

Optimisation des boucles

Christophe Moulleron



1 Exemple du produit de matrices

- Permutation des boucles
- Autres optimisations

2 Cas général : nid de boucles

- Transformation de nids de boucles
- Graphe et vecteurs de dépendance
- Validité d'une transformation

Code (simplifié) :

```
for (i = 0; i < N; ++i)
  for (j = 0; j < N; ++j)
    for (k = 0; k < N; ++k)
      C[i][j] = C[i][j] + A[i][k] * B[k][j] ;
```

Objectif : Accélérer le calcul en exploitant au mieux le matériel

- 1 Exemple du produit de matrices
 - Permutation des boucles
 - Autres optimisations
- 2 Cas général : nid de boucles
 - Transformation de nids de boucles
 - Graphe et vecteurs de dépendance
 - Validité d'une transformation

Changer l'ordre des boucles

↪ voir `kij.c` et `kji.c`

Résultat :

- aucun changement sémantique ...
- ... mais gros écarts dans les temps d'exécution

Explication :

Changer l'ordre des boucles

↪ voir `kij.c` et `kji.c`

Résultat :

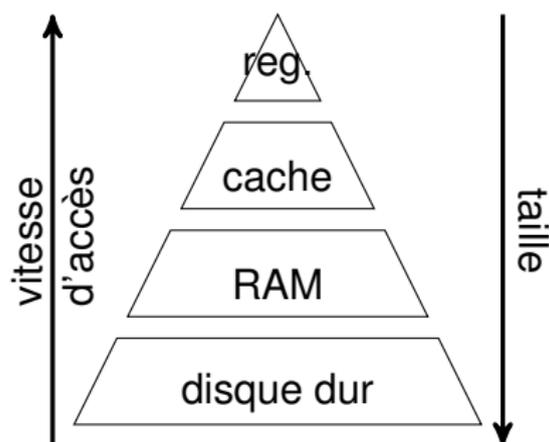
- aucun changement sémantique ...
- ... mais gros écarts dans les temps d'exécution

Explication :

- **coûts des accès mémoire** différents selon l'algo.

Hiérarchie mémoire

Coût d'accès à la mémoire selon l'endroit disposant de la donnée



Accès efficace si :

- **localité** = utilisation de données adjacentes
- **temporalité** = réutilisation d'une donnée récemment utilisée

Modèle mémoire simplifié

Hiérarchie mémoire :

- **niveaux** multiples
- **paramètres** dépendants de la machine

tailles des caches

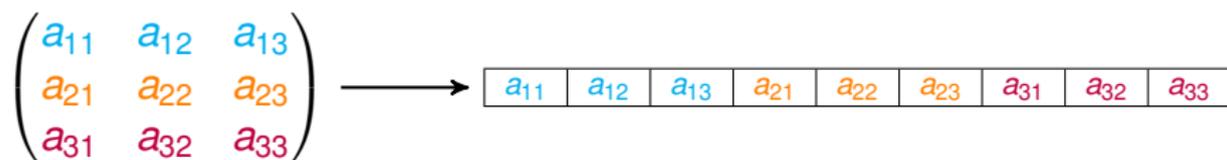
Modèle simplifié pour analyser les algos :

- 1 mémoire principale infinie
- 1 cache pouvant accueillir L lignes
 - ▶ 1 ligne = S octets consécutifs de la mémoire
 - ▶ taille totale = $L \times S$
- gestion optimale du cache (oracle)

Retour sur le produit de matrices

Expliquer les différences = comparer les nb. de défauts de cache

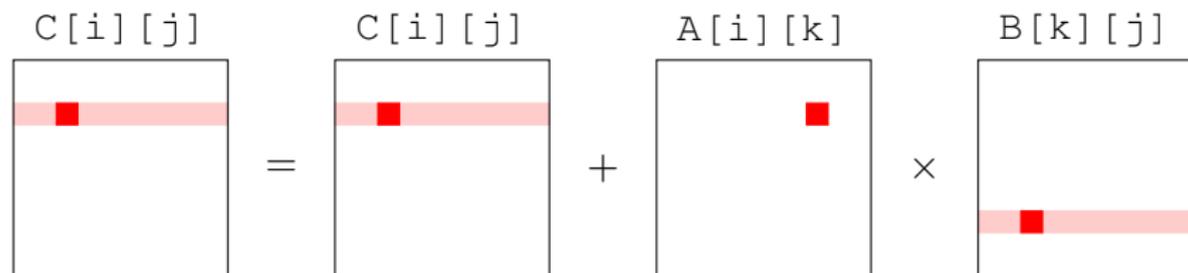
Stockage d'une matrice en C :



