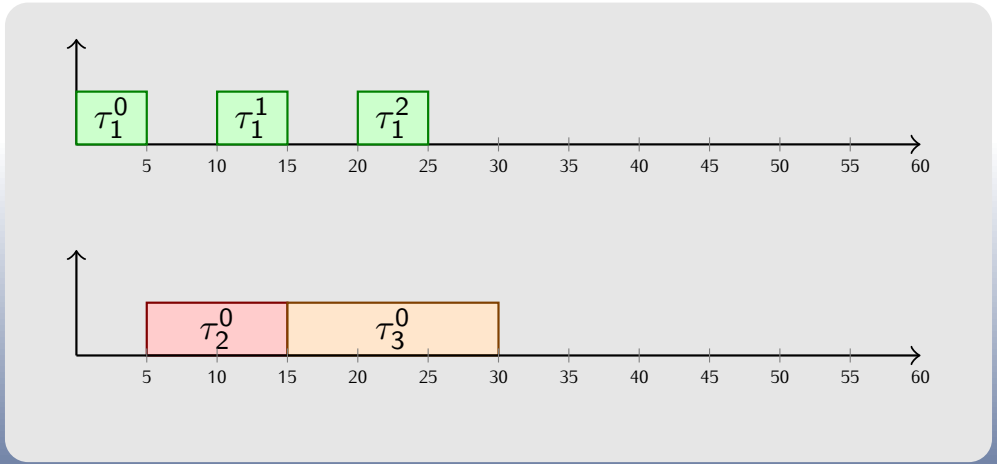


Séquence de configurations pour Sampling

τ_i	O_i	T_i	D_i	C_i	$\mathcal{R}$
τ_1	0	10	10	5	$\tau_1 \xrightarrow{(0,0)} \tau_2$
τ_2	0	30	30	10	$\tau_2 \xrightarrow{(0,0)} \tau_3$
τ_3	1	60	60	20	$\tau_1 \xrightarrow{(0,0)} \tau_3$

politique : FP, priorité $\tau_1 < \tau_2 < \tau_3$
architecture : 2 processeurs

time	0	1	5	10	30	61
τ_1	$(0, 10, 10, 5)$	$(0, 9, 9, 4)$	$(0, 5, 5, 0)$	$(0, 0, 0, 0) \rightarrow (0, 10, 10, 5)$	$(0, 0, 0, 0) \rightarrow (0, 10, 10, 5)$	
τ_2	$(0, 30, 30, 10)$	$(0, 29, 29, 10)$	$(0, 25, 25, 10)$	$(0, 20, 20, 5)$	$(0, 0, 0, 0) \rightarrow (0, 30, 30, 10)$	
$M_{1,2}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{\}$	$\{\}$	$\{(0, 0)\}$	
τ_3	$(1, 60, 60, 20)$	$(0, 60, 60, 20)$	$(0, 56, 56, 20)$	$(0, 51, 51, 20)$	$(0, 21, 21, 5)$	
$M_{1,3}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{\}$	$\{\}$	$\{\}$	
$M_{2,3}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{\}$	

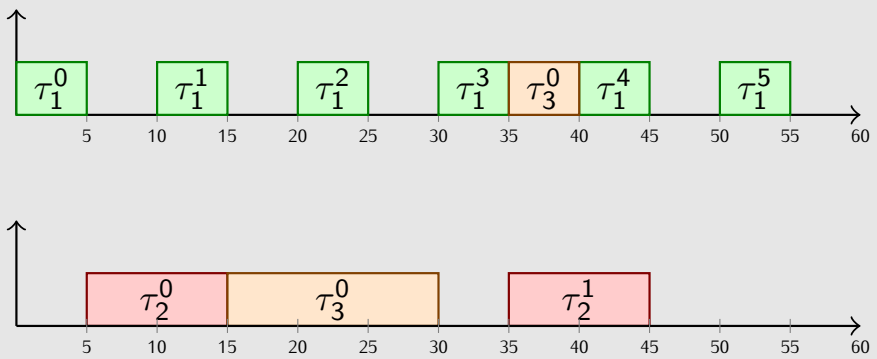


Séquence de configurations pour Sampling

τ_i	O_i	T_i	D_i	C_i	$\mathcal{R}$
τ_1	0	10	10	5	$\tau_1 \xrightarrow{(0,0)} \tau_2$
τ_2	0	30	30	10	$\tau_2 \xrightarrow{(0,0)} \tau_3$
τ_3	1	60	60	20	$\tau_1 \xrightarrow{(0,0)} \tau_3$

