Compilation polyhédrale

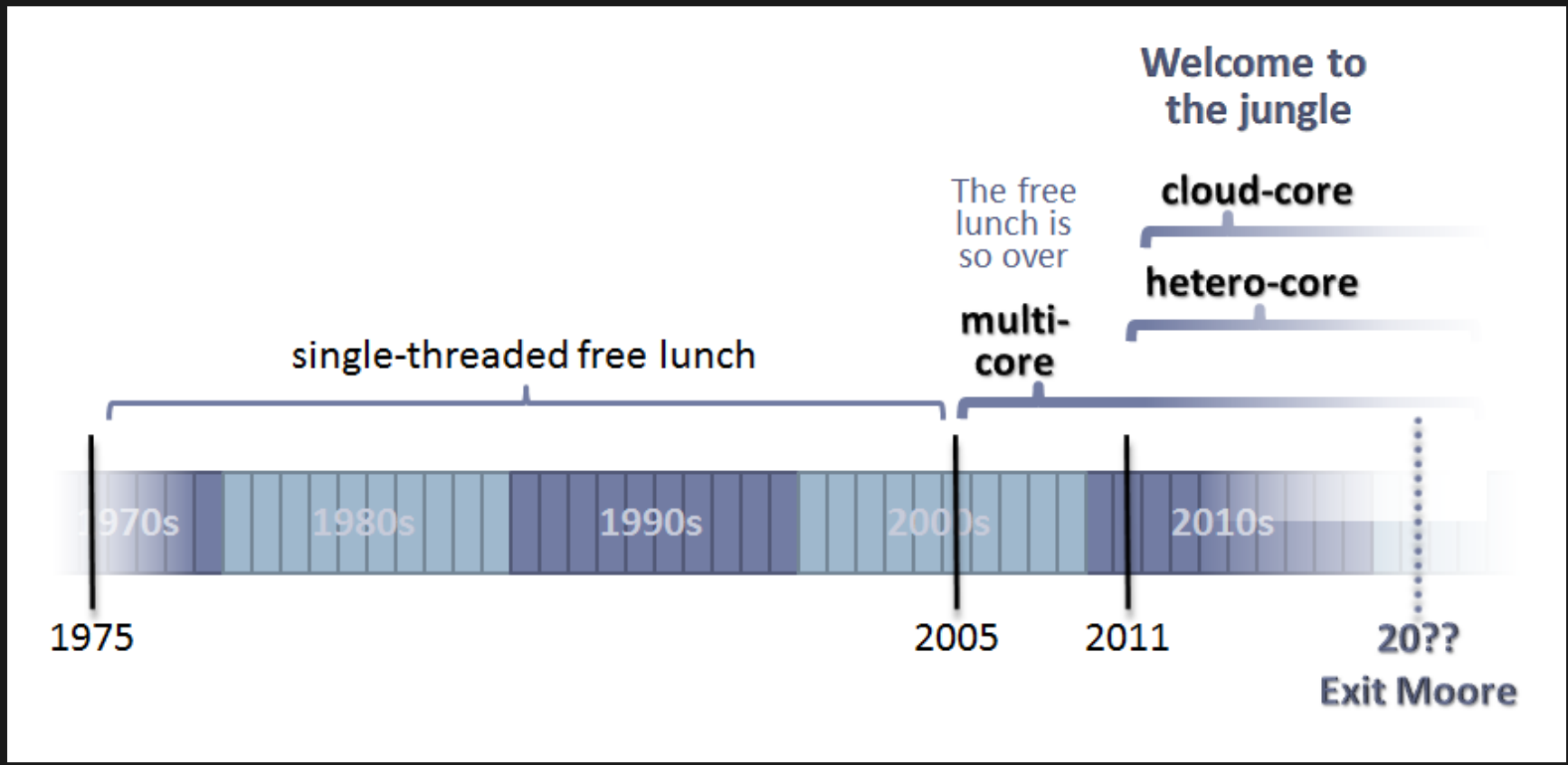
Félix-Antoine Ouellet

Université de Sherbrooke

4 décembre 2014

Plan

L'ère du parallélisme



Problèmes courants

Rendre le parallélisme accessible

On cherche toujours les meilleures abstractions pour le calcul parallèle

- *Threads*
- Tâches
- Langages dédiés
- Parallélisme implicite

Problèmes courants

Parallélisation d'applications existantes

Comment améliorer la performance de *legacy code*?

