Partitionnement Déterministe pour Résoudre les Problèmes de Programmation Par Contraintes en utilisant le Framework Parallèle Bobpp

Tarek Menouer et Bertrand Le Cun

Université de Versailles Saint-Quentin-en-Yvelines, France
Laboratoire PRISM

25 Avril 2014 | Neuchâtel, Suisse
ComPAS'2014

Outline

1. Introduction
2. Parallélisation Non Déterministe
3. Parallélisation Déterministe
4. Expérimentation
5. Conclusion & Perspectives

Outline

1. Introduction
2. Parallélisation Non Déterministe
3. Parallélisation Déterministe
4. Expérimentation
5. Conclusion & Perspectives

Introduction

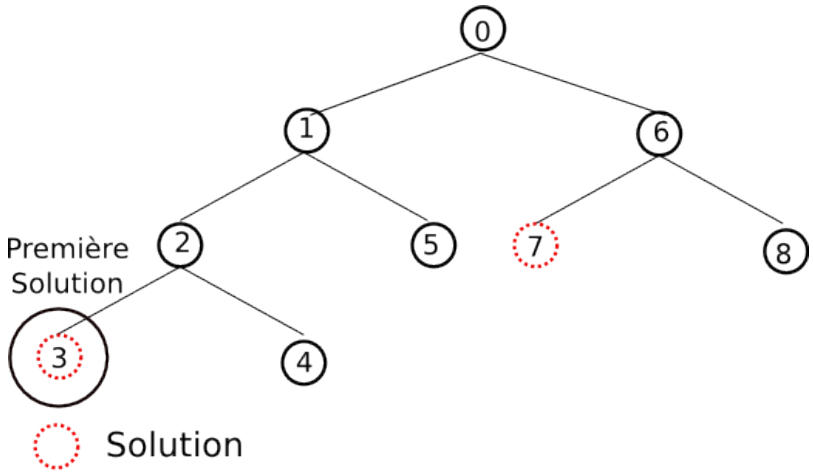
Programmation Par Contraintes (PPC)

- Résoudre des problèmes combinatoire de grande taille
- Problèmes de Satisfaction de Contraintes : Toutes les solutions ont la même évaluation
  - N-Queens, Sudoku, ...
- Problèmes d'Optimisation sous contraintes : Toutes les solutions n'ont pas la même évaluation
  - Affectation Quadratique, Golomb Ruler, ...
- Solveur : Google OR-Tools, Gecode, CHOCO, ...

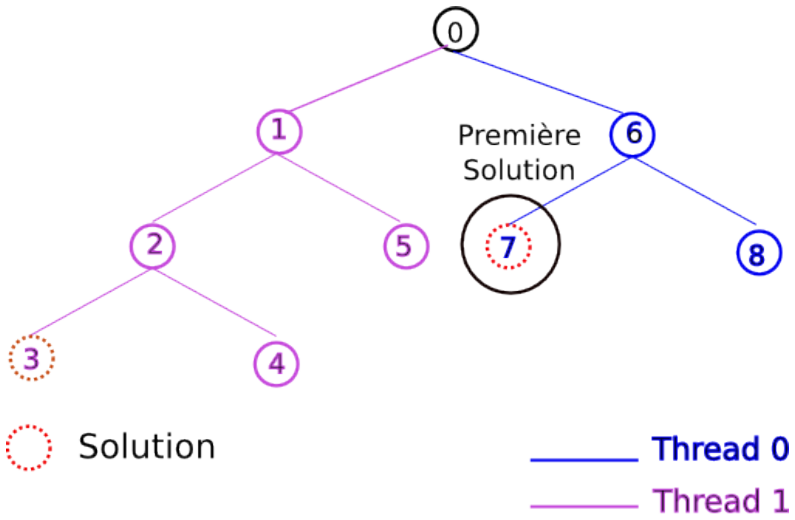
Le solveur OR-Tools

- Solveur open source séquentiel développé en C++
- Trouver la première solution (optimale)

Parcours des arbres de recherche



Parcours Séquentiel



Parcours Parallèle

- La première solution trouvée en séquentiel est la solution représentée par le nœud 3
- La première solution trouvée en parallèle est la solution représentée par le nœud 7
- Solution trouvée en séquentielle $\neq$ Solution trouvée en parallèle

Déterminisme vs non déterministe

Déterminisme ?

- Deux exécutions du même solveur sur une même instance donnée, retournent toujours les mêmes solutions

Avantages

✓ Facilite les tests et débugage d'un code

✓ Demande des utilisateurs de solveurs pour avoir une stabilité dans la solution retournée

Inconvénient

✗ Perte en performance par rapport à une parallélisation non déterministe ! ! !

Introduction

Le déterminisme pour le parcours séquentiel

- Nous considérons un algorithme séquentiel déterministe :
  - Ordonnancement du parcours est déterministe
  - Pas d'aléatoire sur le choix des variables dans le branchement
- La solution retournée est la première solution (optimale) trouvée
- Recherche en profondeur d'abord : la solution (optimale) est celle la plus à gauche

OR-Tools séquentiel est déterministe

Pour le parallélisme

- Ordonnancement parallèle $\approx$ Ordonnancement séquentiel déterministe
  - Pour s'en rapprocher, maintenance d'une connaissance globale

Outline

1. Introduction
2. Parallélisation Non Déterministe
3. Parallélisation Déterministe
4. Expérimentation
5. Conclusion & Perspectives

Parallélisation non déterministe

Principe

- Partitioner l'arbre de recherche en un ensemble de sous-arbres dynamiquement et pendant l'exécution de l'algorithme de recherche en utilisant Bobpp

