MIF20 Projet De Recherche
Analyses de pointeurs dans LLVM

Aurélien Chemier

Table des matières

1 Contexte du projet de recherche

Le projet de recherche a été proposé par Laure Gonnord qui travaille au Laboratoire de l'Informatique du Parallélisme.

Le calcul scientifique, qui sert notamment à effectuer diverses simulations (météorologiques, physiques), ainsi qu'à analyser de grandes quantités de données, occupe une part importante de la recherche informatique actuellement. Il est donc crucial d'améliorer le temps d'exécution des calculs, ainsi que de diminuer les ressources nécessaires pour ce calcul. Un certain nombre d'optimisations peuvent être réalisées statiquement (à la compilation), notamment en utilisant un cadre formel appelé *modèle polyédrique* (voir la section 2.1).

Nous reprenons pour cela un travail existant, celui réalisé par C. Bacara pendant son stage de L3 à Lille [1] à l'aide du framework Clang/LLVM (voir la section 2.2) qui est un compilateur C à C [2].

L'objectif sera donc de reprendre l'implémentation de C. Bacara, et après d'éventuelles améliorations, de développer une analyse de pointeurs précise mais à faible coût, qui sera utilisée comme phase préliminaire à d'autres analyses et optimisations , à l'aide des outils disponibles dans le framework. À cette occasion de nombreux jeux de tests seront réalisés.

Le projet de recherche se déroulera au Laboratoire de l'Informatique du Parallélisme sous la direction de Laure Gonnord.

2 Aspect technique

2.1 Modèle polyédrique

Le modèle polyédrique est un modèle mathématique de représentation de boucles imbriquées. Dans ce modèle, les bornes de boucles *for* sont des fonctions affines des indices des boucles englobantes, et les fonctions d'accès aux tableaux sont aussi des fonctions affines des indices de boucles. Sous cette restriction, on est en mesure de représenter les espaces d'itérations (nids de boucles) par des polyèdres, ainsi que les dépendances de données entre différentes *instructions* du programme. Certaines optimisations (parallélisation automatique, amélioration de placement mémoire) sont alors possibles.

Les analyses polyédriques sont restreintes syntaxiquement et ne sont pas applicables à un programme C quelconque facilement. elles ont pour objectif d'optimiser le traitement des tableaux .

1. `http://laure.gonnord.org/pro/papers/rapportBacara.pdf`
2. Un compilateur C à C prend un code C en entrée et délivre en sortie un autre code C. LLVM peut également générer de l'assembleur.

2.2 LLVM

Le projet LLVM [3] est né en 2000 à l'Université de l'Illinois, sous la direction de Chris Lattner et Vikram Adve. Il s'agit d'une infrastructure de compilateur, fondée sur une représentation intermédiaire du code, de type SSA [4] , conçue pour l'optimisation d'un programme à tous les niveaux (compilation, édition de liens, exécution). LLVM est composé de librairies qui permettent de raisonner/Manipuler le code A l'origine, l'implémentation concernait les langages C et C++, mais il existe désormais une grande variétés de FRONT-END [5] : Ruby, Python,Java, PHP, et Fortran, parmi d'autres.

LLVM contient nombre d'optimisations, ainsi que des générateurs de code pour de nombreux processeurs. LLVM est diffusé sous licence University Of Illinois Open Source, licence de type BSD. LLVM est codé en C++.

2.3 Clang

Clang (codé en C++) et est le FRONT-END de LLVM. C'est celui qui est chargé de faire les analyses lexicales et syntaxiques, de construire l'arbre de syntaxe abstrait du programme, et de le convertir en code intermédiaire, compréhensible par LLVM.

3 Besoins du projet

Ce projet de recherche arrive à la suite du Stage de L3 de C. Bacara qui a commencé une implémentation à l'aide du framework Clang/LLVM. Son rapport va servir de base au projet.

L'analyse actuelle utilise le FRONT-END clang et des analyses ad-hoc pour calculer des informations sur les pointeurs du programme. Le résultat actuel est un stockage dans une structure de données d'un certain nombre d'informations concernant les pointeurs du programme (pointeurs constants, pointeurs cachant des tableaux [6]).

À la fin de la recherche, l'objectif est de regénérer un C débarasser des pointeurs inutiles.

4 Calendrier prévisionnel

Le stage se découpe en trois parties de 2 semaines :

3. `http://llvm.org`
4. *Single-Static Assignment*, représentation intermédiaire de code source dans laquelle chaque variable n'est assignée qu'une fois.
5. Passes de compilation qui effectuent l'analyse lexicale et syntaxique de la source en entrée afin de la transformer en représentation intermédiaire, nécessaire à la suite du processus de compilation (optimisation, édition de liens, ...)
6. Pointeur indiquant l'adresse de la première case d'un tableau.

1. **Prise en main** : lire le rapport de C. Bacara, installer llvm , faire des exemples classiques,installer le code existant et comprendre ce qui est fait. Enfin, trouver une caractérisation de l'ensemble des programmes pouvant être optimisés *avec le modèle polyédrique*(indépendances des instructions dans les boucles [7] ...).
2. **Algorithmique** : c'est une étude en profondeur : quelles infos sont calculées par le code et comment. Nous étudirons le cas où les pointeurs sont des tableaux déguisés (statiques).
3. **Code** : l'implémentation du code et des tests (tests minimals ,tests de LLVM [8] (tests fonctionnels) et des tests spécifiques aux optimisations (polyBench)).

7. Deux instructions sont indépendantes quand les variables d'une instruction ne sont ni modifiées, ni utilisées par l'autre instruction.

8. `http://llvm.org/docs/TestingGuide.html`