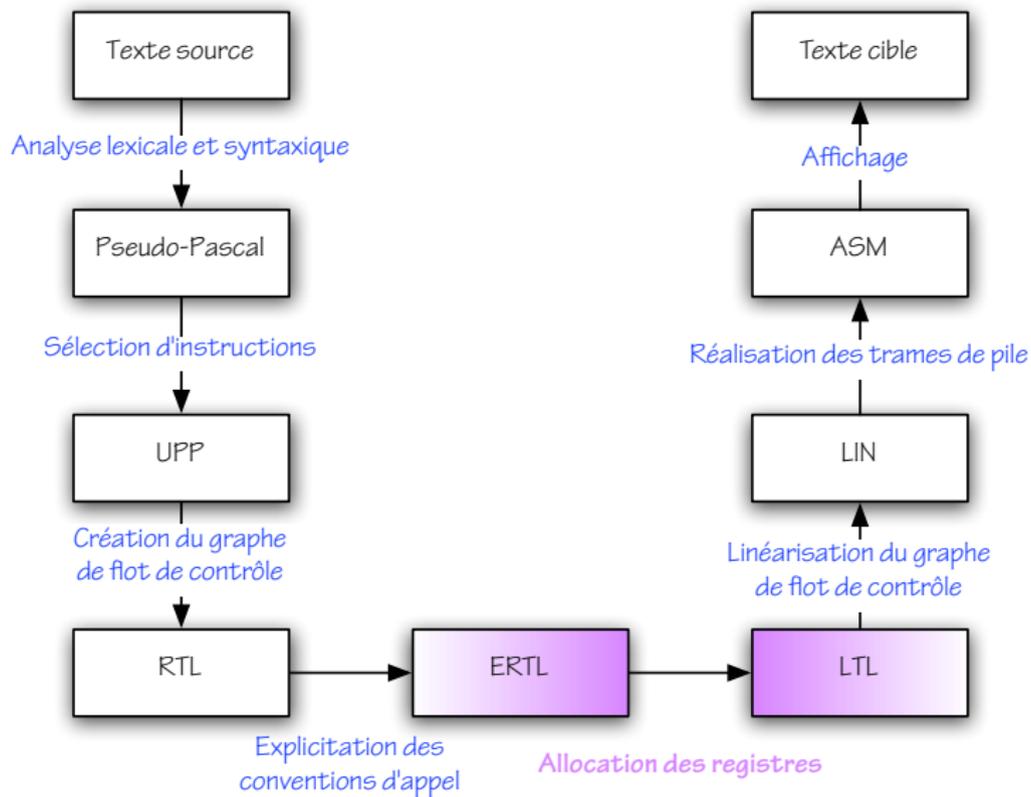


Compilation (INF 553)

Allocation de registres par coloriage de graphes

François Pottier

21 mars 2007



Allocation de registres

On souhaite *réaliser* chaque pseudo-registre par un registre physique, ou, à défaut, un emplacement de pile.

On s'intéresse uniquement aux registres physiques dits *allouables*, c'est-à-dire *non réservés* pour un usage particulier, comme le sont par exemple \$gp ou \$sp.

Les registres exploités par la convention d'appel, à savoir \$v0, \$a0-\$a3, et \$ra *sont* allouables. Les registres \$t0-\$t9 et \$s0-\$s7 le sont également.

De l'allocation de registres au coloriage de graphes

Nous avons construit un *graphe d'interférences* dont:

- ▶ les sommets sont *les pseudo-registres et les registres physiques allouables*;
- ▶ une *arête d'interférence* relie deux sommets qui doivent recevoir des emplacements distincts;
- ▶ une *arête de préférence* ou arête « move » relie deux sommets à qui on souhaiterait attribuer le même emplacement.

De l'allocation de registres au coloriage de graphes

Supposons que l'on dispose de k registres physiques allouables. Alors le problème de l'allocation de registres *semble* se résumer à:

- ▶ attribuer une *couleur* parmi k à chaque sommet représentant un pseudo-registre,
- ▶ de façon à ce que *deux sommets reliés par une arête d'interférence ne reçoivent jamais la même couleur*,
- ▶ et si possible de façon à ce que deux sommets reliés par une arête de préférence reçoivent la même couleur.

Le graphe est dit *k -colorable* si ce nouveau problème admet une solution.

