Introduction à la librairie HySoP v.2.0

Jean-Baptiste Keck

Laboratoire Jean Kuntzmann - Grenoble

28 mars 2018

Table des matières

1. Présentation de la librairie
2. Exemple de simulation réalisée avec HySoP
3. HySoP en pratique

1. Présentation de la librairie
   - Introduction
   - Concepts
   - Schéma d'utilisation
2. Exemple de simulation réalisée avec HySoP
   - Transport turbulent d'un scalaire passif
3. HySoP en pratique
   - Domaines
   - Discrétisations et topologies
   - Partition de grilles cartésiennes
   - Variables
   - Opérateurs
   - Vue d'ensemble d'une simulation
   - Implémenter son propre opérateur discret

Présentation de la librairie

- HySoP est une librairie développée au sein du Laboratoire Jean Kuntzmann à Grenoble. Il est le résultat de plusieurs années de développements destinés à la recherche. Chacun ajoute les modules qui lui sont nécessaires.
- A l'origine Scales, c'était un code Fortran pour étudier les méthodes semi-Lagrangiennes (remaillage).
- Une surcouche utilisateur en Python a été ajoutée et Scales est devenu HySoP (Hybrid Simulation with Particles).
- La surcouche Python permet maintenant d'intégrer sans trop de difficultés du code C, C++ et Fortran mais aussi du code GPU en passant par le standard OpenCL.
- La librairie a pour objectif de tourner sur des architecture parallèles hybrides (cluster avec coprocesseurs et/ou accélérateurs type GPU).

Concepts

La librairie s'articule autour de quatre concepts simples :

- **Un problème :** Il correspond au problème mathématique global que l'on cherche à résoudre.
- **Des opérateurs :** Généralisation de la notion mathématique d'opérateur, il définit un opérateur mathématique élémentaire comme le gradient ou le laplacien et, plus généralement, un terme d'une équation du problème. Il peut être vu comme un sous problème élémentaire.
- **Des variables :** Ce sont les entités mathématiques mises en relation par les opérateurs au sein du problème, qu'elles soient données ou inconnues.
- **Des domaines :** Ils décrivent le contexte formel sur lequel sont définies les variables. Il comprend la spécification d'une géométrie ainsi que les conditions aux limites.

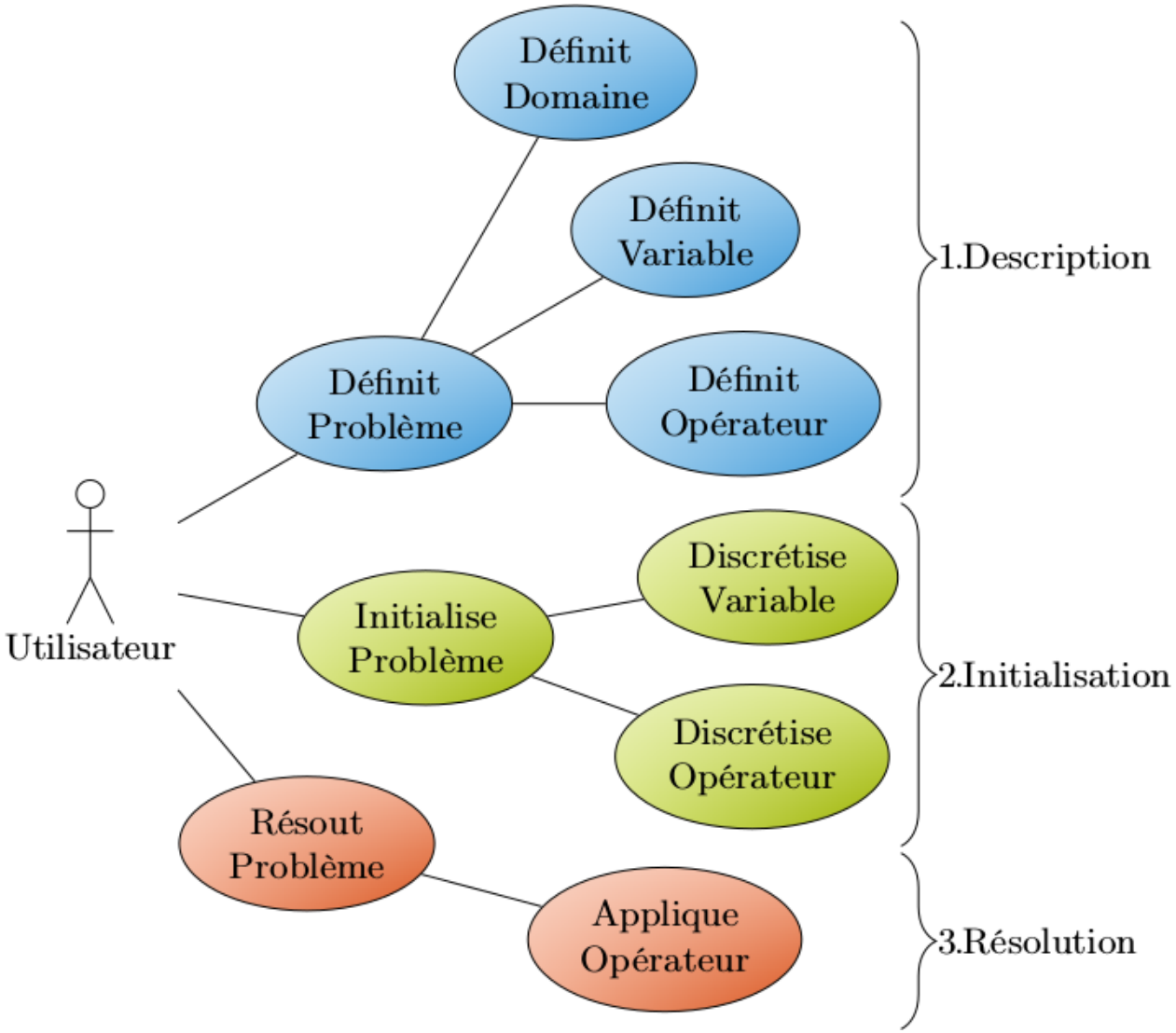
Schéma d'utilisation

Du point de vue utilisateur, la description d'un problème et sa résolution se décomposent selon les étapes suivantes :

1. Description du problème à partir d'un domaine sur lequel sont définies des variables utilisées par des opérateurs. Plus précisément l'utilisateur va créer un ou plusieurs graphes d'opérateurs.
2. Initialisation du problème à travers la discrétisation et l'initialisation des variables. Cette étape consiste en l'allocation des zones mémoires ainsi qu'au branchement des différentes méthodes numériques parmi l'ensemble des versions disponibles.
3. Résolution du problème en appliquant des opérateurs sur leurs variables.

La configuration d'un problème se fait de manière simple en utilisant les facilités du langage `Python`. Les étapes deux et trois sont maintenant quasi automatisés entièrement par la librairie.