localité = passer de $M[i][j]$ à $M[i][j + 1]$
= traiter les matrices par lignes

Analyse de `kij.c`

Boucle interne :



→ 1 ligne de matrice = $\frac{N}{S}$ lignes de cache

Défauts de cache dans la boucle interne quand $N \gg S$:

pour A 0 ou 1
pour C et B $O\left(\frac{N}{S}\right)$

efficace

Boucle interne :



→ 1 colonne de matrice = $N \times$ (1 coef. dans 1 ligne de cache)

Défauts de cache dans la boucle interne quand $N \gg S$:

pour B 0 ou 1
pour A et C $O(N)$

inefficace

Analyse des différentes permutations

Bilan : parcours en ligne \Rightarrow moins de défauts de cache

Parcours des matrices suivant l'ordre des boucles :

	<i>A</i>	<i>B</i>	<i>C</i>
<i>ijk</i>			
<i>ikj</i>			
<i>jik</i>			
<i>jki</i>			
<i>kij</i>			
<i>kji</i>			

Analyse des différentes permutations

Bilan : parcours en ligne \Rightarrow moins de défauts de cache

Parcours des matrices suivant l'ordre des boucles :

	<i>A</i>	<i>B</i>	<i>C</i>
<i>ijk</i>	ligne	colonne	ligne
<i>ikj</i>	ligne	ligne	ligne
<i>jik</i>	ligne	colonne	colonne
<i>jki</i>	colonne	colonne	colonne
<i>kij</i>	colonne	ligne	ligne
<i>kji</i>	colonne	ligne	colonne