Historique

L'idée de réduire l'allocation de registres au coloriage de graphes date des années 1960, mais a été mise en pratique pour la première fois par Chaitin en 1981.

Ce cadre théorique est attirant de par sa *simplicité*, mais quelques problèmes demeurent...

Difficultés et limitations

Premier problème:

- ▶ le problème du coloriage de graphes est *NP-complet*, d'où, en pratique, impossibilité de construire une solution optimale.

En réponse, tous les compilateurs actuels utilisent des *heuristiques* de complexité linéaire ou quasi-linéaire.

Voir « *A generic NP-hardness proof for a variant of Graph Coloring* » (Bodlaender, 2001).

Difficultés et limitations

Deuxième problème:

- ▶ si le graphe *n'est pas* k -colorable ou si on ne *trouve pas* de k -coloriage, que faire?

L'idée la plus simple est de permettre à certains sommets de rester *non colorés* et de réaliser ensuite ces pseudo-registres par des emplacements de pile. On parle alors de « *spill* ».

Les détails de ce processus sont *plus subtils* qu'il n'y paraît, et ce problème, dans toute sa généralité, offre lui aussi un espace de choix *colossal*.

Difficultés et limitations

Troisième et dernier problème:

- ▶ certaines architectures existantes n'offrent pas k registres physiques *indépendants* et *interchangeables*.

Heureusement, on peut modifier l'algorithme de coloriage de graphes pour refléter les irrégularités et particularités les plus courantes.

Algorithmes de coloriage

Algorithmes de « spill »

Quelques problèmes pratiques

Simplification

Kempe (1879) et Chaitin (1981) ont observé qu'un sommet s de degré strictement inférieur à k est *trivialement colorable*: le graphe G est k -colorable si et seulement si G privé de s est k -colorable.

On peut répéter cette *simplification* autant de fois que possible. De plus, le fait de supprimer un sommet trivialement colorable peut *rendre d'autres sommets* trivialement colorables.

L'algorithme de Chaitin

```
procédure Colorier( $G$ )  
  si il existe un sommet  $s$  trivialement colorable  
  alors  $\left\{ \begin{array}{l} \text{Colorier}(G \setminus s) \\ \text{attribuer une couleur disponible à } s \end{array} \right.$   
  sinon si il existe un sommet  $s$   
  alors  $\left\{ \begin{array}{l} \text{Colorier}(G \setminus s) \\ \text{« spiller » } s \end{array} \right.$ 
```

Cet algorithme peut être implanté avec une *complexité linéaire*.
(Comment?)

Le comportement de l'algorithme est influencé par la façon dont certains *choix* sont effectués...

Choix de sommets

Le choix d'un sommet trivialement colorable, lorsqu'il en existe plusieurs, n'est pas fondamental. (Pourquoi?)

Le *choix d'un sommet à « spill »* est critique:

- ▶ pour une meilleure efficacité, il faut choisir un pseudo-registre *peu utilisé* ou *utilisé en des points peu critiques* du code;
- ▶ pour faciliter la suite du coloriage, il vaut mieux choisir un sommet de *fort degré*.

On fait appel à une fonction de coût qui combine ces critères.

Emploi des registres « callee-save »

Cette heuristique permet naturellement un bon emploi des registres « *callee-save* ».

En effet, les pseudo-registres introduits pour sauvegarder le contenu des registres « *callee-save* » sont *peu utilisés* — une écriture et une lecture — et ont une très longue durée de vie, donc un *fort degré*.

Ils seront donc « *spillés* » de préférence. Ainsi, les registres *callee-save* seront sauvegardés à l'entrée, restaurés à la sortie, et disponibles entre les deux.

Choix de couleurs

Le *choix de la couleur* attribuée à un sommet s trivialement colorable après coloriage de $G \setminus s$ est important: on a ici une occasion de respecter les souhaits exprimés par les *arêtes de préférence*.

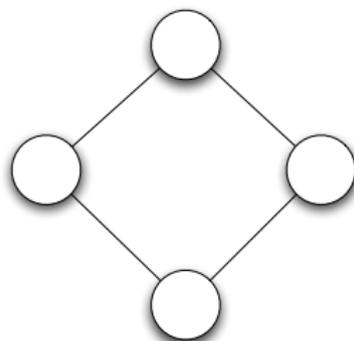
Supposons t relié à s par une arête de préférence.

