Compilation polyhédrale

Félix-Antoine Ouellet

Université de Sherbrooke

4 décembre 2014

1. Motivation
2. Compilation traditionnelle
3. Analyse de dépendences
4. Approche polyhédrale
5. Parallélisation automatique
6. Conclusion

Plan

1 Motivation

2 Compilation traditionnelle

3 Analyse de dépendences

4 Approche polyhédrale

5 Parallélisation automatique

6 Conclusion

L'ère du parallélisme

Problèmes courants

Optimisations des boucles

Les compilateurs modernes sont très loin derrière la théorie

- Fusion
- Tuilage
- *Skewing*
- Etc...

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

Rendre le parallélisme accessible

On cherche toujours les meilleures abstractions pour le calcul parallèle

- *Threads*
- Tâches

Plan

1 Motivation

2 Compilation traditionnelle
- Bases de la compilation
- Processus de compilation

3 Analyse de dépendences

4 Approche polyhédrale

5 Parallélisation automatique

6 Conclusion

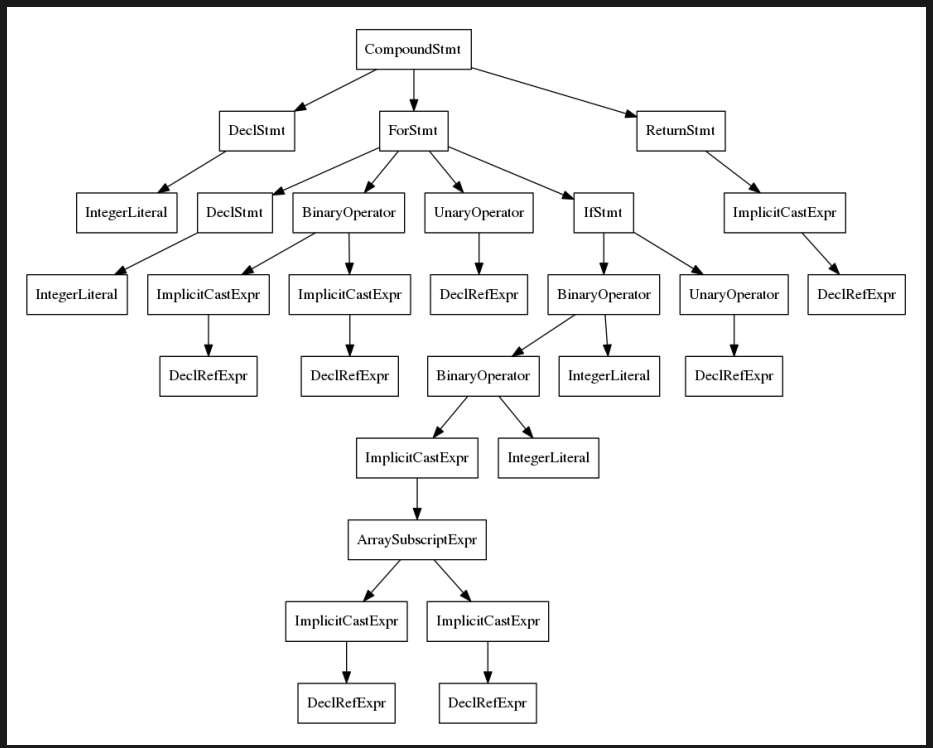
Notions importantes

- 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.