Schéma d'utilisation



1 Présentation de la librairie
- Introduction
- Concepts
- Schéma d'utilisation

2 Exemple de simulation réalisée avec HySoP
- Transport turbulent d'un scalaire passif

3 HySoP en pratique
- Domaines
- Discrétisations et topologies
- Partition de grilles cartésiennes
- Variables
- Opérateurs
- Vue d'ensemble d'une simulation
- Implémenter son propre opérateur discret

Exemple : Transport turbulent d'un scalaire passif

Problème : Transport de scalaire passif dans un jet plan turbulent tridimensionnel incompressible. Le domaine de calcul est une boite unitaire périodique $[0,1]^3$.

Modélisation :

- Navier-Stokes incompressible en formulation vitesse-vorticité ($\mathbf{u}$,ω) pour le fluide.
- Equation de transport pour le scalaire passif θ.

$$\begin{aligned}
\mathrm{div}(\mathbf{u}) &= 0 && (1)\\
\omega &= \mathbf{rot}(\mathbf{u}) && (2)\\
\frac{\partial \omega}{\partial t} + (\mathbf{u}\cdot\nabla)\,\omega &= (\omega\cdot\nabla)\mathbf{u} + \nu\Delta\omega && (3)\\
\frac{\partial \theta}{\partial t} + (\mathbf{u}\cdot\nabla)\,\theta &= \kappa\Delta\theta && (4)
\end{aligned}$$

Exemple : Transport turbulent d'un scalaire passif

- Ecoulement : jet plan, modélisé par un champ de vitesse initialement nul dans tout le domaine excepté dans une région d'épaisseur w.
- Champ de vitesse initial : perturbé par un champ aléatoire $\mathbf{P}$ de magnitude 0.05.
- Le scalaire passif est initialisé de la même manière que le champ de vitesse.

Les valeurs initiales du champ de vitesse et du scalaire sont données par les expressions suivantes :

$$\theta(x,y,z) = \frac{1}{2}(1+0.3\,\sin(8\pi x))\left[1+\tanh\left(\frac{0.1-2|y-0.5|}{4w}\right)\right] \quad (5)$$

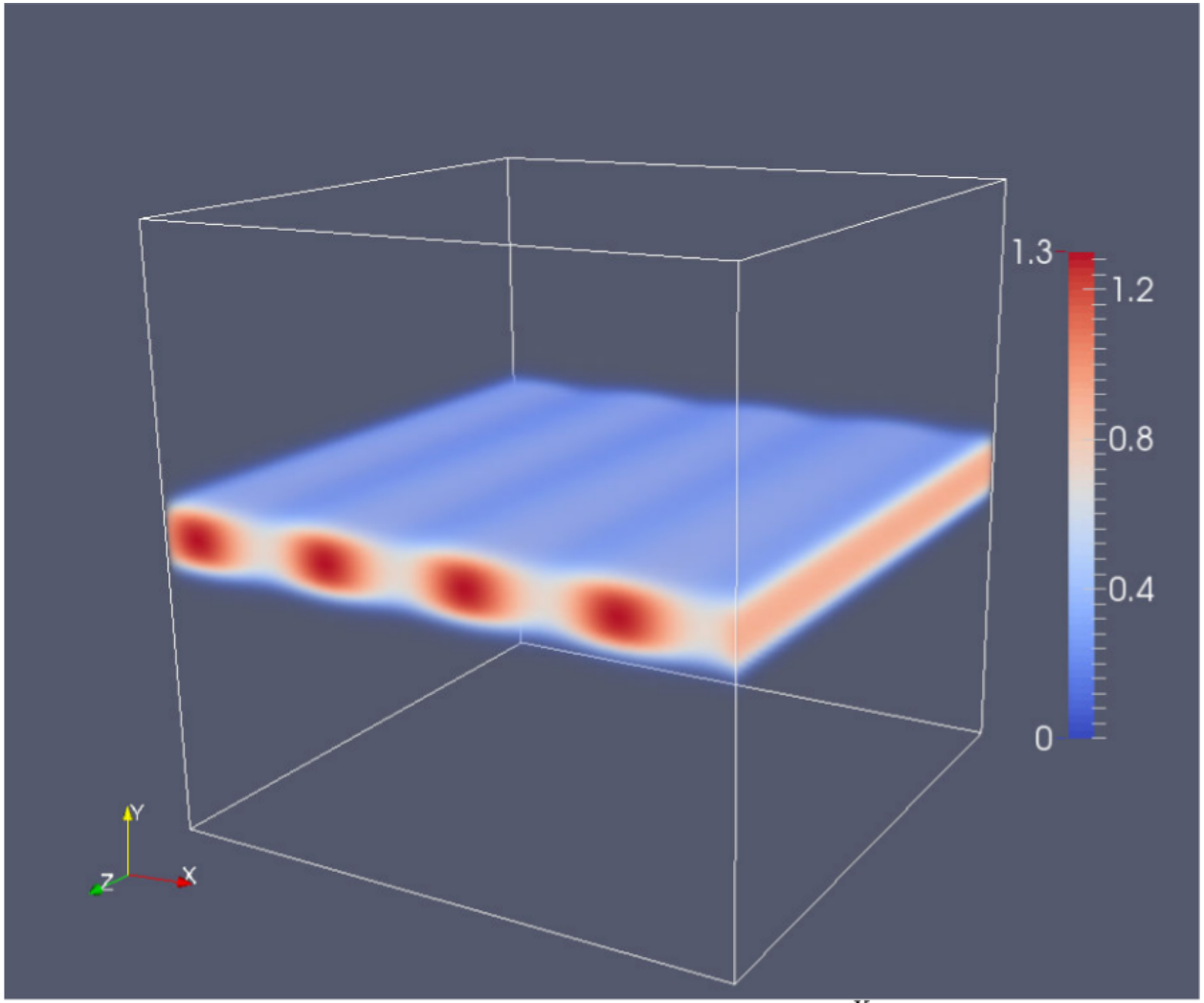
$$u_x(x,y,z) = \theta(x,y,z)\;P_x(x,y,z) \quad (6)$$

