# Vers un partitionnement automatique d'applications en codelets spéculatifs pour les systèmes hétérogènes à mémoires distribuées

Eric Petit

Travail effectué à l'IRISA
sous la direction de François Bodin

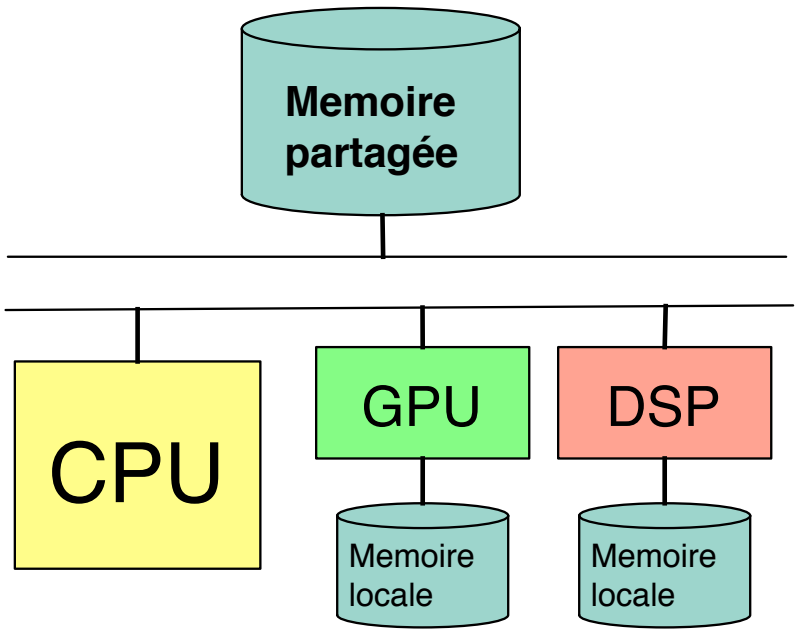
3 décembre 2009

# Architectures nouvelles - problèmes nouveaux

## Contexte de l'étude

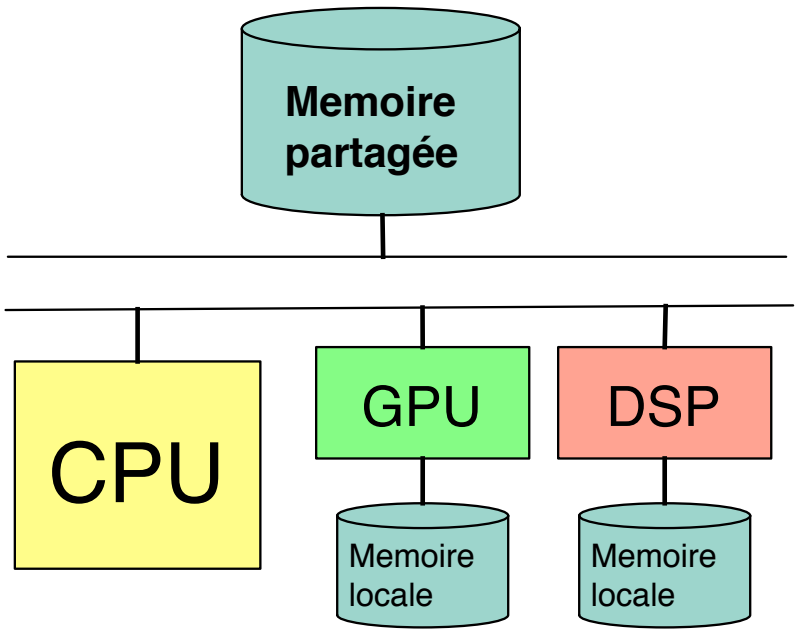
- Une nouvelle tendance forte en architecture :
  - multiplication des unités immédiatement accessibles
  - introduction de l'hétérogénéité
  - utilisation de co-processeurs spécialisés
  - mémoires et médias de communication complexes
  - différents niveaux de parallélisme

# Architectures nouvelles - problèmes nouveaux




- Toutes les combinaisons sont possibles
  $\Rightarrow$ grand nombre d'architectures

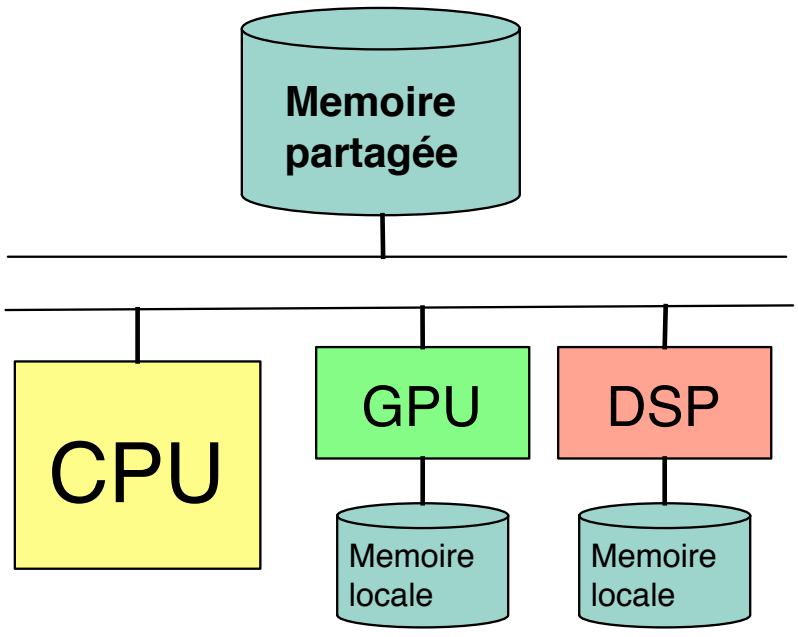
# Architectures nouvelles - problèmes nouveaux




- Toutes les combinaisons sont possibles
  $\Rightarrow$ grand nombre d'architectures

- Ce modèle tend à se généraliser du HPC aux systèmes embarqués
  $\Rightarrow$ grand nombre d'applications **variées**

# Architectures nouvelles - problèmes nouveaux

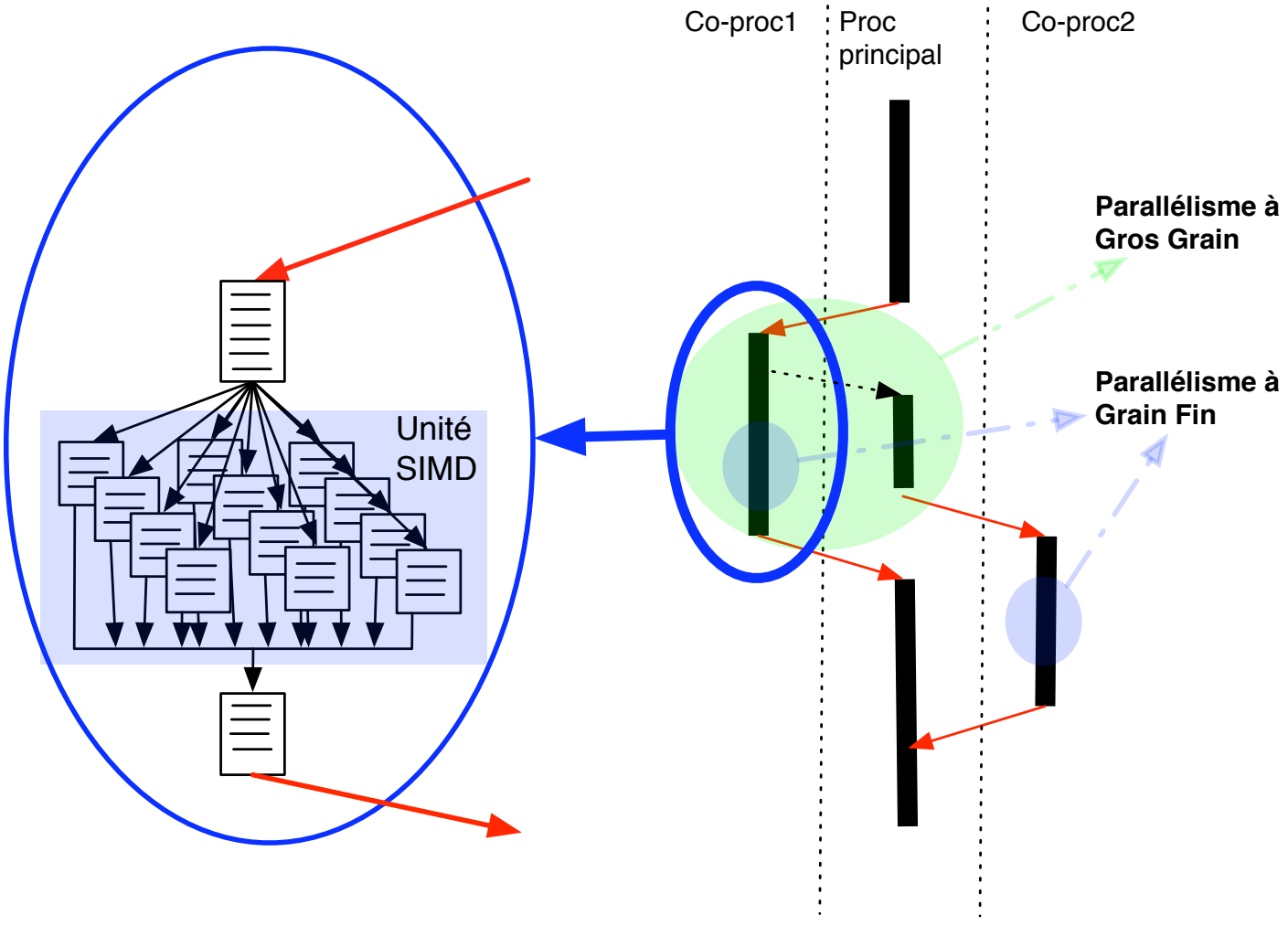



- Toutes les combinaisons sont possibles
  $\Rightarrow$ grand nombre d'architectures

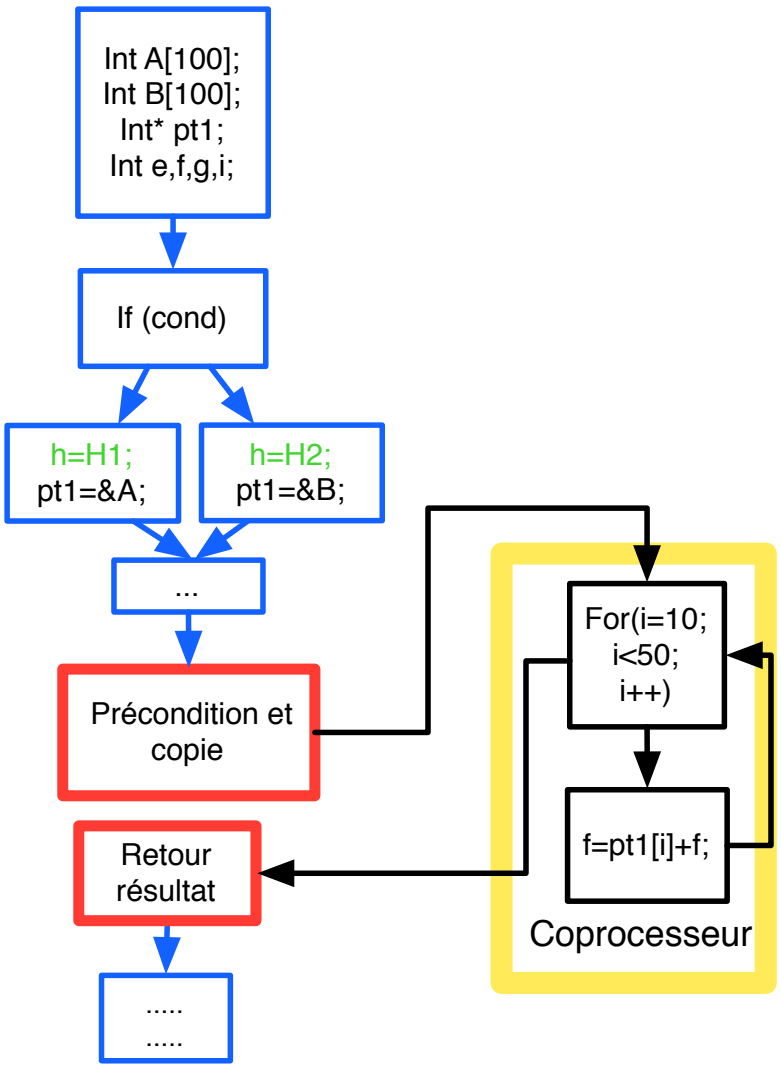
- Ce modèle tend à se généraliser du HPC aux systèmes embarqués
  $\Rightarrow$ grand nombre d'applications **variées**

