Formation « Calcul parallèle et applications aux plasmas froids »

–

Présentation du problème de Laplace et méthodes numériques associées

Violaine Louvet [1]

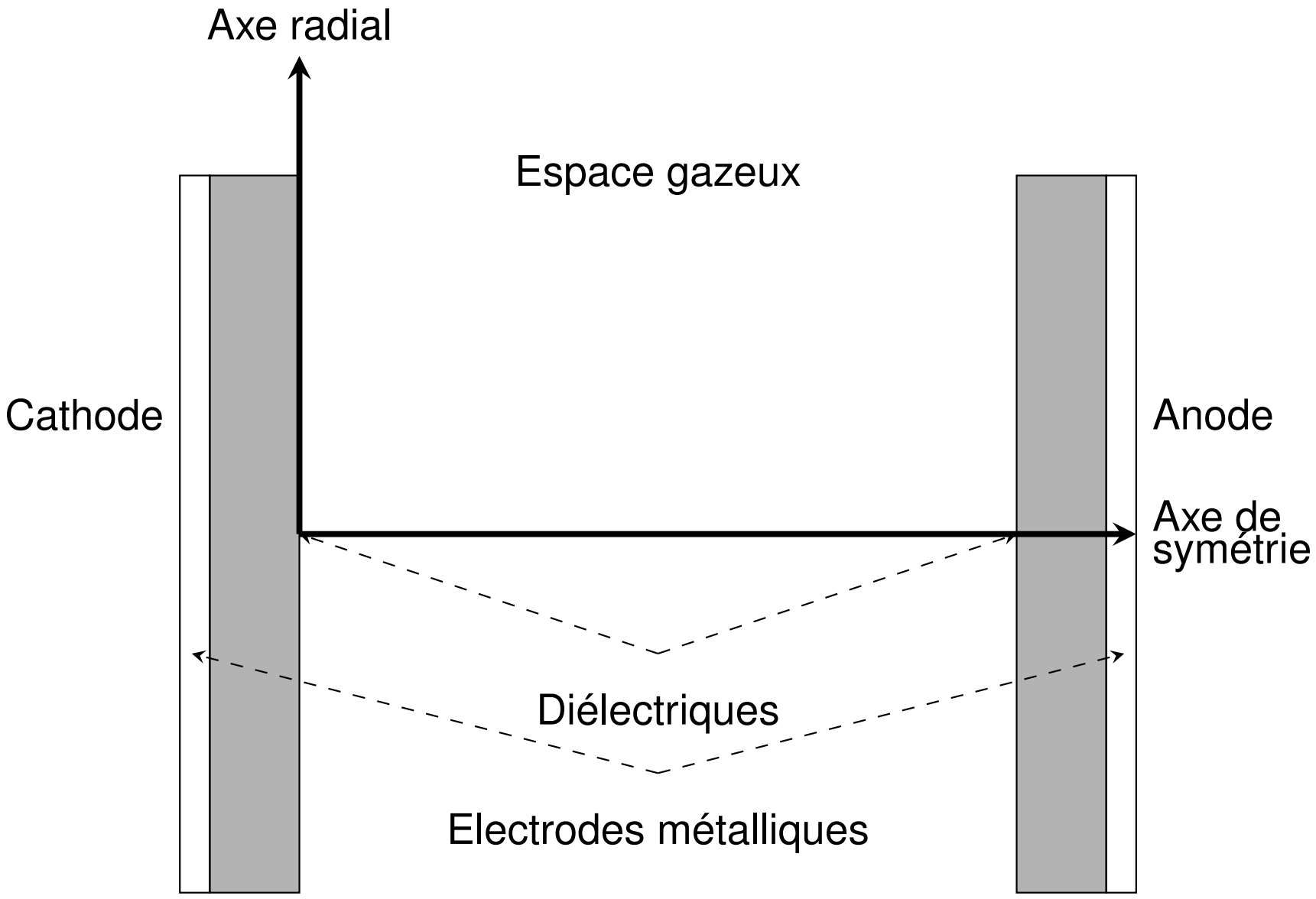
[1] Institut Camille jordan - CNRS

10-14/10/2011

Positionnement du problème

- Configuration de décharge à barrière diélectrique plan–plan en 2D
- Géométrie axisymétrique
- Cathode en $x = 0$ fixée à un potentiel $V_c = 0$
- Anode à droite en $x = 1$ fixée à un potentiel $V_a = 1$
- Les deux électrodes métalliques sont recouvertes d'un matériau diélectrique de permittivité $\varepsilon = 5$ et de $1mm$ d'épaisseur
- La permittivité de l'air dans l'espace entre les deux diélectriques est $\varepsilon = 1$

Configuration étudiée



Modélisation associée

- Résolution d'une équation de Laplace à $t = 0$ pour avoir la distribution de champ Laplacien
- Pendant la décharge, résolution dans tout le volume entre les deux électrodes métalliques de l'équation :

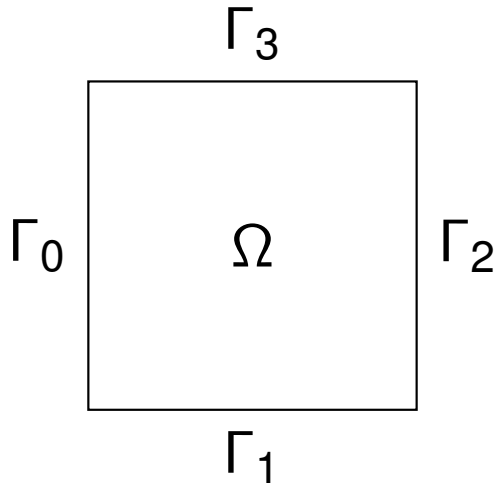
$$\frac{\partial}{\partial x}(\varepsilon\frac{\partial V}{\partial x}) + \frac{\partial}{\partial y}(\varepsilon\frac{\partial V}{\partial y}) = -q_e(n_p - n_n - n_e) + \sigma\delta_S$$

avec

- q_e : charge unitaire
- n_p, n_e, n_n : densités des ions positifs, des électrons et des ions négatifs dans l'air
- σ : charge déposée par la décharge de surface à l'interface air/diélectrique.

En résumé : Résolution de l'équation de Poisson 2D

$$\begin{cases} \dfrac{\partial}{\partial x}(\varepsilon \dfrac{\partial V}{\partial x}) + \dfrac{\partial}{\partial y}(\varepsilon \dfrac{\partial V}{\partial y}) = 0 & \text{sur } \Omega = [0,1] \times [0,1] \setminus \partial\Omega \\ V = V_c & \text{sur } \Gamma_0 = \{(x,y) \in \Omega; x = 0\} \\ V = V_a & \text{sur } \Gamma_2 = \{(x,y) \in \Omega; x = 1\} \\ \dfrac{\partial V}{\partial n} = 0 & \text{sur } \Gamma_1 = \{(x,y) \in \Omega; y = 0\} \\ & \text{et } \Gamma_3 = \{(x,y) \in \Omega; y = 1\} \end{cases}$$



Sommaire

Sommaire

Sommaire

Différentes méthodes de discrétisation

Différences finies (DF)

Les inconnues discrètes sont localisées aux sommets du maillage. On remplace les dérivées par des quotients différentiels.

Eléments finis (EF)

On écrit une formulation faible (variationnelle) de l'équation dans un espace fonctionnel adapté H. On cherche la solution par composition de fonctions (polynomiales) formant une base de l'espace discret approchant H.

Principes des Volumes finis

- On considère des équations de bilan par maille :

flux de masse sortant de la maille
-
flux de masse entrant dans la maille
=
création/consommation de la masse dans la maille

- Par intégration sur chaque maille
- La difficulté est d'approximer ces flux de la meilleure façon possible par des flux numériques
- Les inconnues discrètes s'interprètent comme des valeurs moyennes sur chaque maille