$$u_y(x,y,z) = P_y(x,y,z) \quad (7)$$

$$u_z(x,y,z) = P_z(x,y,z) \quad (8)$$

Exemple : Transport turbulent d'un scalaire passif

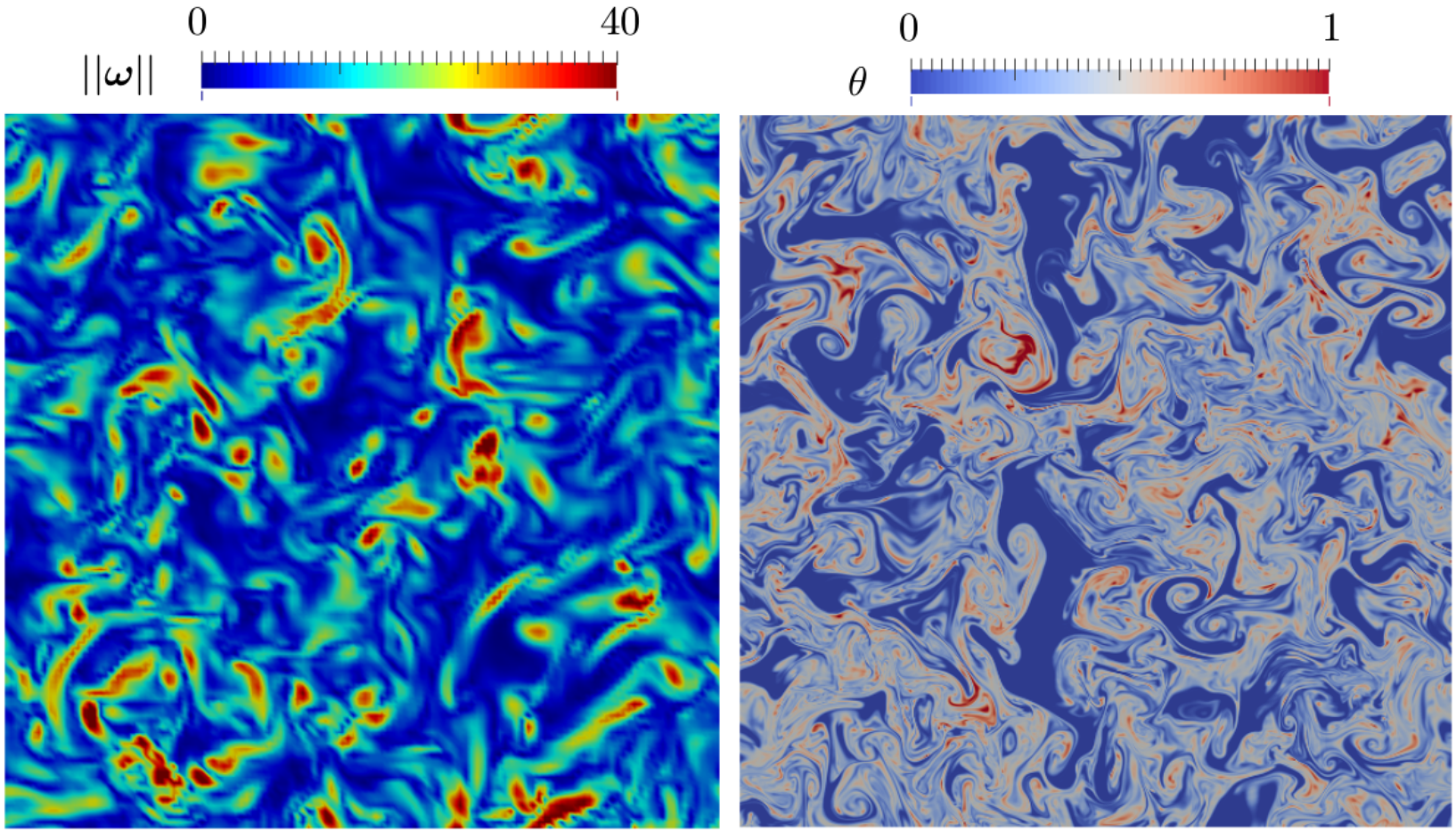
Norme du champ de vitesse initial :



Champ de vitesse initial, composante $\boldsymbol{u}^X$

Exemple : Transport turbulent d'un scalaire passif

Coupe au bout d'un certain temps de simulation :



(a) Norme de la vorticité

(b) Scalaire

1 Présentation de la librairie
- Introduction
- Concepts
- Schéma d'utilisation

2 Exemple de simulation réalisée avec HySoP
- Transport turbulent d'un scalaire passif

3 HySoP en pratique
- Domaines
- Discrétisations et topologies
- Partition de grilles cartésiennes
- Variables
- Opérateurs
- Vue d'ensemble d'une simulation
- Implémenter son propre opérateur discret

Domaines

Un domaine constitue le domaine physique où les champs et les opérateurs seront définis. Il est définit par :

- Une dimension (1d, 2d ou 3d)
- Une géométrie
- Des conditions au bord

Pour le moment `HySoP` ne propose que des domaine sous forme de "boites" :

```
# Créer un domaine 1D, avec conditions aux bords périodiques
dom = Box(length=[2.], origin=[-1.], bc=PERIODIC)

# Créer un domaine 3D
dom = Box(length=[1., 3., 5.], origin=[-1., 2., 3])

# Récuperer les coordonnées de l'origine
print dom.origin
```

Les domaines peuvent être distribués sur plusieurs processus afin de paralléliser la simulation sur plusieurs processus grâce au standard `MPI`.

Discrétisations et topologies

Une topologie décrit la discrétisation d'un domaine (locale et globale à chaque processus `MPI`). A chaque processus est attribué :

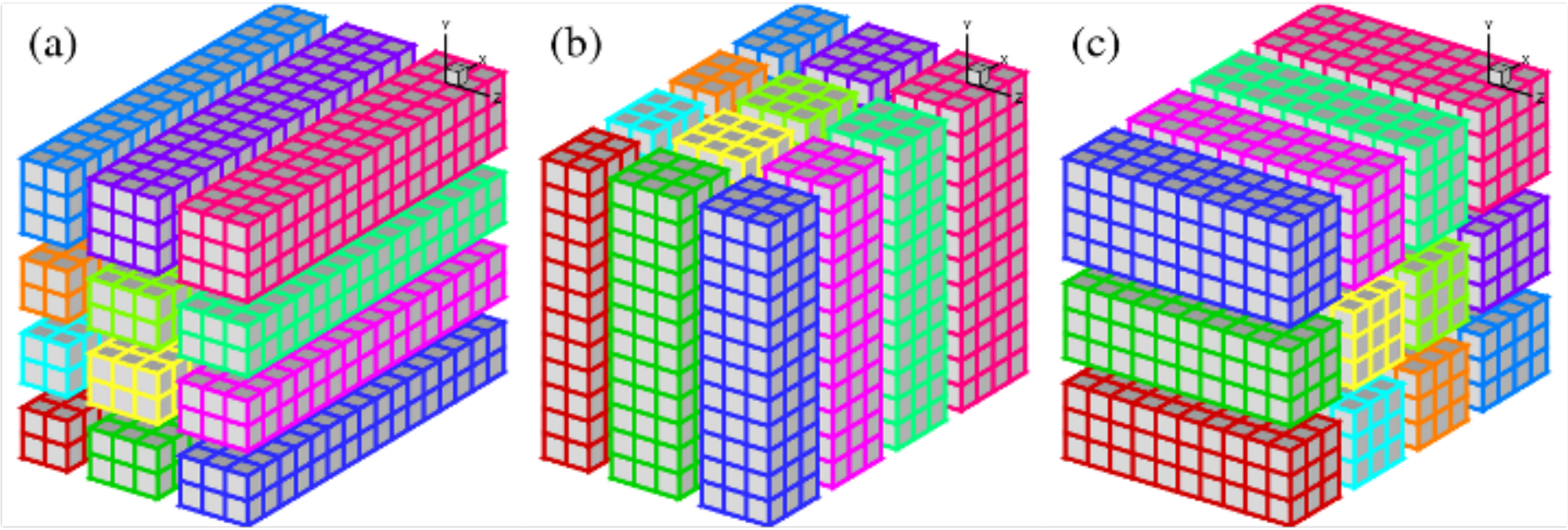
- Un domaine local discrétisé (`mesh`)
- Une position dans la grille de processus
- Des éventuels `ghosts` pour l'échange de données aux bords des domaines locaux.