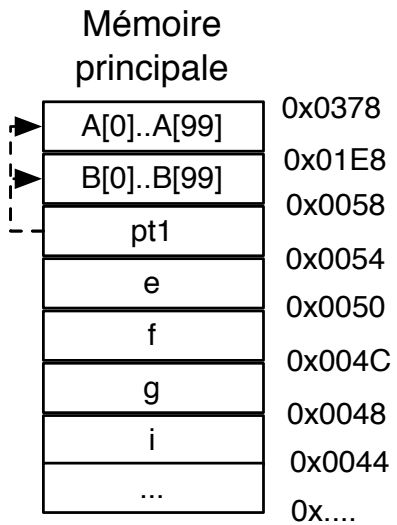
- **Quelle est l'influence de ces changements sur les applications ?**

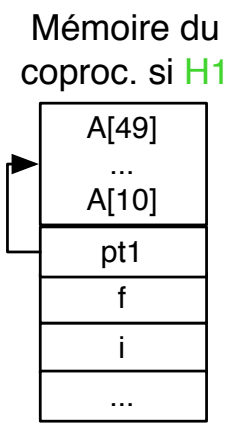
# L'exécution multitâche et parallèle

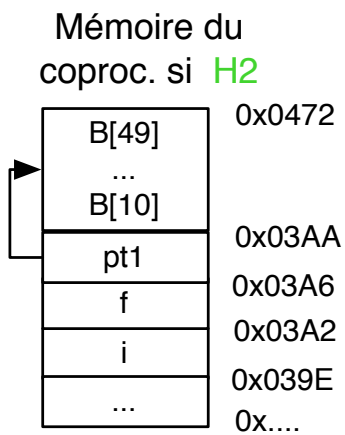


# Translation de la mémoire



# Architectures nouvelles - problèmes nouveaux

- Exploiter les co-processeurs
  - identifier des tâches
  - vérifier leur compatibilité
  - optimiser leur exécution

- Gérer les mémoires distribuées
  - communications
  - alias et dépendances
  - arithmétiques de pointeur

# Architectures nouvelles - problèmes nouveaux

- Exploiter les co-processeurs
  - identifier des tâches
  - vérifier leur compatibilité
  - optimiser leur exécution

- Gérer les mémoires distribuées
  - communications
  - alias et dépendances
  - arithmétiques de pointeur

- **Les modèles actuels de programmation ne suffisent plus**

# Architectures nouvelles - problèmes nouveaux

## Problèmatique

- Il faut pour ces architectures des applications adaptées
  - complexes à programmer
  - très grand nombre d'applications et bibliothèques standards
  - portabilité des codes produits

# Architectures nouvelles - problèmes nouveaux

## Problèmatique

- Il faut pour ces architectures des applications adaptées
  - complexes à programmer
  - très grand nombre d'applications et bibliothèques standards
  - portabilité des codes produits

$\Rightarrow$ Peut-on exploiter automatiquement ces nouvelles architectures ?

# Architectures nouvelles - problèmes nouveaux

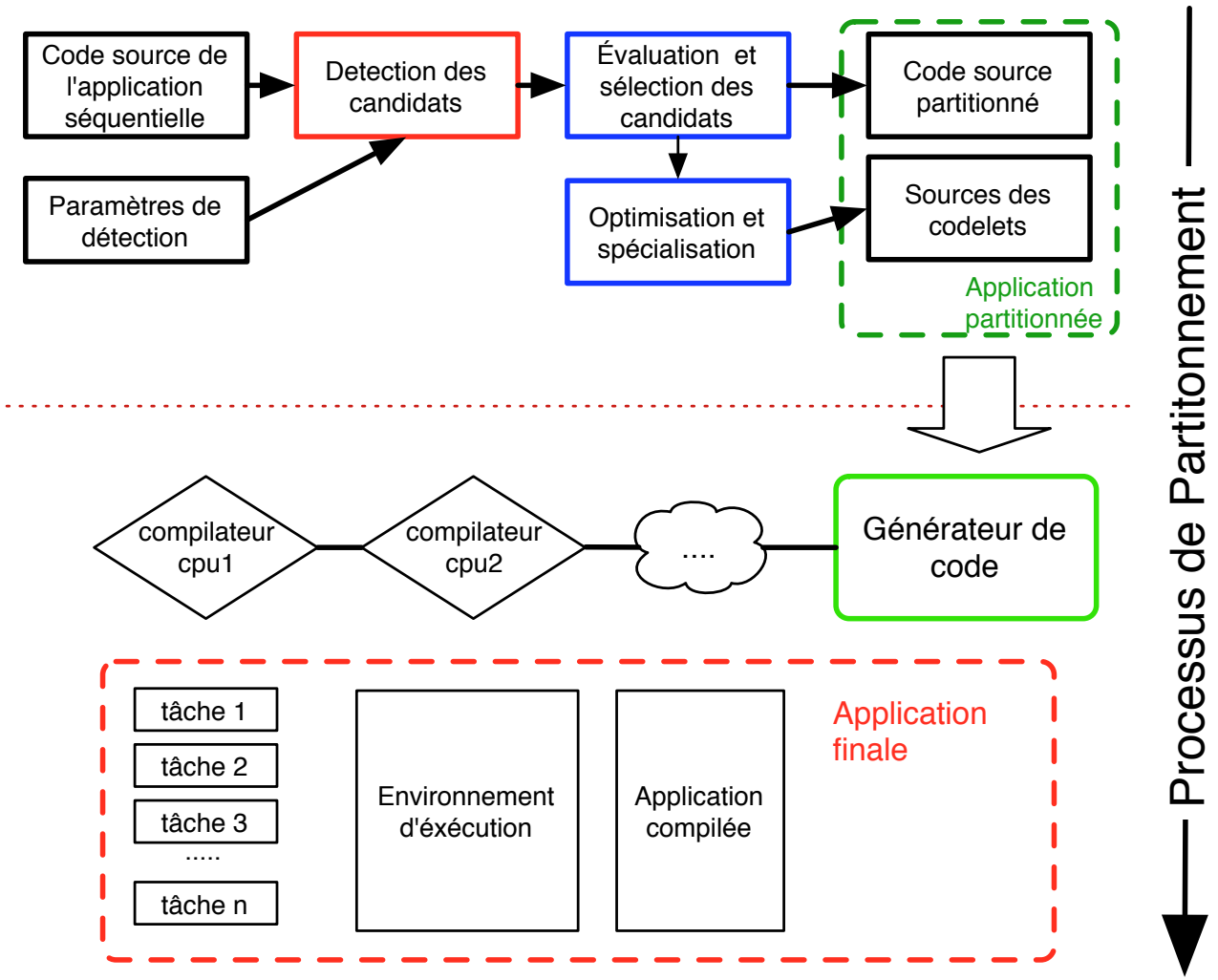
## Problèmatique

- Il faut pour ces architectures des applications adaptées
  - complexes à programmer
  - très grand nombre d'applications et bibliothèques standards
  - portabilité des codes produits

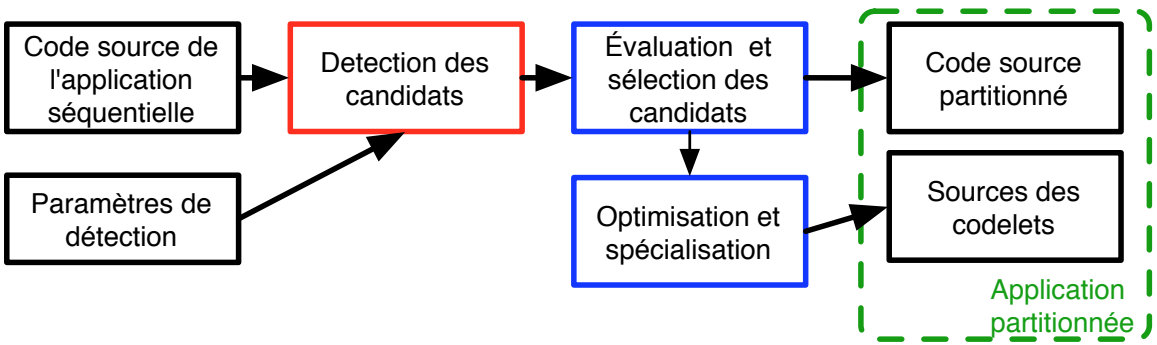
⇒ Peut-on exploiter automatiquement ces nouvelles architectures ?

⇒ Existe-t-il un modèle unifié des programmes qui puisse permettre l'utilisation de toutes ces architectures et leurs évolutions ?

# Le processus de partitionnement



# Le processus de partitionnement



# Première contribution

## Définir un modèle portable des partitions

- Définir l'unité du partitionnement : les codelets
- Gérer les mémoires distribuées non cohérentes
- Adresser les différentes unités d'un système hétérogène

## Définir et implanter un processus automatique de calcul de partition

- Extraire d'une application les portions de code "intéressantes"
- Récolter toutes les informations nécessaires au modèle
  $\Rightarrow$ utilisation de données spéculatives

# Première contribution

## Objectifs principaux

- Utilisation d'accélérateurs matériels (e.g. GPU)
- Gestion de la mémoire distribuée
  - complexifie les communications
  - simplifie l'exécution spéculative

# Seconde contribution

## L'optimisation des communications

- Montrer l'influence critique des communications, en particulier dans le cadre du GPGPU
- Construire la modélisation du problème d'optimisation des communications
  - modéliser le programme et les communications
  - définir la validité d'une solution
  - définir un modèle d'évaluation des solutions : fonction de coût
- Proposer des algorithmes solutions

# Les processus automatiques de partitionnement

## Les méthodes issues de l'analyse statique

- À la compilation
- Information : code de l'application

# Les processus automatiques de partitionnement

## Les méthodes issues de l'analyse statique

- À la compilation
- Information : code de l'application

- Vision globale de l'application
- Pas de surcoût de partitonnement à l'exécution
- Permet l'utilisation de méthodes statiques évoluées

# Les processus automatiques de partitionnement

## Les méthodes issues de l'analyse statique

- À la compilation
- Information : code de l'application

- Vision globale de l'application
- Pas de surcoût de partitonnement à l'exécution
- Permet l'utilisation de méthodes statiques évoluées

- Sur-évaluation nécessaire des contraintes
- La qualité de la solution dépend de celle du code en entrée
- La détection de candidats est figée sur des modèles prédéterminés

