Documentation des structures de données parallèles de Trio_U

Benoît Mathieu

Mise à jour : 4/12/2006, Trio_U version 1.5.1 beta

Ce document présente les concepts et structures de données sur lesquelles repose le parallélisme du code Trio_U.

Les identifiants (noms de classes, méthodes et variables) qui figurent dans ce document peuvent être un peu différents de ceux présents dans Trio_U mais les concepts sont normalement à jour avec le code…

Modèle de programmation : SPMD (Single Program, Multiple Data)

Définition du SPMD / MPMD

Le code Trio_U fonctionne la plupart du temps selon un modèle SPMD : tous les processeurs exécutent le même code et se trouvent à un instant donné au même endroit dans le code. Ainsi, tous les processeurs résolvent le même système d'équations simultanément (mêmes équations, même discrétisation, même schéma en temps, etc...). Dans ce modèle, les communications font l'objet de *points de synchronisation* chaque fois que les processeurs doivent échanger des données.

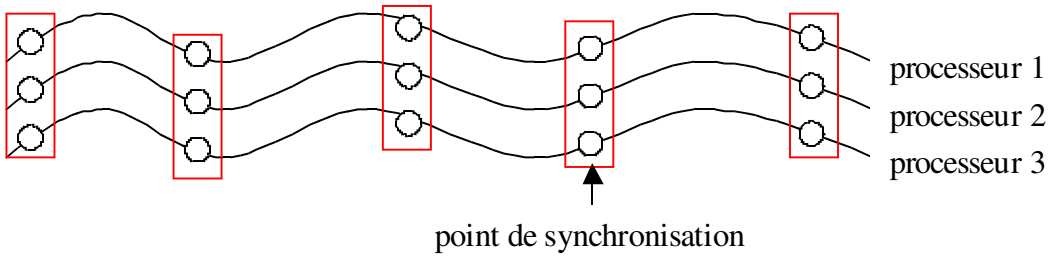


Illustration du modèle SPMD: chaque processeur suit exactement le même chemin dans le code

Le modèle SPMD s'oppose au modèle MPMD dans lequel chaque processeur exécute un code différent. Les communications entre processeurs font alors l'objet de *points de rendez-vous* moins fréquents et plus difficiles à organiser. Par exemple, si Trio_U utilisait un mode MPMD, on pourrait imaginer qu'un processeur résolve une équation de Navier-Stokes pendant qu'un autre résout un problème de thermique.

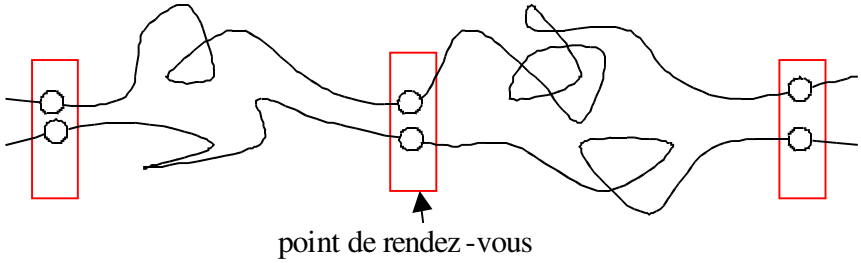


Illustration du modèle MPMD: chaque processeur exécute un code différent et organise des points de rendez-vous avec les autres processeurs

Modèle SPMD de Trio_U

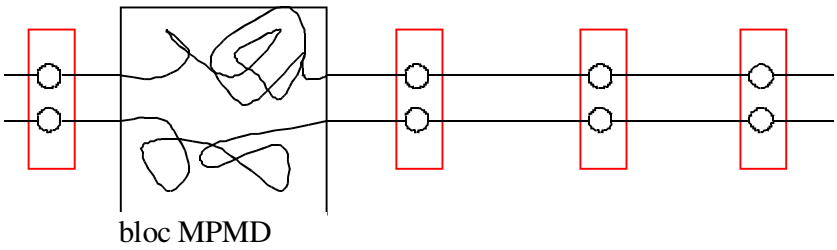
Voici un exemple de code SPMD au sens strict :

```
for(i=0; i<n; i++)
{
   valeur[i] = a[i] * b[i];
   somme[i] = mpsum(valeur[i]);
}
```

Cette portion de code est exécutée simultanément par tous les processeurs car à chaque itération de la boucle il existe un *point de synchronisation* (les processeurs échangent des données). Par conséquent, la boucle doit être exécutée le même nombre de fois sur tous les processeurs, et la valeur de n doit donc être identique sur tous les processeurs.

