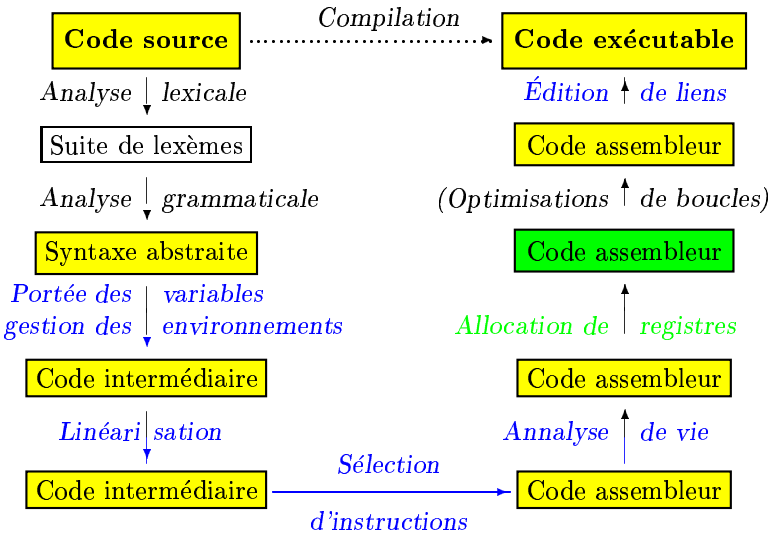
Allocation de registres.

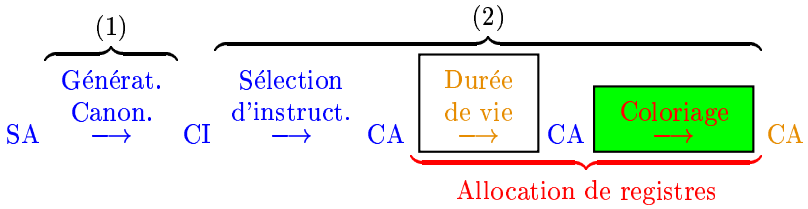
Didier Rémy
Octobre 2000

http://cristal.inria.fr/~remy/poly/compil/6/
http://w3.edu.polytechnique.fr/profs/informatique/Didier.Remy/compil/6/

Slide 1



Sa place dans la chaîne de compilation



Slide 2

On veut allouer les registres au mieux afin
- d'effectuer le minimum de sauvegardes en pile et
- d'éliminer le maximum d'instructions MOVE.

C'est un des maillons les plus connus de la chaîne de compilation.

Enjeux

Slide 3

L'allocation triviale semble avoir déjà des performances raisonnables. Pourquoi aller plus loin ?

1. La pression en registres, faible sur des petits exemples, peut devenir importante sur des exemples de taille réaliste, et en particulier si le compilateur expanse en ligne des appels à de petites procédures.
2. La pression en registres augmente aussi considérablement sur des architectures qui ont peu de registres.
3. Enfin, les performances d'une machine RISC sont très bonnes parce qu'il y a beaucoup de registres et la plupart des calculs intermédiaires peuvent être faits dans des registres, mais elles se dégradent rapidement si des sauvegardes superflues doivent être faites en pile.

Une allocation de très bonne qualité n'est donc pas du superflue.

Rappel de l'allocation triviale

Slide 4

Allocation

1. On choisit un temporaire t non colorié au hasard.
2. Les couleurs interdites sont l'ensemble des couleurs des temporaires déjà coloriés qui interfèrent avec t.
3. On choisit au hasard une couleur non interdite.
4. En cas d'échec, on décide de placer t en pile.

Élimination des MOVE

On choisit par préférence la couleur d'un temporaire déjà colorié en relation avec t (par une ou plusieurs instructions MOVE) ... tout en évitant la couleur préférée des temporaires qui interfèrent avec t qui ne sont pas encore coloriés.

Critiques de l'allocation triviale

Slide 5

Superposition des temporaires

Les temporaires sont traités dans un ordre arbitraire, sans tenir compte de la topologie.

