UFR 919 Informatique – Master Informatique

Spécialité STL – UE ILP

TME9 — Caches de méthodes

Christian Queinnec

1 Présentation

Ce TME porte sur ILP6. Il s'agit d'améliorer la compilation de l'envoi de méthodes. Le coût de l'appel de méthode est principalement dû à la recherche de la méthode à invoquer. Une fois trouvée, invoquer cette méthode a le même coût qu'un appel de fonction. Aujourd'hui, dans le compilateur, cette recherche est effectuée par la fonction (en C) `ILP_find_method`. La recherche est décomposée en :

1. vérifier que le receveur est une instance (directe ou indirecte) de la classe ayant introduit la méthode ;
2. vérifier l'arité de l'invocation ;
3. déterminer où est rangée la fonction à invoquer ;
4. invoquer cette fonction.

D'antiques mesures ont montré qu'il est utile de mémoriser les méthodes trouvées car elles ont de grandes chances d'être cherchées à nouveau dans les instants qui suivent. Cette localité temporelle peut être mise à profit à l'aide de cache(s) mémorisant le(s) dernier(s) résultat(s).

Le programme qui suit (ne fait rien d'intéressant) mais sert d'exemple aux questions qui suivent.

```
1  class Point extends Object {
2     field x
3     field y
4     method longueur () {
5        return this.x + this.y
6     }
7     method distance (p) {
8        return this.longueur()           // site
9             + p.longueur()              // site
10 }
11 class PointColore extends Point
12    field color
13    method longueur () {
14       return 2 * super()               // site(?)
15    }
16 }
17
18 let c = 1 in
19 let p0 = new PointColore(0, 0, "red") in
20 while ( true ) {
21    let p1 = new Point(c, ++c) in
```

```
22    p0.print()                          // site
23    p1.distance(p0)                     // site
24    p0 = p1
25 }
```

Objectifs : Implanter les différents types de cache proposés ci-dessous et réaliser une évaluation de leur efficacité.

Buts :

— Comprendre l'envoi de méthodes
— Comprendre les différents types de cache
— Apprendre à réaliser un benchmark simple

2 Travail à réaliser

Nous vous demandons d'implanter le cache global (section 2.2) et les suivants (sections 2.3 et 2.4) et, pour chacun, de mesurer le gain qu'il apporte. Pour cela vous créerez une extension à ILP6 redéfinissant l'envoi de méthodes afin d'y incorporer la gestion du cache et la mise à jour de deux compteurs : le premier compte le nombre d'invocations de méthodes, le second compte le nombre de fois que la méthode a été trouvée dans le cache. Ces deux compteurs seront imprimés (dans la sortie d'erreur pour éviter d'impacter le code java existant) après l'évaluation du programme (modifier le patron C pour cela) sous la forme suivante : N_1/N_2 où N_1 est le nombre de fois où le cache est utilisé et N_2 le nombre total d'appels de méthodes.

Créer un sous-paquetage `fr.upmc.ilp.ilp6tme9` qui contiendra tous les fichiers que vous créerez dans ce tme. Comme nous nous intéressons qu'à la partie compilation, supprimer les tests concernant l'interprétation dans vos classes `ProcessTest` étendus.

2.1 Banc d'essai

Écrire un programme effectuant au moins cent mille envois de messages. Ce programme servira d'étalon pour apprécier l'efficacité des questions suivantes. Il est recommandé de lire la suite afin de ne pas écrire un programme dégénéré qui masquerait toutes les améliorations que proposent les questions suivantes.

Quelles sont les qualités à mettre en évidence dans ce programme ? Quelles configurations de classes, de méthodes et d'instances illustrerez-vous ? Ω

Aide : Vous pourrez utiliser le programme de test `u8999-6.xml`.

2.2 Cache global

On décide de modifier la fonction `ILP_find_method` ainsi. La première fois qu'elle est appelée avec des arguments (r, m, n) (c'est-à-dire receveur, méthode, arité), on calcule la méthode à invoquer f, on vérifie que les conditions d'invocation sont réunies, on mémorise le quadruplet (r, m, n, f) puis on invoque f. La seule modification est la mémorisation du quadruplet (on peut d'ailleurs plutôt mémoriser la classe de r plutôt que r).

La seconde fois, lorsque la fonction `ILP_find_method` est invoquée avec des arguments (r', m', n') alors la méthode à invoquer est f si r et r' ont même classe, si $m = m'$ et si $n = n'$. Lorsque la méthode à invoquer change, le cache est vidé et re-rempli avec les nouvelles données.

Le C engendré pour un programme ILP (par exemple celui donné plus haut) contient donc un unique cache global.

Implanter ce nouvel algorithme à l'aide de variables globales et mesurer son efficacité à l'aide du banc d'essai précédent. Il faudra étendre la classe `fr.upmc.ilp.ilp6.ast.CEASTsend` en une classe `fr.upmc.ilp.ilp6tme9.CEASTsend`. Les fichiers C à créer seront appelés `ilpObj-cg.c` et `ilpObj-cg.h`.

2.3 Cache unique par méthode

L'ennui du schéma précédent est que si le banc d'essai évalue de multiples fois la séquence `o1.M1(); o2.M2()` alors rien n'est gagné car chaque envoi de méthode réinitialise le cache global (les objets `o1` et `o2`, les méthodes `M1` et `M2` sont supposés différents).

Des mesures montrent qu'avoir un cache par méthode est intéressant. On modifiera donc la définition de la structure `ILP_Method` pour contenir en plus un cache qui ne sera utilisé que pour les invocations de cette méthode. Ainsi dans le cas précédent, à chaque fois que l'on ré-exécutera `o1.M1()` ou `o2.M2()`, on retrouvera, dans le cache approprié, la méthode précédemment déterminée.

Le C engendré pour un programme ILP contient donc un cache par méthode. Pour le programme exemple donné plus haut, il y a trois caches car il y a trois méthodes (`longueur`, `distance` et `print`).

Implanter ce nouvel algorithme et mesurer son efficacité à l'aide du banc d'essai précédent. Les fichiers C à créer seront appelés `ilpObj-cm.c` et `ilpObj-cm.h`.

2.4 Cache local aux sites d'invocation

En fait, il y a encore mieux ! Associer un cache à chaque emplacement où est réalisé un envoi de méthode est bien meilleur. Ainsi, un envoi de méthode `o.M()` sera-t-il associé à un cache aucunement partagé.

Le C engendré pour un programme ILP contient donc un cache par envoi de méthode. Pour le programme exemple donné plus haut, il y a cinq sites d'envoi de message donc cinq caches.

Implanter ce nouvel algorithme et mesurer son efficacité à l'aide du banc d'essai précédent. Il faudra étendre la classe `fr.upmc.ilp.ilp6.ast.CEASTsend` en une classe `fr.upmc.ilp.ilp6tme9.CEASTsendLocal`. Les fichiers C à créer seront appelés `ilpObj-local.c` et `ilpObj-local.h`.