Allocation de registres

Allocation des registres

Pablo Oliveira <pablo@sifflez.org>

UVSQ

Outline

Références

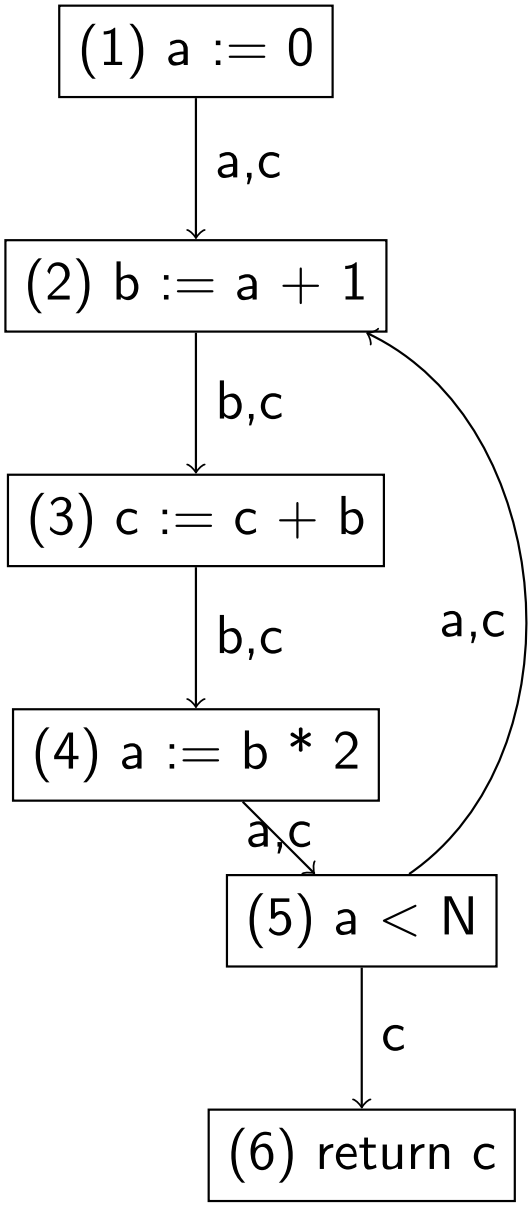
- Modern Compiler Implementation in Java/C/ML, A. Appel
- Cours de compilation de A.Demaille et R.Levillain.
- Cours de compilation de S.Tardieu
- Compilers Principles, Techniques and Tools, Aho, Lam, Sethi, Ullman

Exemple de fonction

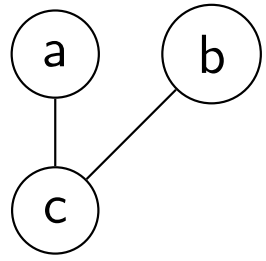
```
int f(int c) {
  int a,b;

  a = 0;
  while(a < N) {
    b=a+1;
    c=c+b;
    a=2*b;
  }
  return c;
}
```

- On pourrait mettre à jour les valeurs des variables sur la pile.
- Les registres sont plus rapides !
- Utiliser les registres comme variables temporaires.

Durée de vie



- c est vivant de l'entrée à la sortie de la fonction f.
- b est vivant entre : $2 \rightarrow 4$
- a est vivant entre : $1 \rightarrow 2$ et $4 \rightarrow 2$.
- Graphe d'interférence : deux noeuds sont connectés s'ils sont vivants en même temps.



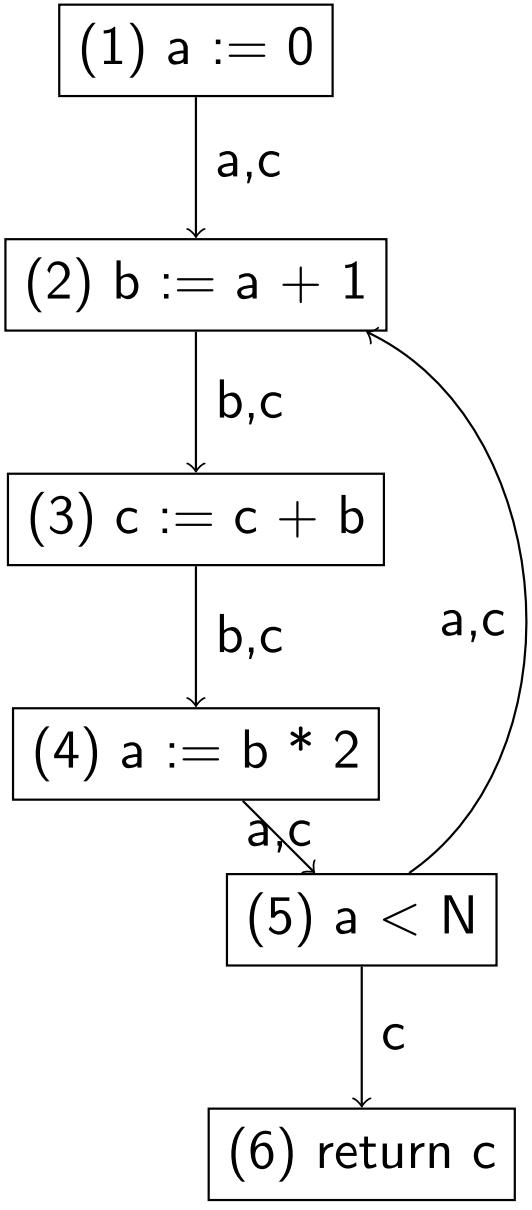
Equations de flots

$$in(N) = use(N) \cup (out(N) - def(N))$$

$$out(N) = \bigcup_{s \in succ(N)} in(S)$$

- Les variables utilisées dans un noeud sont vivantes en entrée
- Les variables vivantes en sortie d'un noeud sont soit vivantes en entrée, soit définies dans le noeud.
- Les variables vivantes en entrée d'un noeud sont nécessairement vivantes en sortie des noeud précédents.