Dans Trio_U, le code n'est pas SPMD au sens strict car les données à traiter par chaque processeur n'ont pas toujours exactement la même taille. De plus, il existe dans certains modèles non linaires (turbulence, opérateurs de convections, etc...) des traitements conditionnels ("if") dont le résultat dépend de valeurs locales et peut donc être différent d'une processeur à l'autre.

Trio_U est donc constitué de blocs de code MPMD au niveau le plus fin des algorithmes (boucles sur les items géométriques dont le nombre varie d'un processeur à l'autre, boucles sur les éléments d'une matrice, etc). Aucune communication entre processeurs ne doit prendre place dans ces blocs. Ces blocs sont eux même organisés selon un modèle SPMD : ils sont exécutés simultanément sur tous les processeurs.



Modèle parallèle de Trio_U: code SPMD contenant des blocs MPMD exempts de points de synchronisation.

Convention développeur :

En principe, les algorithmes de haut niveau de Trio_U sont codés selon le modèle SPMD (opérateurs, solveurs matriciels, méthodes du framework, etc). Par conséquent, tous ces services doivent être appelés simultanément sur tous les processeurs.

En particulier:

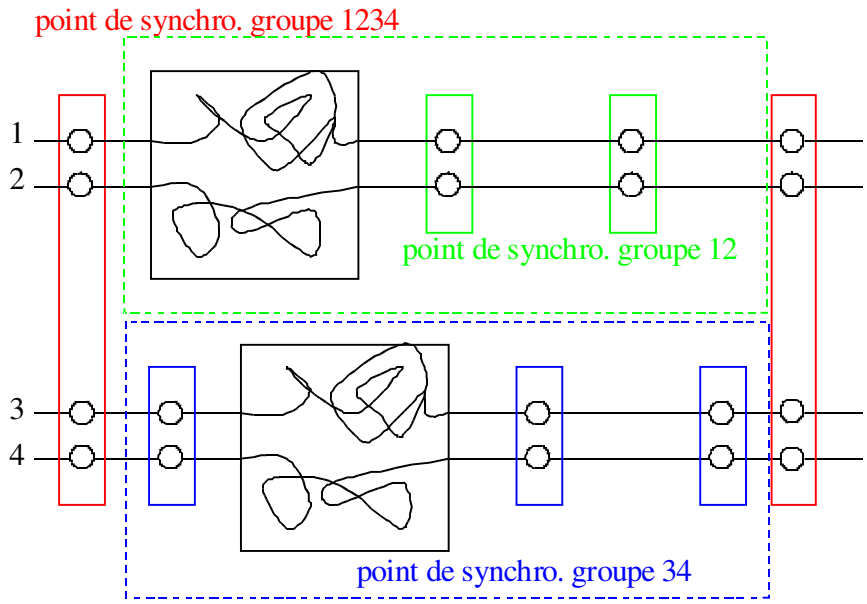
- si un service est appelé dans un test "if", ce test doit donner le même résultat sur tous les processeurs
- si un service est appelé dans une boucle "while" ou "for", cette boucle doit être exécutée le même nombre de fois sur tous les processeurs

Un service est SPMD dès lors qu'il contient des points de synchronisation entre processeurs mais cette propriété n'est malheureusement pas documentée dans le code pour l'instant (au développeur de vérifier).

Si un branchement conditionnel (if, for, etc), peut donner un résultat différent sur chaque processeur, le bloc doit être identifié comme bloc MPMD et ne pas contenir de point de synchronisation

Trio_U supporte théoriquement un fonctionnement MPMD dans les algorithmes de haut niveau. Ce support passe par l'utilisation de la notion de *groupe* (classe *Process*). Un groupe est un ensemble de processeurs qui exécutent simultanément une portion de code SPMD. Ainsi, si on utilise l'instruction *mpsum* , la somme porte sur l'ensemble des processeurs du groupe auquel appartient le processeur courant.

Cette notion n'est cependant pas utilisée pour l'instant (à part dans la maquette mise en œuvre par Eli Laucoin pour l'AMR et la répartition de charge dynamique). Une application possible des groupes pourrait être de faire calculer par autant de groupes différents les sous-problèmes d'un problème couplé. Afin d'utiliser proprement cette notion, il faudrait auparavant redéfinir la notion d'*objet distribué* pour ajouter à chaque objet une référence au *groupe* sur lequel cet objet est distribué.