politique : FP, priorité $\tau_1 < \tau_2 < \tau_3$
architecture : 2 processeurs

time	0	1	5	10	30	61
τ_1	$(0, 10, 10, 5)$	$(0, 9, 9, 4)$	$(0, 5, 5, 0)$	$(0, 0, 0, 0) \rightarrow (0, 10, 10, 5)$	$(0, 0, 0, 0) \rightarrow (0, 10, 10, 5)$	$(0, 9, 9, 4)$
τ_2	$(0, 30, 30, 10)$	$(0, 29, 29, 10)$	$(0, 25, 25, 10)$	$(0, 20, 20, 5)$	$(0, 0, 0, 0) \rightarrow (0, 30, 30, 10)$	$(0, 29, 29, 10)$
$M_{1,2}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{\}$	$\{\}$	$\{(0, 0)\}$	$\{(0, 0)\}$
τ_3	$(1, 60, 60, 20)$	$(0, 60, 60, 20)$	$(0, 56, 56, 20)$	$(0, 51, 51, 20)$	$(0, 21, 21, 5)$	$(0, 0, 0, 0) \rightarrow (0, 60, 60, 20)$
$M_{1,3}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{\}$	$\{\}$	$\{\}$	$\{(0, 0)\}$
$M_{2,3}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{(0, 0)\}$	$\{\}$	$\{(0, 0)\}$



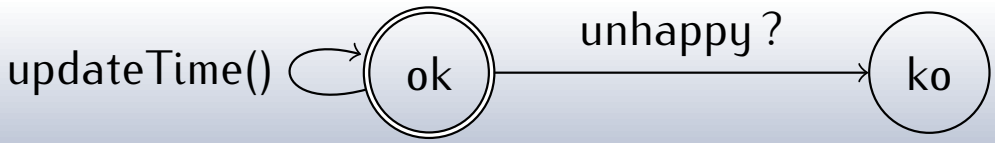
Analyse d'ordonnançabilité pour des politiques existantes

Étant donné :

- une politique d'ordonnancement $\mathcal{P}$ (FP, gEDF, gLLF ou LLREF)
- un ensemble de tâches dépendantes $\langle \mathcal{S}, \mathcal{R} \rangle$
- une plateforme $\mathcal{A}$ composée de p processeurs.

Génère un **automate de recherche** (UPPAAL et C) :

- l'automate décrit l'évolution concurrente des configurations
- la politique d'ordonnancement donne la sémantique de la transition **updateTime**.



A[] not ko : $\langle \mathcal{S}, \mathcal{R} \rangle$ est ordonnançable avec $\mathcal{P}$ sur $\mathcal{A}$ ssi l'état ko n'est pas accessible

Expérimentations avec SchedMCore Converter

Critères d'évaluation

- Taux d'ordonnançabilité : OK, KO, time-out et overflow
- Performances temporelles
- Types d'expérimentation : **nombre de tâches**, nombre de processeurs, contraintes de précédence et hyper-période