Bobpp

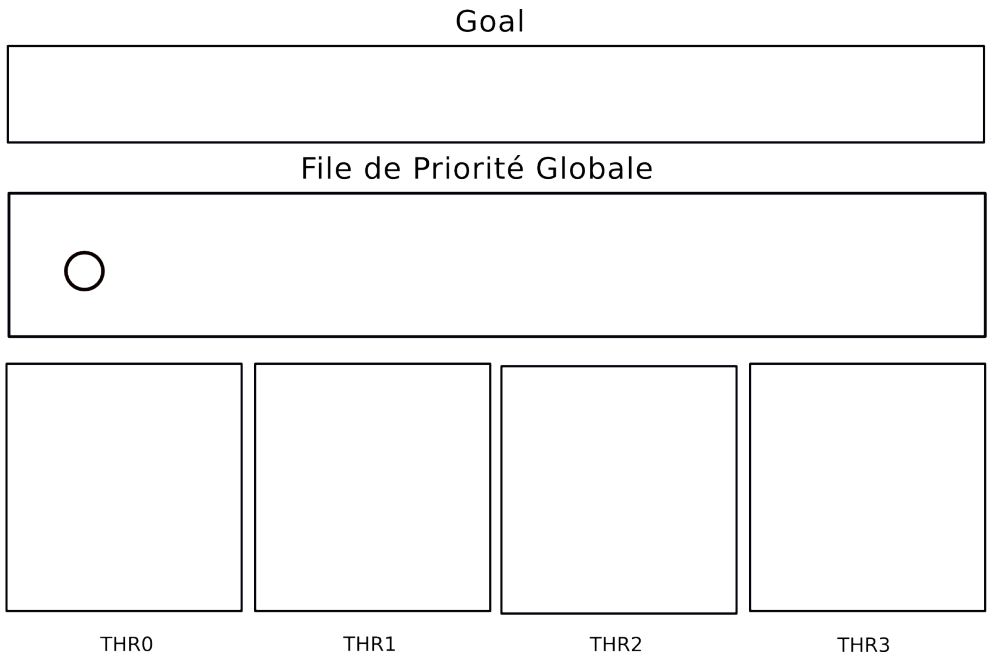
- Framework parallèle qui propose l'utilisation de :
  - Plusieurs algorithmes de recherche : B&B, D&C, ...
  - Plusieurs environnent de programmation parallèle : Pthreads, MPI, ...
- L'ordonnancement parallèle
  - Insérer des nœuds dans une File de Priority Globale (FPG)
  - Chaque thread prend un nœud de la FPG pour effectuer une exploration locale et générer des nouveaux nœuds
  - Un nœud $\Leftrightarrow$ Un sous-arbre

Parallélisation non déterministe

Algorithme

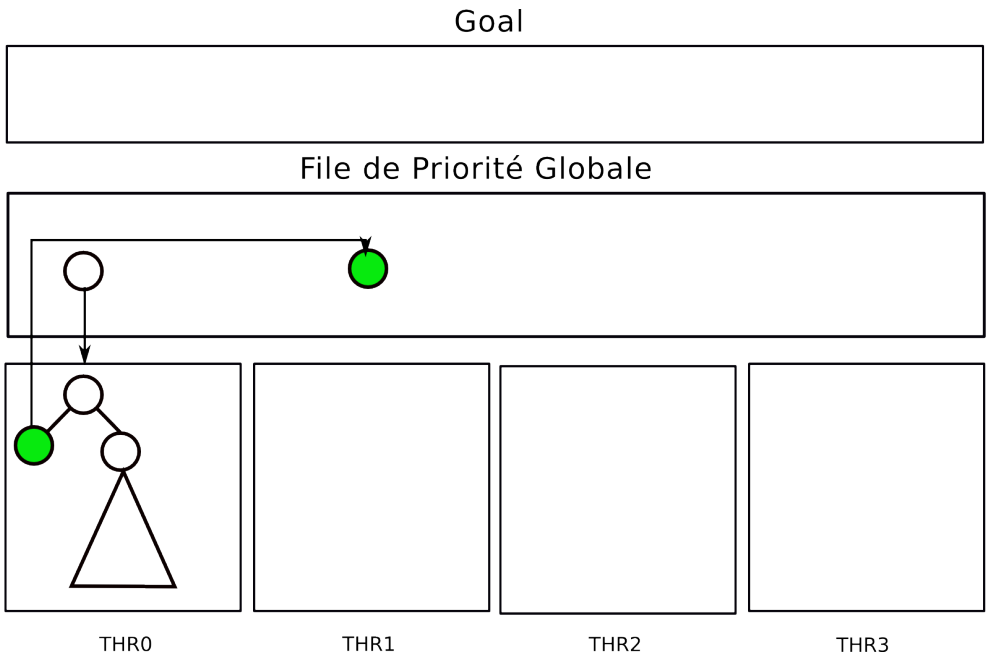
- Thread qui travaille (parcours un sous-arbre)
  - Pour chaque itération, teste s'il $\exists$ au mois un thread en attente :
    - Couper son sous-arbre en deux sous-arbres
    - Garder un sous-arbre et donner le deuxième au thread qui est en attente
  - Sinon, effectuer une exploration séquentielle (locale) de son sous-arbre de recherche

- Thread en attente (fini le parcours de son sous-arbre)
  - Si un nouveau nœud est inséré dans la File de Priorité Globale, il le récupère et commence la recherche



