

Compilation - Projet n°2

Allocation des registres : scan linéaire

Vous pouvez trouver tous les sujets au format électronique sur internet à l'adresse :
<http://www.pps.jussieu.fr/~ayache>
dans la rubrique Enseignements.

Ce projet propose d'implanter un algorithme d'allocation des registres différent de celui à base de coloration de graphe, déjà présent dans CerCo.

Pensez à tester régulièrement votre code !

1 Présentation

Dans le compilateur CerCo, c'est lors du passage de ERTL à LTL que l'allocation des registres est effectuée. Celle-ci est basée tout d'abord sur une *analyse de vivacité* qui permet de construire un *graphe d'interférence* dont les sommets représentent chaque pseudo-registre d'une fonction ERTL. Une arête dans le graphe d'interférence entre deux pseudos-registres signifie que les pseudos-registres *interfèrent*, c'est-à-dire qu'ils peuvent être *vivants* en même temps dans la fonction, et qu'on ne peut donc pas leur associer le même emplacement physique. Un algorithme de *coloration* est alors déployé sur le graphe d'interférence afin de déterminer un registre physique ou un emplacement sur la pile pour chaque pseudo-registre, de sorte que des pseudos-registres qui interfèrent ne se voient pas associer le même emplacement physique.

Le but du projet est d'implanter un algorithme d'allocation des registres différent, appelé *scan linéaire* [1].

2 Description de l'algorithme de scan linéaire

L'algorithme est décrit dans un article laissé en référence. Nous résumons très brièvement son concept, et rappelons son pseudo-code (légèrement modifié).

Tout d'abord, l'algorithme suppose qu'un ordre a été établi entre les sommets du graphe de la fonction ERTL pour laquelle on souhaite faire l'allocation des registres (l'ordre utilisé dans l'article est une *recherche en profondeur*). Ensuite et à partir de l'analyse de vivacité, on construit un *intervalle de vivacité* pour chaque pseudo-registre de la fonction ERTL : les bornes en sont le plus petit et le plus grand sommet (d'après l'ordre établi) où le pseudo-registre est vivant. L'algorithme parcourt alors tous les intervalles ; seuls les pseudos-registres dont les intervalles associés se chevauchent sont susceptibles d'interférer. En parcourant les intervalles une seule fois selon l'ordre établi et en considérant la liste des intervalles *actifs* (c'est-à-dire qui peuvent chevaucher l'intervalle courant), il est possible de réaliser une allocation complète. L'algorithme est décrit en figure 1.

Pour le cas de CerCo, on considère dans cet algorithme qu'un *intervalle de vivacité* est en fait un couple composé d'un pseudo-registre (de type OCaml `Register.t`) et de son intervalle de vivacité. La table *location* est le résultat de l'algorithme, et associe à chaque pseudo-registre de la fonction ERTL un

```

LINEARSCANREGISTERALLOCATION
  active    fg
  for each live interval  $(r, i)$ , in order of increasing start point
    EXPIREOLDINTERVALS( $r, i$ )
    if  $\text{length}(\text{active}) = R$  then
      SPILLATINTERVAL( $r, i$ )
    else
       $\text{location}(r)$     a register removed from pool of free registers
      add  $(r, i)$  to active, sorted by increasing end point

EXPIREOLDINTERVALS( $r, i$ )
  for each interval  $(r', j)$  in active, in order of increasing end point
    if  $\text{endpoint}(j) < \text{startpoint}(i)$  then
      return
    else
      remove  $(r', j)$  from active
      add  $\text{location}(r')$  to pool of free registers

SPILLATINTERVAL( $r, i$ )
   $(\text{spill}, j)$     last interval in active
  if  $\text{endpoint}(j) > \text{endpoint}(i)$  then
     $\text{location}(r)$      $\text{location}(\text{spill})$ 
     $\text{location}(\text{spill})$     next stack offset
    remove  $(\text{spill}, j)$  from active
    add  $(r, i)$  to active, sorted by increasing end point
  else
     $\text{location}(r)$     next stack offset

```

FIGURE 1 – Pseudo-code de l'allocation des registres par scan linéaire

emplacement physique (de type OCaml ERTLToLTLI.decision). L'algorithme suppose également que l'on dispose d'un ensemble de *registres physiques libres* (initialisé par `Driver.TargetArch.allocatable` dans notre cas), de la taille R de l'ensemble des *registres physiques allouables* (c'est une constante égale à `Driver.TargetArch.RegisterSet.cardinal Driver.TargetArch.allocatable`), et d'un moyen de connaître le prochain offset libre dans la pile (chaque pseudo-registre en pile prendra `Driver.TargetArch.int_size` octets).

