Rapport projet Cassiopée 93

Structural souce code properties

Julien DENIZE Cyprien LAMBERT
Tuteur: Jorge-Eleazar LOPEZ-CORONADO

25 mai 2019

Table des matières

1 Introduction **2**
1.1 Qu'est ce que l'optimisation . . . 2
1.1.1 Définition . . . 2
1.1.2 Problèmes à résoudre par optimisation . . . 2
1.1.3 Outils existants . . . 3
1.2 Notre approche . . . 3
1.2.1 Les Control Flow Graphs (CFG) . . . 3
1.2.2 Détection de code mort . . . 3
1.2.3 Notre projet . . . 3

2 Control Flow Graphs **4**
2.1 Introduction . . . 4
2.2 Nos recherches . . . 4
2.3 Implémentation . . . 4
2.3.1 Notre script de modifications des CFG . . . 4
2.3.2 Notre analyse des CFG . . . 4

3 Dead Code Detection **5**
3.1 Introduction . . . 5
3.2 Nos recherches . . . 5
3.3 Implémentation . . . 5

Chapitre 1

Introduction

1.1 Qu'est ce que l'optimisation

1.1.1 Définition

Selon la définition donnée par Wikipédia :

> En programmation informatique, l'optimisation de code est la pratique consistant à améliorer l'efficacité du code informatique ou d'une librairie logicielle. [1]

Ainsi l'optimisation en informatique fait partie intégrante du développement. En effet, il est crucial pour différentes raisons détaillés à la section suivante que les développeurs optimisent leur code afin d'améliorer son rendement.

Des recherches ont donc été effectuées durant de nombreuses années au début de l'informatique afin d'optimiser le compilateur par exemple, pour avoir des compilations plus rapides ou/et plus efficaces. Aujourd'hui encore, la rercheche en optimisation est importante car mêmes si les machines sont de plus en plus puissantes, les algorithmes sont de plus en plus gourmands également.

Le Big Data illustre parfaitement ceci, les machines peuvent traiter des millions et des millions de données, mais si les algorithmes mettent des mois à s'exécuter, alors cela aurait eu peu de valeur.

1.1.2 Problèmes à résoudre par optimisation

Les optimisations permettent d'améliorer nombreuses choses : une exécutions plus rapide, une place en mémoire réduite, une limitation des ressources consommées comme les fichiers et enfin une diminution de la consommation électrique.

En effet, les optimisations permettent de supprimer des choses inutiles par exemple le code suivant

```
int foo(const int d) {
    int a = d;
    int b = 1 + a;
    return b;
}
```

pourrait être optimisé de la façon suivante :

```
int foo(const int d) {
    return 1 + d;
}
```

Cette "simple" optimisation permet de répondre aux différents critères d'amélioration. Le code est plus rapide, plus court donc il prend moins de place en mémoire, consomme moins d'énergie et enfin il utilise moins de ressources en RAM.

1.1.3 Outils existants

De très nombreux outils plus ou moins connus sont dédiés à l'optimisation de code, et ce dans tous les languages de programmation. La bibliothèque de compilation GCC [1], par exemple permet à l'utilisateur d'appliquer facilement nombre d'optimisations à son code. L'optimisation présentée à la section précédente est par exemple appliquée par GCC.

Les outils procèdent généralement à une analyse statique du code afin de l'optimiser ou de signaler des erreurs aux programmeurs. Cela signifie que le programme analysé n'est pas exécuté mais que les outils procèdent à une analyse syntaxique du programme afin d'étudier sa structure. L'analyse dynamique, avec exécution du programme, est d'avantage utilisée pour faire des tests afin de vérifier que le programme répond à des spécifications.

1.2 Notre approche

1.2.1 Les Control Flow Graphs (CFG)

Nous nous sommes d'abord concentrés sur les Control Flow Graphs qui sont un outil essentiel en optimisation de compilation et permettent de visualiser le code en suivant son flux par la modélisation d'un graphe. Il permet donc d'observer tous les chemins possibles d'un programme en séparant l'arbre en plusieurs branches lorsqu'il y a notemment des conditions.

1.2.2 Détection de code mort

Après avoir obtenu quelques résultats sur les CFG, nous avons décidé de nous tourner vers le problème de la détection de code mort. En effet, une branche importante de l'optimisation est de détecter le code qui n'est jamais exécuté dans un programme afin de le supprimer, ou de détecter des erreurs.

1.2.3 Notre projet

Le but de notre projet était à l'origine d'étudier la structure des codes sources de différents programmes afin de détecter des similitudes entre ces structures et les CFG. Après avoir procédé au développement de deux outils, nous nous sommes finalement tourné vers le problème de détection de code mort sur lequel il y a encore beaucoup de recherches.

1. GCC (GNU Compiler Collection) est une suite de logiciels libres de compilation. On l'utilise dans le monde Linux dès que l'on veut transcrire du code source en langage machine, c'est le plus répandu des compilateurs. - `https://doc.ubuntu-fr.org/gcc`

Chapitre 2

Control Flow Graphs

2.1 Introduction

2.2 Nos recherches

2.3 Implémentation

2.3.1 Notre script de modifications des CFG

2.3.2 Notre analyse des CFG

Chapitre 3

Dead Code Detection

3.1 Introduction

3.2 Nos recherches

3.3 Implémentation

Bibliographie

[1] Wikipedia. Optimisation de code — Wikipedia, the free encyclopedia. http://fr.wikipedia.org/w/index.php?title=Optimisation\%20de\%20code&oldid=153609138, 2019. [Online ; accessed 25-May-2019].