UMR IRISA

# Minimiser l'impact des communications lors de l'ordonnancement d'applications parallèles temps réel sur des multi-cœurs

## Thèse de doctorat

**Benjamin Rouxel**
Équipe Pacap/Cairn
Directeur: Isabelle Puaut
Co-directeur: Steven Derrien

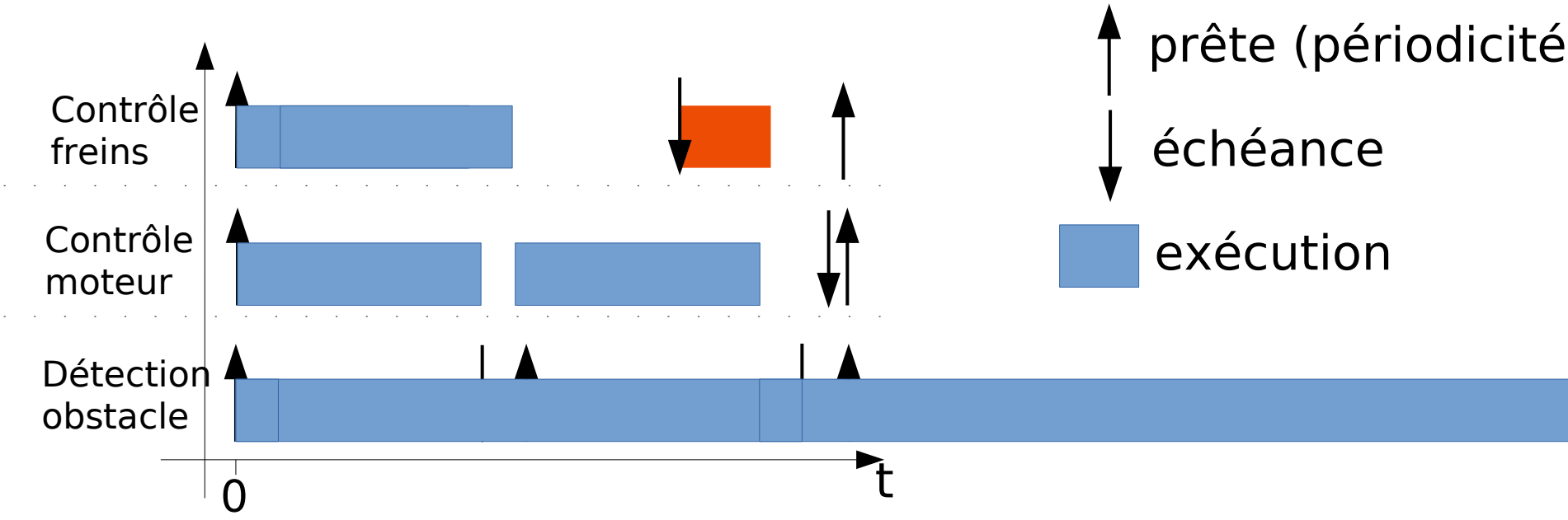
Institut de Recherche en Informatique et Systèmes Aléatoires

# Systèmes embarqués temps réel

- Spécialisés
- Autonomes
- Capacité limitée

- Contraintes temporelles


# lématiques des systèmes temps réel mono-cœ



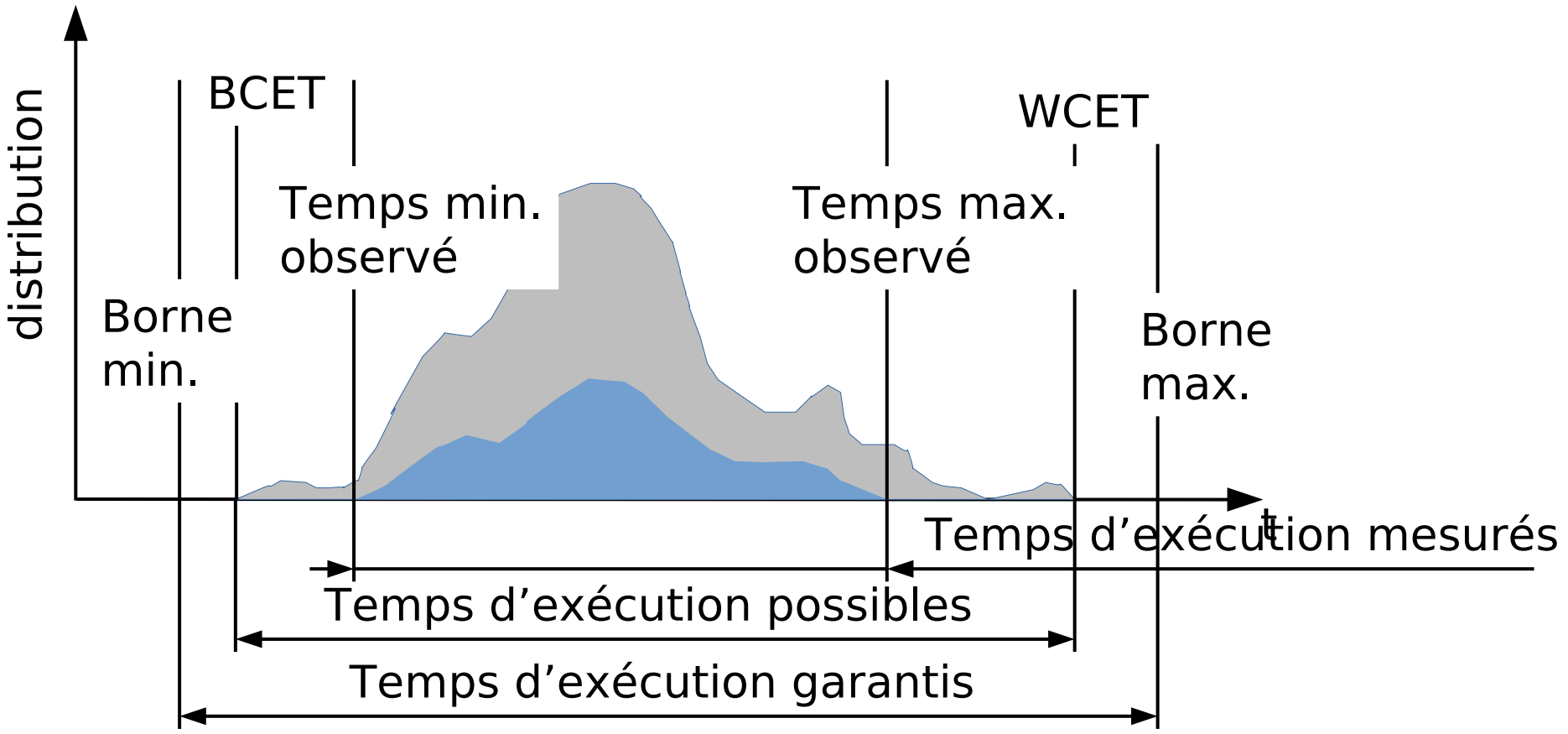

Estimation du pire temps d'exécution

Stratégie d'ordonnancement

# Estimation du pire temps d'exécution sur 1 cœur

- **Mesures**
- **Statique**

*Wilhelm'2008*



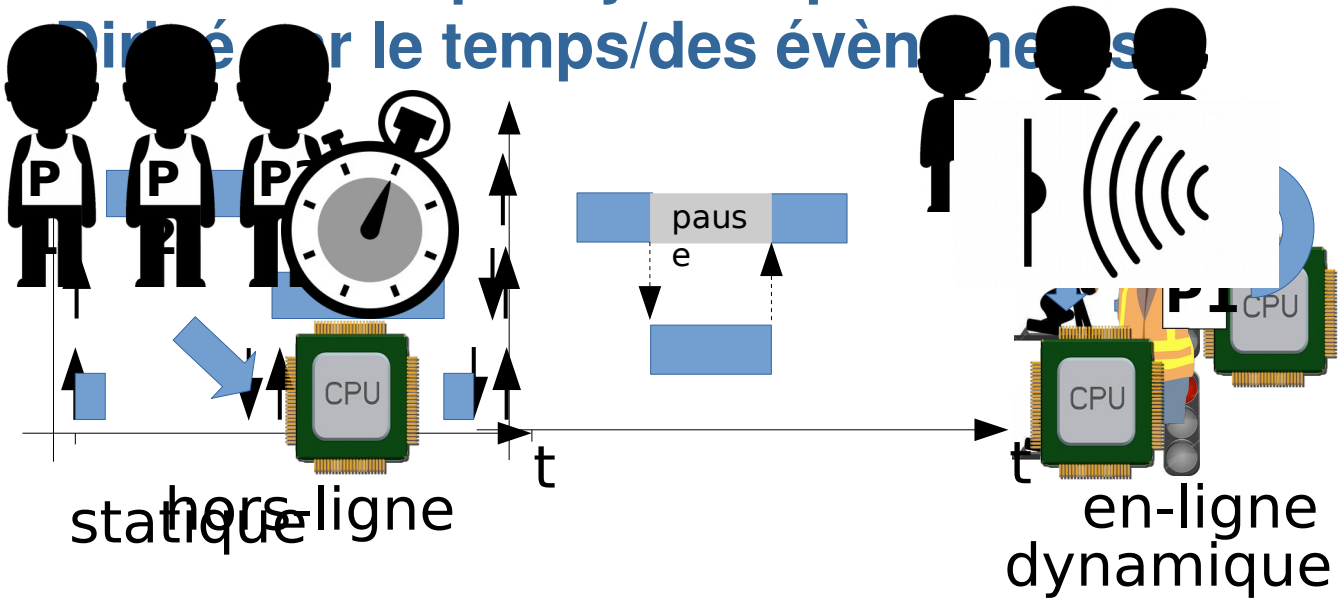
# Stratégie d'ordonnancement mono-cœur

## Définition

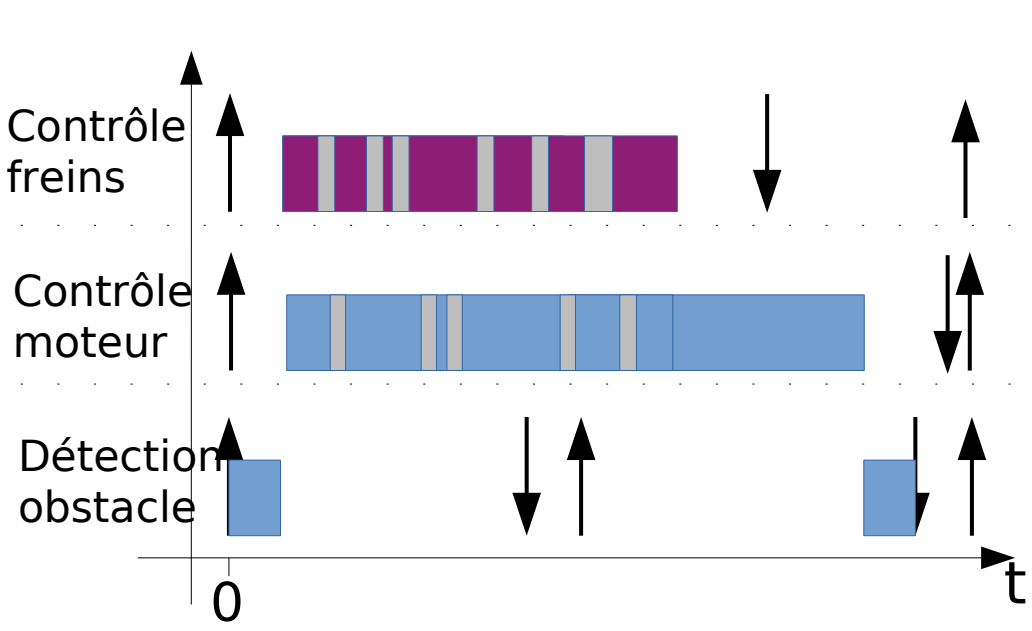
- **Déterminer l'ordre d'exécution des tâches**