Pour le moment `HySoP` ne propose que des topologies cartésiennes périodiques, mais l'on peut déjà découper de plusieurs manières :

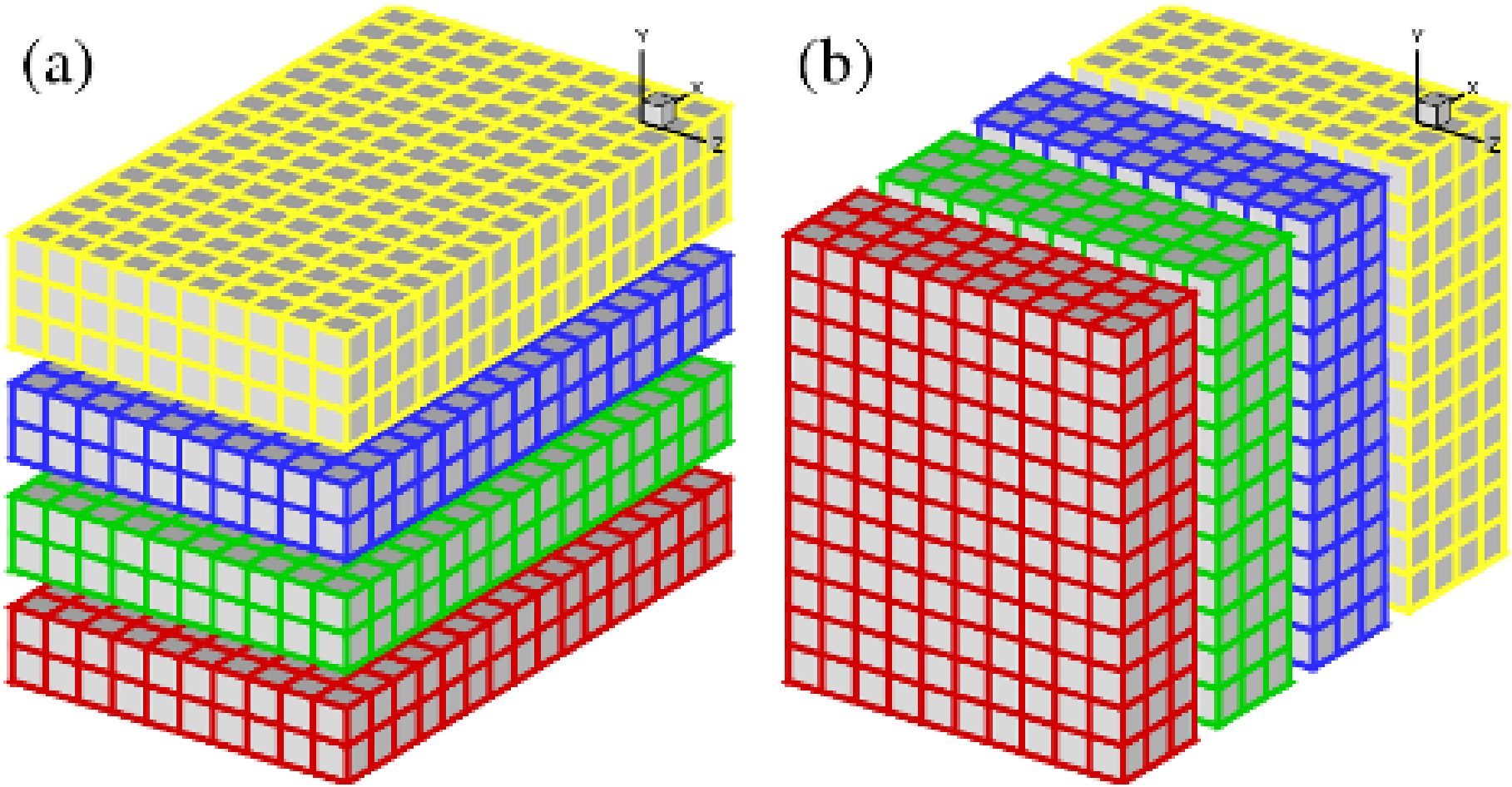
```
# Créer une discretisation et une topologie cartésienne 3d sans ghosts
d3d           = Discretization ( resolution =[256,1024,512], ghosts=None)
topo_no_ghosts = Cartesian(box,d3d)

# Créer une discretisation et une topologie cartésienne 3d avec des ghosts
# Si plusieurs processus sont présent, le domaine ne peut que être découpé
# via le dernier axe.
d3dg          = Discretization ( resolution =[256,1024,512], ghosts=[2,4,2])
topo_ghosts = Cartesian(box,d3dg, split_dir =[0,0,1])
```

