TP 1 : À quoi ça sert un compilateur?

bogdan.pasca@ens-lyon.fr

14 Sptembre 2010

Introduction

Q 0.1 Répondez à la question du titre.

Q 0.2 Qu'attendez-vous d'un compilateur ?

1 Optimisation de code C

Le but de cette partie est de découvrir les différents types d'optimisations et ce qu'on peut attendre d'un compilateur.

1.1 Propagation de constantes

```
int main(int argc, char** argv)
{
        int i = 0, j = 0;
        j += 1;
        return j;
}
```

Q 1.1 Compiler le code ci-dessus en assembleur (`gcc -Wall -s`)sans optimisation (`-O0`). Quelles optimisations pourraient être effectuées ? (valeurs constantes, utilisation des registres, variables non-utilisées)

Q 1.2 Compiler avec les optimisations (`-O1`), vérifier le résultat de l'assembleur.

Q 1.3 Quelle différence trouve-t-on entre `-O1` et `-O2` ? `-O2` et `-O3` ?

Q 1.4 Essayer avec des expressions plus complexes.

1.2 Optimisations de boucles

```
int main(int argc, char** argv)
{
        int i = 0, j = 0;
        for ( ; i < 100; i++)
                j += 1;
        return j;
}
```

Q 1.5 Quelles optimisations pourraient être effectuées ? (valeurs calculées statiquement, etc.)

Q 1.6 Compiler avec les optimisations (`-O1`), vérifier le résultat de l'assembleur.

Q 1.7 Observer les différences entre les différents niveaux d'optimisations.

Q 1.8 Complexifier l'expression dans la boucle (`j = i*j + 42;` par exemple), que se passe-t-il ?

```
int fortytwo(void)
{
        return 42;
}
```

1.3 Analyse inter-procédurale

Il s'agit d'une phase où l'on analyse l'influence des fonctions entre elles, si l'on doit ou pas "inliner" une fonction, si une fonction est "pure" (dans ce cas on a le droit de construire une table pour cacher le résultat), etc. Dans gcc peu d'optimisations sont implementées.

```
int fortytwo(void)
{
        return 42;
}
```

Q 1.9 Introduire une fonction "pure" comme ci-dessus (il faut l'appeler et utiliser le résultat dans `main`). Observer les effets des différents niveaux d'optimisation.

Q 1.10 Jouer avec le mot clé `const`, que se passe-t-il ?

Q 1.11 Tester avec des fonctions pures plus complexes, des fonctions non-pures.

1.4 Optimisation à la génération du code

Dans la génération de code, on distingue plusieurs phases, l'allocation de registres, la sélection d'instructions, l'ordonnancement d'instructions...

Q 1.12 Comment place-t'on la valeur `0` dans un registre ? En O1 ? en O2 ?

Q 1.13 Quel code est généré par le compilateur pour une expression de la forme `x = c1*y + c2` ? (on peut introduire des variables que le compilateurs ne connaît pas en déclarant un `extern int y;` dans le scope global)

2 Exemple sur un langage fonctionnel : OCaml

2.1 Familiarisation

Dans la distribution d'OCaml, deux compilateurs sont proposés : `ocamlc` et `ocamlopt`.

Q 2.1 Quelle est la différence ?

Q 2.2 Quels sont les avantages et les inconvénients de chacun d'eux ?

Q 2.3 Faites un exemple basique et compilez avec `ocamlc -dinstr`. Que pouvez vous dire du code produit (par rapport à la partie 1) ?

Q 2.4 Compilez maintenant votre exemple avec `ocamlopt -S`. Même question.

Q 2.5 En OCaml, il n'y a pas de fonction principale `main`. Comment cela est-il géré ?

À présent, on s'intéresse uniquement au compilateur `ocamlopt`.

2.2 Optimisations standard

Q 2.6 OCaml gère-t'il l'*inlining* ? Confirmez ou infirmez par l'invention d'exemples judicieux.

Q 2.7 Idem pour la propagation de constantes.

Q 2.8 Commentez le code produit pour l'exemple suivant :

```
let a = 42 ;;
let b = if true then a else a ;;
print_int b ;;
```

2.3 Récursivité

Q 2.9 Écrivez deux versions, une récursive terminale et l'autre non, d'une fonction qui réalise la somme des éléments d'une liste.

Q 2.10 Comparez le code assembleur produit pour ces fonctions. Correspond-il à vos attentes ? Quels sont alors les avantages pour une fonction d'être récursive terminale ?

2.4 Sécurité

```
let t = [| 1; 2; 3; 4 |] ;;
let i = Random.int 4 ;;
print_int t.(i);;
```

Q 2.11 Observez le code précédent. Comparez avec le même code compilé avec l'option `-unsafe`.

Q 2.12 Méditez sur la citation suivante. Quels sont les différences entre C et OCaml sur ce point de vue ?

> Life is too short to spend time chasing down irreproducible bugs, and money is too valuable to waste on the purchase of flaky software. When a program has a bug, it should detect that fact as soon as possible and announce that fact (or take corrective action) before the bug causes any harm. A. Appel

3 Culture Gé

Rule of full employment of compiler researchers/engineers... : "It is impossible to build a fully optimizing compiler."

Q 3.1 Traduisez cette citation en français à votre voisin qui ne parle pas anglais si vous en avez un.

Q 3.2 Qu'en pensez-vous ?