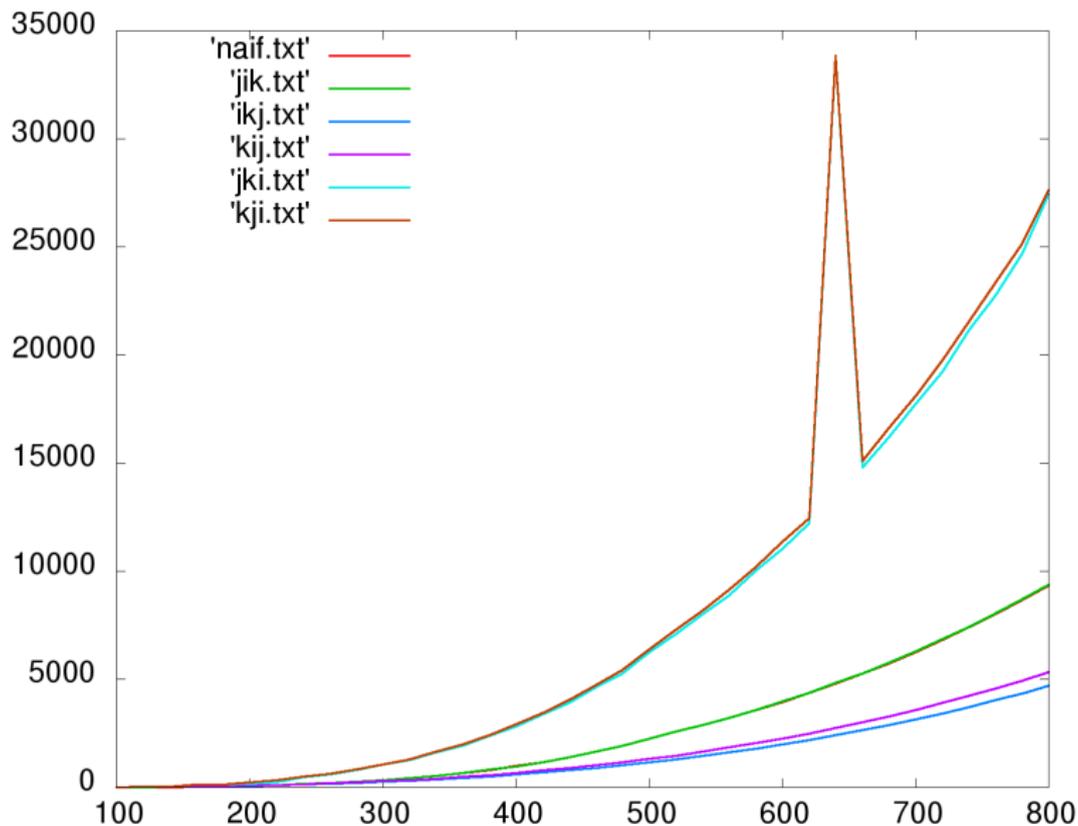
Analyse des différentes permutations

Bilan : parcours en ligne \Rightarrow moins de défauts de cache

Parcours des matrices suivant l'ordre des boucles :

gris = boucles externes

	<i>A</i>	<i>B</i>	<i>C</i>	
<i>ijk</i>	ligne	colonne	ligne	3 ^e
<i>ikj</i>	ligne	ligne	ligne	1 ^{er}
<i>jik</i>	ligne	colonne	colonne	4 ^e
<i>jki</i>	colonne	colonne	colonne	6 ^e
<i>kij</i>	colonne	ligne	ligne	2 ^e
<i>kji</i>	colonne	ligne	colonne	5 ^e



- 1 Exemple du produit de matrices
 - Permutation des boucles
 - **Autres optimisations**
- 2 Cas général : nid de boucles
 - Transformation de nids de boucles
 - Graphe et vecteurs de dépendance
 - Validité d'une transformation

Déroulage de la boucle interne

↪ voir `unroll.c`

Résultat :

- code plus gros ...
- ... mais léger gain de temps

voire pas

Explications :

Déroulage de la boucle interne

↪ voir `unroll.c`

Résultat :

- code plus gros ...
- ... mais léger gain de temps

voire pas

Explications :

- moins de sauts \Rightarrow pipeline mieux rempli
- corps de boucle plus gros \Rightarrow meilleur usage de l'ILP

cf cours 6

Utilisation de la temporalité grâce au *Blocking*

↪ voir `block.c`

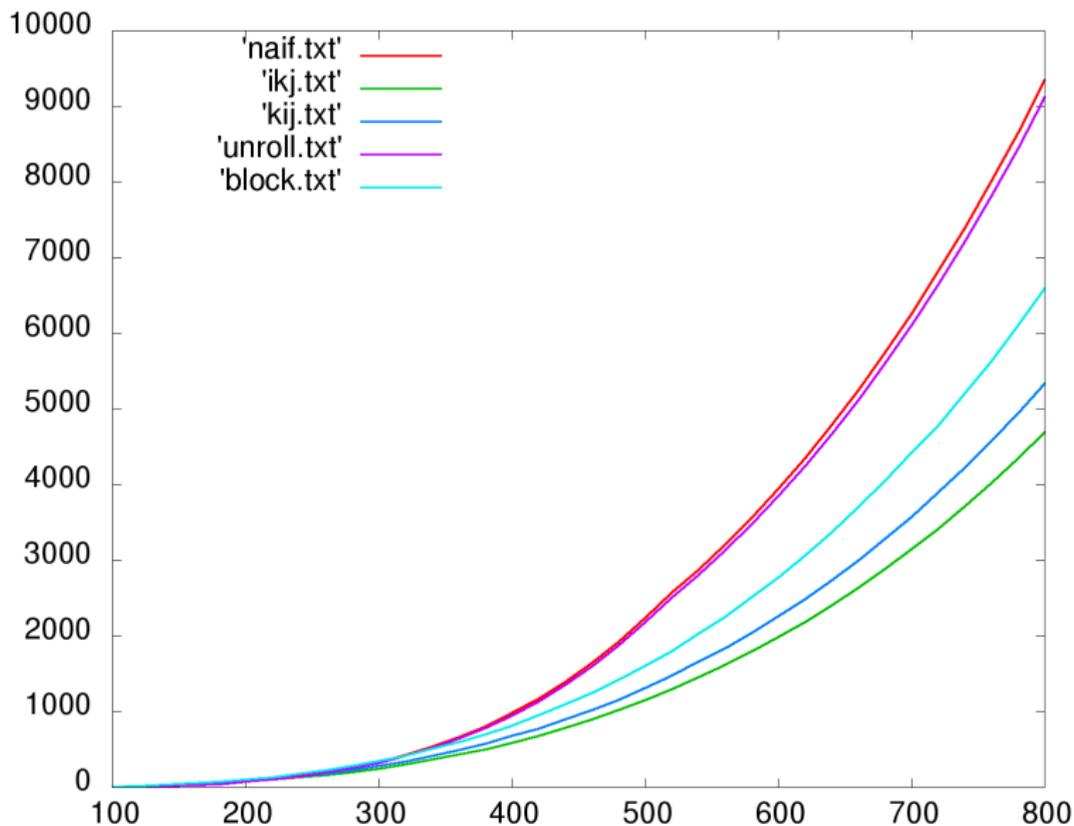
Idées :