# Les processus automatiques de partitionnement

## Les méthodes dynamiques

- À l'exécution
- Information : trace des dernières instructions

# Les processus automatiques de partitionnement

## Les méthodes dynamiques

- À l'exécution
- Information : trace des dernières instructions

- Adaptabilité

# Les processus automatiques de partitionnement

## Les méthodes dynamiques

- À l'exécution
- Information : trace des dernières instructions

- Adaptabilité

- Visibilité réduite à la fenêtre d'observation
- Performances tributaires de mécanismes matériels dédiés
- Surcoût en silicium, performance, énergie

# Une nouvelle approche spéculative

## Hybridation info dynamique - analyse statique

- À la compilation, complète l'analyse statique
- Informations : code de l'application et données de profils d'exécution

# Une nouvelle approche spéculative

## Hybridation info dynamique - analyse statique

- À la compilation, complète l'analyse statique
- Informations : code de l'application et données de profils d'exécution

- Détection basée sur des données réelles
- Permet l'utilisation de méthodes statiques évoluées
- Adaptabilité (limitée), portabilité

# Une nouvelle approche spéculative

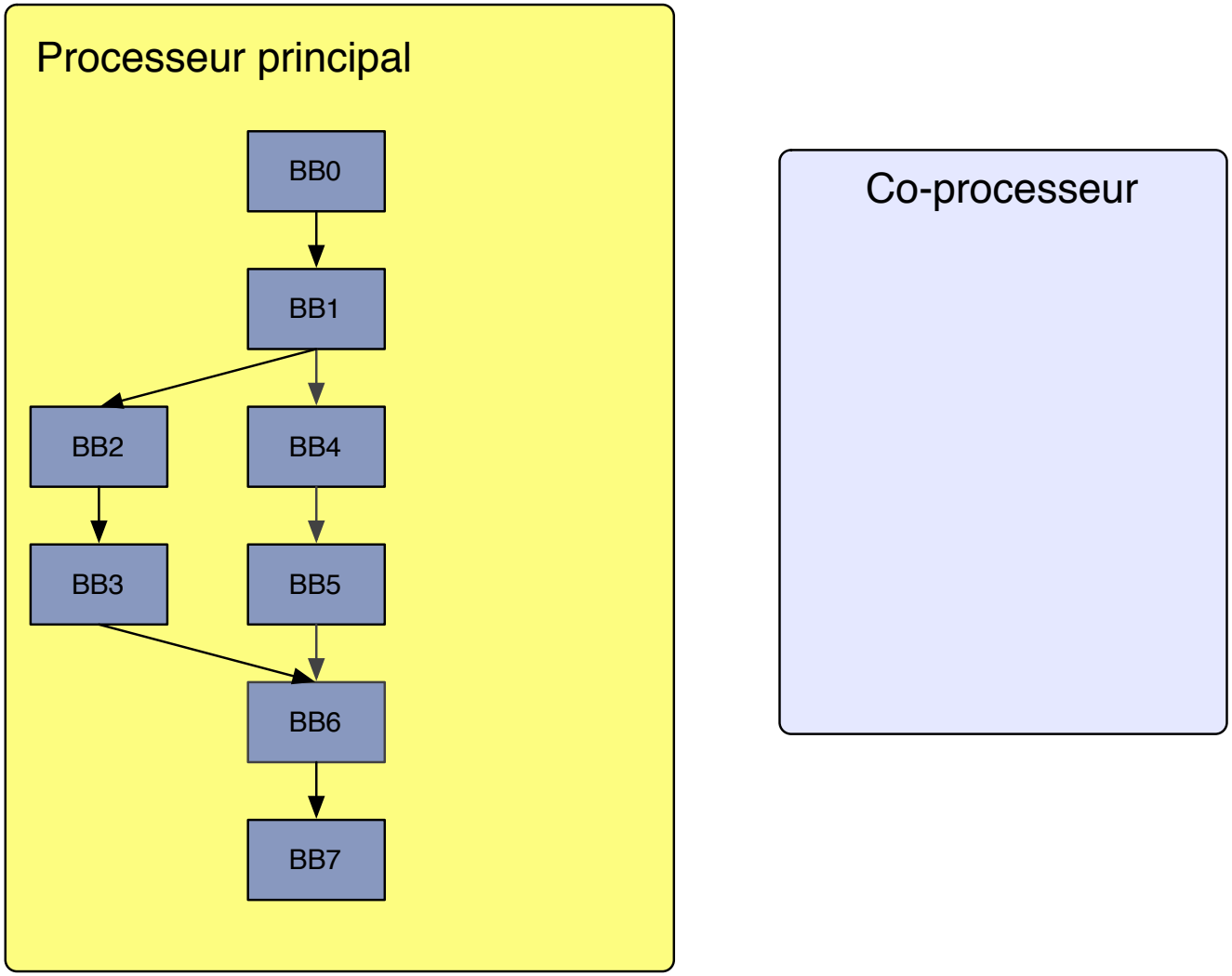
## Hybridation info dynamique - analyse statique

- À la compilation, complète l'analyse statique
- Informations : code de l'application et données de profils d'exécution

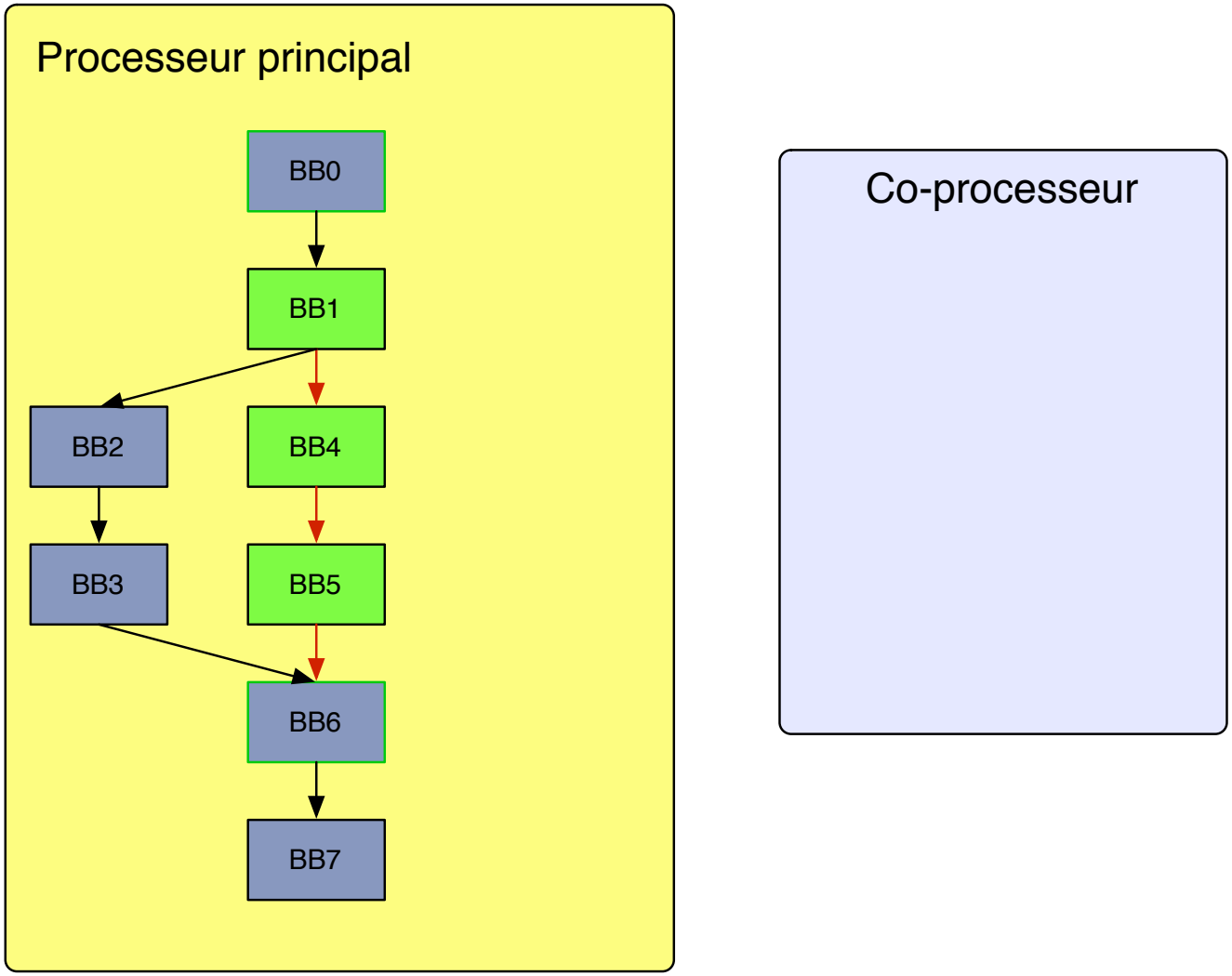
- Détection basée sur des données réelles
- Permet l'utilisation de méthodes statiques évoluées
- Adaptabilité (limitée), portabilité

- Données spéculatives $\Rightarrow$ exécution spéculative
- Surcoût des tests et des erreurs potentiels
- La qualité de la solution dépend de celle des profils d'exécution en entrée

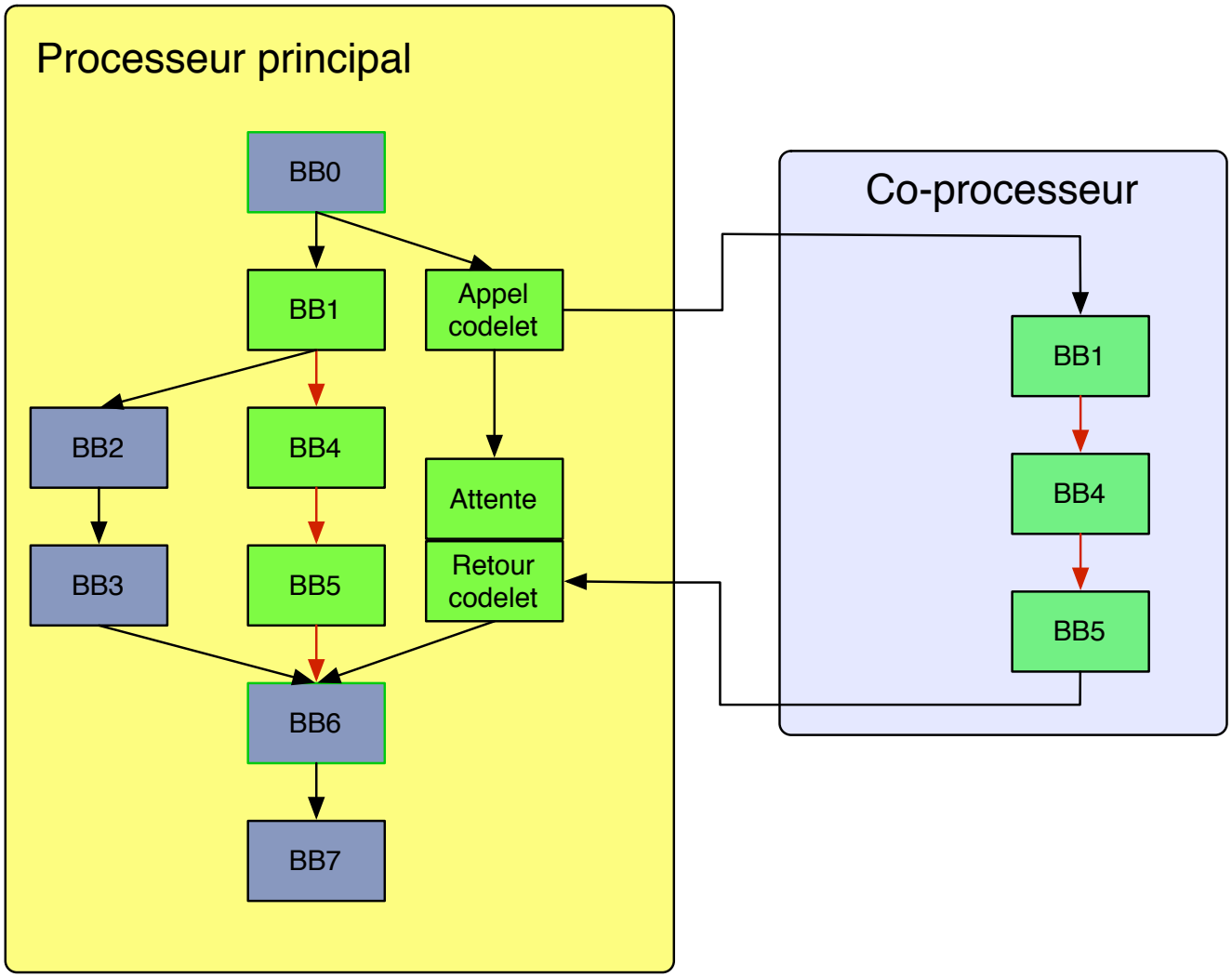
# Un modèle d'exécution spéculative pour ASTEX



