Tuilage paramétré avec réutilisation inter-tuiles

Alexandre Isoard **Alain Darte**

Compsys, LIP (Laboratoire de l'Informatique du Parallélisme), Lyon

7èmes journées compilation
Dammarie les Lys
4 décembre 2013

Plan de l'exposé

Déportation de noyau (Kernel Offloading)

Un noyau de calcul :

- Intensif en calcul.
- Beaucoup de données.

À déporter vers un accélérateur (FPGA, GPU, carte dédiée, multi-cœur) :

- Grosse puissance de calcul.
- Bonne bande passante en local.
- Faible latence en local.

Avec des contraintes de ressources :

- Petite mémoire locale.
- Faible débit global $\leftrightarrow$ local.
- Grosse latence global $\leftrightarrow$ local.

Déportation de noyau (Kernel Offloading)

Un noyau de calcul :

- Intensif en calcul.
- Beaucoup de données.

À déporter vers un accélérateur (FPGA, GPU, carte dédiée, multi-cœur) :

- Grosse puissance de calcul.
- Bonne bande passante en local.
- Faible latence en local.

Avec des contraintes de ressources :

- Petite mémoire locale. ☛ calculs par blocs
- Faible débit global $\leftrightarrow$ local. ☛ réutilisation des données
- Grosse latence global $\leftrightarrow$ local. ☛ pipelining/prefetching

Restrictions dues au polyédrique

Restrictions :

- Séquences et imbrications de `if` et boucles `for`.
- Tableaux (multidim.) de données directes (pas de pointeurs).
- Contrôle statique et affine :
  - Conditions des `if` : affines.
  - Bornes des boucles `for` : affines.
  - Incréments des boucles `for` : constants.
  - Fonctions d'accès aux tableaux : affines.

☛ Possibilité d'extensions par approximations.

Exemples :

- Polyédriques : algèbre linéaire dense, Cholesky, stencils, ...
- Non polyédriques : FFT, algèbre linéaire creux, ...

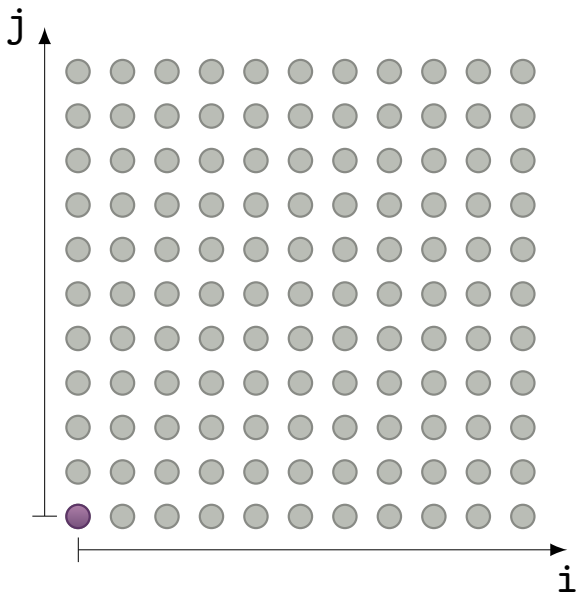
Lectures, écritures, dépendances, ordonnancement

```
for(int k=0; k<2*n-1; k++)
S0: C[k] = 0;
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
S1:     C[i+j] += A[i]*B[j];
```

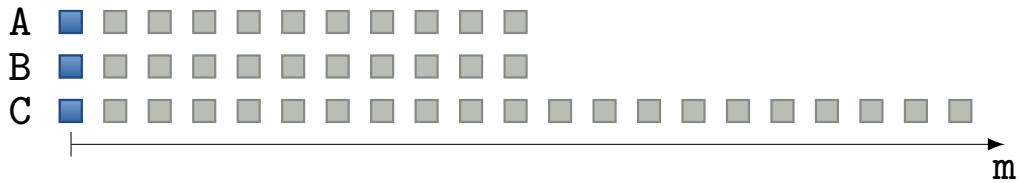
Produit de deux polynômes :

- coefficients dans A et B ;
- résultat dans C.

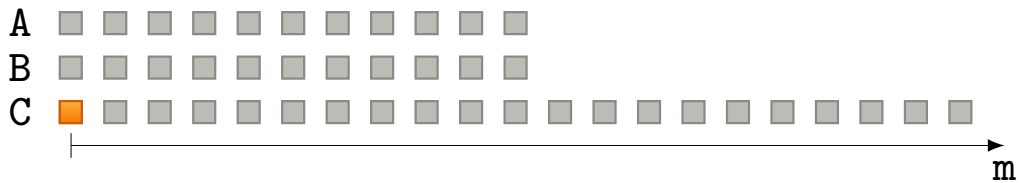
Schedule :



Read :



Write :



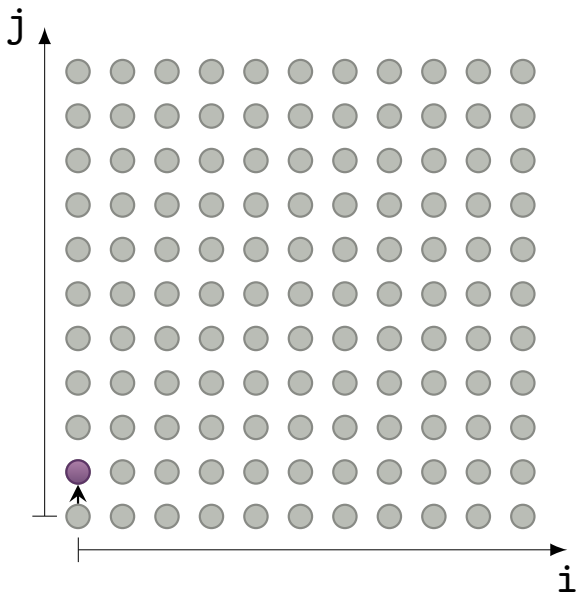
Lectures, écritures, dépendances, ordonnancement

```
for(int k=0; k<2*n-1; k++)
S0: C[k] = 0;
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
S1:     C[i+j] += A[i]*B[j];
```

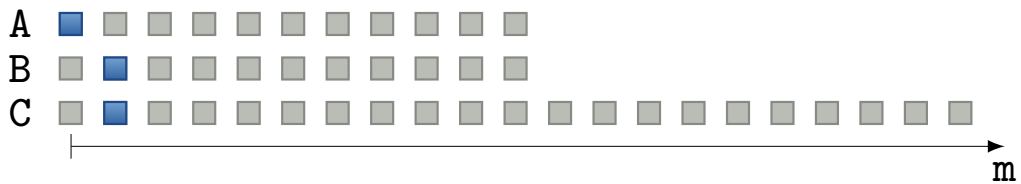
Produit de deux polynômes :

- coefficients dans A et B ;
- résultat dans C.

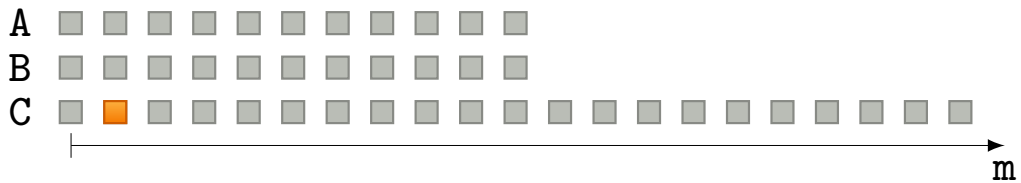
Schedule :



Read :



Write :



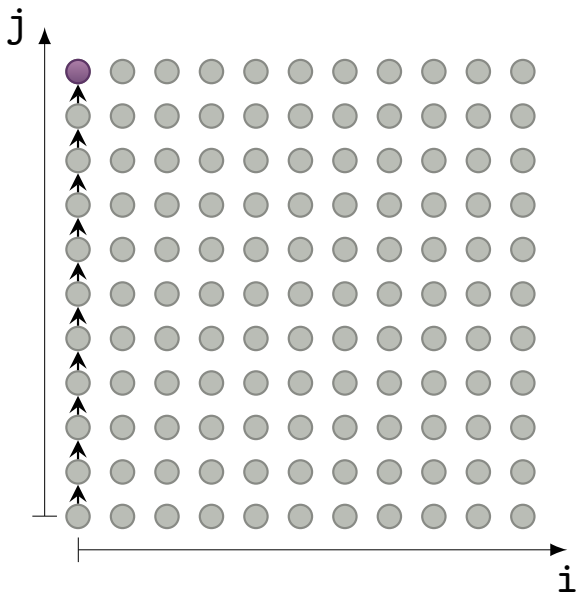
Lectures, écritures, dépendances, ordonnancement

```
for(int k=0; k<2*n-1; k++)
S0: C[k] = 0;
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
S1:     C[i+j] += A[i]*B[j];
```

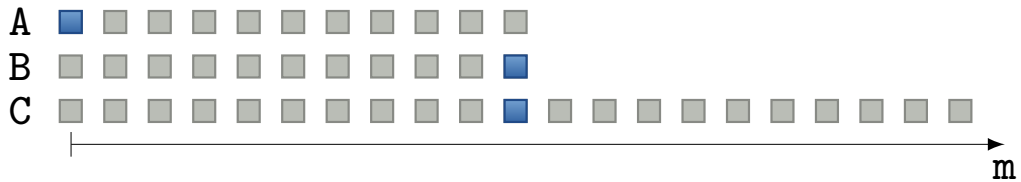
Produit de deux polynômes :

- coefficients dans A et B ;
- résultat dans C.

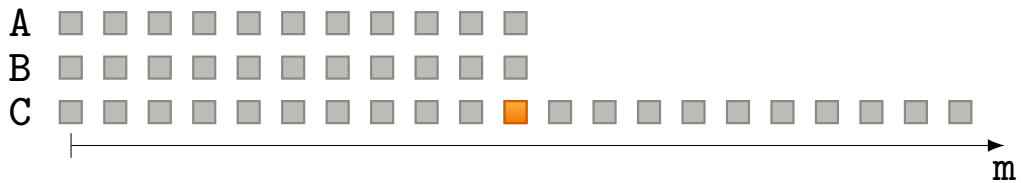
Schedule :



Read :



Write :



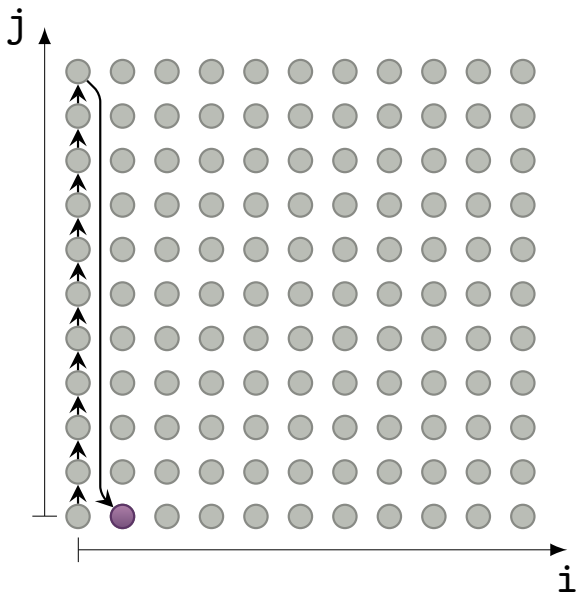
Lectures, écritures, dépendances, ordonnancement

```
for(int k=0; k<2*n-1; k++)
S0: C[k] = 0;
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
S1:     C[i+j] += A[i]*B[j];
```

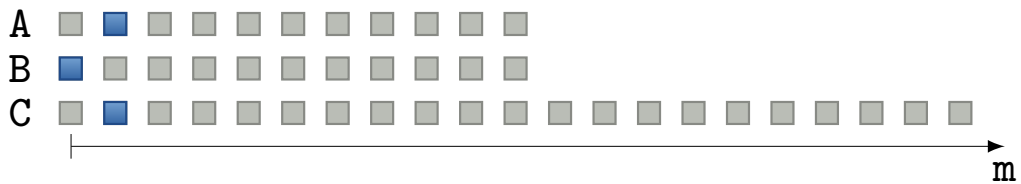
Produit de deux polynômes :

- coefficients dans A et B ;
- résultat dans C.