Equations de flots : example



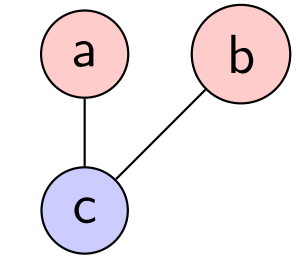
- $in(6) = use(6) = \{c\}$
- $out(5) = in(6) = \{c\}$
- $in(5) = (out(5) - def(5)) \cup use(5) = \{a, c\}$
- $out(4) = in(5) = \{a, c\}$
- $in(4) = (out(4) - def(4)) \cup use(4) = \{b, c\}$
- $out(3) = in(4) = \{b, c\}$
- $in(3) = (out(3) - def(3)) \cup use(3) = \{b, c\}$
- $out(2) = in(3) = \{b, c\}$
- $in(2) = (out(2) - def(2)) \cup use(2) = \{a, c\}$
- $out(1) = in(2) = \{a, c\}$
  $out(5) = in(2) = \{a, c\}$
- $in(5) = (out(5) - def(5)) \cup use(5) = \{a, c\}$
  Point Fixe !

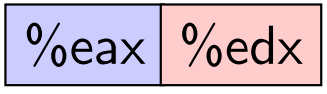
Discussion

- Pourquoi y a-t-il toujours un point fixe ? (au tableau)
- Proposition d'algorithme ?
  - anti parcours du graphe plus efficace. Pourquoi ?
- L'analyse de flots est un outil très puissant :
  - à la base de nombreuses optimisations
  - eg. propagation de constantes

Allocation de registres = Coloration de graphe

- Graphe d'interférence : deux noeuds connectés sont vivants en même temps.
- Règle de coloriage : les noeuds connectés doivent être affectés des registres différents.






```
    %edx = 0
L1:%edx = %edx + 1
    %eax = %eax + %edx
    %edx = %edx * 2
    if %edx < N goto L1
    return %eax
```

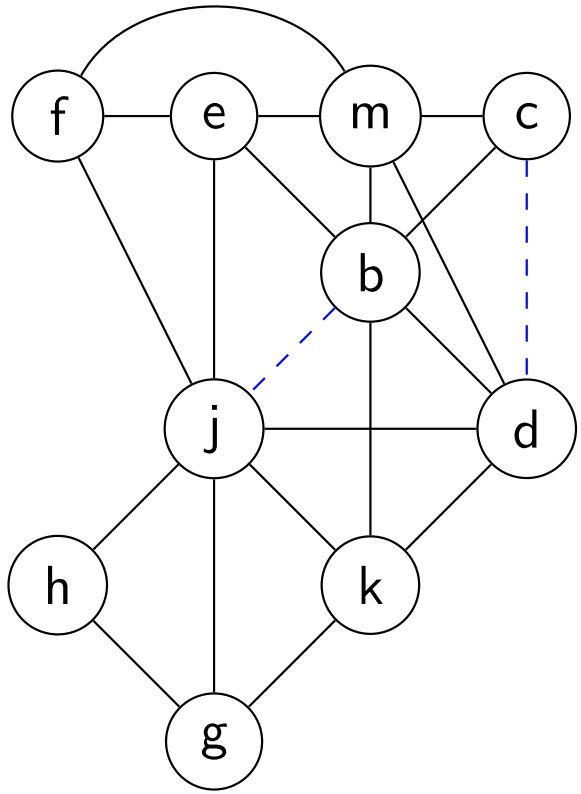
Coloriage de graphe

- Si on dispose de k registres physiques, le problème d'allocation de registres est équivalent au problème de coloriage du graphe d'interférence avec k couleurs.
- NP-complet.
- Mais heuristique performante : coloriage par simplification.

Graphe d'interférence

- On dispose de 4 registres physiques : r1, r2, r3, r4

```
in : k j
g = *(j+12)
h = k - 1
f = g * h
e = *(j+8)
m = *(j+16)
b = *(f)
c = e + 8
d = c
k = m + 4
j = b
out : d k j
```



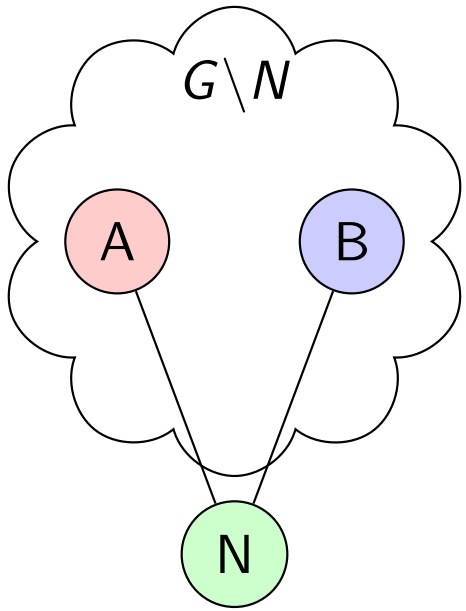
Les arcs en pointillées sont des copies.

