Algorithmique pour l'algèbre linéaire creuse

Introduction aux méthodes itératives

Pascal Hénon

5 janvier 2009

Plan

Plan

1. Introduction aux méthodes de Krylov préconditionnées
2. Factorisations incomplètes
3. Utilisation du complément de Schur

Introduction aux méthodes de Krylov préconditionnées

Méthode de Krylov :

On cherche une approximation de la solution de A.x = b dans un sous-espace "de Krylov" :

$$K(m, b) = span\{b, A.b, .., A^{m-1}.b\}$$

$$x_m = \alpha_1.b + \alpha_2.A.b + .. + \alpha_{m-1}.A^{m-1}.b$$

telle que $b - A.x_m \perp \mathcal{L}_m$ (condition de Petrov-Galerkin) ; $\mathcal{L}_m$ est un sous espace vectoriel de dimension m (ex : $\mathcal{L}_m = A.K_m$)

Ceci revient à dire que l'on cherche un polynôme $\mathcal{P}$ tel que :

$$A^{-1}.b \approx \mathcal{P}(A).b$$

Introduction aux méthodes de Krylov préconditionnées

Il existe beaucoup de méthodes itératives de Krylov. Les variations résident généralement dans :

- le choix de $\mathcal{L}_m$,
- le calcul d'une base pour $K(m,b)$,
- Restart : on réinitialise la base de Krylov toutes les s itérations.

exemple : GMRES ($\mathcal{L}_m = A.K_m$), Gradient Conjugué ($\mathcal{L}_m = K_m$), BICG ($\mathcal{L}_m = K_{A^t}(b,m)$), ...

Référence

Iterative Methods for Sparse Linear Systems. Yousef Saad

Introduction aux méthodes de Krylov préconditionnées

Méthodes de Krylov :

Trouver itérativement une solution approximée (ex : dans Krylov, l'espace de recherche est augmenté à chaque itération)

La convergence dépend du nombre de conditionnement :

$$||A||_p||A^{-1}||_p$$

Méthode de préconditionnement : On transforme le système à l'aide d'un opérateur linéaire M tel que le nouveau système ait un meilleur conditionnement.

Plusieurs variantes sont possibles :

- précond. à gauche : $M^{-1}.A.x = M^{-1}.b$;
- précond. à droite : $A.M^{-1}u = b$, $x = M^{-1}.u$;
- précond. découpé (cas sym.) : $M^{-1}.A.M^{-t}.u = M^{-1}.b$, $x = M^{-1}.u$;

Introduction aux méthodes de Krylov préconditionnées

- En règle générale, on ne stocke pas $M^{-1}.A$ sous forme matricielle on utilise $(M^{-1}.A).x = M^{-1}.(A.x)$. Pour que ce soit rentable il faut que $M^{-1}.y$ soit peu coûteux à calculer.
- Usuellement, on cherche M proche de A (i.e. $M^{-1} \approx A^{-1}$).
- M n'est pas nécessairement une matrice ; par exemple $M^{-1}.y$ peut être obtenu par un autre processus itératif.
- Si M varie à chaque itération, on utilise alors une methode dite "flexible".
- On va s'intéresser au préconditionneur utilisant une technique de factorisation incomplète. $M^{-1} :\simeq A^{-1} \approx U^{-1}.L^{-1}$.

Plan

- Factorisation directe : $A = L.U$
- Factorisation incomplète : $A = L.U + R$

On peut choisir d'imposer des contraintes sur R (remplissage, norme).

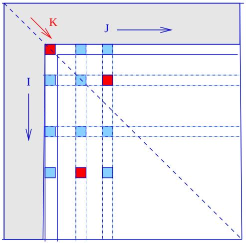
Principe générale d'une factorisation incomplète

On distingue généralement 2 grandes familles :

- les factorisations dont le remplissage est déterminé à l'avance (de manière statique) ; ces méthodes consistent à déterminer un schéma de remplissage à partir du graphe d'adjacence $G(V, E)$ de la matrice creuse A.
- les factorisations dont le remplissage est déterminé au cours de la factorisation (de manière dynamique) ; ces méthodes consistent à ne garder au cours de la factorisation que les termes vérifiant un certain critère numérique (généralement une norme suffisamment élevée).

Factorisation ILU avec remplissage prédéfini

pour $(i,j) \in \mathcal{P}$ **faire** $a_{ij} := 0$
pour $k = 1$ *à* $n-1$ **faire**
    pour $i = k+1$ *à* n *tq* $(i,k) \notin \mathcal{P}$ **faire**
        $a_{ik} := a_{ik}/a_{kk}$
        pour $j = k+1$ *à* n *tq* $(i,j) \notin \mathcal{P}$ **faire**
            $a_{ij} := a_{ij} - a_{ik} * a_{kj}$
        fin
    fin
fin



Factorisation ILU avec remplissage prédéfini

- Généralement, on construit $\mathcal{P}$ en se basant uniquement sur le graphe d'adjacence $G(V, E)$ de A (valeurs numériques pas prises en compte) ;
- Exemple le plus classique : ILU(0), IC(0),
  pour $i > j$ $A_{i,j} = 0 => L_{ij} = 0$ (idem pour U).