Cas du laplacien 1D, homogène et non-homogène

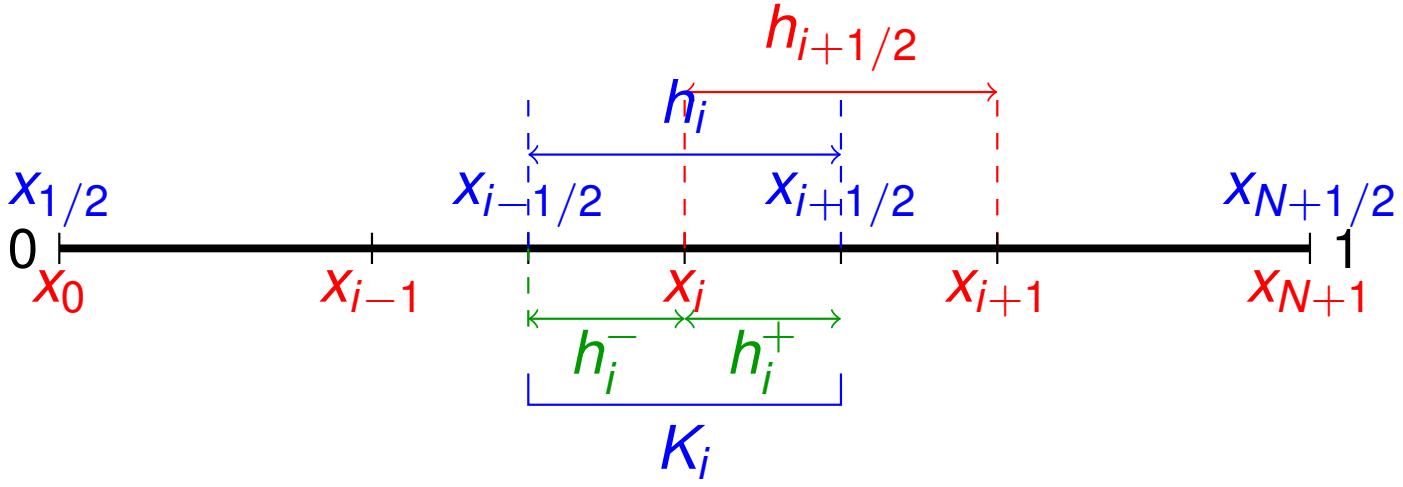
Laplacien homogène

$$\begin{cases} -u_{xx} = f(x) \quad sur \quad \Omega =]0,1[\\ u(0) = u(1) = 0 \end{cases}$$

Laplacien non homogène

$$\begin{cases} -(\varepsilon(x)u_x)_x = f(x) \quad sur \quad \Omega =]0,1[\\ u(0) = u(1) = 0 \end{cases}$$

Maillage



- Les inconnues sont les valeurs moyennes au centre des mailles $K_i = [x_{i-1/2}, x_{i+1/2}]$
- Les flux sont évalués aux arêtes en $x_{i-1/2}$, $x_{i+1/2}$

Volumes finis 1D pour le laplacien homogène

- On intègre sur K_i

$$-u_x(x_{i+1/2}) + u_x(x_{i-1/2}) = h_i f_i$$

avec

$$f_i = \frac{1}{h_i} \int_{K_i} f(x) dx$$

- On considère des flux numériques approximant de façon raisonnable les valeurs $u_x(x_{i+1/2})$ et $u_x(x_{i-1/2})$:

$$F_{i+1/2} - F_{i-1/2} = h_i f_i$$

- Approximation des flux :

$$F_{i+1/2} = \frac{u_{i+1} - u_i}{h_{i+1/2}}$$

- Système linéaire associé :

$$\frac{1}{h_i} \left\{ \left(\frac{1}{h_{i+1/2}} - \frac{1}{h_{i-1/2}} \right) u_i + \frac{1}{h_{i+1/2}} u_{i+1} + \frac{1}{h_{i-1/2}} u_{i-1} \right\} = f_i$$

Volumes finis 1D sur laplacien non homogène

- On intègre sur K_i

$$-\varepsilon(x_{i+1/2})u_x(x_{i+1/2}) + \varepsilon(x_{i-1/2})u_x(x_{i-1/2}) = h_i f_i$$

- On considère des flux numériques approximant de façon raisonnable les valeurs $\varepsilon(x_{i+1/2})u_x(x_{i+1/2})$ et $\varepsilon(x_{i-1/2})u_x(x_{i-1/2})$:

$$\bar{F}_{i+1/2} - \bar{F}_{i-1/2} = h_i f_i$$

- Les ε_i sont définis sur les mailles K_i.

Volumes finis 1D sur laplacien non homogène - suite

- Pour évaluer les $\bar{F}$, on utilise la conservation du flux numérique à l'interface :

$$\bar{F}^-_{i+1/2} = \bar{F}^+_{i+1/2}$$

avec

$$\begin{cases} \bar{F}^-_{i+1/2} = -\varepsilon_i \dfrac{u_{i+1/2} - u_i}{h^+_i} \\ \bar{F}^+_{i+1/2} = -\varepsilon_{i+1} \dfrac{u_{i+1} - u_{i+1/2}}{h^-_{i+1}} \end{cases}$$

- On élimine les $u_{i+1/2}$:

$$\bar{F}_{i+1/2} = -\frac{2\varepsilon_{i+1}\varepsilon_i}{h_i(\varepsilon_{i+1} + \varepsilon_i)}(u_{i+1} - u_i)$$

Laplacien 1D non homogène - système linéaire

- Système linéaire associé :

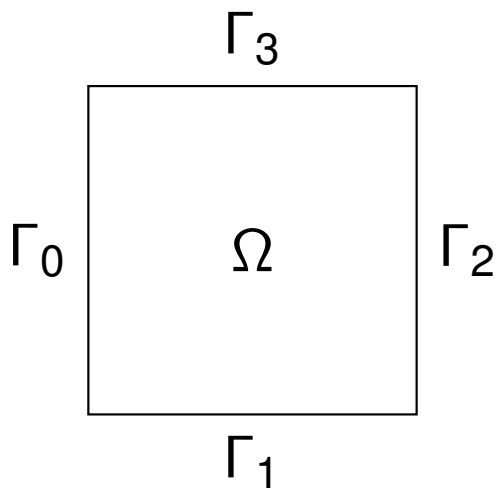
$$\frac{1}{h_i}\left\{\left(\frac{2\varepsilon_{i+1}\varepsilon_i}{h_i(\varepsilon_{i+1}+\varepsilon_i)}+\frac{2\varepsilon_i\varepsilon_{i-1}}{h_i(\varepsilon_i+\varepsilon_{i-1})}\right)u_i-\frac{2\varepsilon_{i+1}\varepsilon_i}{h_i(\varepsilon_{i+1}+\varepsilon_i)}u_{i+1}\right.$$

$$\left.-\frac{2\varepsilon_i\varepsilon_{i-1}}{h_i(\varepsilon_i+\varepsilon_{i-1})}u_{i-1}\right\}=f_i$$

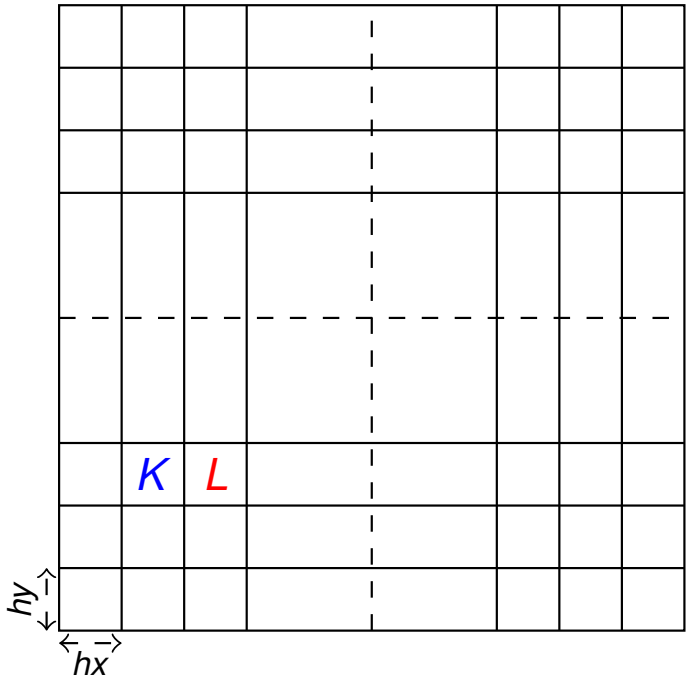
Sommaire

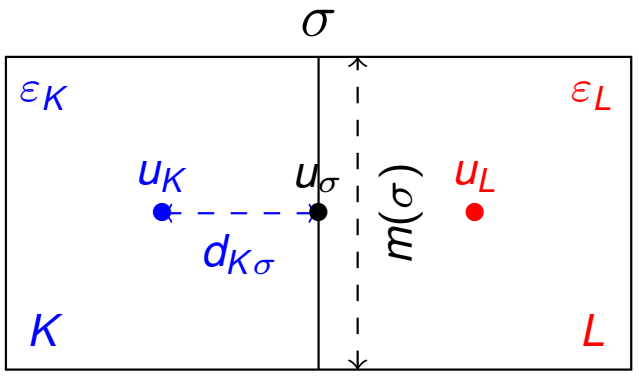
Positionnement du problème

$$\begin{cases} -div(\varepsilon \nabla u) = f \ \ sur \ \ \Omega = [0,1] \times [0,1] \setminus \partial\Omega \\ -\varepsilon \nabla u.\vec{n} = 0 \ \ sur \ \ \Gamma_1 \bigcup \Gamma_3 \\ u(x,y) = g(x,y) \ \ sur \ \ \Gamma_0 \bigcup \Gamma_2 \end{cases}$$