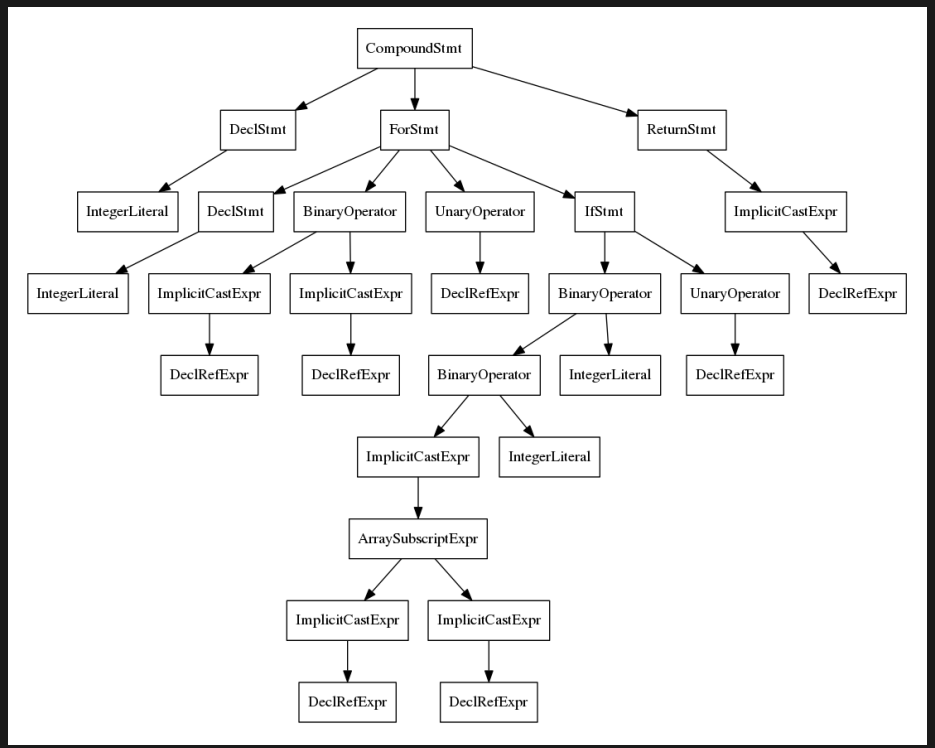
Architecture usuelle

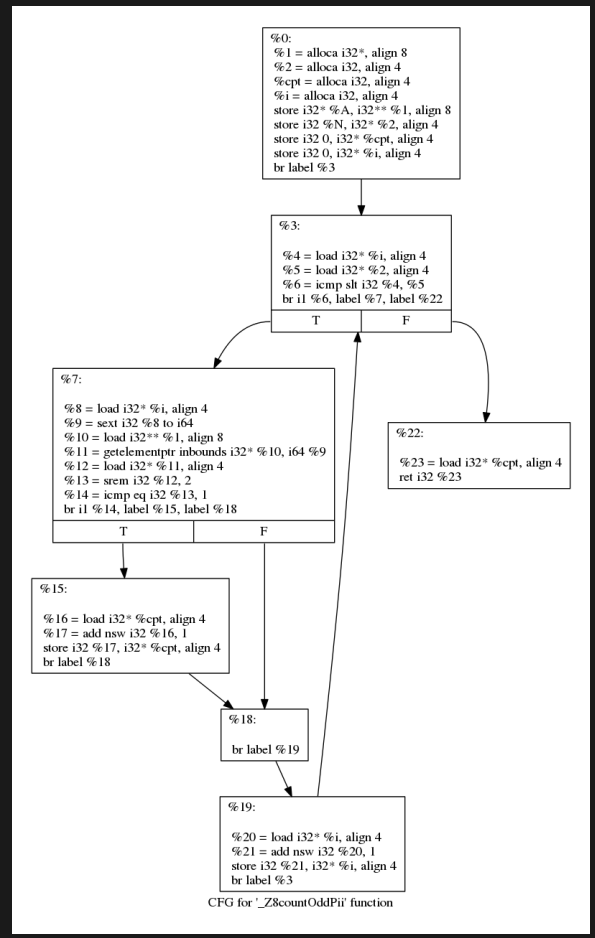
Étape 1 - AST

```
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 - CFG


CompoundStmt
DeclStmt
ForStmt
ReturnStmt
IntegerLiteral
DeclStmt
BinaryOperator
UnaryOperator
IfStmt
ImplicitCastExpr
IntegerLiteral
ImplicitCastExpr
ImplicitCastExpr
DeclRefExpr
BinaryOperator
UnaryOperator
DeclRefExpr
DeclRefExpr
DeclRefExpr
BinaryOperator
IntegerLiteral
DeclRefExpr
ImplicitCastExpr
IntegerLiteral
ArraySubscriptExpr
ImplicitCastExpr
ImplicitCastExpr
DeclRefExpr
DeclRefExpr


%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
%3:
%4 = load i32* %i, align 4
%5 = load i32* %2, align 4
%6 = icmp slt i32 %4, %5
br i1 %6, label %7, label %22
T
F
%7:
%8 = load i32* %i, align 4
%9 = sext i32 %8 to i64
%10 = load i32** %1, align 8
%11 = getelementptr inbounds i32* %10, i64 %9
%12 = load i32* %11, align 4
%13 = srem i32 %12, 2
%14 = icmp eq i32 %13, 1
br i1 %14, label %15, label %18
T
F
%22:
%23 = load i32* %cpt, align 4
ret i32 %23
%15:
%16 = load i32* %cpt, align 4
%17 = add nsw i32 %16, 1
store i32 %17, i32* %cpt, align 4
br label %18
%18:
br label %19
%19:
%20 = load i32* %i, align 4
%21 = add nsw i32 %20, 1
store i32 %21, i32* %i, align 4
br label %3
CFG for '_Z8countOddPii' function

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

Raisonnement

- Mieux analyser le programme donné
- Permettre des optimisations indépendantes de la machine
- Vérifier les optimisations effectuées

Plan

1 Motivation

2 Compilation traditionnelle

3 Analyse de dépendences

4 Approche polyhédrale

5 Parallélisation automatique

6 Conclusion

Définition

Situation dans laquelle deux instructions accèdent à la même donnée.

Importance des dépendences

Indique quelles transformations sont légales

