Cours de compilation

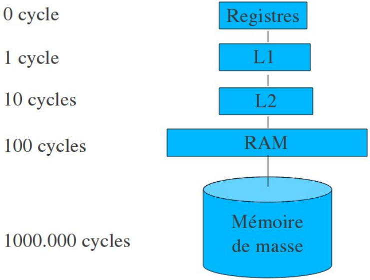
Chapitre 10. Allocation de registres

Master informatique fondamentale 1 – ENS-Lyon

Christophe Alias
Christophe.Alias@ens-lyon.fr
http://perso.ens-lyon.fr/christophe.alias

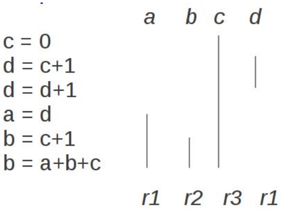
Introduction

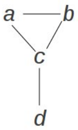
- Après sélection et ordonnancement, on obtient du code machine avec des temporaires. Où les stocker?
- De préférence dans des registres.
- En mémoire (dans la pile) quand tous les registres sont déjà pris.



Durées de vie, interférences

- Une variable est vivante en un point du graphe de flot de contrôle s'il existe un chemin de ce point vers une lecture sans passer par une écriture.
- x interfère avec y s'il existe un point du CFG sur lequel x et y sont vivants en même temps.
  ☛ils doivent alors être placés dans des registres differents.
- On construit un graphe d'interférence, dont les noeuds sont les variables. Un arc entre x et y indique une interférence entre x et y.

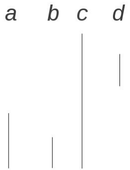


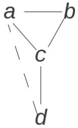


Copies

- La sortie de la forme SSA produit de nombreuses instructions de copie (`MOV a,d`) qu'on aimerait supprimer en affectant le même registre à `a` et `d`.
- On modélise cette situation en ajoutant un arc d'affinité entre `a` et `d` dans le graphe d'interférence.

```
c = 0
d = c+1
d = d+1
a = d
b = c+1
b = a+b+c
```





Allocation de registres

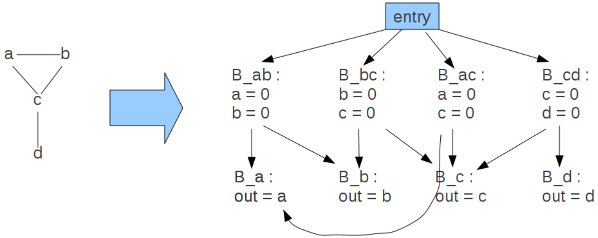
Le problème de l'allocation de registres pour k registres consiste à attribuer une couleur parmis k à chaque temporaire de telle façon que:

- deux sommets reliés par un arc d'interférence ne recoivent pas la même couleur.
- deux sommets reliés par un arc d'affinité recoivent la même couleur, si possible.

S'il existe une solution, on dit que le graphe est k-colorable.

Complexité

Le graphe d'interférence peut être quelconque:



Or, le problème du coloriage de graphes est NP-complet...

Coloriage par simplification

Un sommet de degré strictement inférieur à k est trivialement colorable.
G est k-colorable ssi $G - \{s\}$ est k-colorable

On peut répéter cette simplification autant de fois que possible.

Le fait de supprimer un sommet trivialement colorable peut rendre d'autres sommets trivialement colorables...

Algorithme de Chaitin

```
colorier(G)
  si il existe un noeud s avec < k voisins
  alors //cas trivial
    colorier(G − {s})
    affecter une couleur restante à s
  sinon
    choisir un noeud s
    colorier(G − {s})
    si il reste une couleur alors l'affecter
    sinon spiller s
  fin
```

- Le choix du sommet à “spiller” est critique.
- Pour faciliter la suite du coloriage, il vaut mieux choisir un sommet de fort degré.
- Le choix de la couleur doit tenir compte des arcs de préférence (coloriage biaisé).

Coalescing

Une approche plus radicale que le coloriage biaise, consiste à d'abord fusionner (coalescer) tous les sommets reliés par des arcs d'affinité avant d'effectuer le coloriage.