Maillage


K
L
hy
hx


σ
ε_K
ε_L
u_K
u_σ
u_L
$d_{K\sigma}$
$m(\sigma)$
K
L

Approximation Volumes Finis

- Bilan par maille :

$$\int_K -div(\varepsilon_K \nabla u) dxdy = \int_K f dxdy$$

- Formule de Stokes :

$$\int_{\partial K} -\varepsilon_K \nabla u.\vec{n} = m(K) f_K \text{ avec } m(K) = hx \times hy$$

- Bilan des flux pour la maille K :

$$\sum_{\sigma \in \tau_K} \int_\sigma -\varepsilon_K \nabla u.\vec{n}_\sigma d\sigma = m(K) f_K$$

Détermination des flux numériques

- Approximation du flux numérique :

$$F_{K\sigma} = -\varepsilon_K \frac{u_\sigma - u_K}{d_{K\sigma}} m(\sigma)$$

- Par conservativité du flux aux interfaces, on élimine la variable u_σ :

$$F_{K\sigma} = -\frac{\varepsilon_K \varepsilon_L}{d_{L\sigma}\varepsilon_K + d_{K\sigma}\varepsilon_L} m(\sigma)(u_L - u_K)$$

Système linéaire associé

- Flux au sud :

$$F_{K_{(i,j)}K_{(i,j-1)}} = -\frac{2\varepsilon_{(i,j)}\varepsilon_{(i,j-1)}}{hy\varepsilon_{(i,j)} + hy\varepsilon_{(i,j-1)}} hx(u_{(i,j-1)} - u_{(i,j)})$$

- Flux à l'est :

$$F_{K_{(i,j)}K_{(i+1,j)}} = -\frac{2\varepsilon_{(i,j)}\varepsilon_{(i+1,j)}}{hx\varepsilon_{(i,j)} + hx\varepsilon_{(i+1,j)}} hy(u_{(i+1,j)} - u_{(i,j)})$$

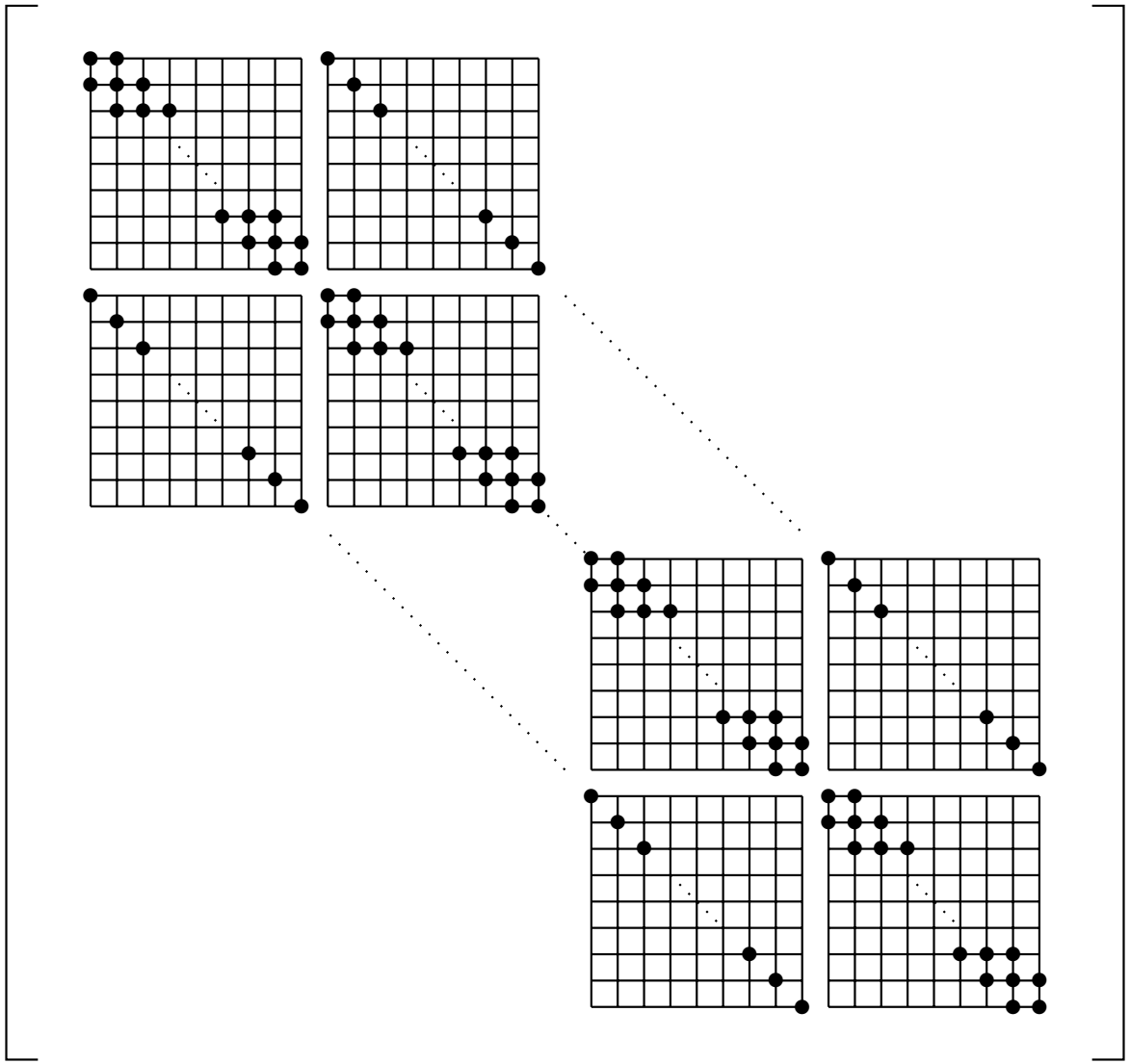
- Flux au nord :

$$F_{K_{(i,j)}K_{(i,j+1)}} = -\frac{2\varepsilon_{(i,j)}\varepsilon_{(i,j+1)}}{hy\varepsilon_{(i,j)} + hy\varepsilon_{(i,j+1)}} hx(u_{(i,j+1)} - u_{(i,j)})$$

- Flux à l'ouest :

$$F_{K_{(i,j)}K_{(i-1,j)}} = -\frac{2\varepsilon_{(i,j)}\varepsilon_{(i-1,j)}}{hx\varepsilon_{(i,j)} + hx\varepsilon_{(i-1,j)}} hy(u_{(i-1,j)} - u_{(i,j)})$$

Remplissage de la matrice



Sommaire

Méthodes de résolution de systèmes linéaires

- Matrice creuse ou dense ?
- Propriétés numériques (symétrique, définie, positive ...) et structurelles (réelle/complexe, par bandes, par blocs ...) ?
- Contexte : un seul système à résoudre, plusieurs systèmes, successivement, simultanément, dont les matrices sont proches ?
- Ordre de grandeur de la taille du problème par rapport aux capacités de traitements des CPU, des mémoires ?

Rappels

- 1 octet = 1 byte = 8 bits
- Réels simple précision stockés sur 32 bits (4 octets), double précision sur 64 bits (8 octets)
- $A(5000, 5000)$ en double précision = 200 Go

Méthodes directes, méthodes itératives

Méthodes directes $\longrightarrow$ matrices denses

- Robustes, aboutissent en un nombre fini d'opérations (théoriquement)
- Requièrent des capacités de stockage qui croissent rapidement avec la taille du problème : limite la scalabilité

Méthodes itératives $\longrightarrow$ matrices creuses

- Plus « scalables » lorsque l'on augmente le nombre de processeurs
- En pratique, leur convergence en un nombre « raisonnable » d'itérations, n'est pas toujours acquise (dépend de la structure de la matrice, du point de départ, du critère d'arrêt ...)
- Besoin en stockage moindre : on a juste besoin de l'action de la matrice sur un vecteur

Sommaire

Résolution par Jacobi

$$Ax = b$$

- Les éléments diagonaux doivent être non nuls
- Stockage double pour x : on ne peut pas en écraser les éléments
- Les éléments du nouvel itéré sont indépendants les uns des autres
- Taux de convergence lent

$$x_i^{(k+1)} = \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right) / a_{ii} \quad i = 1, .., n$$

Sommaire

Résolution par Gauss Seidel

$$Ax = b$$

- Accélération de la convergence de Jacobi en prenant en compte les nouveaux éléments de x au fur et à mesure de leur calcul
- Pas besoin de stockage double pour x : les éléments sont mis à jour au cours du calcul
- Nouvelles dépendances entre les éléments : ils doivent être évalués successivement

$$x_i^{(k+1)} = \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right) / a_{ii} \quad i = 1, .., n$$

Résolution par Relaxation

Formalisation

- $A = L + D + U$
- Décomposition de la matrice pour les méthodes itératives : $A = M - N$
- Méthodes itératives : $Mx^{(k+1)} = Nx^{(k)} + b$
  - Jacobi : $M = D$, $N = -(L + U)$
  - Gauss Seidel : $M = D + L$, $N = -U$
- La convergence vers $x = A^{-1}b$ dépend des valeurs propres de $M^{-1}N$

- Modification de l'algorithme de Gauss Seidel (cas où le rayon spectral de $M^{-1}N$ proche de 1)
- $0 < \omega < 2$
- Convergence plus rapide que GS si ω optimal

$$x^{(k+1)} = x^{(k)} + \omega(x_{GS}^{(k+1)} - x^{(k)})$$

Sommaire

Autres méthodes de résolution

Méthodes itératives

- Méthodes de Krylov :
  - gradient conjugué
  - GMRES
  - BiCG ...

