# Compilateur Deca : Documentation de l'extension : Optimisation

Équipe 58

January 25, 2017

## 1 Introduction

Pour notre compilateur Decac nous avons dû choisir une extension parmi plusieurs extensions possibles, et notre choix s'est porté sur l'optimisation. L'optimisation consiste à rendre un code plus performant, de donner de meilleurs résultats, ou y arriver plus vite ou avec le moins de ressources possibles, 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.Plusieurs types d'optimisation de code sont possibles et ont été déjà implémentées, il s'agissait donc pour nous de reprendre ces méthodes classiques et en utiliser dans notre propre compilateur.

## 2 Spécification de l'extension Optimisation

### 2.1 Optimisations implémentées

Voici la liste des différentes spécifications de l'optimisation:

- `DeadStore`
  `decac -o1`
  Le dead store consiste à éliminer les variables qui ne sont pas utiles dans le code. typiquement , si on initialise des valeurs, mais que lors d'autres initialisations ou lors de la liste d'instructions, notre variable n'est pas utilisée, alors elle est considérée comme inutile pour le programme, est peut être supprimée.
  Exemple de programme utilisant le Deadstore :

  int a=2;
  int b=4;
  boolean f=false;
  b=b*b+a*a;
  a=(a*b)/2;

  Dans ce programme là, on observe que le booléen f n'est pas utilisée dans le programme. Elle n'est pas utilisée pour une initialisation, ni pour les instructions. Elle sera ainsi supprimée de l'arbre, au niveau des déclarations de variables. Mais on pourrait fortement discuter de la notion de "programme inutile. En effet, dans le programme deca

précédent, rien n'est retourner, donc les variables a,b et f sont en soit "inutiles" pour d'autres programmes , et pourraient en théorie être éliminié. En fait on peut éliminer tout le programme ici car il est bien inutile. Mais comment définir l'utilité d'un programme ? Par les valeurs retorunées ? les paramètres ? Pour des questions de temps et de simplicité

- `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

### 3.1 Liens sur l'optimisation de la compilation en Java

- `http://www.javaworld.com/article/2078623/core-java/jvm-performance-optimization-part-`
  Ce lien donne des informations générales sur la compilation en Java en don-

nant quelques concepts généraux sur l'optimisation du Java et du Bytecode Java.

- http://www.javaworld.com/article/2078635/enterprise-middleware/jvm-performance-optimi
  Ce lien explixicite plus clairement les différentes optimisations, les avantages et les inconvénients de la compilation statique et ceux de la compilation dynamique.

- http://www.javaworld.com/article/2076060/build-ci-sdlc/compiler-optimizations.html
  Ce lien donne des techniques d'optimisation et des exemples de celles-ci (le constant folding et le dead code elimination).

- http://web.stanford.edu/class/archive/cs/cs108/cs108.1082/handouts/39ImplementationHo
  Ce lien donne l'implémentation de la machine Java Hotspots et les optimisations de cette machine Java, comme le inlining.

- http://www.oracle.com/technetwork/java/whitepaper-135217.html
  Ce lien présente les performances de la machine Java Hotspots et son architecture.

- https://www.usenix.org/legacy/publications/library/proceedings/jvm02/yu/yu_html/node3.
  Ce lien envoie donne de nombreuses références de livres sur la compilation etl'optimisation en Java.

## 3.2 Liens sur les compilateurs Just In Time

- http://docs.oracle.com/cd/E15289_01/doc.40/e15058/underst_jit.htm
  Ce lien présente avec des schémas le fonctionnement de la compilation JIT et présente des techniques d'optimisations étapes par étapes.

- http://www.sdsc.edu/ allans/cs231/openjit.pdf
  Ce lien présente le design, l'implémentation d'un compilateur JIT et évalue les différentes optimisations.

## 3.3 Liens sur la compilation statique

- http://blog.soat.fr/2015/10/java-entre-compilation-statique-et-dynamique-mon-coeur-ba
  Ce lien présente les différents types de compilateurs et explique les différents types de compilations et d'optimisations avec des schémas. Il présente aussi quelques techniques d'optimisations un peu moins évidente.

- https://briangordon.github.io/2014/01/javac-optimizations.html
  Ce lien présente différentes techniques d'optimisations que les machines

Java font en donnant des exemples simples et en les expliquant.

