Rapport projet COCass

Benjamin Loison

15 janvier 2021

Contents

1 Les parties réalisées

Concrètement ma version de mcc compile tous les exemples données ex0 à ex12, ordre, fact et cat mais pas sieve.
Mon compilateur arrive à bien compiler aussi les tests de base des années précédentes (testsuite_we) mais pas unitest0_vf.
De plus j'ai essayé de faire fonctionner unitest0_vf ou d'autres programmes plus ou moins élémentaires à la fin et j'ai eu des mauvaises surprises sur la compilation et fonctionnement de certains programmes.

2 L'approche suivie

Au début j'ai utilisé godbolt pour obtenir l'assembleur des fichiers sources .c associés. Cependant j'ai rapidement été déçu de ce site et j'ai donc préféré utiliser *gcc -S source.c -o source.s*, tout en faisant attention à certains détails comme utiliser des *long* au lieu des *int* afin de ne pas jongler entre des instructions assembleurs de 4 et 8 octets.
J'ai donc écrit petit à petit les cas du "match" d'OCaml en faisant le parallèle entre les sources .c des exercices données par Madame Ledein et le .s associé par gcc.
Ainsi j'ai réussi à faire fonctionner jusqu'à cat assez facilement mais après la taille des programmes assembleurs et de l'AST étaient assez importante et complexes.
J'ai alors perdu du temps à corriger mon mcc de cette manière même si j'ai finalement réussi.
C'est alors que j'ai imité certains camarades qui ont récupéré des tests unitaires hérités des années précédentes et qui ont permi de mettre en exergue facilement des petits problèmes.

3 Les choix effectués

J'ai donc opté pour les instructions 64 bits au lieu des 32, après avoir travaillé au début en 32 bits, le 64 bits semblait bien plus uniforme à utiliser en général. Ce qui a été difficile au début n'était pas le langage assembleur en lui-même dont on avait une bonne connaissance avec les cours de programmation du premier semestre mais plus les headers comme .data .text .align 8 que j'ai dû découvrir et réussir à faire fonctionner moi-même.

4 Les difficultés

J'ai aussi perdu du temps en cherchant des erreurs d'inattention comme des '*' mis au lieu de '-'...
En plus de l'utilisation des headers que j'ai dû deviner, il y a certains points qui n'avait pas été soulevés lors du cours de programmation 1, comme le fait que le code suivant:

```
cmpq $3, %rax
addq $1, %rax
jl .L0
```

ne va pas conserver le drapeau de comparaison alors que le code suivant va bien le conserver jusqu'au saut conditionnel:

```
cmpq $3, %rax
jl .L0
```

De cette manière j'ai eu des problèmes sur la post incrémentation sur un code C– comme:

```
if(i++ == 2)
```

Il a fallu aussi garder son calme lorsque lors de l'éxécution de ma version de mcc OCaml m'affichait une erreur fatale par rapport à String.sub alors qu'il y a un nombre important de cette instruction dans mon code et le simple fait de savoir quel String.sub est problématique prend un certain temps alors que si OCaml ou un outil comme GDB pour OCaml nous donnait un backtrace cela m'aurait économiser beaucoup de temps.
Finalement après avoir eu des retours de mes camarades sur mon compilateur, je pense que le caractère hybride de certains fonctions de type void ou string et qui peuvent aussi écrire dans le fichier final n'est pas assez propre et induit les problèmes sur sieve et unitest0_vf, une approche avec plus de récursion au lieu de variables globales mutables aurait changer la donne, je pense.

5 Les tests et résultats

Je n'ai pas personnellement écrit beaucoup de tests mais il m'ai beaucoup arrivé de réduire un .c d'un exercice que je trouvais comme étant une "grande marche". Mais parfois cela n'était pas suffisant avec des structures comme dans le cat:

```
while ((c = fgetc (f))!=EOF)
```

A ce moment là j'ai réécrit l'AST de ce programme dans mon code OCaml afin de le simplifier afin de résoudre des problèmes intermédiraires mais ce fut assez long. Un convertisseur string (comme l'affichage AST) vers le type de l'AST m'aurai alors bien aidé.
Cependant j'ai écrit test.sh que j'éxécutais régulièrement pour vérifier que chacun des tests que l'on nous avait donné fonctionnait toujours bien après modification du code source du compilateur. Heureusement que je l'ai fait parce que sinon je pense que j'aurai remarqué après coup longtemps après des problèmes générés par de vieilles modifications.
De plus à la fin j'ai bien fait fonctionner mon compilateur sur l'un des deux tests proposés par les années passés.

6 Retenons le positif

Je ne sais pas si l'évaluation se base sur d'autres programmes complexes que ceux précédemments cités, si c'était le cas mon compilateur pourrait avoir des difficultés mais j'aimerais mettre en avant le fait que sur la grande majorité des tests donnés, mon compilateur compile et donne un éxécutable correct.
Je suis assez fier de ma maîtrise actuelle de l'assembleur et du "grand match OCaml" qui arrange la sémantique en lignes d'assembleurs que j'ai essayé de réduire au maximum.
Je suis aussi content d'avoir pu approfondir ma connaissance de GDB avec l'interface assembleur qu'il permet et je comprends maintenant mieux en pratique les optimisations qui sont permises avec le paramètre -O de gcc et les problématiques qui peuvent y être liés.
Contrairement à beaucoup de camarades j'ai essayé d'optimiser le nombre de lignes d'assembleur résultant en n'utilisant pas de variables temporaires lorsqu'il n'y en avait pas besoin. Ce point me permettait une relecture plutôt aisé du code assembleur.
De plus j'ai peu communiquer avec mes camarades sur ce projet parce que les compilateurs m'ont longtemps intrigué et je voulais passer le pas moi-même. Cependant ce manque de communication m'a sûrement fait perdre du temps sur certains points où mes camarades auraient pû m'expliquer brièvement et me donner des astuces comme l'utilisation de l'instruction *set*. Mais de cette manière j'ai entièrement compris, contrairement à certains camarades, le fonctionnement de chaque instruction généré par mon compilateur.