Expérimentations avec SchedMCore Converter

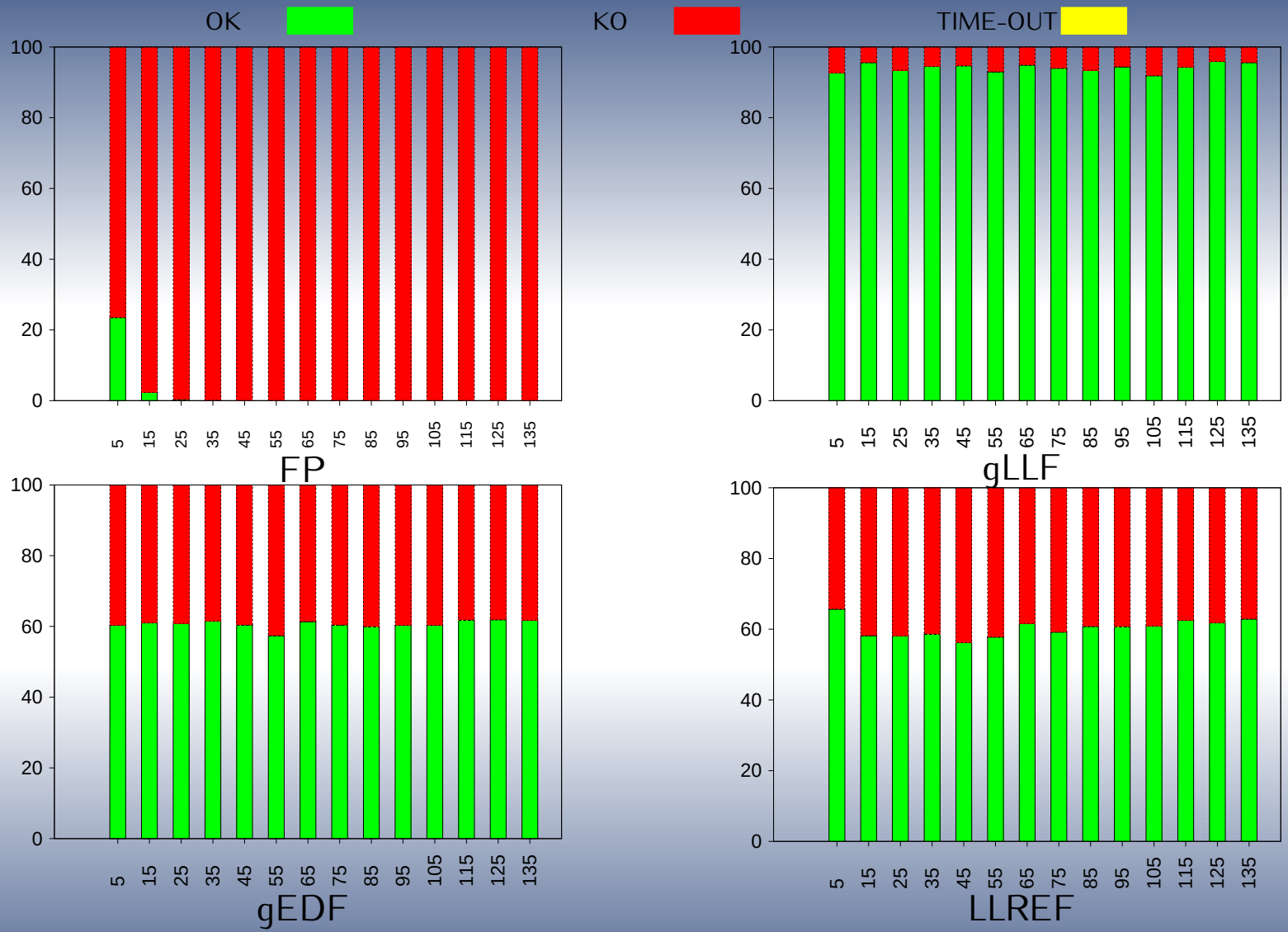
Critères d'évaluation

- Taux d'ordonnançabilité : OK, KO, time-out et overflow
- Performances temporelles
- Types d'expérimentation : **nombre de tâches**, nombre de processeurs, contraintes de précédence et hyper-période

Robustesse par rapport au nombre de tâches

- Charge processeur 80%
- Hyper-période de 10000 (géométriques, 4 périodes différentes)
- Nombre de tâches variable entre 5 et 135 (1000 échantillons)
- Sans contraintes de précédence
- Architecture avec 2 cœurs