- utiliser le meilleur ordre pour les boucles
- découper en blocs de taille \approx cache L2

Problème = algo. dépendant de la taille des caches

Solutions :

- générer le code en fonction de la machine ATLAS
- utiliser un algo. *cache-oblivious* algo. *Divide and Conquer*



Bilan sur le produit de matrices

Produit de matrice =

- calcul intensif
- grosses données en entrée

comparé aux caches

Permutation des boucles :

- pas de changement sémantiques
- différence d'**accès à la mémoire**
- écart en temps d'un **facteur 5**

Autres techniques :

- déroulage de boucle \Rightarrow meilleur ILP
- *blocking* = autre façon d'améliorer l'accès mémoire

1 Exemple du produit de matrices

- Permutation des boucles
- Autres optimisations

2 Cas général : nid de boucles

- Transformation de nids de boucles
- Graphe et vecteurs de dépendance
- Validité d'une transformation

Optimisation d'un nid de boucles

Nid de boucles = emboîtement de boucles / instructions simples

- pas des boucles constant
- nombre fixe d'itérations
- accès aux tableaux linéaires
- instructions = arithmétique + affectations

$A[2*i][N+j-i]$

Problème

Étant donné un code avec un nid de boucle, comment réorganiser les boucles pour augmenter les performances ?

- 1 Exemple du produit de matrices
 - Permutation des boucles
 - Autres optimisations
- 2 Cas général : nid de boucles
 - Transformation de nids de boucles
 - Graphe et vecteurs de dépendance
 - Validité d'une transformation

Opérations sur les boucles

Échange

```
for (i=0; i<N; ++i)
  for (j=0; j<N; ++j)
    B[i][j] = A[j][i];
```

→

```
for (j=0; j<N; ++j)
  for (i=0; i<N; ++i)
    B[i][j] = A[j][i];
```

$$\begin{pmatrix} i \\ j \end{pmatrix} \mapsto \begin{pmatrix} j \\ i \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix}$$

Opérations sur les boucles

Déroutage

```
for (i=0; i<N; ++i)  
  A[i] = A[i-1] + 1;
```

→

```
for (bi=0; bi<N; bi+=S)  
  for (k=bi; k<bi+S; ++k)  
    A[k] = A[k-1] + 1;
```

$$(i) \mapsto \begin{pmatrix} bi \\ k \end{pmatrix} = \begin{pmatrix} i \div S \\ i \bmod S \end{pmatrix}$$

Coalescing

```
for (bi=0; bi<N; bi+=S)  
  for (k=bi; k<bi+S; ++k)  
    A[k] = A[k-1] + 1;
```

→

```
for (i=0; i<N; ++i)  
  A[i] = A[i-1] + 1;
```

$$\begin{pmatrix} bi \\ k \end{pmatrix} \mapsto (i) = (S \ 1) \begin{pmatrix} bi \\ k \end{pmatrix}$$