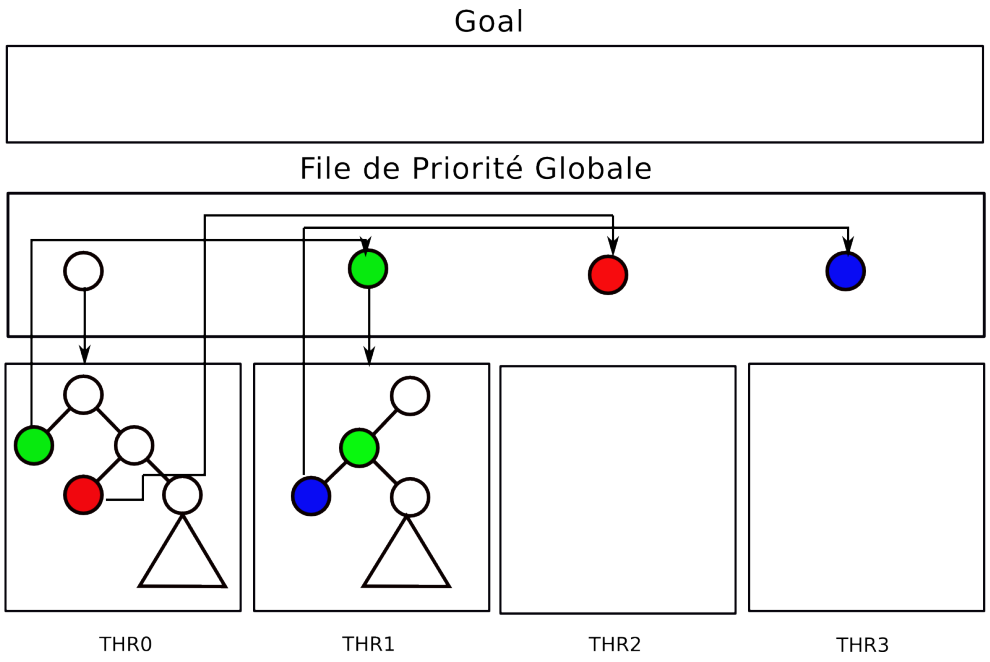
Étapes

- Nœud de départ



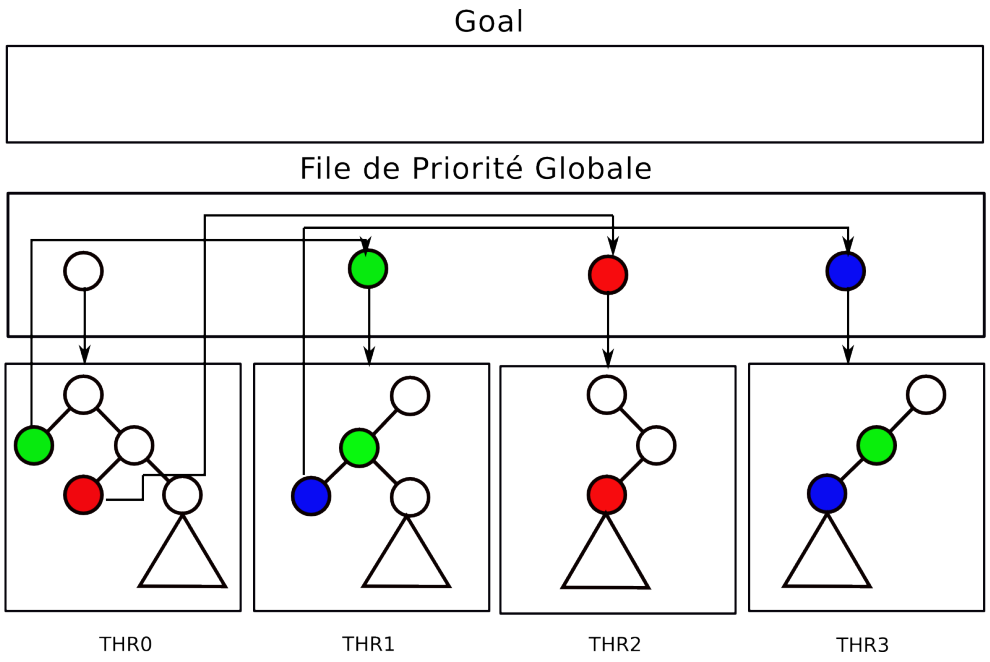
Étapes

- Nœud de départ
- Thread 0 commence



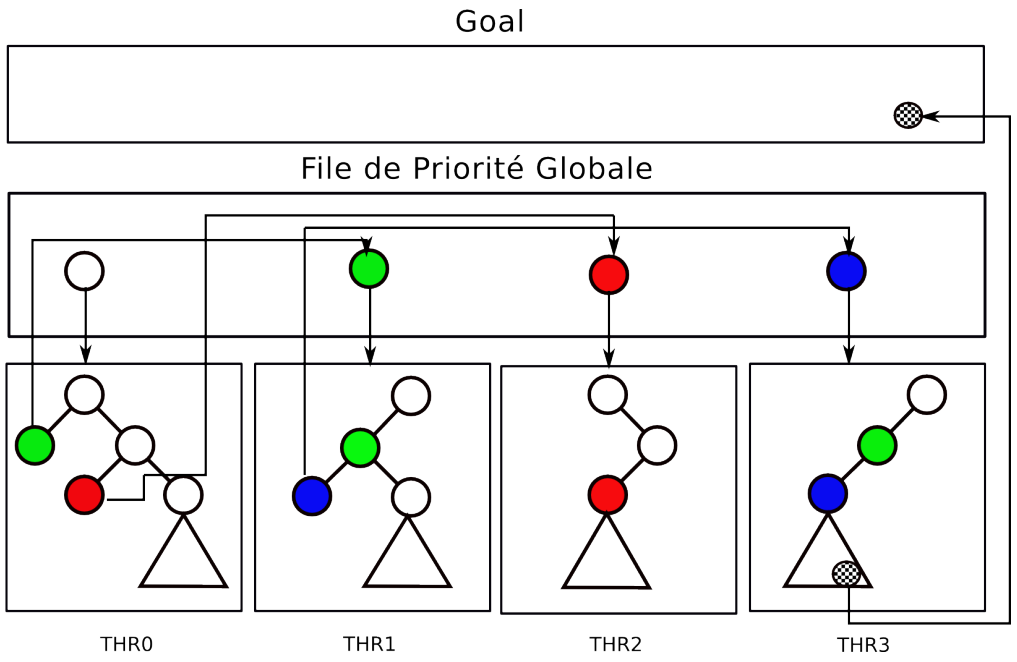
Étapes

- Nœud de départ
- Thread 0 commence
- Threads 0 et 1 travaillent



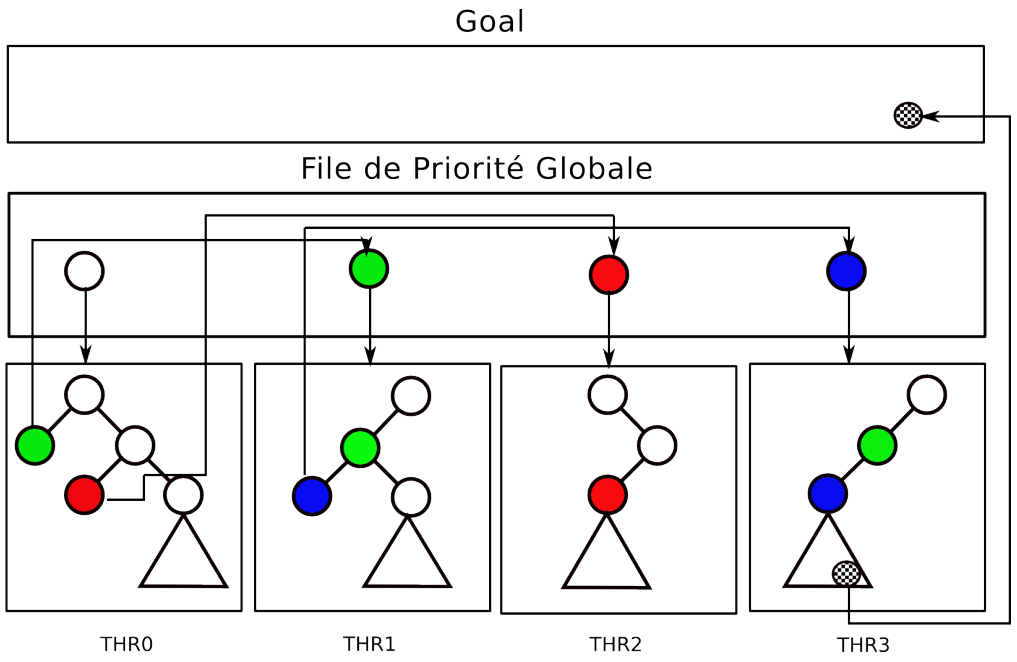
Étapes

- Nœud de départ
- Thread 0 commence
- Threads 0 et 1 travaillent
- Tous les threads travaillent



Étapes

- Nœud de départ
- Thread 0 commence
- Threads 0 et 1 travaillent
- Tous les threads travaillent
- Si solution, insérer dans le Goal



Étapes

- Nœud de départ
- Thread 0 commence
- Threads 0 et 1 travaillent
- Tous les threads travaillent
- Si solution, insérer dans le Goal

Avantages

✓ Partitionnement dynamique

✓ Bon équilibrage de charge

Inconvénient

✗ Solution retournée change suivant le mode d'exécution

Outline

1. Introduction
2. Parallélisation Non Déterministe
3. Parallélisation Déterministe
4. Expérimentation
5. Conclusion & Perspectives