Taux d'ordonnançabilité en C



Résultats pour gLLF très performants

Modèle UPPAAL vs. modèle C

Particularités UPPAAL

- overflow (LLREF $\sim$ 30%-40% d'overflow)
- time-out (gLLF à partir de 100 tâches $\sim$ 90%-100% de time-out)

Limites

- Modèle en C $\rightarrow$ limite d'encodage de C
- Modèle en UPPAAL
  - gEDF 300 (overflow)
  - gLLF 100 (time-out)
  - LLREF 100 (overflow et time-out)

Expérimentations

- Modèles en C $\rightarrow\sim$ 6 heures
- Modèles en UPPAAL $\rightarrow\sim$ 4 semaines

Plan

Recherche d'un ordonnancement (optimal) hors-ligne

Principe

Obtention d'une séquence finie valide qui puisse être exécutée en boucle

Recherche d'un ordonnancement (optimal) hors-ligne

Principe

Obtention d'une séquence finie valide qui puisse être exécutée en boucle

Optimalité

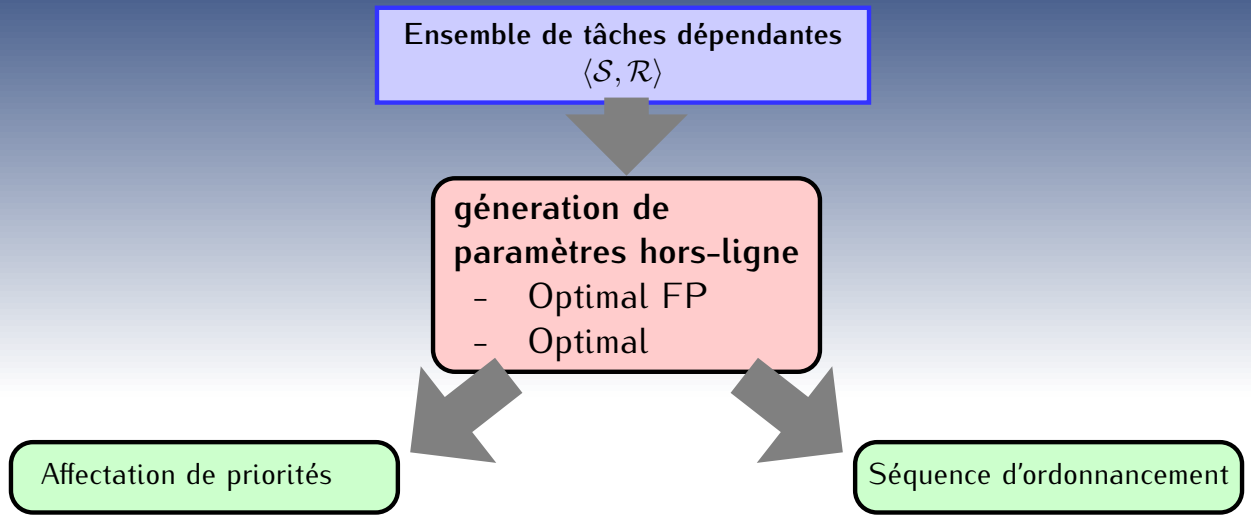
Une recherche optimale peut trouver un ordonnancement valide s'il en existe un

Trois solutions existent :

- Séparation et évaluation → non asynchrone
- Réseaux de Petri → explosion combinatoire
- Automates temporisés avec coût → jobs

Plan

Recherche de paramètres hors-ligne



Optimal FP

- deux méthodes : brute force et heuristique sous optimale
- génération de modèles C et UPPAAL pour l'exploration

Optimal

- encodage de la recherche d'une séquence sous forme d'un chemin dans un **automate** fini
- génération d'un modèle UPPAAL pour l'exploration.