Simplification

Lemma

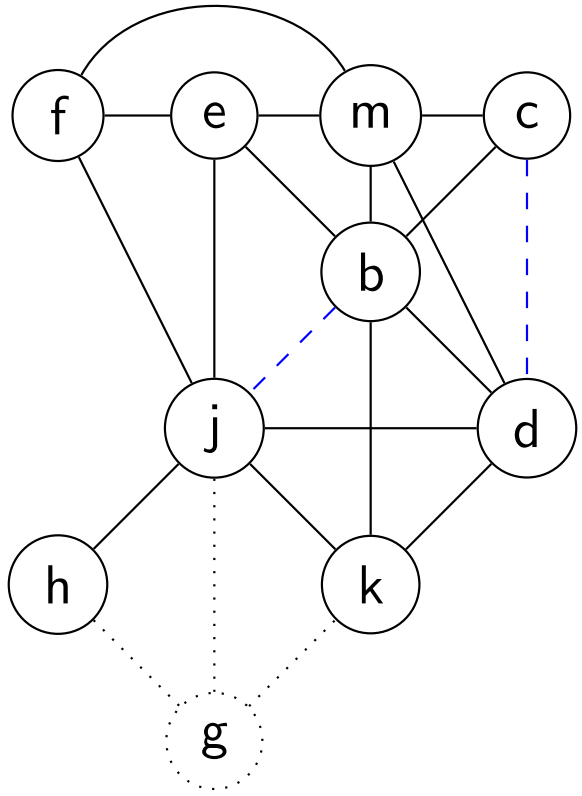
Soit un graphe G et un de ses noeuds N. On veut colorier G avec K couleurs. Si le degré de N est strictement inférieur à K, alors si on peut colorier $G \setminus N$, on peut colorier G.

Soit K=3.