Parallélisation Déterministe

Propriétés

- Le branchement des nœuds est déterministe,
- Donc l'arbre est déterministe
  - Identifiant affecté à chaque nœud y compris les solutions,
  - Ordre total sur les identifiants
- L'algorithme séquentiel retourne la solution avec l'identifiant le plus petit,

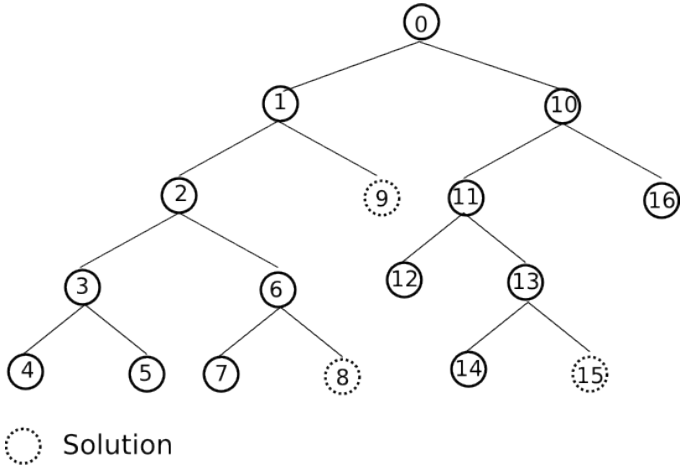
Principes

- Utilisation des identifiants pour rendre l'algorithme parallèle déterministe.
- Algorithme de partitionnement parallèle précédent peut être réutilisé
- Indépendant de l'environnement d'exécution : machine, bibliothèque, ordo des threads.

Parallélisation Déterministe

Pour résoudre les problèmes de satisfaction de contraintes

- S = Première solution trouvée
- Stopper tous les threads qui travaillent à droite de la solution S
- A chaque fois qu'on trouve une nouvelle solution N
  - Remplacer S par N
  - Stopper tous les threads qui travaillent à droite de la solution S
- Retourner S

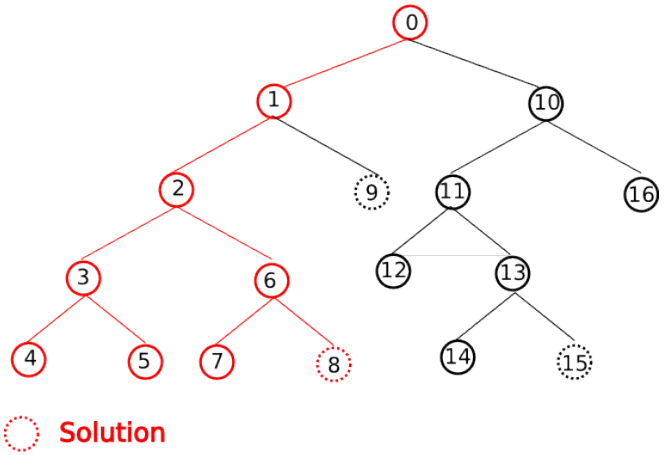


Parallélisation Déterministe

Pour résoudre les problèmes de satisfaction de contraintes

- S = Première solution trouvée
- Stopper tous les threads qui travaillent à droite de la solution S
- A chaque fois qu'on trouve une nouvelle solution N
  - Remplacer S par N
  - Stopper tous les threads qui travaillent à droite de la solution S
- Retourner S

Parcours Séquentiel

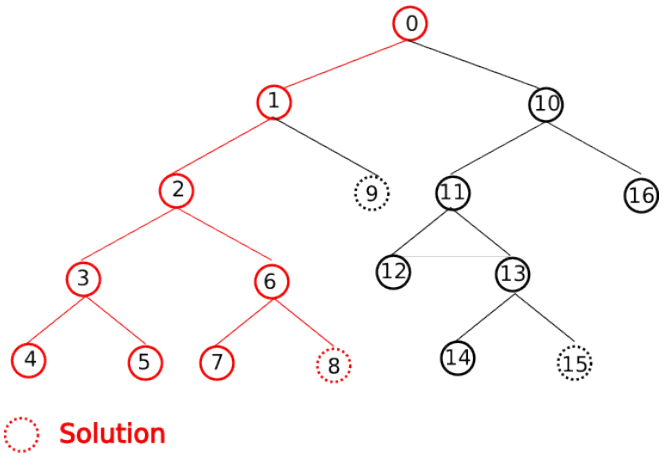


Parallélisation Déterministe

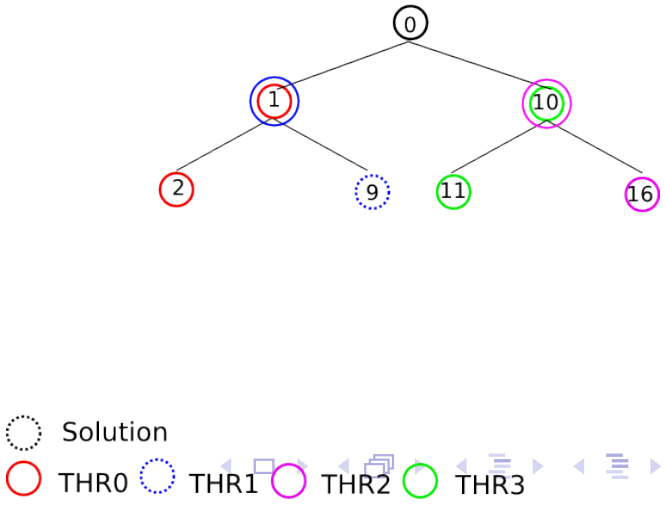
Pour résoudre les problèmes de satisfaction de contraintes

- S = Première solution trouvée
- Stopper tous les threads qui travaillent à droite de la solution S
- A chaque fois qu'on trouve une nouvelle solution N
  - Remplacer S par N
  - Stopper tous les threads qui travaillent à droite de la solution S
- Retourner S