Opérations sur les boucles

Fusion

```
for (i=0; i<N; ++i) {  
    for (j=0; j<N; ++j)  
        B[i] += A[i][j];  
    for (j=0; j<N; ++j)  
        C[i][j] = 0;  
}
```

→

```
for (i=0; i<N; ++i)  
    for (j=0; j<N; ++j) {  
        B[i] += A[i][j];  
        C[i][j] = 0;  
    }
```

Fission

```
for (i=0; i<N; ++i)  
    for (j=0; j<N; ++j) {  
        B[i] += A[i][j];  
        C[i][j] = 0;  
    }
```

→

```
for (i=0; i<N; ++i) {  
    for (j=0; j<N; ++j)  
        B[i] += A[i][j];  
    for (j=0; j<N; ++j)  
        C[i][j] = 0;  
}
```

Opérations sur les boucles

Décalage

```
for (i=0; i<N; ++i)  
  A[i+2] = A[i+1] + A[i];
```

→

```
for (j=2; j<N+2; ++j)  
  A[j] = A[j-1] + A[j-2];
```

$$(i) \mapsto (j) = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} i \\ 1 \end{pmatrix}$$

Inversion

```
for (i=0; i<=N; ++i)  
  C += A[i] * B[N-i];
```

→

```
for (j=N; j>=0; --j)  
  C += A[N-j] * B[j];
```

$$(i) \mapsto (j) = \begin{pmatrix} -1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} i \\ N \end{pmatrix}$$

Opérations sur les boucles

Transformation unimodulaire

$$\begin{pmatrix} i \\ j \end{pmatrix} \mapsto \begin{pmatrix} u \\ v \end{pmatrix} = U \begin{pmatrix} i \\ j \end{pmatrix}$$

avec $U \in \mathbb{Z}^{d \times d}$ telle que $\det U = \pm 1$

Exemple avec $U = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$:

```
for (i=0; i<N; ++i)
  for (j=0; j<N; ++j)
    C[i] = A[i] + B[i+j];
```

→

```
for (u=0; u<N; ++u)
  for (v=u; v<N+u; ++v)
    C[u] = A[u] + B[v];
```

- 1 Exemple du produit de matrices
 - Permutation des boucles
 - Autres optimisations
- 2 Cas général : nid de boucles
 - Transformation de nids de boucles
 - **Graphe et vecteurs de dépendance**
 - Validité d'une transformation

Validité d'une transformation (1)

```
for (i=1; i<N; ++i)
  for (j=0; j<N; ++j)
    A[i][j] = A[i-1][j];
```

?

→

```
for (j=0; j<N; ++j)
  for (i=1; i<N; ++i)
    A[i][j] = A[i-1][j];
```

Peut-on échanger les deux boucles ?

Validité d'une transformation (1)

```
for (i=1; i<N; ++i)
  for (j=0; j<N; ++j)
    A[i][j] = A[i-1][j];
```

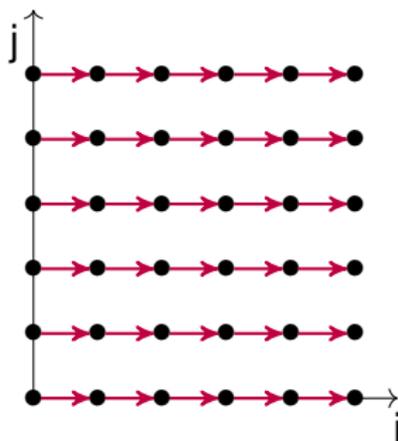
?

```
for (j=0; j<N; ++j)
  for (i=1; i<N; ++i)
    A[i][j] = A[i-1][j];
```

Peut-on échanger les deux boucles ?

oui

Graphe de dépendance :



Validité d'une transformation (2)

```
for (i=1; i<N; ++i)
  for (j=0; j<N-1; ++j)
    A[i][j] = A[i-1][j+1];
```

?

→

```
for (j=0; j<N-1; ++j)
  for (i=1; i<N; ++i)
    A[i][j] = A[i-1][j+1];
```

Peut-on échanger les deux boucles ?