Partition de grilles cartésiennes

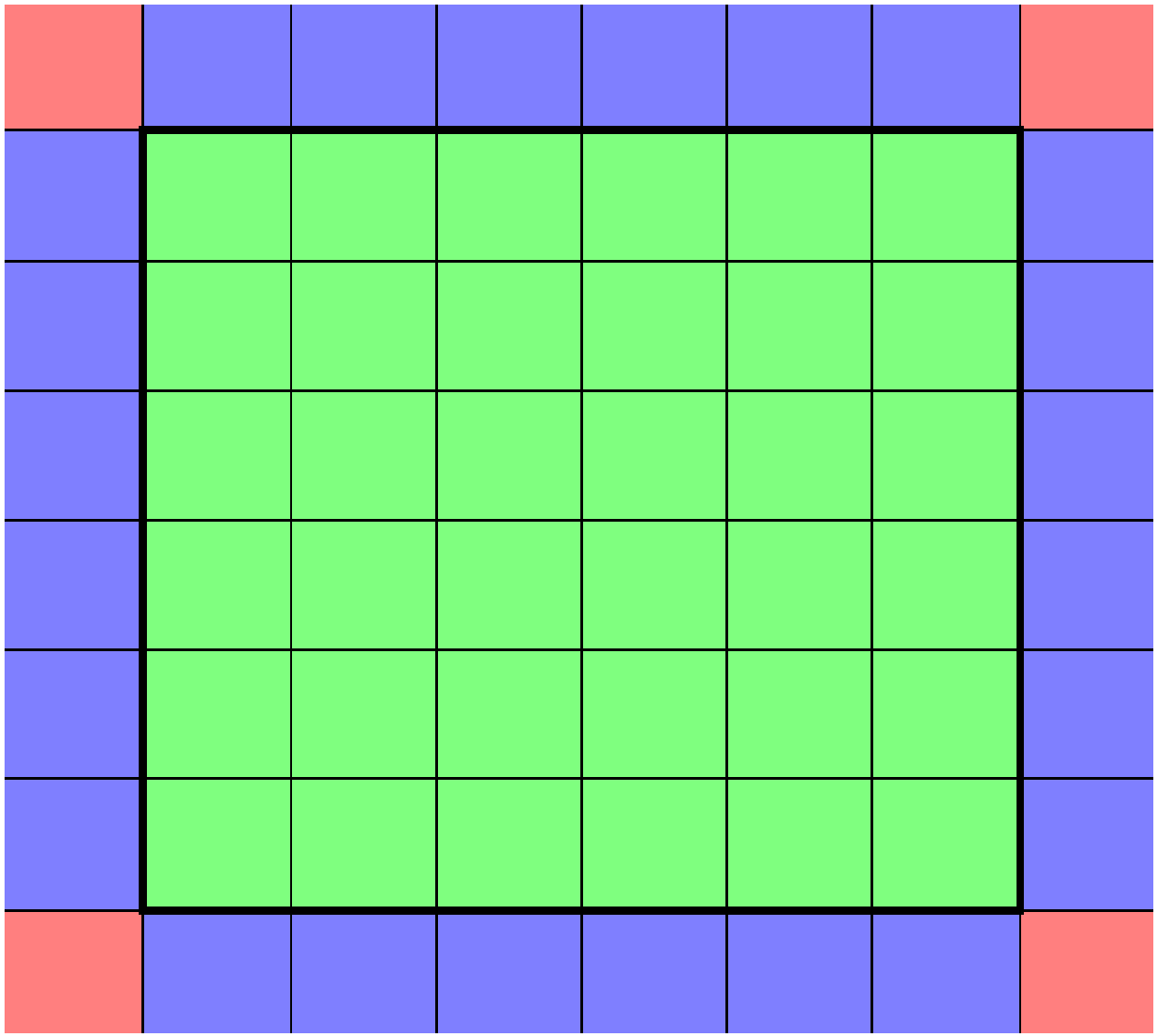


Partition de grilles cartésiennes



Partition de grilles cartésiennes

Ghosts



Variables

Il faut distinguer les champs continus et les champs discrets.

- Un champ continu est une variable abstraite (au sens mathématique), scalaire ou vectorielle, définie sur un domaine.

```
# Création d'un domaine
box = Box()

# Définition d'un champ scalaire
scal = Field(domain=box, name='s1')

# Définition d'un champ vectoriel à la dimension du domaine
vec = Field(domain=box, name='v1', is_vector=True)

# Définition d'un champ vectoriel à n composantes
n = 4
vec_n = Field(domain=box, name='Vn', nb_components=n)
```

Variables

Il faut distinguer les champs continus et les champs discrets.

- Un champ discret est la discrétisation d'un champs continu sur une topologie :

```
# Création d'un domaine
box = Box()

# Création d'un champ continu
vec = Field(domain=box, name='vn', is_vector=True)

# Discrétisation du domaine et création d'une topologie
d3d = Discretization([65, 65, 65])
topo = box.create_topology(d3d)

# Construction d'un champ discret
vd = vec.discretize(topo)

# On peut récuperer toutes les discrétisations à partir du champ continu:
vec.discrete_fields[topo] # --> retourne vd
```

Un champ continu peut avoir plusieurs discrétisations.

Opérateurs

Au niveau des opérateurs il faut aussi distinguer les opérateurs continus et les opérateurs discrets.

Un opérateur continu est un opérateur élémentaire abstrait (au sens large du terme), qui prend des champs continus en entrés et en sortie.

Il ne s'agit par forcément d'un opérateur mathématique :

- Sortie de fichier (écriture des champs)
- Affichage, print, analyse de résultats...

Les opérateurs continus réprésentent une interface commune qui rend accessible plusieurs implémentations discrètes possibles (que l'on va appeler backend), comme par exemple une implémentation `Python`, `OpenCL` ou encore `Fortran`.

Pour utiliser `HySoP` il faut découper son problème en un enchainement de sous problèmes élémentaires. Bien souvent le découpage se fait par splitting temporel et spatial d'opérateurs.

Opérateur continu

Exemple de partitionnement de problème pour l'exemple du jet plan :

$$\begin{aligned}
\omega &= \mathbf{rot}(\mathbf{u}) \\
\mathrm{div}(\mathbf{u}) &= 0 \\
\frac{\partial \omega}{\partial t} + (\mathbf{u}\cdot\nabla)\,\omega &= (\omega\cdot\nabla)\mathbf{u} + \nu\Delta\omega \\
\frac{\partial \theta}{\partial t} + (\mathbf{u}\cdot\nabla)\,\theta &= \kappa\Delta\theta
\end{aligned}$$

peut se décomposer en :

$$\begin{aligned}
\frac{\partial \omega}{\partial t} + (\mathbf{u}\cdot\nabla)\,\omega &= 0 \\
\frac{\partial \omega}{\partial t} &= (\omega\cdot\nabla)\mathbf{u} \\
\frac{\partial \omega}{\partial t} &= \nu\Delta\omega
\end{aligned}$$

Opérateur continu

Exemple de partitionnement de problème pour l'exemple du jet plan :

$$\begin{aligned}
\omega &= \mathbf{rot}(\mathbf{u}) \\
\mathrm{div}(\mathbf{u}) &= 0 \\
\frac{\partial \omega}{\partial t} + (\mathbf{u}\cdot\nabla\,)\,\omega &= (\omega\cdot\nabla)\mathbf{u} + \nu\Delta\omega \\
\frac{\partial \theta}{\partial t} + (\mathbf{u}\cdot\nabla\,)\,\theta &= \kappa\Delta\theta
\end{aligned}$$

peut se décomposer en :

$$\begin{aligned}
\frac{\partial \theta}{\partial t} + (\mathbf{u}\cdot\nabla\,)\,\theta &= 0 \\
\frac{\partial \theta}{\partial t} &= \kappa\Delta\theta
\end{aligned}$$

Opérateur continu

Exemple de partitionnement de problème pour l'exemple du jet plan :