Recherche d'une solution optimale hors-ligne

Étant donné :

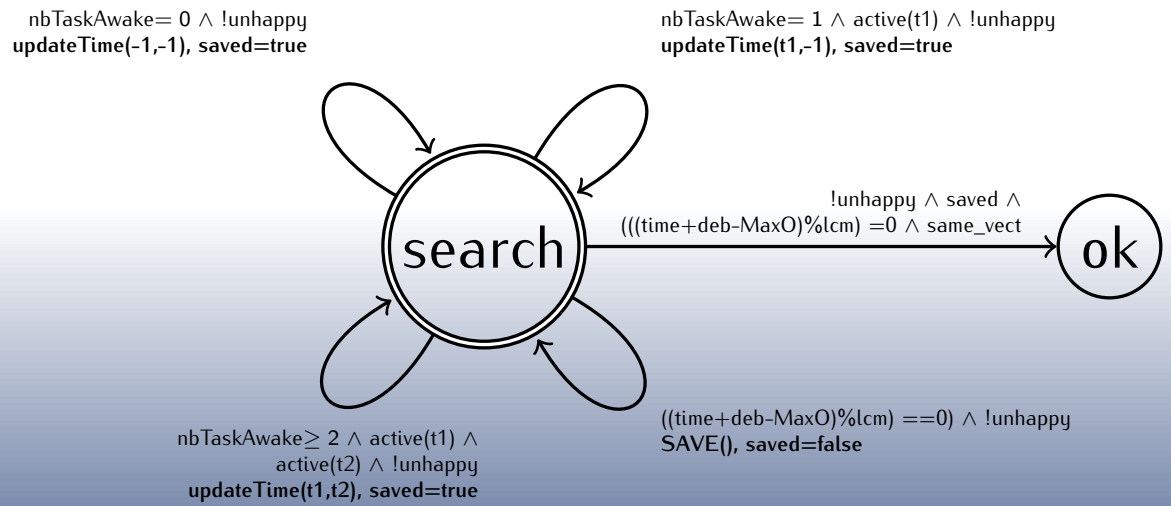
- un ensemble de tâches dépendantes $\langle \mathcal{S}, \mathcal{R} \rangle$
- une plateforme $\mathcal{A}$ composée de p processeurs

Sortie :

- Séquence d'exécution

Génère un **automate de recherche** (UPPAAL) :

- Produit un **automate de recherche** (exemple 2 processeurs)
- Recherche d'un cycle dans la séquence de configurations

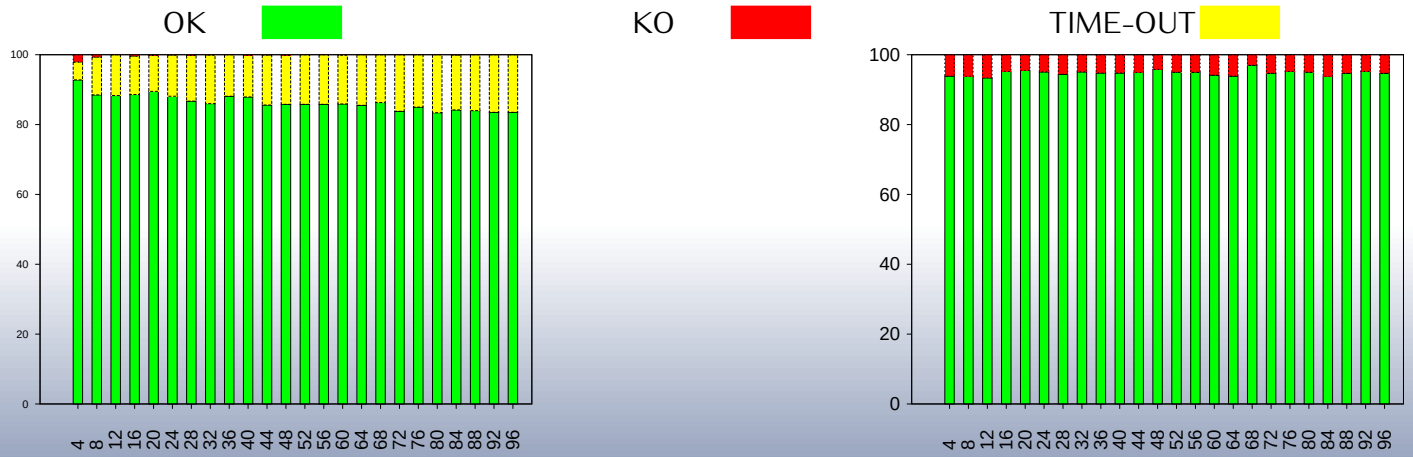


E <> ok : Une séquence valide existe ssi l'état ok est accessible

Performances pour la recherche hors-ligne optimale (optimal vs. gLLF)