Méthodes directes

- Factorisation de Cholesky
- Factorisation de Gauss ...

Sommaire

Sommaire

Problématiques

Architecture cible

- Type de parallélisme : partagé, distribué
- Structuration de la mémoire : hiérarchie, tailles (caches)
- Réseau : topologie, bande passante

Algorithmes

- Décomposition en tâches
- Dépendances des données et des tâches
- Equilibrage des calculs, répartition de la charge
- Minimisation des communications

Parallélisation des méthodes itératives

- Méthodes itératives composées d'opérations de base :
  - opérations linéaires sur les vecteurs (type saxpy)
  - produits scalaires
  - produits matrice-vecteur
- Certaines opérations (test de convergence) nécessitent des synchronisations entre les tâches

Partitionnement des vecteurs/matrices

- On découpe les vecteurs en parties homogènes
- Les matrices peuvent être partitionnée en colonnes, lignes ou blocs
  - le découpage par ligne ou colonne est souvent plus homogène (même nombre de zéros en général)

Sommaire

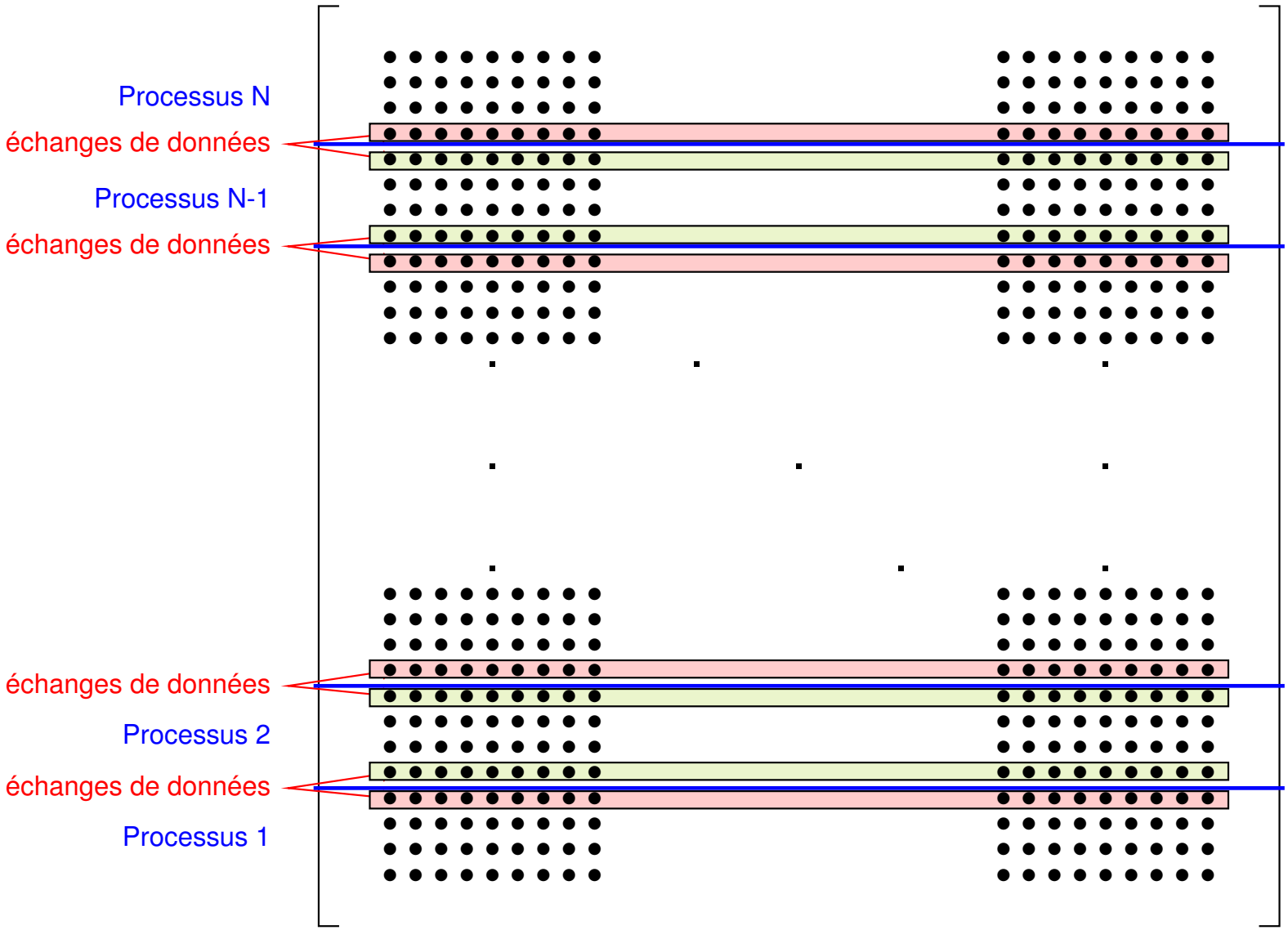
Jacobi parallèle

Algorithme

```
V given
while not convergence:
   V_old = V
   do j = 1,ny
      do i = 1,nx
         V(i,j) = (sou(i,j)
              + ve(i,j) * V_old(i+1,j) + vw(i,j) * V_old(i-1,j)
              + vn(i,j) * V_old(i,j+1) + vs(i,j) * V_old(i,j-1))
              * vc(i,j)
      end do
   end do
   compute err = norm(V - V_old)
end while
```

- Aucune dépendance au niveau de la boucle en i et j
- Dépendances uniquement vers des valeurs anciennes
- Les données doivent être à jour pour le calcul de l'erreur : synchronisation

Jacobi parallèle sur mémoire distribuée



Jacobi parallèle sur mémoire distribuée

Algorithme

```
V given
while not convergence:
    V_old = V
    send(V(nx_min,:) && V(nx_max,:))
    recv(V(nx_min-1,:) && V(nx_max+1,:))
    do j = 1,ny
        do i = nx_min,nx_max
            V(i,j) = (sou(i,j)
                + ve(i,j) * V_old(i+1,j) + vw(i,j) * V_old(i-1,j)
                + vn(i,j) * V_old(i,j+1) + vs(i,j) * V_old(i,j-1))
                * vc(i,j)
        end do
    end do
    compute err_loc = norm(V - V_old)
    send(err_loc)
    recv(convergence)
end while
```

Sommaire

Analyse du problème

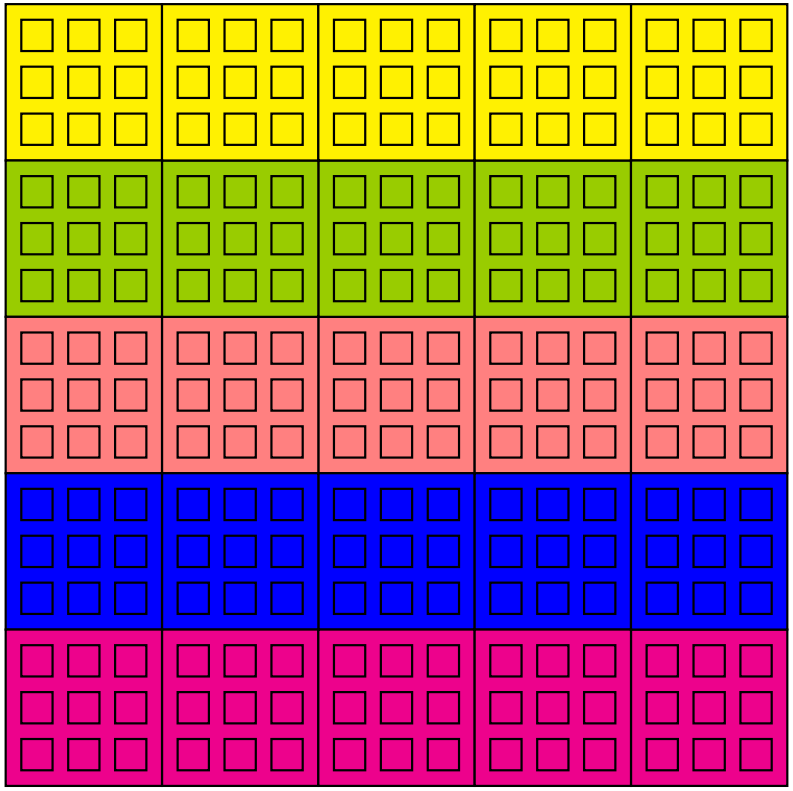
Algorithme

```
V given
while not convergence:
   V_old = V
   do j = 1,ny
      do i = 1,nx
         V(i,j) = (sou(i,j)
              + ve(i,j) * V(i+1,j) + vw(i,j) * V(i-1,j)
              + vn(i,j) * V(i,j+1) + vs(i,j) * V(i,j-1))
              * vc(i,j)
      end do
   end do
   compute err = norm(V - V_old)
end while
```

- Dépendances arrière en i et j : on a besoin des valeurs en $(i-1, j)$ et $(i, j-1)$ pour calculer les suivantes
- Aucune des 2 boucles n'est intrinsèquement parallèle !