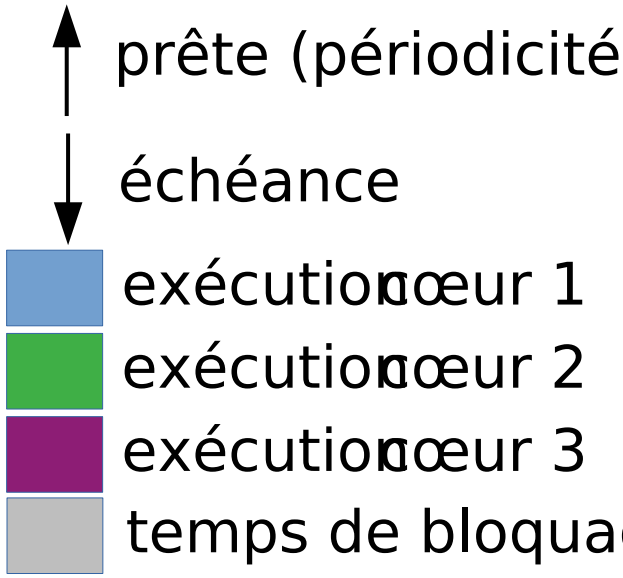
## Taxonomie *George'96 Davis'09*

- **Hors-/En-ligne**
- **(Non-)pré-emptif**
- **Priorité statique/dynamique**
- **Piloté par le temps/des évènements**



# ématiques des systèmes temps réel multi-cœu




Estimation du pire temps d'exécution

Stratégie d'ordonnancement

Stratégie de placement

Gestion des ressources partagées

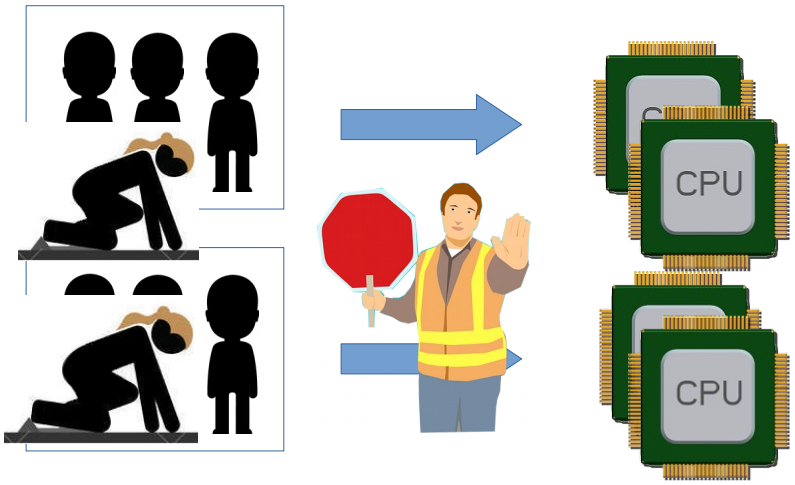
# Placement & ordonnancement

## Définition

- **Déterminer le cœur et l'ordre d'exécution des tâches**

## Taxonomie *George'96 Davis'09*

- Hors-/En-line
- (Non-)pré-emptif
- Priorité statique/dynamique
- Dirigé par le temps/des évènements
- Partitionné
- Global



# Gestion des ressources à l'ordonnancement


| | Pire contention | Contention considérée | Sans contention |
|---|---|---|---|
| Accès mémoire bloquants | *Tendulkar'14*<br>*Alhammad' 14* | *Nguyen'17*<br>*Puffitsch'15*<br>*Skalistis'17*<br>Contribution 1 | *Becker'16*<br>*Alhammad' 14* |
| Accès mémoire masqués | *Kudlur'08*<br>*Cho'09* | Composition | *Becker'18*<br>Contribution 2 |

# Contributions principales

Contribution 1 :

- **Génération d'ordonnancements consciente des interférences**

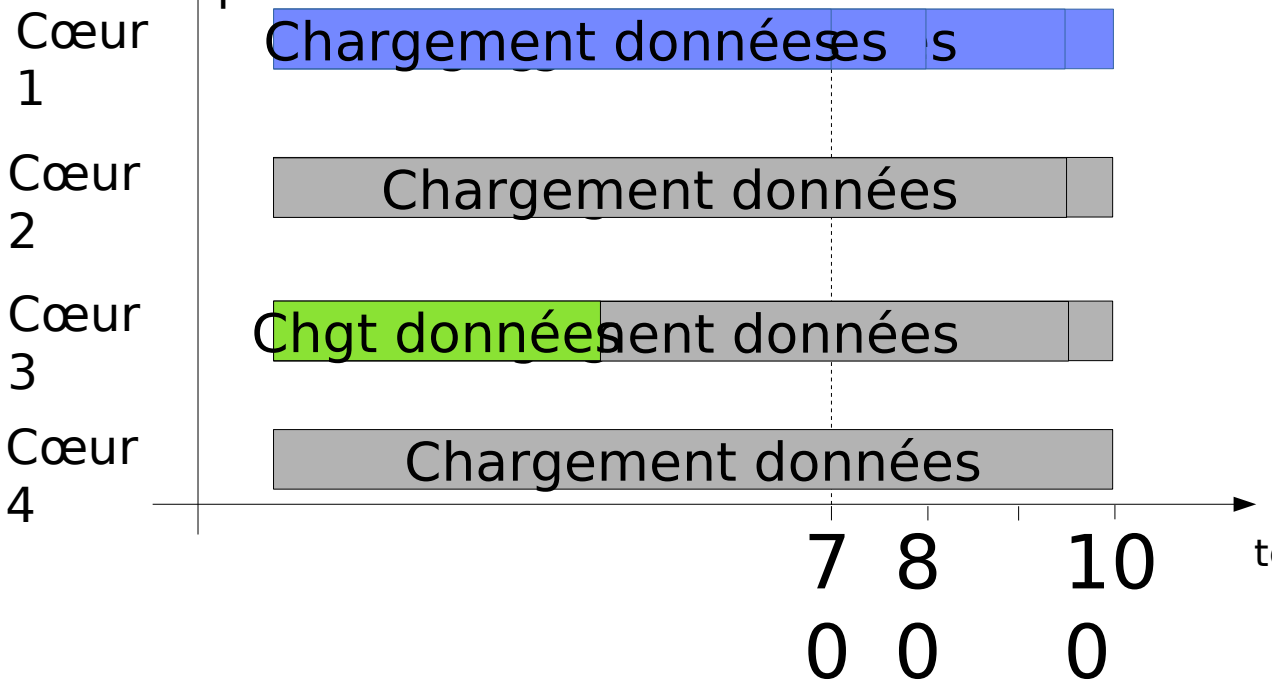
Contribution 2 :

- **Génération d'ordonnancements masquant les latences de communication**

# Quantifier les interférences en ordonnançant hors-ligne

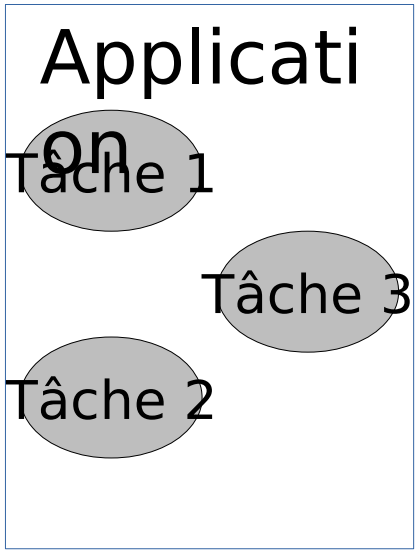
## Exemple

Combien de temps est nécessaire dans le pire cas ?



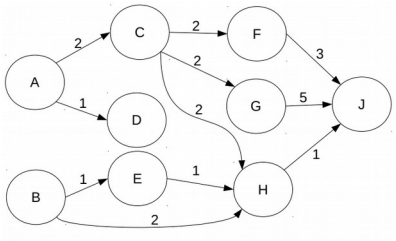

Connaissance du logiciel

Connaissance de l'ordonnancement

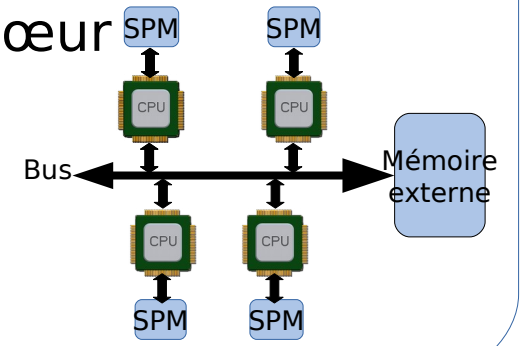


# Contributions (1): Réduire les délais de contention

Application en graphe




Architecture multi-cœur

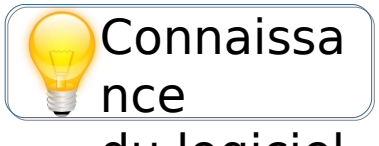



Pire temps d'exécution

- Traitement
- Communication

Stratégie d'ordonnancement

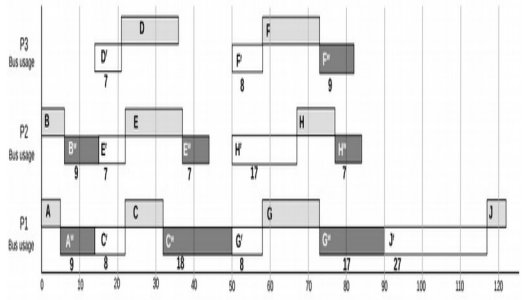
- ILP
- Heuristique



Connaissance du logiciel

Connaissance de l'ordonnancement

Ordonnancement conscient de la contention

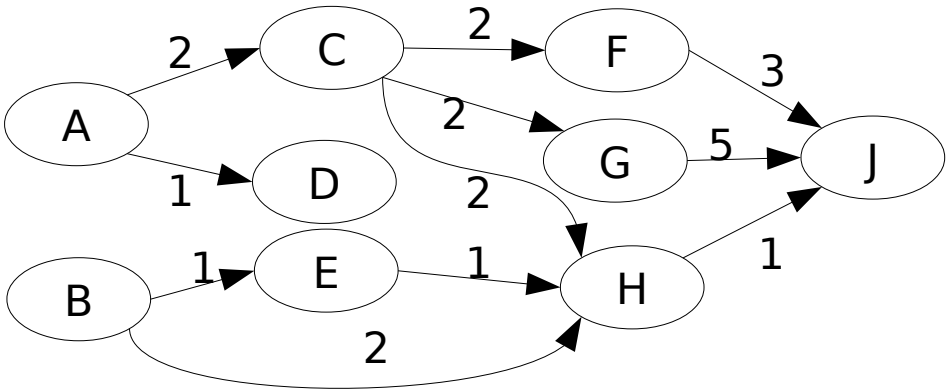


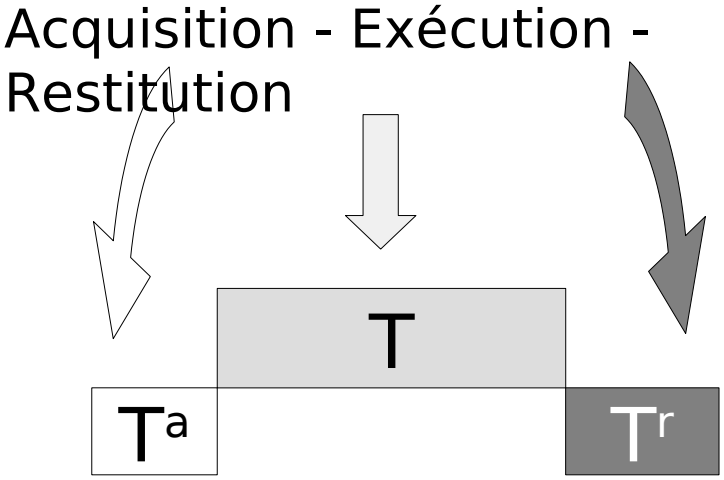

- Hors-ligne
- Partitionné
- Dirigé par le temps
- Non-préemptif

*Rouxel'2017*

# Graphe d'application & Modèle d'exécution

- **Graphe orienté acyclique (DAG)**
- **Modèle d'exécution prédictible (AER)** *Maia'2016*

