Mise en place d'une simulation instationnaire avec le schéma à pas de temps local de FastS

1 Introduction

Dans ce rapport, on indique la procédure pour faire un calcul avec le schéma à pas de temps local de FastS. On explique tout d'abord comment générer les arbres t et tc avec toutes les données nécessaires au fonctionnement du schéma (les niveaux en temps, les données supplémentaires aux raccords pour assurer la transition entre zones de niveau en temps différent...). Ensuite, on indique comment lancer le calcul (load de t et tc, mots clés à renseigner dans la carte de commande.)

2 Création de t et tc à l'aide du script "prep_dtLocUnsteady.py"

La création des arbres t et tc se fait à l'aide du script Python :
$CASSIOPEE/Apps/PModules/FastS/prep_dtLocUnsteady.py

En input de ce script, il faut un maillage muni du **champ initial**, **des conditions aux limites** et **des raccords**. Une capture d'écran de ce script est visible sur la figure 1. Pour le lecteur intéressé, nous présentons les différentes fonctions qui constituent ce script.

```
### Ajout des ghost-cells pour permettre le calcul de la CFL
C.addState2Node__(t, 'EquationDimension', dim)
t = Internal.addGhostCells(t, t, 2, adaptBCs=1, fillCorner=0)
if (dim == 2):
    t = T.addkplane(t)
    t = T.contract(t, (0,0,0), (1,0,0), (0,1,0),0.025)
    t = T.makeDirect(t)



### Met la CFL en chaque cellule dans l arbre t ###
t, exposant_max = FastS.computeCFL_dtlocal(t)
#################################################



### Decoupe le maillage en zones de niveaux en temps different ###
t = FastS._decoupe2(t, exposant_max, NP = NP)
#################################################################


#### On remet les GhostCells car supprimees dans decoupe2
t = Internal.addGhostCells(t, t, 2, adaptBCs=1, fillCorner=0)
if (dim == 2):
    t = T.addkplane(t)
    t = T.contract(t, (0,0,0), (1,0,0), (0,1,0),0.025)
    t = T.makeDirect(t)


tc = C.node2Center(t)

tc = X.setInterpData2(t, tc, nature=1, loc='centers', storage='inverse',
                      sameName=1, method='lagrangian',dim=dim)
tc = C.rmVars(tc, 'FlowSolution')
tc = C.rmVars(tc, 'CellN')


C.convertPyTree2File(t, 't.cgns')
C.convertPyTree2File(tc, 'tc.cgns')
```

Figure 1: Capture d'écran du script prep_dtLocUnsteady.py

Premièrement, comme le montre la figure 1, les Ghostcells sont ajoutées pour permettre le calcul du nombre CFL en chaque cellule par la fonction "computeCFL_dtLocal(t)".

2.1 Nombre CFL en chaque cellule : fonction "computeCFL_dtLocal(t)"

L'argument de cette fonction est l'arbre t, c'est à dire le maillage donnée en input du script "prep_dtLocUnsteady.py", auquel viennent d'être ajouté les Ghsot cells.

La fonction "computeCFL_dtLocal(t)" **crée un champ "centers:CFL" dans t et le remplit en calculant le nombre CFL en chaque cellule**. Le nombre CFL se calcule avec un pas de temps égal à 1 (nous sommes en phase de pré-traitement, aucun pas de temps n'a été encore défini, d'où cette valeur arbitraire). Le calcul du nombre CFL se fait à l'aide de la fonction C "prep_cfl.cpp", qui appelle la fonction "calcul_cfl.for". Ces deux fonctions se trouvent dans $CASSIOPEE/Apps/PModules/FastS/FastS/Compute/DTLOC. Remarquons que "calcul_cfl.for" a la même structure que "cpcfl3.for" (la fonction de FastS pour le calcul de la CFL au cours de la simulation), à deux exceptions près :

- Comme dit plus haut, le pas de temps est fixé à 1
- La CFL est écrite dans un tableau "ipt_cfl" qui n'est autre que le pointeur du champs "centers:CFL" que l'on veut remplir. Dans "cpcfl3.for", ce pointeur n'existe pas, le CFL en chaque cellule est écrit dans un tableau FastS appelé "cfl".