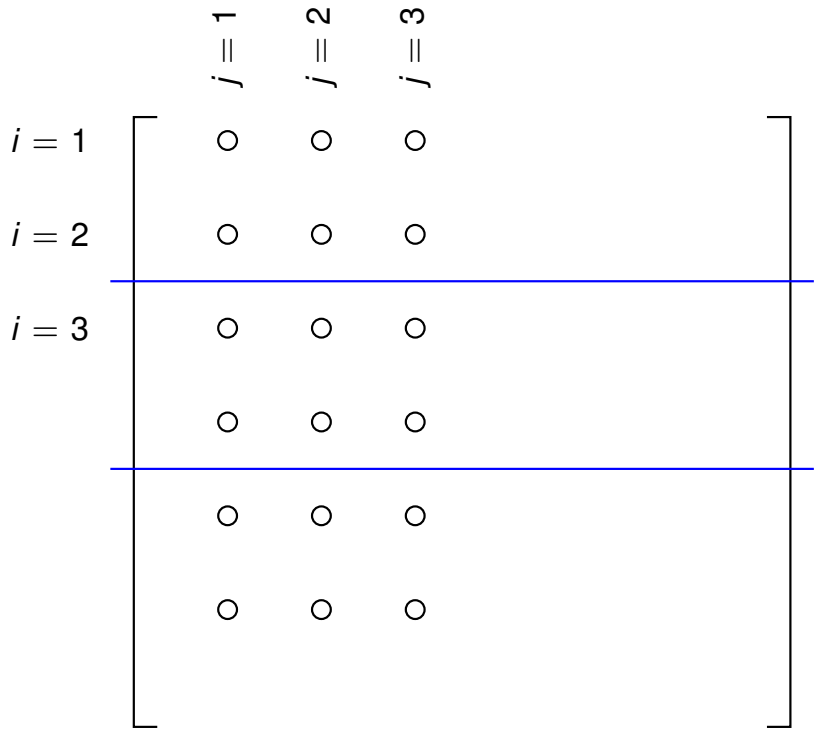
Algorithme software pipelining

- Principe : paralléliser par blocs la boucle la plus interne (i) et jouer sur les itérations de la boucle externe (j) pour ne pas casser les dépendances
- On découpe la matrice en tranches horizontales (par bloc de lignes)
- A chaque thread est attribué une tranche :



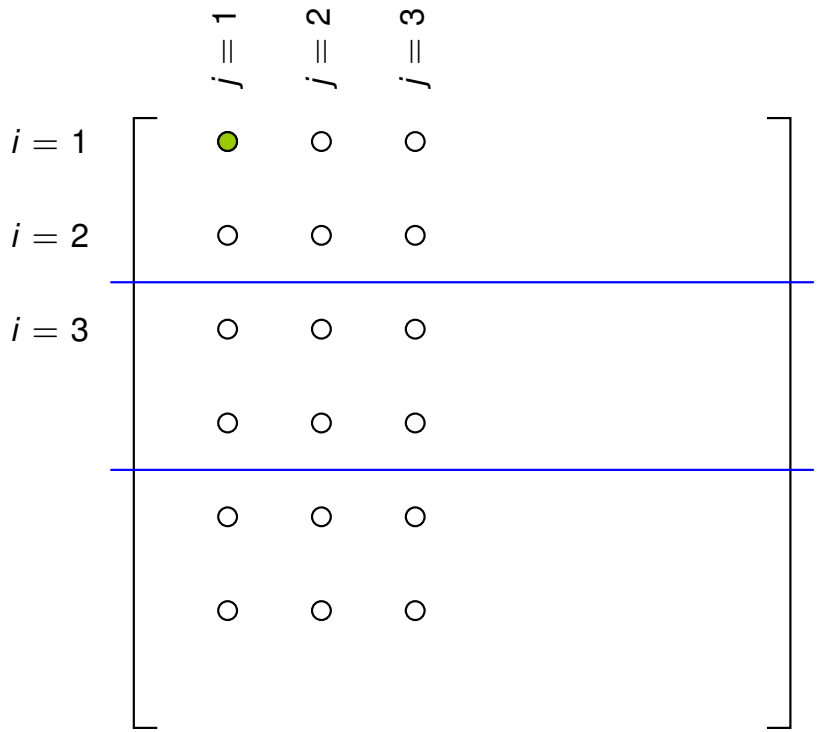
Algorithme software pipelining

- Les dépendances de l'algorithme font que le thread 0 doit toujours traiter une itération de la boucle externe en j qui doit être supérieure à celle du thread 1 etc.



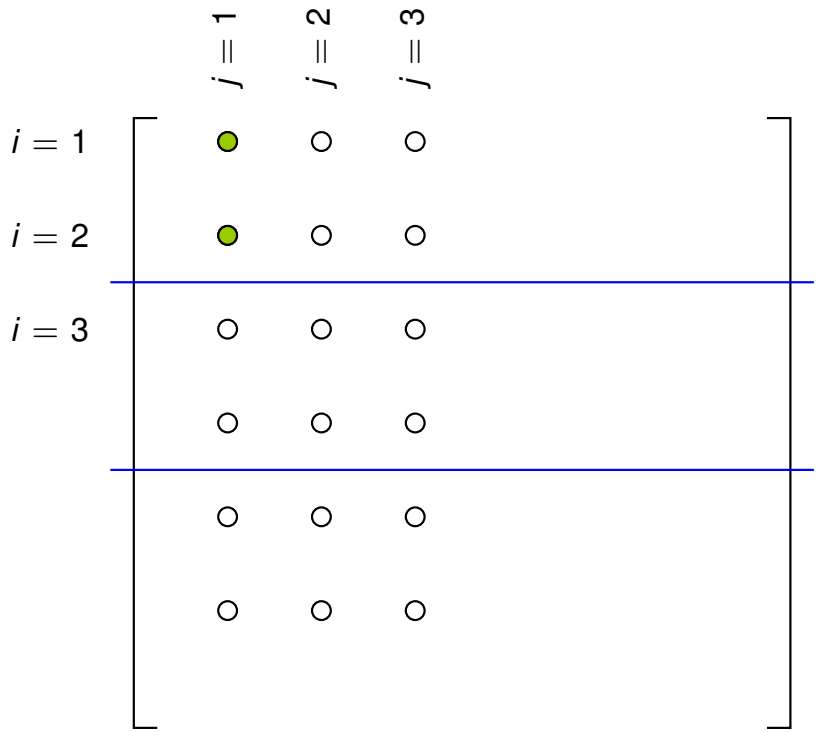
Algorithme software pipelining

- Les dépendances de l'algorithme font que le thread 0 doit toujours traiter une itération de la boucle externe en j qui doit être supérieure à celle du thread 1 etc.



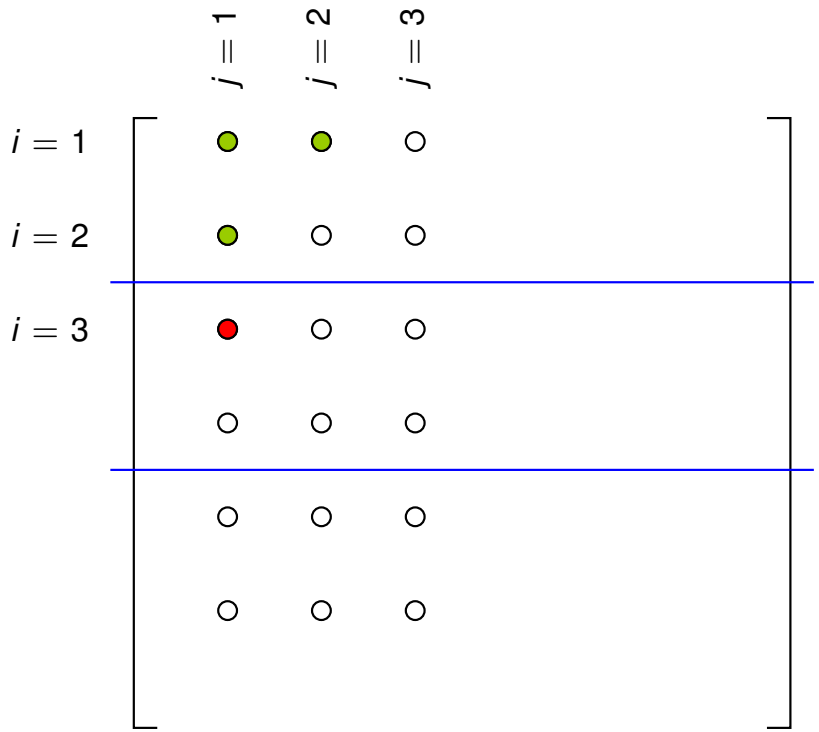
Algorithme software pipelining

- Les dépendances de l'algorithme font que le thread 0 doit toujours traiter une itération de la boucle externe en j qui doit être supérieure à celle du thread 1 etc.