La couleur attribuée à plusieurs sommets fusionnés est celle attribuée a leur aggrégat.

Inconvénient: fusionner plusieurs sommets peut créer un sommet de très fort degre, qui sera probablement "spillé".

Coalescing conservatif

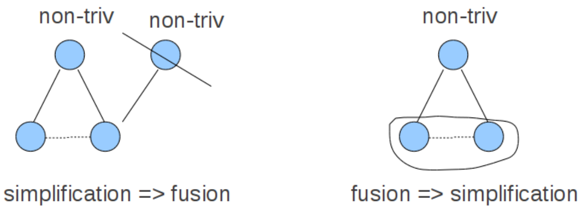
On ne fusionne deux sommets que si on a la certitude de préserver la k-colorabilité du graphe.

- Critère de Briggs
  Deux sommets peuvent etre fusionnés si le sommet résultant a moins de k voisins non trivialement colorables
  ➠Trop conservatif

- Critère de Georges
  Deux sommets a et b peuvent être fusionnés si tout voisin non trivialement colorable de a est également voisin de b.
  ➠Préféré en pratique

Interactions simplification/fusion

- 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 alterner simplification et fusion.
- Quand le critère n'est pas rempli, il faut renoncer à une fusion pour permettre de nouvelles simplifications.



Iterated Register Coalescing

Etape 1. **Simplify**: Eliminer les noeuds simplifiables à l'exclusion des noeuds reliés par un arc d'affinité. Chaque noeud eliminé est placé sur une pile.

- Le degré des noeuds diminue, ce qui créé des opportunités pour le coalesce conservatif (étape 2).

Etape 2. **Coalesce**: Fusionner les noeuds reliés par un arc d'affinité, quand le critère de George le permet. Itérer en 1.

- Cette étape peut réduire le degré de noeuds connexes aux noeuds fusionnés, créant de nouvelles opportunités pour la simplification, d'où l'itération.

Etape 3. **Freeze**: Si le critère de George n'est jamais vérifié, supprimer un arc d'affinité qui relie deux noeuds de faible degré. Itérer en 1.

- Certaines fusions sont structurellement impossibles. Par exemple, si x et y interférent et z est relié à x et à y par un arc d'affinité, on ne peut pas à la fois fusionner (x, z) et (y, z).

Iterated Register Coalescing

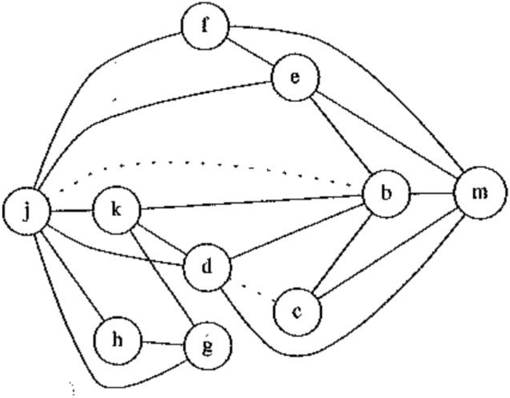
Etape 4. Spill: S'il reste encore des noeuds et que tous les arcs d'affinité ont étés supprimés, choisir et supprimer un noeud de fort degré pour le *spill*. Itérer en 1.

Etape 5. Select: Lorsque tous les noeuds ont étés supprimés, on les dépile pour construire l'allocation de registres.

- Si des noeuds ont étés sélectionnés pour être spillés, il faudrait reconstruire le graphe d'interférence (pourquoi?) et itérer en 1.
- Si la machine dispose de suffisament de registres, on peut éviter cette itération en réservant un registre pour le spill, et en allouant d'emblée pour $k - 1$ registres.

Exemple

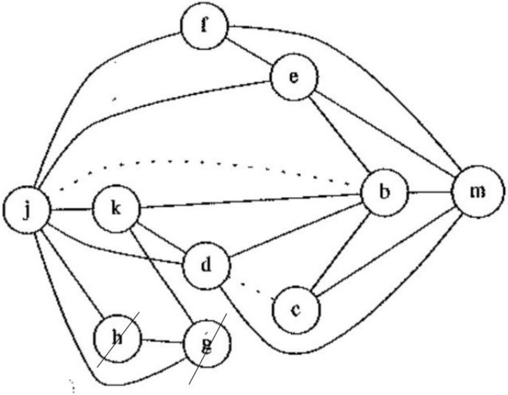
```
live-in: k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out: d k j
```



- Allocation pour $k = 3$ registres + 2 registres réservés pour le spill.

