Introduction aux méthodes de décomposition de domaine

Jacques Laminie

Groupe de Recherche en Informatique et en Mathématiques Appliquées des Antilles et de la Guyane, Campus de Fouillole, Pointe-à-Pitre 97157 cédex, Guadeloupe

du 1er au 5 décembre 2008
ANGD 2008

Introduction

Quel type de parallélisme ? ou de quelle machine dispose-t-on ?
Quelle granularité ?
Comment exprimer celui–ci ?
Pourquoi ?

- boucles
- sous–programmes

- Problème de temps de restitution du résultat
- Problème de capacité mémoire

Pourquoi ajouter quelque chose ?

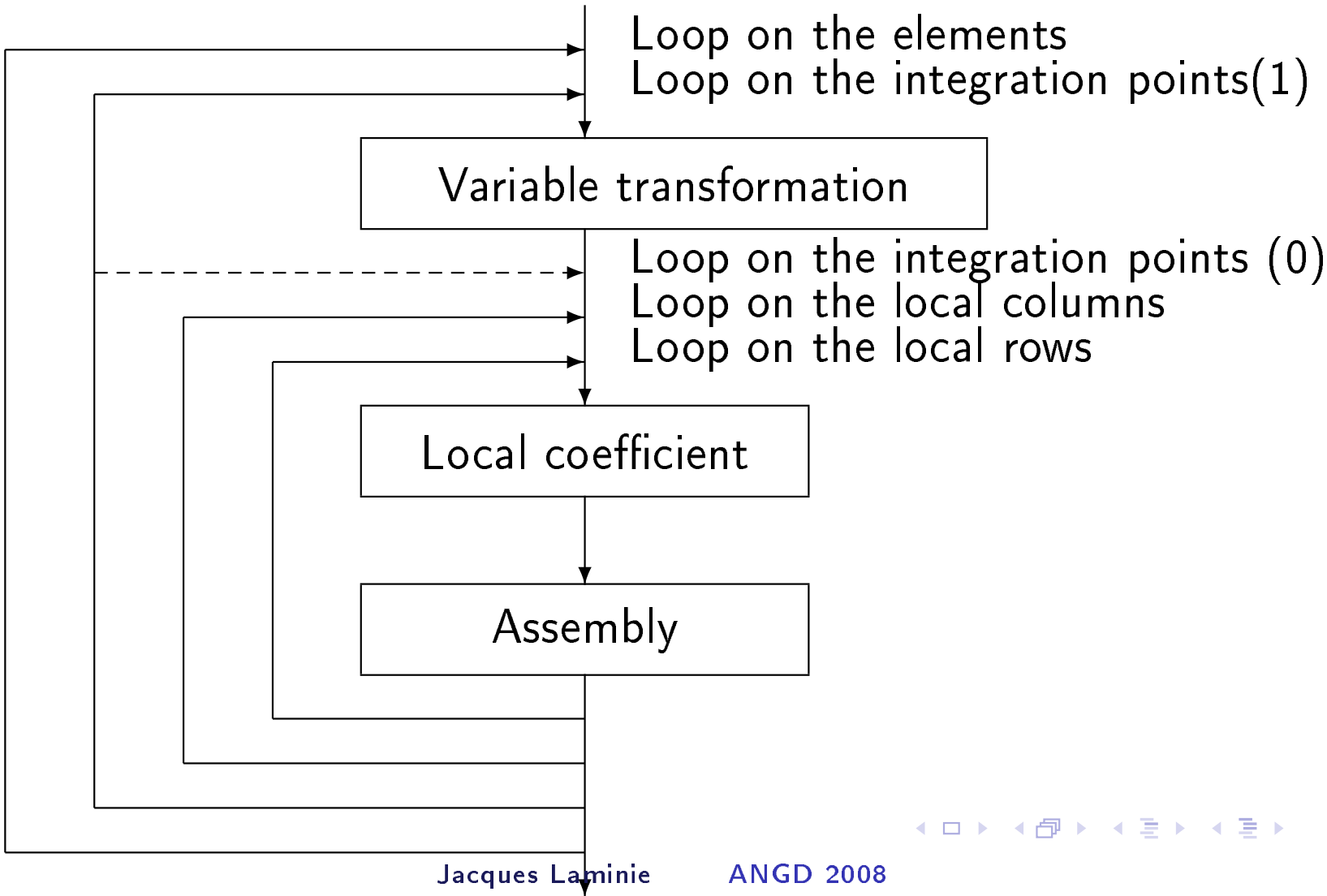
- Analyse du potentiel “naturel” d'un code
- Que faire pour améliorer cet état de fait ?
- Réécriture du code : que peut-on conserver du programme séquenciel ?