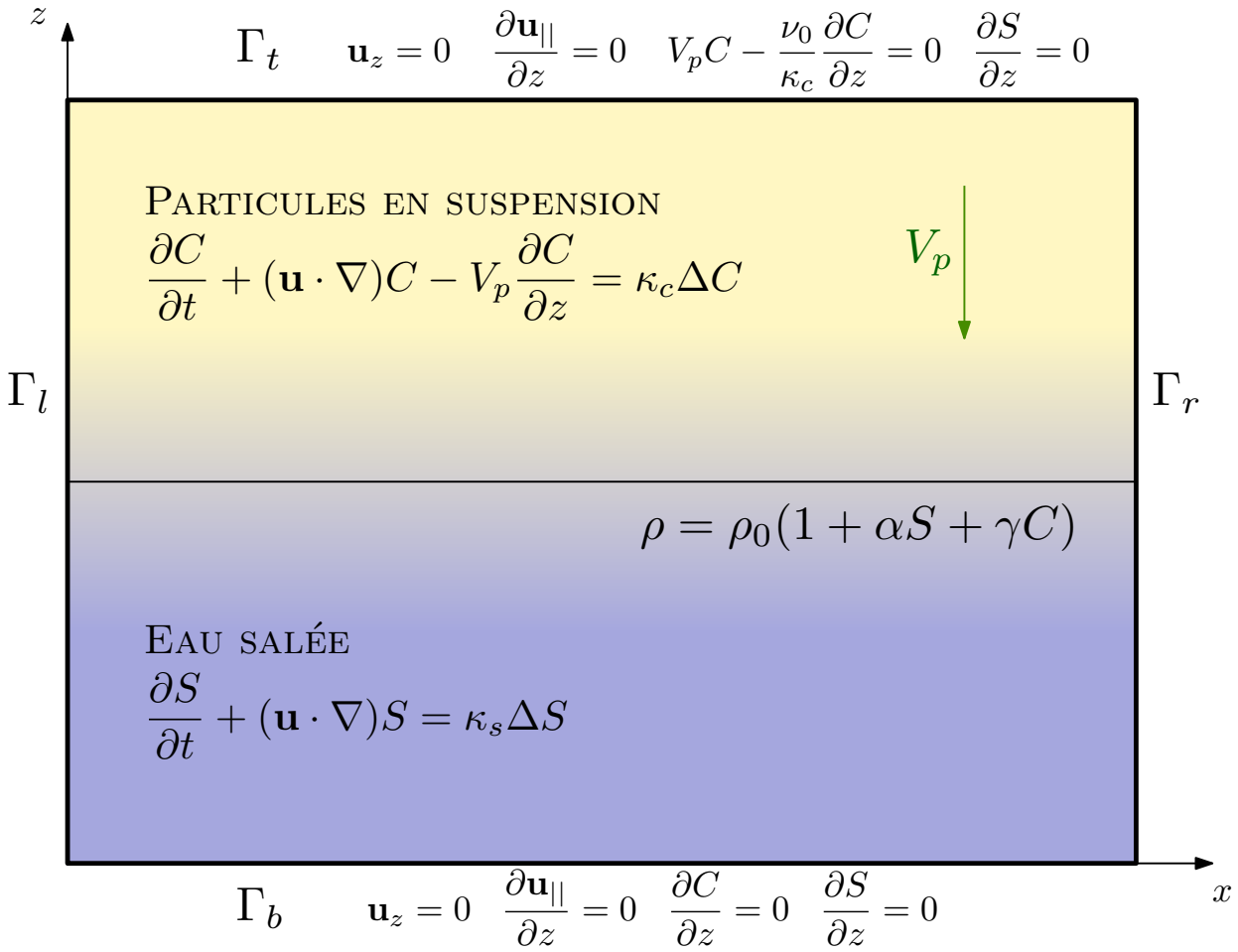
$$\begin{aligned} \omega &= \mathbf{rot}(\mathbf{u}) \\ \mathrm{div}(\mathbf{u}) &= 0 \\ \frac{\partial \omega}{\partial t} + (\mathbf{u}\cdot\nabla\,)\,\omega &= (\omega\cdot\nabla)\mathbf{u} + \nu\Delta\omega \\ \frac{\partial \theta}{\partial t} + (\mathbf{u}\cdot\nabla\,)\,\theta &= \kappa\Delta\theta \end{aligned}$$

peut se décomposer en :

$$\begin{aligned} \Delta\psi &= -\omega \\ \mathbf{u} &= \mathbf{rot}(\psi) \\ \mathrm{div}(\mathbf{u}) &= 0 \end{aligned}$$

Vue d'ensemble d'une simulation sous HySoP

Pour donner un exemple un peu plus concret de partition en sous-problèmes nous allons nous interesser au modèle suivant :



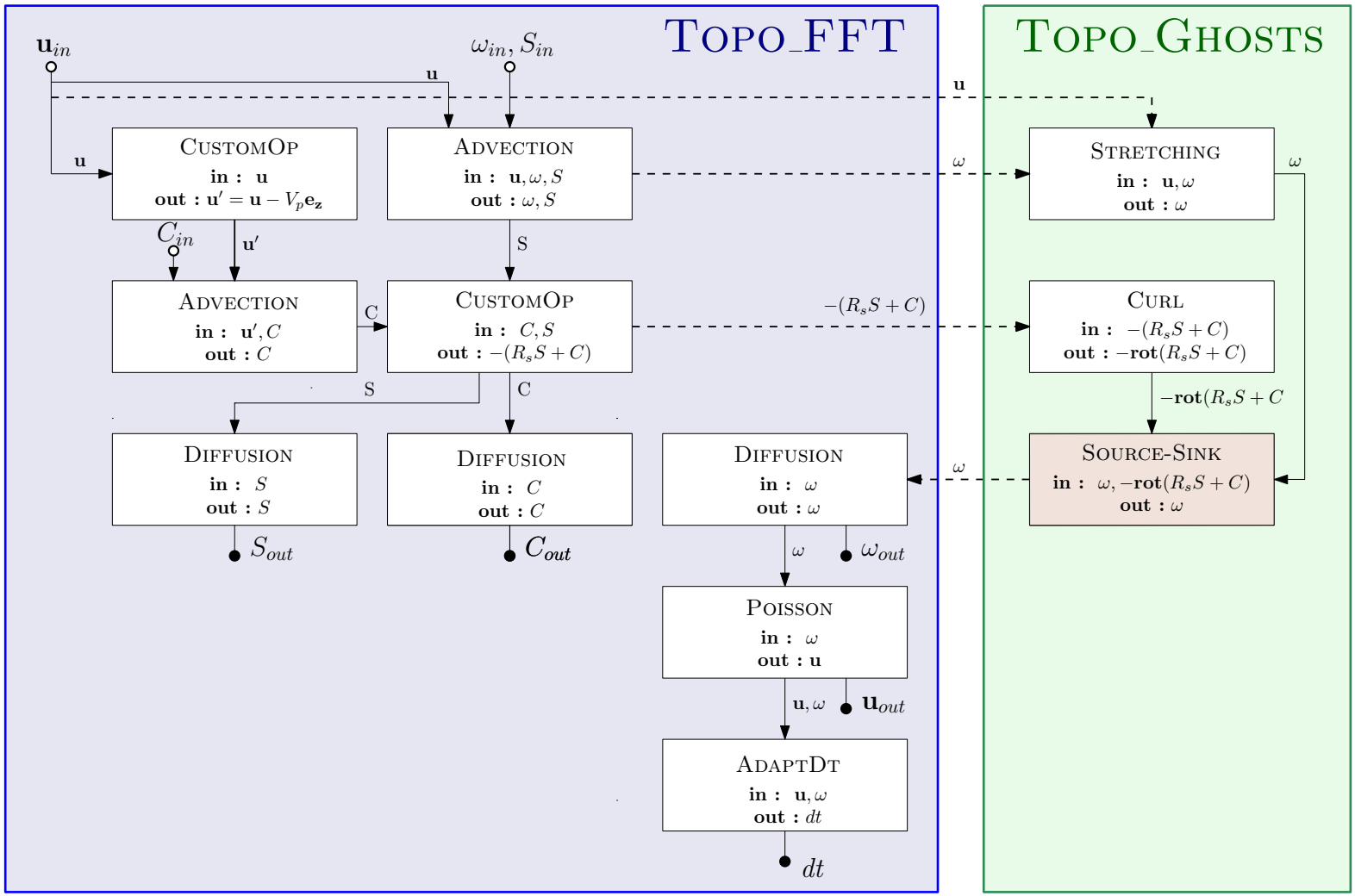
Vue d'ensemble d'une simulation sous HySoP