Caractérisation du remplissage dans les méthodes directes

Théorème

Un terme nul a_{ij} *de* A *deviendra non nul dans la factorisation* $A = L.U$ *ssi il existe un chemin allant du sommet* i *au sommet* j *en ne passant que par des sommets de plus petit numéro que* i *et* j *dans le graphe* $G(V, E)$ *d'adjacence de* A*.*

Caractérisation du remplissage en ILU(k)

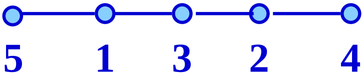
Théorème

Un terme nul a_{ij} de A deviendra non nul dans la factorisation ILU(k) de A ssi il existe un plus court chemin de longeur $k+1$ allant du sommet i au sommet j en ne passant que par des sommets de plus petit numéro que i et j dans le graphe $G(V,E)$ d'adjacence de A.

Factorisation ILU(k)

Au cours du processus d'élimination, le niveau de remplissage d'un coefficient a_{ij} est défini par : Initialisalition : $\mathrm{lev}(a_{ij}) = 0$ si $a_{ij} \neq 0$; sinon $\mathrm{lev}(a_{ij}) = \inf$

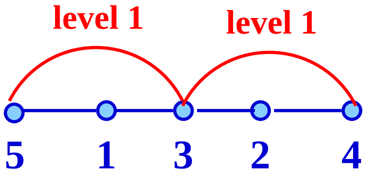
$$
\begin{array}{cc}
 & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{bmatrix}
\times & & \times & & \times \\
 & \times & \times & \times & \\
\times & \times & \times & & \\
 & \times & & \times & \\
\times & & & & \times
\end{bmatrix}
\end{array}
$$



Factorisation ILU(k)

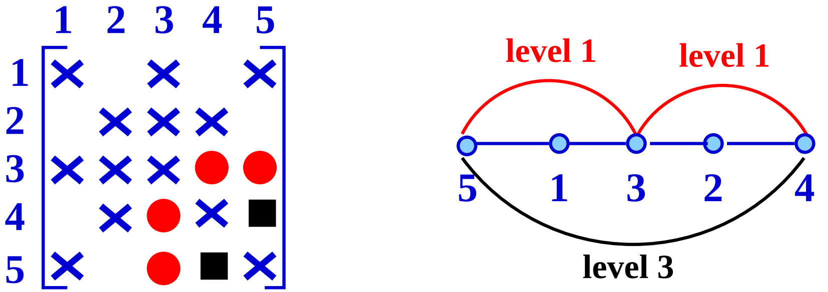
Au cours du processus d'élimination, le niveau de remplissage d'un coefficient a_{ij} est défini par : $\text{lev}(a_{ij}) = \min(\text{lev}(a_{ij}), \text{lev}(a_{ik}) + \text{lev}(a_{kj})+1)$;

$$
\begin{array}{c c}
 & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{bmatrix}
\times & & \times & & \times \\
 & \times & \times & \times & \\
\times & \times & \times & \bullet & \bullet \\
 & \times & \bullet & \times & \\
\times & & \bullet & & \times
\end{bmatrix}
\end{array}
$$



Factorisation ILU(k)

Au cours du processus d'élimination, le niveau de remplissage d'un coefficient a_{ij} est défini par :



Factorisation ILU(k)

`Initialisation`

pour *tous les termes de A* **faire**
- **si** $a_{ij} \neq 0$ **alors** $\text{lev}(a_{ij}) := 0$
- **si** $a_{ij} = 0$ **alors** $\text{lev}(a_{ij}) := \infty$

pour $k = 1$ *à* $n - 1$ **faire**
- **pour** $i = k + 1$ *à* n **faire**
  - **si** $lev(a_{ik}) \leq p$ **alors**
    - $a_{ik} = a_{ik} / a_{kk}$
    - **pour** $j = k + 1$ *à* n **faire**
      - $\text{lev}(a_{ij}) = \min(\text{lev}(a_{ij}), \text{lev}(a_{ik}) + \text{lev}(a_{kj}) + 1)$
      - **si** $lev(a_{ij}) \leq p$ **alors**
        - $a_{ij} := a_{ij} - a_{ik} * a_{kj}$

Factorisation symbolique ILU(k) (classique)