Certains nœuds peu contraints sont servis en premier et reçoivent une couleur arbitraire, alors qu'il aurait été plus judicieux de les servir plus tard et leur attribuer une couleur restant disponible.

Élimination des MOVE

L'élimination des MOVE est traitée de façon ad-hoc, en aval, et sans collaborer avec l'allocation.

Un temporaire impliqué dans un MOVE devrait être privilégié pour lui donner plus de chance de choisir sa couleur égale à celle d'un temporaire avec lequel il est en relation.

Allocation fine

Principe

– Allocation des registres par un algorithme de coloriage de graphe,
– Prise en compte simultanée des instructions MOVE par une variation sur l'algorithme de coloriage.

Slide 6

Heuristiques

La méthode est générale, mais approchée.

Les choix sont guidés par des heuristiques *modulaires* plutôt que par des solutions *ad-hoc*.

Coloriage de graphe

Nous nous intéressons d'abord à l'algorithme de coloriage sans nous soucier des instructions MOVE.

Problème Étant donné un graphe et un ensemble de K couleurs, il s'agit d'attribuer une couleur à chaque nœud du graphe de telle façon qu'un arc relie toujours des nœuds de couleurs différentes.

Slide 7

Complexité Le problème est NP-complet.

Histoire Le problème de l'allocation de registres a été résolu par un mathématicien Russe au 19e siècle par coloriage de graphe (en fait, il a résolu le problème analogue de la minimisation du nombre de variables globales dans un programme).

Une solution approchée du coloriage

Principe

Si un nœud n est de degré (*i.e.* nombre de voisins) plus petit que K et si le graphe $G - \{n\}$ est K-coloriable, alors le graphe G est K-coloriable. En effet, une fois $G - \{n\}$ K-colorié il reste au moins une couleur qui ne soit pas celle d'un voisin de n.

Slide 8

Procédure récursive

1. Retirer les nœuds de faible degré (plus petit que K). Cela diminue le degré des nœuds restant et permet de continuer au mieux jusqu'à ce que le graphe soit vide.
2. Dans ce cas le graphe est coloriable, et on est certain de pouvoir attribuer correctement les couleurs au retour de la procédure récursive.
3. Sinon, le graphe peut ne pas être coloriable.

Une stratégie optimiste

À l'aller Lorsque tous les nœuds sont de fort degré le graphe peut ne pas être coloriable. On retire un nœud qui sera éventuellement placer en pile, et on poursuit comme précedemment.

Slide 9

Au retour, on essaye de colorier ce nœud : en effet la solution précédente étant approchée, il se peut que le graphe soit malgré tout coloriable.

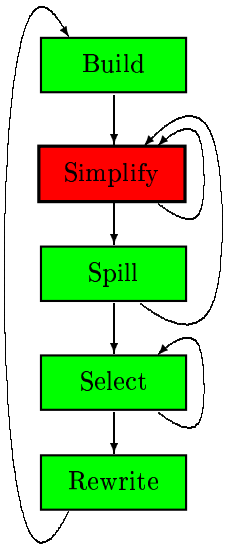
Si ce n'est pas possible, alors on décide de placer définitivement le nœud en pile, sans lui attribuer de couleur, et on poursuit.

À la fin S'il y a eu des placements en pile, il faut réécrire le code, et recommencer : en effet, le placement en pile introduit de nouveaux temporaires...

Procédure itérative

Slide 10

En empilant les nœuds retirés, on obtient :



- Construire le graphe d'interférence
- Retirer un nœud de faible degré du graphe et l'empiler
- Retirer un nœud de fort degré du graphe et l'empiler
- Dépiler et colorier un nœud ou décider de le placer en pile
- Réécrire le code

Heuristique

Slide 11

Lorsqu'un nœud de fort degré est placé en pile, le choix de ce nœud parmi les candidats possibles n'est pas anodin. On veut :

- que ce choix ait un effet maximal, débloquant d'autres nœuds : il faut choisir un nœud de degré élevé.
- que la sauvegarde en pile ne soit pas trop coûteuse : choisir un nœud (temporaire) peu utilisé (dont le nombre d'occurrences dans `Def` et `Use` est faible).