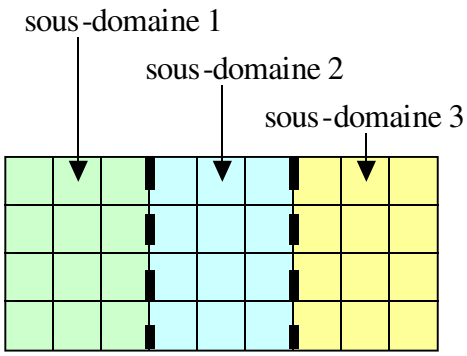


Modèle Trio_U complet à trois niveaux: MPMD au niveau des groupes, SPMD au niveau intermédiaire, MPMD au niveau le plus fin (algorithmes). Le niveau MPMD des groupes n'est pas utilisé pour l'instant...

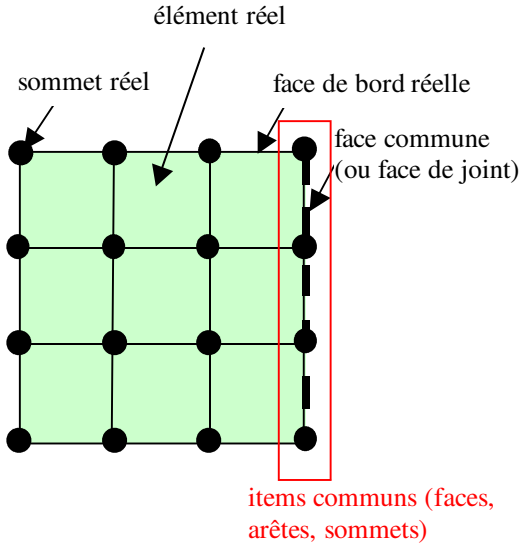
Maillage distribué et structures de données

Définitions des entités géométriques

Le domaine de calcul est découpé en N parties qui forment une *partition* de l'ensemble des éléments (ou mailles) du maillage (un élément appartient à un et un seul sous-domaine du découpage).



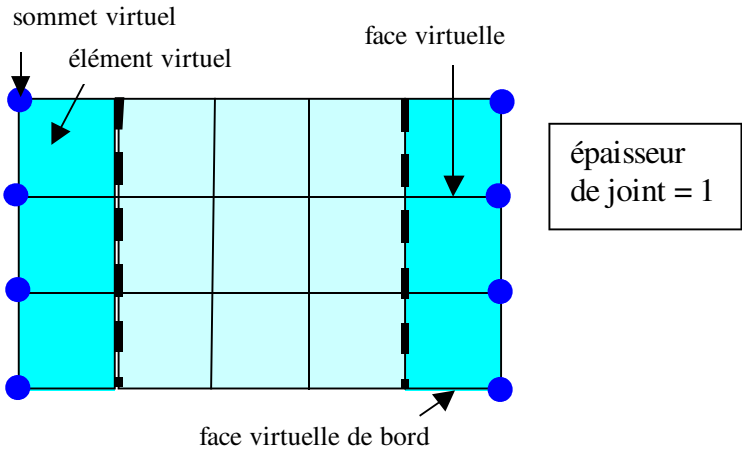
Un processeur traite un sous-domaine et connaît les éléments du sous-domaine ainsi que tous les items géométriques attachés à ce sous-domaine (sommets de l'élément, faces, arêtes, faces de bord). Les items géométriques attachés aux éléments du sous-domaine sont appelés *items réels* (on a donc des éléments réels, sommets réels, faces réelles, arêtes réelles). Certains items réels (sommets, faces et arêtes) sont partagés par plusieurs processeurs. Ils sont appelés *items communs*.



On appelle joint la frontière entre deux sous-domaines (ensemble des items communs et informations relatives au raccord avec les sous-domaines voisins).

Les schémas numériques aux différences finies calculent des valeurs sur les items (sommets, faces, éléments, etc) en fonction des valeurs sur les items voisins. Pour que les valeurs puissent être calculées sur les items proches des joints, un processeur a besoin de connaître les items voisins des items réels. Les items supplémentaires connus par un processeur sont appelés items (élément/face/sommet/arête) virtuels. Le voisinage connu est déterminé par un paramètre appelé « épaisseur de joint ».

Si l'épaisseur de joint vaut N, un sous-domaine connaît l'ensemble des éléments voisins des éléments d'épaisseur N-1, ainsi que tous les sommets, faces et arêtes attachés aux éléments connus. Le voisinage est pris au sens suivant : deux éléments sont voisins s'ils partagent un sommet commun.