Initialisation
pour *tous les termes de A* **faire**
    si $a_{ij} \neq 0$ **alors** $\text{lev}(a_{ij}) := 0$
    si $a_{ij} = 0$ **alors** $\text{lev}(a_{ij}) := \infty$
pour $k = 1$ *à* $n - 1$ **faire**
    pour $i = k + 1$ *à* n **faire**
        si $lev(a_{ik}) \leq p$ **alors**
            pour $j = k + 1$ *à* n **faire**
                $\text{lev}(a_{ij}) = \min(\text{lev}(a_{ij}), \text{lev}(a_{ik}) + \text{lev}(a_{kj}) + 1)$
                si $lev(a_{ij}) \leq p$ **alors**
                    on ajoute (i, j) dans la structure des nz de L, U

Factorisation symbolique ILU(k)

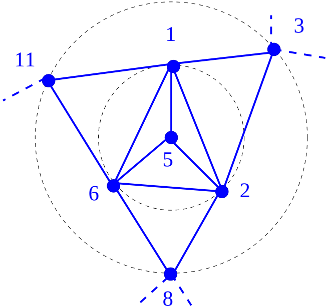
Facto. symbolique à partir du graphe d'adj. de A

On peut aussi effectuer la factorisation symbolique en recherche tous les chemins d'élimination de longueurs $k+1$.

D. Hysom and A. Pothen. "A scalable parallel algorithm for incomplete factor preconditioning" SIAM SISC, 22(6) :2194–2215, 2001.

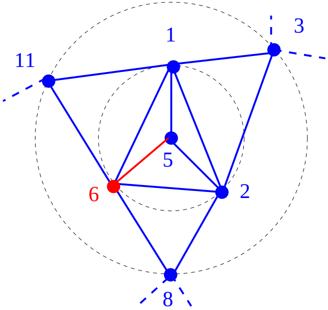
Factorisation symbolique ILU(k)

$Q \longleftarrow \{i\}$
length $[i] \longleftarrow 0$
visited $[i] \longleftarrow i$
tant que $Q \neq \emptyset$ **faire**
    $h \longleftarrow$ `Depile(`Q`)`
    pour $t \in \text{adj}(h)$ *avec* **visited** $t \neq i$ **faire**
        visited$[t] \longleftarrow i$
        si $t < i$ *et* length $[h] < l$ **alors**
            `Empile(`Q, t`)`
            length $[t] \longleftarrow$ length $[h] + 1$
        fin
        si $t > i$ **alors** on insère t dans new_adj(i)
    fin
fin



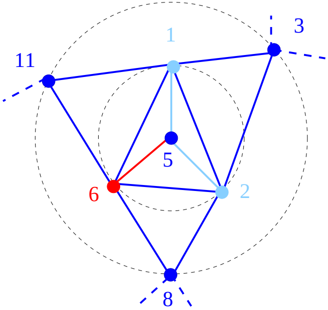
Factorisation symbolique ILU(k)

$Q \longleftarrow \{i\}$
length $[i] \longleftarrow 0$
visited $[i] \longleftarrow i$
tant que $Q \neq \emptyset$ **faire**
  $h \longleftarrow$ `Depile`(Q)
  pour $t \in$ adj(h) *avec* **visited** $t \neq i$ **faire**
    visited$[t] \longleftarrow i$
    si $t < i$ *et* length $[h] < l$ **alors**
      `Empile`(Q, t)
      length $[t] \longleftarrow$ length $[h]$ $+1$
    fin
    si $t > i$ **alors** on insère t dans new_adj(i)
  fin
fin



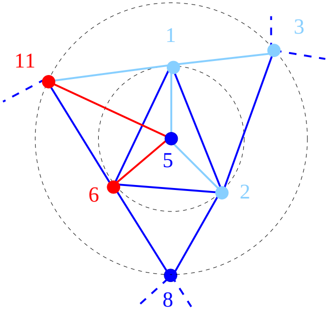
Factorisation symbolique ILU(k)

$Q \longleftarrow \{i\}$
length $[i] \longleftarrow 0$
visited $[i] \longleftarrow i$
tant que $Q \neq \emptyset$ **faire**
    $h \longleftarrow$ `Depile`(Q)
    pour $t \in \mathrm{adj}(h)$ *avec* **visited** $t \neq i$ **faire**
        visited$[t] \longleftarrow i$
        si $t < i$ *et* length $[h] < l$ **alors**
            `Empile`(Q, t)
            length $[t] \longleftarrow$ length $[h] + 1$
        fin
        si $t > i$ **alors** on insère t dans new_adj(i)
    fin
fin



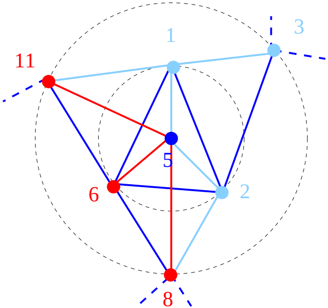
Factorisation symbolique ILU(k)