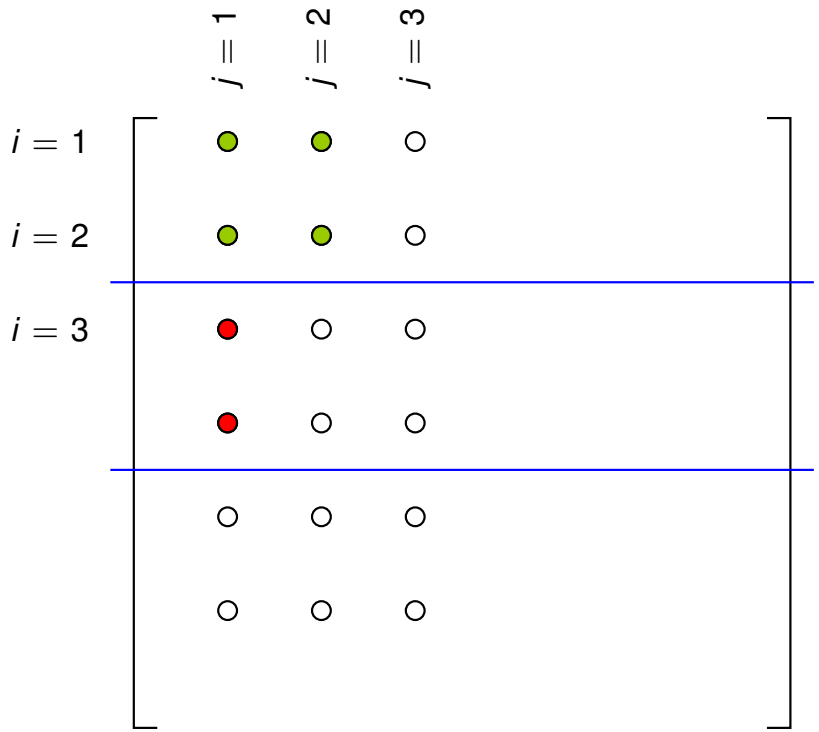
Algorithme software pipelining

- Les dépendances de l'algorithme font que le thread 0 doit toujours traiter une itération de la boucle externe en j qui doit être supérieure à celle du thread 1 etc.



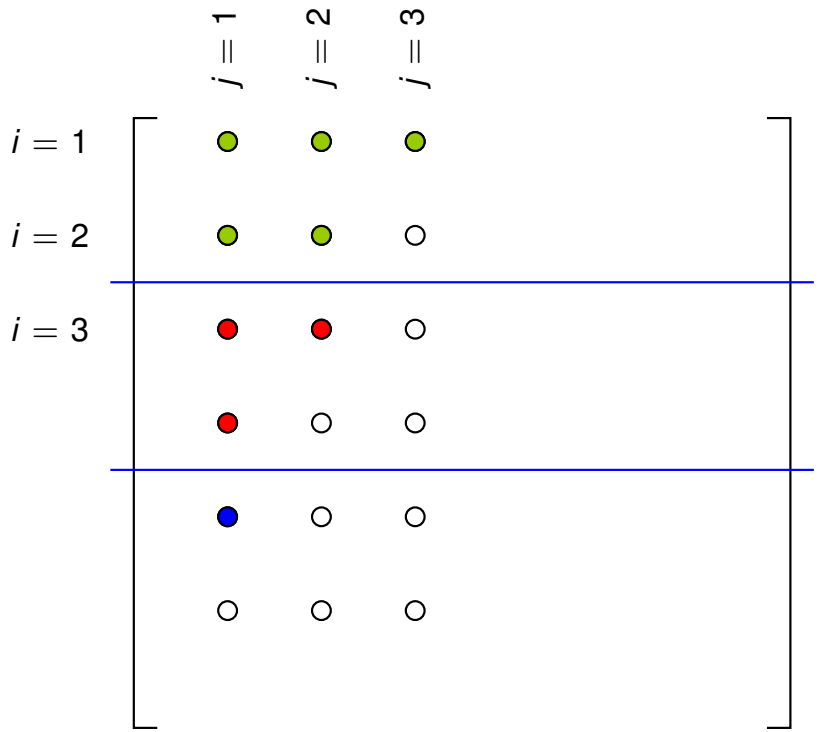
Algorithme software pipelining

- Les dépendances de l'algorithme font que le thread 0 doit toujours traiter une itération de la boucle externe en j qui doit être supérieure à celle du thread 1 etc.



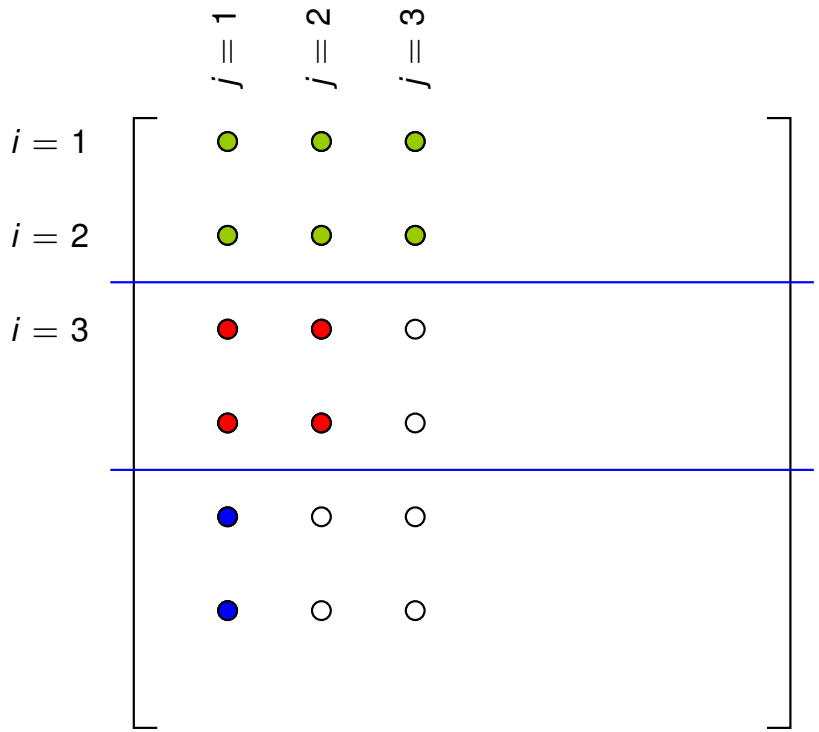
Algorithme software pipelining

- Les dépendances de l'algorithme font que le thread 0 doit toujours traiter une itération de la boucle externe en j qui doit être supérieure à celle du thread 1 etc.



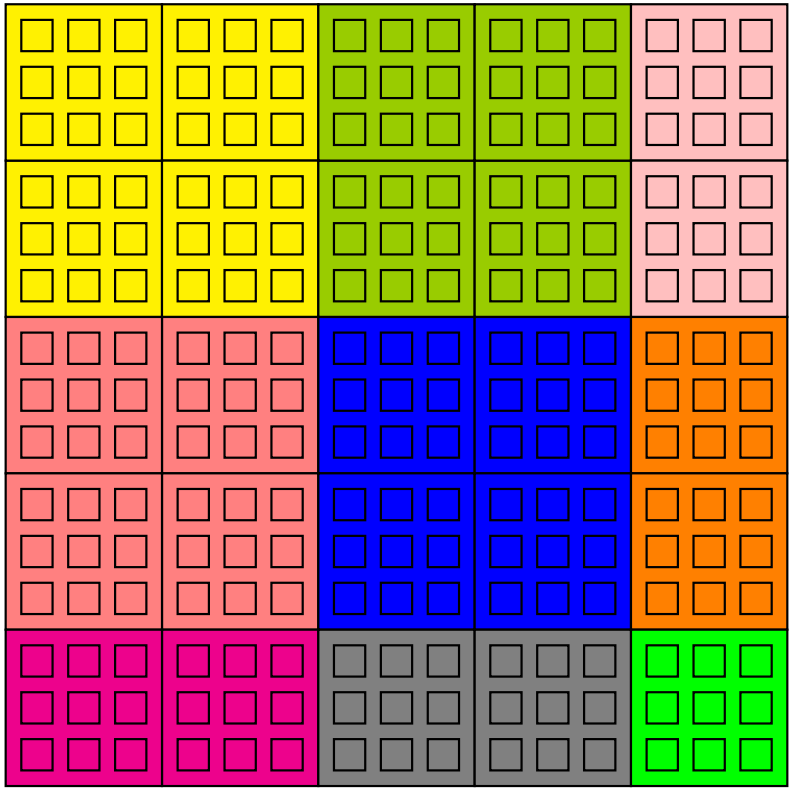
Algorithme software pipelining

- Les dépendances de l'algorithme font que le thread 0 doit toujours traiter une itération de la boucle externe en j qui doit être supérieure à celle du thread 1 etc.