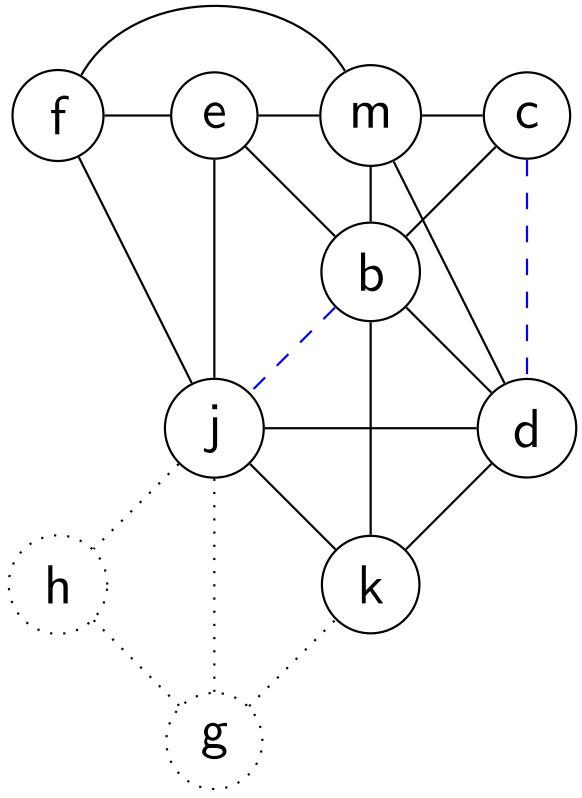
Simplification du graphe d'interférence

g



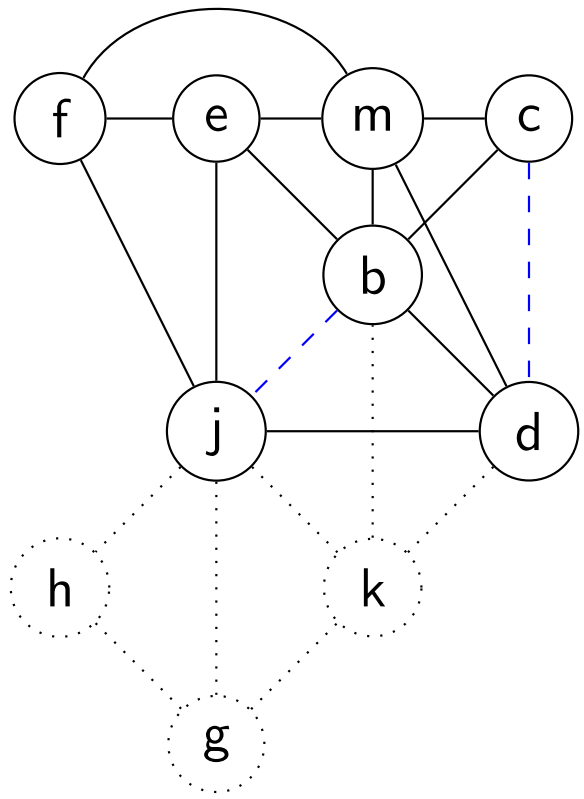
Simplification du graphe d'interférence

g h



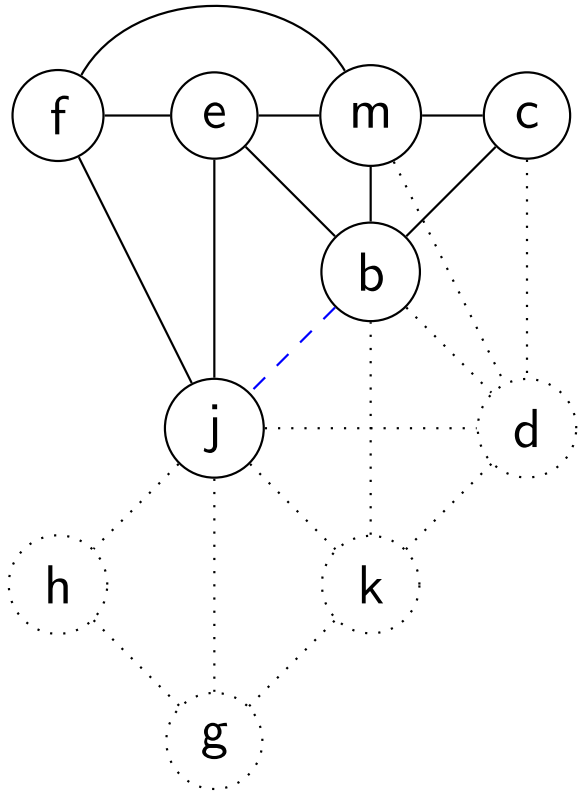
Simplification du graphe d'interférence

g h k



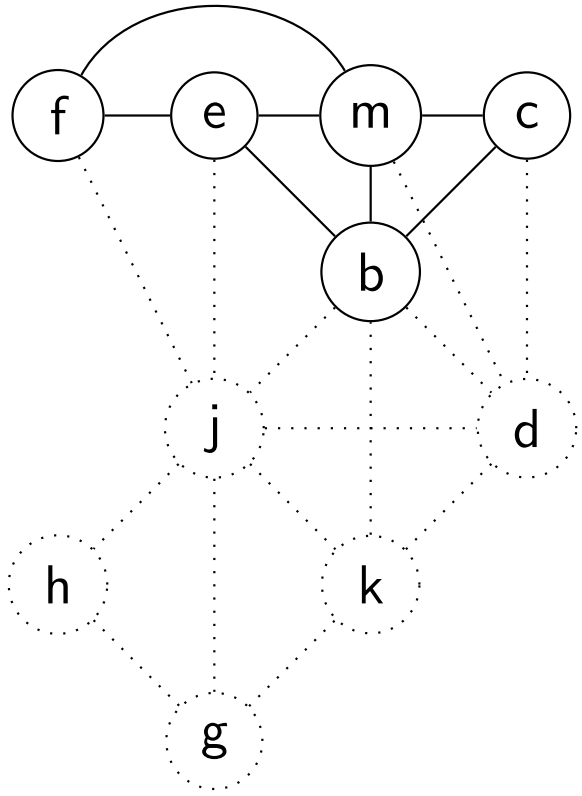
Simplification du graphe d'interférence

g h k d



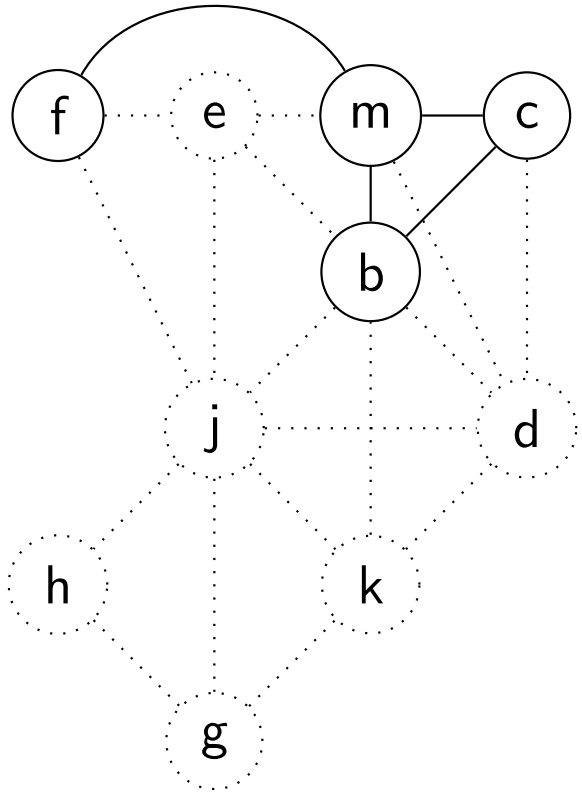
Simplification du graphe d'interférence

g h k d j



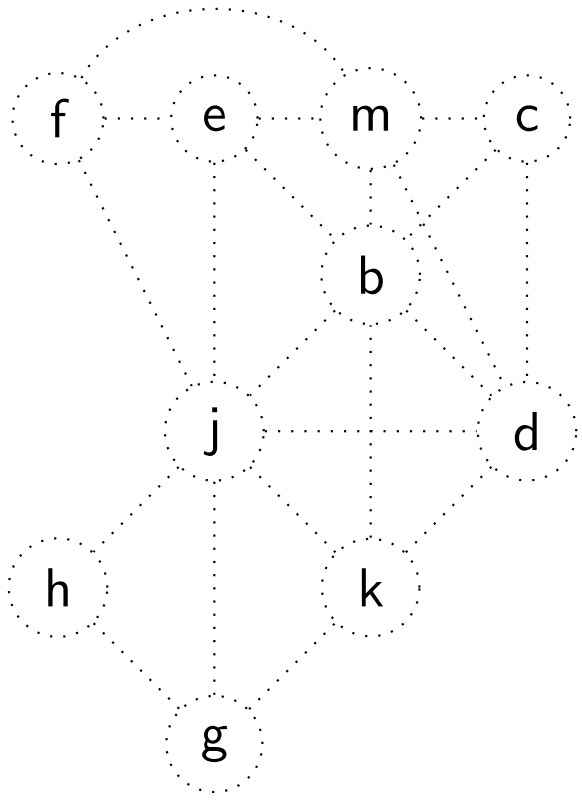
Simplification du graphe d'interférence

g h k d j e



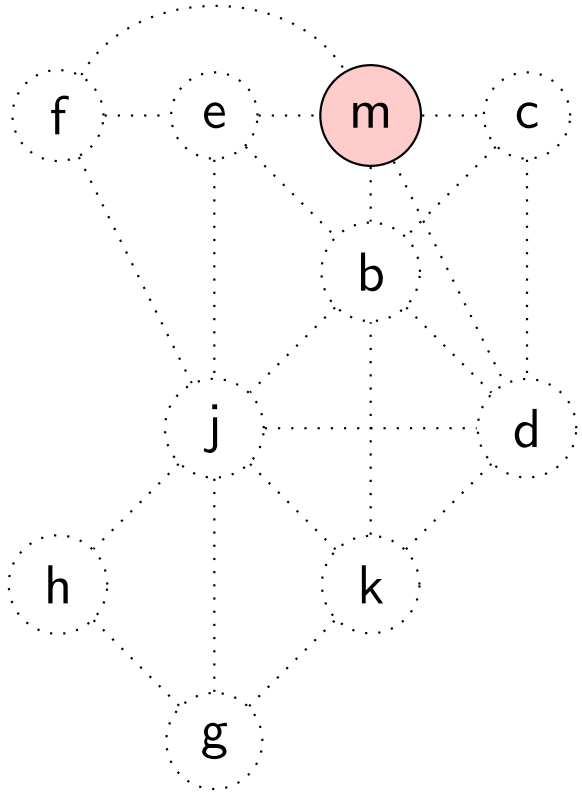
Simplification du graphe d'interférence : etc...