On attribuera à s *la couleur déjà attribuée* à t , si t est déjà coloré; ou bien *une couleur encore permise* pour t , si t n'est pas encore coloré; etc.

Cette technique de *coloriage biaisé* est simple mais limitée. Le «*coalescing*» lui sera supérieur.

Des méfaits du pessimisme

Pour $k = 2$, ce graphe est-il coloriable? Que donne l'algorithme de Chaitin?



Comment améliorer l'algorithme?

Coloriage optimiste

```
procédure Colorier( $G$ )  
  si il existe un sommet  $s$  trivialement colorable  
  alors { Colorier( $G \setminus s$ )  
          attribuer une couleur disponible à  $s$   
  sinon si il existe un sommet  $s$   
  alors { Colorier( $G \setminus s$ )  
          si une couleur reste disponible pour  $s$   
          alors la lui attribuer  
          sinon « spiller »  $s$ 
```

Voir [Improvements to Graph Coloring Register Allocation](#) (Briggs et al., 1994).

« Coalescing »

Revenons à la question du respect des *arêtes de préférence*.

Une approche plus radicale que le coloriage biaisé, également due à Chaitin, consiste à d'abord *fusionner* (*coalescer*) tous les sommets reliés par des *arêtes de préférence* avant d'effectuer le coloriage.

La couleur attribuée à deux ou plusieurs sommets fusionnés est celle attribuée à leur agrégat.

Fusionner deux sommets peut amener une *arête d'interférence* et une *arête de préférence* à se chevaucher; dans ce cas, la seconde disparaît.

Avantages et inconvénients du « coalescing » sauvage

Cette forme dite *agressive* ou *sauvage* de « coalescing » présente plusieurs avantages:

- ▶ *toutes* les arêtes de préférence, sauf celles éclipsées par une arête d'interférence, sont respectées;
- ▶ fusionner deux sommets *diminue le degré* de leurs voisins communs et peut donc les rendre trivialement colorables.

Cependant, elle souffre également d'un inconvénient:

- ▶ fusionner deux ou plusieurs sommets peut créer un sommet de *très fort degré*, qui sera probablement « spillé », tandis que, sans fusion, un ou plusieurs des sommets initiaux auraient peut-être pu être colorés.

« Coalescing » conservatif

Le « coalescing » *conservatif* ou *prudent* consiste à ne fusionner deux sommets que si l'on a la *certitude* de *préserver la k-colorabilité* du graphe.

Briggs et George proposent chacun une *condition suffisante* pour cela...

Le critère de Briggs

Briggs propose le critère suivant:

Deux sommets peuvent être fusionnés si le sommet résultant a moins de k voisins non trivialement colorables.

On exige en fait que le sommet résultant soit trivialement colorable *modulo simplification*. Parce que le « coalescing » peut avoir lieu avant certaines étapes de simplification, on *anticipe* l'effet de celles-ci.

Voir Appel, p. 232.

Le critère de George

George propose le critère suivant:

Deux sommets a et b peuvent être fusionnés si tout voisin non trivialement colorable de a est également voisin de b .

On *anticipe* encore une fois l'effet des simplifications futures. Une fois celles-ci effectuées, tout voisin de a sera voisin de b , donc *toute couleur permise pour b le sera également pour a .*

Voir Appel, p. 232.

Les critères de Briggs et George sont incomparables. (Pourquoi?)

Complications

Le « coalescing » conservatif induit quelques *subtilités* qui viennent compliquer l'algorithme...

- ▶ deux sommets non fusionnables *peuvent le devenir* suite à une simplification ;
- ▶ un sommet non trivialement colorable *peut le devenir* suite à une fusion.

Il faut donc *alterner* entre les deux phases. Cependant...

Complications

Reste un point délicat:

- ▶ la simplification doit *éviter de retirer un sommet susceptible d'être fusionné* à l'avenir...
- ▶ ... *sauf* si cela empêche toute simplification.

Un sommet portant une arête de préférence (« move-related ») n'est donc jamais simplifié.

Lorsque ni simplification ni fusion ne sont possibles, on « *gèle* » — en fait, on *supprime* — certaines arêtes de préférence. Cela peut permettre de nouvelles simplifications, donc de nouvelles fusions, etc.