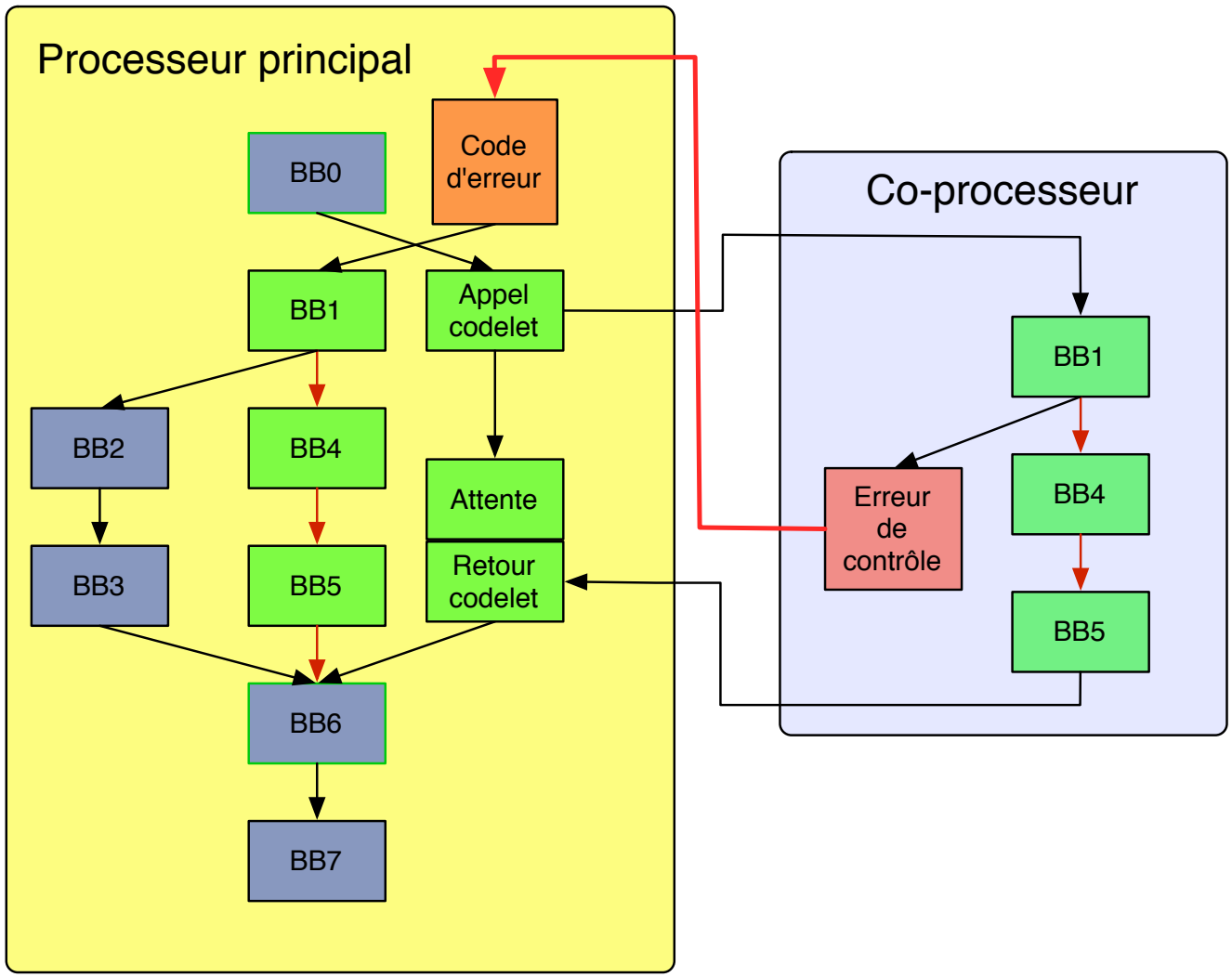
# Un modèle d'exécution spéculative pour ASTEX



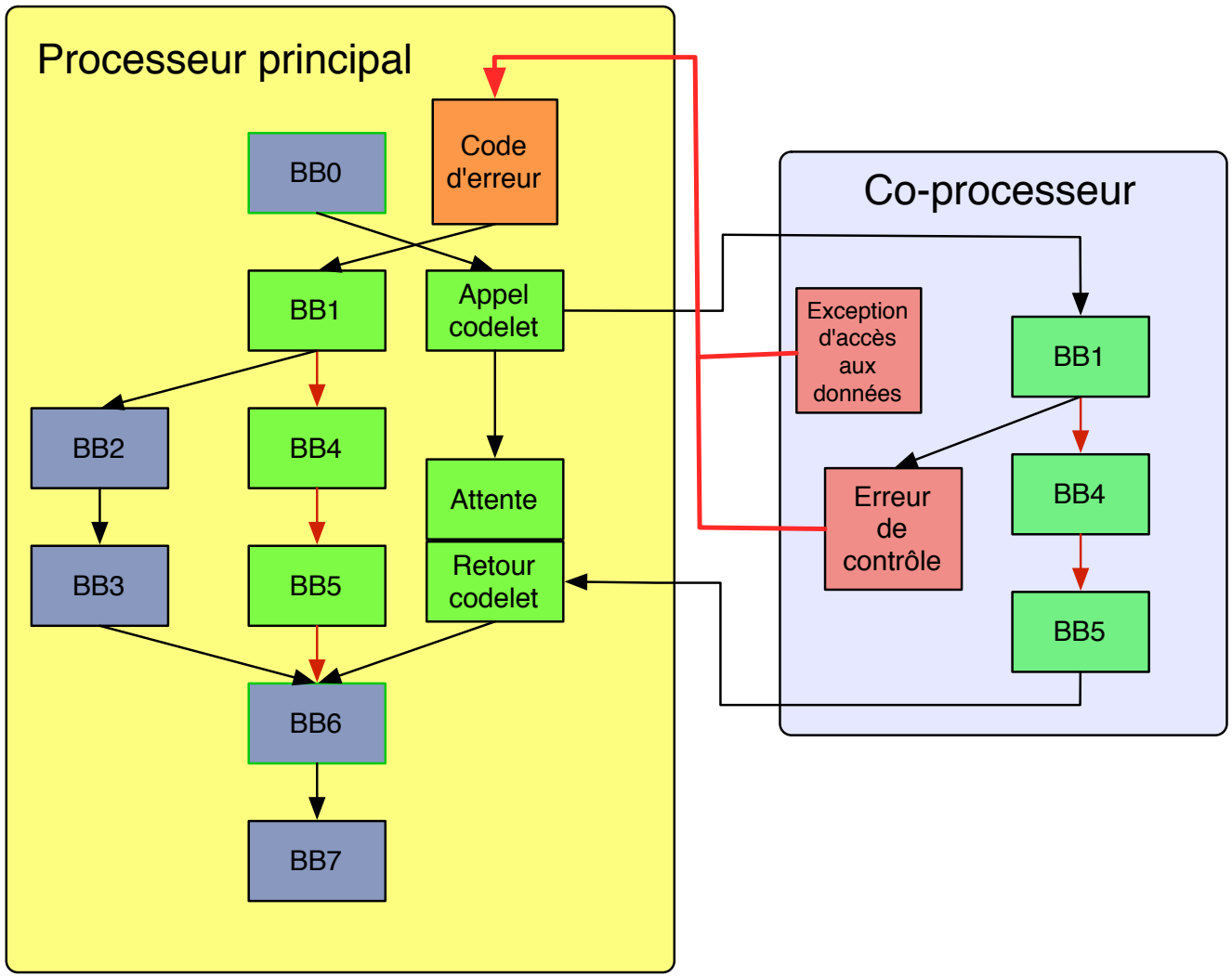
# Un modèle d'exécution spéculative pour ASTEX



# Un modèle d'exécution spéculative pour ASTEX



# Un modèle d'exécution spéculative pour ASTEX



# La spéculation dans Astex

- Sur les chemins dans le CFG et leur couverture en temps
  - Accélération potentielle significative : loi d'Amdhal
  - Recherche de "hotpaths" (Basée sur le travail de G. Pokam)
  - Pré-sélection : temps d'exécution, modèle, logiciel

# La spéculation dans Astex

- Sur les chemins dans le CFG et leur couverture en temps
  - Accélération potentielle significative : loi d'Amdhal
  - Recherche de "hotpaths" (Basée sur le travail de G. Pokam)
  - Pré-sélection : temps d'exécution, modèle, logiciel

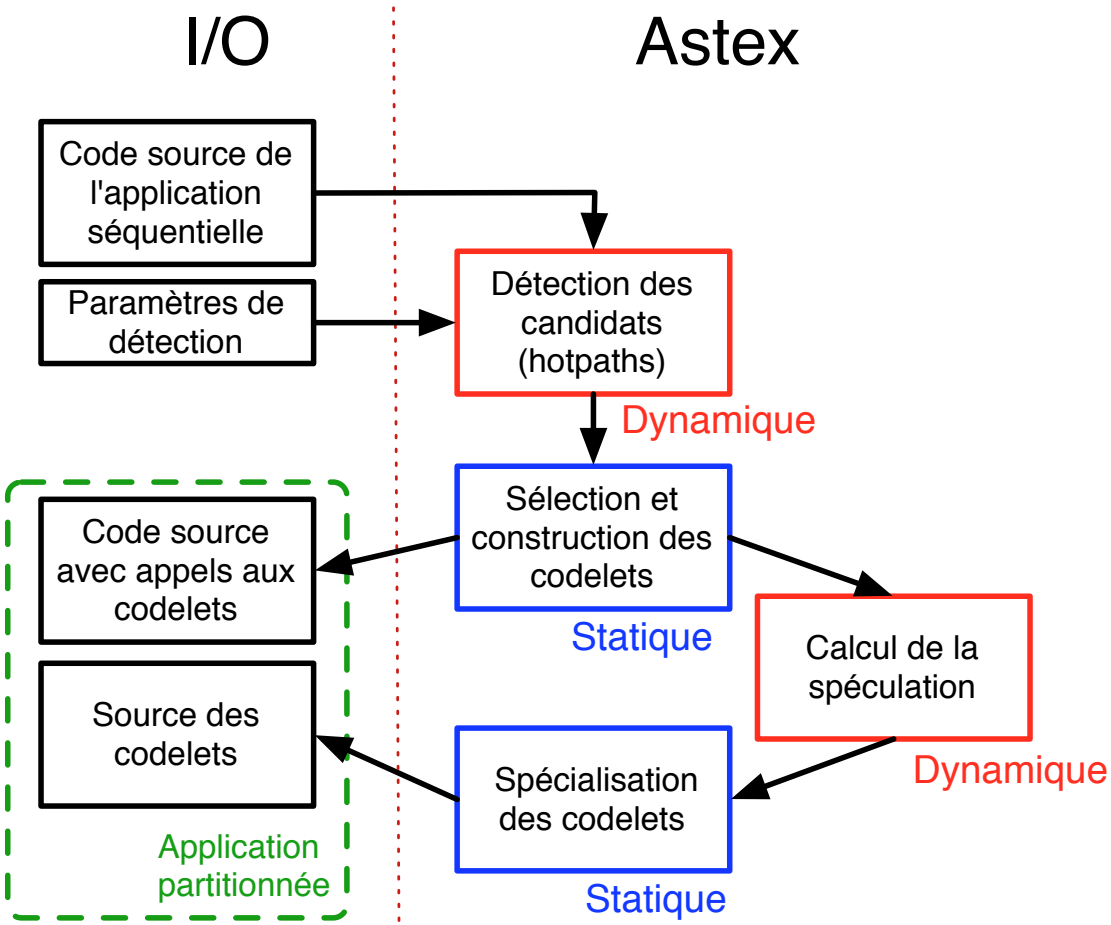
- Sur les données des codelets
  - Entrées sorties
  - Plages d'accès aux blocs mémoires copiés
  - Valeurs des variables scalaires
  - Alignements des pointeurs