g h k d j e f b c m



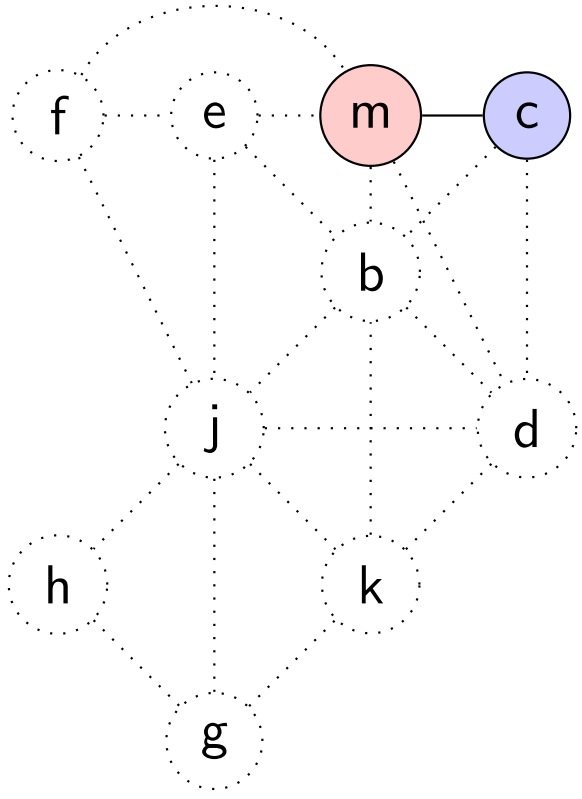
Coloriage

g h k d j e f b c m



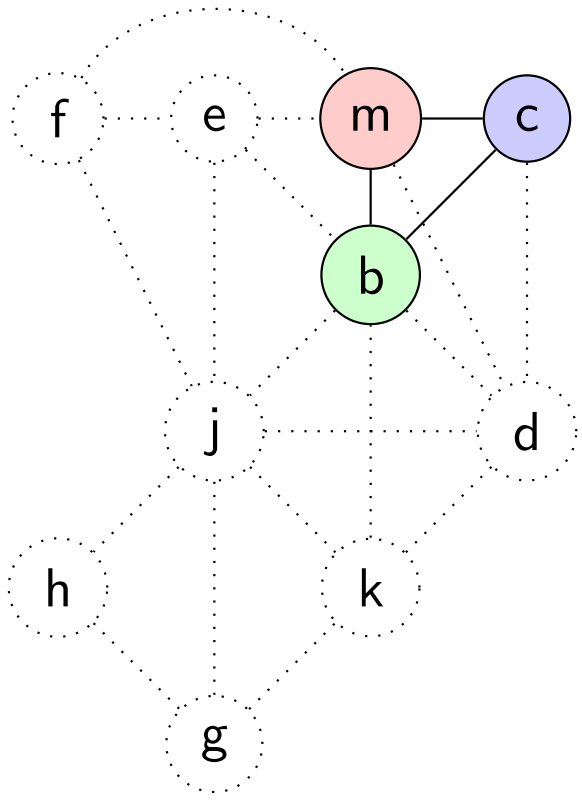
Coloriage

g h k d j e f b c



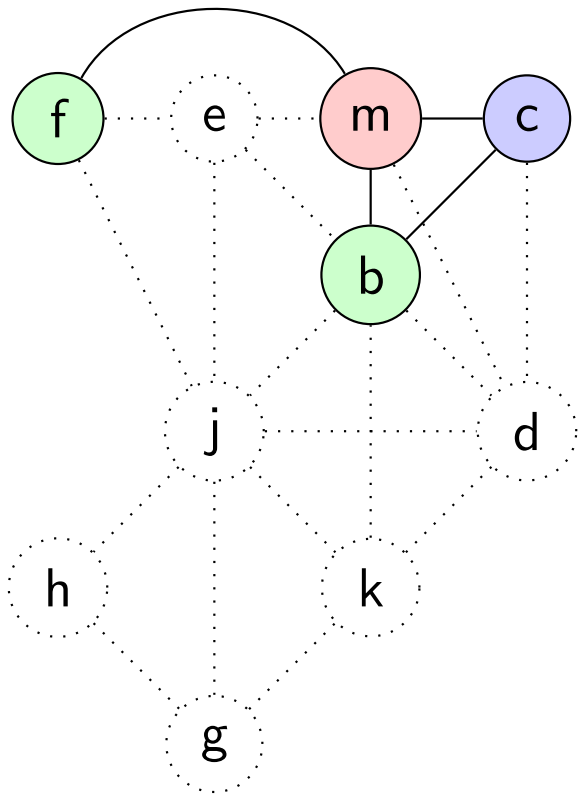
Coloriage

g h k d j e f b

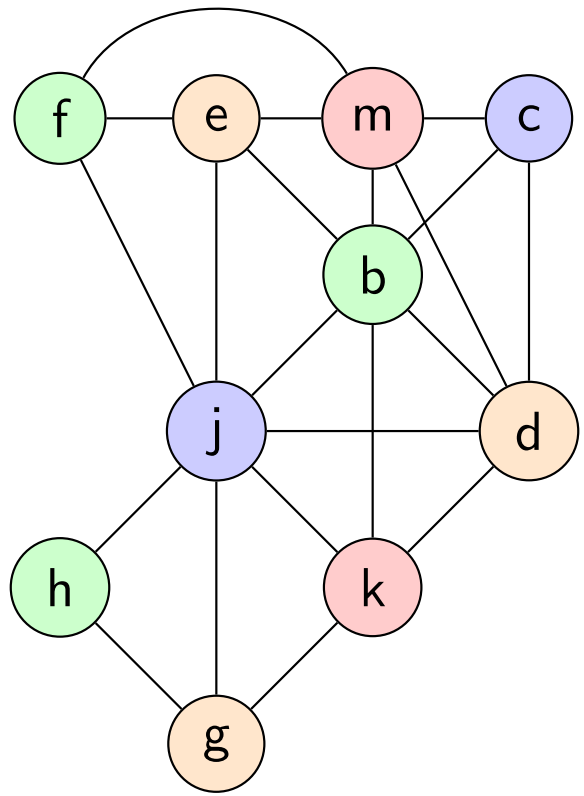


Coloriage

g h k d j e f



Coloriage : etc...



Allocation simple de registres

$\rightarrow$ Build $\rightarrow$ Simplify $\rightarrow$ Select

1. Build : On construit le graphe d'interférence
2. Simplify : On enlève un à un les noeuds de degré $< K$
3. Select : On colorie les noeuds en reconstruisant le graphe
   - On choisit une couleur différente de celle de nos voisins.

A.B. Kempe. On the Geographical problem of the four Colors.

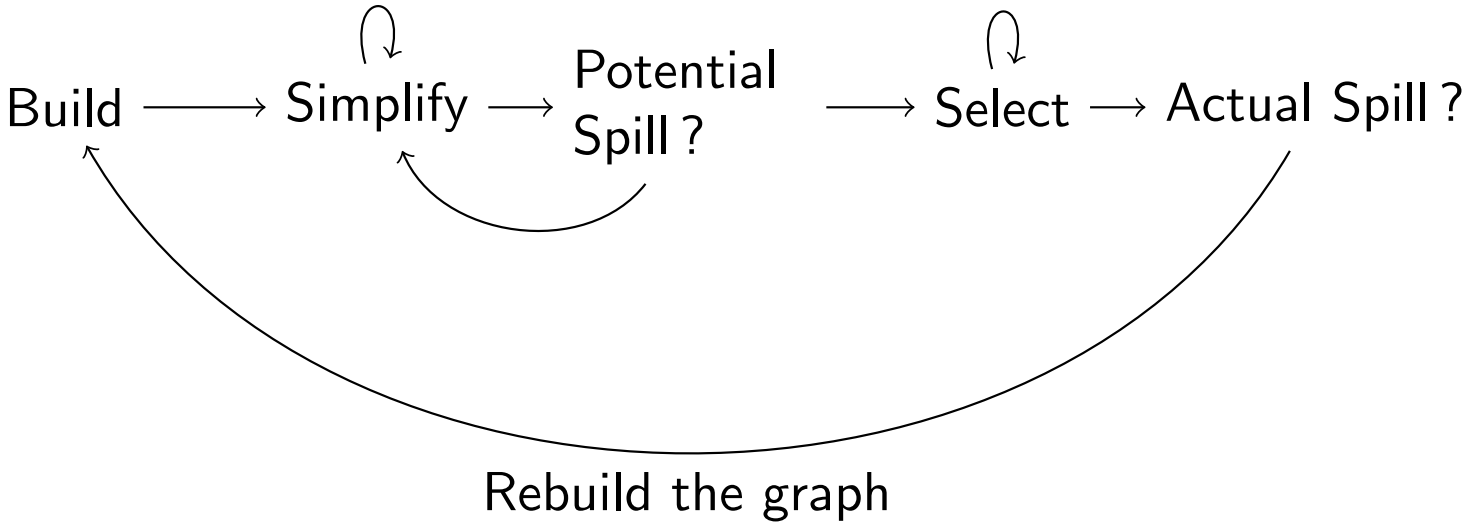
Problème