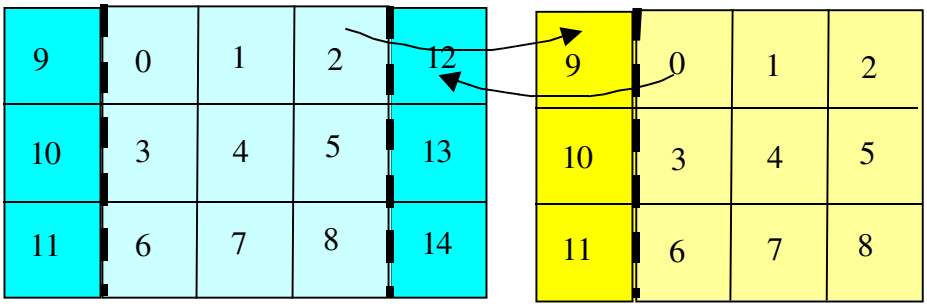


Par convention :

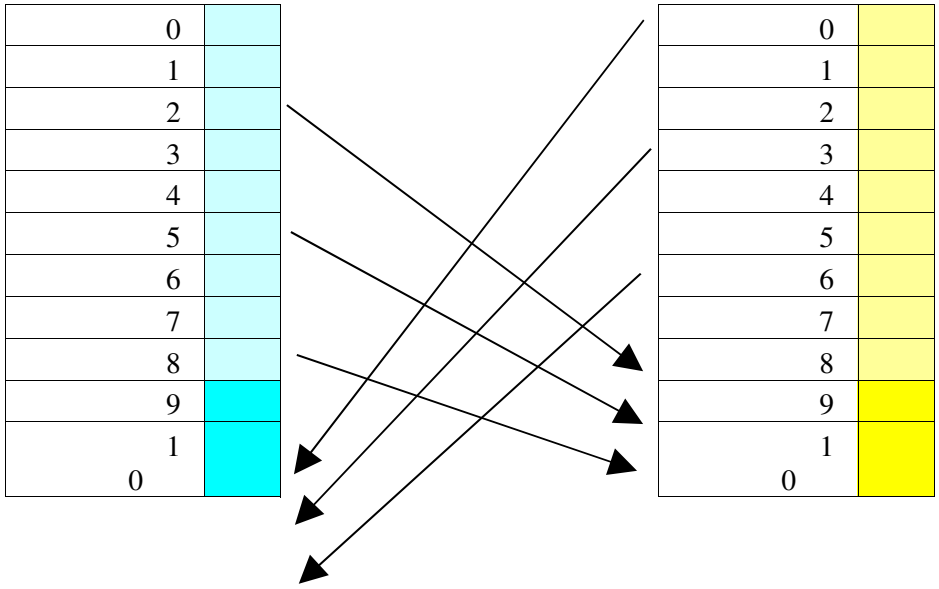
- les « items communs » sont aussi des « items réels »
- les « faces de bord » sont aussi des « faces »

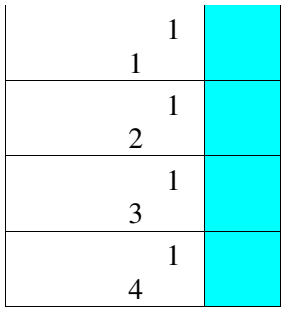
Tableaux distribués

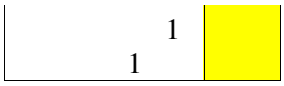
Un tableau distribué est un tableau de valeurs dont certaines valeurs sont partagées par plusieurs processeurs. Par exemple, pour un champ de température défini au centre des éléments du maillage, les valeurs sur les éléments virtuels sont égales aux valeurs de l'élément réel correspondant sur le sous-domaine voisin :



L'échange de l'espace virtuel consiste pour chaque sous-domaine à réaliser les affectations du tableau ci-dessous par échange de messages entre les processeurs :







En principe, les algorithmes calculent les valeurs associées aux items réels, puis on réalise un « échange espace virtuel » lors duquel les valeurs réelles sont envoyées aux processeurs voisins pour mettre à jour leurs items virtuels. A partir de ce moment, on dit que l'espace virtuel du tableau est à jour.

Les items réels d'un sous-domaine qui doivent être envoyés à un processeur voisin sont appelés *items distants* du processeur voisin.

Dans un tableau distribué, les valeurs associées aux items réels apparaissent en premier, dans le même ordre que celui où sont définis les items géométrique (case n du tableau = température du n-ième élément dans notre exemple). Les valeurs associées aux items virtuels apparaissent à la fin. Les items virtuels sont placés à la fin du tableau, dans l'ordre croissant de l'indice du sous-domaine qui possède l'item, et pour chaque sous-domaine, dans l'ordre défini par le tableau des *items distants* sur le sous-domaine.