Etape 1: Simplify

live-in: k j
```
g := mem[j+12]
h := k - 1
f := g * h
e := mem[j+8]
m := mem[j+16]
b := mem[f]
c := e + 8
d := c
k := m + 4
j := b
```
live-out: d k j

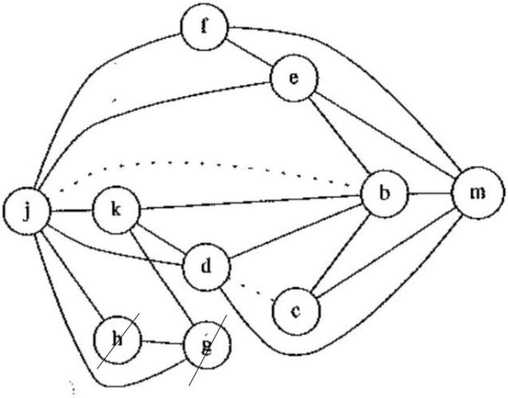


Simplify

Pile: h,g

Etape 2: Coalesce

```
live-in: k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out: d k j
```



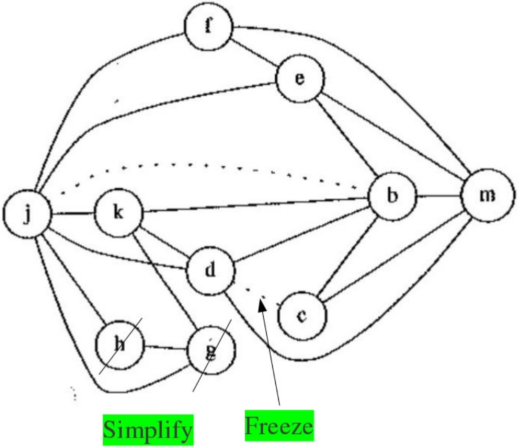
Simplify

Pile: h,g

- Critère de George non-vérifié

Etape 3: Freeze

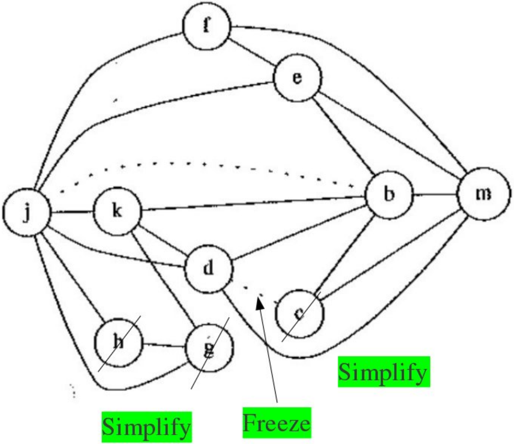
```
live-in: k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out: d k j
```



Pile: h,g

Etape 1: Simplify

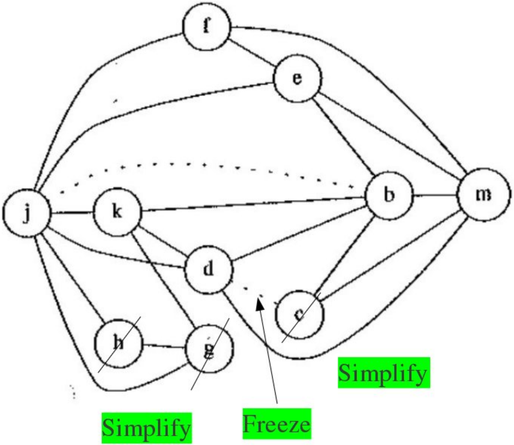
```
live-in: k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out: d k j
```



Pile: h,g,c

Etape 2: Coalesce

```
live-in:  k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out:  d k j
```