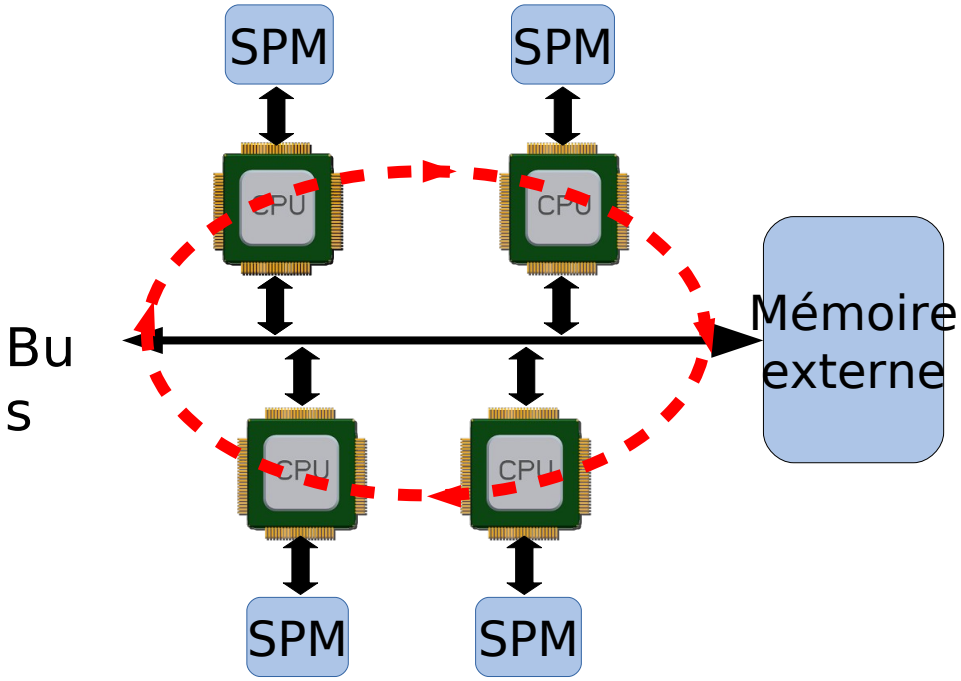


# Architecture matérielle & modèle de communication

- **Architecture multi-coeur**
- **Mémoire ScratchPad (SPM)**
- **Bus arbitré avec une politique FAIR-Round-Robin**

*Enslow'197*




$T_{créneau}$ : temps max d'accès

Accès mémoire bloqu

# mation temporelle des phases

Pire temps d'exécution pour la phase: *exécution*

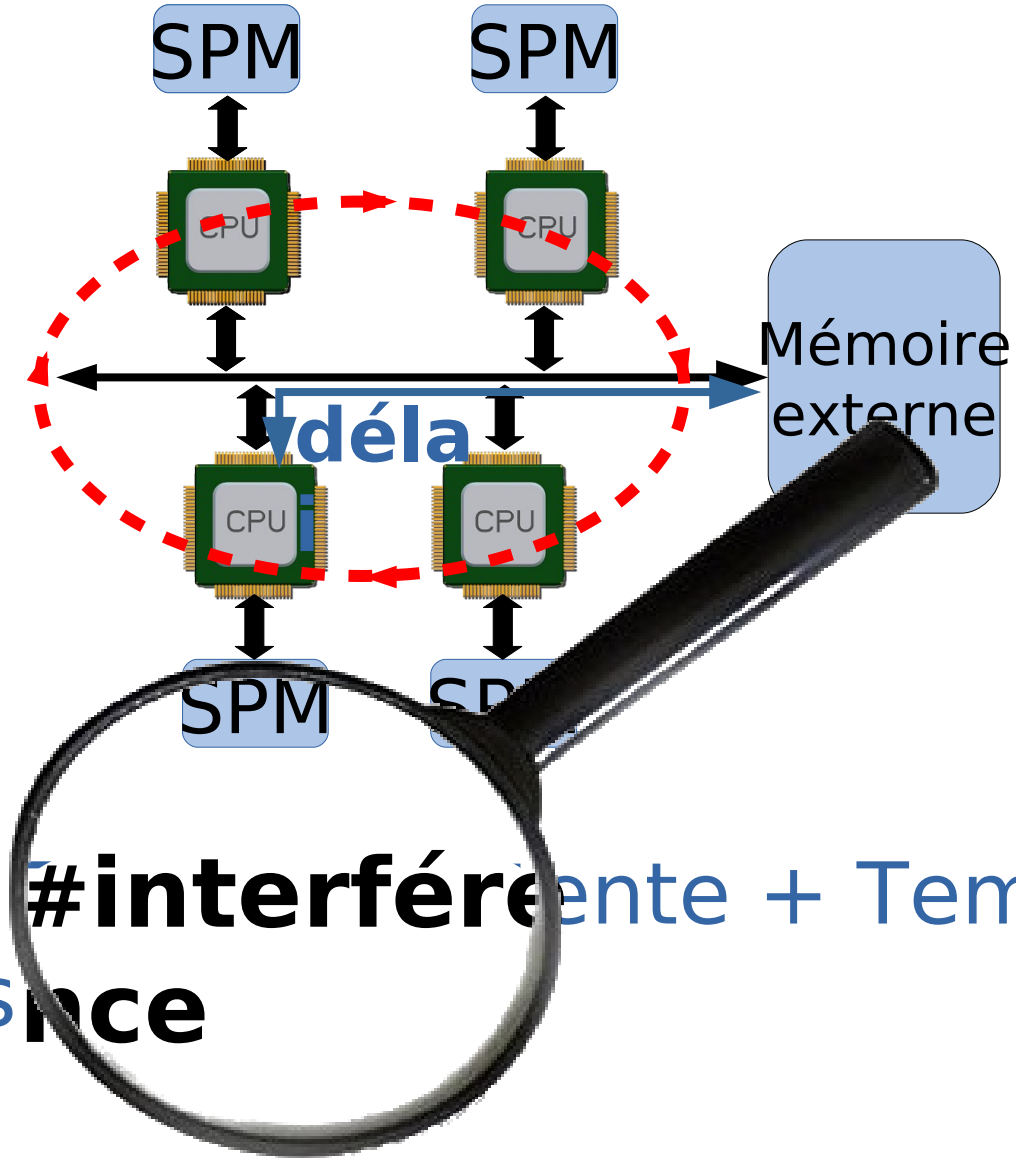
- **En isolation des autres tâches**
- **Utilisation des méthodes statiques de l'état de l'art**

Pire temps d'exécution pour les phases: *acquisition/restitution*

- **Utilisation d'un modèle du bus**
- **Nécessite de déterminer la contention (nombre d'interférences)**

# Calcul du coût de communication

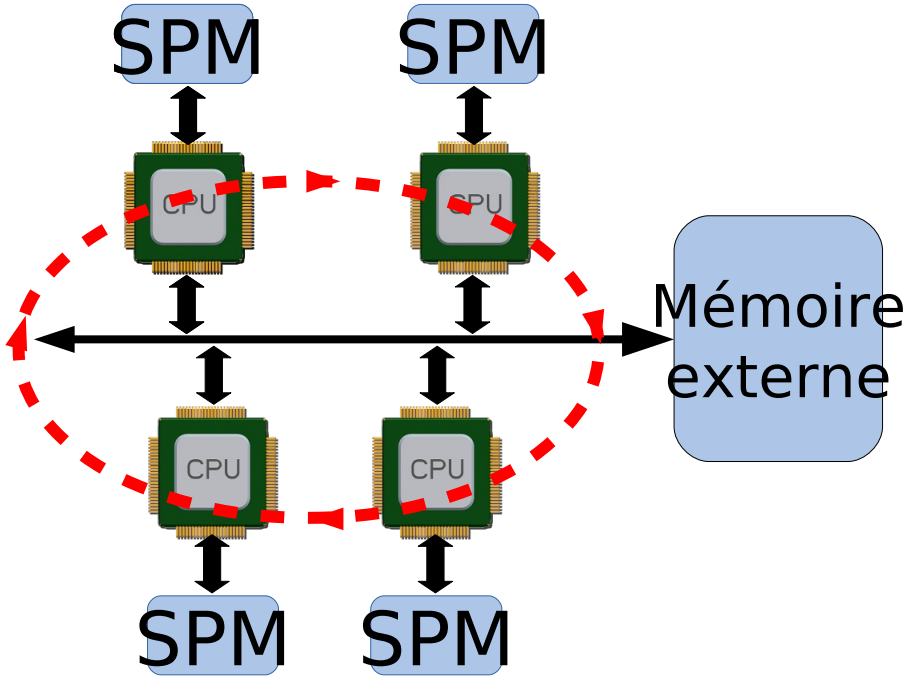
Arbitrage FAIR-Round-Robin




délai = #interférente + Temps d'accèsnce

# Calcul du pire degré d'interférence

Pire degré d'interférences:  NBCœurs - 1



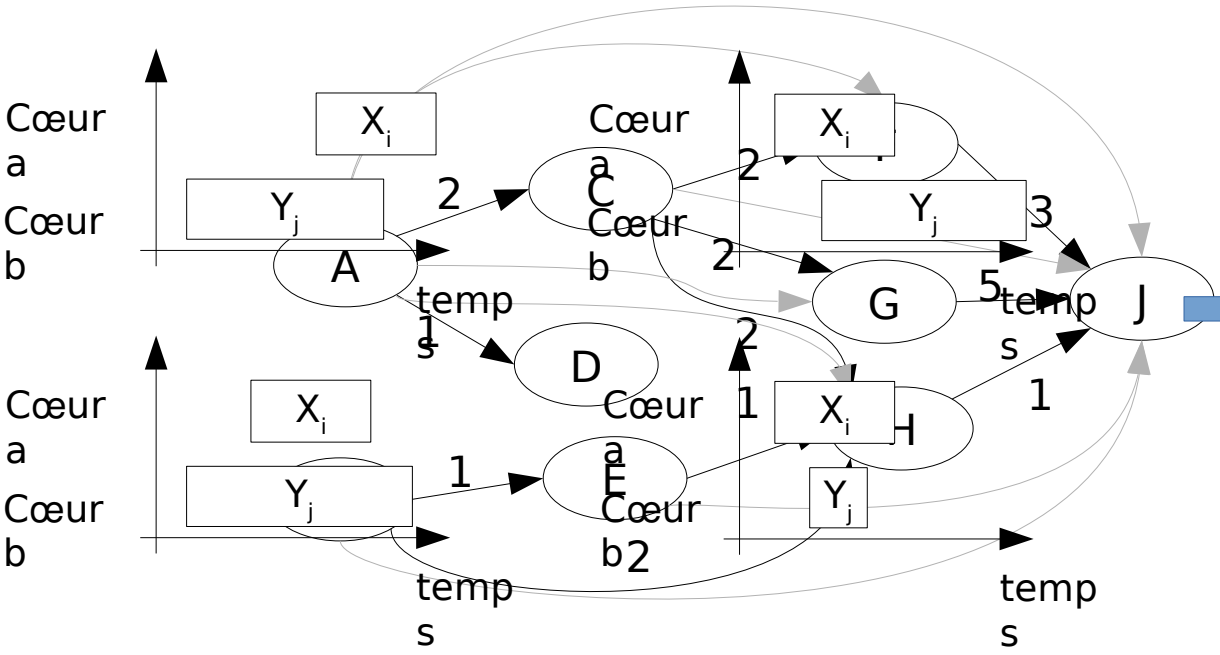
# Calcul du degré d'interférence à l'ordonnancement

## Calcul du degré d'interférence

- **Nombre de cœurs ayant des phases chevauchantes**

Connaissance du logiciel

- **Seules les tâches concurrentes se chevauchent**
  - Fermeture transitive du DAG avec l'algorithme de Warshall
- **Chevauchement des phases *i* et *j***




Connaissance de l'ordonnancement

$recouvre(X_i, Y_j)$

# Calcul du degré d'interférence à l'ordonnancement

Connaissance de l'ordonnancement

Nombre d'

$$interférence_i^X = \sum_{p \in P} \Big[ \bigvee_{j \in T \vee concurrent(i,j)} ((recouvre(X_i, acquisition_j) \vee recouvre(X_i, restitution_j)) \wedge placé(j, p)) \Big]$$

où X peut représenter une acquisition ou res

Connaissance du logiciel

# Technique d'implémentations

- **Programmation linéaire en nombre entiers (ILP)**
  - Solution exactes
  - Lent
- **Forward List Scheduling (FLS)**
  - Solutions approximées
  - Rapide