Parcours Séquentiel



Parcours Parallèle

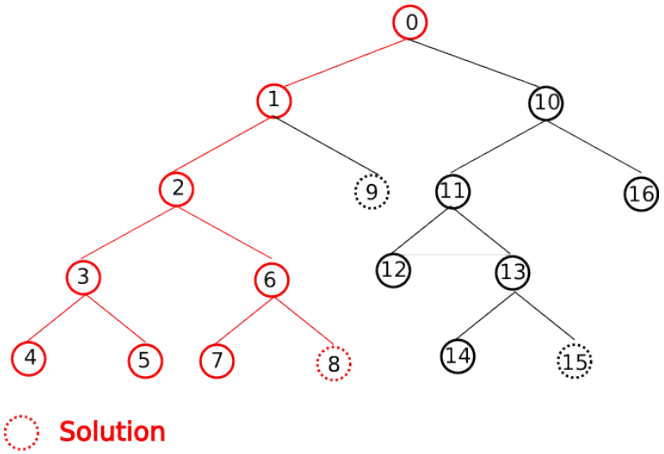


Parallélisation Déterministe

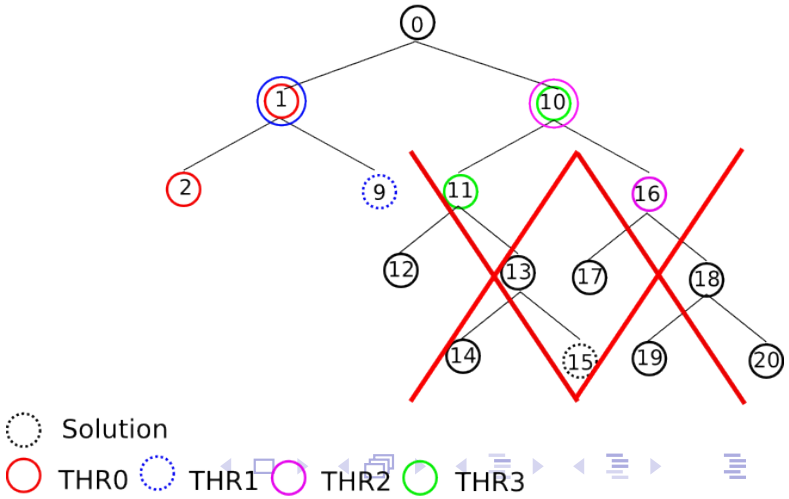
Pour résoudre les problèmes de satisfaction de contraintes

- S = Première solution trouvée
- Stopper tous les threads qui travaillent à droite de la solution S
- A chaque fois qu'on trouve une nouvelle solution N
  - Remplacer S par N
  - Stopper tous les threads qui travaillent à droite de la solution S
- Retourner S

Parcours Séquentiel



Parcours Parallèle

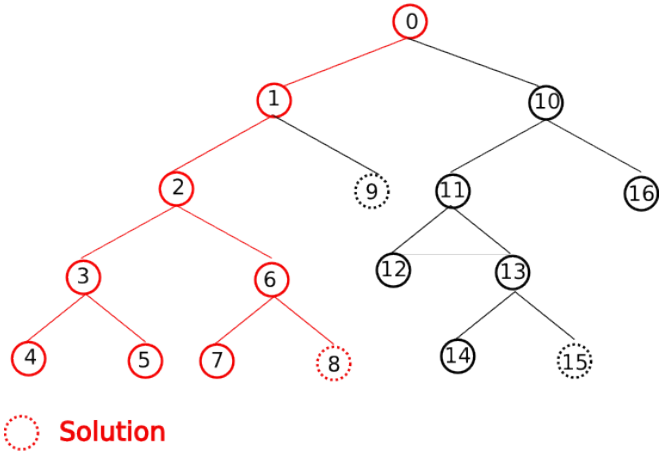


Parallélisation Déterministe

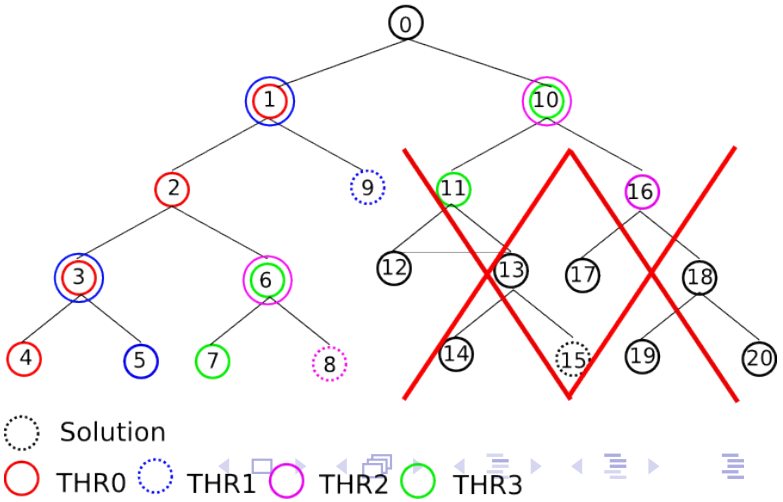
Pour résoudre les problèmes de satisfaction de contraintes

- S = Première solution trouvée
- Stopper tous les threads qui travaillent à droite de la solution S
- A chaque fois qu'on trouve une nouvelle solution N
  - Remplacer S par N
  - Stopper tous les threads qui travaillent à droite de la solution S
- Retourner S

Parcours Séquentiel



Parcours Parallèle



Parallélisation Déterministe

Pour résoudre les problèmes d'optimisation de contraintes

- S = Première solution trouvée
- Si une nouvelle solution N est trouvée, on remplace S par N si :
  - N est meilleur que S
  - N et S ont la même évaluation, mais l'identifiant de N est plus petit que l'identifiant de S
- Retourner S

Outline

1. Introduction
2. Parallélisation Non Déterministe
3. Parallélisation Déterministe
4. Expérimentation
5. Conclusion & Perspectives