Lorsque que le champs "centers:CFL" est rempli, les pas de temps minimum et maximum ($dtmin$ et $dtmax$) sur le maillage sont calculés :

$$dtmin = 1/max(CFL) \tag{1}$$

$$dtmax = 1/min(CFL)$$

Enfin, l'exposant maximal $emax$ est calculé. Il s'agit du plus grand entier tel que :

$$2^{emax} \times dtmin \leq dtmax \tag{2}$$

ce qui donne :

$$emax = \left\lfloor \frac{log\left(\frac{dtmax}{dtmin}\right)}{log(2)} \right\rfloor \quad \text{(arrondi à l'entier inférieur)} \tag{3}$$

IMPORTANT : Le niveau en temps maximal $niveauMax$ vaut : $niveauMax = 2^{emax}$. On a une nouvelle définition de $dtmax$:

$$dtmax = niveauMax \times dtmin \tag{4}$$

C'est cette définition de $dtmax$ qu'on garde pour la suite

Maintenant que le nombre CFL est renseigné dans chaque cellule, le maillage est prêt à être découpé en zones de niveau en temps différents à l'aide de la fonction "decoupe2(t, exposant_max, NP)".

IMPORTANT : Le champs centers:CFL sert seulement à la découpe du maillage en zones de niveaux en temps différents. Dans le solveur, il n'intervient pas.

2.2 Découpe du maillage en zones de niveau en temps différents : fonction "decoupe2(t, exposant_max, NP)"

Les arguments de cette fonction sont :

- l'arbre t, c'est à dire le maillage en input du script "prep_dtLocUnsteady.py", auquel les Ghsotcells, puis le champs centers:CFL ont été rajoutés
- l'exposant max calculé par la fonction "computeCFL_dtLocal(t)"
- le nombre de processus MPI NP sur lequel le calcul va tourner

Premièrement, cette fonction supprime les Ghostcells car elle va découper le maillage.

Deuxièmement, la fonction découpe le maillage en blocs de $25 \times 25 \times 25$ cellules, détermine le nombre CFL maximal dans le bloc ($CFLmax\_bloc$) et en déduit le pas de temps du bloc ($dtbloc$) : $dtbloc = 1/CFLmax\_bloc$.

Remarque : L'utilisateur peut changer la taille du bloc via la variable "taille_bloc" en début de fonction. Mais l'expérience a montré qu'une valeur de 25 est un bon compromis : une valeur plus grande va engendrer un découpage plus grossier ce qui est susceptible de diminuer l'efficacité du schéma. Une valeur plus faible va engendrer un découpage plus fin et précis mais il y aura aussi beaucoup de zones et donc beaucoup de temps passés dans les transferts.

Troisièmement, elle calcule pour chaque bloc un exposant $ebloc$ en fonction de $dtbloc$. L'exposant $ebloc$ est le plus petit entier tel que :

$$\frac{dtmax}{2^{ebloc}} \leq dtbloc. \tag{5}$$

Explication : Dans le schéma à pas de temps local, seuls les pas de temps $dtmax$, $dtmax/2^1$, $dtmax/2^2$,...,$dtmax/2^{emax}$ sont utilisés. Il n'y aucune raison que $dtbloc$ soit égal à un de ces pas de temps, comme le montre la figure 2. En résolvant l'équation (5), on cherche **le pas de temps du schéma qui se rapproche le plus de $dtbloc$, en étant plus petit que lui** (pour assurer la stabilité). Sur la figure 2, il s'agit de $dtmax/2^2$. Si on assignait le pas de temps $dtmax/2^1$ à la zone de pas de temps $dtbloc$ le calcul planterait car $dtmax/2^1$ est plus grand que $dtbloc$. Sur cet exemple, l'exposant qui sera assigné à la zone de pas de temps $dtbloc$ est donc $exbloc = 2$.

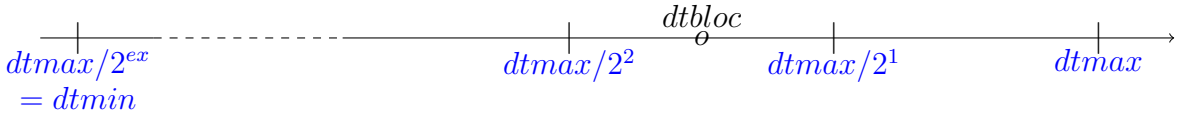

Figure 2: Schéma explicatif représentant les pas de temps utilisés dans le schéma à pas de temps local et le pas de temps $dtbloc$

En utilisant (4), (5) devient :

$$\frac{2^{emax} \times dtmin}{2^{ebloc}} \leq dtbloc, \tag{6}$$

avec $emax$ l'exposant maximal défini par (2) et $dtmin$ le pas de temps minimal défini par (1).