# Programmation linéaire en nombre entiers (ILP)

## Rappel

- **Fonction objectif (maximiser ou minimiser)**
- **Ensemble de variables et d'inégalités linéaires (contraintes)**
- **Formalisation non-ambigüe**

# Notre implémentation ILP consciente de la contention

- **Objectif: minimiser la taille de l'ordonnancement**
- **Contraintes de placement**
  - Une tâche est placée sur un et un seul coeur
- **Contraintes d'ordonnancement**
  - Pas de chevauchement de phases de même type sur le même coeur
  - Précédences entre phases et entre tâches
- **Calcul des coûts de communication**

# Forward List Scheduling (FLS)

Rappel

- Heuristique
- Solutions approximées
- Pas de remise en question des décisions passées
- Passage à l'échelle

# Notre implémentation FLS consciente de la contention

- **Ordonner les tâches**
- **Essayer de placer une tâche le plus tôt sur chaque coeur**
  - Pas de chevauchement de phases de même type sur le même coeur
  - Précédences entre phases et entre tâches
  - Calcul des coûts de communication
- **Garder le placement/ordonnancement avec la longueur minimale**
- **Recommencer avec une autre tâche**

Problèmes cachés ici, mais détaillés dans la thèse:

- Générer un ordonnancement sans contention
- Calculer les interférences modifie les décisions passées

# Dégradation de l'heuristique comparé à l'ILP

- **Générateur de graphes de tâches (TGFF)** *Dick'1998*
- **Cas de tests synthétiques, large ensemble de topologies**
- **Paramètres:**

  - Nombre de graphes : 50
  - Nombres de tâches : [5 ; 69]
  - Nombre de coeurs: {2, 4, 8, 12}
  - $T_{créneau}$: [1 ; 10]
  - Timeout : 11h

# Dégradation de l'heuristique comparé à l'ILP

Dégradation

- **Minimal : 0 %**
- **Moyenne : 2 %**
- **Maximal : 20 %**

Temps de génération moyen

- **ILP : 4,2 heures**
- **FLS : 1,5 secondes**

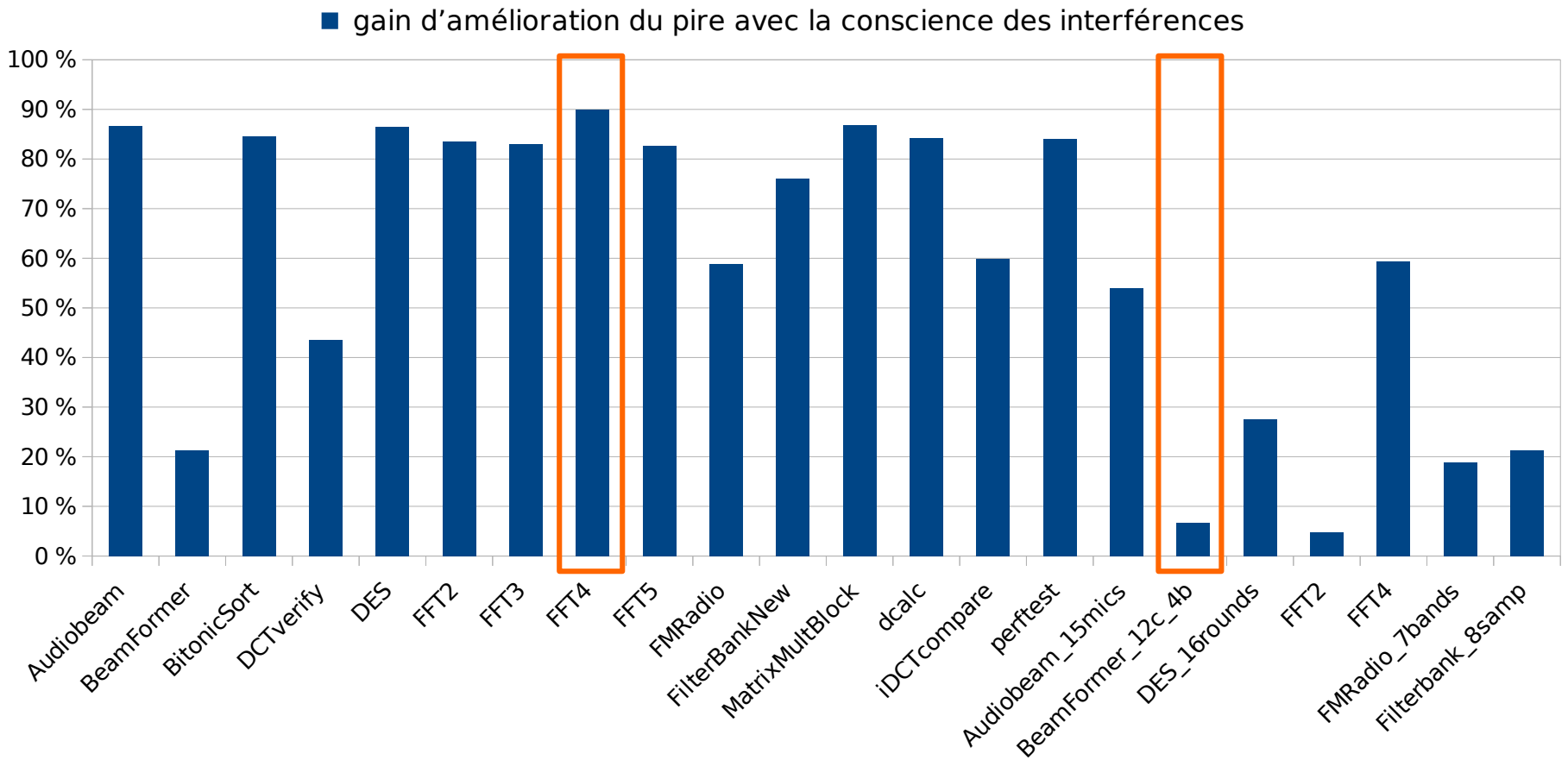
# Gain d'amélioration du pire cas (NBCores-1)

- **Base : contention pessimiste**
- **Suite de cas de tests STR2RTS**
- **Cas de tests réels**
- **Graphe de type fork-join**
- **Flux de données fixes → WCET connu à la compilation**

*Rouxel'2017*

- #Tâches : [7;201]
- Largeur max : [2;36]
- Nombre de coeurs: {2,4,8,12}
- $T_{créneau}$: [1;10]

# Gain d'amélioration du pire cas (NBCores-1)




**59% en moyenne**

# Conclusions contribution 1

Objectif

- **Améliorer l'estimation pessimiste des interférences à l'ordonnancement hors-ligne**

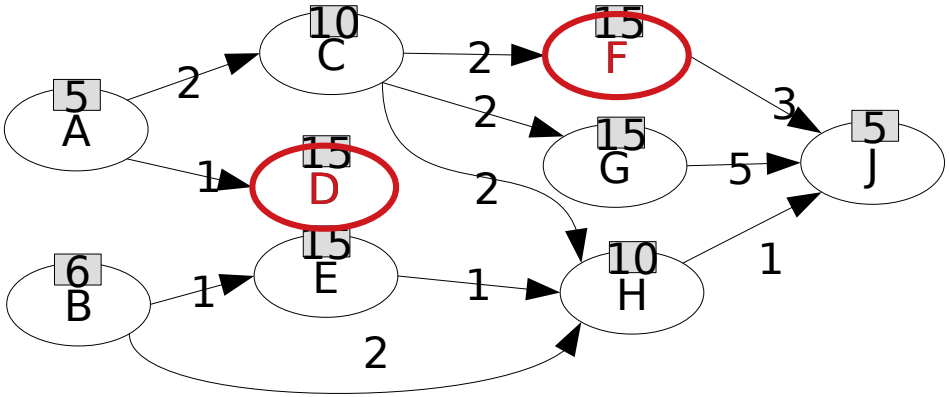
Leçons apprises

- **Méthode de calcul des interférences**
- **Faible dégradation de l'heuristique**
- **Amélioration du pire cas**