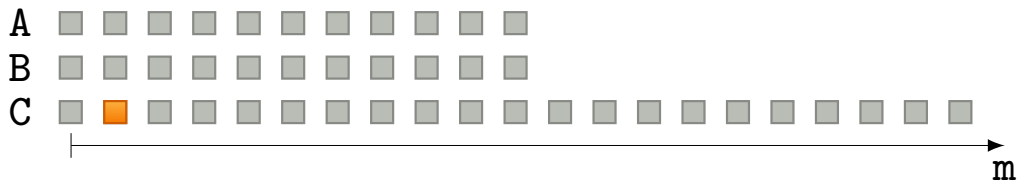
Schedule :



Read :



Write :



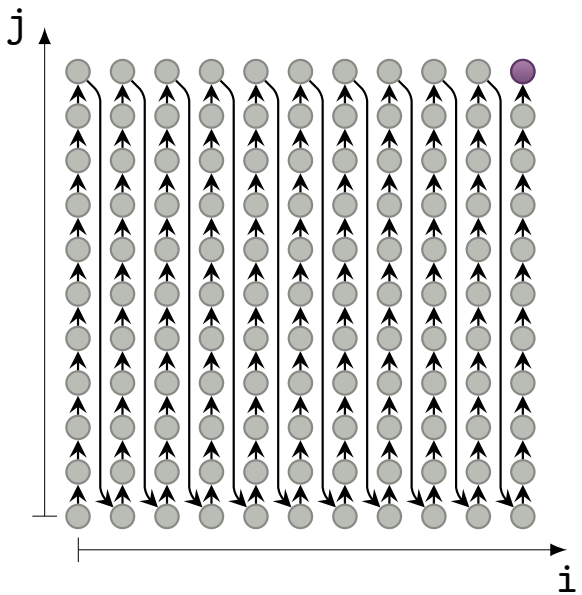
Lectures, écritures, dépendances, ordonnancement

```
for(int k=0; k<2*n-1; k++)
S0: C[k] = 0;
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
S1:     C[i+j] += A[i]*B[j];
```

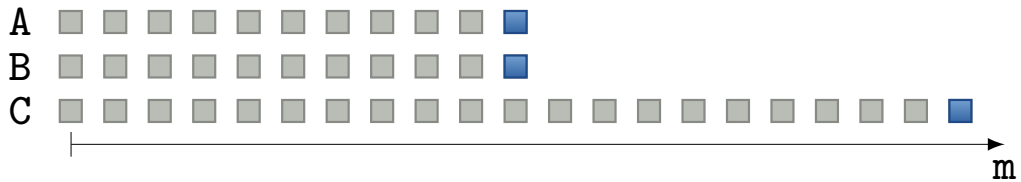
Produit de deux polynômes :

- coefficients dans A et B ;
- résultat dans C.

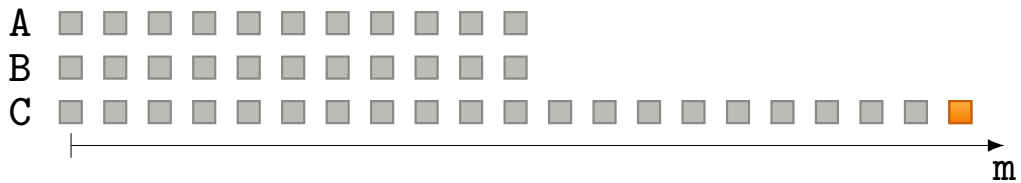
Schedule :



Read :



Write :



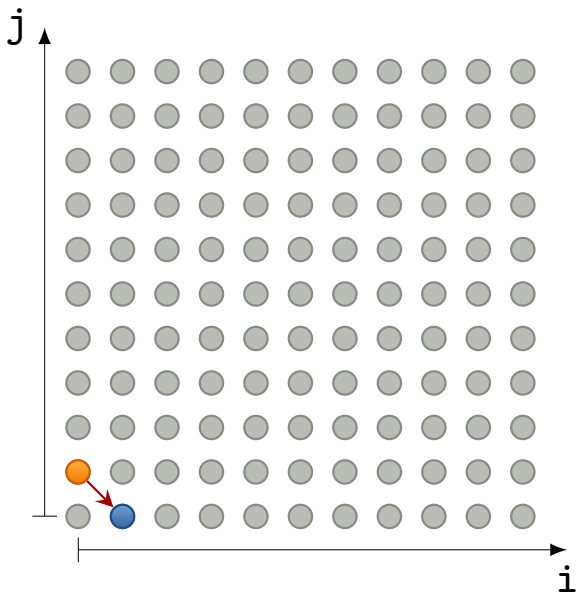
Lectures, écritures, dépendances, ordonnancement

```
for(int k=0; k<2*n-1; k++)
S0: C[k] = 0;
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
S1:     C[i+j] += A[i]*B[j];
```

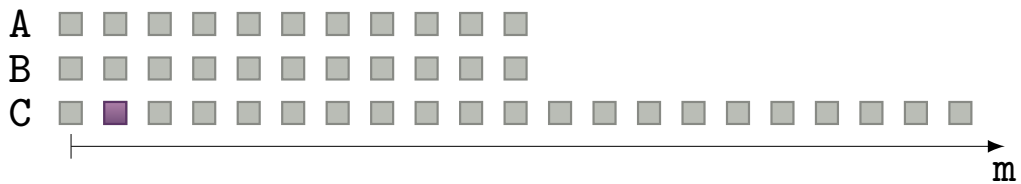
Produit de deux polynômes :

- coefficients dans A et B ;
- résultat dans C.

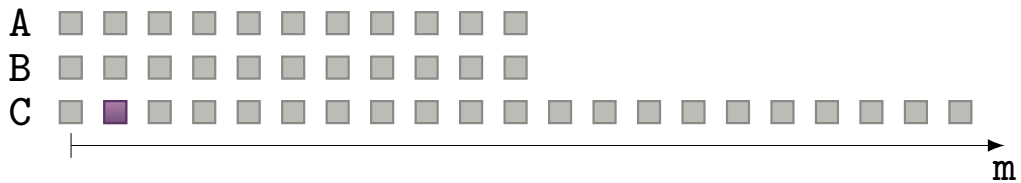
Schedule :



Read :



Write :



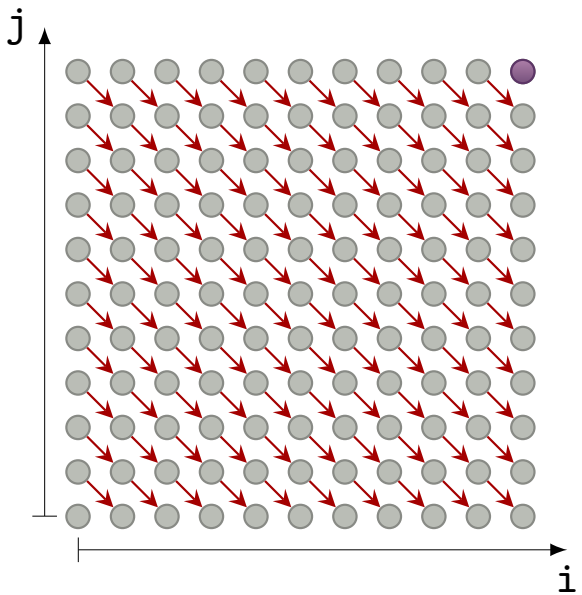
Lectures, écritures, dépendances, ordonnancement

```
for(int k=0; k<2*n-1; k++)
S0: C[k] = 0;
for(int i=0; i<n; i++)
    for(int j=0; j<n; j++)
S1:     C[i+j] += A[i]*B[j];
```

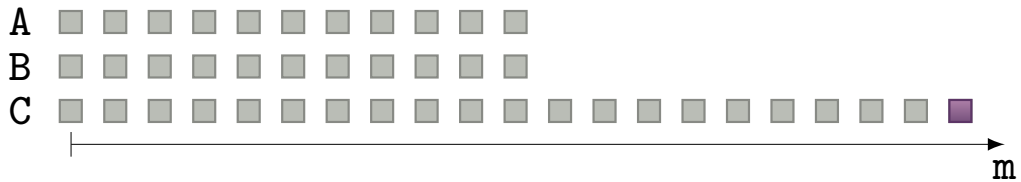
Produit de deux polynômes :

- coefficients dans A et B ;
- résultat dans C.

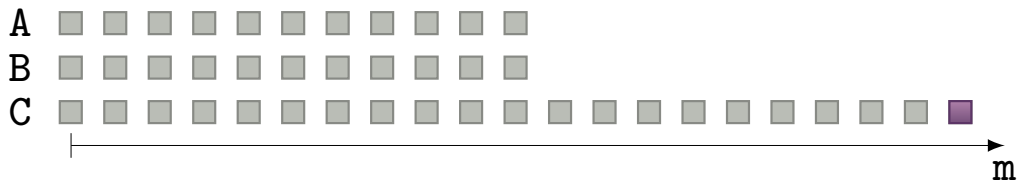
Schedule :



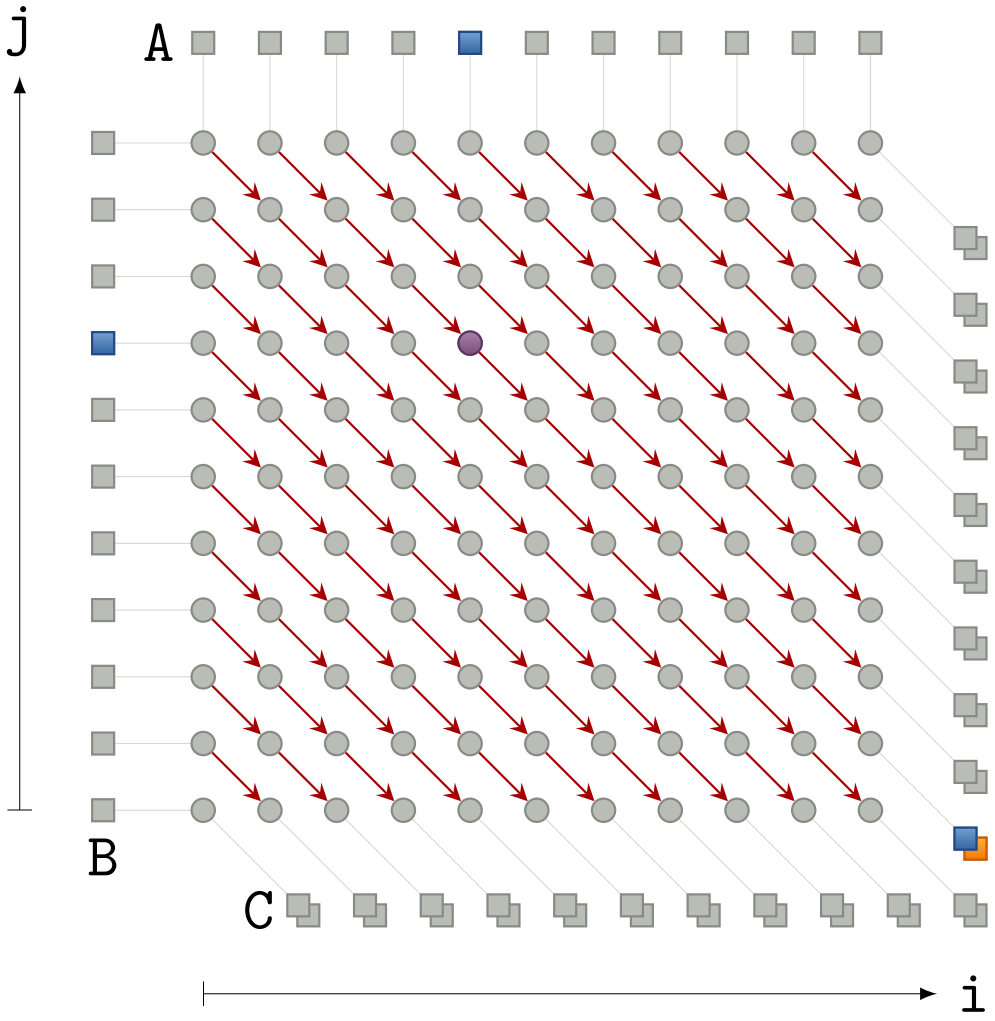
Read :