- Mettre tout à terre et recommencer
- Payer des développeurs pour améliorer des sections critiques
- Espérer qu'un outil améliore magiquement la situation

Problèmes courants

Compilateurs

Les compilateurs modernes ont beaucoup de difficulté à extraire du parallélisme d'applications écrites de façon séquentielles

- Problèmes théoriques très difficile à résoudre
- Aucun succès majeur jusqu'à maintenant
- Les compilateurs modernes manquent d'outils pour la tâche

Plan

Notions importantes

- Transforme un programme écrit dans un langage (de haut niveau) en un programme écrit dans un autre langage (de bas niveau).
- Maintient la sémantique du programme original.

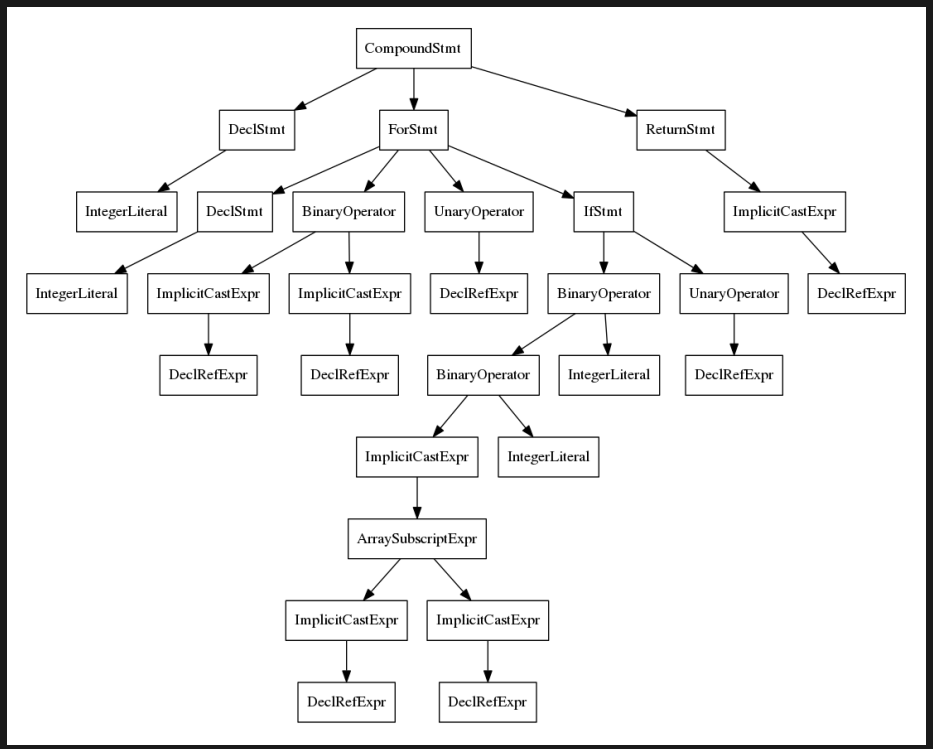
Architecture usuelle

Étape 1 - *Frontend*

- *Lexing*
- *Parsing*
- Analyse sémantique
- Travaille sur le code et sur un AST