Robustesse par rapport au nombre de tâches

- Hyper-période = 1000 (géométrique, 4 périodes différentes)
- Charge du processeur 80%
- Nombre de tâches variable entre 4 et 96 (1000 échantillons)
- Sans contrainte de précédence
- 2 processeurs



Optimal

gLLF en C

Résultats très similaires

Plan

Environnement d'exécution

Objectif d'exécuter $\langle \mathcal{S}, \mathcal{R}, \mathcal{C} \rangle$

- Correction fonctionnelle et temporelle
- Surcoûts liés à l'architecture

Trois approches existantes :

- Simulation
- Système d'exploitation temps réel
- Solution **ordonnanceur utilisateur**

État de l'art

Simulateurs

Permet de simuler le comportement temporel et/ou fonctionnel du système

- Simulation temporelle
  - → Simulateur de l'ordonnancement (Storm)
  - → Simulateur de la trace d'exécution (MAST)
- Simulation temporelle et fonctionnelle monoprocesseur
  - → Simulateur fonctionnel (Luciole, Scade Suite)
- Simulation temporelle et fonctionnelle multiprocesseur
  - → Simulateur d'architecture (SystemC, SIMICS, Graphite)

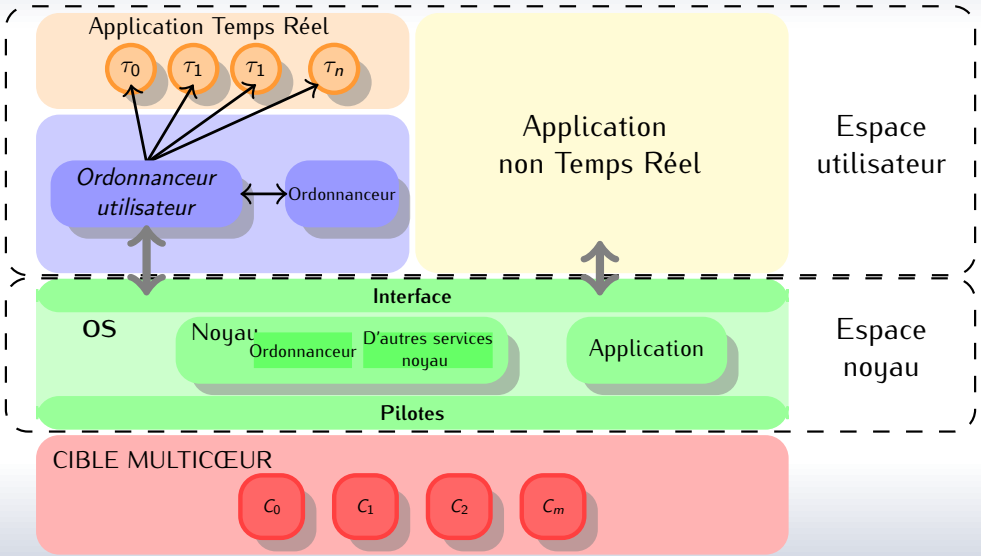
Systèmes d'exploitation

+ Bonnes performances car la gestion est faite en espace noyau
- Modification implique un changement dans le noyau
- Développement délicat
- À chaque modification il faut reconstruire tout le système (développement long)

État de l'art : Solution ordonnanceur utilisateur

Ordonnanceur utilisateur

Solution intermédiaire qui ajoute une couche supplémentaire en espace utilisateur pour fournir la gestion temporelle sur un système non nécessairement temps réel