L'algorithme de George et Appel (1/4)

procédure Colorier(G)
Simplifier(G)

procédure Simplifier(G)

si s est trivialement colorable

et ne porte aucune arête de préférence

alors $\left\{ \begin{array}{l} \text{soit } G' = G \setminus s \\ \text{Simplifier}(G') \\ \text{attribuer une couleur disponible à } s \end{array} \right.$

sinon Fusionner(G)

La simplification est *restreinte* de façon à ne pas supprimer d'opportunités de fusion. Lorsqu'elle ne s'applique plus, la *fusion* commence.

L'algorithme de George et Appel (2/4)

procédure Fusionner(G)

si a et b peuvent être fusionnés

alors $\left\{ \begin{array}{l} \text{soit } G' = G \text{ où un unique sommet } ab \text{ remplace } a \text{ et } b \\ \text{Simplifier}(G') \\ \text{attribuer à } a \text{ et } b \text{ la couleur attribuée à } ab \end{array} \right.$

sinon Geler(G)

La fusion est *restreinte* par un critère conservatif (Briggs, George, ou autre). Après chaque fusion, la *simplification* reprend. Si aucune fusion (donc aucune simplification) n'est possible, le *gel* commence.

L'algorithme de George et Appel (3/4)

```
procédure Geler( $G$ )  
  si  $s$  est trivialement colorable  
  alors { soit  $G' = G$  privé des arêtes de préférence issues de  $s$   
          Simplifier( $G'$ )  
  sinon Spill( $G$ )
```

S'il existe un sommet que l'on n'a pas simplifié dans l'espoir d'une fusion, le gel consiste à *abandonner cet espoir* pour permettre la simplification. Si aucun gel (donc ni simplification ni fusion) n'est possible, on passe au « spill » optimiste.

L'algorithme de George et Appel (4/4)

procédure Spill(G)

si s est un sommet de coût minimal

alors $\left\{ \begin{array}{l} \text{soit } G' = G \setminus s \\ \text{Simplifier}(G') \\ \text{si une couleur reste disponible pour } s \\ \text{alors la lui attribuer} \\ \text{sinon « spiller » } s \end{array} \right.$

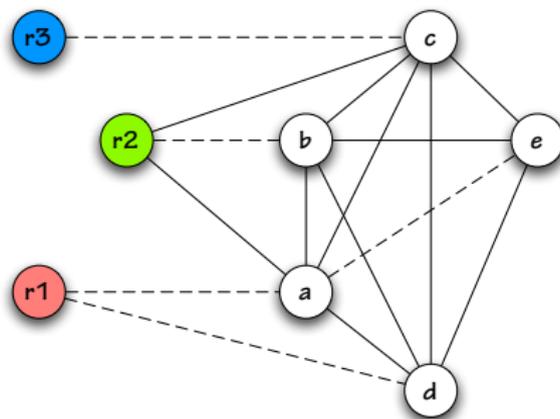
Le « spill » optimiste est identique à celui discuté précédemment.

Cet algorithme peut être implanté avec une *complexité linéaire*, si on suppose le graphe *de degré borné*. Toutefois, le code devient complexe.

Voir Appel, chapitre 11.

Exemple

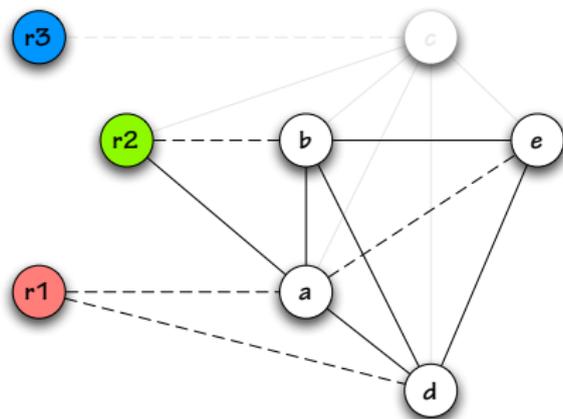
On suppose $k = 3$. On applique le critère de George.



Aucune simplification, fusion ou gel n'est possible... On va donc « spill » un sommet, *par exemple c*.