Analyse du potentiel "naturel" d'un code

Pour fixer un cadre de travail, parlons d'éléments finis.

- Calcul des matrices
- Factorisation plus ou moins complètes
- Méthodes de résolution des systèmes linéaires itératives

Calcul des matrices



Factorisations plus ou moins complètes

Version IKJ de LU

```
for i = 2 to n do
    for k = 1 to i-1 do
        A(i,k) = A(i,k) / A(k,k)
        for j = k+1 to n do
            A(i,j) = A(i,j) - A(i,k)*A(k,j)
        end do
    end do
end do
```

Méthodes de résolution des systèmes linéaires itératives

Produit Matrice–Vecteur

Matrices creuses : CSR

```
for i = 1 to n do
    k1 = IA(i)
    k2 = IA(i+1)-1
    for k = k1 to k2 do
        j = JA(k)
        y(i) = y(i) + CA(k) * x(j)
        y(j) = y(j) + CA(k) * x(i)
    end do
end do
```

Matrices creuses : ELL

```
for i = 1 to n do
    for k = 1 to IA(i,0) do
        j = IA(i,k)
        y(i) = y(i) + CA(i,k) * x(j)
        y(j) = y(j) + CA(i,k) * x(i)
    end do
end do
```

Mieux encore si `nbc` est le nombre maximal de coefficients non nuls par ligne. On peut intervertir les deux boucles.

Matrices creuses : ELL

```
for k = 1 to nbc do
    for i = 1 to n do
        j = IA(i,k)
        y(i) = y(i) + CA(i,k) * x(j)
        y(j) = y(j) + CA(i,k) * x(i)
    end do
end do
```

ATTENTION : `IA(i,*) = 0` pour les coefficients ajoutés et `CA(i,*) = 0.0`

Conclusions

On se rappelle que, pour que le calcul parallèle soit “rentable” il faut environ que 90% du code s'exécute de façon concurrente. Ou bien on a un problème de mémoire (c'est souvent le cas).

Il faut trouver / créer plus de potentiel ...

L'une des réponses : les méthodes de décompositions de domaine

Contexte :

Maillage (régulier ou non)

Différences finies ; Eléments finis ; Volumes finis
Equations aux dérivées partielles

L'IDÉE : Couper le domaine de calcul en plusieurs sous–domaines

Peut s'étendre à d'autres applications.

Les principales méthodes

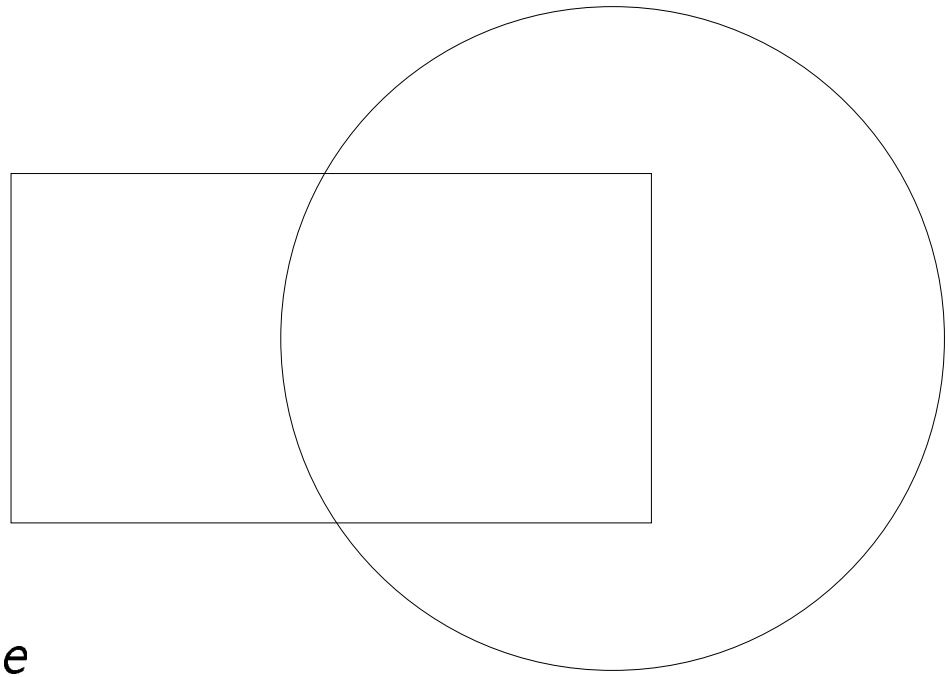
On agit sur le domaine de calcul.