Le cas des items communs est particulier : en théorie, tous les sous-domaines pourraient calculer les valeurs sur les items communs. S'ils utilisent les mêmes valeurs en entrée (cohérence des tableaux distribués en entrée), ils devraient aboutir au même résultat et il ne devrait pas être nécessaire de synchroniser les valeurs des items communs. En raison des erreurs d'arrondi, on réalise un « échange des items communs » selon le principe suivant : le sous-domaine de rang le plus bas qui possède un item commun impose sa valeur à l'item sur tous les autres processeurs.

Quelques éléments importants :

- Un item peut être envoyé à plusieurs processeurs mais est toujours reçu d'un seul processeur.
- Un item peut être à la fois distant et commun.

Construction des tableaux distribués d'items

On préconise d'utiliser la méthode générique Scatter ::creer_tableau_distribue(...)

La construction d'un tableau distribué est faite à partir de la liste d'items communs et de la liste d'items distants pour chaque sous-domaine voisin :

On augmente la taille du tableau du nombre d'items virtuels (ce nombre est la somme du nombre d'items distants envoyés par les processeurs voisin).

L'ordre des items virtuels est implicitement défini par l'ordre dans lequel ils sont envoyés par le processeur voisin.

Le tableau distribué possède pour chaque processeur voisin une liste de cases virtuelles (toujours contigües, la liste est donc implicitement définie par l'indice de la première case et le nombre de cases), une liste de case distantes et une liste de cases communes.

Les tableaux distribués d'items géométriques sont construits par la même méthode que les tableaux de champs. Par exemple, pour construire le tableau distribué des coordonnées des sommets, on crée un tableau distribué à partir des coordonnées des sommets réels.

Cas particulier des tableaux contenant des indices d'items géométriques

De nombreux tableaux contiennent non pas des valeurs numériques (température, coordonnée, etc...) mais des indices d'items géométriques. Par exemple, le tableau des éléments contient pour chaque élément les indices locaux des sommets dans le sous-domaine. Par conséquent, faire un échange espace virtuel du tableau d'indices n'est pas pertinent car on obtiendrait des indices de sommets sur un autre processeur.

La classe Scatter fournit des outils permettant de traduire automatiquement le contenu des tableaux d'indices en indices locaux lors de la construction des tableaux distribués.

Par exemple, la création du tableau distribué des sommets (coordonnées des sommets) est faite de façon « normale » (voir méthode Scatter :: construire_espace_virtuel_sommets)

On crée la structure d'espace virtuel du tableau à partir des informations contenues dans les Joints :

```
creer_tableau_distribue(dom, Joint::SOMMET, sommets);
```

On échange les valeurs pour les sommets communs et virtuels :

```
sommets.echange_espace_virtuel();
```

Le tableau des éléments quant à lui est crée de la façon suivante (voir Scatter::construire_structures_paralleles)

```
construire_espace_virtuel_traduction(dom, dom.nb_som(),
                                     Joint::ELEMENT /* type index */,
                                     Joint::SOMMET /* type valeur */,
                                     dom.zone(0).les_elems());
```

Cette méthode commence par échanger les valeurs comme un tableau traditionnel, sachant que le tableau est indexé par des indices d'éléments. Puis elle traduit les valeurs reçues en indices locaux, sachant qu'il s'agit d'indices de sommets.

Les mêmes mécanismes sont utilisés pour créer l'ensembles des tableaux distribués (faces, faces de bord, champs aux sommets, aux éléments, aux faces, etc...)

Construction des structures de données

Dans Scatter ::interpreter :

1) Remplissage des items réels de la classe Domaine et Zone, et les joints :
   Scatter::lire_domaine

(tableau des **domaine.sommets**, tableau des **zone.elements**, **zone.bords/raccords/joints.les_faces.sommets**)

On suppose que les joints contiennent les tableaus faces_joint[i].sommets() (sommets de joint isolés) et faces_joint[i].faces().les_sommets (faces de joint) remplis.

2) construire_correspondance_sommets_par_coordonnees

Pour l'instant ce n'est pas fait au découpage. Cet algorithme recherche la correspondance entre les sommets communs sur les domaines voisins (sommet i sur domaine 1 identique au sommet j sur domaine 2). A terme cette opération sera faite par le découpeur (cette information est d'ailleurs obligatoire dans le format MED fichier parallèle).