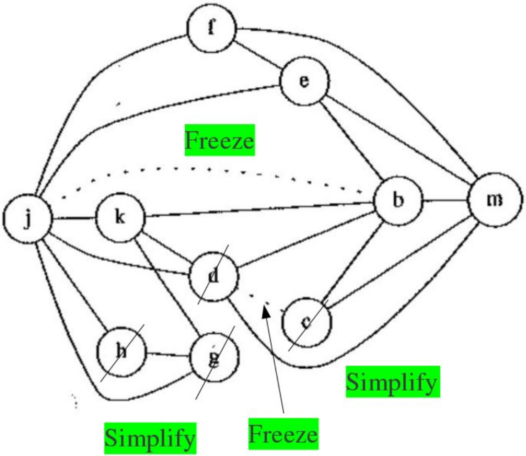


Pile: h,g

- Critère de George non-vérifié

Etape 3: Freeze

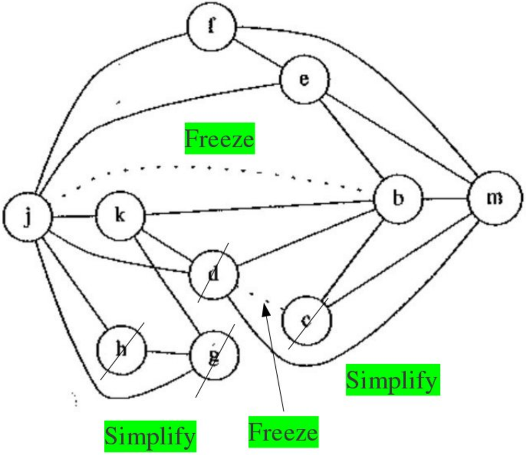
```
live-in: k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out: d k j
```



Pile: h,g

Etape 1: Simplify

```
live-in: k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out: d k j
```

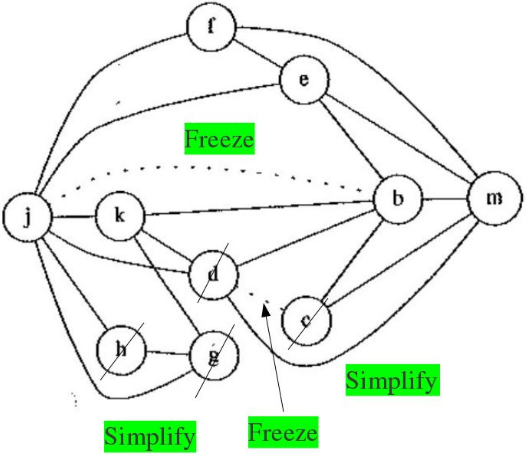


Pile: h,g,c

- Pas de simplication possible

Etape 2: Coalesce

```
live-in: k j
      g := mem[j+12]
      h := k - 1
      f := g * h
      e := mem[j+8]
      m := mem[j+16]
      b := mem[f]
      c := e + 8
      d := c
      k := m + 4
      j := b
live-out: d k j
```

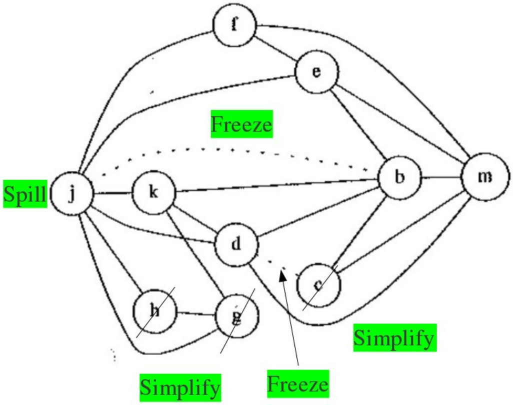


Pile: h,g,c

- Plus d'arc d'affinité

Etape 4: Spill

```
live-in: k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out: d k j
```