Deux grandes familles de méthodes

- les méthodes avec recouvrement
- les méthodes sans recouvrement

La méthode de Schwartz (1869)

Solution analytique du problème de Poisson dans un domaine complexe.



Ω_1 : *Le disque*

Ω_2 : *Le rectangle*

Comment ça marche ?

- C'est une méthode itérative.
- On se donne une solution intiale sur tout le domaine, en particulier sur le bord de tous les sous–domaines.
- Sur chaque sous–domaine, on a un problème de Poisson bien défini (Condition de Dirichlet sur chaque sous–domaine).
- On résout sur chaque sous–domaine.
- On a de nouvelles valeurs sur les bords de chaque sous–domaine.

Deux techniques, qui dépendent de l'ordre dans lequel les étapes précedentes sont faites.

La méthode alternée de Schwarz

u_2^0 donné dans Ω_2 et on pose $n = 0$

Etape 1 : On résout

$$\begin{aligned} -\Delta u_1^{n+1} &= f && \text{dans } \Omega_1 \\ u_1^{n+1} &= 0 && \text{sur } \partial\Omega \cap \partial\Omega_1 \\ u_1^{n+1} &= u_2^n && \text{sur } \partial\Omega_2 \cap \partial\Omega_1 \end{aligned}$$

Etape 2 : On résout

$$\begin{aligned} -\Delta u_2^{n+1} &= f && \text{dans } \Omega_1 \\ u_2^{n+1} &= 0 && \text{sur } \partial\Omega \cap \partial\Omega_2 \\ u_2^{n+1} &= u_1^n && \text{sur } \partial\Omega_1 \cap \partial\Omega_2 \end{aligned}$$

Etape 3 : Convergence ou
$n = n + 1$ et goto Etape 1

C'est la méthode multiplicative et c'est toujours séquentiel ...

Du point de vue matriciel on peut réécrire comme suit :

$$\begin{aligned} u_1^{n+1} - u^n &= -A_1^{-1} R_1 (A u^n - f) \\ u^{n+1/2} &= u^n + R_1^t (u_1^{n+1} - u^n) \end{aligned}$$

$$\begin{aligned} u_2^{n+1} - u^{n+1/2} &= -A_2^{-1} R_2 (A u^{n+1/2} - f) \\ u^{n+1} &= u^{n+1/2} + R_2^t (u_1^{n+1/2} - u^n) \end{aligned}$$

C'est un problème de point fixe (avec $Au = f$)

$$u^{n+1} - u = (I - R_2^t A_2^{-1} R_2 A)(I - R_1^t A_1^{-1} R_1 A)(u^n - u)$$

La méthode additive :

$$u^{n+1} - u = (I - R_2^t A_2^{-1} R_2 A - R_1^t A_1^{-1} R_1 A)(u^n - u)$$

Généralisation à K sous–domaines

La méthode multiplicative ;

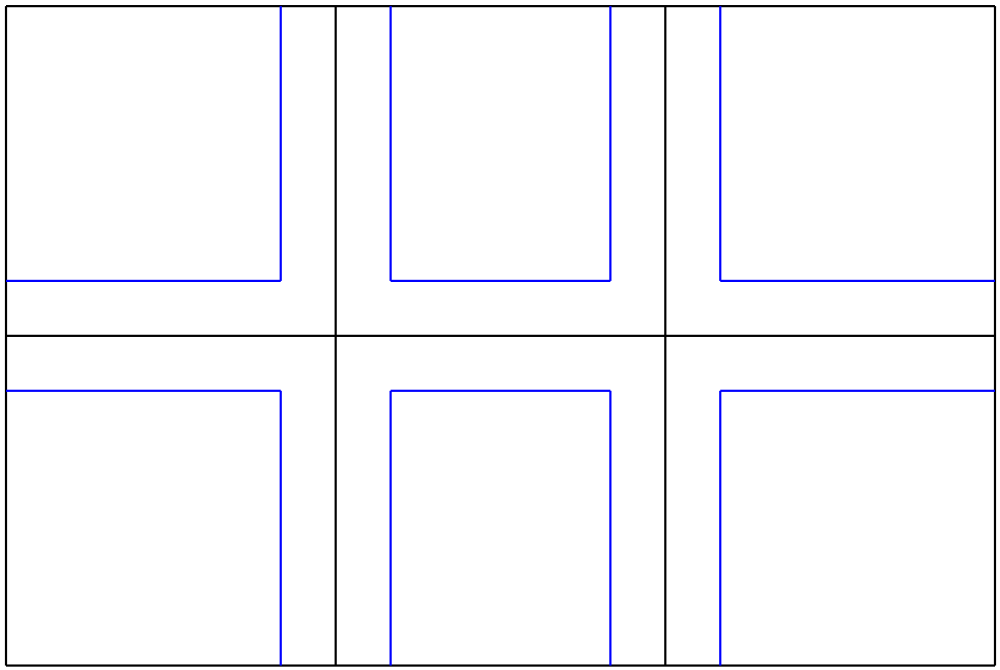
$$u^{n+1} - u = \left(\prod_{k=1}^{K} (I - R_k^t A_k^{-1} R_k A) \right) (u^n - u)$$