Finalement l'exposant *ebloc* est donné par :

$$ebloc = \left\lceil \frac{log\left(\frac{2^{emax} \times dtmin}{dtbloc}\right)}{log(2)} \right\rceil \quad \text{(arrondi à l'entier supérieur)} \tag{7}$$

Le niveau en temps du bloc $niveauBloc$ est donc : $niveauBloc = 2^{ebloc}$

Quatrièmement, la fonction contrôle si parmi tous les blocs adjacents, le rapport entre leur niveau en temps ne dépasse pas 2. Si c'est le cas, elle modifie les niveaux en temps des blocs concernés.

Cinquièmement, les blocs de même niveau en temps sont mergés et le champ "centers:niveaux_temps" est créé dans les zones obtenues à l'issue des merge.

IMPORTANT : Contrairement au champs CFL, le champs centers:niveaux_temps est très important pour le solveur puisqu'il permet de connaître le pas de temps pour chaque zone. Par exemple, pour une zone dont le champ centers:niveaux_temps vaut 8, le pas de temps appliqué dans cette zone sera $dtmax/8$.

Sixièmement, si NP > 0 le maillage est distribué sur NP processus MPI en suivant la stratégie spécifique au schéma à pas de temps local : chaque niveau en temps est distribué de manière indépendante sur NP processus MPI.

2.3 Fonction setInterpData2

Cette fonction se trouve dans : $CASSIOPEE/Apps/Modules/Connector/OversetDataDtlocal.py

Elle est très similaire à la fonction "classique" setInterpData. Comme le montre la figure 3 , elle ajoute des données dans les noeuds "ZoneSubRegion_t". Il faudrait la fusionner avec la fonction setInterpData car elle peut passer à côter de certaines mises à jour faites sur setInterpData. Les données ajoutées sont utilisées dans la fonction "$FAST/FastS/Compute/DTLOC/dtlocal2para_c".Elles sont nécessaires pour assurer la transition entre zones de niveau en temps différent.

Les données ajoutées sont :

- $PointRange$ est la fenêtre des points receveurs
- $PointRangeDonor$ est la fenêtre des points donneurs
- $DirReceveur$ est la normale sortant de la zone receveuse au niveau du raccord
- $DirDonneur$ est la normale sortant de la zone donneuse au niveau du raccord
- $Transform$ est la transformation entre les indices (i, j, k) de la donneuse et ceux de la zone receveuse
- $PointPivot$ est le symétrique dans la zone receveuse du point $(imin, jmin, kmin)$ de la zone donneuse.
- $Profondeur$ est le débordement de la zone receveuse sur la zone donneuse (pour un raccord Match, $Profondeur$ vaut donc 2 mais en IBC, $Profondeur$ peut valoir 3,4,5..).
- $NMratio$ est le ratio entre le nombre de points de la zone donneuse et celui de la zone receveuse (1/2, 1/3...)