Il faut trouver un compromis entre l'efficacité du spill et son coût. On paramètre ce choix par une fonction de priorité p, par exemple, $p(n) = d(n)/u(n)$ où $u(n)$ et $d(n)$ sont le nombre d'utilisations du nœud n et son degré. (Il faudrait pondérer chaque utilisation par une estimation de sa fréquence d'utilisation : une instruction à l'intérieur d'une boucle interne sera exécutée plus souvent qu'une instruction plus externe.)

Itération

Slide 12

Lorsqu'il y a un placement en pile. La réécriture du code dans la phase en aval va introduire de nouveaux temporaires (dit *auxiliaires*).

Les temporaires auxiliaires ont une durée de vie très courte et en général cela ne pose pas de problème. Ils trouveront leur place dans des interstices. Mais s'il y a une forte demande de registres, il peut être nécessaire d'allouer d'autres temporaires en pile pour faire de la place pour les temporaires auxiliaires.

Ainsi, il faut recommencer l'allocation après la réécriture jusqu'à ce qu'il n'y ait plus de spill. En général, une ou deux itérations suffisent.

On aurait pu aussi pré-réserver des registres pour les temporaires auxiliaires, mais c'est en général un gaspillage de registres (une recette ad-hoc).

Terminaison

Slide 13

Le programme peut donc boucler... Mais cela ne peut se produire que si un temporaire auxiliaire est à nouveau placé en pile, ce qui n'a pas de sens (son propre auxiliaire lui sera isomorphe).

Normalement, cela peut être évité par un réglage de la priorité. (En effet, si tous les temporaires sauf les auxiliaires sont placés en pile, il suffit de trois registres machine pour compiler du code 3-adresses (`a0`, `v0` et `ra`)

On peut détecter le risque de non terminaison et lever une exception si il n'y a plus d'autres solutions que celle de placer un temporaire auxiliaire en pile. Il s'agit alors d'une erreur de conception ou de réglage : trop peu de registres t ou bien d'une mauvaise fonction de priorité (les temporaires utilisés pour un spill dit de deuxième génération ont un coût de spill le plus élevé : à ne spiller qu'en dernier recours).

Prise en compte des instructions MOVE

Slide 14

On peut internaliser le traitement des instructions MOVE dans la procédure d'allocation. Ce qui permet de mieux choisir l'ordre de coloriage et de favoriser l'élimination des MOVE.

Plutôt qu'un choix passif en aval *en remarquant qu'on peut donner la même couleur à deux temporaires*, il est préférable de prévoir en amont de donner ultérieurement la même couleur à deux temporaires.

Cela revient à supposer que leurs nœuds seront fusionnés. Il faut alors mettre à jour le graphe, ce qui

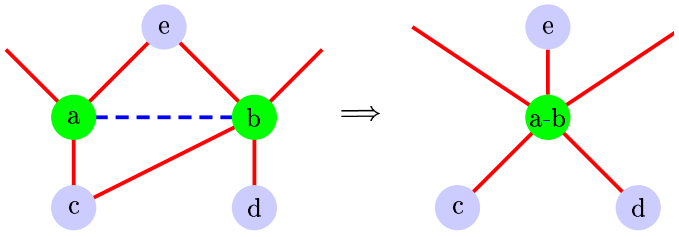
- augmente (ou laisse inchangé) le degré des nœuds fusionnés,
- diminue (ou laisse inchangé) le degré des autres nœuds (voisins).

L'allocateur prendra alors automatiquement ces choix en compte pour déterminer un meilleur ordre de coloriage.

Fusion (coalescence) et son effet

Slide 15

On représente les MOVE par des arcs en pointillés.



Le degré des nœuds a et b augmente.

Le degré du nœud c diminue.

Le degré des autre nœuds sont inchangés.

Une stratégie sure

Ne fusionner les nœuds (a) et (b) dans le graphe G que si cela préserve sa coloriabilité.