La méthode additive :

$$u^{n+1} - u = (I - (\sum_{k=1}^{K} R_k^t A_k^{-1} R_k A)(u^n - u)$$

Comment faire ?

Décomposition du domaine de calcul sans recouvrement. Création d'un sous–domaine autour des interfaces.



La vitesse de convergence dépend de la taille du recouvrement

Quelques remarques complémentaires

Technique d'accélération

On peut voir S.M. comme la méthode de Gauss–Seidel et S.A. comme celle de Jacobi.
S.A. peut être généralisé comme une méthode de Richardson préconditionnée, puis comme un GRADIENT CONJUGUÉ préconditionné.
Pour S.M. analyse analogue mais vers GMRES (S.M. est non symétrique)

Ajout d'un problème global (grille grossière)

Résultat de convergence dépendant du diamètre des sous–domaines

Schwarz additif

$$
\begin{aligned}
u^{n+1/2} &= u^n &&+R_1^t A_1^{-1} R_1 (f - A u^n) \\
u^{n+1} &= u^{n+1/2} &&+R_2^t A_2^{-1} R_2 (f - A u^n)
\end{aligned}
$$

Richardson préconditionnée $\rho_n = 1$

$$
\begin{aligned}
u^{n+1} &= u^n &&+\rho_n R_1^t A_1^{-1} R_1 (f - A u^n) \\
& &&+\rho_n R_2^t A_2^{-1} R_2 (f - A u^n)
\end{aligned}
$$

Gradient conjugué préconditionné

puis on remplace A_i^{-1} par une approximation B_i

Pratiquement les performances se dégradent quand le nombre de sous–domaines augmente.

LE REMÈDE : ajouter un maillage global grossier.

Il permet un échange d'information en une seule étape.

Pour la méthode de Schwarz multiplicatif : une démarche analogue MAIS matrice non symétrique => GMRES

Les méthodes sans recouvrement

Une numérotation particulière des nœuds

Ω_4	Ω_5	Ω_6
Ω_1	Ω_2	Ω_3

On numérote les nœuds de chaque sous–domaine puis ceux des interfaces (Σ) entre les sous–domaines.

Action de cette numérotation sur le graphe de la matrice

$$\begin{pmatrix} A_1 & & & & & & C_1 \\ & A_2 & & & & & C_2 \\ & & A_3 & & & & C_3 \\ & & & A_4 & & & C_4 \\ & & & & A_5 & & C_5 \\ & & & & & A_6 & C_6 \\ B_1 & B_2 & B_3 & B_4 & B_5 & B_6 & A_\Sigma \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_\Sigma \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_\Sigma \end{pmatrix}$$

Jolie structure n'est–ce pas ?

Produit matrice–vecteur, Factorisation, . . . ;
il y a beaucoup de possibilités de parallélisme.