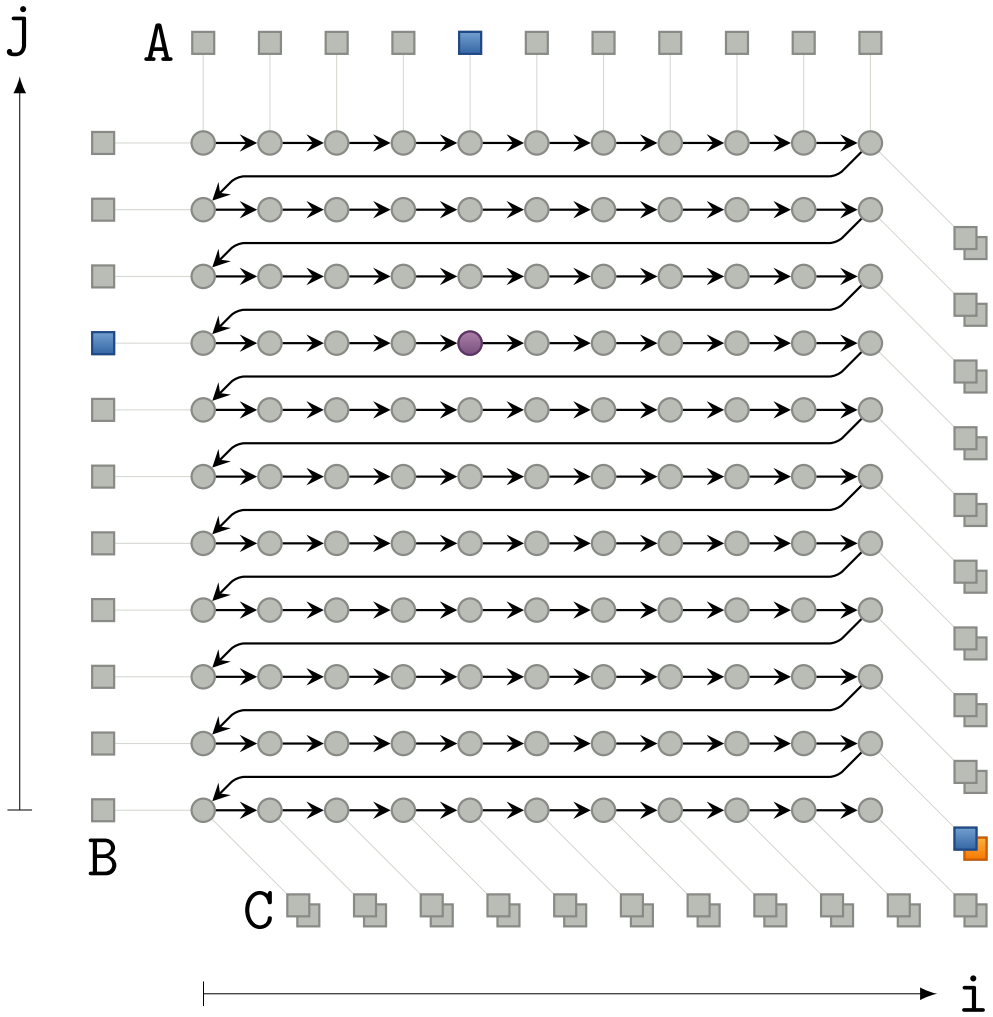
Write :



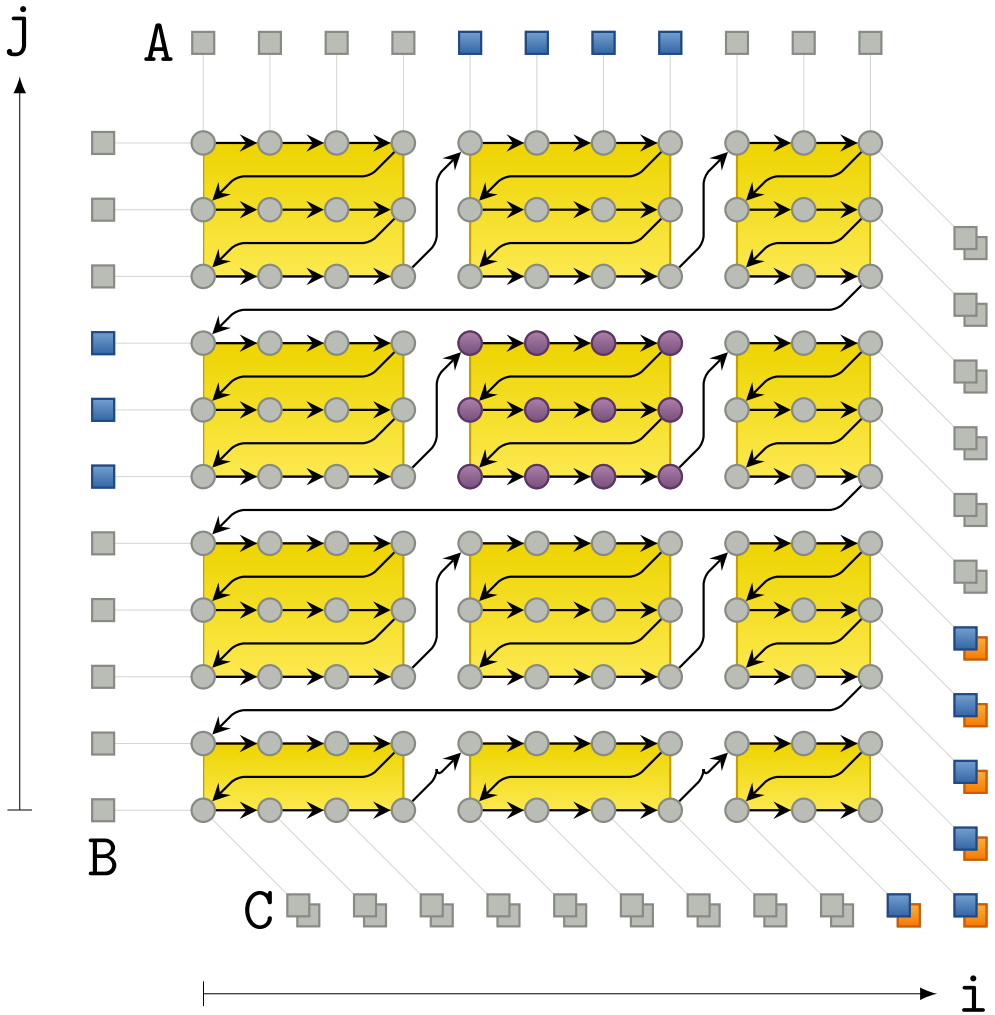
Ordonnancements alternatifs : dépendances



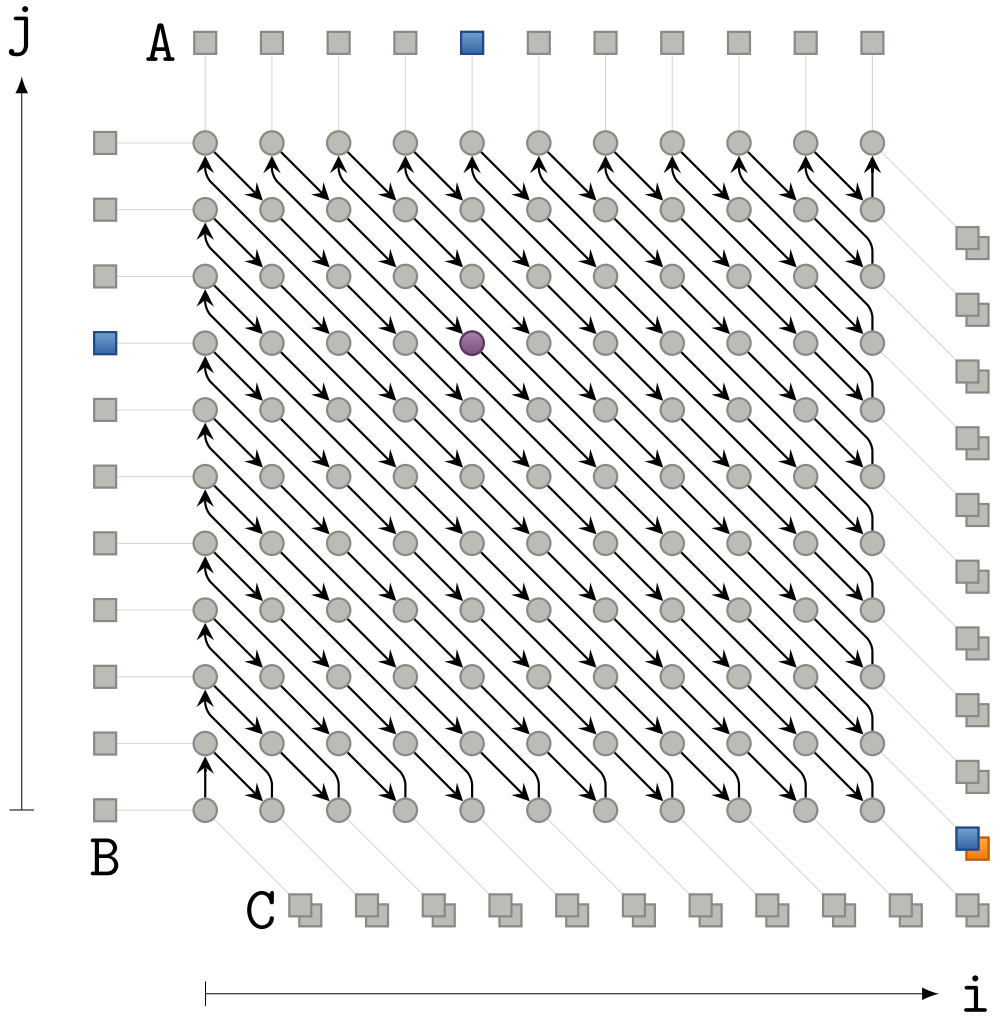
Ordonnancements alternatifs : inversion de boucles



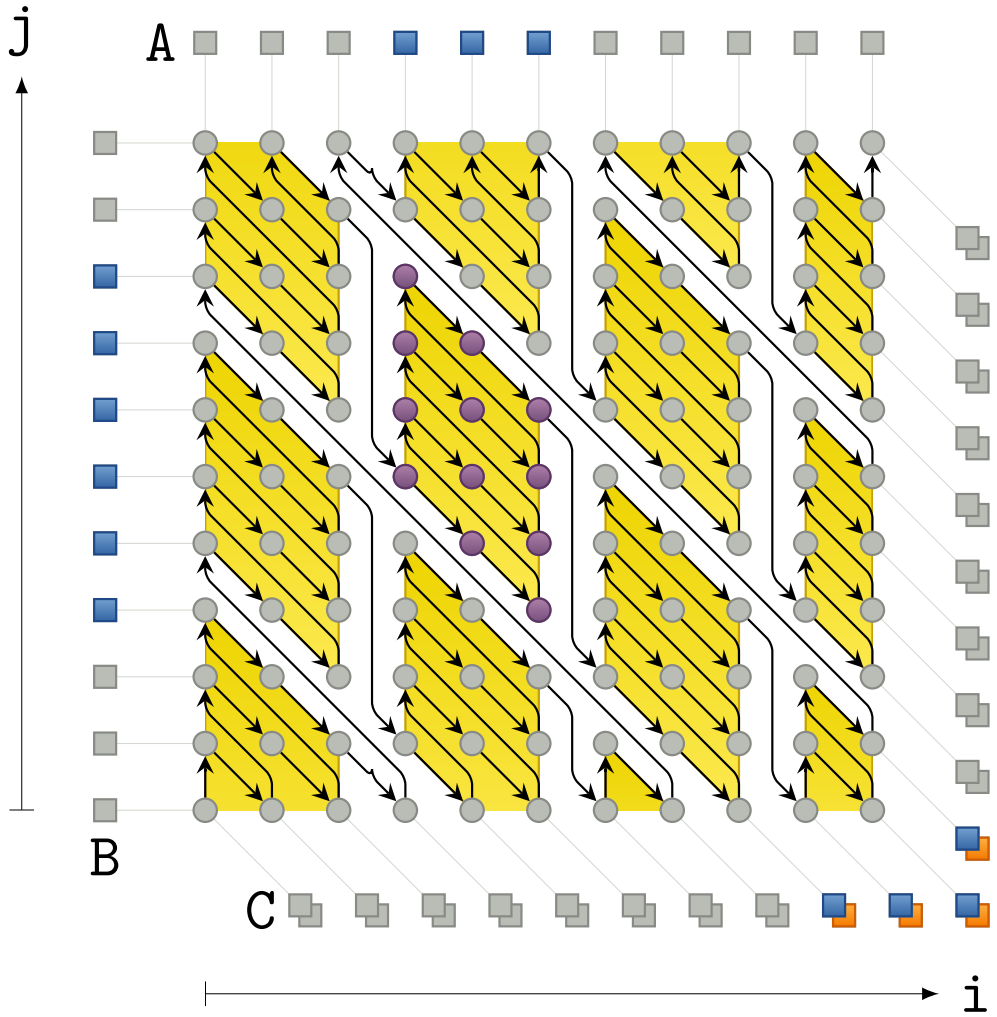
Ordonnancements alternatifs : inversion + tuilage