On remplit **joints[i].joint_item(SOMMET).items_communs()** et **joints[i]. joint_item(SOMMET).renum_items_communs()**)

3) construire_structures_paralleles

On fait les opérations suivantes :

- calculer_espace_distant_elements (en fonction de l'épaisseur de joint, chercher les éléments à envoyer aux voisins). Algorithme à corriger pour le VDF épaisseur 2 (il manque des éléments). remplissage de **joints[i]. joint_item(ELEMENT).items_distants()/nb_items_virtuels()**
- calculer_espace_distant_sommets (en fonction des éléments distants, chercher les sommets à envoyer aux voisins (tous les sommets des éléments distants)). Remplissage de **joints[i].joint_item(SOMMET).items_distants()/ nb_items_virtuels()**
- construire_espace_virtuel_sommets : crée l'espace virtuel du tableau des sommets (en fonction des sommets distants et communs). Complète **domaine.sommets**
- construction de l'espace virtuels des éléments (utilise construire_espace_virtuel_traduction). Complète **zone.elements**
- reordonner_faces_de_joint (on reordonne les lignes du tableau **faces_joint [i].faces().les_sommets** pour que les faces apparaissent dans le meme ordre sur un processeur et son voisin.

La suite de l'initialisation a lieu dans ZoneXXX ::discretiser

D'abord ZoneVF ::discretiser

4) Construction des faces réelles (tableaux faces_sommets, faces_voisins et elem_faces)

C'est fait dans faces_builder.creer_faces_reeles. Les faces apparaissent dans un ordre arbitraire (faces frontieres au debut quand- meme)

On remplit aussi :

- num_premiere_face des frontières
- joints[i].joint_item(Joint ::FACE).items_communs() (indices des faces de joint dans la zone_VF)

5) Reordonner les faces selon la discretisation

En VDF : d'abord les faces frontieres (inchangees), puis les faces x, puis y, puis z.

En VEF : d'abord les faces non standard (dont les volumes de controle sont modifiés par les Cl), puis les autres.

Attention, on met à jour les tableaux qui contiennent des indices de faces (**elem_faces**, et **joints[i]. joint_item(Joint ::FACE).items_communs()**)

5bis) Remplissage de **joints[i]. joint_item(Joint ::FACE).renum_items_communs()**

6) Remplissage des tableaux face_voisins des frontières
7) calculer_espace_distant_faces (recherche des faces à envoyer aux voisins en fonction des éléments distants) **joints[i].joint_item(Joint ::FACE).item_distants** et) **joints[i].joint_item(Joint ::FACE).nb_items_virtuels**
8) calculer_espace_distant_faces_frontieres (recherche des faces frontières distantes) **joints[i].joint_item(Joint ::FACE_FRONTIERE).item_distants** et **joints[i]. joint_item(Joint ::FACE_FRONTIERE).nb_items_virtuels**
9) parallélisation du tableau faces_sommets (construction des descripteurs, items communs, espaces virtuels... avec construire_espace_virtuel_traduction
10) parallélisation du tableau faces_voisins (idem)
11) parallélisation du tableau elem_faces (idem)
12) remplissage de ind_faces_virt_bord (méthode « creer_faces_virtuelles » mal nommée)
13) Remplissage d'un tas de tableaux volumes, centres de gravité, etc (attention, ils ne sont pas « parallèles » : dimension(0) vaut parfois le nombre d'items total et non le nombre d'items reels.
14) Appel à remplir_elem_faces() : Le nom est trompeur, voir implémentation en VEF (ça parallélise le tableau ind_faces_nstd et ça remplit ind_faces_virt_non_std_

La suite dans ZoneXXX ::discretiser :

Modification des tableaux faces voisins pour respecter certaines convention VDF ou VEF.

Tableaux supplémentaires spécifiques à la discrétisation (normales, orientation, h...)

Mécanismes génériques de gestion des tableaux parallèles

La plupart des tableaux distribués de Trio_U sont indexés par des items géométriques. Exemples :

- Pour les tableaux Domaine ::sommets, et les champs de valeurs aux sommets, le premier indice du tableau est un numéro local de sommet dans le domaine.
- Pour les tableaux Zone ::mes_elems, les champs de valeurs P0, le tableau elem_faces, le tableau xp, etc... le premier indice est un numéro local d'élément dans la zone.
- Pour les tableaux Zone_VF ::face_sommets_, Zone_VF ::face_voisins_, les champs aux faces, etc... le premier indice est un numéro local de face dans la zone discrète.
- Pour les champs frontières, c'est un indice de face de frontière (pour les faces réelles, c'est le même indice que la face dans la zone_dis car les faces frontières sont regroupées au début de la zone_dis, pour les faces virtuelles, l'indice est différent).
- On trouvera des tableaux similaires indexés par les arêtes...