Étape 1 - *Frontend*
AST

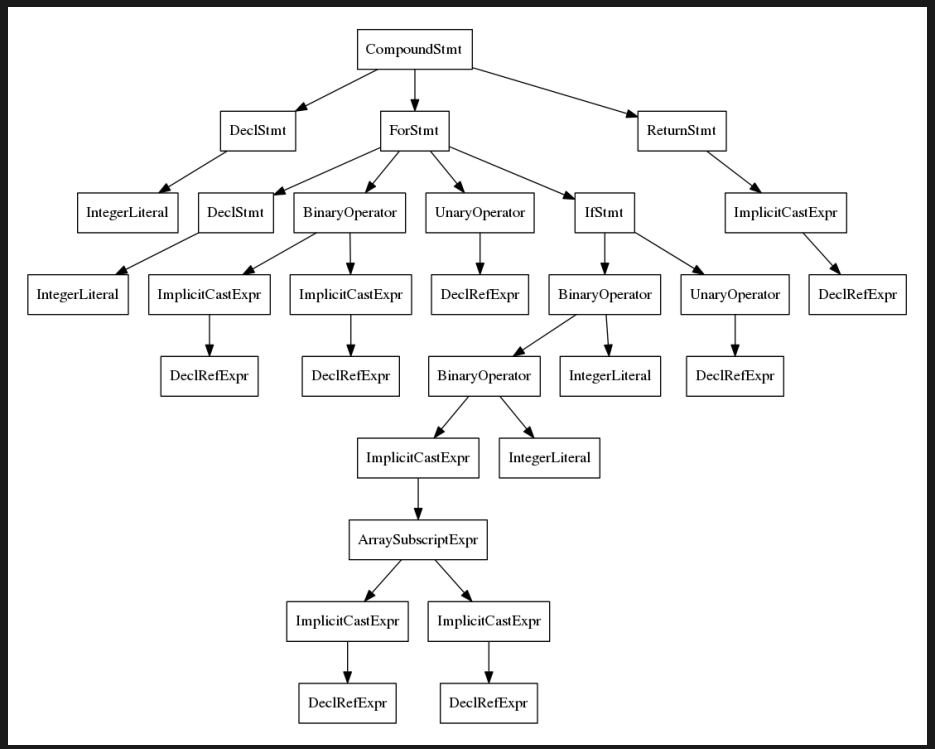
```
int countOdd(int A[],
             int N) {
  int cpt = 0;
  for (int i = 0; i < N;
  ++i)
    if (A[i] % 2 == 1)
      cpt++;
  return cpt;
}
```

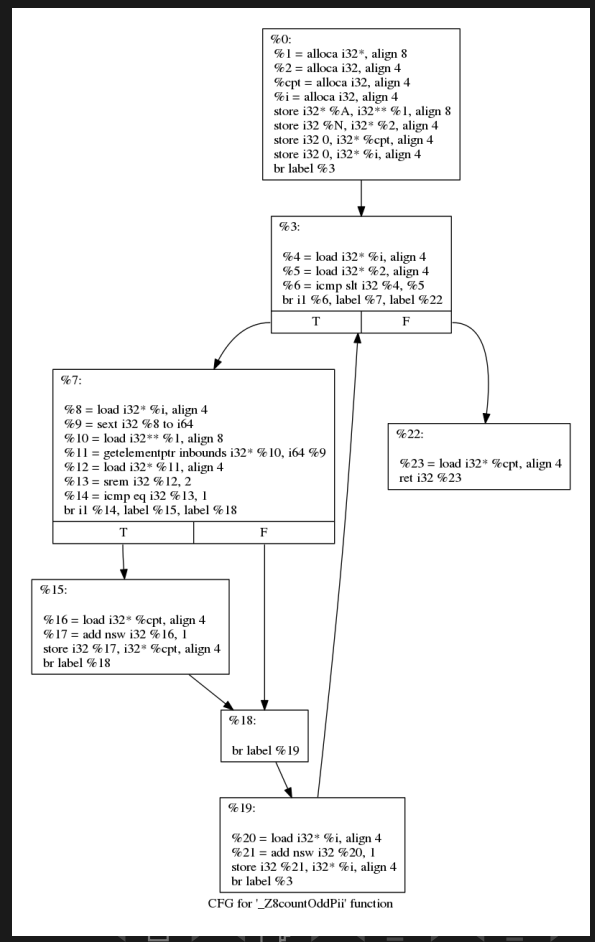


Étape 2 - Optimisateur

- Analyse le flot des données
- Optimisations indépendantes de la machine
- Travaille sur un CFG

Étape 2 - Optimisateur
CFG





Étape 3 - *Backend*

- Optimisations spécifiques à la machine
- Génére le code machine
- Travaille sur un CFG

Représentation intermédiaire
Illustration

```
%0:
 %1 = alloca i32*, align 8
 %2 = alloca i32, align 4
 %cpt = alloca i32, align 4
 %i = alloca i32, align 4
 store i32* %A, i32** %1, align 8
 store i32 %N, i32* %2, align 4
 store i32 0, i32* %cpt, align 4
 store i32 0, i32* %i, align 4
 br label %3
```

Représentation intermédiaire
Forme SSA

- Chaque variable est affectée une seule fois
- Chaque variable est définie avant d'être utilisée
- Une variable dans un programme est représentée par plusieurs variables en forme SSA