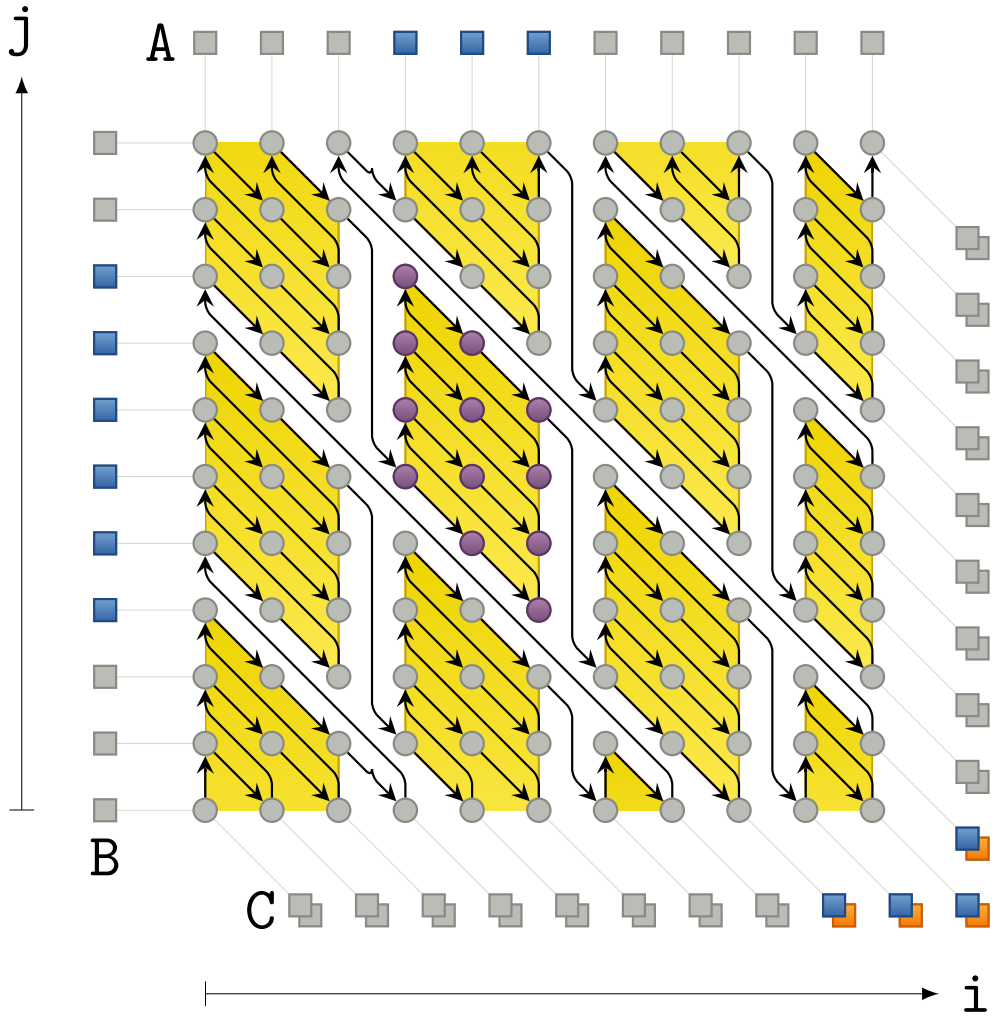
Ordonnancements alternatifs : torsion



Ordonnancements alternatifs : torsion + tuilage



Ordonnancements alternatifs : torsion + tuilage

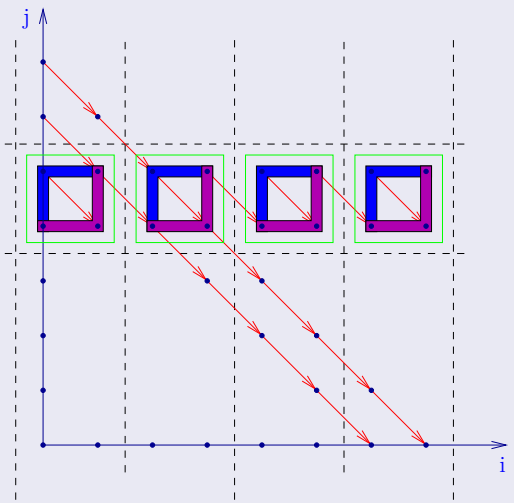


+ possibilité de tuilage hiérarchique et de parallélisme intra-tuile.

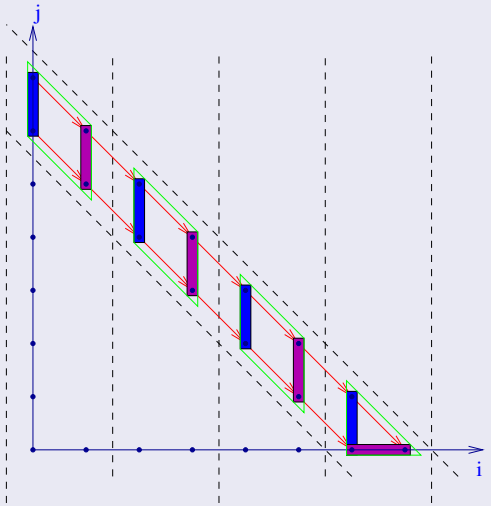
Réutilisation des données dans une bande de tuiles

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

$(i,j) \mapsto (n-j-1,i)$



$(i,j) \mapsto (i+j,i)$

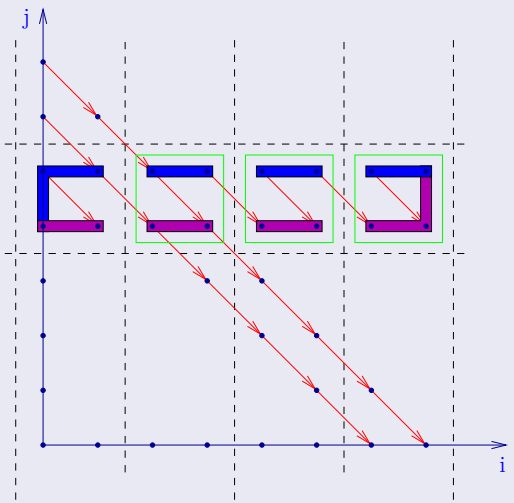


Dans une bande, **Load** $\simeq$ premières lectures, **Store** $\simeq$ dernières écritures.

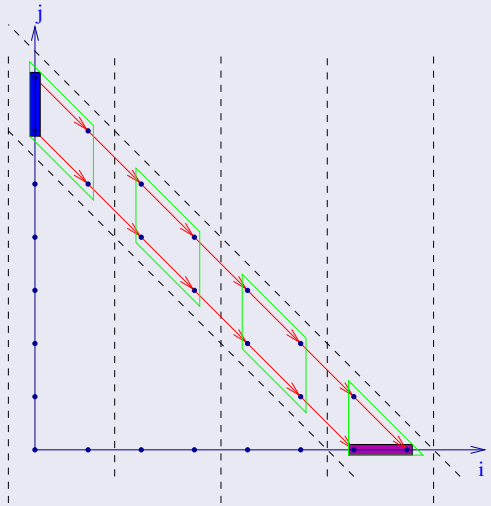
Réutilisation des données dans une bande de tuiles

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

$(i,j) \mapsto (n-j-1,i)$

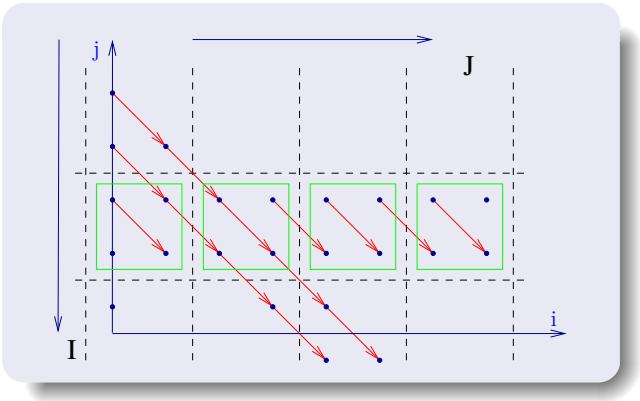


$(i,j) \mapsto (i+j,i)$



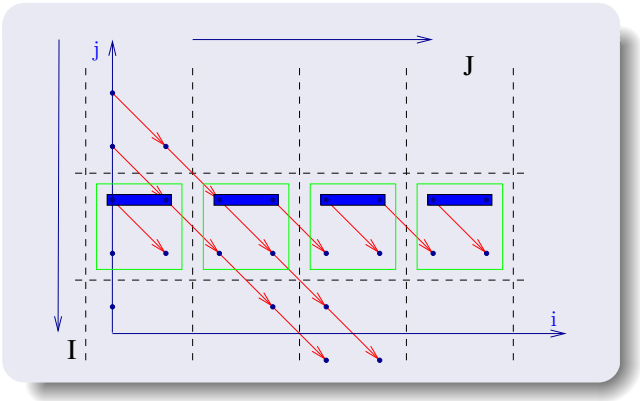
Dans une bande, **Load** $\simeq$ premières lectures, **Store** $\simeq$ dernières écritures.

Ce qu'on cherche à obtenir : transferts et tailles mémoire



- Borne n, taille de tuile b.
- Tuilage avec $(i,j) \mapsto (i',j') = (n-j-1, i)$.
- Fonctions d'accès $m = i+j = j'+n-i'-1$.
- Premier indice de tuile (bI, bJ).
- Transferts Load_A, Load_B, Load_C, Store_C.

Ce qu'on cherche à obtenir : transferts et tailles mémoire



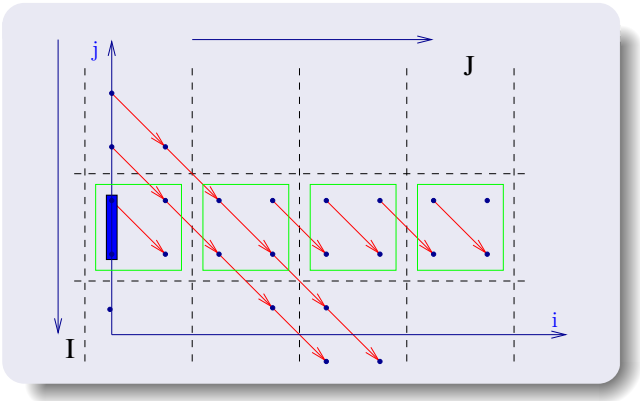
- Borne n, taille de tuile b.
- Tuilage avec $(i,j) \mapsto (i',j') = (n-j-1, i)$.
- Fonctions d'accès $m = i+j = j'+n-i'-1$.
- Premier indice de tuile (bI, bJ).
- Transferts Load_A, Load_B, Load_C, Store_C.

En version "double-buffering", tailles des mémoires locales :

$$\mathrm{Load}_A = \{m \mid 0 \leq m \leq n-1,\ bJ \leq m \leq bJ+b-1\}$$

- taille $2b$, si $n \geq 2b+1$: 2 tuiles en même temps.
- taille n si $n \leq 2b$: moins de 2 tuiles.

Ce qu'on cherche à obtenir : transferts et tailles mémoire



- Borne n, taille de tuile b.
- Tuilage avec $(i,j) \mapsto (i',j') = (n-j-1, i)$.
- Fonctions d'accès $m = i+j = j'+n-i'-1$.
- Premier indice de tuile (bI, bJ).
- Transferts Load_A, Load_B, Load_C, Store_C.

En version "double-buffering", tailles des mémoires locales :

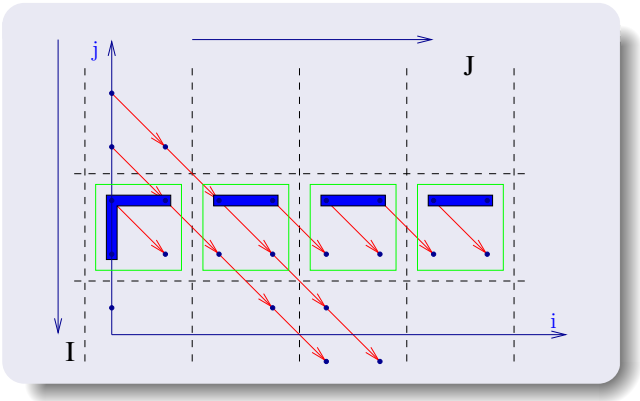
$\mathrm{Load}_A = \{m \mid 0 \leq m \leq n-1,\ bJ \leq m \leq bJ+b-1\}$

- taille $2b$, si $n \geq 2b+1$: 2 tuiles en même temps.
- taille n si $n \leq 2b$: moins de 2 tuiles.

$\mathrm{Load}_B = \{m \mid J = 0,\ 0 \leq m \leq n-1,\ n-bI-b \leq m \leq n-bI-1\}$

- taille b si $n \geq b$: 1 tuile complète.
- taille n si $n \leq b-1$: 1 tuile partielle.

Ce qu'on cherche à obtenir : transferts et tailles mémoire



- Borne n, taille de tuile b.
- Tuilage avec $(i,j) \mapsto (i',j') = (n-j-1, i)$.
- Fonctions d'accès $m = i+j = j'+n-i'-1$.
- Premier indice de tuile (bI, bJ).
- Transferts Load_A, Load_B, Load_C, Store_C.

En version "double-buffering", tailles des mémoires locales :

$$\mathrm{Load}_A = \{m \mid 0 \le m \le n-1,\ bJ \le m \le bJ+b-1\}$$

- taille $2b$, si $n \ge 2b+1$: 2 tuiles en même temps.
- taille n si $n \le 2b$: moins de 2 tuiles.

$$\mathrm{Load}_B = \{m \mid J = 0,\ 0 \le m \le n-1,\ n-bI-b \le m \le n-bI-1\}$$

- taille b si $n \ge b$: 1 tuile complète.
- taille n si $n \le b-1$: 1 tuile partielle.