$Q \longleftarrow \{i\}$
length $[i] \longleftarrow 0$
visited $[i] \longleftarrow i$
tant que $Q \neq \emptyset$ **faire**
  $h \longleftarrow$ `Depile(`Q`)`
  pour $t \in \text{adj}(h)$ *avec* **visited** $t \neq i$ **faire**
    visited$[t] \longleftarrow i$
    si $t < i$ *et* length $[h] < l$ **alors**
      `Empile(`Q, t`)`
      length $[t] \longleftarrow$ length $[h] + 1$
    fin
    si $t > i$ **alors** on insère t dans new_adj(i)
  fin
fin



Factorisation symbolique ILU(k)

$Q \longleftarrow \{i\}$
length $[i] \longleftarrow 0$
visited $[i] \longleftarrow i$
tant que $Q \neq \emptyset$ **faire**
    $h \longleftarrow$ `Depile(`Q`)`
    pour $t \in$ adj(h) *avec* **visited** $t \neq i$ **faire**
        visited$[t] \longleftarrow i$
        si $t < i$ *et* length $[h] < l$ **alors**
            `Empile(`Q, t`)`
            length $[t] \longleftarrow$ length $[h]$ $+1$
        fin
        si $t > i$ **alors** on insère t dans new_adj(i)
    fin
fin

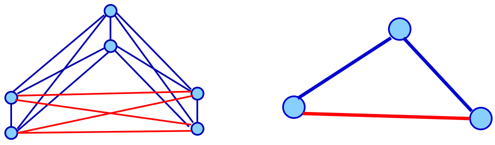


Factorisation symbolique ILU(k)

A la fin de l'algorithme new_adj(i) contient les indices des termes non nuls dans la colonnes i. L'intérêt de cet algorithm est qu'il permet de calculer indépendamment la structure de toutes les colonnes.

Factorisation symbolique par blocs pour ILU(k)

- La factorisation symbolique par blocs pour le directe est basée sur : $Q(G,P) \longrightarrow Q(G,P)^* = Q(G^*,P)$.
- Existe-t-il une partition telle que $Q(G,P) \longrightarrow Q(G,P)^k = Q(G^k,P)$?
- Il en existe au moins 1 type : lorsque le graphe de A peut se quotienter “naturellement”.



Factorisation symbolique par blocs pour ILU(k)

- Ex : dans une discrétisation de type éléments finis, un nœud du maillage représente plusieurs inconnues (ex : degrés de liberté) couplés entre eux ; ainsi chaque nœud du maillage représente une clique dans $G(V, E)$ le graphe d'adj. de A. On prend alors P la partition des nœuds.
- La factorisation symbolique a, dans ce cas, une complexité fonction du nombre de nœuds du maillage et non du nombre d'inconnues (au contraire de la factorisation numérique).

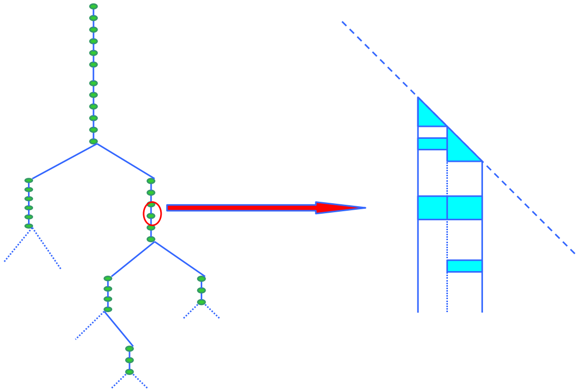
Factorisation ILU(k) par blocs

L'utilisation d'une structure bloc-dense dans les facteurs permet l'utilisation des routines BLAS niveau 3 (factorisation) et BLAS niveau 2 (résolution).
Si la taille initiale des blocs n'est pas suffisante on peut utiliser un algorithme d'amalgamation (ILU(k) par blocs dans PaStiX).

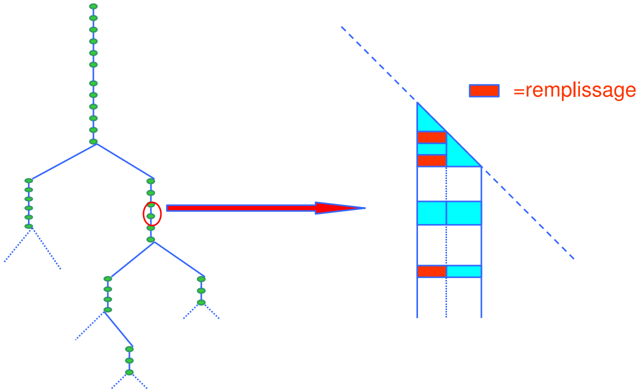
Algorithme d'amalgamation

Le principe est d'autoriser une certaine tolérance de remplissage supplémentaire (éventuellement stocker des zéros) pour augmenter la taille des blocs denses (augmentation des performances des routines BLAS).