Représentation intermédiaire
Avantages

- Simplifie et améliore les analyses de flot de données
- Simplifie et améliore les optimisations indépendantes de la machine
- Permet de vérifier plus facilement les transformations effectuées

Représentation intermédiaire
Limitations

- Requiert des extensions pour bien optimiser les boucles
- Ne permet pas d'effectuer plusieurs transformations de boucles simultanément
- Requiert des extensions pour gérer le parallélisme

Plan

Cible

L'optimisation polyhédrale touche les parties à controle statique (SCoP) d'un programme

SCoP
Définition

- Ensemble d'énoncés dans une boucle
- Boucle dont les bornes sont des fonctions affines des itérateurs et paramètres avoisinants
- Accès mémoire doivent être des fonctions affines des itérateurs et paramètres avoisinants

SCoP
Exemple

```
for (int i = 0; i < 32; ++i)
  for (int j = 0; j < 1000; ++j)
    A[i][j] += 10;
```

Représentation polyhédrale

Composantes

Une représentation polyhédrale doit pouvoir encapsuler toutes les informations concernant:

- Le domaine de chaque énoncé
- L'ordre d'exécution de chaque instances de chaque énoncé
- Les accès mémoire effectuées par chaque énoncé

Représentation polyhédrale

Exemple

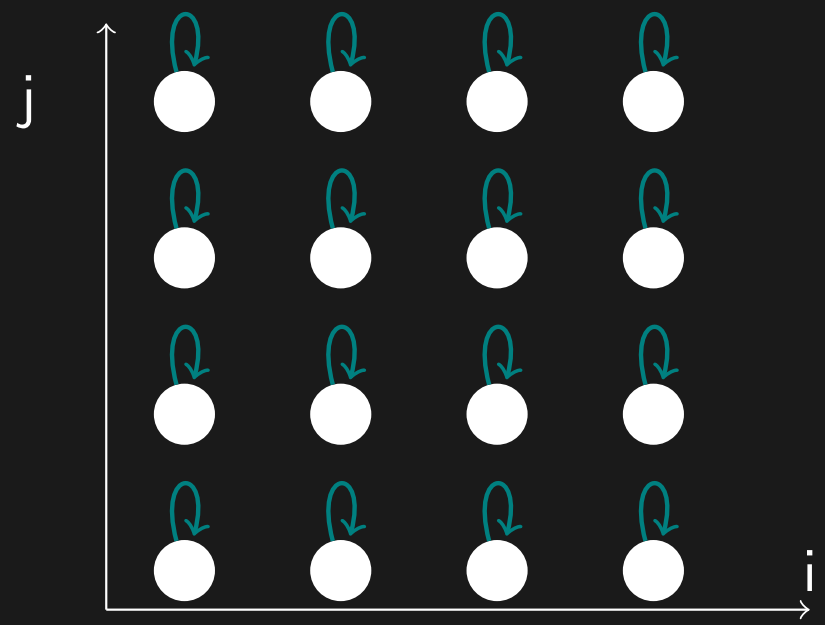
```
for (int i = 0; i < 32; ++i)
  for (int j = 0; j < 1000; ++j)
    A[i][j] += 10; // Stmt1
```

$$D_{Stmt1} = \{Stmt1[i,j] : 0 \leq i < 32 \wedge 0 \leq j < 1000\}$$
$$S_{Stmt1} = \{Stmt1[i,j] \rightarrow [i,j]\}$$
$$A_{Stmt1} = \{Stmt1[i,j] \rightarrow A[i,j]\}$$

Représentation polyhédrale

Illustration

```
for (int i = 0; i < 32; ++i)
  for (int j = 0; j < 1000; ++j)
    A[i][j] += 10; // Stmt1
```



Transformations

- Une transformation consiste à appliquer une opération algébrique sur l'ordre d'exécution du *SCoP* traité
- Plusieurs transformations peuvent être effectuée en un seul calcul

Exemple - Partie 1

$$T_{Interchange} = \{[i,j] \rightarrow [j,i]\}$$

$$D_{Stmt1} = \{Stmt1[i,j] : 0 \leq i < 32 \wedge 0 \leq j < 1000\}$$
$$S'_{Stmt1} = S \circ T_{Interchange}$$
$$A_{Stmt1} = \{Stmt1[i,j] \rightarrow A[i,j]\}$$

```
for (int j = 0; j < 1000; ++j)
  for (int i = 0; i < 32; ++i)
    A[i][j] += 10; // Stmt1
```