# Modèle de description d'une partition

- Des éléments globaux :
  - Les zones mémoire abstraites
  - Les contextes

- Les codelets :
  - Un ensemble de blocs de base connexes du CFG
  - Pour chaque contexte :
    - Un résumé des accès aux zones pour chaque expression
    - Les valeurs observées des variables scalaires
    - Les alignements observés des pointeurs
  - Les différentes versions spécialisées et la garde associée

# Le processus d'extraction d'ASTEX



# Du hotpath au codelet spécialisé

## Formation du codelet

```
{
   i_sum += abs(pix1[x] - pix2[x]);
}
```

# Du hotpath au codelet spécialisé

## Formation du codelet

```
for (y = 0; y < 8; y++)
{
   for (x = 0; x < 8; x++)
   {
      i_sum += abs(pix1[x] - pix2[x]);
   }
pix1 += i_stride_pix1;
pix2 += i_stride_pix2;
}
```

# Du hotpath au codelet spécialisé

## Formation du codelet

```
void Astex_codelet__1(uint8_t *pix1, int i_stride_pix1,
uint8_t *pix2, int i_stride_pix2, int __Astex_addr__i_sum[1])
{
   int y;
   int x;
   int i_sum = __Astex_addr__i_sum[0];
   Astex_thread_begin : {
      for (y = 0; y < 8; y++)
      {
         for (x = 0; x < 8; x++)
         {
            i_sum += abs(pix1[x] - pix2[x]);
         }
      pix1 += i_stride_pix1;
      pix2 += i_stride_pix2;
      }
   }
   Astex_thread_end :;
   __Astex_addr__i_sum[0] = i_sum;
}
```

# Du hotpath au codelet spécialisé

## Formation du codelet

```
void Astex_codelet__1(uint8_t *pix1, int i_stride_pix1,
uint8_t *pix2, int i_stride_pix2, int __Astex_addr__i_sum[1])
{
   int y;
   int x;
   int i_sum = __Astex_addr__i_sum[0];
   Astex_thread_begin : {
      for (y = 0; y < 8; y++)
      {
         for (x = 0; x < 8; x++)
         {
            i_sum += abs(pix1[x] - pix2[x]);
         }
      pix1 += 16;
      pix2 += i_stride_pix2;
      }
   }
   Astex_thread_end :;
   __Astex_addr__i_sum[0] = i_sum;
}
```

# Du hotpath au codelet spécialisé

## Génération de la garde

```
inline int Astex_guard__1(uint8_t *pix1, int i_stride_pix1,
uint8_t *pix2, int i_stride_pix2, int __Astex_addr__i_sum[1])
{
   const void *__Astex_region_addr__pix1;
   unsigned long int __Astex_region_size__pix1;
   __Astex_get_region(
      &__Astex_region_addr__pix1,
      &__Astex_region_size__pix1, pix1);
   return i_stride_pix1 == 16 &
      (__Astex_region_addr__pix1 != (const void *) (0) &
      (pix2 < __Astex_region_addr__pix1 |
      pix2 >= __Astex_region_addr__pix1 + __Astex_region_size__pix1));
   }
```

# Du hotpath au codelet spécialisé

## Insertion du codelet dans le code original

```
if Astex_guard__1(pix1, i_stride_pix1, pix2,
i_stride_pix2, __Astex_addr__i_sum[1])
{
   #pragma hmpp astex_codelet__1 callsite
   void Astex_codelet__1(pix1, i_stride_pix1, pix2,
   i_stride_pix2, __Astex_addr__i_sum[1])
}
else
{
      for (y = 0; y < 8; y++)
      {
         for (x = 0; x < 8; x++)
         {
            i_sum += abs(pix1[x] - pix2[x]);
         }
         pix1 += i_stride_pix1;
         pix2 += i_stride_pix2;
      }
   }
}
```

# Du hotpath au codelet spécialisé

## Insertion du codelet dans le code original

```
if Astex_guard__1(pix1, i_stride_pix1, pix2,
i_stride_pix2, __Astex_addr__i_sum[1])
{
   #pragma hmpp astex_codelet__1 callsite
   void Astex_codelet__1(pix1, i_stride_pix1, pix2,
   i_stride_pix2, __Astex_addr__i_sum[1])
}
else
{
      for (y = 0; y < 8; y++)
      {
         for (x = 0; x < 8; x++)
         {
            i_sum += abs(pix1[x] - pix2[x]);
         }
         pix1 += i_stride_pix1;
         pix2 += i_stride_pix2;
      }
   }
}
```

# Expérimentations

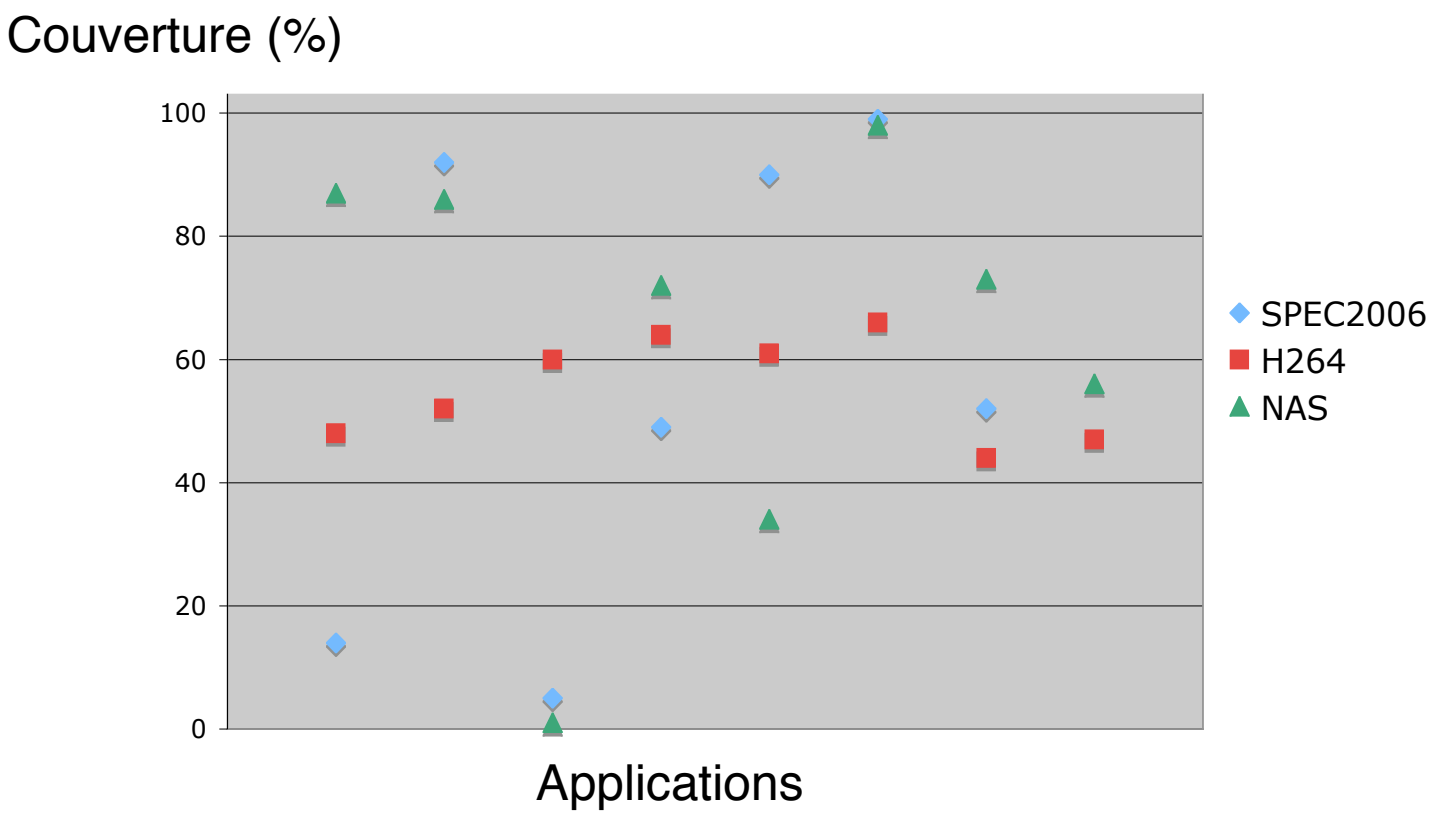
## Les familles d'applications de test selectionnées

- Les NAS parallel benchmarks : algorithmes parallèles en C et openMP
  $\Rightarrow$ vérifier la conformité des résultats produits
- H264 : code multimédia au contrôle complexe
  $\Rightarrow$ résultat sur un code réputé difficile
- Les SPEC2006 : complexes et variées
  $\Rightarrow$ test de la robustesse

# Expérimentations

- Couverture du temps d'exécution des codelets par application (%)

