Compilateur Deca : Documentation de l'extension : Optimisation

Équipe 58

January 24, 2017

1 Introduction

Pour notre compilateur Decac nous avons dû choisir une extension parmis plusieurs extension, et notre choix s'est porté sur l'optimisation. L'optimisation consiste à rendre un code plus performant, c'est à dire à l'exécuter plus rapidement en faisant moins d'opérations au niveau du processeur, ou des opérations qui sont plus rapides.

2 Spécification de l'extension Optimisation

2.1 Optimisations implémentées

Voici la liste des différentes spécifications de l'optimisation:

- `Dead Store`
  `decac -o1`
  Le dead store consiste à éliminer les variables qui ne sont pas utiles dans le code.

- `Constant Folding`
  `decac -o2`
  Le constant folding est une optimisation qui calcul tout les calculs constants et stocke les résultats au lieu de stocker le calcul. A l'exécution le processeur n'a plus besoin de faire les opérations, il à déjà le résultat en mémoire.

2.2 Optimisations envisagées

- `Inlining`
  Le inlining est une optimisation qui consiste à remplacer les appels des méthodes par le corps des méthodes, dans le cas ou la méthode ne prend pas trop de place en mémoire. Ceci permet de gagner du temps car la méthode n'est pas appelée lors de l'exécution, mais cette optimisation ne peut pas être utilisé sur les méthodes qui prendraient trop de place en mémoire.

- `Strengh Reduction`
  Le strengh reduction est une optimisation qui remplace certaines opérations par d'autres qui sont équivalentes. Une multiplication peut être remplacé par une succession d'addition, une multiplication par une puissance de 2 peut être remplacé par un décalage à gauche. Il y a de multiples façons de faire des strengh reductions qui améliore localement l'optimalité du code.

- `Common Subexpression`
  Le common subexpression est une optimisation qui consiste à ne calculer qu'une seule fois les sous expressions qui apparaissent à plusieurs endroits dans le code, de manière à les stocker pour ne plus les recalculer ensuite. Lorsque le calcul réapparaît dans le programme le processeur a juste besoin de charger la valeur stockée et n'a pas besoin de refaire le calcul.

- `Optimiastion des registres au niveau du processeur`
  Cette optimisation consiste à regarder dans le code généré en assembleur s'il y a des améliorations possibles. Dans certains cas le processeur fait un calcul sur une variable puis la stocke, mais la recharge juste après pour la réutiliser. On gagne en rapidité si on supprime tous les stockages et les chargements inutiles de variables dans le processeur.

3 Analyse bibliographique

4 Choix de conception

5 Méthode de validation

6 Résultat

7 Améliorations possibles de l'extension