Formulation vitesse-vorticité du modèle :

$$\rho = 1 + \alpha S + \gamma C$$
$$\omega = \mathbf{rot}\,[\mathbf{u}]$$
$$\text{div}\,[\mathbf{u}] = 0$$
$$\frac{\partial \omega}{\partial t} + (\mathbf{u}\cdot\nabla)\,\omega - (\omega\cdot\nabla)\mathbf{u} = \Delta\,[\omega] - \mathbf{rot}\,[(R_s S + C)\,\mathbf{e_z}]$$
$$\frac{\partial S}{\partial t} + (\mathbf{u}\cdot\nabla)\,S = \frac{1}{S_c}\Delta\,[S]$$
$$\frac{\partial C}{\partial t} + (\mathbf{u}\cdot\nabla)\,C - V_p\frac{\partial C}{\partial z} = \frac{1}{\tau S_c}\Delta\,[C]$$

Vue d'ensemble d'une simulation sous HySoP

On peut envisager une résolution du problème en utilisant le graphe d'opérateurs suivants :



Opérateur discret

Un opérateur discret est une implémentation concrete d'un opérateur continu. Le choix et la configuration de l'opérateur discret est réalisé grâce aux paramètres suivants (que l'on applique sur l'opérateur continu) :

- Certains types d'oprérateur continus permettent de choisir l'implémentation cible (`Python`, `Fortran`, `OpenCL`, ...).
- Un paramètre 'method' permet de configurer les méthodes numériques (ordre en temps, ordre en espace, noyau de remaillage, ...). Une méthode peut aussi être spécifiée globalement à un sous-graphe d'opérateurs.
- Chaque opérateur offre une méthode par défaut qui sera prise en compte tant qu'elle n'est pas écrasée.

Par exemple un opérateur calculant le rotationnel d'un champ donné pourra avoir une implémentation par différences finies en `Python` et une implémentation par `FFT` en `Fortran` et une autre implémentation `FFT` en `OpenCL`.

Opérateur discret (ancienne méthode)

Exemple de définition d'un opérateur d'advection :

```
# Définit, discrétise et prepare un opérateur d'advection sur GPU.
# u et theta sont deux champs continus prédéfinis.
# Les variables sont discrétisées sur la topologie topo lors de l'appel
# de la fonction discretize.
advec = Advection(velocity=u,
                  advected_fields=[theta],
                  discretization=topo,
                  method={
                      Backend: OpenCL,
                      TimeIntegrator: RK4,
                      Remesh: L8_4,
                      Interpolation: Linear,
                      SpaceDiscretisation: FD_C_4,
                      Precision: HYSOP_REAL
                  },
                  )
advec.discretize()
advec.setup()
...
advec.apply()
```

Enchainer des opérateurs (ancienne méthode)

Pour enchainer les opérateurs, on va s'aider d'un objet `Simulation` :

```
# Etape de création et de discrétisation des operateurs
op1 = ...
op2 = ...
opn = ...

# Création d'une simulation
simu = Simulation( tinit =0.0, tend=10.0, time_step=0.1)
simu.initialize()

while simu.is_not_over():
    simu. print_state ()
    op1.apply(simu)
    op2.apply(simu)
     ...
    opn.apply(simu)
    simu.advance()
```

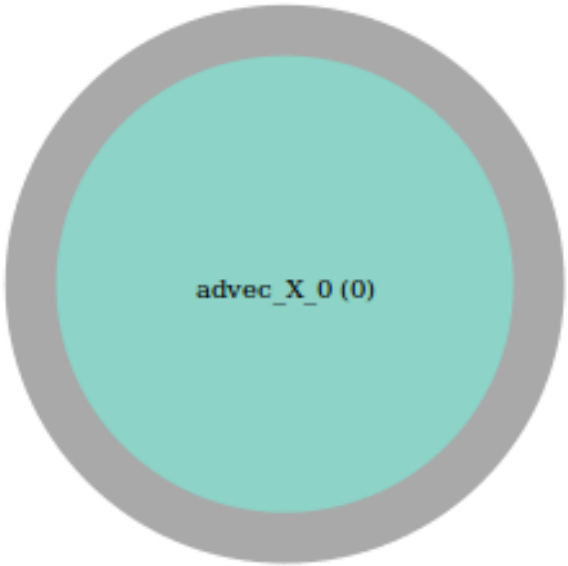
Introduction de graphes d'opérateurs dans la librairie

Motivations :