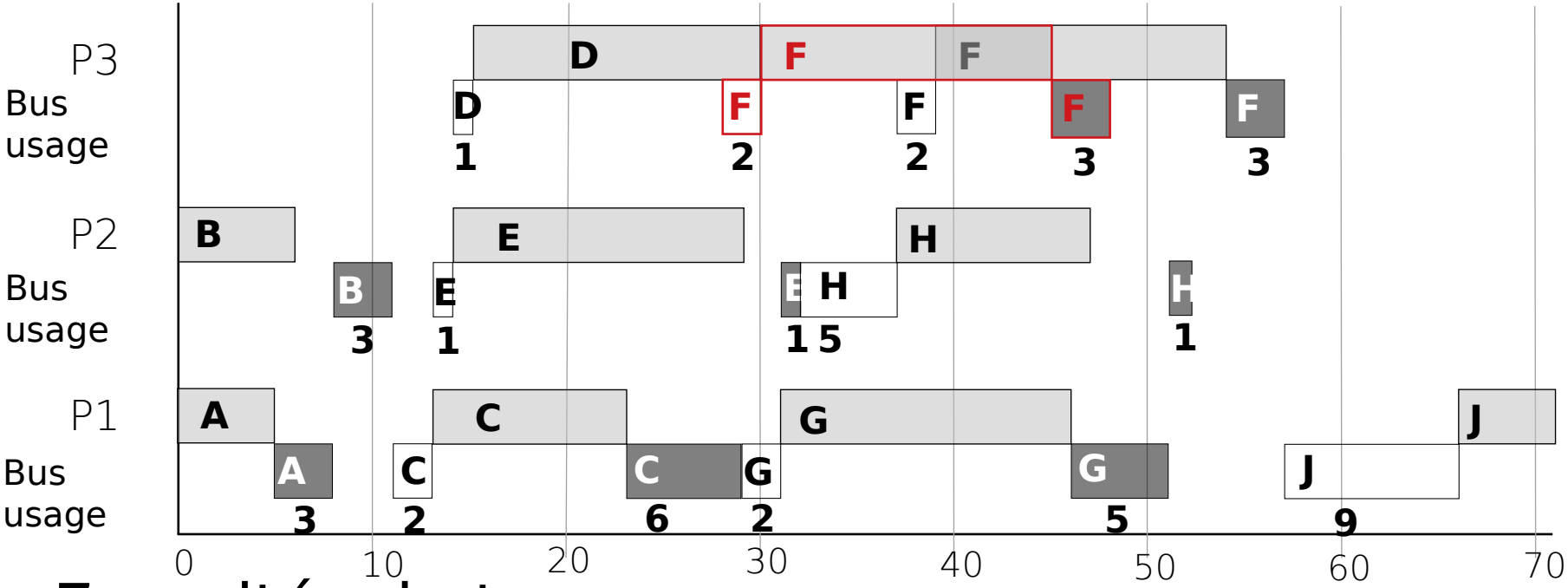
Limite

- **Gain similaire aux ordonnancements sans contention**

# Zoom sur un ordonnancement sans contention



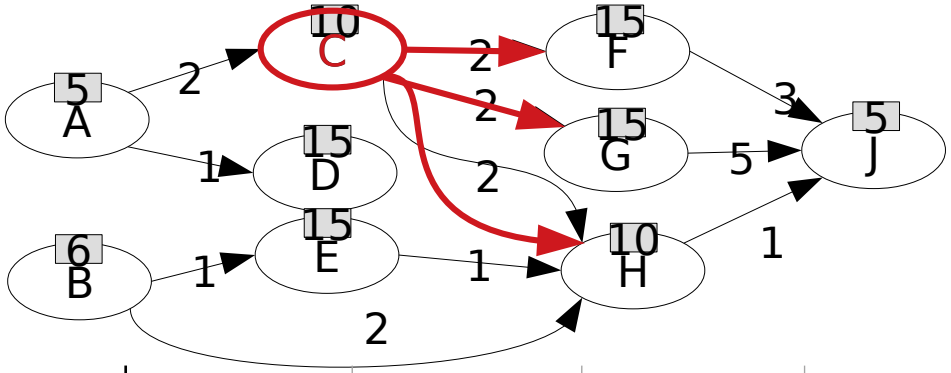

Masquer les communications



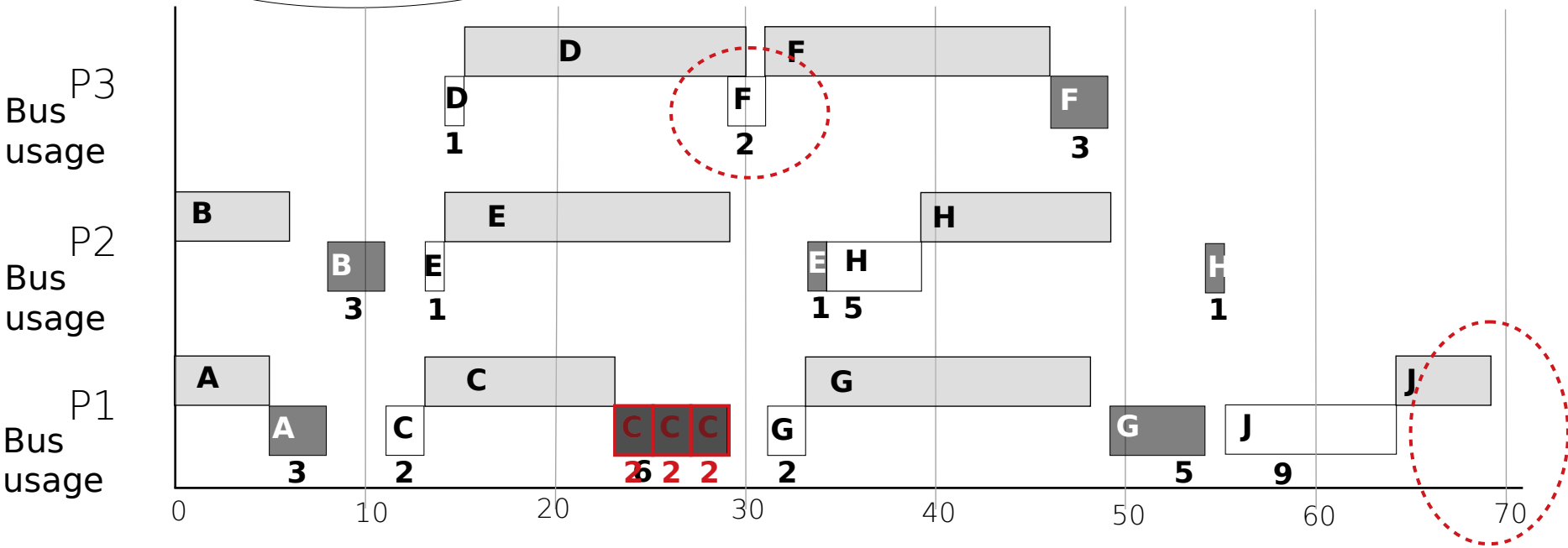

7 unités de temps

1

# Masquer la latence des communications



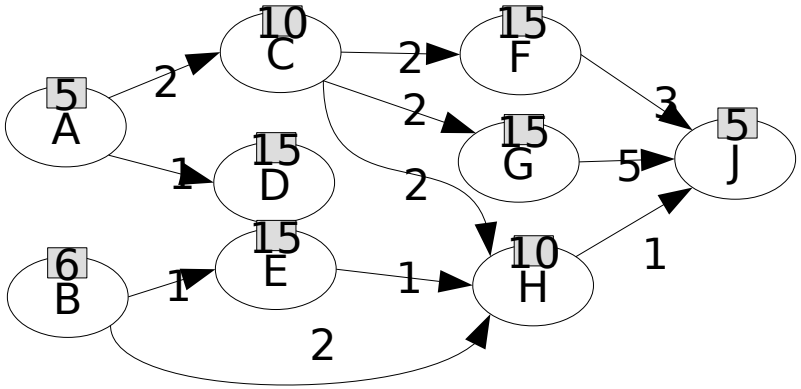

Fragmenter les communications



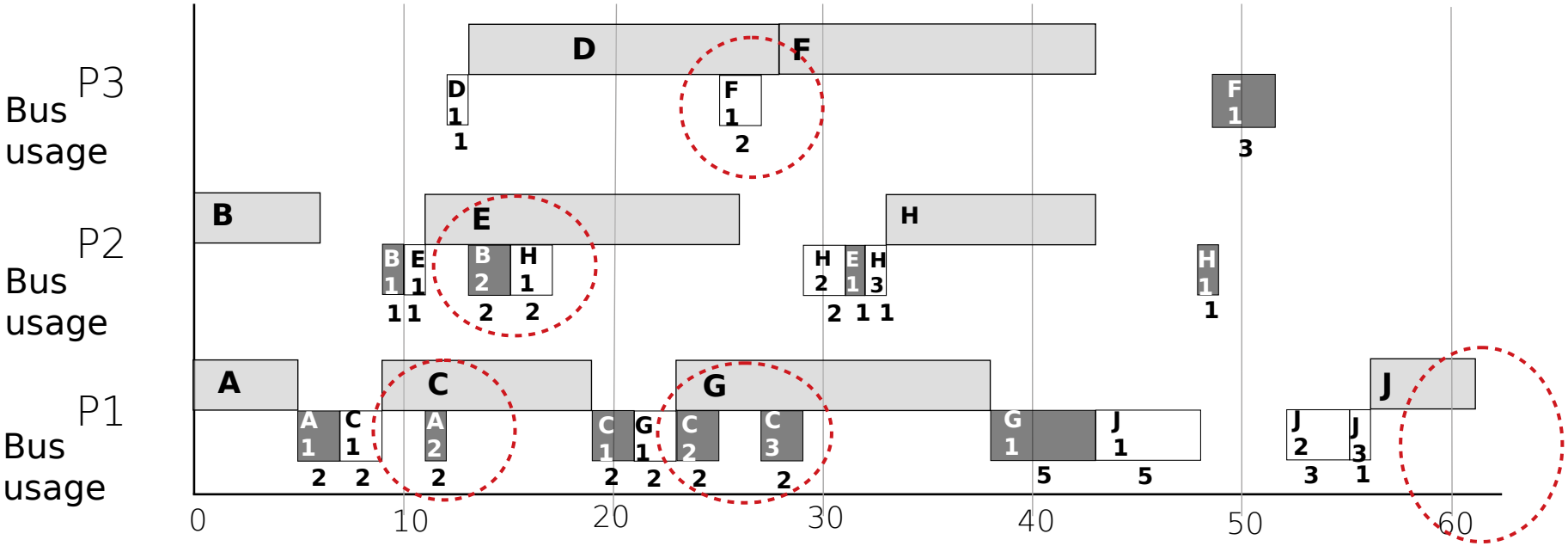

~~7~~ 1

69unités de temps

# Fragmenter les phases de communication



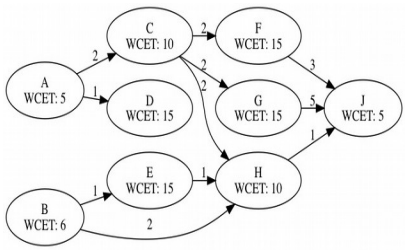

**Gain : 20%**




7 69 6 unités de temps

# Contributions (2): Cacher les délais de communication

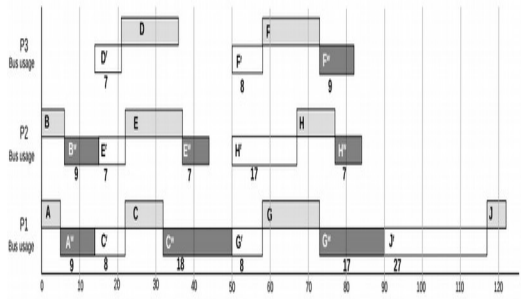
Graphe d'application



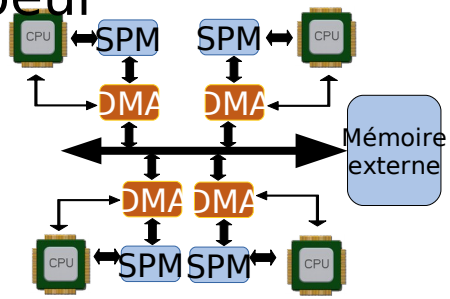

Pire temps d'exécution

- Traitement
- Communication

Ordonnancement sans contention & schéma d'allocation en




Architecture multi-coeur




Stratégie d'ordonnancement

- ILP
- Heuristique

Masquer les communications

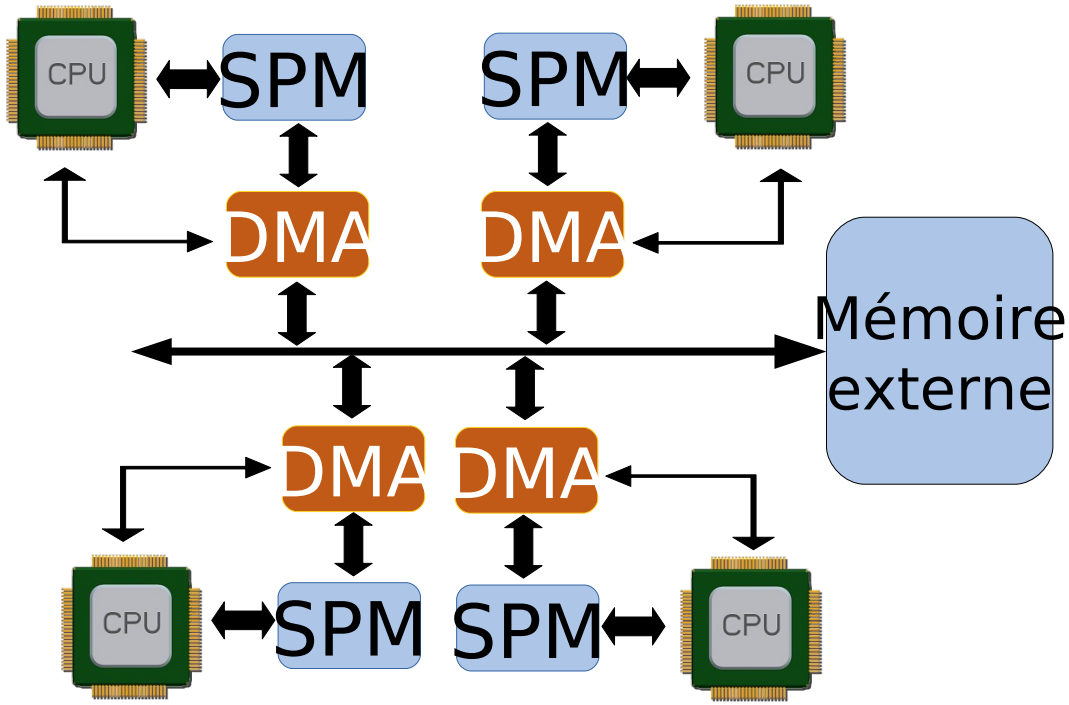
Fragmenter les communications

- Hors-ligne
- Partitionné
- Non-préemptif

*Rouxel'2019 (en cours de validation)*

# Support matériel

- Moteur d'accès direct à la mémoire (DMA)
- SPM à double accès



# ILP masquant les communications fragmentées

- **Objectif: minimiser la longueur de l'ordonnancement**
- **Contraintes de placement**
- **Contraintes d'ordonnancement**
  - Aucun chevauchement des phases de communication

- **Contraintes de réservation de mémoire en SPM**
  - Une phase de communication est allouée à une et une seule région
  - Aucun chevauchement des temps de réservation