Validité d'une transformation (2)

```
for (i=1; i<N; ++i)
  for (j=0; j<N-1; ++j)
    A[i][j] = A[i-1][j+1];
```

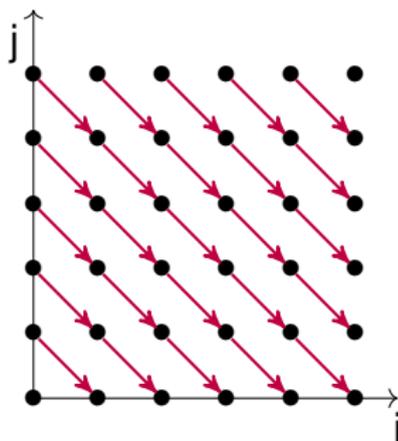
?

```
for (j=0; j<N-1; ++j)
  for (i=1; i<N; ++i)
    A[i][j] = A[i-1][j+1];
```

Peut-on échanger les deux boucles ?

non

Graphe de dépendance :



Dépendances dans un code

3 types de dépendances :

flot écriture puis lecture

$X = \dots$
$\dots = X$

anti lecture puis écriture

$\dots = X$
$X = \dots$

écriture écriture puis écriture

$X = \dots$
$X = \dots$

Dépendance quand :

- deux accès à la même case mémoire
- au moins une écriture

Vecteurs d'itération (1)

Problème : Étant donné une case mémoire, identifier les instants où on accède à cette case.

Cas de boucles imbriquées :

```
for (i=1; i<N; ++i)
  for (j=0; j<N-1; ++j)
    A[i][j] = A[i-1][j+1];
```

Accès à $A[u][v]$:

- en écriture quand $i = u$ et $j = v$
- en lecture quand $i - 1 = u$ et $j + 1 = v$

↪ deux **vecteurs d'itération** de la forme $\begin{pmatrix} i \\ j \end{pmatrix}$

$\begin{pmatrix} u \\ v \end{pmatrix}$ et $\begin{pmatrix} u + 1 \\ v - 1 \end{pmatrix}$

Vecteurs d'itération (2)

Cas d'un code séquentiel :

```
B[i]      = A[i][j];  
C[i]      = C[i] + B[i];  
A[i][j]   = B[i+1] + B[i-1];
```

Accès à $A[u][v]$:

- uniquement quand $u = i$ et $v = j$
- en lecture ligne $\ell = 1$
- en écriture ligne $\ell = 3$

↪ deux **vecteurs d'itération** pour $A[i][j]$ de la forme (ℓ) **(1)** et **(3)**

Vecteurs d'itération (3)

Exemple plus général :

```
for (i=1; i<N; ++i) {  
    for (j=1; j<N; ++j)  
        B[i][j] += A[i-1][j];  
    for (j=1; j<N; ++j)  
        A[i][j] = B[i][j-1];  
}
```

Accès à $A[u][v]$:

- en lecture dans la 1^{re} boucle interne quand $u = i - 1$ et $v = j$
- en écriture dans la 2^e boucle interne quand $u = i$ et $v = j$

↪ deux **vecteurs d'itération** de la forme $\begin{pmatrix} i \\ n \\ j \end{pmatrix}$ $\begin{pmatrix} u+1 \\ 1 \\ v \end{pmatrix}$ et $\begin{pmatrix} u \\ 2 \\ v \end{pmatrix}$

$n = \text{num. de la boucle interne}$

Vecteurs d'itération (4)

Cas général = ajouter dans l'ordre :

- une composante par niveau de boucle
- une composante pour compter les étapes d'un calcul séquentiel

Propriété

l a lieu avant l' si et seulement si $l \leq_{\text{lex}} l'$.

Vecteurs d'itération (4)

Cas général = ajouter dans l'ordre :

- une composante par niveau de boucle
- une composante pour compter les étapes d'un calcul séquentiel

Propriété

l a lieu avant l' si et seulement si $l \leq_{\text{lex}} l'$.