- `https://en.wikipedia.org/wiki/List_of_optimization_software`
  Donne une liste de software qui optimise les programmes dans de nombreux langages.

## 3.4 Liens sur les différentes techniques d'optimisations

### 3.4.1 Techniques générales

- `https://en.wikipedia.org/wiki/Optimizing_compiler`
  Ce lien donne une liste assez complète des techniques d'optimisations de programme.

### 3.4.2 Techniques d'optimisations de boucles

- `https://en.wikipedia.org/wiki/Loop_fission`
  Ce lien présente la technique du Loop fission qui consiste a diviser une boucle en deux, de manière à avoir une meilleur gestion des adresses accédées.

- `https://en.wikipedia.org/wiki/Loop_fusion`
  Ce lien présente la technique du Loop fusion qui consiste à fusionner 2 boucles entres elles si elles parcourent le même index et qu'elles sont indépendantes l'une de l'autre. Cette technique est l'inverse de la technique précédente et s'applique donc dans des cas différents.

- `https://en.wikipedia.org/wiki/Loop_interchange`
  Ce lien présente la technique du Loop interchange qui consiste à permutter les indices dans une double boucle, ce qui permet dans le cas d'un tableau double dimensions d'accèder aux cases du tableaux contigües dans la mémoire.

- `https://en.wikipedia.org/wiki/Loop_inversion`
  Ce lien présente la technique du Loop inversion qui consiste à remplacer un while par un if + do while ce qui fait faire deux sauts de moins.

- `https://en.wikipedia.org/wiki/Loop-invariant_code_motion`
  Ce lien présente la technique du Loop invariant code motion qui consiste à sortir les opérations constantes d'une boucle en dehors de la boucle.

- `https://en.wikipedia.org/wiki/Loop_unrolling`
  Ce lien présente la technique du Unrolling qui consiste à allonger le corps d'une boucle de manière à faire plus d'opérations en un tour et faire moins

de tours en tout quand cela est possible.

- `https://en.wikipedia.org/wiki/Loop_splitting`
  Ce lien présente la technique du Loop splitting qui consiste à simplidfier ou élminer les dépendances d'une boucle en la divisant en plusieurs boucle plus petite qui ont des indexage contigus.

- `https://en.wikipedia.org/wiki/Polytope_model`
  Ce lien présente la technique du Polytope model qui est une technique assez complexe qui fait des transformations complexes dans des boucles multiples de manières à les parcourir de façon optimisée.

- `https://en.wikipedia.org/wiki/Loop_tiling`
  Ce lien présente la technique du Loop tiling qui consiste à changer les indices de la boucle et à la diviser de manière à forcer les variables à rester dans le cache pour y avoir plus rapidement accès, technique qui est difficile à exploiter correctement à notre niveau.

- `https://en.wikipedia.org/wiki/Loop_unswitching`
  Ce lien présente la technique du Loop unswitching qui consiste à mettre une condition intérieur de la boucle à l'extérieur puis à dupliquer la boucle, ceci facilite la parallèlisation de la boucle.

- `https://en.wikipedia.org/wiki/Automatic_parallelization`
  Ce lien présente la technique de l'Automatic parallelization qui consiste à parcours des boucles en parallèle sur des multiprocesseurs, technique qui est difficilement implémentable à notre niveau.

- `https://en.wikipedia.org/wiki/Loop_scheduling`
  Ce lien présente la technique du Loop scheduling qui consiste à parcourir une boucle sur plusieurs processeur, ce qui est difficilement implémentable à notre niveau.

- `https://en.wikipedia.org/wiki/Software_pipelining`
  Ce lien présente la technique du Software pipelining qui est une technique assez complexe qui parallèlise les boucles. Cette technique est complexe à implémenter.

- `https://en.wikipedia.org/wiki/Automatic_vectorization`
  Ce lien présente la technique du Loop scheduling qui consiste à faire le plus d'itérations possible sur le plus de processeurs possibles, technique qui est difficielement implémentable à notre niveau.

#### 3.4.3 Techniques d'optimisations de boucles

- 
- 
- 

# 4 Choix de conception

## 4.1 Constant Folding

# 5 Méthode de validation

## 5.1 Constant Folding

# 6 Résultat

## 6.1 Constant Folding

# 7 Améliorations possibles de l'extension