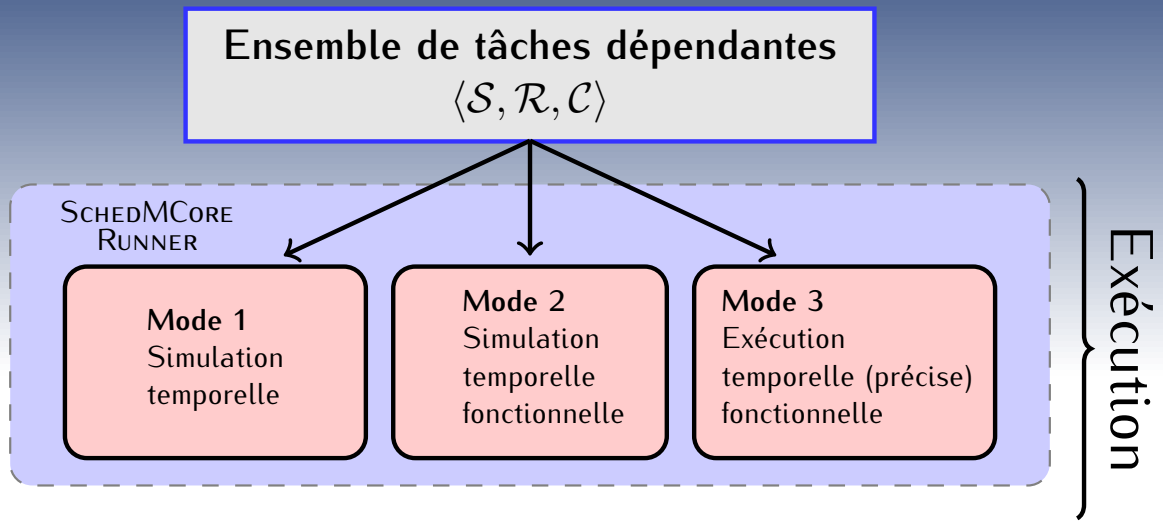
\+ Portable, plus facile à maintenir et à actualiser