Protocole

Machine

- Intel Xeon X5650 (2.67 GHz), 12 cœurs physiques et 48 Go de RAM

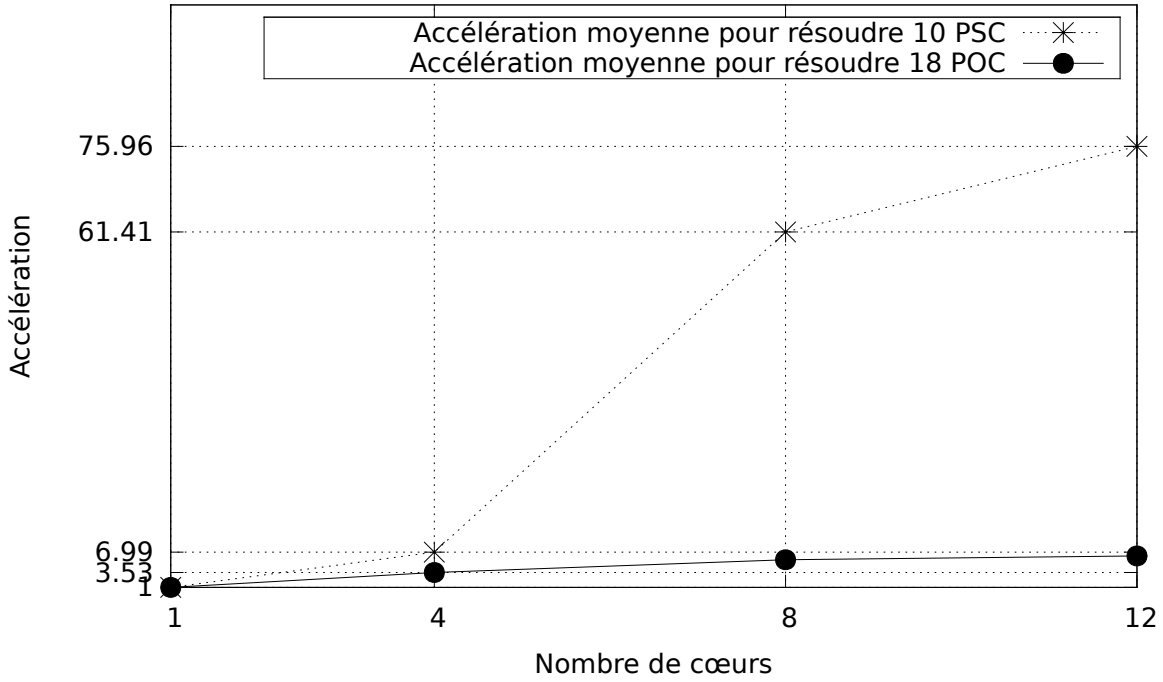
Logiciels

- Solveur OR-Tools (Version 2727)
- Bobpp Framework

Problèmes

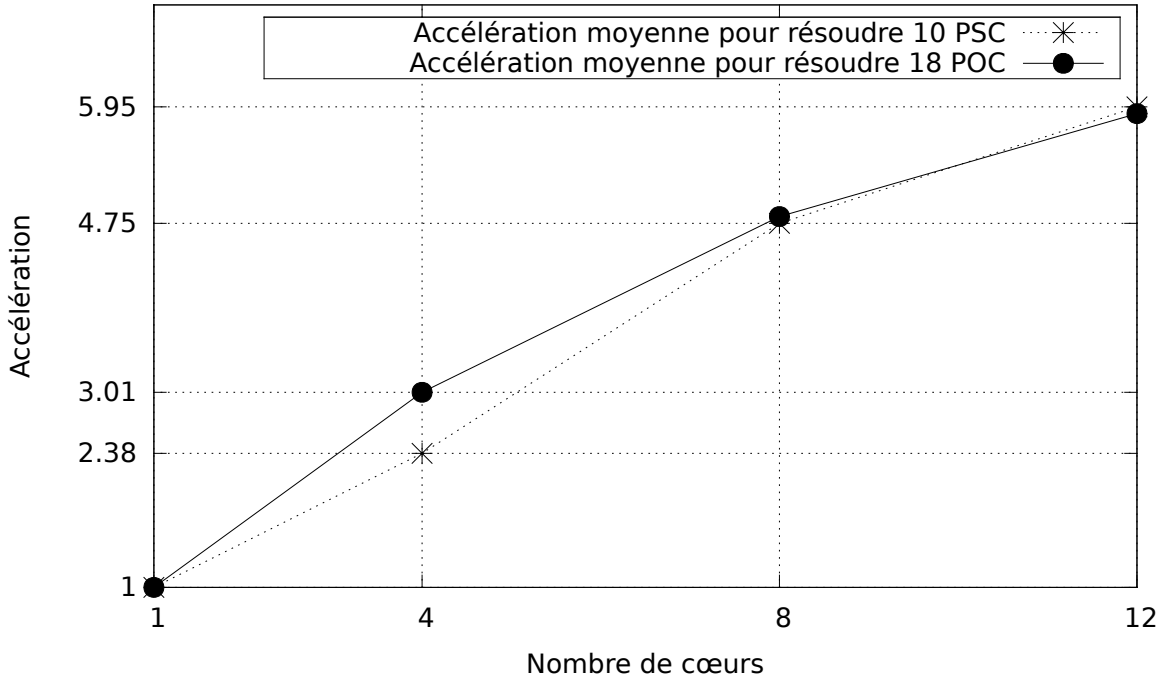
- MiniZinc Challenge 2012 (Format FlatZinc)
  - Problèmes de Satisfaction de Contraintes
  - Problèmes d'Optimisation de Contraintes

Accélération moyenne pour résoudre les problèmes de Programmation Par Contraintes en utilisant un algorithme non déterministe



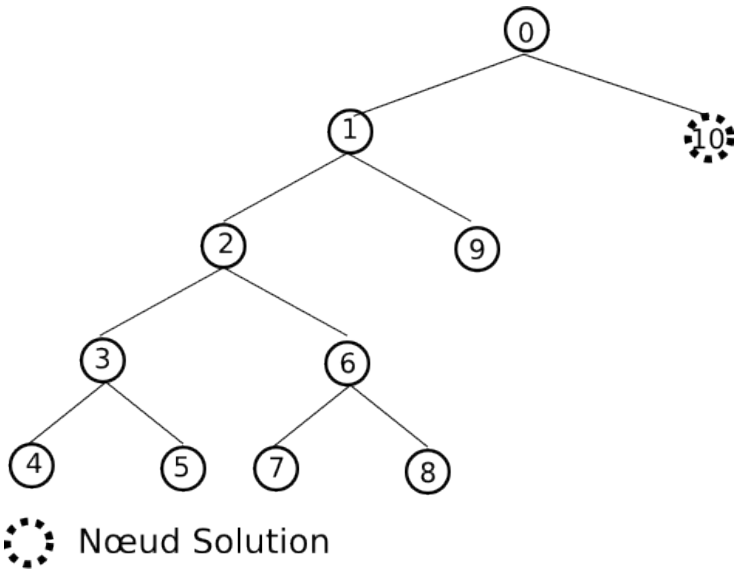
- Accélération moyenne pour résoudre 10 Problème de Satisfaction de Contraintes (PSC) et 18 Problèmes d'Optimisation de Contraintes (POC) avec un **algorithme non déterministe**

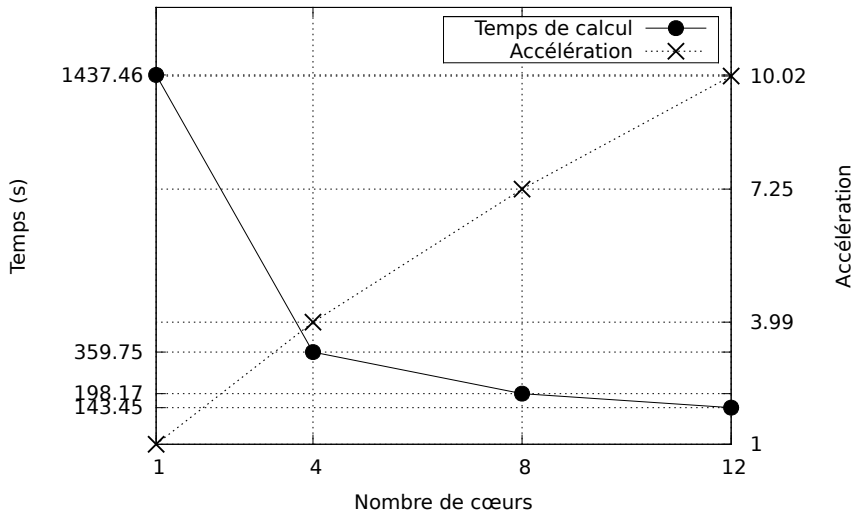
Accélération moyenne pour résoudre les problèmes de Programmation Par Contraintes en utilisant un algorithme déterministe