- L'heuristique ci-dessus peut ne pas fonctionner :
  - Lors de la phase Simplify tous les noeuds sont de degré $\geq K$.
- Solution : Spilling de registres

Spilling

- Pas assez de registres.
- Il faut diminuer le degré d'une variable a dans le graphe d'interférence.
- On alloue une case sur la pile.
- Chaque fois que l'on veut utiliser a, on le charge depuis la pile.
- Après utilisation on le sauve sur la pile.

Simplification avec Spilling



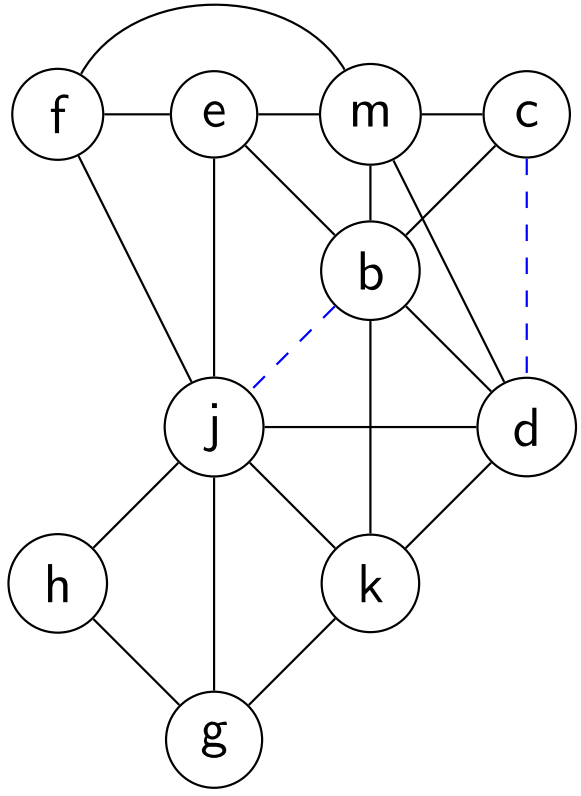
- Si un noeud de degré supérieur à K apparait lors de la simplification. On enregistre un spill possible.
- Parfois lors de la coloration on peut tout de même colorier les spill possibles. Si ce n'est pas le cas, on a un vrai spill.
- Dans ce cas on "spill" le registre sur la pile et on recommence.