\- Architecture monoprocesseur

• MarteOS, MetaScheduler

Plan

Plateforme d'exécution



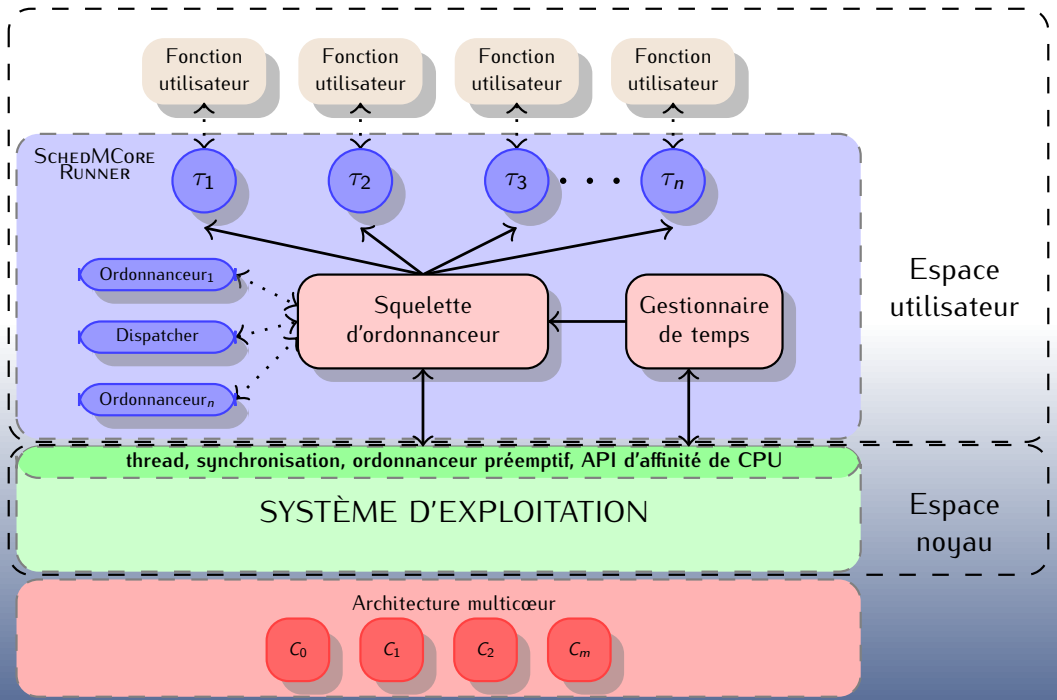
SchedMCore Runner

Plateforme pour l'exécution des ensembles de tâches temps réel sur une architecture multicœur

- trois types d'exécution
  - simulation fonctionnelle
  - simulation de l'exécution
  - exécution en temps précis
- Portable et extensible

Architecture de SchedMCore Runner

- **Gestionnaire de temps** est réveillé périodiquement pour débloquer le squelette d'ordonnanceur
- **Squelette d'ordonnanceur** contrôle la communication avec le système d'exploitation
- **Politique d'ordonnancement** implémente une politique d'ordonnancement (FP, gEDF, gLLF, LLREF et dispatcher).



Contexte

Conclusion

Plateforme de bout-en-bout ouverte et extensible pour le développement de systèmes temps réel multipériodiques

- SchedMCore Converter
  - Analyse d'ordonnancement pour des politiques en-ligne (FP, gEDF, gLLF et LLREF)
  - Calcul d'une affectation de priorités valide en FP
  - Recherche (optimale) d'une séquence valide hors-ligne
- SchedMCore Runner
  - Simulation temporelle
  - Simulation temporelle et fonctionnelle
  - Exécution temps précis

Exemple FAS

Analyse d'ordonnançabilité

- 19 tâches asynchrones et 20 contraintes de précédence
- 2 processeurs
- Hyper-période de 10000 (géométriques, 4 périodes différentes)

politique	ordonnançable ?	temps de calcul C	temps de calcul UPPAAL
FP	Non ordonnançable	0.002s	0.085s
gEDF	Ordonnançable	0.013s	1.064s
gLLF	Ordonnançable	0.039s	2.332s
LLREF	Non ordonnançable	0.002s	0.240s
FP optimal	Non ordonnançable	3.034s	42.110s
optimal	Ordonnançable	-	3m10.142s

La trace obtenue pour la méthode optimale est exprimée sur 10503 unités de temps

Exécution

- Exécution sur Linux dans les 3 modes et avec toutes les politiques d'ordonnancement
- Tests sur 1 à 4 cœurs

Publications

SchedMCore Converter

Multiprocesseur schedulability analyser
Mikel Cordovilla, Frédéric Boniol, Eric Noulard, Claire Pagetti
26th Symposium On Applied Computing (SAC2011)

Prelude + SchedMCore Converter + SchedMCore Runner

Developing critical embedded systems on multicore architectures : the Prelude-Schedmcore
Mikel Cordovilla, Frédéric Boniol, Julien Forget, Eric Noulard, Claire Pagetti
19th International Conference on Real-Time and Network Systems (RTNS2011)

Recherche optimale hors-ligne

Off-line (Optimal) Multiprocessor Scheduling of Dependent Periodic Tasks
Julie Baro, Frédéric Boniol, Mikel Cordovilla, Eric Noulard, Claire Pagetti
27th Symposium On Applied Computing (SAC2012)

Perspectives

Réflexion sur les politiques d'ordonnancement :

- Analyse détaillée des politiques d'ordonnancement
- Optimiser LLREF pour améliorer ses performances en présence de contraintes de précédence

SchedMCore Converter :

- Traduction des modèles des recherche de paramètres hors-ligne à d'autres méthodes de recherche (Fiacre/Tina)
- Sélection de traces d'exécution hors-ligne

SchedMCore Runner :

- Extension à des manycœurs

Maintenance du code :

- Tests de référence et non régression

Questions

Merci pour votre attention !

SchedMCore : `http://sites.onera.fr/schedmcore`