Deux critères sûr

Slide 16

1. Après fusion le nœud (a-b) a moins de K voisins de fort degré.
2. Le nœud (a) est tel que tous ses voisins de fort degré interfèrent avec (b). (*typiquement (b) est un registre*)

Preuve : après avoir éliminé tous les nœuds de faible degré dans le graphe résultant

1. le nœud (a-b) a moins de K voisins et peut aussi être retiré.
2. le nœud (a-b) peut être identifié avec le nœud (b) de G.

Dans les deux cas, il reste un sous-graphe de G, donc coloriable.

Allocation combinée

On représente dans le graphe les MOVE par des arcs en pointillés. Les nœuds qui n'ont pas d'arc MOVE sont dits *simplifiables*, les autres sont dits *complexes*.

La fusion augmente le degré des nœuds fusionnés : on ne peut donc pas simplifier un nœud tant qu'il est complexe.

Slide 17

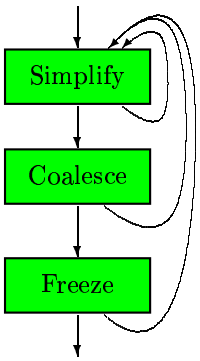
Par priorité :

1. on retire du graphe un nœud simplifiable de faible degré (comme précédemment) ;
2. on effectue une fusion sure (qui préserve la coloriabilité) ;
3. on retire tous les arcs MOVE d'un nœud complexe de faible degré (abandonnant tout espoir de le fusionner) ce qui le rend simplifiable ;
4. on retire un nœud de fort degré qui sera a priori placé en pile (spill).

Allocation combinée, graphiquement

Slide 18

Remplacer la boîte Simplify de l'algorithme précédent par :



- Retirer du graphe un nœud simplifiable de faible degré
- Effectuer une fusion sure
- Retirer un arc MOVE

Au pire, aucune fusion n'est possible : les arcs MOVE sont retirés un à un et on retombe sur (une trace de) l'algorithme précédent.

Mise en œuvre

Slide 19

Au cours du calcul, on maintient des ensembles de nœuds (et d'arcs). Les nœuds sont dans différents états (simple/complexe, faible/fort degré, etc.). Chaque étape revient à choisir un nœud dans un certain état pour le placer dans un autre état (et ajuster le graphe).

Le module `partition` avec l'interface ci-dessous permet de maintenir à jour l'ensemble des nœuds (et d'arcs) dans chaque état de façon efficace (paresseuse).

```
type 'a partition          type 'a elem
val make : int -> 'a partition  array
val create : 'a partition  -> 'a -> 'a elem
val info : 'a elem -> 'a
val belong : 'a elem -> 'a partition  -> bool
val move : 'a elem -> 'a partition  -> unit
val pick : 'a partition  -> 'a elem
```

Représentation des nœuds et des arcs

Slide 20

```
type node_info = {
  temp : temp;                    (* temporaire associé *)
  mutable moves : move list;      (* arcs moves *)
  mutable adj : node list ;       (* arcs d'interference *)
  mutable degree : int;
  mutable alias : node option;    (* noeud fusionné *)
  mutable color : int ;
  mutable occurs : int;           (* nombre d'utilisations *)
}
and move_info =
   { instr : Ass.instr ; left : node; right : node; }
and node = node_info elem
and move = move_info elem;;
```

Les partitions de nœuds

Slide 21

On distingue états pour les nœuds

- pré-coloriés : ne changeront pas d'état
- initiaux : état temporaire de tous les nœuds non coloriés juste après la construction du graphe d'interférence ;
- simplifiables de faible degré : candidats à la simplification ;
- complexes de faible degré : candidats pour être gelés (i.e. pour geler leurs arcs MOVE) ;
- de fort degré : candidats pour être placés en pile ;
- spilled : définitivement programmés pour être placés en pile ;
- fusionés : leur champ alias contient le nœud résultant ;
- empilés : en attente d'être coloriés (l'ordre est essentiel) ;
- coloriés : état final de tous les nœuds non pré-coloriés.