Quel noeud faut il spiller ?

- Un noeud qui éliminera beaucoup d'arcs d'interférence.
- Un noeud utilisé peu fréquemment (le coût de passage par la pile est non négligeable). Éviter de spiller une variable d'induction dans une boucle !

Fusion de registres

```
in : k j
g = *(j+12)
h = k - 1
f = g * h
e = *(j+8)
m = *(j+16)
b = *(f)
c = e + 8
d = c
k = m + 4
j = b
out : d k j
```



Les arcs en pointillées sont des copies.

Fusion de registres

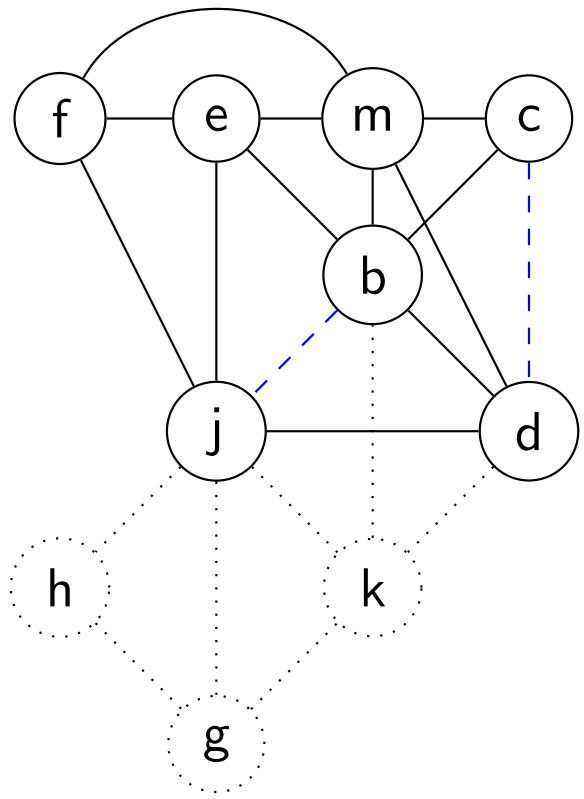
- Dans le graphe d'interférence on peut fusionner deux sommets :
  - ce sont des copies
  - ils n'interfèrent pas entre eux.
- Dans ce cas :
  - Les arcs du nouveau sommet sont l'union des arcs des anciens sommets.
  - On supprimme l'instruction de copie du programme initial.

Quand fusionner

- La fusion peut rendre le graphe non K-coloriable.
- Deux stratégies de fusion entre a et b :
  - Briggs : ab a moins de K voisins de degrée $\geq K$.
  - George : tous les voisins de a sont
    - de degré $< K$
    - ou déjà en interférence avec b