Remarques :

- taille du vecteur d'itération \leftrightarrow profondeur du code
aucun lien avec la dim. du tableau
- ajouter une composante fixée à 1 pour les décalages des indices
- ajouter une composante fixée à N pour les inversions de boucle

Vecteurs de dépendance

Idée :

- dépendances souvent simples
- besoin d'un modèle plus condensé que le graphe de dépendances

Définition

Pour chaque paire de vecteurs d'itération (I, I') telle que :

- $I = (i_1, \dots, i_d)$ et $I' = (i'_1, \dots, i'_d)$

- $I \leq_{\text{lex}} I'$

I a lieu avant I'

- il existe une dépendance entre I et I'

on définit un vecteur de dépendance par

$$\delta = (i'_1 - i_1, \dots, i'_d - i_d).$$

Exemple de vecteur de dépendance

```
for (i=1; i<N; ++i)
  for (j=0; j<N-1; ++j)
    A[i][j] = A[i-1][j+1];
```

Accès à $A[u][v]$:

- en écriture quand $i = u$ et $j = v$
- en lecture quand $i - 1 = u$ et $j + 1 = v$

$$I = \begin{pmatrix} u \\ v \end{pmatrix}$$
$$I' = \begin{pmatrix} u+1 \\ v-1 \end{pmatrix}$$

↪ Vecteur de dépendance : $\begin{pmatrix} u+1 \\ v-1 \end{pmatrix} - \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$

Même vecteur **pour tout** (u, v) .

- 1 Exemple du produit de matrices
 - Permutation des boucles
 - Autres optimisations
- 2 Cas général : nid de boucles
 - Transformation de nids de boucles
 - Graphe et vecteurs de dépendance
 - Validité d'une transformation

Critère de validité d'une transformation unimodulaire

On considère une transformation unimodulaire : $\begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix} \mapsto U \cdot \begin{pmatrix} i_1 \\ \vdots \\ i_d \end{pmatrix}$

Propriété

Une transformation unimodulaire de matrice U est valide si et seulement si, pour tout vecteur de dépendance δ , on a $U \cdot \delta \geq_{\text{lex}} 0$.

Preuve = si dépendance entre I et I' :

- vecteur de dépendance $\delta = I' - I$
- transformation = itération I effectuée à l'étape $U \cdot I$
- $U \cdot \delta \geq_{\text{lex}} 0 \Rightarrow U \cdot I' \geq_{\text{lex}} U \cdot I \Rightarrow$ dépendance respectée
- $U \cdot \delta <_{\text{lex}} 0 \Rightarrow U \cdot I' < U \cdot I \Rightarrow$ transfo. non valide

$$I \leq_{\text{lex}} I'$$

Retour sur les exemples d'échanges

```
for (i=1; i<N; ++i)
  for (j=0; j<N-1; ++j)
    A[i][j] = A[i-1][j+1];
```

vecteurs de dép. : $\left\{ \begin{pmatrix} 1 \\ -1 \end{pmatrix} \right\}$

Échange $\rightsquigarrow U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

\Rightarrow non valide

$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \not\leq_{\text{lex}} \begin{pmatrix} 0 \\ 0 \end{pmatrix}$

Retour sur les exemples d'échanges

```
for (i=1; i<N; ++i)
  for (j=0; j<N-1; ++j)
    A[i][j] = A[i-1][j+1];
```

Échange $\rightsquigarrow U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

\Rightarrow non valide

vecteurs de dép. : $\left\{ \begin{pmatrix} 1 \\ -1 \end{pmatrix} \right\}$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \not\leq_{\text{lex}} \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

```
for (i=1; i<N; ++i)
  for (j=0; j<N; ++j)
    A[i][j] = A[i-1][j];
```

Échange $\rightsquigarrow U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

vecteurs de dép. :

Retour sur les exemples d'échanges

```
for (i=1; i<N; ++i)
  for (j=0; j<N-1; ++j)
    A[i][j] = A[i-1][j+1];
```

Échange $\rightsquigarrow U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

\Rightarrow non valide

vecteurs de dép. : $\left\{ \begin{pmatrix} 1 \\ -1 \end{pmatrix} \right\}$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \not\geq_{\text{lex}} \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

```
for (i=1; i<N; ++i)
  for (j=0; j<N; ++j)
    A[i][j] = A[i-1][j];
```

Échange $\rightsquigarrow U = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