# FLS masquant les communications fragmentées

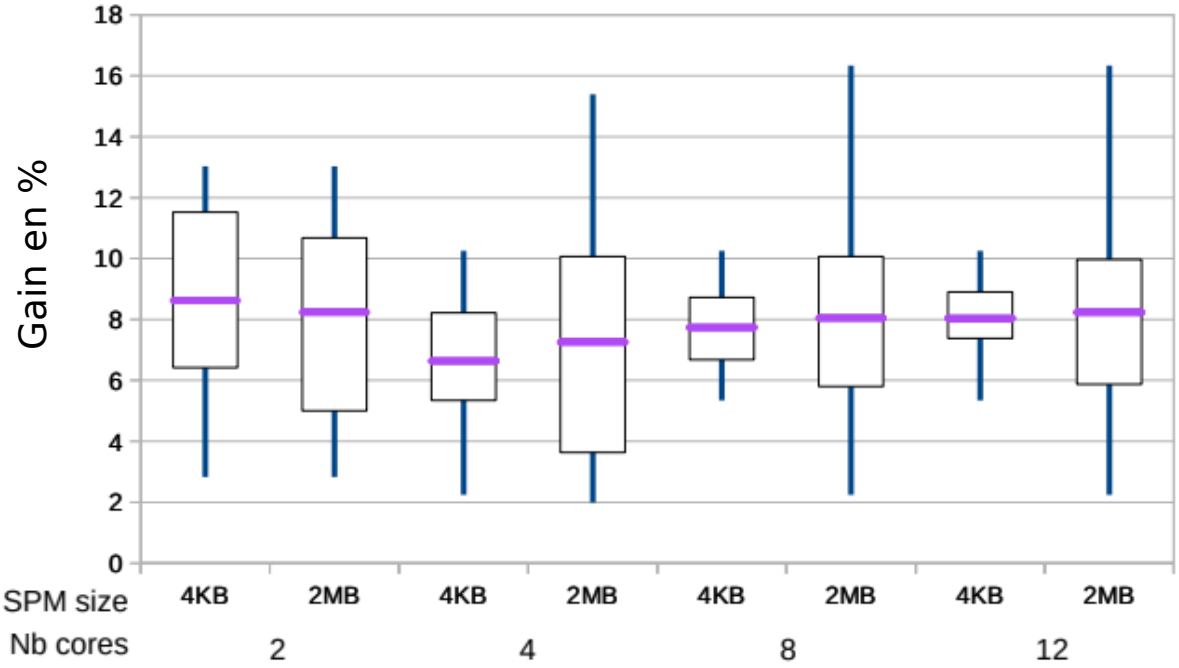
- **Ordonner les tâches**
- **Essayer de placer une tâche le plus tôt possible sur chaque coeur**
  - Aucun chevauchement des phases de communication

- **Allouer les régions de la SPM**
  - Chercher une région déjà allouées mais non réservée sur la période d'utilisation de la tâche
  - Si non trouvée, créer une nouvelle région
  - Si pas assez d'espace mémoire, lever une exception

- **Garder le placement/ordonnancement avec la longueur d'ordonnancement minimale**
- **Recommencer avec une autre tâche**

# Gain observé sur les benchmarks synthétiques

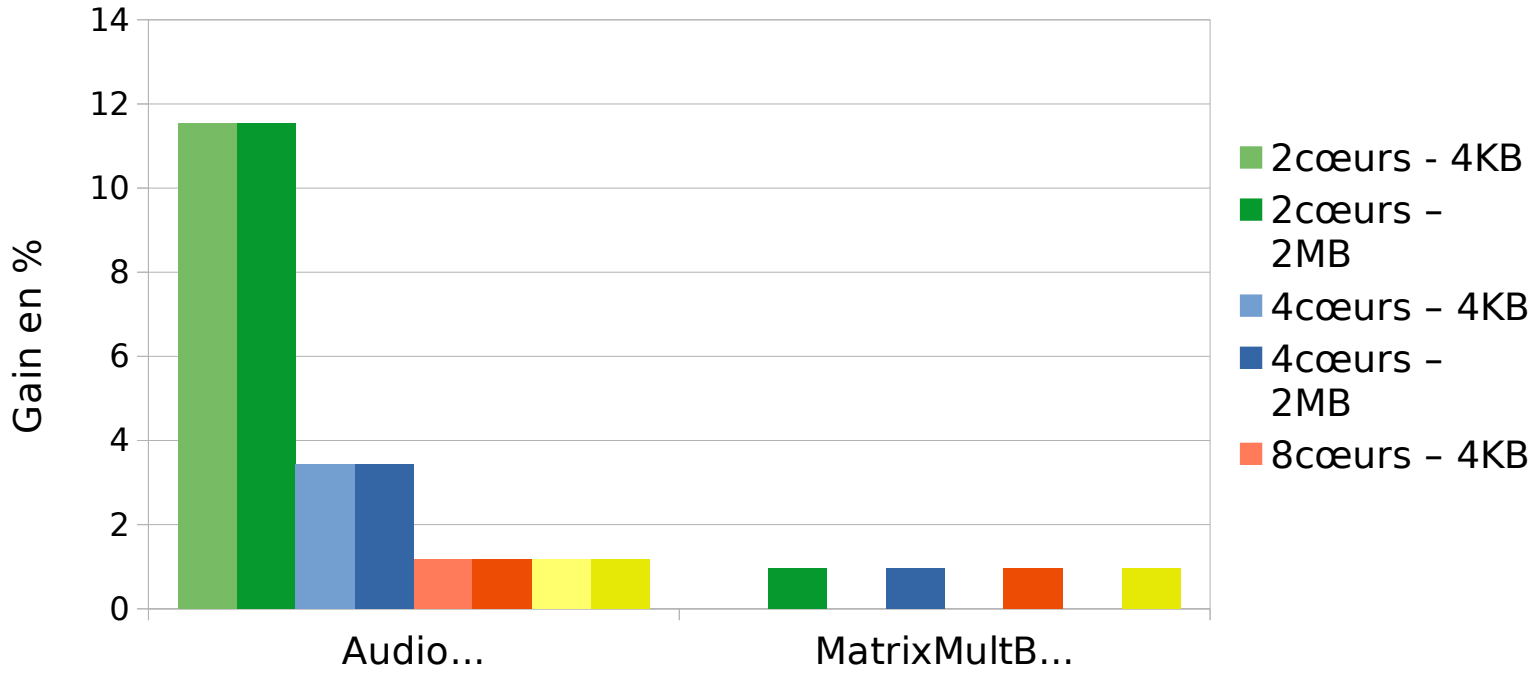
- Base : non-masqué et non-fragmenté
- Cas de tests synthétiques (TGFF) *Dick'1998*




**8% en moyenne**

# Gain observé sur les benchmarks réels

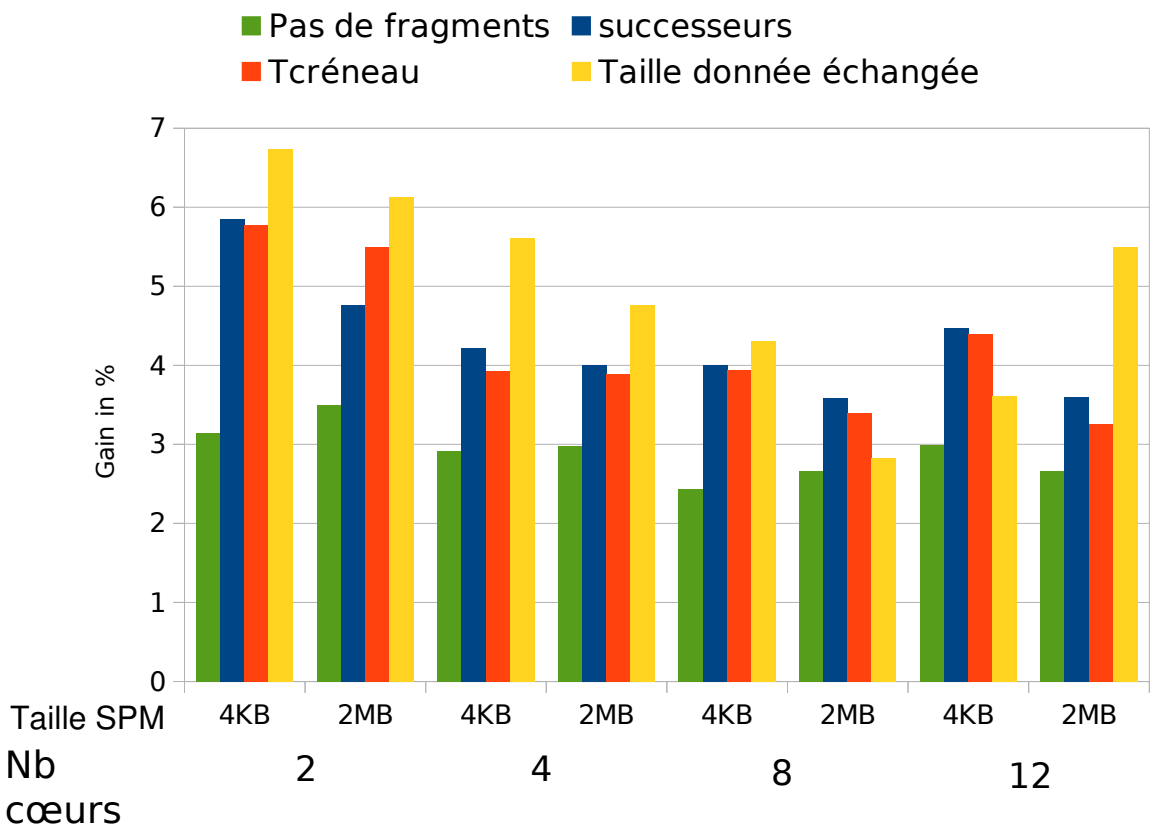
- **Base : non-masqué et non-fragmenté**
- **Cas de tests réels avec STR2RTS** *Rouxel'2017*




**4% en moyenne**

# Choix de la taille des fragments

- **Base : communications bloquantes**




*Rouxel'2017*

Temps de génération m

- **Pas de fragments : < 1**
- **successeurs : < 1 sec.**
- **$T_{créneau}$ : > 5 min.**
- **TDE : < 4 min.**

# Implémentation de l'ordonnancement

- Kalray MPPA 256 Bostan
- 1 grappe, 16 coeurs
- 16 bancs mémoires privatisés
- 8 bus, arbitrage par priorités fixes

- Limité à 8 cœurs
- 1 coeur réservé: DMA applicatif
- WCET & coûts de communication mesurés

Gain d'amélioration du pire cas, base : *Pas de fragments*

- Par successeurs : 36 %
- Par $T_{créneau}$ : 22 %
- Par taille de données : 12 %

# Conclusions contribution 2

Objectif

- **Réduire la longueur d'ordonnancements sans contention**

Leçons apprises

- **Fragmenter les communications**
- **Amélioration du pire cas**

Limite

- **Graphe de type fork-join**

# Conclusions

## Objectif

- **Réduire l'impact des communications sur l'ordonnancement hors-ligne**

## Méthodes

- **Réduire les délais de contention**

  **avec** Connaissance du logiciel **&** Connaissance de l'ordonnancement

- **Réduire la longueur d'ordonnancement**

  **avec** Masquer les communications **&** Fragmenter les communications

## Résultats

- **Gain en considérant les interférences**
- **Gain en augmentant les opportunités d**

# Perspectives

- **Étendre vers les réseaux sur puces**
- **Autoriser les communications de SPM à SPM**

- **Étendre le calcul des interférences**
  - au modèle non-AER
  - au modèle pré-emptif
- **Affiner le calcul des interférences**