Les méthodes primales

On a les équations suivantes :

$$A_k u_k + C_k u_\Sigma = f_k \text{ pour } k = 1, \ldots, 6$$

et

$$\sum_{k=1}^{6} B_k u_k + A_\Sigma u_\Sigma = f_\Sigma$$

Mais A_k est la matrice de notre problème (quelque soit celui–ci) sur le domaine Ω_k (Dirichlet sur le bord). Donc si on connait u_Σ, on sait résoudre sur chaque sous–domaine.

$$u_k = A_k^{-1}(f_\Sigma - C_k u_\Sigma)$$

On reporte ces valeurs dans la dernière équation. On obtient un problème en u_Σ

$$\left(A_\Sigma - \sum_{k=1}^{6} B_k A_k^{-1} C_k \right) u_\Sigma = f_\Sigma - \sum_{k=1}^{6} B_k A_k^{-1} f_k$$

En général on peut écrire

$$A_\Sigma = \sum_{k=1}^{6} A_{\Sigma,k} \quad \text{et} \quad f_\Sigma = \sum_{k=1}^{6} f_{\Sigma,k}$$

On obtient encore en généralisant à K sous–domaines

$$\left(\sum_{k=1}^{K} A_{\Sigma,k} - B_k A_k^{-1} C_k \right) u_\Sigma = \sum_{k=1}^{K} f_{\Sigma,k} - B_k A_k^{-1} f_k$$

On résout ce système par une méthode de descente (Gradient Conjugé, GMRES, ...). Le produit matrice–vecteur peut se faire en parallèle, le calcul du second membre également.

Pour chaque produit il y a un système linéaire **local** à résoudre. On résout localement

$$(A_{\Sigma,k} - B_k A_k^{-1} C_k) u_{\Sigma,k} = f_{\Sigma,k} - B_k A_k^{-1} f_k \quad \text{et} \quad u_\Sigma = \sum_{k=1}^{K} u_{\Sigma,k}$$

Ce système est petit (par comparaison avec le problème initial). On peut le faire de façon itérative MAIS AUSSI DE FAÇON DIRECTE.

Sur chaque sous–domaine ON GARDE TOUT SON SAVOIR–FAIRE

Préconditionnement

De façon heuristique, sur chaque sous–domaine, on donne une valeur de u sur tous les bords en particulier sur l'interface. En quelque sorte, l'opérateur local donne en réponse une dérivée normale sur l'interface. Globalement on obtiendrait ainsi un saut des dérivées normales de part et d'autre de l'interface. Construire un bon préconditionnement consisterait à définir le problème dual : donner une dérivée normale (ou plutot un saut de dérivées normales) pour obtenir une valeur sur le bord.

Comme pour les méthodes de Schwarz il faut aussi ajouter un problème grossier.

Les méthodes duales

- Maillage sans recouvrement
- Mais : numérotations multiples des nœuds de l'interface. Plus précisément, on a une numérotation différente, propre à chaque sous–domaine, des nœuds.
- On n'a plus la continuité de la solution u sur l'interface.
- Par contre on a le raccord des dérivées normales à l'interface.

La continuité de la solution est considérée comme contrainte.

On reprend le problème de Poisson sur deux sous–domaines.

$$\begin{aligned} -\Delta u_1 &= f + P_1\lambda && \text{dans } \Omega_1 \\ u_1 &= 0 && \text{sur } \partial\Omega \cap \partial\Omega_1 \\ \frac{\partial u_1}{\partial n} &= 0 && \text{sur } \partial\Omega_2 \cap \partial\Omega_1 \end{aligned}$$

$$\begin{aligned} -\Delta u_2 &= f + P_2\lambda && \text{dans } \Omega_2 \\ u_2 &= 0 && \text{sur } \partial\Omega \cap \partial\Omega_2 \\ \frac{\partial u_2}{\partial n} &= 0 && \text{sur } \partial\Omega_1 \cap \partial\Omega_2 \end{aligned}$$

Avec la contrainte

$$P_2^t u_2 = P_1^t u_1 \quad \text{sur } \partial\Omega_1 \cap \partial\Omega_2$$

Une fois discrétisé, en notation matricielle, on a

$$\begin{aligned} A_1 u_1 &= f_1 + P_1 \lambda \\ A_2 u_2 &= f_2 + P_2 \lambda \end{aligned}$$

$$P_1^t u_1 = P_2^t u_2$$

Comme précédemment, on utilise le complément de Schur pour réécrire ces équations en un problème en λ. On généralise à plus de deux sous–domaines.

$$\left(\sum_{k=1}^{K} P_k^t A_k^{-1} P_k \right) \lambda = \sum_{k=1}^{K} P_k^t A_k^{-1} f_k$$

Préconditionnement, Problème grossier, ...

- Comme pour la méthode primale, il est nécessaire de préconditionner le problème en λ.
- On peut définir un problème grossier.

Il y a quelques variantes de cette méthode duale.
En particulier **FETI, FETI-DP**.

Mise en œuvre

Quelle type de parallélisme ?

Programme complexe (Résolution sur un sous–domaine).

Quelles données sur quel processeur ?