\Rightarrow valide

vecteurs de dép. : $\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right\}$

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \geq_{\text{lex}} \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Exemple de fusion de boucle

```
for (i=1; i<N; ++i) {  
  for (j=1; j<N; ++j)  
    B[i][j] += A[i-1][j];  
  for (j=1; j<N; ++j)  
    A[i][j] = B[i][j-1];  
}
```

?
→

```
for (i=1; i<N; ++i)  
  for (j=1; j<N; ++j) {  
    B[i][j] += A[i-1][j];  
    A[i][j] = B[i][j-1];  
  }
```

Vecteurs de dépendance ?

code de gauche

Exemple de fusion de boucle

```
for (i=1; i<N; ++i) {  
  for (j=1; j<N; ++j)  
    B[i][j] += A[i-1][j];  
  for (j=1; j<N; ++j)  
    A[i][j] = B[i][j-1];  
}
```

?

→

```
for (i=1; i<N; ++i)  
  for (j=1; j<N; ++j) {  
    B[i][j] += A[i-1][j];  
    A[i][j] = B[i][j-1];  
  }
```

Vecteurs de dépendance ?

code de gauche

• pour $A[u][v]$: $\delta_A = \begin{pmatrix} u+1 \\ 1 \\ v \end{pmatrix} - \begin{pmatrix} u \\ 2 \\ v \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$

• pour $B[u][v]$: $\delta_B = \begin{pmatrix} u \\ 2 \\ v+1 \end{pmatrix} - \begin{pmatrix} u \\ 1 \\ v \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$

Matrice de transfo. = $U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

$$U \cdot \delta_A \geq_{lex} 0 \quad U \cdot \delta_B \geq_{lex} 0$$

⇒ fusion **valide**

Exemple de fission de boucle

```
for (i=1; i<N; ++i) {  
  for (j=1; j<N; ++j)  
    B[i][j] += A[i-1][j];  
  for (j=1; j<N; ++j)  
    A[i][j] = B[i][j-1];  
}
```

?

→

```
for (i=1; i<N; ++i)  
  for (j=1; j<N; ++j)  
    B[i][j] += A[i-1][j];  
for (i=1; i<N; ++i)  
  for (j=1; j<N; ++j)  
    A[i][j] = B[i][j-1];
```

Vecteurs de dépendance : $\delta_A = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$ $\delta_B = \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix}$

Matrice de transfo. = $U = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ $U \cdot \delta_A = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \not\equiv_{lex} 0$

↪ fission **non valide**

Exemple d'inversion de boucle

```
for (i=1; i<=N-1; ++i)
  for (j=1; j<=N-1; ++j)
    A[i-1][N-j] = A[i][N-j-1];
```

?

```
for (i=1; i<=N-1; ++i)
  for (j=1; j<=N-1; ++j)
    A[i-1][j] = A[i][j-1];
```

Exemple d'inversion de boucle

```
for (i=1; i<=N-1; ++i)
  for (j=1; j<=N-1; ++j)
    A[i-1][N-j] = A[i][N-j-1];
```

?

```
for (i=1; i<=N-1; ++i)
  for (j=1; j<=N-1; ++j)
    A[i-1][j] = A[i][j-1];
```

Accès à $A[u][v]$?

- en écriture si $u = i - 1$ et $v = N - j$
- en lecture si $u = i$ et $v = N - j - 1$

$$\delta_A = \binom{u+1}{N-v} - \binom{u}{N-v-1} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

$$\text{Matrice de transfo.} = U = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$U \cdot \delta_A = \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix} \geq_{lex} 0$$

↪ inversion **valide**

Conclusion sur l'optimisation d'un nid de boucles

- 1 Déterminer des transformations qui améliorent :
 - ▶ la localité
 - ▶ l'ordre d'accès aux coefficients d'une matrice
 - ▶ le parallélisme
- 2 Trier ces transformations :
 - ▶ garder uniquement les transfo. valides
 - ▶ privilégier les transfo. améliorant plusieurs critères
- 3 Comparer expérimentalement :
 - ▶ hiérarchie mémoire complexe
 - ▶ optimisations parfois mutuellement exclusives