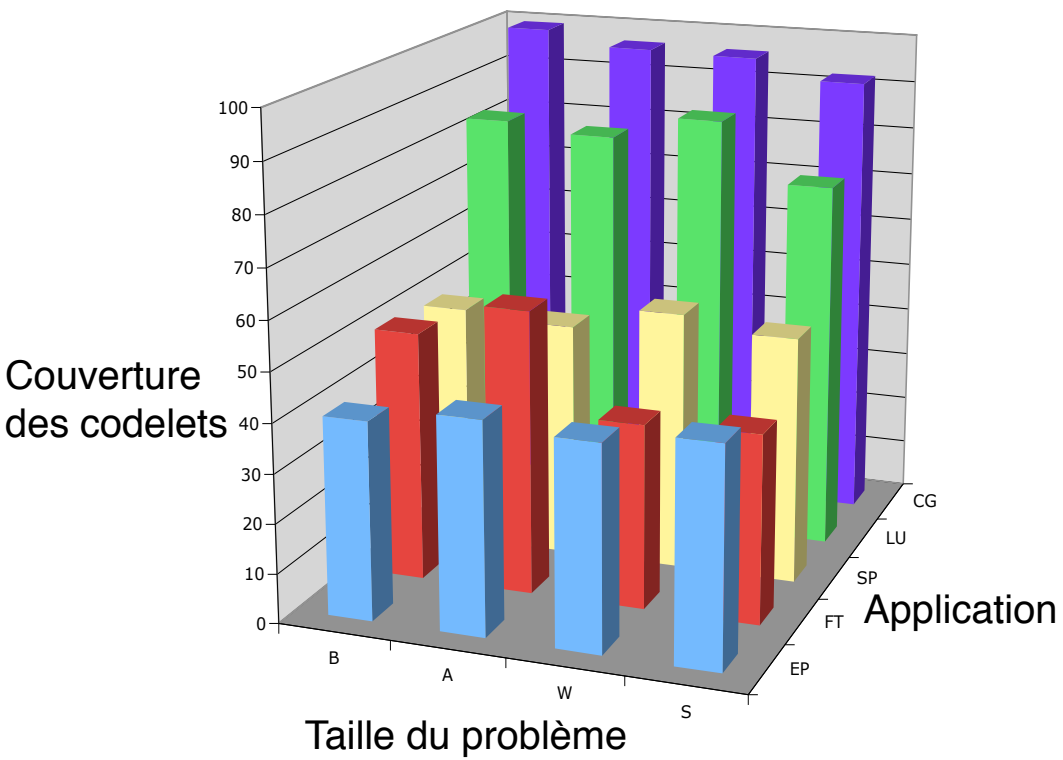
  NAS, H264, SPEC2006



# Expérimentations

- Stabilité des codelets

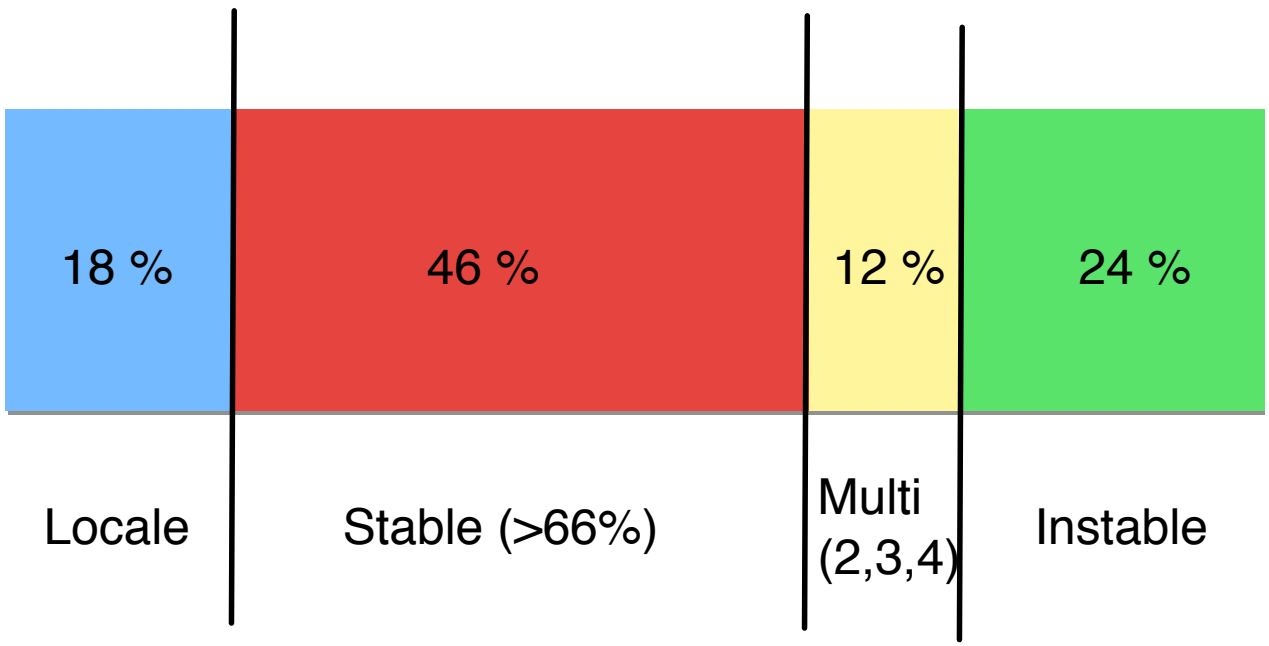
  NAS



# Expérimentations

- Spéculation sur les données scalaires

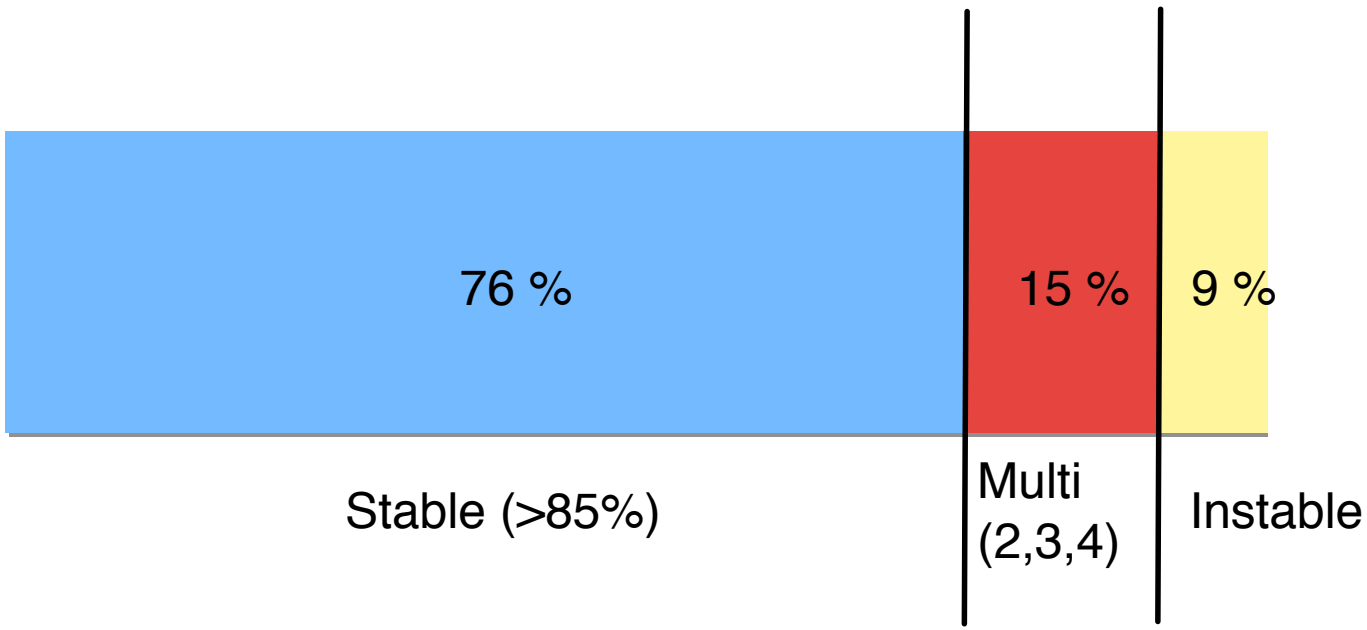
  589 variables de codelets observées dans NAS, H264, SPEC2006



# Expérimentations

- Spéculation sur les alignements de pointeurs

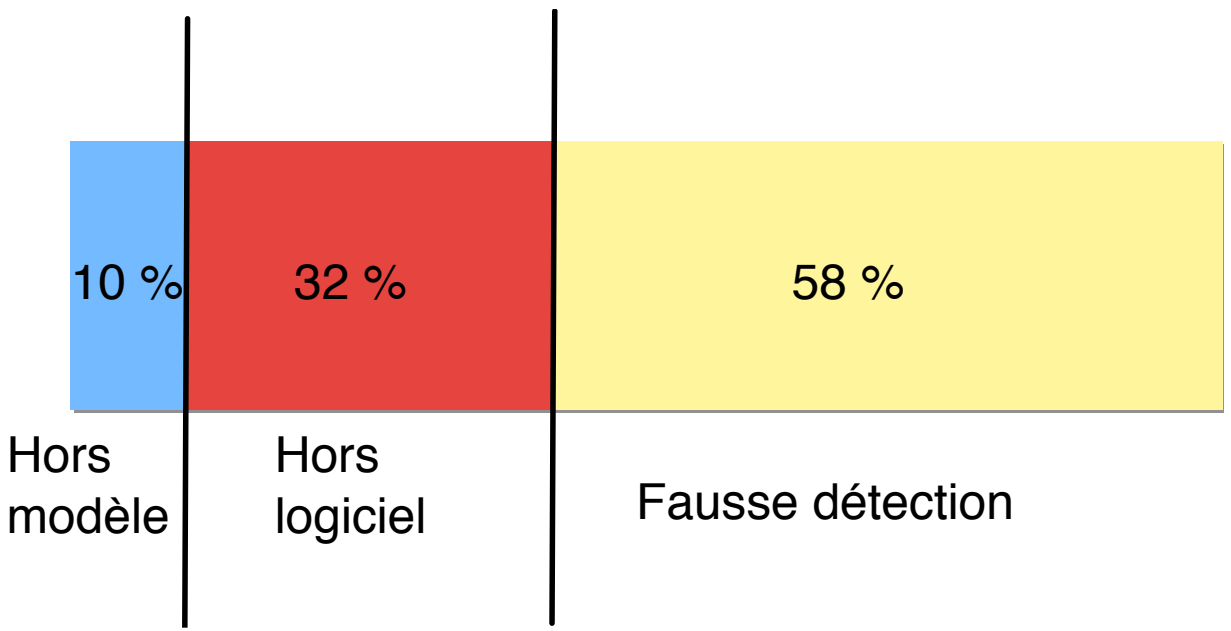
  905 zones mémoire observées dans NAS, H264, SPEC2006



# Expérimentations

- Causes de rejet lors de la transistion hotpath-codelet

  Sur 417 hotpaths des SPEC2006, 212 rejets



# Conclusions partielles

- L'outil de partitionnement d'application mis au point est :
  - fiable : noyaux détectés
  - robuste : majorité des applications traitées
  - efficace : résultats pertinents
  - utilisable : complexité maîtrisée

# Conclusions partielles

- L'outil de partitionnement d'application mis au point est :
  - fiable : noyaux détectés
  - robuste : majorité des applications traitées
  - efficace : résultats pertinents
  - utilisable : complexité maîtrisée
- Le modèle et ses informations spéculatives couvrent la majeure partie des utilisations

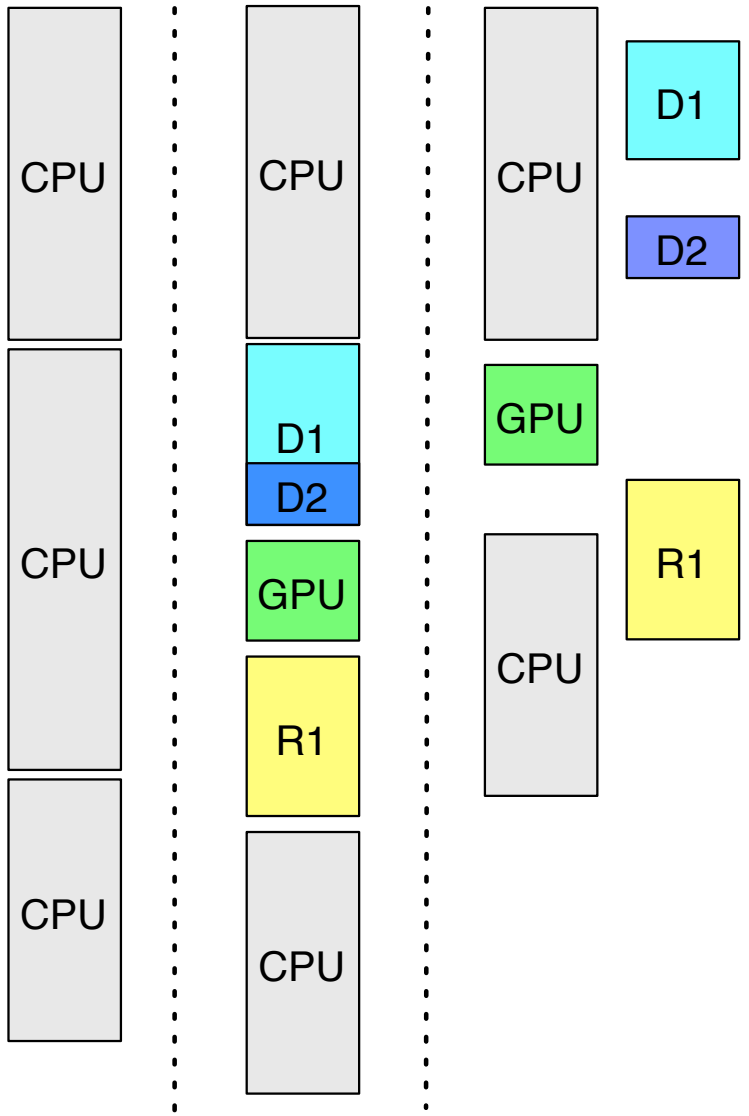
# Conclusions partielles

- L'outil de partitionnement d'application mis au point est :
  - fiable : noyaux détectés
  - robuste : majorité des applications traitées
  - efficace : résultats pertinents
  - utilisable : complexité maîtrisée
- Le modèle et ses informations spéculatives couvrent la majeure partie des utilisations
- Associé aux fonctionnalités d'HMPP, on se rapproche d'un outil automatique pour l'utilisation de co-processeur