```
for (int i = 0; i < N; ++i)
  for (int j = 0; j < N; ++j)
    A[i][j] = A[i+1][j-1];
```

✓ Déroulement

× Inter-échange

Importance des dépendences

Indique les opportunités de parallélisme

```
for (int i = 0; i < N; ++i)
  for (int j = 0; j < N; ++j)
    A[i][j] = A[i+1][j-1];
```

- Parallélisme d'intructions au niveau de la boucle interne
- Parallélisme de fils d'exécution au niveau de la boucle externe

Représentation
Vecteur de distance

Vecteur **d(i, j)** tel que $\mathbf{d(i, j)}_k = \mathbf{j}_k$ - $\mathbf{i}_k$ où **i** et **j** sont des vecteurs d'itérations.

Représentation

Vecteur de distance

```
for (int i = 0; i < N; ++i)
  for (int j = 0; j < M; ++j)
    for (int k = 0; k < L; ++k)
      A[i+1][j][k-1] = A[i][j][k] + 10;
```

$\mathbf{d(i, j)}_k = (-1, 0, 1)$

Représentation
Vecteur de direction

Vecteur **D(i, j)** tel que:

$$D(i,j)_k = \begin{cases} " < " & \text{si } d(i,j)_k > 0 \\ " = " & \text{si } d(i,j)_k = 0 \\ " > " & \text{si } d(i,j)_k < 0 \end{cases}$$

Représentation
Vecteur de direction

```
for (int i = 0; i < N; ++i)
  for (int j = 0; j < M; ++j)
    for (int k = 0; k < L; ++k)
      A[i+1][j][k-1] = A[i][j][k] + 10;
```

$\mathbf{D(i, j)}_k = (<, =, >)$

Tests de dépendences

Points à améliorer

Ce modèle éprouve de la difficulté au niveau de:

- Parallélisme de niveau plus haut niveau que celui des instructions
- Parallélisme de tâches
- Distribution de données

Plan

1 Motivation

2 Compilation traditionnelle

3 Analyse de dépendences

4 Approche polyhédrale
- Représentation
- Optimisations
- Limitations

5 Parallélisation automatique

6 Conclusion

Représentation

Limitations

- Accès non affines
- Boucles irrégulières

Plan

1 Motivation

2 Compilation traditionnelle

3 Analyse de dépendences

4 Approche polyhédrale

5 Parallélisation automatique
- Mémoire partagée
- Mémoire distribuée
- Support présent

6 Conclusion

Intuition

Mémoire partagée

Mémoire distribuée

Support présent

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

Plan

1 Motivation

2 Compilation traditionnelle

3 Analyse de dépendences

4 Approche polyhédrale

5 Parallélisation automatique

6 Conclusion

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