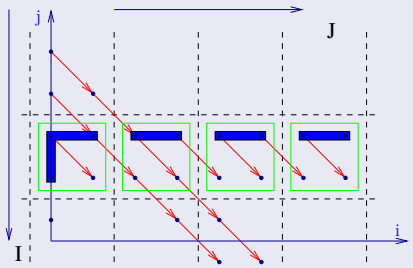
$$\mathrm{Load}_C = \{m \mid 0 \le m,\ n-bI-b \le m \le n-1-bI,\ J = 0\}$$
$$\cup \{m \mid \max(1, bJ) \le m+bI-n+1 \le \min(n-1, bJ+b-1)\}$$

- taille $(2b-1)+b = 3b-1$ si $n \ge 2b+1$: 2 tuiles complètes.
- taille $(2b-1)+(n-b) = b+n-1$ si $b \le n \le 2b$: 1 tuile complète, une partielle.
- taille $2n-1$ si $n \le b-1$: une seule tuile.

Plan de l'exposé

Ex : calcul de $\mathrm{Load}(J)$ pour une tuile $b \times b$ indexée par J

$(i,j) \mapsto (n-j-1, i)$



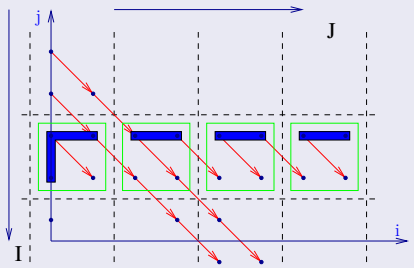
Lectures de $c[m]$ en fonction de (i,j) :

$$\begin{cases} i+j=m \\ 0 \le i \le n-1,\ 0 \le j \le n-1 \end{cases}$$

bleu=constante, rouge=paramètre

Ex : calcul de $\mathrm{Load}(J)$ pour une tuile $b \times b$ indexée par J

$(i,j) \mapsto (n-j-1, i)$



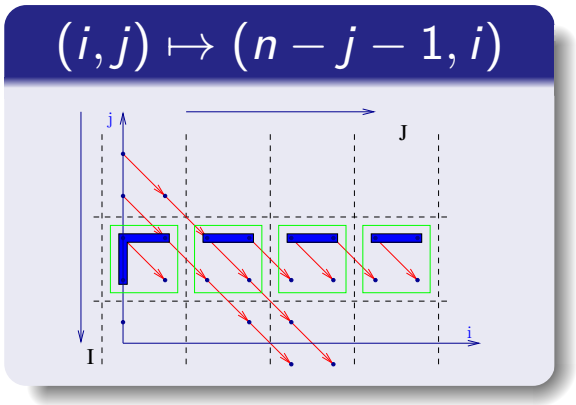
Changement de base $(i,j) \mapsto (n-1-j, i)$:

$$\begin{cases} i+j = m,\ i' = n-1-j,\ j' = i \\ 0 \le i \le n-1,\ 0 \le j \le n-1 \end{cases}$$

bleu=constante, rouge=paramètre

Ex : calcul de $\mathrm{Load}(J)$ pour une tuile $b \times b$ indexée par J

$(i,j) \mapsto (n-j-1, i)$

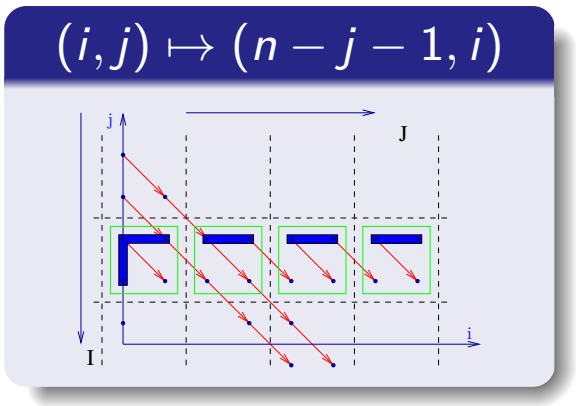


Tuilage $(I,J) = (\lfloor \frac{i'}{b} \rfloor, \lfloor \frac{j'}{b} \rfloor)$, I paramètre :

$$\begin{cases} i+j = m,\ i' = n-1-j,\ j' = i \\ 0 \le i \le n-1,\ 0 \le j \le n-1 \\ bI \le i' \le b(I+1)-1 \\ bJ \le j' \le b(J+1)-1 \end{cases}$$

bleu=constante, rouge=paramètre

Ex : calcul de $\mathrm{Load}(J)$ pour une tuile $b \times b$ indexée par J



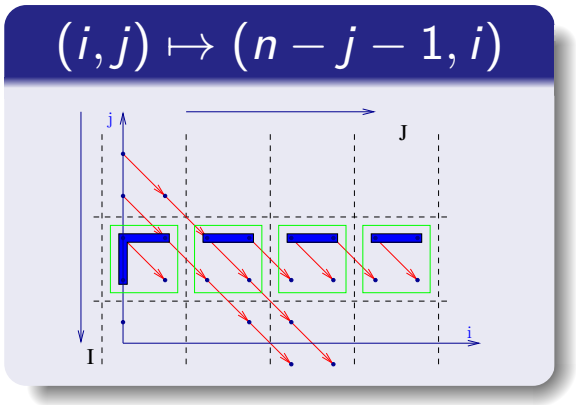
PIP calcule la première lecture dans une bande : minimum lexicographique de (J, i', j', i, j) :

$$\min_{\prec_{lex}} \begin{cases} i + j = m,\ i' = n - 1 - j,\ j' = i \\ 0 \leq i \leq n-1,\ 0 \leq j \leq n-1 \\ bI \leq i' \leq b(I+1) - 1 \\ bJ \leq j' \leq b(J+1) - 1 \end{cases}$$

bleu=constante, rouge=paramètre

Ex : calcul de $\mathrm{Load}(J)$ pour une tuile $b \times b$ indexée par J

$(i,j) \mapsto (n-j-1, i)$



PIP calcule la première lecture dans une bande : minimum lexicographique de (J, i', j', i, j) :

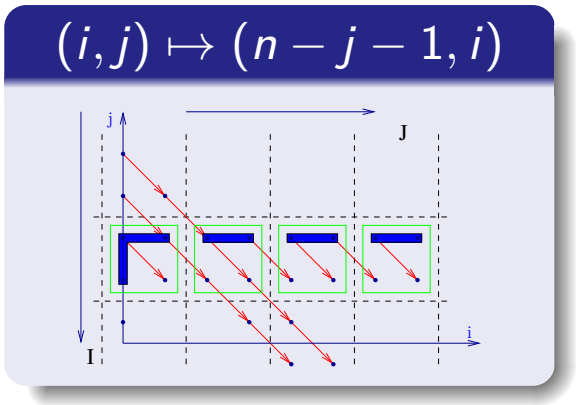
$$\min_{\prec_{lex}} \begin{cases} i+j = m,\ i' = n-1-j,\ j' = i \\ 0 \le i \le n-1,\ 0 \le j \le n-1 \\ bI \le i' \le b(I+1)-1 \\ bJ \le j' \le b(J+1)-1 \end{cases}$$

bleu=10, rouge=paramètre

if $(-10I + N - m \ge 0)$
  if $(10I - N + m + 9 \ge 0)$ /* segment vertical, première tuile */
    $(J, i', j', i, j) = (0, N-m, 0, 0, m)$
  else $\perp$ /* signifie non défini */
else
  if $(-10I + 2N - m \ge 0)$
    if $(-10I + N - m + 9 \ge 0)$ /* segment horizontal, première tuile */
      $(J, i', j', i, j) = (0, 10I, 10I - N + m, 10I - N + m, N - 10I)$
    else with $k = \lfloor \frac{N+9m+9}{10} \rfloor$ /* segment horizontal générique */
      $(J, i', j', i, j) = (I + m - k, 10I, 10I - N + m, 10I - N + m, N - 10I)$
  else $\perp$ /* non défini */

Ex : calcul de $\mathrm{Load}(J)$ pour une tuile $b \times b$ indexée par J

$(i,j) \mapsto (n-j-1, i)$



PIP calcule la première lecture dans une bande : minimum lexicographique de (J, i', j', i, j) :

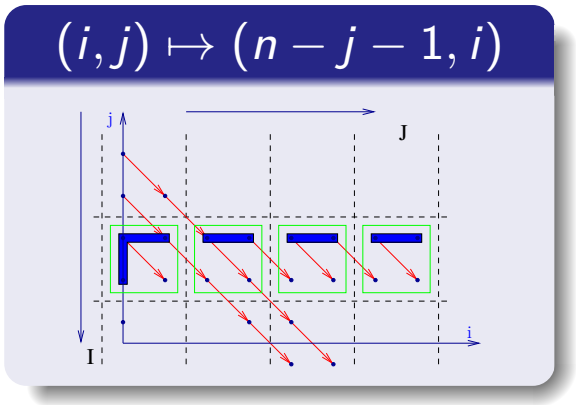
$$\min_{\prec_{lex}} \begin{cases} i + j = m,\ i' = n - 1 - j,\ j' = i \\ 0 \le i \le n-1,\ 0 \le j \le n-1 \\ bI \le i' \le b(I+1) - 1 \\ bJ \le j' \le b(J+1) - 1 \end{cases}$$

bleu=10, rouge=paramètre

if $(-10I + N - m \ge 0)$
  if $(10I - N + m + 9 \ge 0)$ /* segment vertical, première tuile */
    $(i,j) = (0, m)$
  else $\perp$ /* signifie non défini */
else
  if $(-10I + 2N - m \ge 0)$
    if $(-10I + N - m + 9 \ge 0)$ /* segment horizontal, première tuile */
      $(i,j) = (10I - N + m, N - 10I)$
    else with $k = \lfloor \frac{N+9m+9}{10} \rfloor$ /* segment horizontal générique */
      $(i,j) = (10I - N + m, N - 10I)$
  else $\perp$ /* non défini */

Ex : calcul de $\mathrm{Load}(J)$ pour une tuile $b \times b$ indexée par J

$(i,j) \mapsto (n-j-1, i)$



PIP calcule la première lecture dans une bande : minimum lexicographique de (J, i', j', i, j) :

$$\min_{\prec_{lex}} \begin{cases} i+j = m,\ i' = n-1-j,\ j' = i \\ 0 \le i \le n-1,\ 0 \le j \le n-1 \\ bI \le i' \le b(I+1)-1 \\ bJ \le j' \le b(J+1)-1 \end{cases}$$

bleu=10, rouge=paramètre

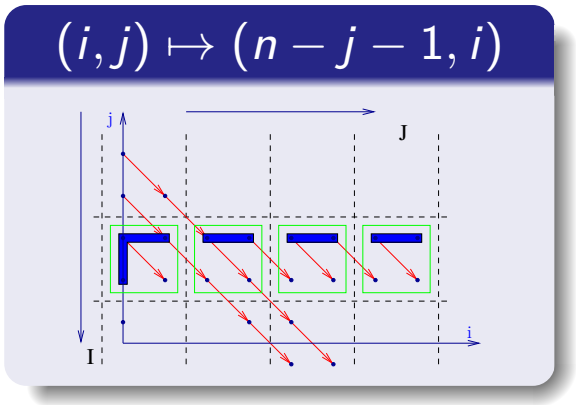
if $(-10I + N - m \ge 0)$
  if $(10I - N + m + 9 \ge 0)$
    $(i,j) = (0,m)$ /* segment vertical, première tuile */
  else $\perp$
else
  if $(-10I + 2N - m \ge 0)$
    $(i,j) = (10I - N + m, N - 10I)$ /* segment horizontal générique */
  else $\perp$ /* non défini */

Accessoirement, ceci donne les éléments du tableau c accédés :

$$\{m \mid \max(0, N-10I-9) \le m \le N-10I\} \cup \{m \mid N-10I+1 \le m \le 2N-10I\}$$

Ex : calcul de $\mathrm{Load}(J)$ pour une tuile $b \times b$ indexée par J

$(i,j) \mapsto (n-j-1, i)$



PIP calcule la première lecture dans une bande : minimum lexicographique de (J, i', j', i, j) :

$$\min_{\prec_{lex}} \begin{cases} i+j = m,\ i' = n-1-j,\ j' = i \\ 0 \le i \le n-1,\ 0 \le j \le n-1 \\ bI \le i' \le b(I+1)-1 \\ bJ \le j' \le b(J+1)-1 \end{cases}$$

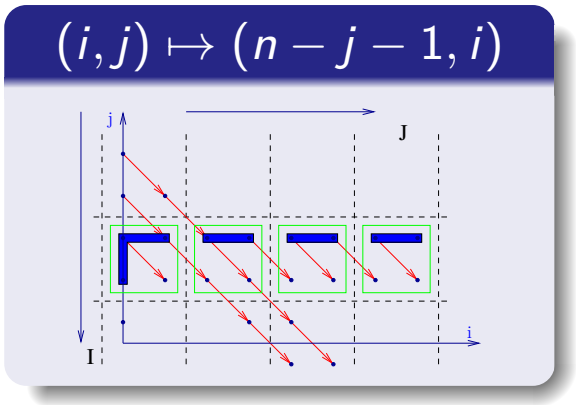
bleu=10, rouge=paramètre

Après simplification :

$$\mathrm{FirstOpRead}(m) = \{(i,j) \mid (i,j) = (0,m),\ 0 \le m,\ n-10-10I \le m \le n-1-10I\}$$
$$\cup \{(i,j) \mid (i,j) = (10I-n+1+m, n-1-10I),\ n-10I \le m \le 2n-2-10I\}$$