# L'utilisation d'accélérateurs matériels

## Problèmes :

- Les transferts de données entre processeur et co-processeur sont très coûteux
  $\Rightarrow$ limite l'accélération potentielle



# L'optimisation des communications

## Deux voies principales d'optimisations

- Diminuer le volume de données à copier
  - prédiction de valeurs
  - zones résidentes
  - résolution des alias
  - modification des limites du codelet

$\Rightarrow$ Astex le fait déjà

# L'optimisation des communications

## Deux voies principales d'optimisations

- Diminuer le volume de données à copier
  - prédiction de valeurs
  - zones résidentes
  - résolution des alias
  - modification des limites du codelet

  $\Rightarrow$ Astex le fait déjà

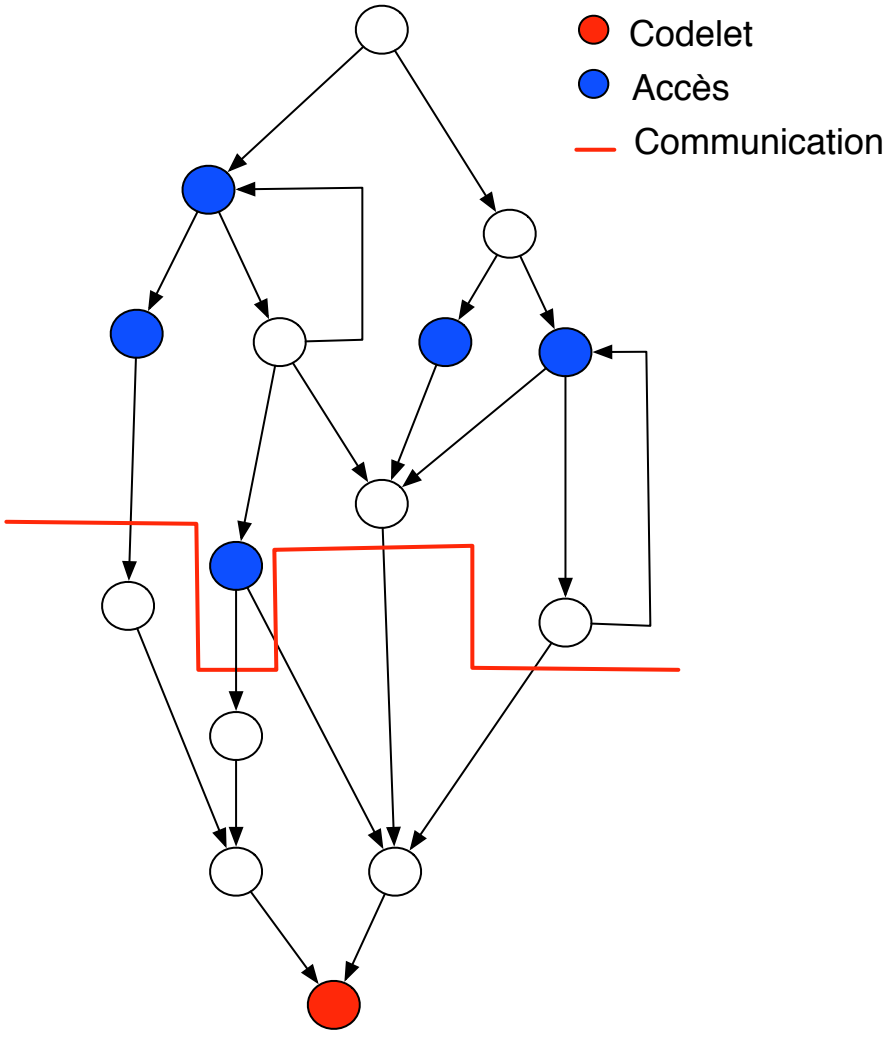
- Précharger les données en évitant les chargements inutiles

  $\Rightarrow$ **L'objet de ma seconde contribution**

# L'optimisation des communications

## Objectifs

- Effectuer les communications de manière **asynchrone**
  - au plus tôt
  - sans redondance
- Condition de validité



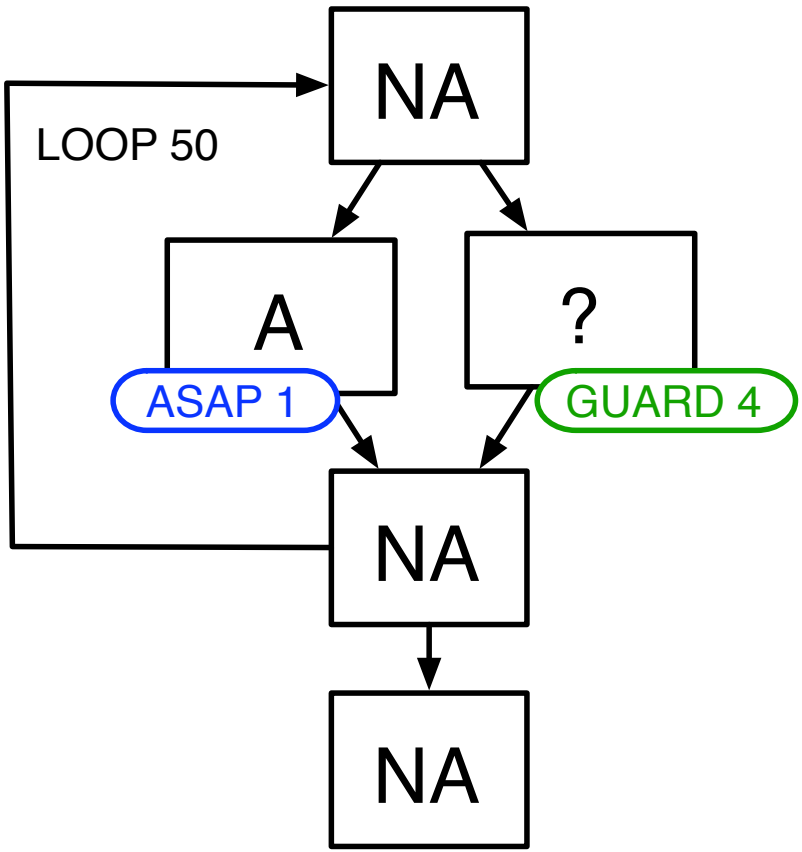
# Une approche basée sur ASTEX

## La classification des accès

| S | Statut Statique | D | Statut Spéculatif |
|---|---|---|---|
| A | Zone accédée | A | Zone accédée |
| NA | Zone non accédée | ? | Accès à la zone non vu |
| ? | Indéterminé | ? | Pas d'information |

1. {(A, A) ; (A, ?) ;( ?,A)} : accès sûr ou probable
   ⇒ communication au plus tôt : **ASAP**
2. {( ?, ?)} : accès peu probable ou improbable
   ⇒ communication si accès : **GUARD**
3. {(NA, ?)} : Il n'y a pas d'accès

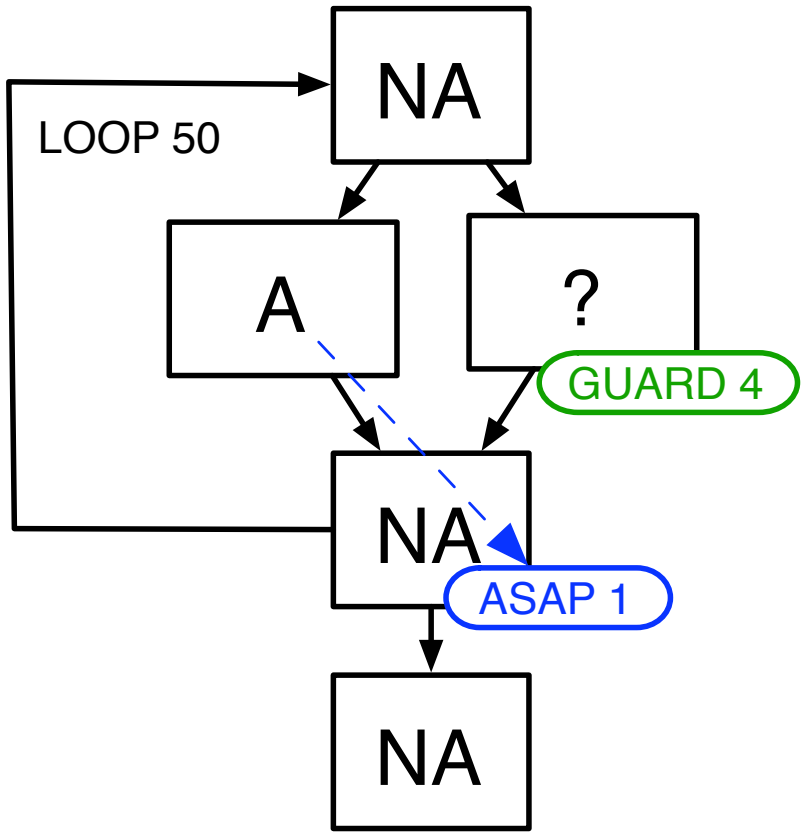
# Exemple de placement des communications




ASAP 1
Data= tab[1..50]

GUARD 4
Data=?[10..15]

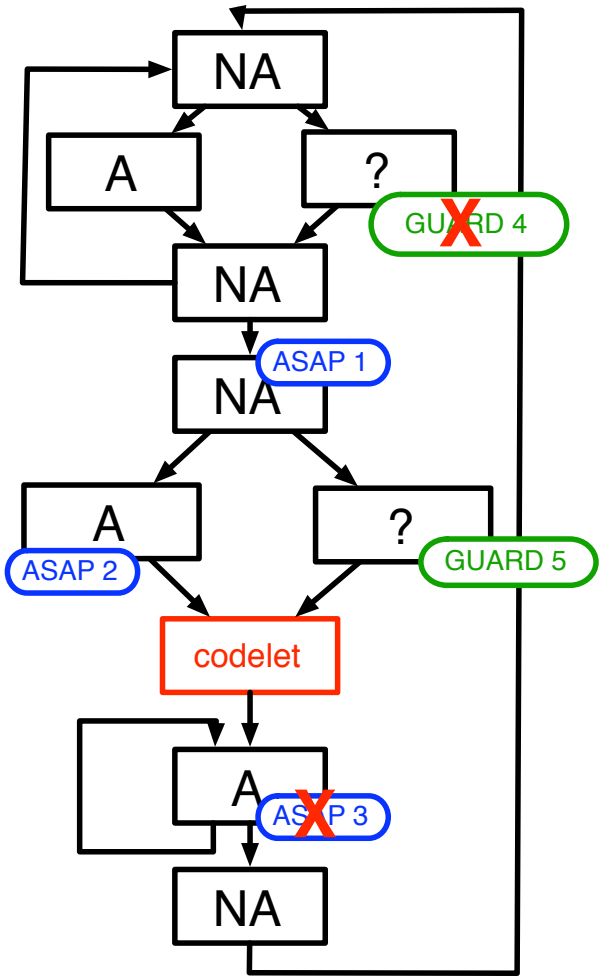
# Exemple de placement des communications