- Accélération moyenne pour résoudre 10 Problème de Satisfaction de Contraintes (PSC) et 18 Problèmes d'Optimisation de Contraintes (POC) avec un **algorithme déterministe**

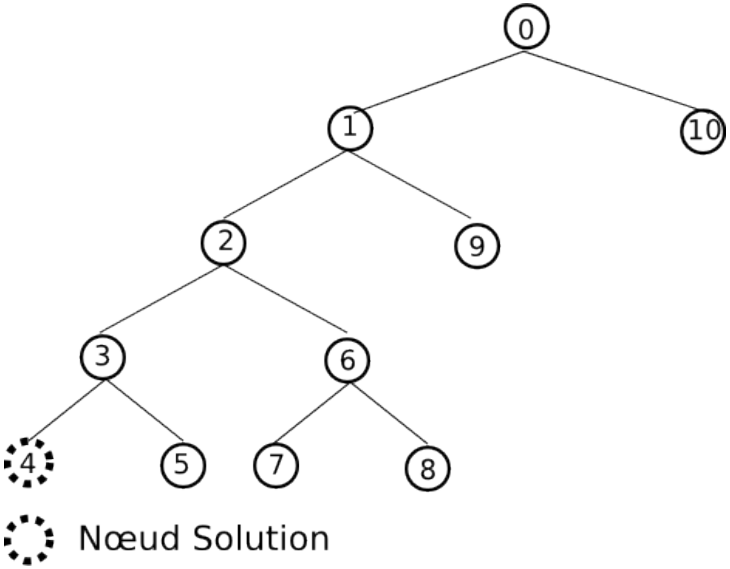
Problème de Satisfaction de Contraintes : Best Case

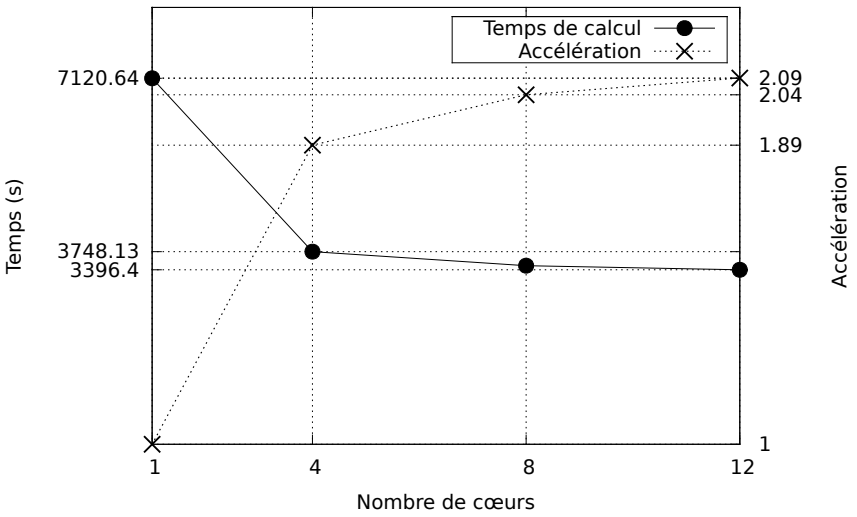




Best Case pour résoudre le problème de Quasigroup7_10.fzn

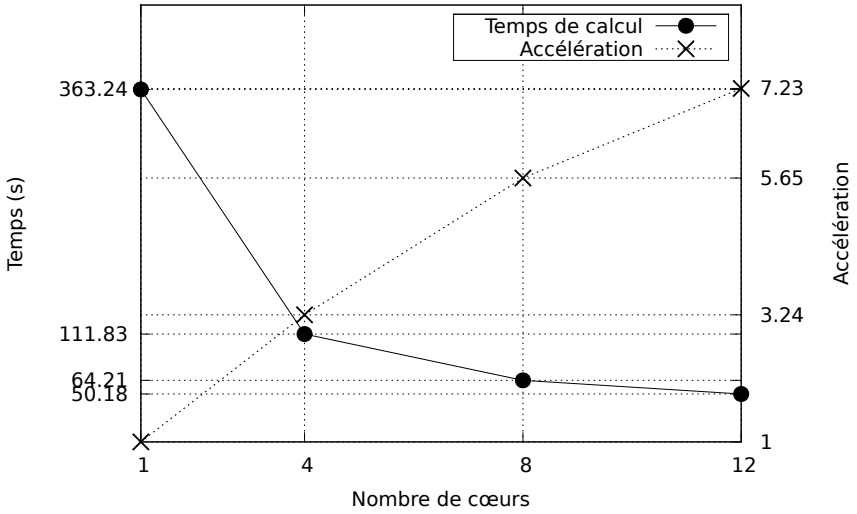
Problème de Satisfaction de Contraintes : Worst Case