Ex : calcul de $\mathrm{Load}(J)$ pour une tuile $b \times b$ indexée par J

$(i,j) \mapsto (n-j-1, i)$



PIP calcule la première lecture dans une bande : minimum lexicographique de (J, i', j', i, j) :

$$\min_{\prec_{lex}} \begin{cases} i+j = m,\ i' = n-1-j,\ j' = i \\ 0 \le i \le n-1,\ 0 \le j \le n-1 \\ bI \le i' \le b(I+1)-1 \\ bJ \le j' \le b(J+1)-1 \end{cases}$$

bleu=10, rouge=paramètre

Après simplification :

$$\mathrm{FirstOpRead}(m) = \{(i,j) \mid (i,j) = (0,m),\ 0 \le m,\ n-10-10I \le m \le n-1-10I\}$$
$$\cup \{(i,j) \mid (i,j) = (10I-n+1+m, n-1-10I),\ n-10I \le m \le 2n-2-10I\}$$

Introduction des tuiles et expression de m en fonction de J :

$$\mathrm{FirstReadInTile}(J) = \{m \mid \max(0, n-10I-10) \le m \le n-1-10I,\ J=0\}$$
$$\cup \{m \mid \max(1, 10J) \le m+10I-n+1 \le \min(n-1, 10J+9)\}$$

Calcul des In/Out

On veut regrouper les accès mémoire :

In Tuile $\rightarrow$ Adresses lues depuis l'extérieur

Out Tuile $\rightarrow$ Adresses écrites vers l'extérieur

$$\mathrm{In}(T) = \bigcup_{\vec{i} \in T} \mathrm{Read}(\vec{i})$$

$$\mathrm{Out}(T) = \bigcup_{\vec{i} \in T} \mathrm{Write}(\vec{i})$$

Calcul des In/Out

On veut regrouper les accès mémoire :

In Tuile $\rightarrow$ Adresses lues depuis l'extérieur

Out Tuile $\rightarrow$ Adresses écrites vers l'extérieur

$$\mathrm{In}(T) = \bigcup_{\vec{i} \in T} \left(\mathrm{Read}(\vec{i}) \setminus \bigcup_{\vec{j} \in T, \vec{j} <_{lex} \vec{i}} \mathrm{Write}(\vec{j}) \right)$$

☛ Écriture avant lecture dans la tuile $\Rightarrow$ pas nécessaire

$$\mathrm{Out}(T) = \bigcup_{\vec{i} \in T} \mathrm{Write}(\vec{i})$$

Calcul des In/Out

On veut regrouper les accès mémoire :

In Tuile $\rightarrow$ Adresses lues depuis l'extérieur

Out Tuile $\rightarrow$ Adresses écrites vers l'extérieur

$$\mathrm{In}(T) = \bigcup_{\vec{i} \in T} \left(\mathrm{Read}(\vec{i}) \setminus \bigcup_{\vec{j} \in T, \vec{j} <_{lex} \vec{i}} \mathrm{Write}(\vec{j}) \right)$$

☛ Écriture avant lecture dans la tuile $\Rightarrow$ pas nécessaire

$$\mathrm{Out}(T) = \bigcup_{\vec{i} \in T} \mathrm{Write}(\vec{i})$$

☛ Double écriture dans la tuile $\Rightarrow$ quand même nécessaire

Calcul des Load/Store

On veut faire de la réutilisation :

Load Tuile $\rightarrow$ Adresses à charger depuis l'extérieur

Store Tuile $\rightarrow$ Adresses à stocker vers l'extérieur

$$\mathrm{Load}(T) = \mathrm{In}(T)$$

$$\mathrm{Store}(T) = \mathrm{Out}(T)$$

Calcul des Load/Store

On veut faire de la réutilisation :

Load Tuile $\rightarrow$ Adresses à charger depuis l'extérieur

Store Tuile $\rightarrow$ Adresses à stocker vers l'extérieur

$$\mathrm{Load}(T) = \mathrm{In}(T) \setminus \bigcup_{T' < T} \mathrm{Out}(T')$$

☛ Écrit dans une tuile précédente $\Rightarrow$ gardé en local

$$\mathrm{Store}(T) = \mathrm{Out}(T)$$

Calcul des Load/Store

On veut faire de la réutilisation : ☛ Calcul récursif

Load Tuile $\rightarrow$ Adresses à charger depuis l'extérieur

Store Tuile $\rightarrow$ Adresses à stocker vers l'extérieur

$$\mathrm{Load}(T) = \mathrm{In}(T) \setminus \left(\bigcup_{T'<T} \mathrm{Out}(T') \cup \bigcup_{T'<T} \mathrm{Load}(T') \right)$$

☛ Écrit dans une tuile précédente $\Rightarrow$ gardé en local

☛ Chargé dans une tuile précédente $\Rightarrow$ gardé en local

$$\mathrm{Store}(T) = \mathrm{Out}(T)$$

Calcul des Load/Store

On veut faire de la réutilisation : ☛ Calcul récursif

Load Tuile $\rightarrow$ Adresses à charger depuis l'extérieur

Store Tuile $\rightarrow$ Adresses à stocker vers l'extérieur

$$\mathrm{Load}(T) = \mathrm{In}(T) \setminus \left(\bigcup_{T'<T} \mathrm{Out}(T') \cup \bigcup_{T'<T} \mathrm{Load}(T') \right)$$

☛ Écrit dans une tuile précédente $\Rightarrow$ gardé en local

☛ Chargé dans une tuile précédente $\Rightarrow$ gardé en local

$$\mathrm{Store}(T) = \mathrm{Out}(T) \setminus \bigcup_{T<T'} \mathrm{Store}(T')$$

☛ Stocké dans une tuile suivante $\Rightarrow$ gardé en local

Calcul des Load/Store

On veut faire de la réutilisation : ☛ Calcul dérécursivé

Load Tuile $\rightarrow$ Adresses à charger depuis l'extérieur

Store Tuile $\rightarrow$ Adresses à stocker vers l'extérieur

$$\mathrm{Load}(T) = \mathrm{In}(T) \setminus \left(\bigcup_{T'<T} \mathrm{Out}(T') \cup \bigcup_{T'<T} \mathrm{In}(T') \right)$$

- ☛ Écrit dans une tuile précédente $\Rightarrow$ gardé en local
- ☛ Chargé dans une tuile précédente $\Rightarrow$ gardé en local

$$\mathrm{Store}(T) = \mathrm{Out}(T) \setminus \bigcup_{T<T'} \mathrm{Out}(T')$$

- ☛ Stocké dans une tuile suivante $\Rightarrow$ gardé en local

Tuilage paramétré quadratique (ex en 2D)

En exprimant les tuiles différemment,

$$T(T_j, T_i) = \{(j, i) \mid s_j T_j \leq j < s_j(T_j + 1) \ \wedge \ s_i T_i \leq i < s_i(T_i + 1)\}$$

devient, avec (J, I) premier point de la tuile :

$$T(J, I) = \{(j, i) \mid J \leq j < J + s_j \ \wedge \ I \leq i < I + s_i\}$$

Tuilage paramétré quadratique (ex en 2D)

En exprimant les tuiles différemment,

$$T(T_j, T_i) = \{(j,i) \mid s_j T_j \leq j < s_j(T_j+1) \ \wedge \ s_i T_i \leq i < s_i(T_i+1)\}$$

devient, avec (J, I) premier point de la tuile :

$$T(J, I) = \{(j,i) \mid J \leq j < J + s_j \ \wedge \ I \leq i < I + s_i\}$$

☛ Calcul de In/Out paramétré !

Tuilage paramétré quadratique (ex en 2D)

En exprimant les tuiles différemment,

$$T(T_j, T_i) = \{(j,i) \mid s_j T_j \leq j < s_j(T_j+1) \ \wedge \ s_i T_i \leq i < s_i(T_i+1)\}$$

devient, avec (J, I) premier point de la tuile :

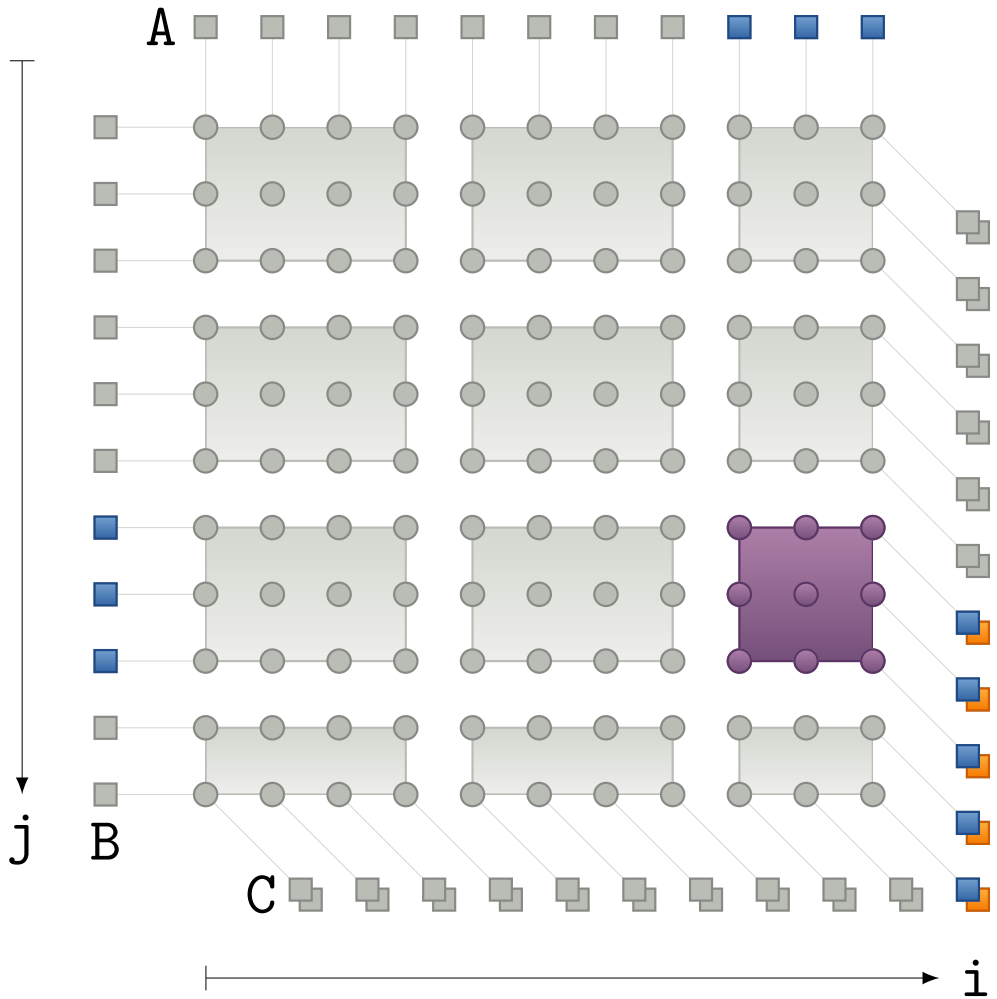
$$T(J, I) = \{(j,i) \mid J \leq j < J + s_j \ \wedge \ I \leq i < I + s_i\}$$

- ☛ Calcul de In/Out paramétré !
- ☛ Calcul de Load/Store toujours problématique :