Gauss Seidel sur architecture distribuée

- Le principe et les dépendances sont les mêmes
- On peut considérer un découpage du domaine selon une topologie cartésienne
- Chaque processus gère alors un sous-domaine, avec des dépendances à respecter avec les processus voisins

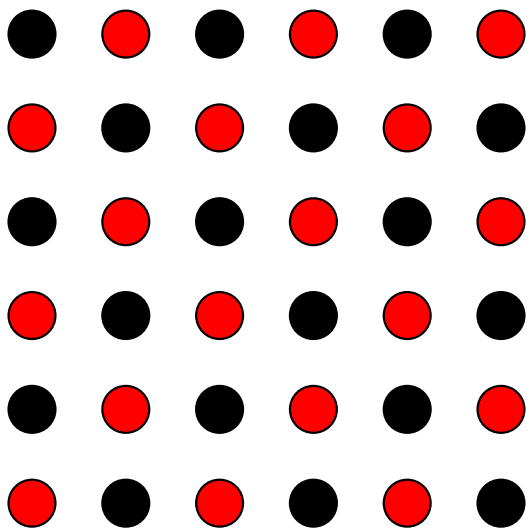


Red-Black Gauss Seidel

- Dans GS classique, on évalue les nouvelles valeurs dans l'ordre :

$$(1,1),(2,1),(3,1)...,(1,2),(2,2)...(nx-1,ny),(nx,ny)$$

- Dans Red-Black GS, renumérotation des noeuds pour limiter les dépendances :



- Les noeuds rouges ne dépendent que des noirs et vice-versa
- On traite les rouges en parallèles puis les noirs en parallèles

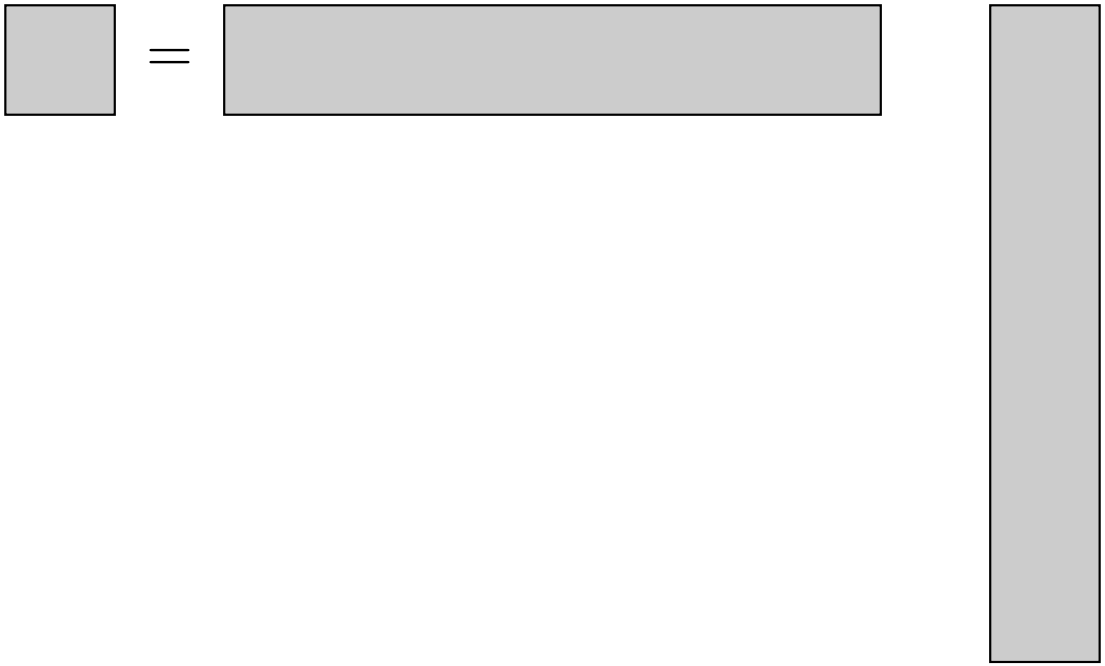
Sommaire

Calcul de l'erreur

- $\Leftrightarrow$ Produit scalaire

```
a = 0.0
do i = 1,n
  a = a + x(i) * y(i)
end do
```

- Toutes les opérations affectent le même scalaire *a*

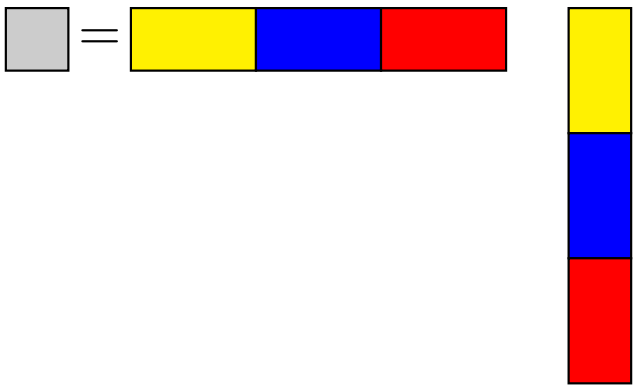


Calcul de l'erreur en parallèle

- Chaque thread/processus réalise le produit sur une portion des vecteurs
- Une opération de réduction impliquant l'ensemble des processus permet d'obtenir la valeur globale du produit scalaire
- Le résultat est ensuite transmis à tous les processus au travers d'une diffusion collective