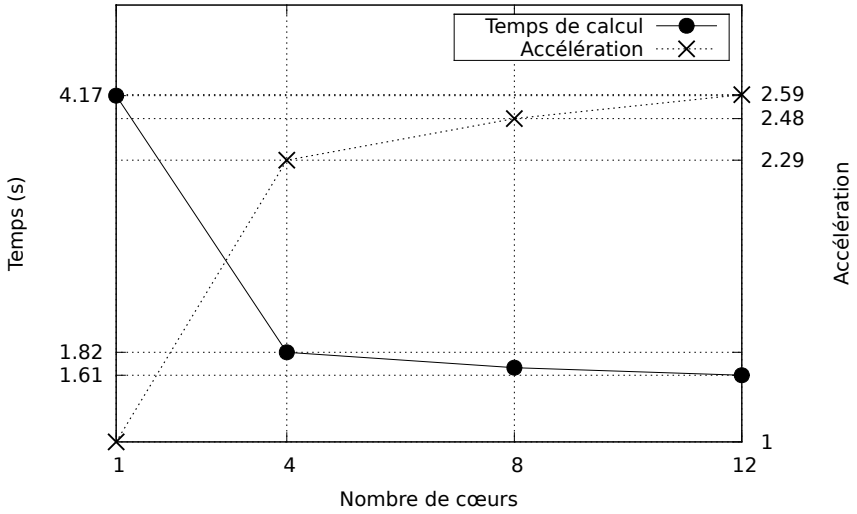


Worst Case pour résoudre le problème de market_split_s5-03.fzn

Problème d'Optimisation de Contraintes : Best et Worst Case

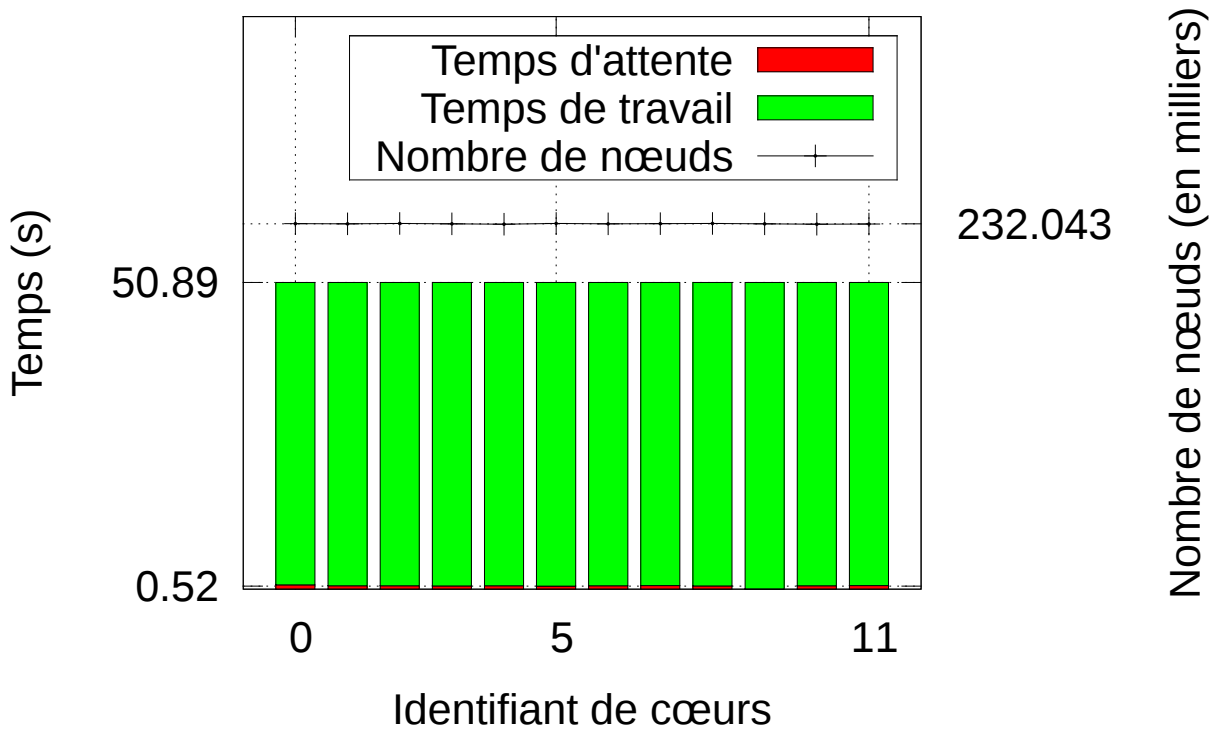


Best Case pour résoudre le problème de fastfood_ff63.fzn



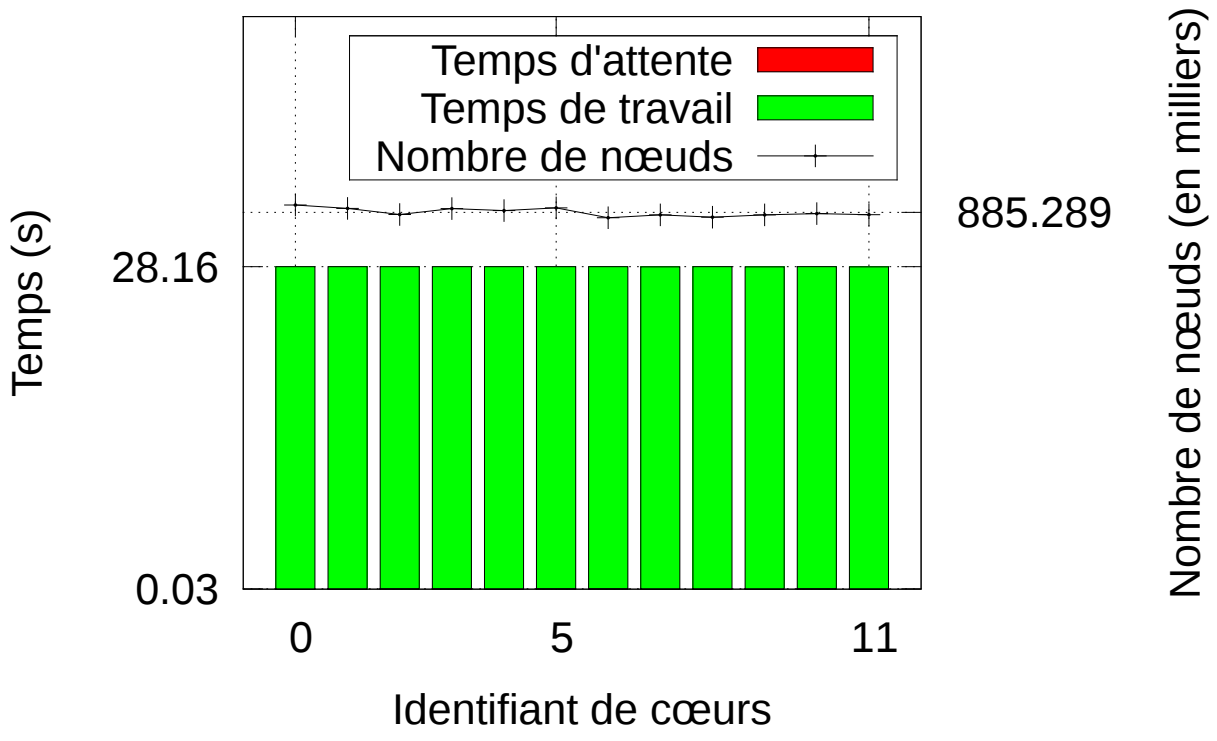
Worst Case pour résoudre le problème de fastfood_ff71.fzn

Équilibrage de charge pour résoudre un Problème de Satisfaction de Contraintes



- Équilibrage de charge pour résoudre un Problème de Satisfaction de Contraintes (Sb sb 13 13 5 1.fzn) en utilisant 12 threads

Équilibrage de charge pour résoudre un Problème d'Optimisation de Contraintes



- Équilibrage de charge pour résoudre un Problème d'Optimisation de Contraintes (fastfood_ff61.fzn) en utilisant 12 threads

Outline

1. Introduction
2. Parallélisation Non Déterministe
3. Parallélisation Déterministe
4. Expérimentation
5. Conclusion & Perspectives

Conclusions & Perspectives

Conclusions

- Algorithme parallèle déterministe pour parcourir les espaces de recherche
- Performance honorable avec la parallélisation déterministe
- Interfaçage Bobpp/OR-Tools sans changer le code source du solveur OR-Tools

Perspectives

- Tester la parallélisation Hybride (MPI + Pthreads) du framework Bobpp
- Proposer une parallélisation déterministe et non déterministe pour résoudre les problèmes de SATisfiability boolean (SAT)

Merci Pour Votre Attention

Les source de Bobpp

`http://forge.prism.uvsq.fr/projects/bobpp`

Travail réalisé dans le cadre du

projet "*PAJERO*" financé par *bipfrance*

Laboratoire PRISM