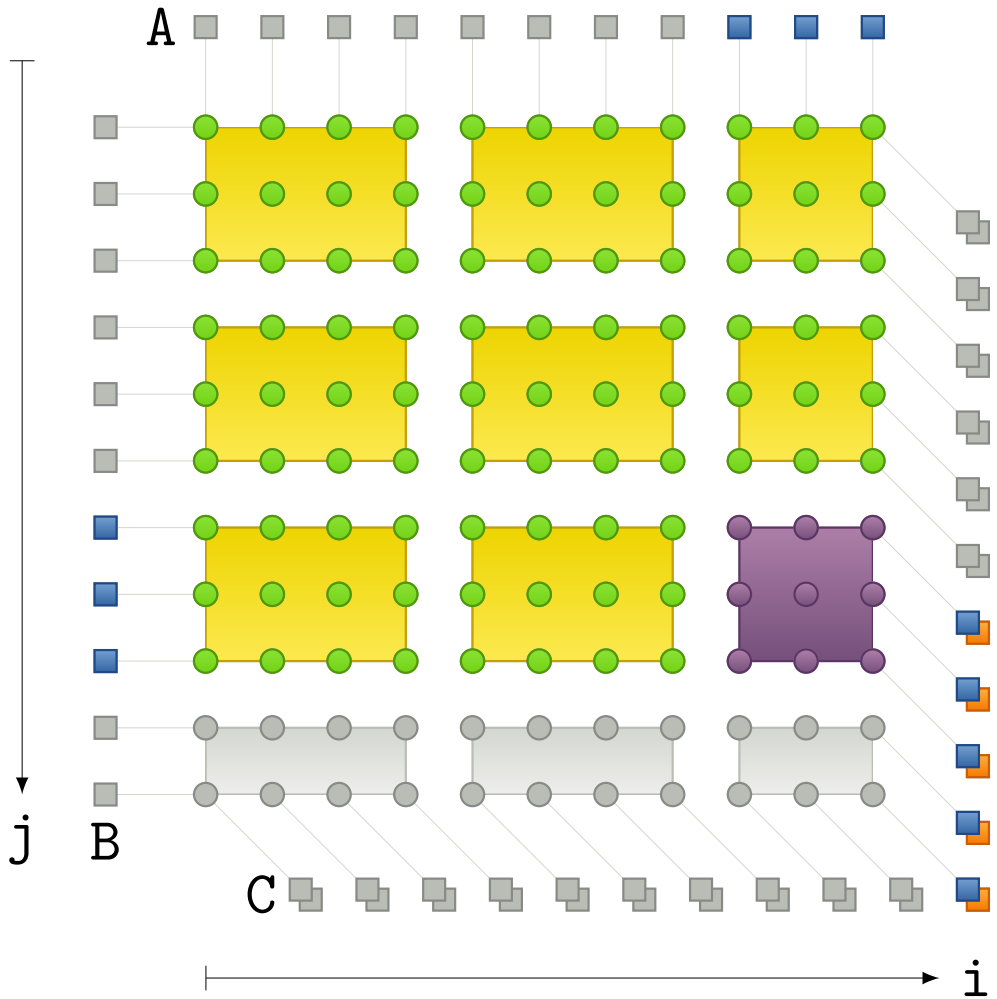
$$T(J', I') < T(J, I) \Leftrightarrow (J', I') <_{lex} (J, I) \wedge J' \underset{s_j}{\equiv} J \wedge I' \underset{s_i}{\equiv} I$$

Plan de l’exposé

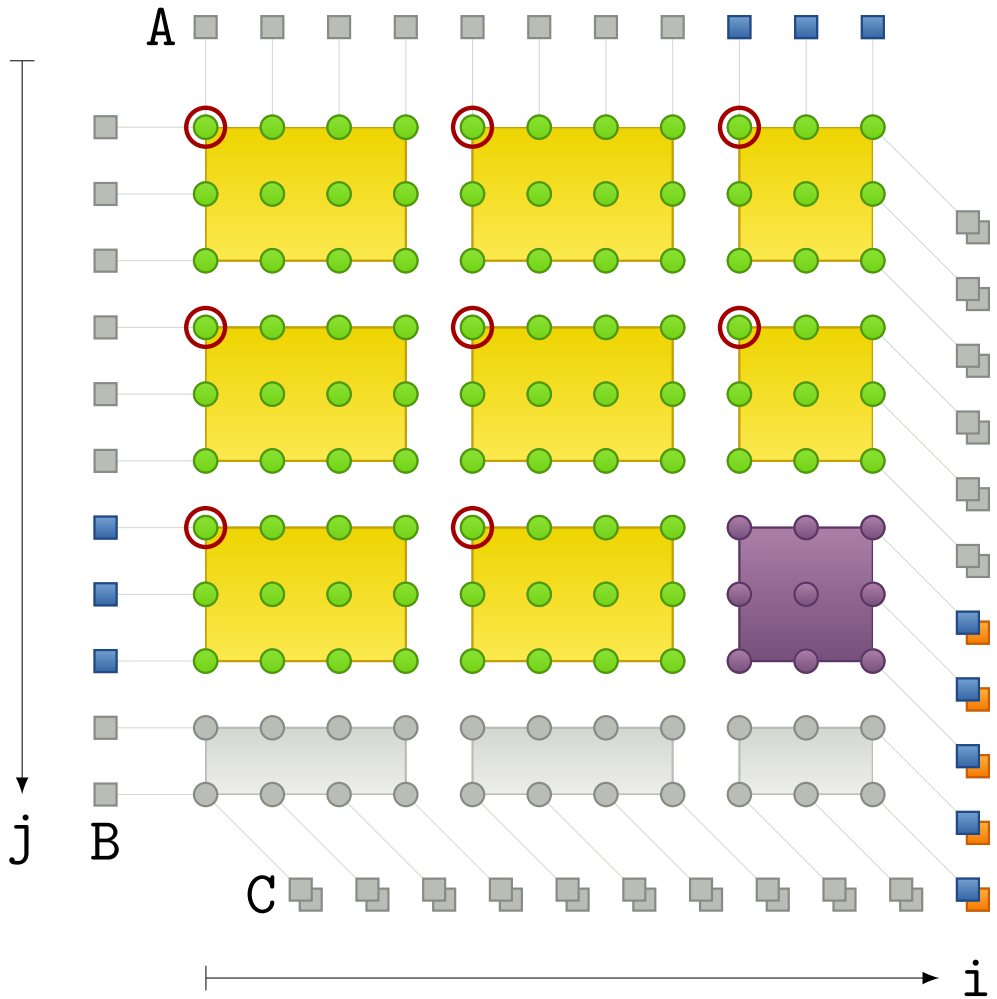
“Tuilage” redondant



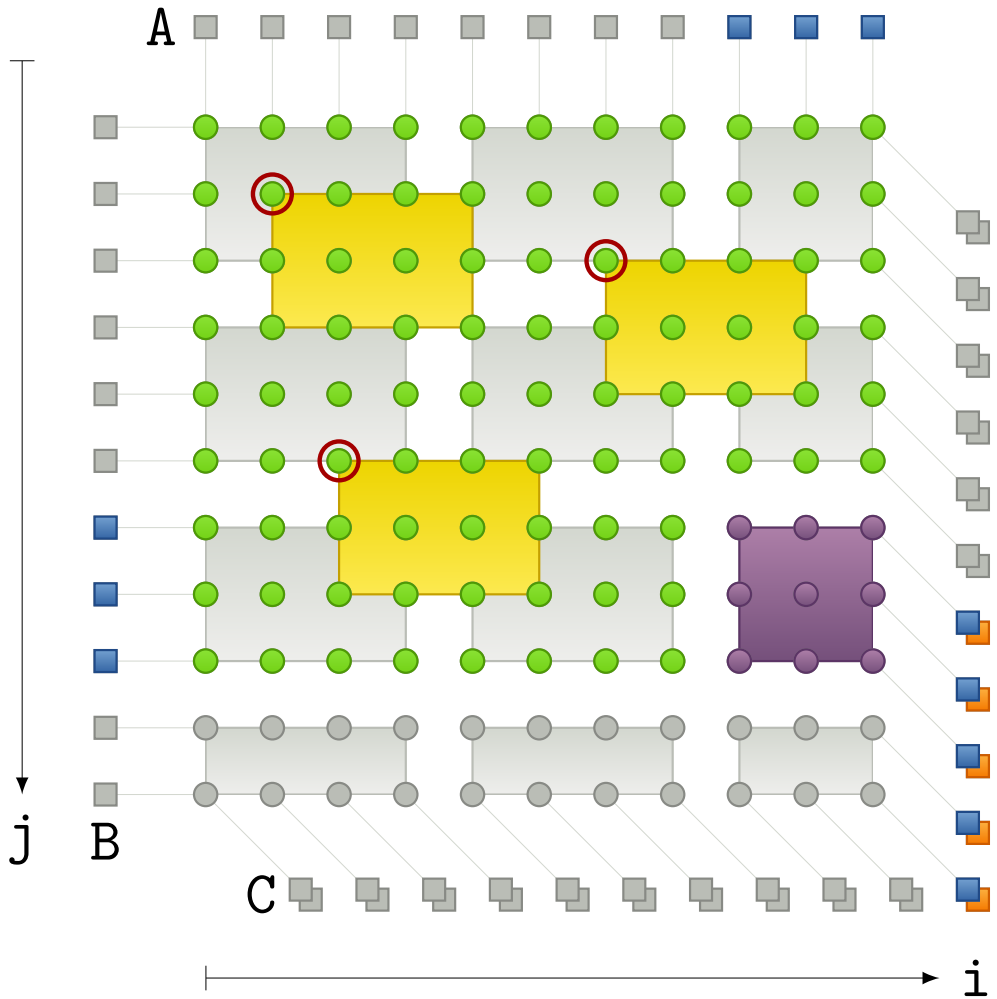
“Tuilage” redondant



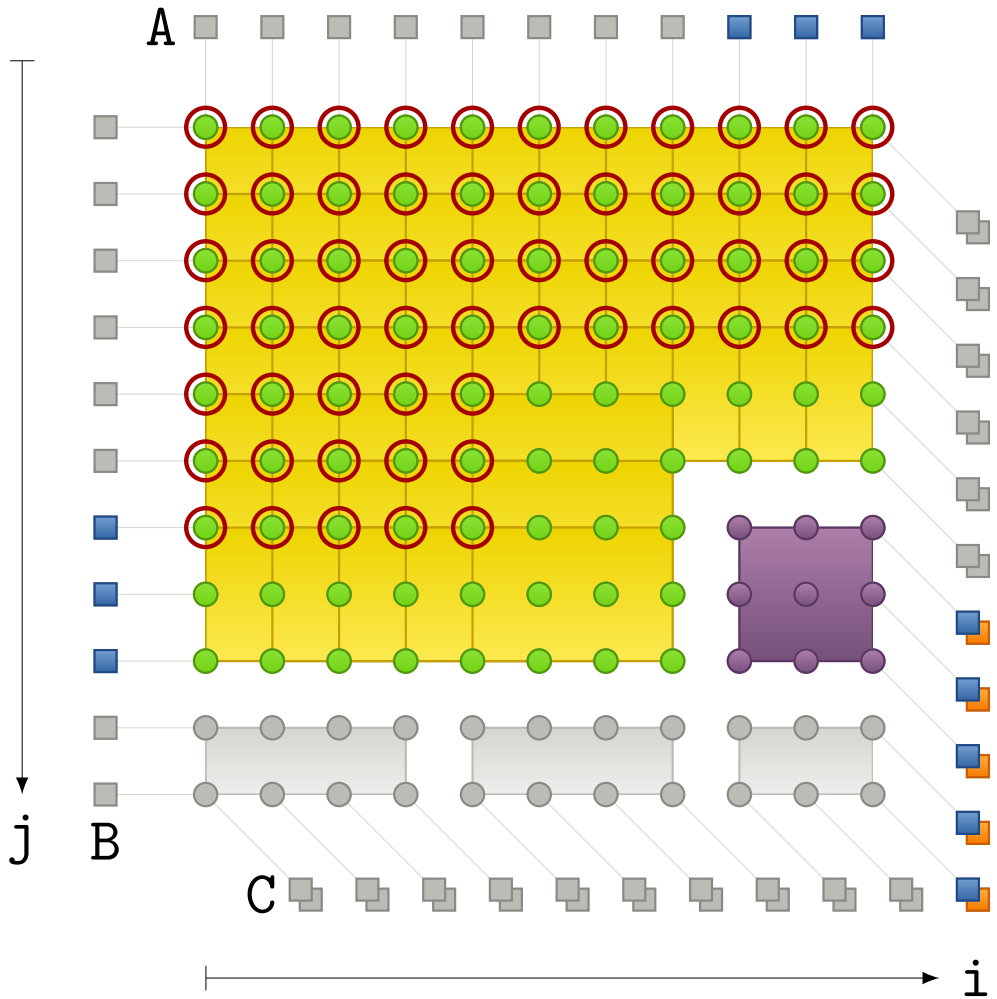
“Tuilage” redondant



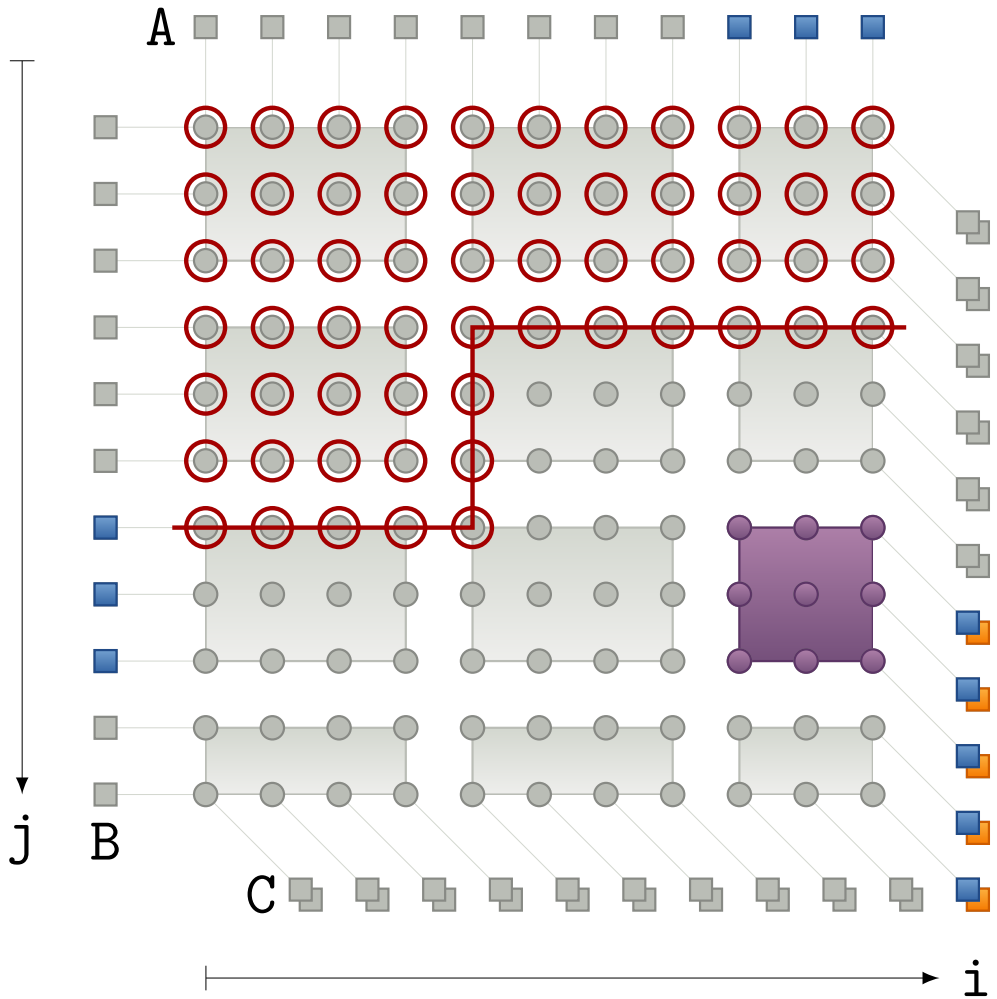
“Tuilage” redondant



“Tuilage” redondant



“Tuilage” redondant



Subtilement correct

Définition de $T(J', I') \prec T(J, I)$:

$$J' \leq J - s_j \vee (J' \leq J \wedge I' \leq I - s_i)$$

Or, bien que :

$$\{(J', I') \mid T(J', I') < T(J, I)\} \neq \{(J', I') \mid T(J', I') \prec T(J, I)\}$$

On a :

$$\bigcup_{T(J',I') < T(J,I)} T(J', I') = \bigcup_{T(J',I') \prec T(J,I)} T(J', I')$$

☛ Ne pas confondre les deux !

Calcul paramétré des Load/Store

$$\mathsf{Store}(T) = \mathsf{Out}(T) \setminus \bigcup_{T<T'} \mathsf{Out}(T')$$

Calcul paramétré des Load/Store

$$\mathrm{Store}(J, I) = \mathrm{Out}(J, I) \setminus \bigcup_{T(J,I)<T(J',I')} \mathrm{Out}(J', I')$$

☛ Premier cas...

Calcul paramétré des Load/Store

$$\mathrm{Store}(J, I) = \mathrm{Out}(J, I) \setminus \bigcup_{T(J,I) < T(J',I')} \mathrm{Out}(J', I')$$

☛ Premier cas...

$$\mathrm{Store}(J, I) = \mathrm{Out}(J, I) \setminus \bigcup_{T(J,I) < T(J',I')} \left(\bigcup_{(j',i') \in T(J',I')} \mathrm{Write}(j', i') \right)$$

Calcul paramétré des Load/Store

$$\text{Store}(J, I) = \text{Out}(J, I) \setminus \bigcup_{T(J,I)<T(J',I')} \text{Out}(J', I')$$

☛ Premier cas...

$$\text{Store}(J, I) = \text{Out}(J, I) \setminus \bigcup_{T(J,I)<T(J',I')} \left(\bigcup_{(j',i')\in T(J',I')} \text{Write}(j', i') \right)$$

☛ Deuxième cas !

Calcul paramétré des Load/Store

$$\text{Store}(J, I) = \text{Out}(J, I) \setminus \bigcup_{T(J,I)<T(J',I')} \text{Out}(J', I')$$

- Premier cas...

$$\text{Store}(J, I) = \text{Out}(J, I) \setminus \bigcup_{T(J,I)<T(J',I')} \left(\bigcup_{(j',i')\in T(J',I')} \text{Write}(j', i') \right)$$

  - Deuxième cas !
- Idée : Out est intrinsèquement définie par point

Calcul paramétré des Load/Store

$$\text{Store}(J, I) = \text{Out}(J, I) \setminus \bigcup_{T(J,I) < T(J',I')} \text{Out}(J', I')$$

- Premier cas...

$$\text{Store}(J, I) = \text{Out}(J, I) \setminus \bigcup_{T(J,I) < T(J',I')} \left(\bigcup_{(j',i') \in T(J',I')} \text{Write}(j', i') \right)$$

  - Deuxième cas !
- Idée : Out est intrinsèquement définie par point

$$\text{Load}(T) = \text{In}(T) \setminus \left(\bigcup_{T' < T} \text{Out}(T') \cup \bigcup_{T' < T} \text{In}(T') \right)$$

- In n'est pas forcément par point !
- Mais In $\cup$ Out l'est toujours.

Fonctions définies par point

Définition (Fonction par point)

$\mathcal{A}$, $\mathcal{B}$ deux ensembles, $\mathcal{C} \subseteq \mathcal{P}(\mathcal{A})$. $F : \mathcal{C} \to \mathcal{P}(\mathcal{B})$ est *par point* s'il existe $f : \mathcal{A} \to \mathcal{P}(\mathcal{B})$ telle que $\forall X \in \mathcal{C}$, $F(X) = \bigcup_{x \in X} f(x)$.

Propriété (Plus grande sous-approximation par point)

Soit $F : \mathcal{C} \subseteq \mathcal{P}(\mathcal{A}) \to \mathcal{P}(\mathcal{B})$. Alors $\underline{F}$, définie par point à partir de $\underline{f}(x) = \bigcap_{Y \in \mathcal{C}, x \in Y} F(Y)$, est la plus grande sous-approximation par point de F. En particulier, F est par point ssi $F = \underline{F}$.

Propriété (Couverture par une union et stabilité des unions)

La fonction $F : \mathcal{C} \subseteq \mathcal{P}(\mathcal{A}) \to \mathcal{P}(\mathcal{B})$ est par point :
ssi $\forall X \in \mathcal{C}$, $\forall \mathcal{C}' \subseteq \mathcal{C}$, $X \subseteq \bigcup_{X' \in \mathcal{C}'} X' \Rightarrow F(X) \subseteq \bigcup_{X' \in \mathcal{C}'} F(X')$ ssi $\forall \mathcal{C}', \mathcal{C}'' \subseteq \mathcal{C}$, $\bigcup_{X \in \mathcal{C}'} X = \bigcup_{X \in \mathcal{C}''} X \Rightarrow \bigcup_{X \in \mathcal{C}'} F(X) = \bigcup_{X \in \mathcal{C}''} F(X)$.

Approximations : pourquoi ?

Execution d'une opération incertaine

- présence de `if` non analysables.

Accès à une donnée incertain

- fonction d'accès non complètement analysable.

Ensembles In/Out approchés

- du fait de l'analyse (ex : regions de tableaux) ;
- par restriction voulue (ex : hyper-rectangles).

Ensembles Load/Store approchés

- pour simplifier le code ;
- pour garantir des communications par blocs ;
- pour simplifier l'allocation mémoire ;
- ...

Conclusion

Développement d'un outil

- Passer d'un script ISCC à un outil complet avec ISL.
- Mettre en place des schémas d'approximations (dues aux codes et/ou par choix). Lien avec PIPS ?
- Améliorer le calcul de taille mémoire : nouvelles situations, nouveaux problèmes.

Expérimentations de calculs par blocs

- Vers FPGA ? Stations de travail ? GPU ? Kalray MPPA ?
- Modèles de coût en lien avec le tuilage hiérarchique ?
- Autres schémas de réutilisation (stockage partiel) ?

Pointwise functions

- Utile pour d'autres approximations ?

Merci

Questions ?

Script ISCC

```
# Inputs
Params := [N, s_1, s_2] -> { : s_1 >= 0 and s_2 >= 0 };
Domain := [N] -> { # Iteration domains
  S_1[k] : 0 <= k < 2N-1;
  S_2[i, j] : 0 <= i,j < N;
} * Params;
Read := [N] -> { # Read access functions
  S_2[i, j] -> A[i];
  S_2[i, j] -> B[j];
  S_2[i, j] -> C[i+j]; } * Domain;
Write := [N] -> { # Write access functions
  S_1[k] -> C[k];
  S_2[i, j] -> C[i+j]; } * Domain;
Theta := [N] -> { # Preliminary mapping
  S_1[k] -> [k, 0, 0];
  S_2[i, j] -> [i+j, i, 1]; };
```

Script ISCC

```
# Tools for set manipulations
Tiling := [s_1, s_2] -> { # Two dimensional tiling
  [[I_1, I_2] -> [i_1, i_2, k]] -> [i_1, i_2, k] :
      I_1 <= i_1 < I_1 + s_1 and I_2 <= i_2 < I_2 + s_2 };
Coalesce := { [I_1, I_2] -> [[I_1, I_2] -> [i_1, i_2, k]] };
Strip := { [I_1, I_2] -> [I_1, I_2'] };
Prev := { # Lexicographic order
  [[I_1, I_2] -> [i_1, i_2, k]] -> [[I_1, I_2] -> [i_1', i_2', k']] :
      i_1' <= i_1 - 1 or (i_1' <= i_1 and i_2' <= i_2 - 1)
      or (i_1' <= i_1 and i_2' <= i_2 and k' <= k - 1) };
TiledPrev := [s_1, s_2] -> { # Special ``lexicographic'' order
  [I_1, I_2] -> [I_1', I_2'] : I_1' <= I_1 - s_1 or
      (I_1' <= I_1 and I_2' <= I_2 - s_2) } * Strip;
TiledNext := TiledPrev^-1;
TiledRead  := Tiling.(Theta^-1).Read;
TiledWrite := Tiling.(Theta^-1).Write;
```

Script ISCC

```
# Set/relation computations
In  := Coalesce.(TiledRead - (Prev.TiledWrite));
Out := Coalesce.TiledWrite;
Load  := In - ((TiledPrev.In) + (TiledPrev.Out));
Store := Out - (TiledNext.Out);
print coalesce (Load % Params);
print coalesce (Store % Params);
```

Tailles mémoires

Schedule	Sequential Memory Size
jacobi-1d-imper	
$S_0(t,i) \mapsto (\underline{t, 2t+i}, 0)$ $S_1(t,j) \mapsto (\underline{t, 2t+j+1}, 1)$	$\mathtt{A}[2s_1 + s_2]$ $\mathtt{B}[2s_1 + s_2 - 1]$
jacobi-2d-imper	
$S_0(t,i,j) \mapsto (\underline{t, 2t+i, 2t+i+j}, 0)$ $S_1(t,i,j) \mapsto (\underline{t, 2t+i+1, 2t+i+j+1}, 1)$	$\mathtt{A}[2s_1 + s_2, \min(2s_1, s_2+1) + s_3]$ $\mathtt{B}[2s_1 + s_2 - 1, \min(2s_1, s_2) + s_3 - 1]$
seidel-2d	
$S_0(t,i,j) \mapsto (\underline{t, t+i, 2t+i+j})$	$\mathtt{A}\begin{bmatrix} s_1 + s_2 + 1, \\ \min(2s_1 + 2, s_1 + s_2, 2s_2 + 2) + s_3 \end{bmatrix}$
floyd-warshall	
$S_0(k,i,j) \mapsto (k, \underline{i, j})$	$\mathtt{path}\begin{bmatrix} \max(k+1, n-k), \\ \max(k+1, n-k) \end{bmatrix}$

Tailles mémoires

Schedule	Pipelined Memory Size
jacobi-1d-imper	
$S_0(t,i) \mapsto (\underline{t, 2t+i}, 0)$ $S_1(t,j) \mapsto (\underline{t, 2t+j+1}, 1)$	$\mathtt{A}[2s_1 + 2s_2]$ $\mathtt{B}[2s_1 + 2s_2 - 2]$
jacobi-2d-imper	
$S_0(t,i,j) \mapsto (\underline{t, 2t+i, 2t+i+j}, 0)$ $S_1(t,i,j) \mapsto (\underline{t, 2t+i+1, 2t+i+j+1}, 1)$	$\mathtt{A}[2s_1 + s_2, \min(2s_1, s_2+1) + 2s_3]$ $\mathtt{B}[2s_1 + s_2 - 1, \min(2s_1, s_2+1) + 2s_3 - 2]$
seidel-2d	
$S_0(t,i,j) \mapsto (\underline{t, t+i, 2t+i+j})$	$\mathtt{A}\begin{bmatrix} s_1 + s_2 + 1, \\ \min(2s_1+2, s_1+s_2, 2s_2+2) + 2s_3 \end{bmatrix}$
floyd-warshall	
$S_0(k,i,j) \mapsto (k, \underline{i, j})$	$\mathtt{path}\begin{bmatrix} \max(k+1, n-k), \\ \max(k+1, n-k, 2s_2) \end{bmatrix}$