ASAP 1
Data= tab[1..50]

GUARD 4
Data=?[10..15]

# Exemple de placement des communications

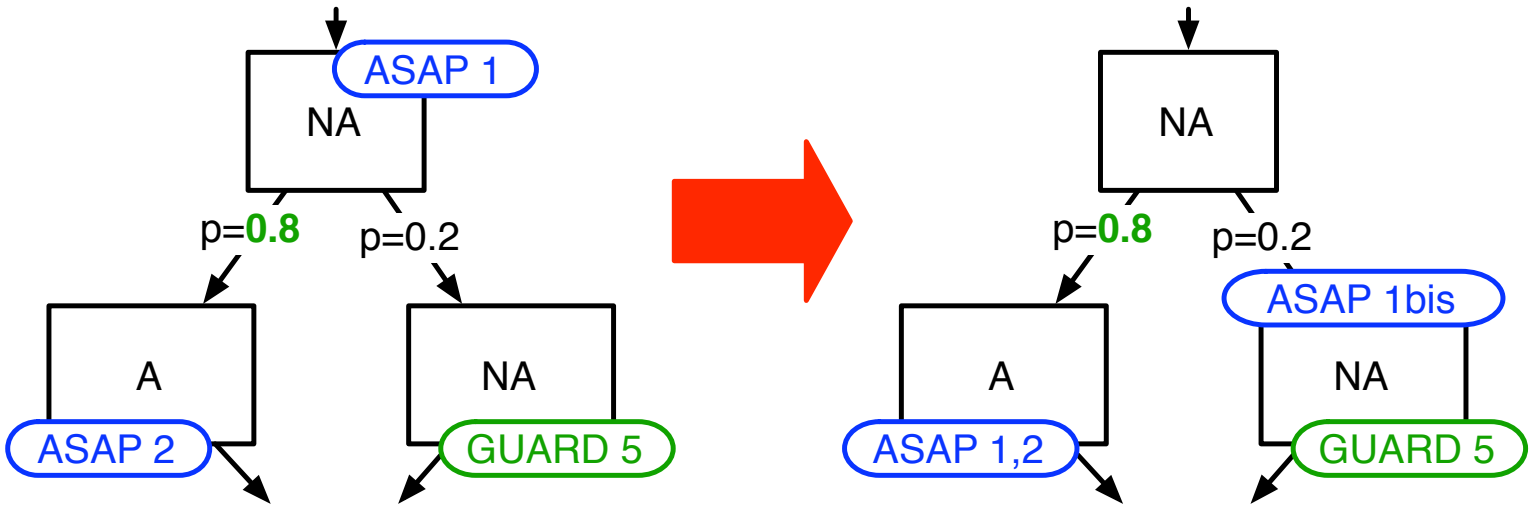


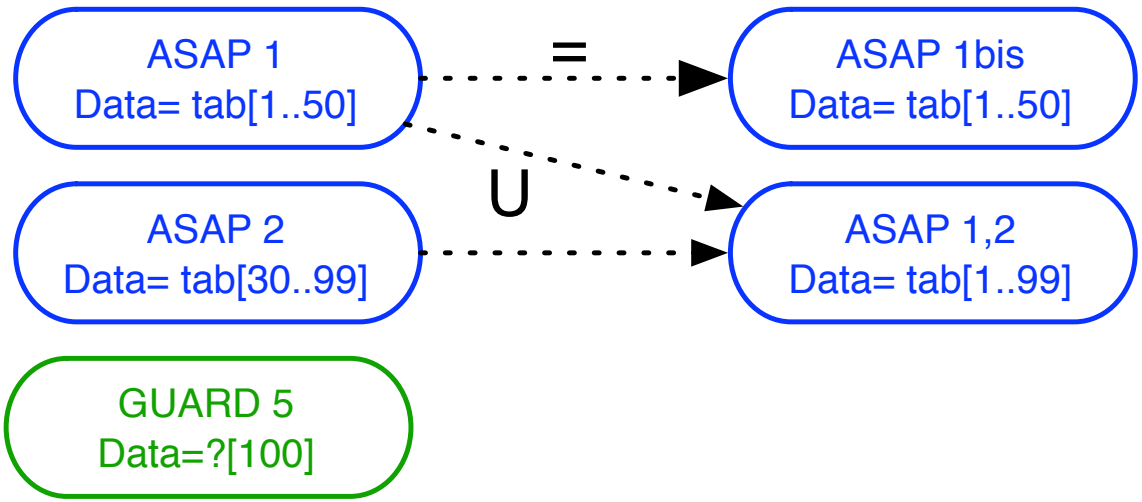

ASAP 1
Data= tab[1..50]

GUARD 4
Data=?[10..15]

ASAP 3
Data= tab[1..25]

# Exemple de placement des communications



# Une première approche simple basée sur ASTEX

## Avantages et limitation

- Relativement efficace
- Peu coûteuse à mettre en œuvre
- Tributaire de la qualité du profil spéculatif

# Expérimentation sur l'optimisation des communications

## Application de test

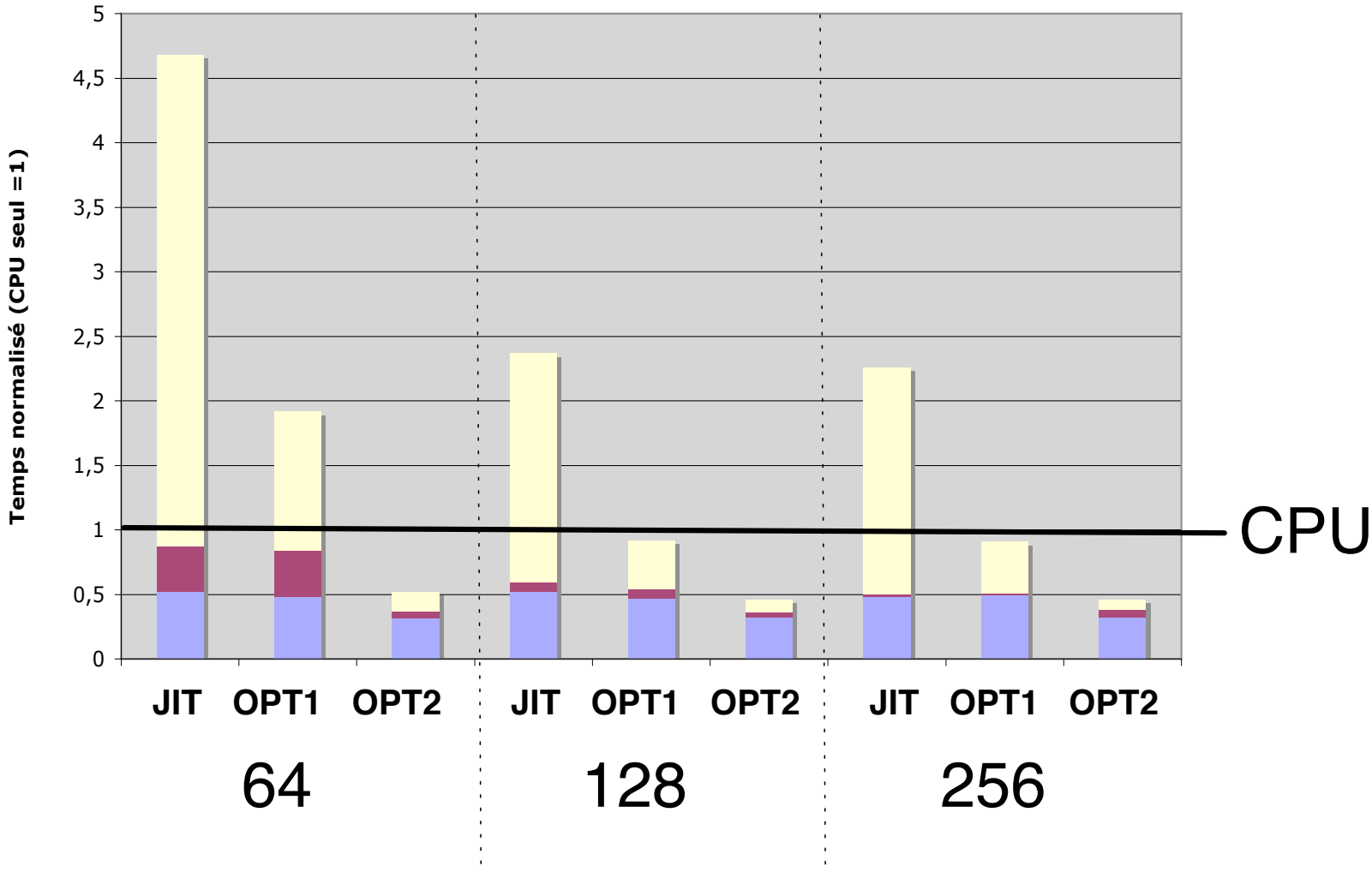
- **FFT3D** de la suite "NAS parallel benchmark"
- **Objectif :** montrer l'intérêt de l'optimisation des communications pour obtenir une accélération

# Expérimentation sur l'optimisation des communications

## Protocole

- Le code pour le GPU est écrit à la main (CUDA-HMPP)
- Un premier niveau d'optimisation avec Astex
  - zones résidentes
  - préchargement asynchrone simple
- Le second niveau
  - modification des limites du codelet
  - préchargement asynchrone interprocédural

# Expérimentation sur l'optimisation des communications



# Conclusions sur mon étude

## Le modèle d'Astex

- L'expérimentation valide l'intérêt et la pertinence de l'approche choisie
- Il est compatible avec d'autres langages de haut niveau

## L'implémentation d'Astex

- Astex est aujourd'hui en phase finale d'industrialisation
- Le logiciel est déjà en production chez certains clients
- Le développement continue, sur les bases de fonctionnalité du premier prototype, mais aussi sur des fonctionnalités nouvelles (e.g. stand-alone codelet)
- Des versions Fortran et Java sont en projets

# Conclusions sur mon étude

## Conclusions sur l'optimisation des communications

- Une optimisation critique pour l'utilisation de co-processeur
- Une modélisation formelle du problème
- Des trois algorithmes mis au point, le plus simple et moins complexe apparaît suffisant sur des exemples synthétiques

# Perspectives

- Multiples déréférences pour les structures régulières
- Utilisation étendue de la spéculation

- Parallélisation, ordonnancement des tâches, allocation des ressources (StarPU)
- Utilisation du modèle comme stratégie de compilation ou de conception d'architecture
- Introduire des éléments permettant l'utilisation de technologie JIT (e.g. TEST, Mitosis)

# Conclusions sur le partitionnement d'application

## Un problème qui reste ouvert

- La génération automatique d'applications partitionnées est indécidable, il y a de nombreux choix possibles générant un large espace de solutions
- Besoin d'uniformisation du logiciel et des architectures pour assurer l'efficacité des outils automatiques. (e.g. OpenCL)

# Merci de votre attention, avez-vous des questions ?