Les partitions d'arcs MOVE

On distingue 5 états pour les arcs MOVE :

- fusionnés : ne seront plus considérés.
- contraints : ne peuvent être fusionnés car source et destination interfèrent. Ne seront plus considérés.
- gelés : ceux pour lesquels la fusion a été définitivement abandonnée ; ils ont été retirés du graphe, et ne seront plus considérés.

Slide 22

- candidats à la fusion.
- candidats à la fusion temporairement bloqués tant que leur fusion n'est pas sure. Ils seront reconsidérés (candidats à la fusion) dès que le degré d'un de leurs voisins aura diminué.

Implémentation

L'implémentation suit l'algorithme pas à pas : sélectionner un nœud ou un arc à traiter, le traiter, puis mettre le graphe (i.e. l'état des nœuds et des arcs voisins) à jour.

Ré-écriture du code

Dans le cas où il y a des registres à placer en pile, il faut réécrire le code et recommencer.

Slide 23

On peut prendre en compte les fusions qui ont eu lieu avant le premier spill, à l'exclusion de celles vers un registre mis en pile[a]. En effet, celles-ci seront forcément reproductibles.

Par contre, il faut ignorer toutes les fusions qui ont eu lieu après un spill. Pour cela, on sauvegarde l'état des temporaires et des MOVE déjà fusionnés au moment du premier spill.

[a] Une fusion conservatrice préserve mais n'impliquement pas la coloriabilité ; un registre peut donc être fusionné vers un registre qui se retrouve spillé.

Exemple : code de fact

Slide 24

```
    fact_f =8
fact:
    sub  $sp, $sp, fact_f
    sw   $s0, 0($sp)
    sw   $ra, 4($sp)
    li   $t9, 1
    bgt  $a0, $t9, L8
L7:
    li   $v0, 1
L9:
    lw   $s0, 0($sp)
    lw   $ra, 4($sp)
    add  $sp, $sp, fact_f
    j    $ra
L8:
    move $s0, $a0
    sub  $a0, $a0, 1
    jal  fact
    mul  $v0, $s0, $v0
    j    L9
```

Comparer avec le code juste après son émission...

Ajustements

On peut assembler l'ensemble de la chaîne de compilation.

Ajustement de la pile Une fois l'allocation de registres effectuée, la taille de la pile est connue. Il reste à déclarer celle-ci comme une constante en tête du prélude par une instruction de la forme :

Slide 25

```
Oper ("fact_f=8", [], [], None)
```

(Dans le cas où aucun spill n'est nécessaire, on pourrait supprimer les instructions d'ajustement de la pile)

Procédure principale En principe, les temporaires pourraient être placés dans des globaux. Pas soucis de simplicité, on peut revoir la compilation de la procédure principale et la traduire en un appel à une procédure `main`. On place donc le code principal dans un appel de procédure.

Exercices

Allocation de registres triviale

Dans un premier temps, on cherche simplement à générer du code exécutable. On peut choisir la sauvegarde systématique des registres $s0, s1, ...$ en pile. Il suffit alors d'implanter le coloriage trivial sans prendre en compte les MOVE.

Slide 26

Spilling

Implémenter le spill d'un registre. On peut maintenant sauver les registres $s0, s1, ...$ dans des temporaires et l'allocateur de registre les sauvera en pile seulement si nécessaire.

Élimination des MOVE

Éliminer les MOVE au moment de la sélection des couleurs. Pour espérer éliminer beaucoup de move, il faut dans le coloriage, privilégier les arcs move.

Mini projet

Algorithme de coloriage

Implémenter l'algorithme de coloriage (sans optimiser les MOVE).

Utilisera une heuristique simple pour le choix des registres proposés pour l'allocation en pile.

Slide 27

Prise en compte simultanée des MOVE

L'implantation de l'algorithme complet est un projet. On pourra utiliser un module implémentant des partitions d'ensembles (partition.ml : partition.mli).

Exhiber des exemples montrant les limites d'une allocation de registres trop élémentaire et l'efficacité de l'allocation fine.