Pour créer la structure de tableau distribué de ces tableaux, on a besoin :

- de la liste des items communs (sommets communs ou faces communes, faces de bord communes, arêtes communes) pour permettre l'échange des items communs,
- de la liste des items distants (sommets, éléments, faces, faces de bord, arêtes) pour l'échange des espaces virtuels.

Ces données sont pour l'instant stockées dans la classe Joint et accessibles par les méthodes **Joint_Item & set_joint_item(type_item)** pour l'initialisation, et **const Joint_Item & joint_item()** par la suite pour l'utilisation des données. Joint_Item propose les services suivants :

- const ArrOfInt &items_distants() const (renvoie les indices locaux des items)
- const ArrOfInt & items_communs() const (renvoie les indices locaux des items communs avec le processeur voisin)
- const IntTab & renum_items_communs() const; (renvoie un tableau de correspondance entre indice local d'un item commun l'indice du même item sur le processeur voisin). La colonne 1 contient les memes indices que le tableau items_communs.

Type_Item est pour l'instant : enum Type_Item { SOMMET, ELEMENT, FACE, ARETE, FACE_FRONT };

La méthode static void Scatter ::creer_tableau_distribue permet de construire automatiquement la structure d'espace virtuel d'un tableau distribué (d'entiers ou de réels). Elle n'échange cependant aucune valeur (ni item commun ni item virtuel).

Spécifications des géométries distribuées

Tableau des sommets :

Coordonnées 2D ou 3D des sommets du domaine.

Ce tableau a une structure de tableau distribué normal, avec items communs et espace virtuel. Le contenu de l'espace virtuel est obtenu par echange_espace_virtuel standard.

Tableau des éléments (Zone ::les_elems):

Pour chaque maille, indices des sommets de la maille dans les sommets.

Tableau distribué standard, avec espace virtuel (pas d'items communs). Le contenu de l'espace virtuel est un indice local de sommet obtenu par construire_espace_virtuel_traduction.

Tableau des faces (Zone_VF ::face_sommets):

Pour chaque face, indices des sommets de la face. L'ordre des sommets dans chaque face est déterminé à partir de Elem_geom_base :: get_tab_faces_sommets_locaux().

Tableau distribué standard avec espace virtuel et items communs. Le contenu du tableau est un indice local obtenu par construire_espace_virtuel_traduction.

Tableau Zone_VF ::elem_faces_ :

Pour chaque maille, indices de ses faces dans Zone_VF ::face_sommet

Tableau distribué standard indexé par les éléments (avec espace virtuel). Contenu obtenu par construire_espace_virtuel_traduction. La i-ième face d'une élément est définie à partir de l'élément de référence (tableau Elem_geom_base :: get_tab_faces_sommets_locaux()) et de l'ordre des sommets de l'élément dans Zone ::les_elems

Tableau Zone_VF ::face_voisins_ :

Pour chaque face, indices de ses éléments voisins. Si une face n'a qu'un voisin, l'autre vaut « -1 ».

Tableau distribué standard indexé par les faces (avec espace virtuel et items communs). Contenu obtenu par espace_virtuel_traduction, puis application des conventions spécifiques à chaque discrétisation :

En VDF : face_voisin_(i ,0) est l'élément qui a une coordonnée inférieure à celle de la face dans la direction normale, face_voisin_(i,1) est celui qui a une coordonnée supérieure. Par conséquent, pour les faces de bord et les faces virtuelles, l'un des deux voisins peut valoir –1. De plus, l'ordre des voisin0 et voisin1 est identique sur tous les processeurs qui connaissent cette face.

En VEF: Aucun ordre particulier entre face_voisin_(i,0) et face_voisin_(i,1) pour les faces ayant 2 voisins. Pour les faces ayant un seul voisin, le voisin connu est toujours dans face_voisin_(i,0), l'autre vaut –1. **Attention** : l'ordre des voisins n'est pas forcément identique sur tous les processeurs qui partagent une face.

Découpeur de maillages

Le découpeur est organisé autour de trois classes :

class Decouper : c'est un *interprete* qui lit les paramètres du partitionneur et effectue les operations demandees (partition du domaine, écriture des fichiers . Zones, écriture du fichier de partitionnement...)