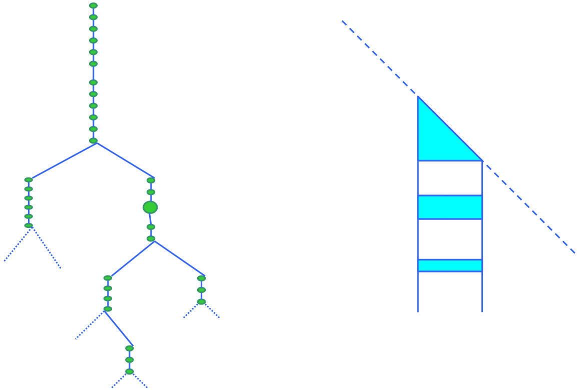
Algorithme d'amalgamation



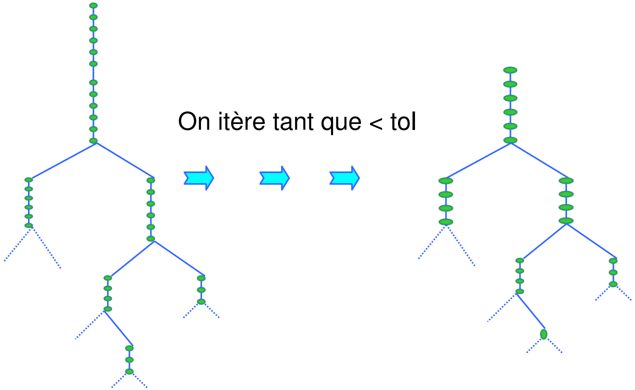
Algorithme d'amalgamation



Algorithme d'amalgamation



Algorithme d'amalgamation

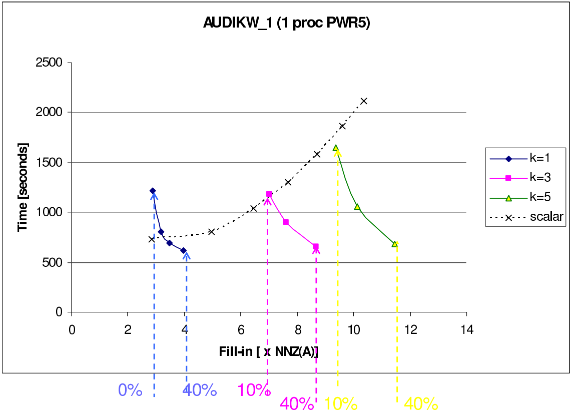


On itère tant que < tol

Conditions d'expérience

- Résultats obtenus sur IBM power5 + switch "Federation" ;
- Résultats en double précision ;
- Méthode de Krylov GMRES ;
- Précision relative 10^{-7}
- AUDI_KW1 (collection PARASOL) : matrice symétrique $n = 943,695$, $nnz(A) = 39,297,771$

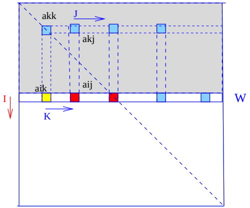
AUDI_KW1 : PaStiX ILU(k)



Factorisation ILUT(τ, p)

pour $i = 1$ *à* n **faire**
- $w := a_{i*}$
- **pour** $k = 1$ *à* $i - 1$ *et si* $w_k \neq 0$ **faire**
  - $w_k = w_k / a_{kk}$
  - Appliquer seuillage sur w_k (R_1)
  - **si** $w_k \neq 0$ **alors**
    - $w := w - w_k * u_{k*}$
  - **fin**
- **fin**
- Appliquer seuillage sur w (R_2)
- $l_{ij} := w_j$ pour $j = 1, ..., i - 1$
- $u_{ij} := w_j$ pour $j = i, ..., n$

fin



- R_1 : w_k est remplacé par 0 si $w_k < \tau.||a_{i*}||_2$.
- R_2 : on applique R_1 à tous les éléments de w. On ne garde que les p plus grands.

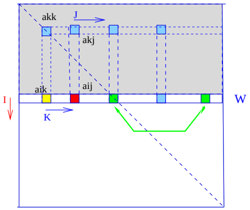
Factorisation ILUT(τ, p)

Quelques remarques :

- structure dynamique (allocation par ligne).
- autre algo : version colonne ILUC(τ) (utile en symétrique).
- utiliser un *scaling* $D_r.A.D_c$ est souvent utile : par exemple D_r est une matrice diagonale telle que $D_r(i,i)$ est $||A_{i*}||^{-1}$ (idem par colonne pour D_c) ;
- utilisation du paramètre p "dangeureuse" ;
- possibilité de pivotage à moindre coût.