Exemple

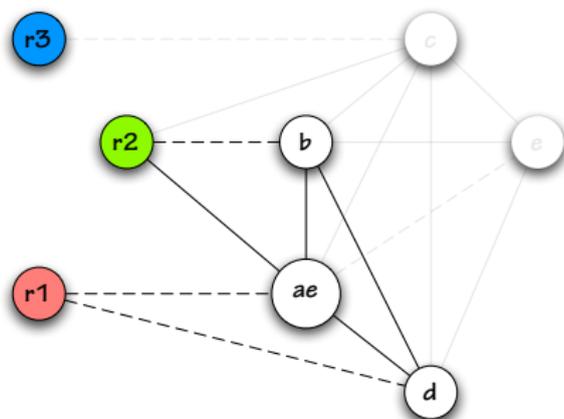
c a été supprimé du graphe.



e est trivialement colorable mais non simplifiable. *r2* et *b* ne sont pas fusionnables car *r2* n'interfère pas avec *d*. *a* et *e* le sont.

Exemple

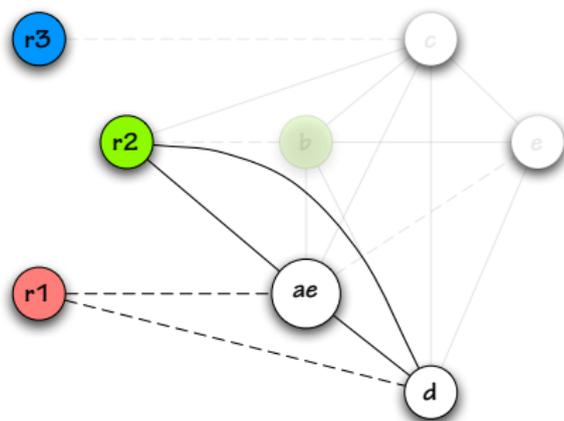
a et *e* ont été fusionnés.



On peut fusionner *r2* et *b* ou bien *r1* et *ae*, car *b* et *d* sont trivialement colorables. On choisit les premiers.

Exemple

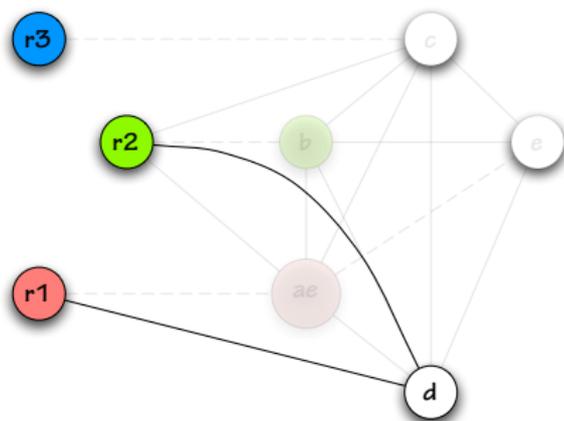
r2 et b ont été fusionnés. r2 interfère maintenant avec d.



Tous les sommets sont trivialement colorables. On peut donc fusionner *r1* et *ae* ou bien *r1* et *d*. On choisit les premiers.

Exemple

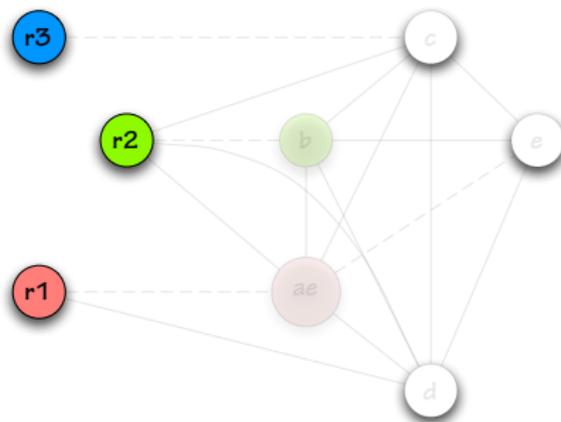
r1 et *ae* ont été fusionnés.



d est trivialement colorable. Il est donc à présent possible de le *simplifier*.