3 Tâches

Vous devez implanter l'algorithme de scan linéaire dans CerCo. Un squelette a été créé dans le module `Allocator_linear` du répertoire `src/ERTL/`. Dans l'état des sources que vous allez récupérer, c'est l'allocation des registres par scan linéaire qui est utilisé, mais qui ne fonctionne pas. Pour vos tests, si vous voulez que la compilation se fasse par la méthode de coloration — qui elle fonctionne —, il vous suffit de modifier le fichier `src/ERTL/choosen_allocator.ml` qui contient simplement la définition de l'algorithme d'allocation choisi pour la compilation.

Au final, le résultat doit être donné dans les fonctions `lookup` (association pseudo-registre / emplacement physique) et `locals` (taille totale des pseudos-registres mis en pile) du foncteur `Allocator_linear.Make`. Le foncteur prend en argument un module qui vous donne accès aux fonctions suivantes :

- **entry** : étiquette d’entrée de la fonction **ERTL** en cours d’allocation des registres ;
- **succs** : successeurs d’une étiquette dans la fonction **ERTL** en cours d’allocation des registres ;
- **pseudo_registers** : liste des pseudos-registres de la fonction **ERTL** en cours d’allocation des registres ;
- **valuation** : pseudos-registres vivants à une étiquette donnée dans la fonction **ERTL** en cours d’allocation des registres.

Vous êtes libres d’implanter les fonctions de l’algorithme comme vous le souhaitez, et même d’ajouter des éléments supplémentaires dans le module argument du foncteur **Allocator_linear.Make**, cependant vous devrez probablement définir les éléments suivants sur lesquelles les fonctions s’appuient :

- un ordre sur les étiquettes du graphe (de préférence suivant une recherche en profondeur) ;
- une notion d’intervalle ;
- l’intervalle de vivacité de chaque pseudo-registre.

4 Modalités

Récupération des sources. Vous pouvez récupérer les sources de **CerCo** complètes à l’aide de la commande suivante :

```
git clone ~ayachen/CerCo répertoire/de/votre/choix/
```

Il est conseillé d’utiliser un répertoire différent de celui que vous avez utilisé pour les séances précédentes afin de ne pas perdre votre travail.

À partir de ces sources, vous pouvez organiser votre projet comme vous le souhaitez : ajouter des fichiers, en retirer, modifier ceux déjà existants, etc.

Délivable.

- Le projet est à réaliser par groupe d’au plus 4 personnes.
- Chaque groupe doit envoyer une tarball compressée à l’adresse **nicolas.ayache@gmail.com** au plus tard le mercredi 18 janvier 2012.
- Le nom de cette archive doit être les noms des membres du groupe séparés par des tirets. *Exemple* : si un groupe est constitué de Vincent Dupont et de Pierre Vacheret, leur archive devra être nommée **dupont-vacheret.tgz**.
- L’archive doit contenir un répertoire du même nom. *Exemple* : dans le cas du groupe de Dupont et de Vacheret, leur archive **dupont-vacheret.tgz** doit contenir le répertoire **dupont-vacheret**.
- Ce répertoire doit à son tour contenir au moins : un répertoire nommé **CerCo** qui contient les sources du projet, et un rapport nommé **rapport.pdf** au format **pdf**. Au final, l’arborescence de la tarball doit ressembler à ce qui suit :

```
` dupont-vacheret/
  ` CerCo/
    ` src/
    ` tests/
    ` Makefile
    ` ...
  ` rapport.pdf
```

- Les sources doivent compiler, et ce par l’unique invocation de la commande **make** à leur racine.

Rapport. Vous êtes libre de rédiger le rapport comme vous le souhaitez. Cependant, il doit faire apparaître au moins les éléments suivants :

- une courte description de l’organisation de votre code ;
- le cas échéant, ce qui ne fonctionne pas correctement, avec une description de l’idée du travail à fournir ou du problème ;
- une comparaison de l’efficacité (taille de la pile) de votre implantation par rapport à la version d’origine par coloration ;
- les problèmes rencontrés s’il y en a eus, quelles que soient leurs natures : bugs importants (qui proviennent éventuellement des sources originales), difficultés d’implantation, d’organisation, de compréhension, etc.

Évaluation. La note attribuée à chaque groupe dépendra de plusieurs critères, par exemple : l’avancée du travail, la qualité du code (lisibilité), sa robustesse (peu d’erreurs à l’exécution), sa correction (peu d’erreurs de traduction), la qualité du rapport, le respect des consignes, le nombre d’étudiants dans le groupe (à qualité de projet égale, un groupe moins nombreux aura une meilleure note), l’autonomie.

Références

- [1] M. Poletto and V. Sarkar. Linear scan register allocation. *ACM Trans. Program. Lang. Syst.*, 21 :895–913, September 1999.