Exemple - Partie 2

$$T_{Interchange} = \{[i,j] \rightarrow [j,i]\}$$
$$T_{StripMine} = \{[i,j] \rightarrow [i,jj,j] : jj \mod 4 = 0 \wedge jj \leq j < jj + 4\}$$

$$D_{Stmt1} = \{Stmt1[i,j] : 0 \leq i < 32 \wedge 0 \leq j < 1000\}$$
$$S'_{Stmt1} = S \circ T_{Interchange} \circ T_{StripMine}$$
$$A_{Stmt1} = \{Stmt1[i,j] \rightarrow A[i,j]\}$$

```
for (int j = 0; j < 1000; ++j)
  for (int ii = 0; ii < 32; ii += 4)
    for (int i = ii; i < ii+4; ++i)
      A[i][j] += 10; // Stmt1
```

Limitations

- Accès non affines
- Boucles irrégulières
- Pointeurs

Plan

Intuition

- Le modèle polyhédral travaille sur les instances des énoncés

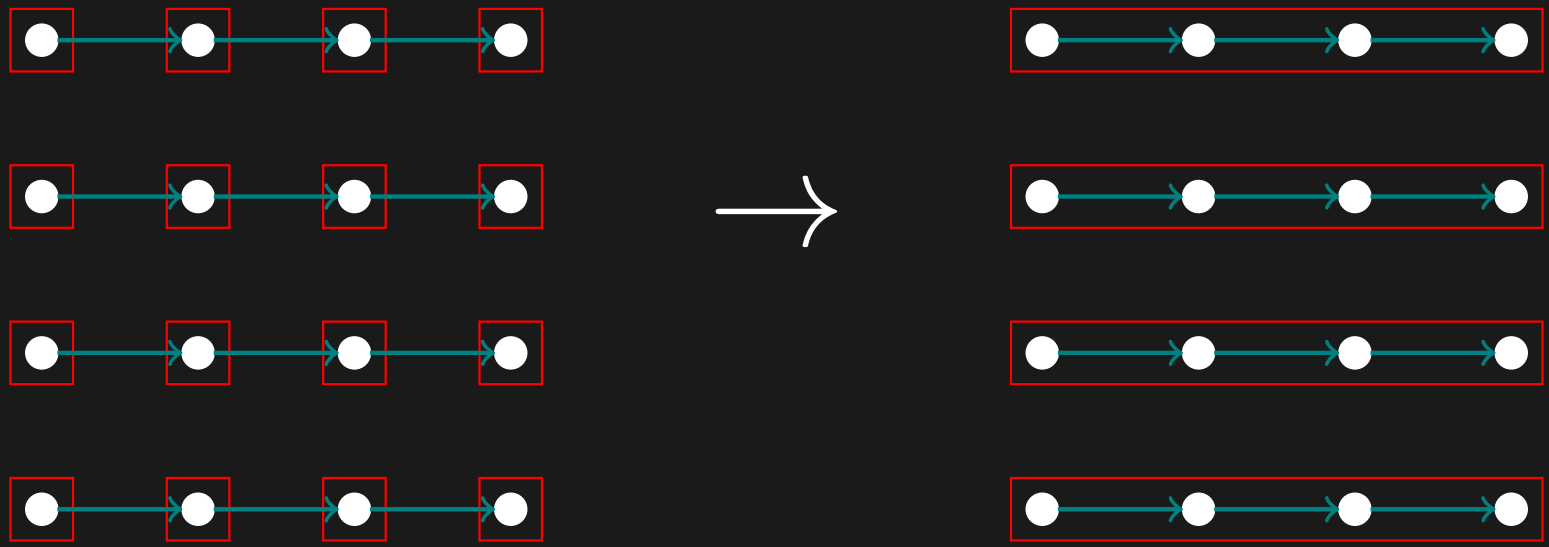
Intuition

- Le modèle polyhédral travaille sur les instances des énoncés
- Chaque instance peut être considéré comme une tâche