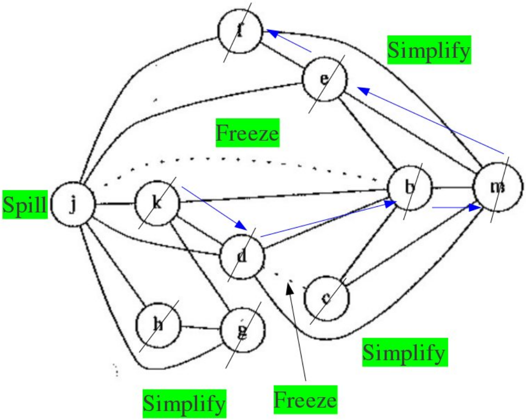


Pile: h,g,c,j

- Noeud de plus fort degré: j

Etape 1: Simplify

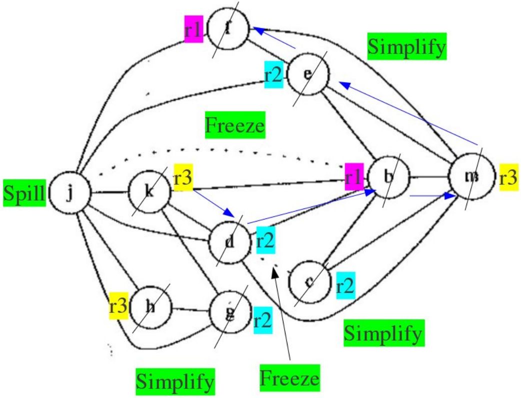
```
live-in: k j
        g := mem[j+12]
        h := k - 1
        f := g * h
        e := mem[j+8]
        m := mem[j+16]
        b := mem[f]
        c := e + 8
        d := c
        k := m + 4
        j := b
live-out: d k j
```



Pile: h,g,c,j,k,d,b,m,e,f

Etape 5: Select

```
live-in: k j
        g  := mem[j+12]
        h  := k - 1
        f  := g * h
        e  := mem[j+8]
        m  := mem[j+16]
        b  := mem[f]
        c  := e + 8
        d  := c
        k  := m + 4
        j  := b
live-out: d k j
```



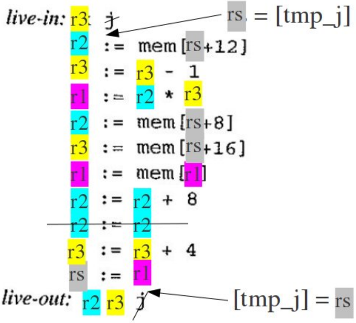
Pile: h,g,c,j, k,d,b,m,e,f

3,2,2,∅,3,2,1,3, 2,1

$\longleftarrow$

Code émis

```
live-in:  k  j
          g  :=  mem[j+12]
          h  :=  k  -  1
          f  :=  g  *  h
          e  :=  mem[j+8]
          m  :=  mem[j+16]
          b  :=  mem[f]
          c  :=  e  +  8
          d  :=  c
          k  :=  m  +  4
          j  :=  b
live-out:  d  k  j
```



- La copie d := c peut être supprimée.

Allocation des variables temporaires

Lorsqu'un temporaire est spillé, sa valeur est placé dans une variable temporaire sur la pile.

On a pas besoin de toutes les variables temporaires en même temps...

On peut optimiser l'espace des temporaires en suivant le même principe: construction d'un graphe d'interférence et coloriage.

- Bien sûr, il n'y a pas de limite sur k.
- Autrement, on pourrait spiller sur le disque...

Linear Scan

- On numérote les instructions du programme et 1 et N, et on considère le plus grand intervalle qui contient les intervalles de vie d'une variable (au pire $[1, N]$).
- Les intervalles obtenus sont triés par ordre croissant du point de départ.
- On parcourt les points d'entrée des intervalles par ordre croissant.
- On se maintient une liste d'intervalles de vie actifs. (triée par point de fin croissant)
- Pour chaque nouvel intervalle:
  - On retire les intervalles dont le point de fin expire (début de liste).
  - Le registre correspondant est alors disponible.
  - Si la liste est de taille k, on spille l'intervalle qui termine le plus tard (dernier de la liste).
  - Sinon, on attribue un registre non utilisé à l'intervalle.

Linear Scan

La complexité est linéaire en le nombre de variables.
➠adapté aux environnements de compilation “au vol”

D’après les auteurs, les performances sont 10% inferieures à IRC.

Il n’y a pas d’élimination de copies (coalesce).