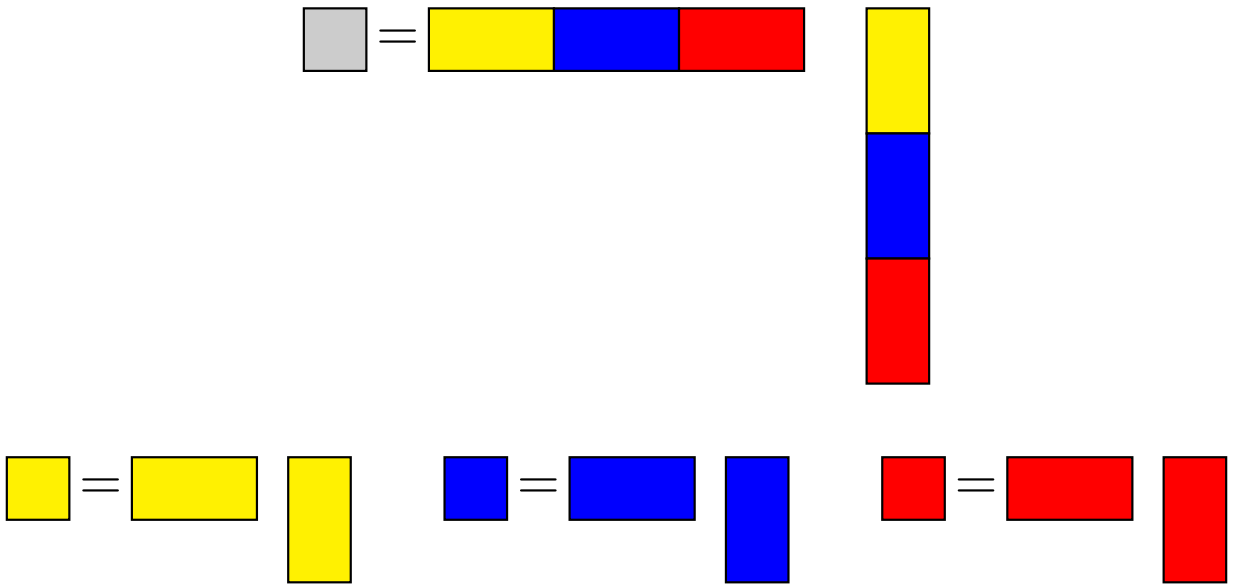
Calcul de l'erreur en parallèle

- On répartit le domaine sur les threads/processus



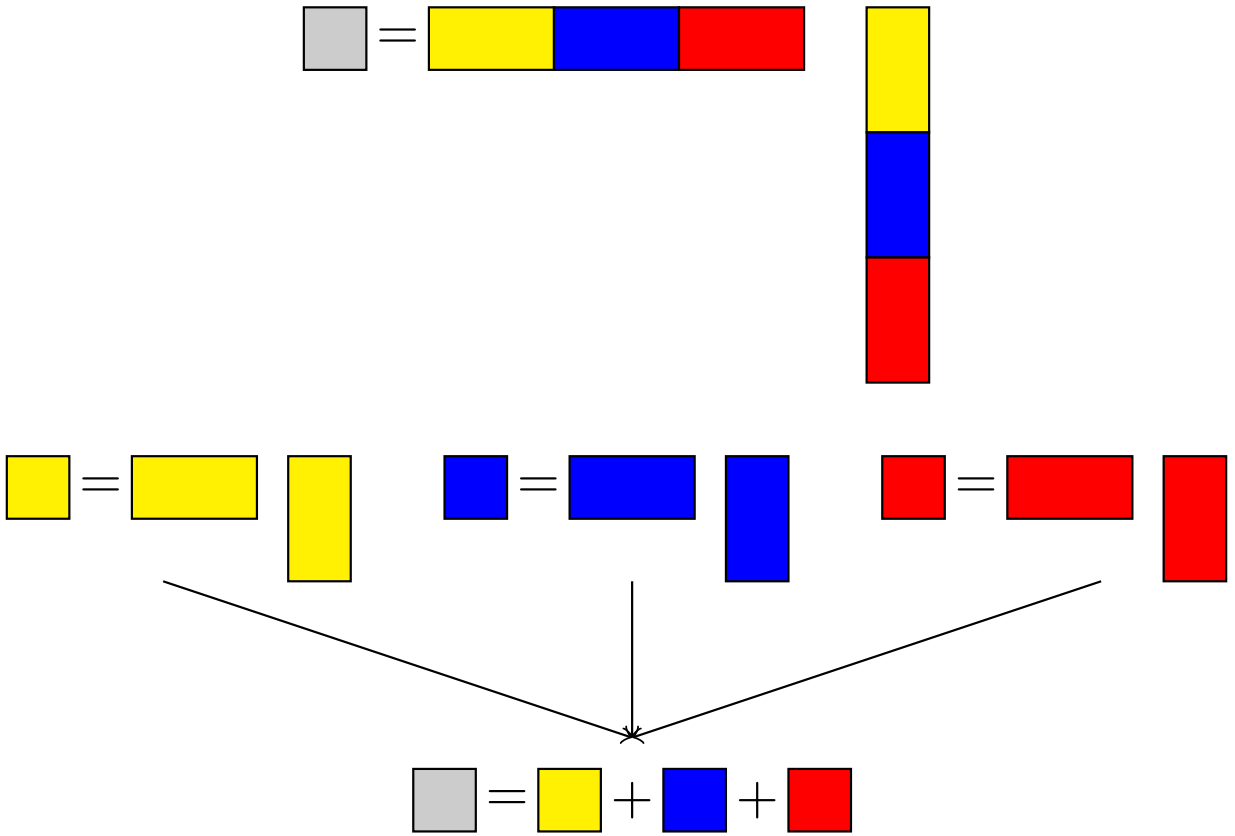
Calcul de l'erreur en parallèle

- Chaque thread/processus calcule le produit sur ses données



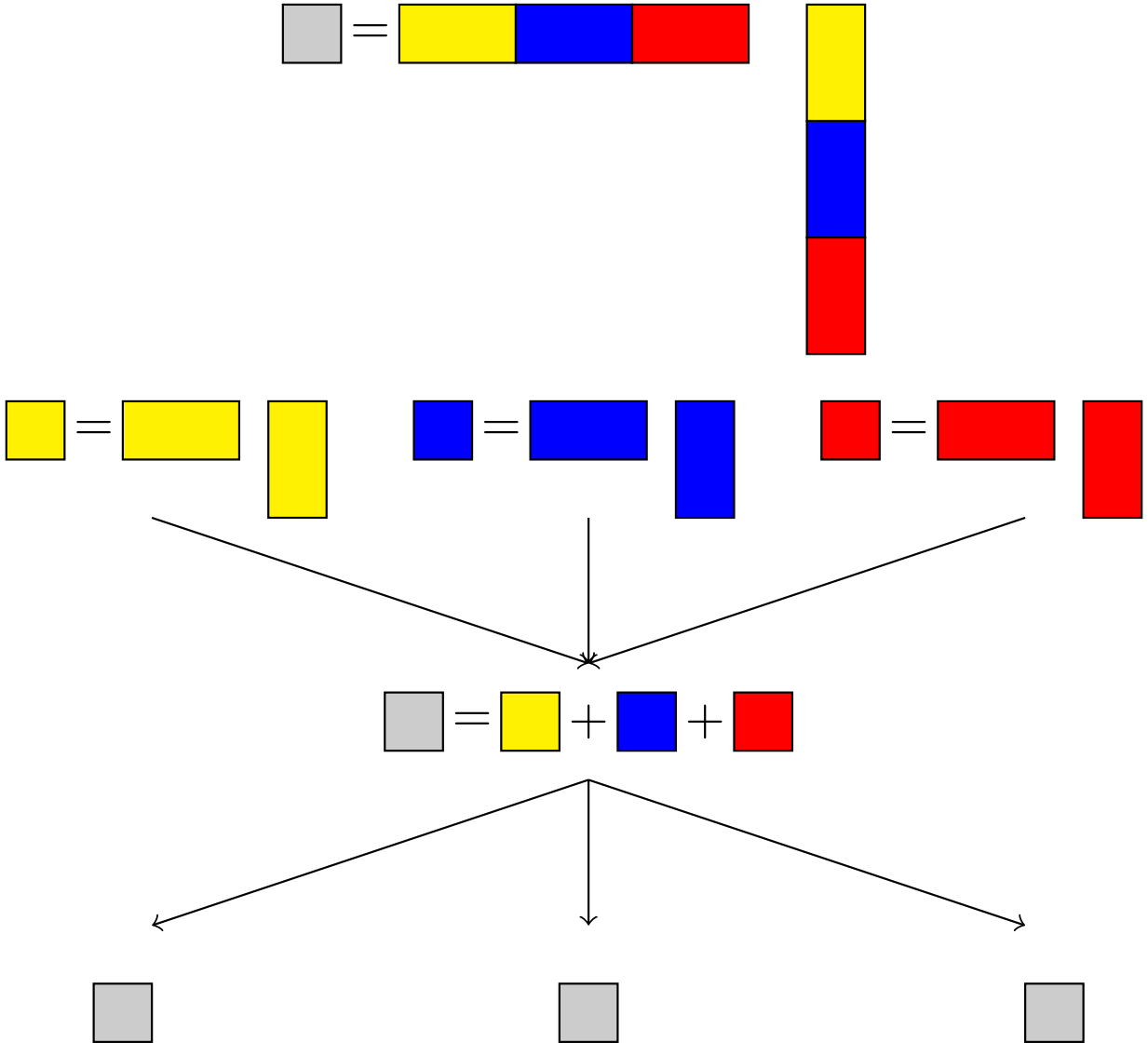
Calcul de l'erreur en parallèle

- On fait la réduction $\rightarrow$ communication collective



Calcul de l'erreur en parallèle

- On renvoie le résultat sur tous les processus $\rightarrow$ broadcast



Sommaire

Bibliothèques d'Algèbre Linéaire

http://www.netlib.org/utk/people/JackDongarra/la-sw.html

Sommaire

Quelques notions de décomposition de domaine

Idée

- Partitionner le domaine global en plusieurs sous-domaines
- Résoudre sur chaque sous-domaines des problèmes locaux quasi-indépendants de même nature que le problème initial
- Utiliser le même solveur pour tous ces problèmes locaux
- Faire la jointure des solutions locales à la frontière des domaines via un solveur d'interface ad hoc : inversion du complément de Schur, ou opérateur de type FETI

Deux grandes familles de méthodes

- Méthodes avec recouvrement (méthodes de Schwarz)
- Méthodes sans recouvrement (méthodes de Schur)

Pour aller plus loin

Ecole thématique du GDR Calcul :

« Méthodes de décomposition de domaine : de la théorie à la pratique »

`http://calcul.math.cnrs.fr/spip.php?article183`

Conclusions

But de ce cours :

- Comprendre les différents éléments présents dans le modèle : discrétisation spatiale, résolution du système
- Appréhender les difficultés liées au parallélisme : « Penser parallèle » : tâches, dépendances ...

Conclusions

But de ce cours :

- Comprendre les différents éléments présents dans le modèle : discrétisation spatiale, résolution du système
- Appréhender les difficultés liées au parallélisme : « Penser parallèle » : tâches, dépendances ...
- Vous êtes prêts à mettre les mains dans le code ! !