Factorisation ILUTP(τ, p)

pour $i = 1$ à n **faire**
  $w := a_{i*}$
  pour $k = 1$ à $i-1$ *et si* $w_k \neq 0$ **faire**
    $w_k = w_k / a_{kk}$
    Appliquer seuillage sur w_k (R_1)
    si $w_k \neq 0$ **alors**
      $w := w - w_k * u_{k*}$
    fin
  fin
  Appliquer seuillage sur w (R_2)
  Echanger w_i avec $w_{max} = max_{k>i}(w_k)$
  $l_{ij} := w_j$ pour $j = 1, ..., i-1$
  $u_{ij} := w_j$ pour $j = i, ..., n$
fin



- R_1 : w_k est remplacé par 0 si $w_k < \tau.||a_{i*}||_2$.
- R_2 : on applique R_1 à tous les éléments de w. On ne garde

Renumérotation des inconnues

La renumérotation des inconnues influence grandement les résultats de convergence :

- Pour ILU(0), parfois mieux de rien faire !
- Reverse Cuthill-McKee (RCM) pour remplissage "moyen" (petit k dans ILU(k) ou grand τ petit p dans ILU(τ,p))
- Dissection emboîtée (+ Minimum degré) pour remplissage plus important (comme en direct).

voir Benzi, Szyld, Duin, "Orderings for incomplete factorization preconditioning of nonsymmetric problems", SIAM SISC, 17(1996), pp1135-1149.

Remarques sur les factorisations ILU

- Factorisation symbolique réutilisable en ILU(k).
- Robustesse depend du remplissage admis.
- Parallélisation "brute" difficile (sauf ILU(0)) :
  ▶ on les utilise souvent en tant que préconditionneurs locaux (i.e. séquentiel) par exemple en décomposition de domaine.
  ▶ on ajoute des contraintes de remplissages supplémentaires pour faciliter le parallélisme.

Plan

Plan

Le complément de Schur

En distinguant les inconnues du système en deux groupes d'inconnues B et C. Le système linéaire $A.x = y$ peut s'écrire :

$$\begin{pmatrix} A_B & A_{BC} \\ A_{CB} & A_C \end{pmatrix} . \begin{pmatrix} x_B \\ x_C \end{pmatrix} = \begin{pmatrix} y_B \\ y_C \end{pmatrix} \quad (1)$$

En exprimant x_B en fonction de x_C dans la 1ere ligne du système on a :

$$x_B = A_B^{-1}.(y_B - A_{BC}.x_C)$$

en reportant x_B dans la 2eme ligne on obtient :

$$(A_C - A_{CB}.A_B^{-1}.A_{BC}).x_C = y_C - A_{CB}.A_B^{-1}.y_B$$

Le complément de Schur

En distinguant les inconnues du système en deux groupes d'inconnues B et C. Le système linéaire $A.x = y$ peut s'écrire :

$$\begin{pmatrix} A_B & A_{BC} \\ A_{CB} & A_C \end{pmatrix} . \begin{pmatrix} x_B \\ x_C \end{pmatrix} = \begin{pmatrix} y_B \\ y_C \end{pmatrix} \tag{1}$$

$A.x = y$ ainsi décomposé peut se résoudre en 3 étapes :

$$\begin{cases} A_B.z_B = y_B \\ S.x_C = y_C - A_{CB}.z_B \\ A_B.x_B = y_B - A_{BC}.x_C \end{cases} \tag{2}$$

avec $S = A_C - A_{CB}.\mathbf{A_B^{-1}}.A_{BC} = A_C - A_{CB}.\mathbf{U_B^{-1}}.\mathbf{L_B^{-1}}.A_{BC}$

Le complément de Schur

Remarque : La factorisation LU de A peut s'écrire sous la forme :

$$A \approx \begin{pmatrix} L_B & \\ A_{CB}U_B^{-1} & L_S \end{pmatrix} \times \begin{pmatrix} U_B & L_B^{-1}A_{BC} \\ & U_S \end{pmatrix} \qquad (2)$$

avec $S = A_C - A_{CB}.\mathbf{A_B^{-1}}.A_{BC} = A_C - A_{CB}.\mathbf{U_B^{-1}}.\mathbf{L_B^{-1}}.A_{BC}$

Exemple d'utilisation : ILU multi-niveaux

On peut construire récursivement une suite de systèmes à résoudre :

$$\begin{cases} A_B^{(k)}.z_B = y_B^{(k)} \\ S^{(k)}.x_C = y_C^{(k)} - A_{CB}^{(k)}.z_B \\ A_B^{(k)}.x_B = y_B^{(k)} - A_{BC}^{(k)}.x_C \end{cases}$$

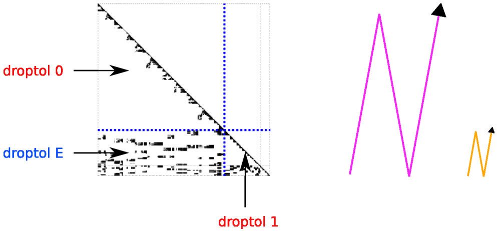
$$\begin{cases} A^{(k+1)} = S^{(k)} \\ y^{(k+1)} = y_C^{(k)} - A_{CB}^{(k)}.z_B \end{cases}$$

Les niveaux peuvent être construits avec différents critères :

- numériques (ARMS,...)
- décomposition du graphe de A (interface : HIPS).