Intuition

- Le modèle polyhédral travaille sur les instances des énoncés
- Chaque instance peut être considéré comme une tâche
- Il suffit donc de trouver un plan d'exécution de ces tâches

Intuition
Illustration



Rappel

Une boucle peut être parallélisée si elle ne contient pas de dépendences mémoire portées par la boucle

Rappel

```
for (int i = 0; i < N; ++i) {
  A[i] = B[i] + C[i];
  D[i] = A[i] + 10;
}
```

```
for (int i = 0; i < N; ++i) {
  A[i+1] = A[i] + B[i];
}
```

Dépendences dans le modèle polyhédral

Pour obtenir les dépendences dans une boucle, il faut:

- Séparer les accès en écriture des accès en lecture
- Trouver l'intersection de ces ensembles d'accès
- Obtenir la distance entre ces accès

Dépendences dans le modèle polyhédral

Séparer les accès en écriture des accès en lecture

```
for (int i = 0; i < 100; ++i) {
  A[i] = B[i] + C[i]; // Stmt1
  D[i] = A[i] + 10;   // Stmt2
}
```

$$A_{Stmt1_W} = \{Stmt1[i] \rightarrow A[i]\}$$
$$A_{Stmt1_R} = \{Stmt1[i] \rightarrow \emptyset\}$$

$$A_{Stmt2_W} = \{Stmt2[i] \rightarrow \emptyset\}$$
$$A_{Stmt2_R} = \{Stmt2[i] \rightarrow A[i]\}$$

Dépendences dans le modèle polyhédral

Trouver l'intersection de ces ensembles d'accès

```
for (int i = 0; i < 100; ++i) {
  A[i] = B[i] + C[i]; // Stmt1
  D[i] = A[i] + 10;   // Stmt2
}
```

$$A_{Stmt1_W} \cap A_{Stmt1_R} = \emptyset$$
$$A_{Stmt1_W} \cap A_{Stmt2_R} = A[i]$$
$$A_{Stmt2_W} \cap A_{Stmt1_R} = \emptyset$$
$$A_{Stmt2_W} \cap A_{Stmt2_R} = \emptyset$$

Dépendences dans le modèle polyhédral

Obtenir la distance entre ces accès

```
for (int i = 0; i < 100; ++i) {
  A[i] = B[i] + C[i]; // Stmt1
  D[i] = A[i] + 10;   // Stmt2
}
```

$$A_{Stmt1_W} = \{Stmt1[i] \rightarrow A[i]\}$$
$$A_{Stmt2_R} = \{Stmt2[i] \rightarrow A[i]\}$$
$$\longrightarrow 0$$

Génération de code parallèle

```
#pragma omp parallel
for (int i = 0; i < 100; ++i) {
  A[i] = B[i] + C[i]; // Stmt1
  D[i] = A[i] + 10;   // Stmt2
}
```