Exemple

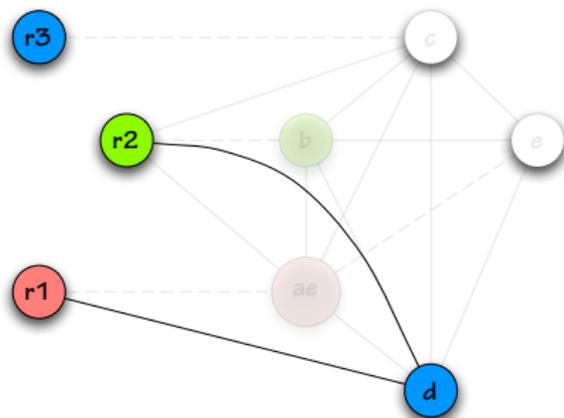
Le graphe est maintenant vide.



On reprend donc la série de transformations qui nous a mené jusqu'ici, *dans l'ordre inverse*, et en attribuant des couleurs.

Exemple

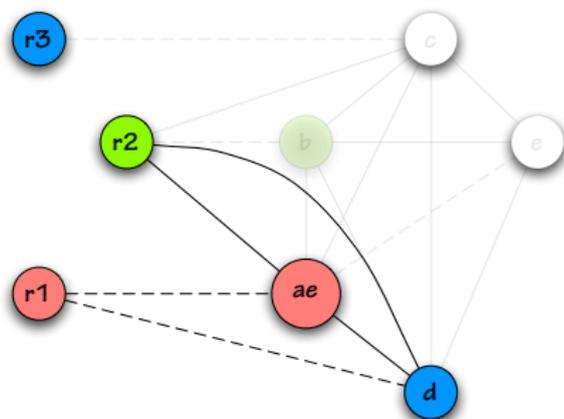
d avait été simplifié.



Il reçoit l'unique *couleur disponible*, celle que ses deux voisins n'exploitent pas, à savoir *r3*.

Exemple

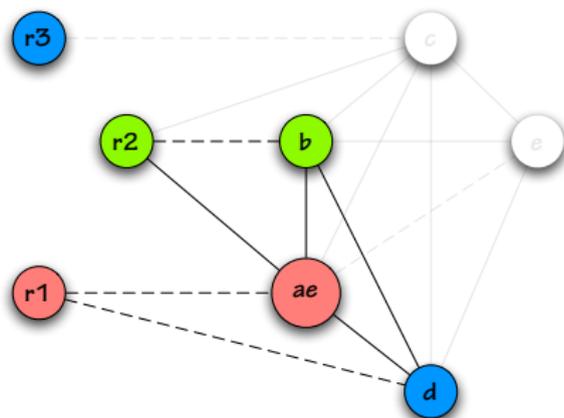
ae avait été coalescé avec *r1*.



Il reçoit donc la couleur *r1*. Bien sûr, nous savions déjà cela.

Exemple

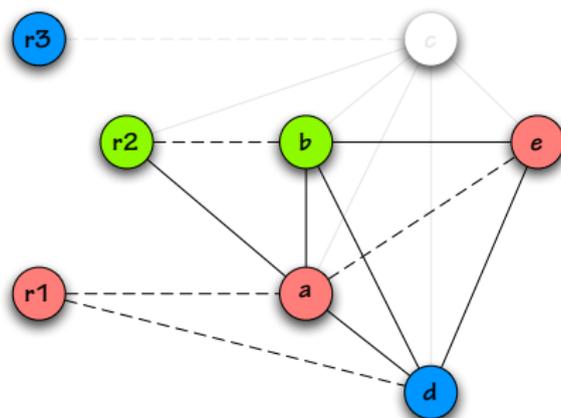
De même, b avait été coalescé avec $r2$.



Il reçoit donc la couleur $r2$.

Exemple

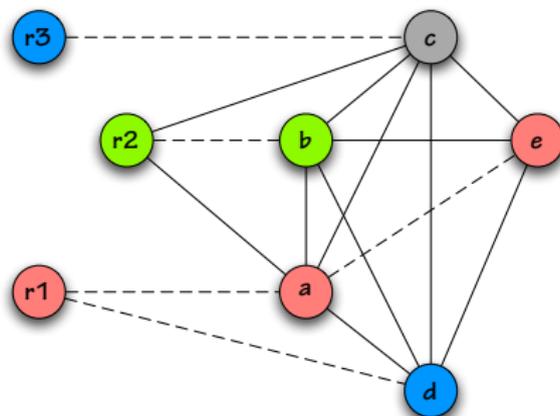
e avait été coalescé avec *a*.



Il reçoit donc la même couleur que *a*, à savoir *r1*.

