Optimisation de code

Il serait souhaitable et intéressant que les compilateurs produisent du code cible de bonne qualité que celui écrit à la main, car il est difficile et pratiquement impossible de le faire.

Optimiser un code, revient à compacter les instructions et à les exécuter de façon plus rapide. Le code obtenu n'est pas surement la plus optimal. L'optimisation ne doit pas modifier le résultat d'exécution.

Optimiser un code veut dire améliorer sa qualité :

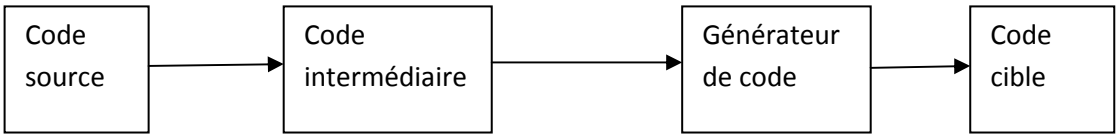
- ✓ Le rendre plus rapide.
- ✓ Le rendre plus compact.
- ✓ Le rendre moins gourmand en énergie et puissance de calcul.

Les compilateurs qui effectuent des transformations dans le but d'optimiser sont appelés les optimisants.

Il existe deux types d'optimisation :

- Les transformations de programme qui améliorent le code cible sans prendre en compte les propriétés de la machine cible.
- Les optimisations dépendantes de la machine, telles que l'allocation de registres et l'utilisation de séquences d'instructions spécifiques.

L'optimisation peut se faire sur plusieurs niveaux :



L'utilisateur peut améliorer

Le compilateur peut optimiser (boucles, appels,…).

Le compilateur peut utiliser des registres, sélectionner des instructions, transformer,…

Il existe différents types de transformation :

1) Transformations concernant les fonctionnalités

Un compilateur peut améliore un code de différentes manières sans changer sa fonctionnalité. Parmi ces opérations, nous citons : élimination des sous expressions communes, la propagation des copies,…

a) Elimination des sous expressions communes

Le but d'éliminer les sous expressions communes est de ne pas recalculer la même expression.
Une occurrence d'une expression E est appelée sous expression commune, si E a été calculée précédemment et si la valeur des variables apparaissant dans E n'a pas changé depuis le calcul précédent. Nous pouvons éviter de recalculer l'expression, si nous pouvons utiliser la valeur calculée précédemment.

Exemple

Avant	**Après**	
t6=4*i	t6=4*i	t6=4*i
x=a [t6]	x=a [t6]	x=a [t6]
t7=4*i	**t7=t6**	
t8=4*j	t8=4*j	t8=4*j
t9=a [t8]	t9=a [t8]	t9=a [t8]
a [t7]=t9	a[t7]=t9	a [**t6**]=t9
t10=4*j	**t10=t8**	
a [t10]=x	a [t10]=x	a [**t8**]=x

b) Propagation des constantes

Il est fréquent de rencontrer une variable dont la valeur est fixée une fois pour toutes. Cette valeur est par la suite utilisée dans des expressions et quelquefois additionnée ou multipliée à d'autres constantes. Il est donc utile de ne pas l'évaluer à l'exécution mais de le faire à la compilation.

Exemple :

PI:=3.14159
S :=(PI*(d**2))/4
Peut se réécrire de la façon suivante :
Kl :3.14159 /4
S :=Kl*(d**2))

c) Elimination de code inutile

L'analyse d'un programme peut faire apparaitre certains résultats de calcul qui sont jamais utilisés, il est donc inutile d'évaluer l'expression qui l'affecte. Il est alors nécessaire de connaitre les domaines d'utilisation des variables dans un programme.

Exemple :

X :=t3

a[t3] :=t5	Transformation	a[t3] :=t5
a[t4] :=t3	si X est inutilisée	a[t4] :=t3

Remarque

Ce cas peut arriver sans que le programmeur l'ai fait explicitement. Cela peut résulter d'une précédente transformation.

2) Optimisation des boucles

L'existence des boucles dans un programme augmente le temps d'exécution. Ce temps peut être optimisé si le nombre d'instruction dans les boucles est diminué.

a) Déplacement du code

Il consiste à transformer du code à l'extérieur des boucles.

Exemple :

Tant que i<=limite-2 ⟶ t :=limite-2

Tant que i<=t

b) L'élimination des variables d'induction

Exemple :

Etiq :j :=j-1

t4 :=4*j
t5 :=a[t4]
Si t5>v aller à etiq

Nous remarquons que les variables j et t sont liées car chaque fois que *j* décroit, t4 décroit de 4. De telles variables sont dits variables d'induction.

On remarque que t4 est toujours égale à 4*j dans la boucle. On peut alors écrire t4=4*j (à l'extérieur de le boule).

Dans la boucle

j=j-1

t4=t4-4

3) Optimisation de la complexité des opérations

Dans certains cas, les multiplications et les exponentiations peuvent être remplacées par des additions ou multiplications (resp). Car quelque soit le type de la machine, les multiplications sont plus coûteuses que les additions en temps d'exécution.