Plan

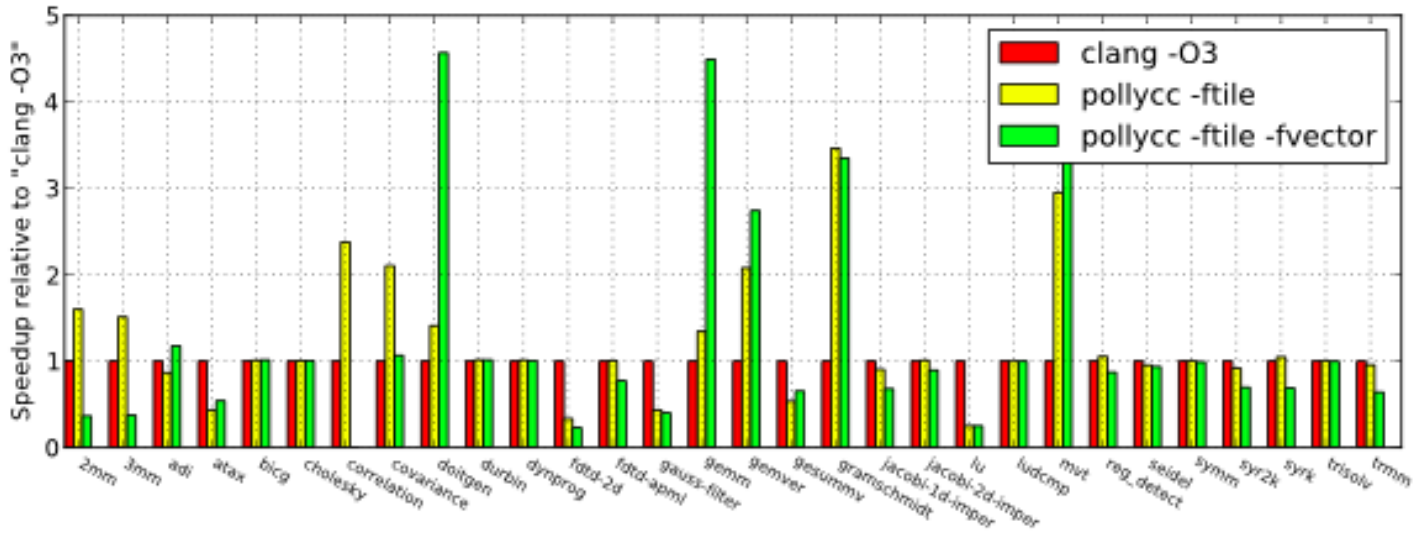
Résultats obtenus

Contexte

- PolyBench
- Clang -O3
- Polly
- Valeurs en virgule flottante avec double précision
- Intel Xeon X5670 @ 2.93 GHz (12 coeurs, 24 *threads*)

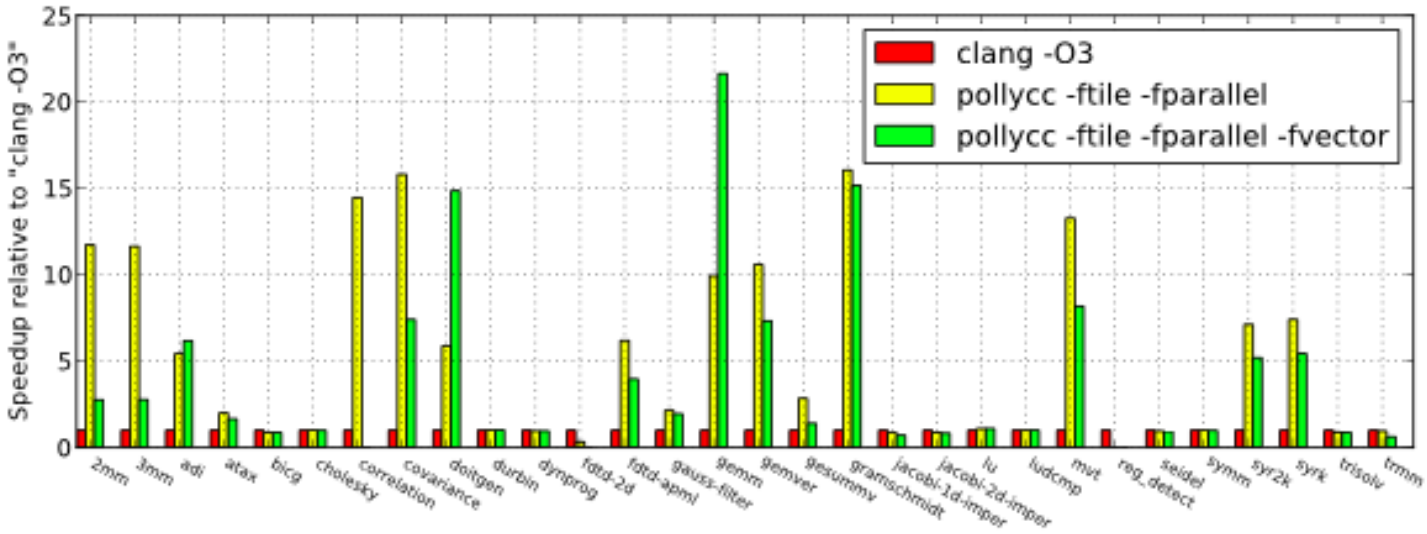
Résultats obtenus

Séquentiel



Résultats obtenus

Parallèle



Support présent

- LLVM (Polly)
- GCC (Graphite)
- Langages expérimentaux (X10)
- Plateformes expérimentales (PLUTO)

Plan

Conclusion

- Offre une façon différente de raisonner sur l'optimisation de boucles et la parallélisation automatique
- Représente possiblement la meilleure chance de produire du parallélisme implicite