- **Améliorer l'implémentation sur le Kalray MPPA**

Merci pour votre attention

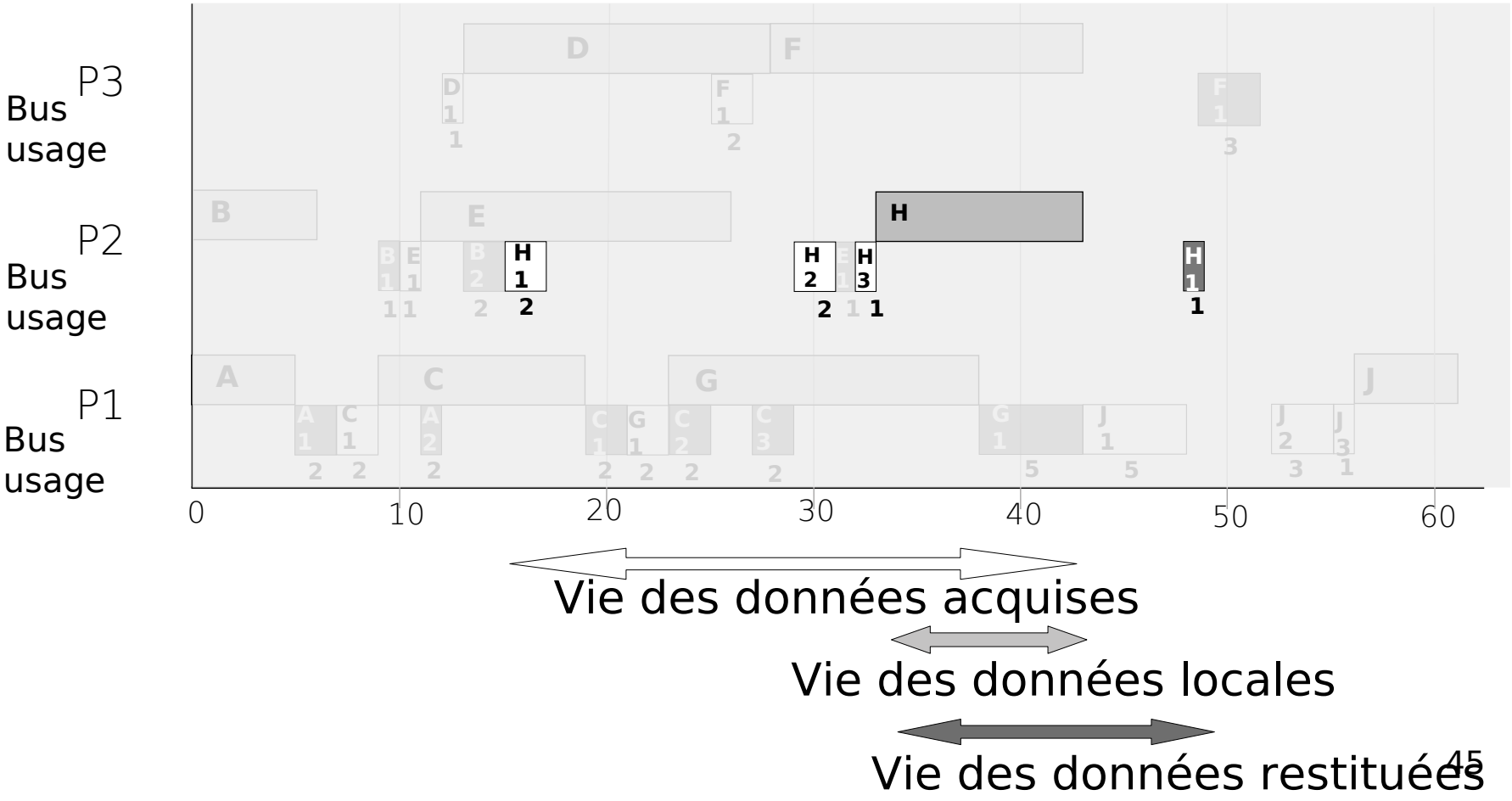
# Liste des publications


- Benjamin Rouxel, Steven Derrien, and Isabelle Puaut. Tightening contention delays while scheduling parallel applications on multi-core architectures. ACM Transactions on Embedded Computing Systems (TECS), 2017, vol. 16, no 5s, p. 164.
- Benjamin Rouxel, Steven Derrien, and Isabelle Puaut. Overlapping communications and computations in SPM-based multi-core architectures. Submitted to RTAS'2019.
- Benjamin Rouxel, and Isabelle Puaut. STR2RTS: Refactored StreamIT benchmarks into statically analyzable parallel benchmarks for WCET estimation & real-time scheduling. In : OASIcs-OpenAccess Series in Informatics. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
- Martin Schoeberl, Benjamin Rouxel, and Isabelle Puaut, « A Time-predictable Branch Predictor », Symposium on Applied Computing (SAC), 2019.
- Damien Hardy, Benjamin Rouxel, and Isabelle Puaut, « The Heptane Static Worst-Case Execution Time Estimation Tool », in: 17th International Workshop on Worst-Case Exe- cution Time Analysis (WCET 2017), vol. 8, Schloss Dagstuhl-Leibniz-Zentrum fuer Infor- matik, 2017, pp. 1–812.

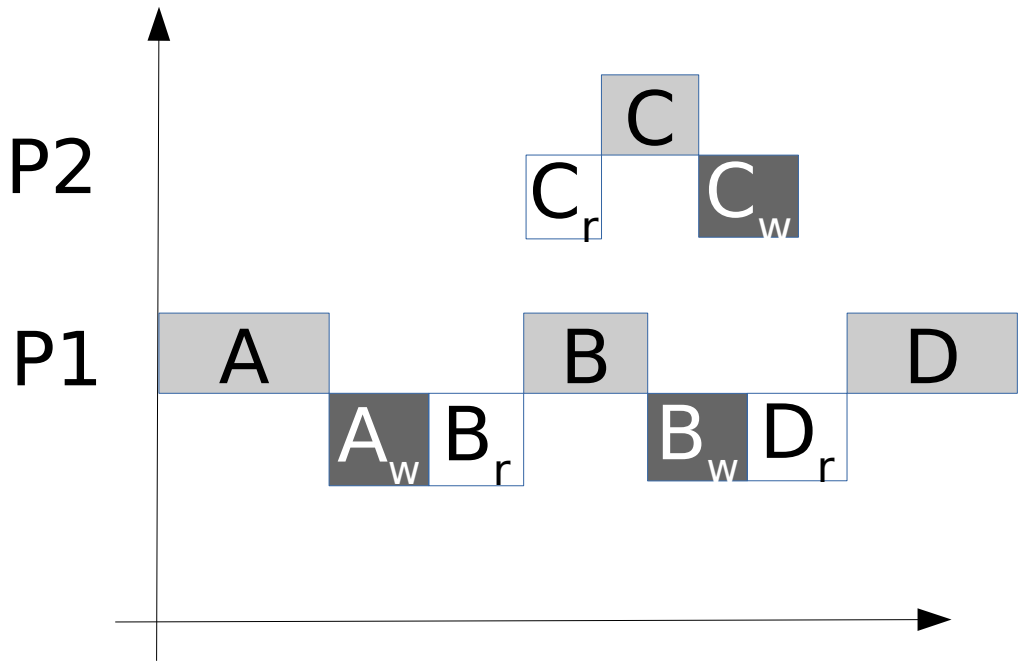
# Schéma d'allocation mémoire en SPM

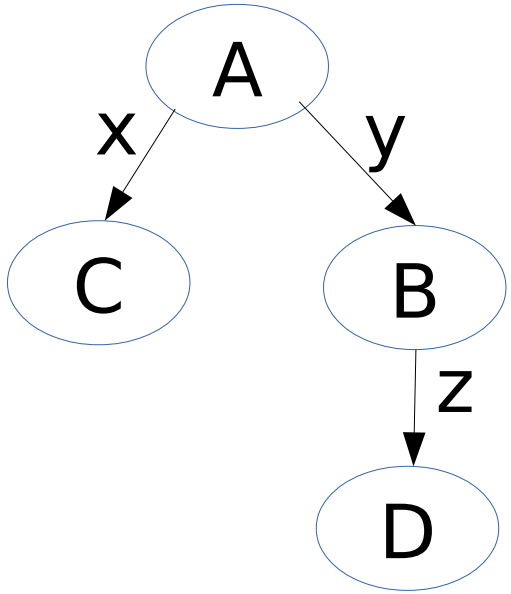
- **Diviser la SPM en régions**
- **Allocation des phases de communication aux régions**
- **Assurer l'intégrité des données**

*Kim'2014*

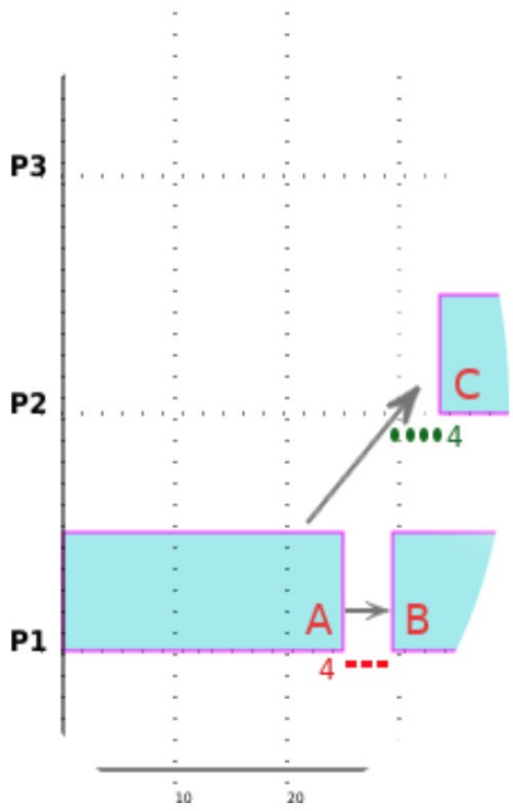


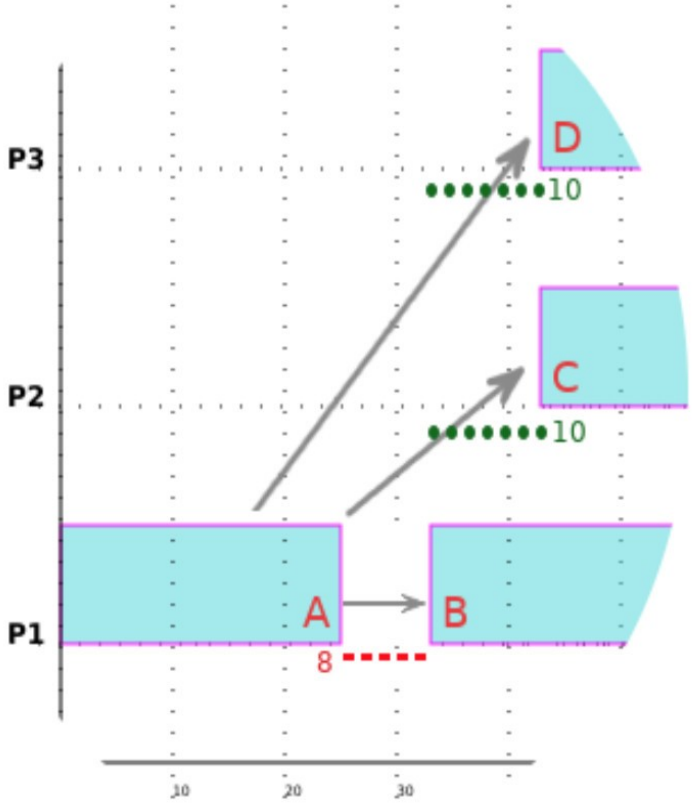
# Construction de l'ensemble des tâches associées