Exemple

Enfin, c était *candidat* au « spill ».



Parce que ses cinq voisins exploitent les $k = 3$ couleurs, il doit *effectivement* être « spillé ».

En résumé

L'algorithme de George et Appel est efficace et produit du code de bonne qualité. *C'est celui implanté* dans le petit compilateur.

« Coalescing » optimiste

Park et Moon (1998) ont suggéré *l'optimisme* plutôt que la prudence:

- ▶ on *fusionne agressivement*, ce qui présente l'avantage d'ouvrir de nouvelles opportunités de simplification;
- ▶ si on est amené à « spiller » un sommet obtenu par fusion, on *défait* la fusion dans l'espoir d'éviter le « spill »;
- ▶ cela crée une *interaction* avec la simplification: défaire une fusion peut rendre non trivialement colorable un sommet déjà retiré du graphe car considéré comme trivialement colorable...

En bref, l'inventivité est sans fin.

Algorithmes de coloriage

Algorithmes de « spill »

Quelques problèmes pratiques

Problème

Lorsque certains sommets n'ont pu être colorés, il faut *modifier le code* pour diminuer le nombre de registres physiques nécessaires simultanément.

Mais comment ?

D'innombrables modifications différentes et correctes sont possibles, car on peut, en tout point du code, décider de *transférer* une valeur d'un registre vers la mémoire ou vice-versa.

Une solution simple

Soit $\%p$ un pseudo-registre « spillé ».

La solution la plus simple, due toujours à Chaitin, consiste à:

- ▶ *réserver* un emplacement de pile;
- ▶ considérer que le contenu de $\%p$ sera stocké *à tout instant* à cet emplacement.

Si la machine cible est *CISC*, les instructions qui définissent ou utilisent $\%p$ peuvent donc être traduites en des instructions qui lisent un opérande ou écrivent un résultat *directement en mémoire*.

Mais, ...

La poule et l'œuf

Toutefois, si la machine cible est *RISC*, seules des instructions dédiées, **lw** et **sw** pour le MIPS, permettent d'accéder à la mémoire.

Il faut donc émettre un **sw** après chaque *écriture* dans %p et un **lw** avant chaque *lecture*.

Une instruction ERTL comme **add** peut ainsi être traduite, au pire, par une séquence de *deux lw*, un **add**, et un **sw**...

Mais *quels registres physiques* utiliseront ces quatre instructions pour communiquer!?

Quelques œufs de côté

Deux réponses sont possibles:

- ▶ on fait apparaître les instructions **lw** et **sw** dans le code *source*, c'est-à-dire ERTL; leurs opérandes sont alors des *pseudo-registres*; et on *recommence* l'allocation de registres à partir du début. Pour garantir la terminaison, on *interdit* de « spiller » les pseudo-registres nouvellement introduits.
- ▶ on réserve deux registres physiques, que l'on rend *non allouables*, et on les emploie ici.

La première approche est plus courante dans la littérature et la *seule viable* lorsque le processeur a *peu de registres*. La seconde est *plus simple* et adoptée dans le petit compilateur.

Allocation des emplacements de pile

Comment décide-t-on que tel pseudo-registre occupera tel emplacement, en *respectant* arêtes d'interférence et de préférence?

C'est simple – cette fois, réellement. Il s'agit d'un problème de *coloriage du graphe résiduel* formé par les pseudo-registres « spillés » seuls, pour $k = \infty$.

On applique les heuristiques de simplification et de « coalescing » agressif.

Au-delà de la solution simple

On peut faire mieux qu'insérer des **lw** et **sw** à chaque utilisation ou définition.

Par exemple, si un site d'utilisation est « proche » d'un site de définition ou d'un site d'utilisation antérieur, alors un nouveau **lw** n'est pas nécessaire.

En d'autres termes, on peut exploiter le fait que, par moments, la valeur existe *simultanément* dans un registre physique et en mémoire.

Au-delà de la solution simple

Deuxième amélioration: parfois, il est moins coûteux de *recalculer* une valeur que de la *stocker* en mémoire.

C'est le cas si cette valeur est *constante* ou bien une *expression connue*, par exemple la somme des valeurs de deux registres physiques toujours disponibles. On détecte ces situations à l'aide d'une technique d'analyse proche de celle requise pour l'élimination des sous-expressions communes (CSE) (cours n° 5).