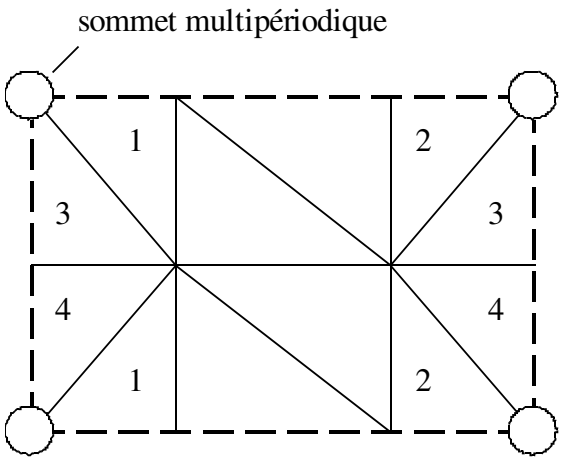
class Partitionneur_base : les classes dérivées de cette classe implémentent les différents algorithmes de partitionnement (Metis, Tranche, etc...)

class DomainCutter : cette classe fournit les outils permettant de générer les sous-domaines de calculs a partir d'un domaine complet et d'un tableau de découpage.

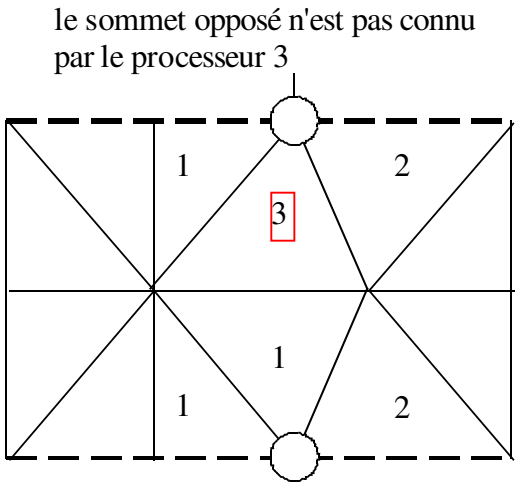
Spécifications communes à toutes les classes Partitionneur

La méthode `construire_partition()` doit construire une partition correcte compte tenu des bords périodiques qui ont été déclarés. Les contraintes à respecter sont les suivantes:

- si un face de bord périodique réelle est sur un processeur, la face opposée doit aussi être sur ce processeur (autrement dit, l'élément en vis-à-vis doit être mis dans le même domaine).
- le tableau renum_som_perio doit indiquer le même sommet réel sur tous les processeurs. Pour cela, si un sommet est multipériodique, les processeurs qui le possèdent doivent posséder aussi tous les autres sommets multipériodiques associés à ce sommet. Le découpage suivant est par exemple interdit:



- Pour la même raison, un processeur ne doit pas posséder de sommet périodique isolé. Le découpage suivant est interdit



Afin de respecter ces contraintes, les partitionneurs peuvent appeler les méthodes `Partitionneur_base::corriger_bords_avec_graphe()` et `Partitionneur_base::corriger_bords_avec_liste()` qui corrigent une partition pour assurer les propriétés suivantes:

- Si un domaine possède une face périodique, alors il possède la face périodique associée.
- Il n'y a pas de sommet périodique isolé sur un domaine (si un domaine possède un sommet adjacent à une face périodique, il possède au moins une des faces périodiques adjacentes). Donc on peut connaître les sommets periodiques d'un domaine en parcourant les faces périodiques.
- Si un sommet est multipériodique, tous les processeurs qui possèdent ce somment possèdent également les autres sommets périodiques associés.

Classe DomaineCutter

Le rôle de cette classe est de construire des sous-domaines parallèles à partir du domaine complet et d'un tableau de partition. Elle effectue les opérations suivantes pour chaque sous-domaine :

- recherche des sommets réels du sous-domaine (ensemble des sommets des élements affectés au sous-domaine).

- construction du tableau des sommets, et du tableau des éléments avec des indices de sommets locaux.
- construction des bords du sous-domaine.
- recherche des sommets de joint (sommets communs entre le sous-domaine et un sous-domaine voisin), ordonnés de sorte qu'ils aient le même ordre sur le sous-domaine voisins.
- recherche des faces de joint.

Pour l'instant, cette classe détermine aussi les éléments distants mais cette opération est facultative et peut être réalisée lors du "Scatter" (notamment pour permettre l'utilisation d'un maillage découpé au format MED ne contenant pas les espaces distants).