# Forward List Scheduling: adjusting the schedule

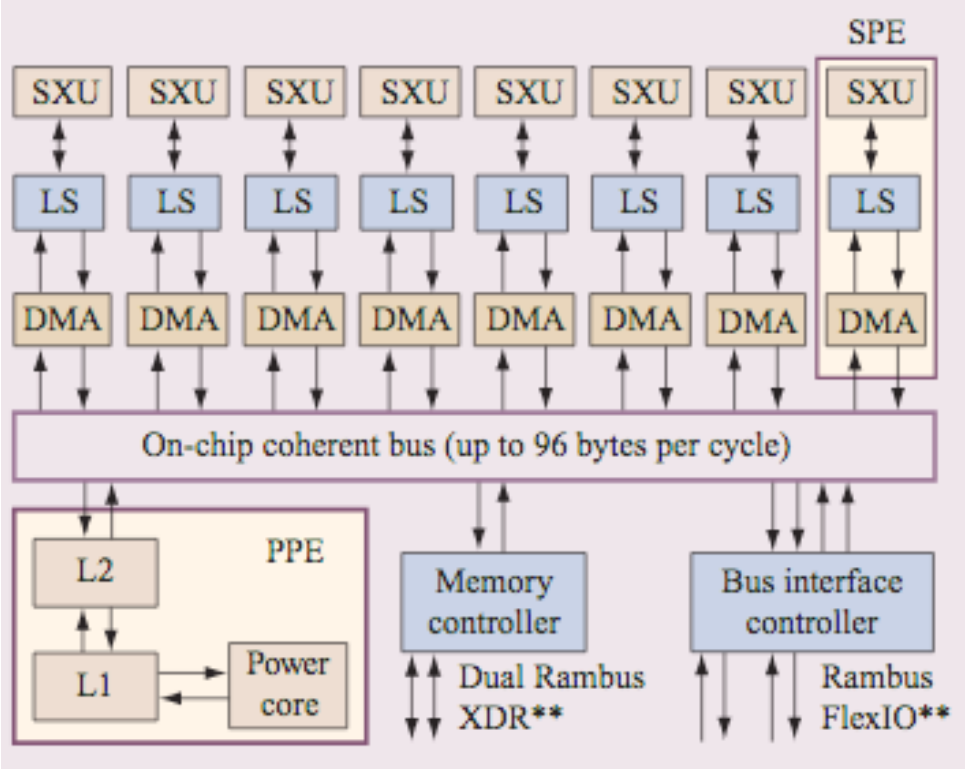


# Pourquoi c’est difficile d’avoir des interférences précises

# Multi-DAG, multi-période → Comment serait étendue la notion de makespan dans le cas d'application multi-périodiques ?

# Cell processor




**SPE : Synergistic Processing**
**SXU : Synergistic Execution**
**LS : Local Storage**

James Kahle, Michael Day, Peter Hofstee, Charles Johns, Theodore Maeurer, and David Shippy. Introduction to the Cell multiprocessor. IBM Journal of Research and Development, 49(4/5):589–604, September 2005.

# Complexité heuristique

```
1  Function ListSchedule(G = (T, E), P)
2      Elist ← BuildListElement(G)
3      Qready ← TopologicalSortNode(Elist)
4      Qdone ← ∅
5      schedule ← ∅
6      while elt ∈ Qready do
7          Qready ← Qready \{t}
8          Qdone ← Qdone ∪{t}
9          if elt is a read fragment ∨ elt is a write fragment then
10             ScheduleElement(Qdone, elt, null)
11         else if elt is an exec phase then
               /* tmpSched contains the best schedule for the current task
12             tmpSched ← schedule with makespan = ∞
13             foreach p ∈ P do
14                 copy ← schedule
15                 ScheduleElement(Qdone, elt, p)
16                 tmpSched ← min_makespan(tmpSched, copy)
17             schedule ← tmpSched
18         if all fragments and exec phase of the task i containing elt are in Qdone then
19             AssignRegion(schedule, Qdone, τ_i^e, SS_t + CS_t, 0, infinity)
20             foreach f ∈ F_i^r do
21                 AssignRegion(schedule, Qdone, f, D^r_(i,f), ρ^r_(i,f), ρ_i^e + C_i)
22             foreach f ∈ F_i^w do
23                 AssignRegion(schedule, Qdone, f, ρ_i^e, ρ^w_(i,f) + DELAY^w_(i,f))
24     return schedule
```

**Polynomial, degré 2**

O(|P|)

O(|T|)

O(|T|)

O(|R|)

O(|F^r_i|*|R|)

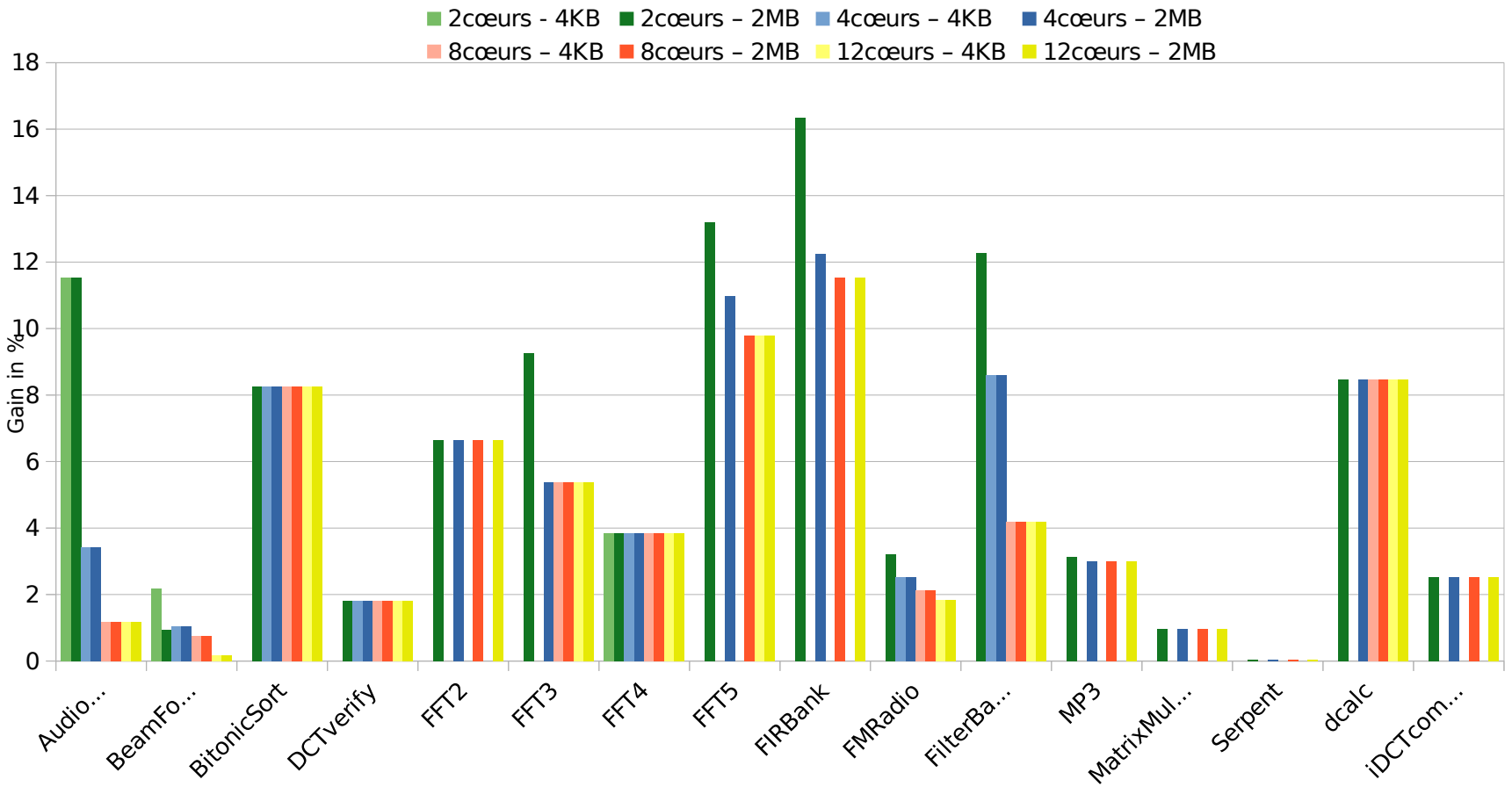
O(|F^w_i|*|R|)

# StreamIT description

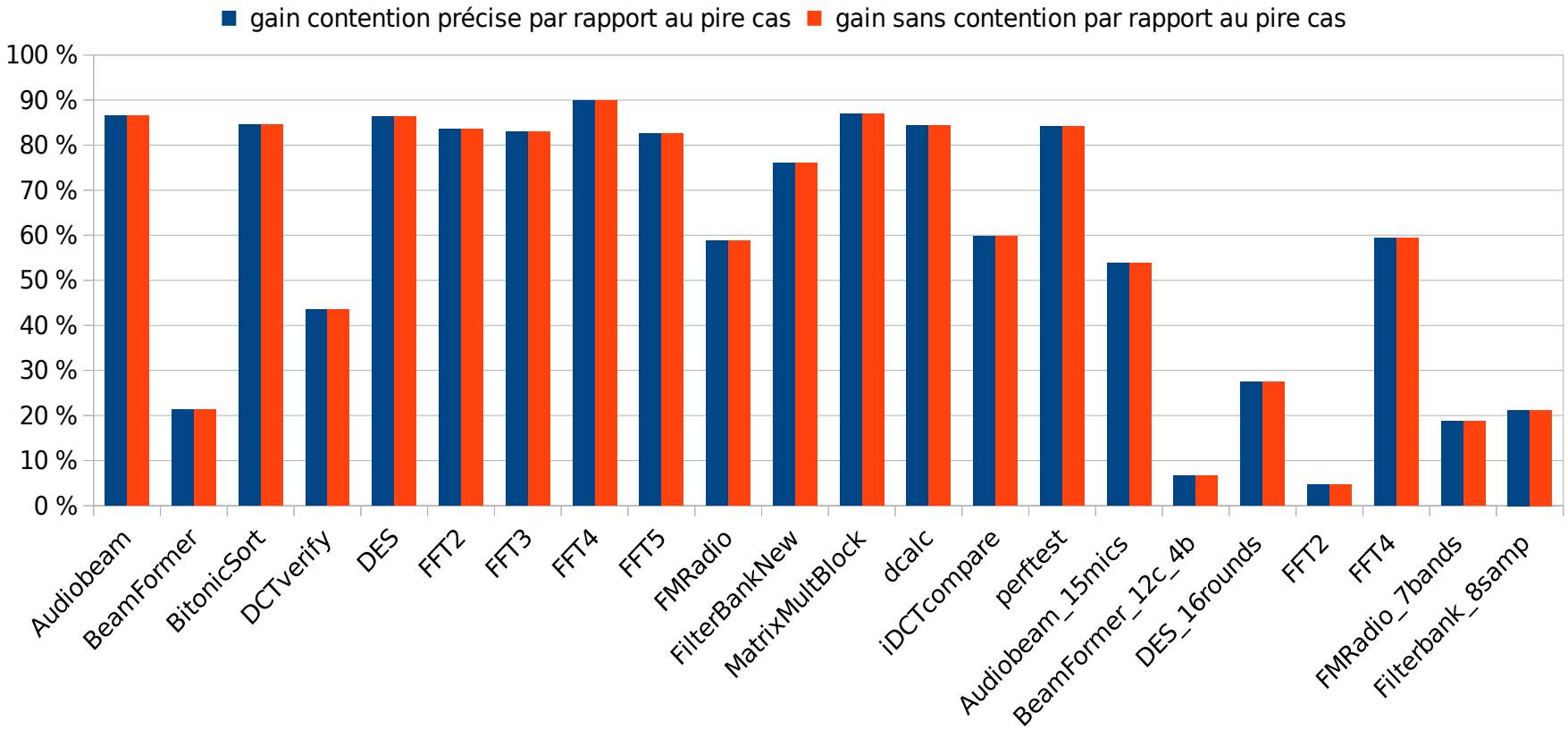
| Name | #Tasks | Width |
|---|---|---|
| 802.11a | 113 | 7 |
| audiobeam | 20 | 15 |
| beamformer | 56 | 12 |
| compCnt | 31 | 8 |
| dct_comp | 13 | 3 |
| dct_verif | 7 | 2 |
| fft2 | 26 | 2 |
| fft3 | 82 | 16 |
| fft4 | 10 | 2 |
| fft5 | 115 | 16 |
| fhr | 29 | 7 |
| filterbank | 53 | 8 |

| Name | #Tasks | Width |
|---|---|---|
| fm | 67 | 20 |
| hdtv | 150 | 24 |
| merge | 31 | 8 |
| mp3 | 116 | 36 |
| mpd | 201 | 5 |
| mpeg2 | 43 | 3 |
| nokia | 178 | 32 |
| perftest4 | 16 | 4 |
| tconvolve | 75 | 36 |
| tde | 15 | 12 |
| vocoder | 115 | 17 |

# Gain observé sur les benchmarks réels



# 2) Gain d'amélioration du pire cas (NBCores-1)




**Gain similaire**

# Gestion des ressources à l'ordonnancement