Exemple d'utilisation : ILU multi-niveaux

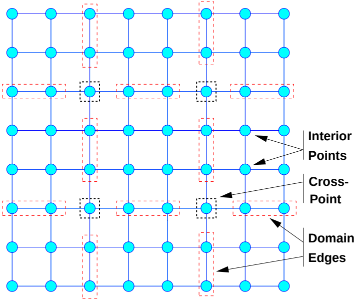
On peut construire récursivement une suite de systèmes à résoudre :



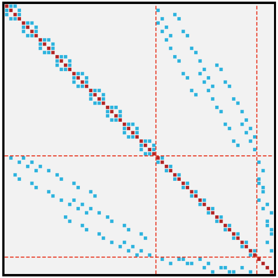
Les niveaux peuvent être construits avec différents critères :

- numériques (ARMS,...)
- décomposition du graphe de A (interface : HIPS).

Exemple d'utilisation : ILU multi-niveaux, HIPS



Grille 2D 8×8.

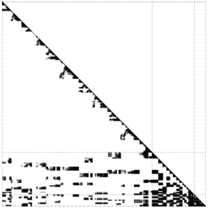


La matrice permutée.

Exemple d'utilisation : méthode hybride directe/itérative

- Exemple de remplissage (bcsstk14)

$$\begin{pmatrix} L_B & \\ A_{CB}{U_B}^{-1} & S \end{pmatrix}$$



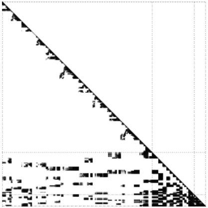
$$\begin{cases} A_B.z_B = y_B \\ S.x_C = y_C - A_{CB}.z_B \\ A_B.x_B = y_B - A_{BC}.x_C \end{cases} \tag{2}$$

avec $S = A_C - A_{CB}.\mathbf{A_B^{-1}}.A_{BC} = A_C - A_{CB}.\mathbf{U_B^{-1}}.\mathbf{L_B^{-1}}.A_{BC}$

Exemple d'utilisation : HIPS

- Exemple de remplissage (bcsstk14)

$$\begin{pmatrix} L_B & \\ A_{CB}{U_B}^{-1} & S \end{pmatrix}$$



- Eviter de stocker $A_{CB}{U_B}^{-1}$ et S en mémoire (surtout en 3D !) :
  - On n'a pas besoin de la matrice S, pour calculer $S.x$ on utilise : $(A_C - A_{CB}.U_B^{-1}.L_B^{-1}.A_{BC}).x$
  - ILUT : $A_{CB}{U_B}^{-1}$ (resp. ${L_B}^{-1}A_{BC}$) est "creusé" par un critère de seuillage au cours de son calcul (algorithme ILUC(t)= "regarde à gauche") : $\tilde{L_S}.\tilde{U_S} \approx \tilde{S} = (A_{CB}.\widetilde{U_B^{-1}}).(\widetilde{L_B^{-1}}.A_{BC}) \approx S$

Test cases

Le remplissage et le nombre d'opérations sont donnés pour une méthode directe.

Matrix	unknowns	non-zeros	fill-in	OPC
MHD1	$485,597$	$24,233,141$	52.4	$9.0 \,.\, 10^{12}$
AUDI	$943,695$	$39,297,771$	41.2	$5.4 \,.\, 10^{12}$
Haltere	$1,288,825$	$10,476,775$	38.7	$7.5 \,.\, 10^{12}$
Amande	$6,994,683$	$58,477,383$	53.9	$1.5 \,.\, 10^{13}$

Expériences avec HIPS

Conditions expérimentales :

10 nodes of 2.6 Ghz quadri dual-core Opteron (Myrinet)

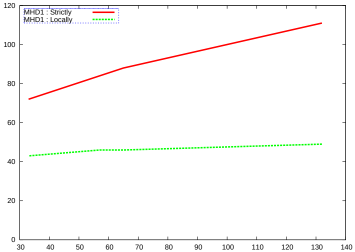
Partitionner : Scotch

$||b - A.x||/||b|| < 10^{-7}$, no restart in GMRES

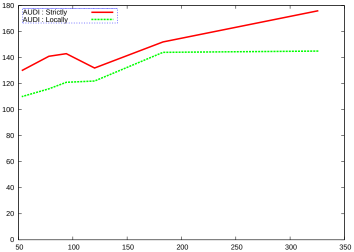
Thresholds : MHD1, Audi, Amande $= 0.001$, Haltere $= 0.01$