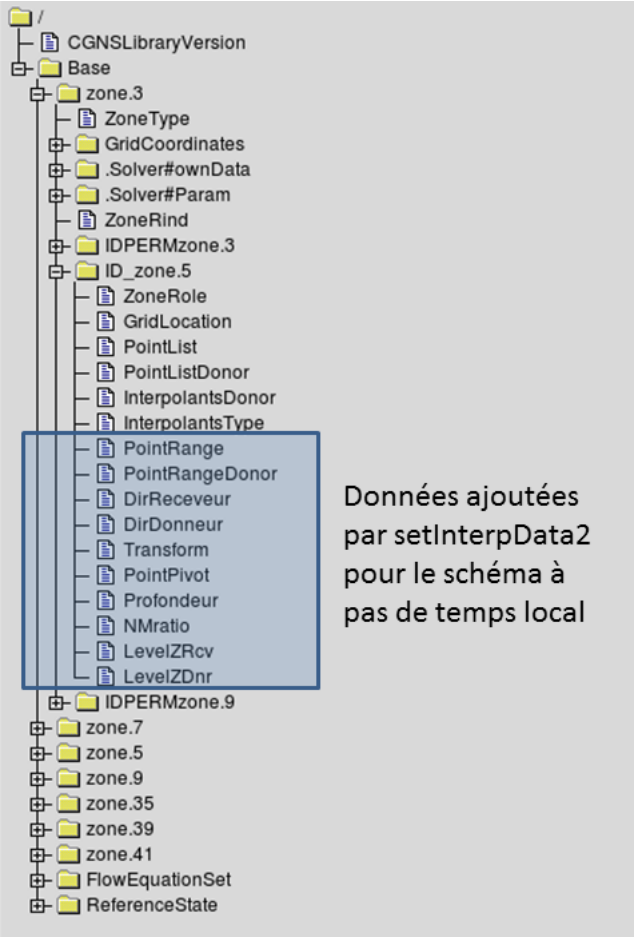

Figure 3: Capture d'écran d'un noeud "ZoneSubRegion_t" avec les données nécessaires au schéma à pas de temps local ajoutées par la fonction setInterpData2

- $LevelZRcv$ est le niveau en temps de la zone receveuse
- $LevelDnr$ est le niveau en temps de la zone donneuse

Modifications possibles ou non :

- $LevelZRcv$ et $LevelDnr$ sont **essentiels et ne peuvent pas être supprimés.**
- $PointRange$, $PointRangeDonor$, $Transform$, $PointPivot$ et $NMratio$ sont nécéssaires pour faire la correction de flux permettant d'assurer la conservativité et **ne peuvent pas être supprimés**.
- $DirReceveur$, $DirDonneur$, $Profondeur$ peuvent être supprimés à condition de modifier les routines "copyflux_rk3localpara", "copy_rk3localpara", "copyflux_rk3local2para", "interp_rk3local3para" et "interp_rk3local4para" (qui sont appelées dans "dtlocal2para_c.cpp"). Les modification doivent permettre le remplissage des cellules transférées avec PointListDonor. Ce n'est pas le cas actuellement, ces cellules sont remplies avec $PointRangeDonor$, $DirDonneur$ et $Profondeur$. On rappelle que le remplissage de ces cellules permet d'assurer la transition entre zones de niveau en temps différent.

3 Lancer une simulation avec le schéma à pas de temps local

3.1 Loader le t et le tc

En séquentiel (pas de MPI), le load de t et tc se fait "normalement", comme pour les autres schémas. **En MPI, il faut loarder le tc avec la fonction loadFile et l'argument exploc=True, comme montré sur la figure 4**. Le graph est alors une liste de plusieurs sous-graphes, avec un sous-graphe pour sous-itération. Le t est loadé normalement.

```
tc, graph = Fast.loadFile(fileName='tc.cgns', split='single', graph=True,  mpirun=True, exploc=True)
```

Figure 4: Capture d'écran de la fonction loadFile permettant de loader l'arbre tc en vue d'un calcul avec le schéma à pas de temps local

ATTENTION ! Quand on fait du MPI en passant par la couche c, il y un "blocage" des processus. La couche Python fonctionne bien. PISTE : dans FastC/setInTerpTrasnfersFastS, le fait de commenter la condition "if (pair_of_queue == NULL)" (ligne 165) permet de d'empêcher ce blocage mais cela crée une fuite mémoire. Voir avec Xavier.

3.2 Mots clés à mettre dans la carte de commande

Comme le montre la figure 5, il faut fixer la clé "temporal_scheme" du dictionnaire numb à "explicit_local" pour que le schéma à pas de temps local soit activé.

La clé "explicit_local_type" concerne la conservativité. Quand elle vaut 1, le schéma à pas de temps local est en mode "conservatif" : à chaque interface entre zones de pas de temps différent une correction de flux est appliquée pour rendre le schéma conservatif. Par défaut la clé "explicit_local_type" vaut 0 : aucune correction de flux n'est appliquée et le schéma n'est pas conservatif.

```
numb = {}
numb["temporal_scheme"]     = "explicit_local"
numb["explicit_local_type"] = 1
```

Figure 5: Capture d'écran des mots clés à mettre dans la carte de commande