Tous ce qui est indicé par k sur le même processeur.

en plus on y met une copie de u_Σ

Quelle communication ?

On a faire le cumul des $u_{\Sigma,k}$ sur l'interface Σ pour obtenir u_Σ

Quelle structure de programmation ?

SIMD MIMD SPMD

Open–MP, MPI, . . . ?

MAIS ...

- Synchronisation (communication) dans chaque produit matrice–vecteur.
- Méthode de descente sur u_Σ. Quelle préconditionnement ?
- Pour la méthode primale, sur Σ on doit avoir la continuité de u_Σ (ça, c'est vrai) mais aussi la continuité des dérivées normales. Ça on ne l'obtient qu'à la convergence.
- Pour la méthode duale, c'est le contraire.

Comment fait–on ?

- Maillage : décomposer le domaine de calcul en sous–domaines.
- Maillage : repèrer les interfaces
- Problème : discrétisation comme d'habitude
- Algèbre linéaire : Opérateurs locaux
- Algèbre linéaire : Opérateurs globaux ==> communications

Comment répartir les données sur l'ensemble des processeurs ?

Décomposer le domaine de calcul en sous–domaines.

- On utilise un maillage grossier
- On découpe en sous–domaines
  - par une méthode lexicographique
  - par voisinage

Repèrer les interfaces

A chaque produit matrice–vecteur, on a vu que l'on calcule sur chaque sous–domaine une contribution locale sur l'interface, dont il faut faire la somme des valeurs obtenues de part et d'autre de Σ.

Il est donc nessaire que chaque sous–domaine transmette au(x) voisin(s) ses données. Il va donc recevoir les données des voisins pour en faire la somme.

Deux types d'interfaces

- Un segment commun à deux sous–domaines
- Un point commun à plusieurs sous–domaines.

Le principe étant un sous-domaine par cœur.

Pour chaque interface, on a besoin de connaître la liste des nœuds de celle–ci et la liste des sous–domaines voisins.

Il faut construire ces listes à partir de la donnée du maillage.

Opérateurs globaux ==> communications

Nous avons besoin principalement de DEUX types de communcations :

- cumul aux interfaces ou transfert de données
- produits scalaires

Transfert de données

```
Send loop
for intf = 1 to nb_interfaces do
    nbv = get_nb_voisin(intf)
    for nv = 1 to nbv do
        uid = get_uid        (intf, nbv)
        nbp = get_nb_points (intf, nbv)
        deb = get_debut_list(intf, nbv)
        for np = 1 to nbp do
            list(np) = data(get_pos(deb+np-1))
        end
        send_list(np, list) to uid
    end
end
```

Transfert de données

```
Receive loop
for intf = 1 to nb_interfaces do
    nbv = get_nb_voisin(intf)
    for nv = 1 to nbv do
        uid = get_uid       (intf, nbv)
        nbp = get_nb_points (intf, nbv)
        deb = get_debut_list(intf, nbv)
        recv_list(np, list) from uid
        for np = 1 to nbp do
            data(get_pos(deb+np-1)) = list(deb+np-1)
        end
    end
end
```

Produits scalaires et plus

```
loc_val = dot_product(data, data)
glb_val = global_sum(loc_val)
```

Barrière ...

Besoin de synchroniser l'ensemble des processus

Besoin de terminer l'exécution
un peu plus compliqué, il faut prévenir tout le monde, éventuellement synchroniser. Puis finir correctement la session parallèle.
ENFIN STOP.

Décomposition de domaine et open–MP

On fait une grosse boucle sur les sous-domaines dès le début du programme et jusqu'à la fin.

Dès qu'il y a besoin de "communiquer", on résout le problème dans une section critique.

Conclusions et commentaires

Les méthodes de décomposition en sous–domaines permettent :

- d'augmenter considérablement le potentiel de mise en concurrence d'un code de calcul.
- de réutiliser l'essentiel du savoir–faire du programme séquentiel

Cette technique s'adapte

- à presque toutes les architectures de machine
- à toutes les méthodes de programmations

Bibliographie

LT94 : P. Le Tallec, Domain decomposition methods in computational mechanics, Comput. Mech. Advances ; North-Holland, *1, 121-220, 1994.*

FR94 : C. Farhat and F.X. Roux, Implicit parallel processing in structural mechanics. Comput. Mech. Advances ; North-Holland, *2, 1-124, 1994.*

Il y a également de nombreux “proceeding” de conférences sur les méthodes de décompositions de domaine.