HIPS : Itérations/nombre de domaines

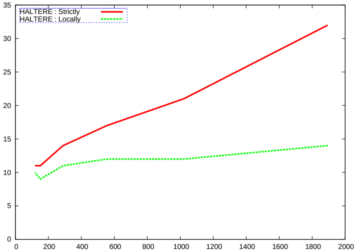
MHD1 (485, 597) : nb domains from 33 to 132



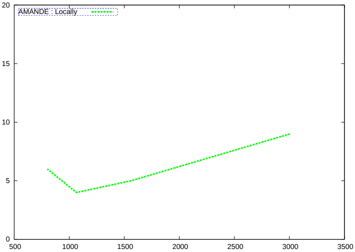
AUDI (943, 695) : nb domains from 53 to 326



HALTERE (1, 288, 825) : nb domains from 119 to 1894

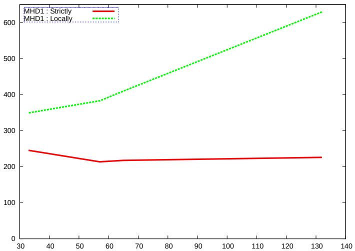


AMANDE (6, 994, 683) : nb domains from 801 to 3004

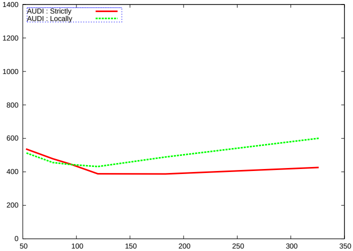


HIPS : Temps (prec+solve seq.)/nombre de domaines

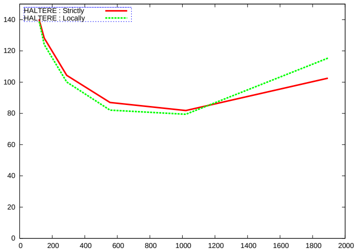
MHD1 (485, 597) : nb domains from 33 to 132



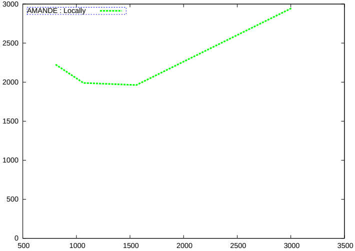
AUDI (943, 695) : nb domains from 53 to 326



HALTERE (1, 288, 825) : nb domains from 119 to 1894

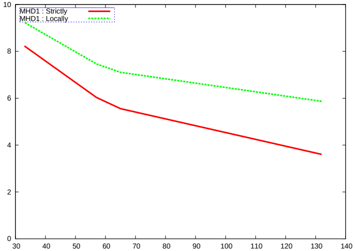


AMANDE (6, 994, 683) : nb domains from 801 to 3004

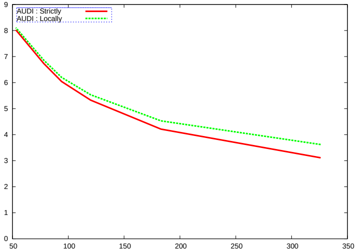


Mémoire / nombre de domaines

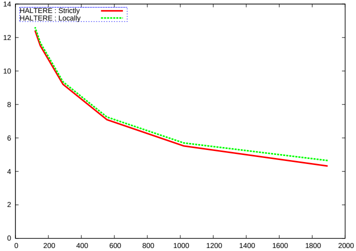
MHD1 (485, 597) : nb domains from 33 to 132



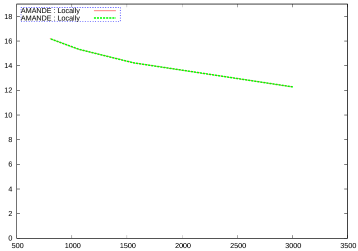
AUDI (943, 695) : nb domains from 53 to 326



HALTERE (1, 288, 825) : nb domains from 119 to 1894



AMANDE (6, 994, 683) : nb domains from 801 to 3004



Logiciels en rapport avec le cours

- ILUPACK : ILU multi-niveaux
  http://www.math.tu-berlin.de/ilupack/
- HIPS : ILU multi-niveaux, hybride directe/itératif
  http://hips.gforge.inria.fr
- PARMS : ILU multi-niveaux
  http://www-users.cs.umn.edu/~saad/
- PaStiX : Direct (MPI/Thread), ILU(k) par blocs
  http://pastix.gforge.inria.fr
- PETSc : meta-package
  http://www-unix.mcs.anl.gov/petsc/petsc-as/
- Trilinos : meta-package
  http://trilinos.sandia.gov/