Cette technique est appelée *rematérialisation*. On notera que c'est essentiellement *l'inverse* de la CSE!

Au-delà de la solution simple

D'autres techniques encore ont été étudiées:

- ▶ « *live range splitting* »: effectuer une mise en forme SSA pour décorréler les différents usages d'un même pseudo-registre;
- ▶ « *interference region splitting* »: supprimer l'interférence entre deux pseudo-registres en émettant des **lw** et **sw** *seulement* dans la région où l'interférence se produit;
- ▶ « *spill propagation* »: effectuer une fusion pour « spiller » sur une région plus grande est parfois bénéfique.

Idées de plus en plus complexes et parfois contradictoires!

Une approche optimale

On peut être tenté de dire « *stop* » et de rechercher une fois pour toutes une solution *optimale*. Diverses méthodes ont été étudiées mais restent pour l'instant *trop inefficaces*.

L'idée (Goodwin & Wilken, 1996) est de réduire le problème à un *système d'équations et inéquations entre variables entières ou booléennes* et de le soumettre à un solveur spécialisé pré-existant (« integer linear programming »).

Dans la même veine, voir également « *Optimal Spilling for CISC Machines with Few Registers* » (Appel et George, 2001).

Algorithmes de coloriage

Algorithmes de « spill »

Quelques problèmes pratiques

Le registre \$zero

Le registre \$zero du MIPS contient toujours la valeur 0, même si on y a écrit une autre valeur. Est-il *inutilisable* ?

En fait, on peut le déclarer *allouable* si l'on considère qu'il *interfère* avec tous les pseudo-registres où l'on écrit des valeurs non nulles. Ainsi, le fragment de code:

```
li    %0, 0
sw    %0, 0($gp)
```

pourra être traduit par:

```
sw    $zero, 0($gp)
```

Code à deux adresses

L'Intel x86 a une instruction d'addition à *deux* adresses au lieu de trois:

add r_1, r_2

Le registre r_1 est à la fois *source* et *destination*.

Si le code intermédiaire est à trois adresses, on peut y *insérer* des instructions **move** pour garantir que la destination et le premier opérande de chaque addition coïncident.

L'idée est de réécrire « **add** r_1, r_2, r_3 » en « **move** r_1, r_2 ; **add** r_1, r_1, r_3 ». Cette transformation est correcte si r_1 et r_3 sont deux pseudo-registres distincts; si par malheur ce n'est pas le cas, il faut ruser...

Classes de registres

La plupart des processeurs ont plusieurs *classes* de registres physiques, par exemple entiers et flottants. Si ces classes sont *disjointes*, cela donne lieu à deux problèmes d'allocation *distincts*.

Mais, parfois, les registres physiques ne sont pas *interchangeables* ou bien pas *indépendants*...

Interchangeabilité

Deux registres sont *interchangeables* s'ils ont exactement les mêmes usages.

Ce n'est pas toujours le cas. Par exemple, sur les Motorola 68K, l'instruction d'addition accepte un registre « d'adresse » ou de « de données » en tant que source, mais l'instruction de multiplication n'accepte qu'un registre « de données ».

Indépendance

Deux registres sont *indépendants* s'il est possible d'écrire dans l'un sans modifier le contenu de l'autre.

Ce n'est pas toujours le cas. Par exemple, sur certains processeurs ARM, les registres flottants peuvent être *organisés au choix* en 32 registres simple précision, 16 registres double précision, 8 registres scalaires et 6 vecteurs de 4 éléments, ou bien 8 registres scalaires et 3 vecteurs de 8 éléments!

Heureusement, on peut traiter les cas les plus courants à l'aide d'un algorithme de coloriage généralisé: cf. « *A Generalized Algorithm for Graph-Coloring Register Allocation* » (Smith et al., 2004).

Pour finir

Pour passer de ERTL à LTL, le petit compilateur utilise:

Kildall	Calcul de point fixe
Liveness	Analyse de durée de vie
Interference	Opérations sur le graphe d'interférences
Zero	Cas particulier pour \$zero
Build	Construction du graphe d'interférences
Uses	Comptage des emplois des pseudo-registres
Coloring	Coloriage à la George et Appel
Spill	Coloriage avec $k = \infty$
Ertl2rtl1	Introduction des lw et sw
Ertl2rtl	Combinaison du tout