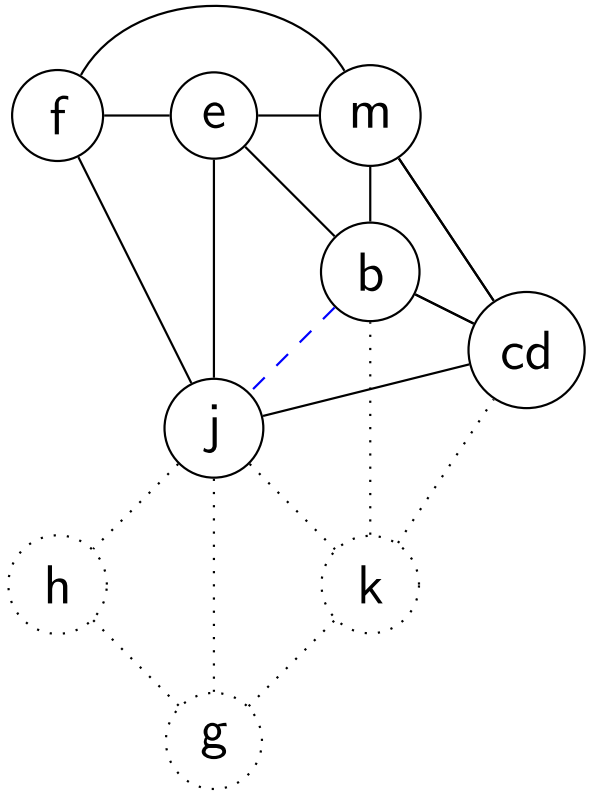
Simplification du graphe d'interférence

g h k



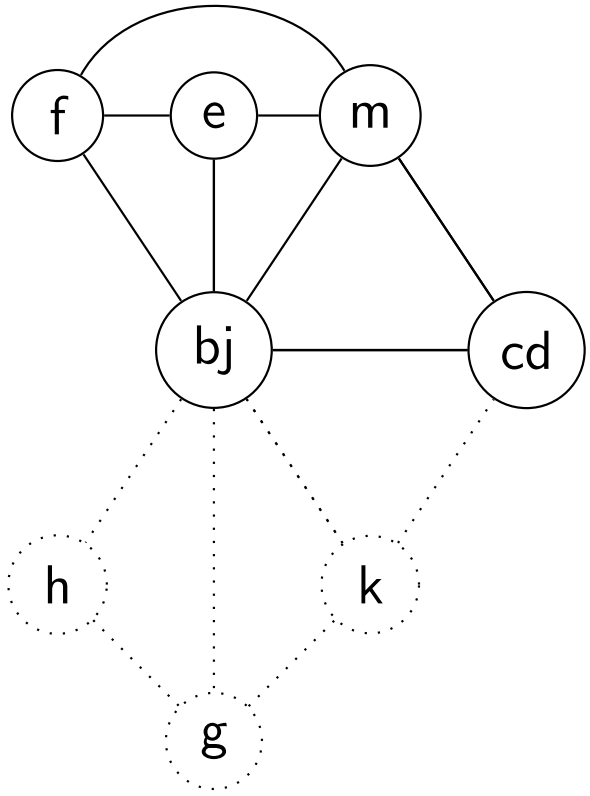
Simplification du graphe d'interférence

g h k c&d



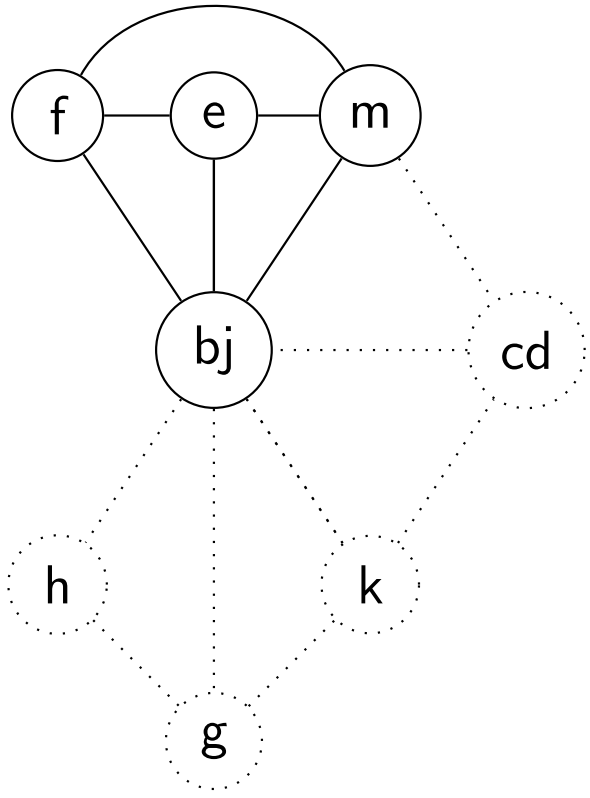
Simplification du graphe d'interférence

g h k c&d b&j

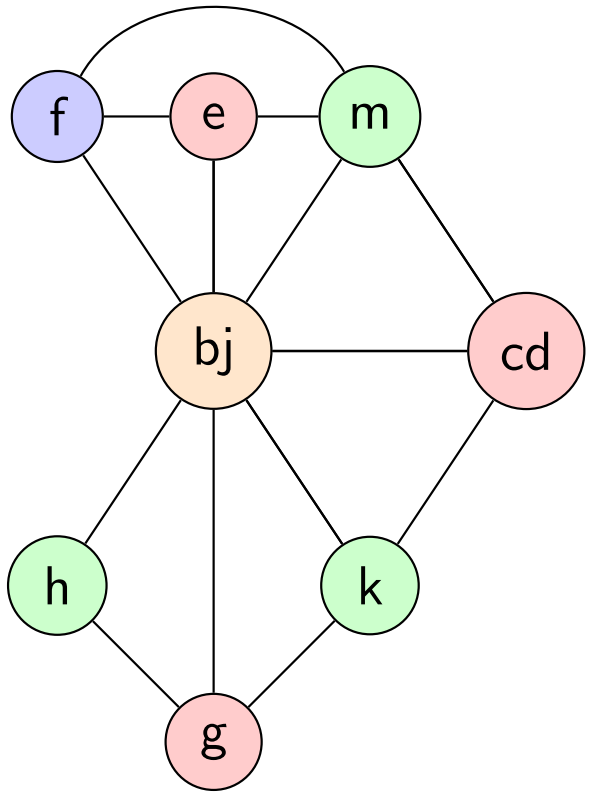


Simplification du graphe d'interférence

g h k c&d b&j cd



Simplification du graphe d'interférence : etc ...




```
in : r2 r4
r1 = *(r4 +12)
r2 = r2 - 1
r3 = r1 * r2
r1 = *(r4 +8)
r2 = *(r4 +16)
r4 = *r3
r1 = r1 + 8
# d = c
r2 = r2 + 4
# j = b
out : r1 r2 r4
```

On évite les deux copies.

Noeuds précoloriés

- Certains registres physiques sont utilisés pour des buts précis : retour d'une fonction, pointeur de pile, etc.
- Pour épargner ces registres on peut précolorier certains noeuds du graphe d'inférence.
- Un sommet précolorié ne sera jamais simplifié.

Registres et appels de fonctions

- Certains registres sont :
  - callee-save : à sauver par l'appelé
  - caller-save : à sauver par l'appelant
- Une instruction call redéfinit tous les caller-save
- On privilégiera :
  - les caller-save pour les variables temporaires éphémères.
    - pas besoin de faire un backup avant l'appel de fonction.
  - les callee-save pour les variables temporaires vivantes autour d'appels de sous programme.

Conclusion

- Allocation de registre = Coloriage de graphe
- Heuristique efficace : build / simplify / select
- De nombreuses optimisations peuvent être utilisés pour : éliminer les copies, préserver certains registres, éviter les spills, etc.