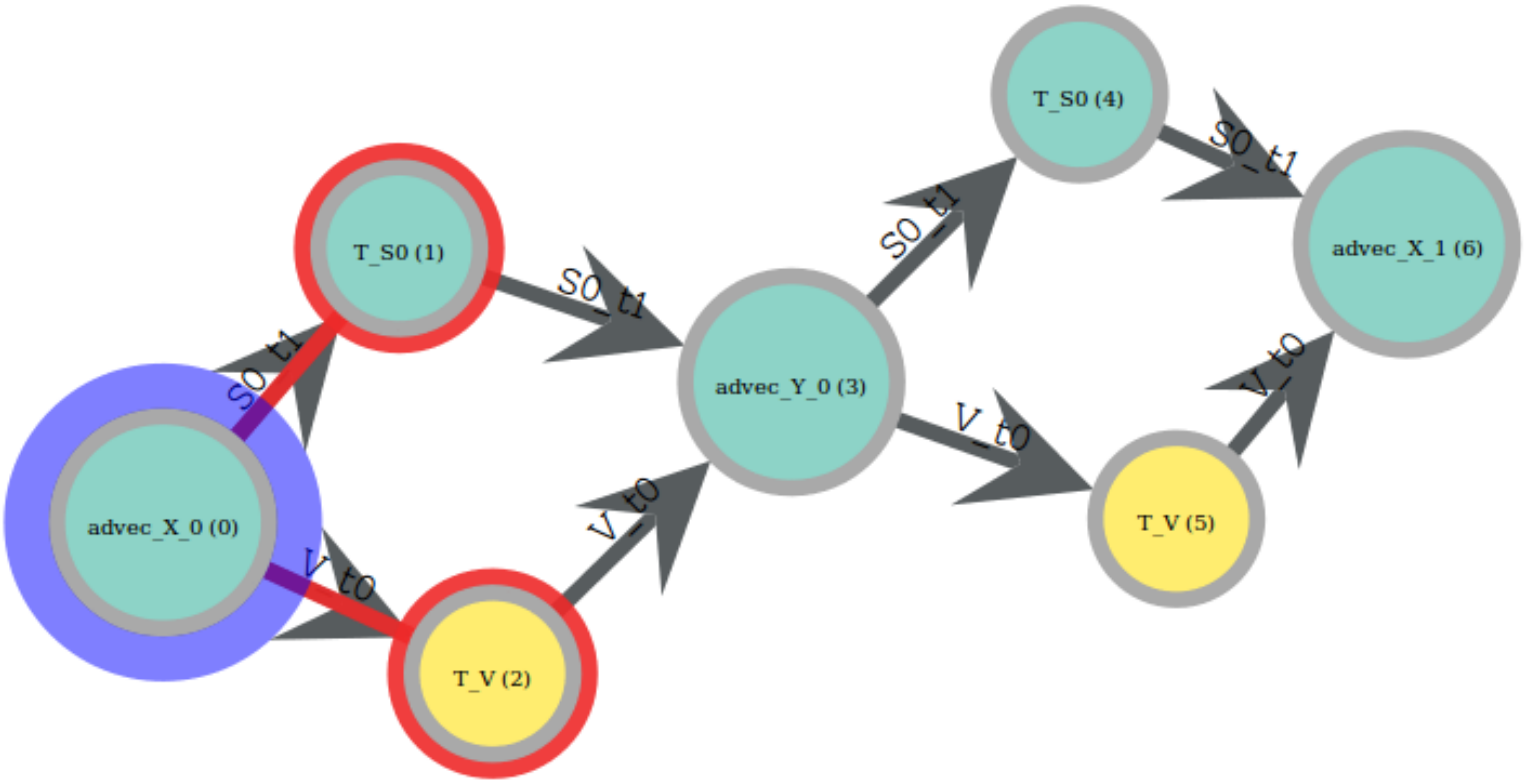
1. L'ancienne méthode ne permettait pas d'établir et d'analyser la dépendance temporelle entre opérateurs.
2. Tous les opérateurs devaient être créés et gérés explicitement par l'utilisateur :
   - Splitting directionnel
   - Transpositions
   - Redistributions inter topologies.
3. Beaucoup de redondance dans le code, de nombreux changements à effectuer lors de l'ajout ou du retrait d'un opérateur.

L'implémentation de problèmes sous forme de graphes résoud tous ses points et nous offre en même temps tout l'arsenal de la théorie des graphes.

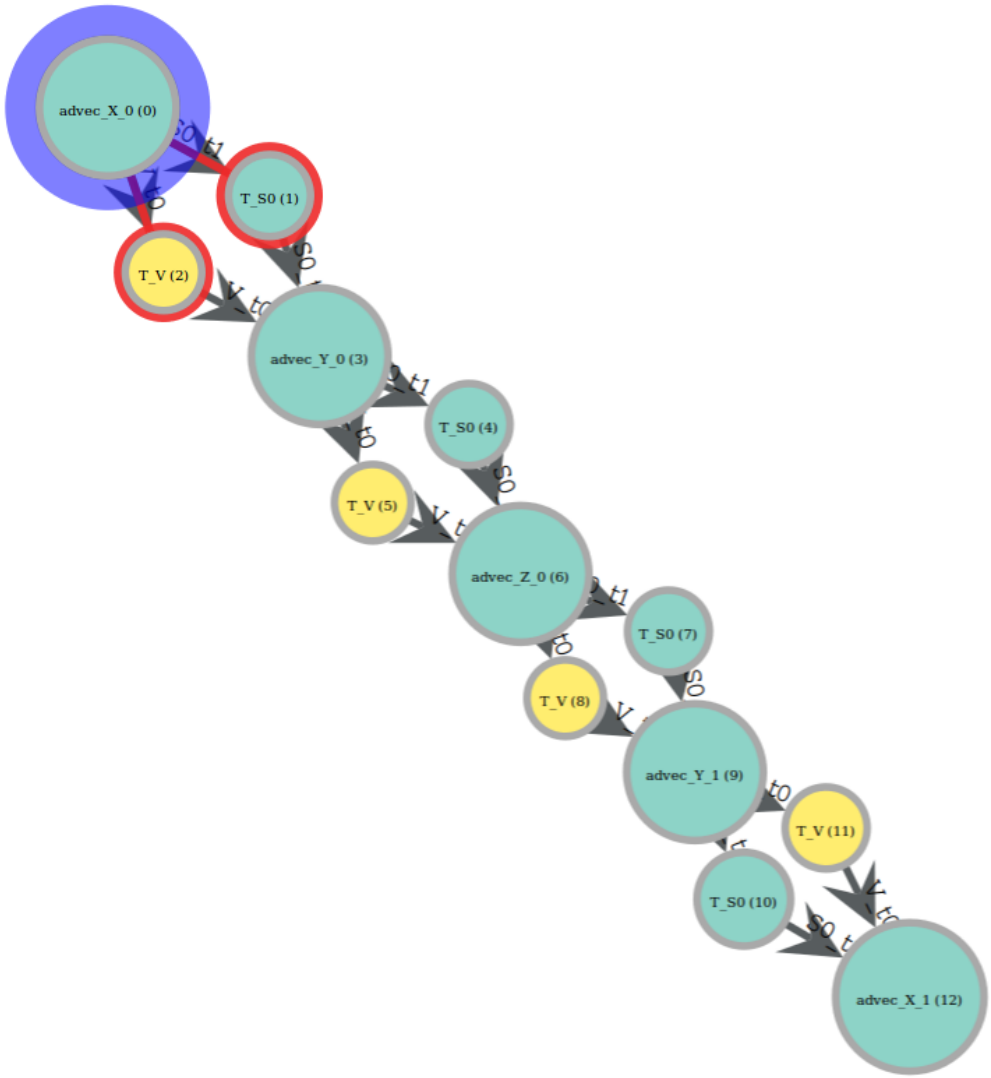
Introduction de graphes dans la librairie



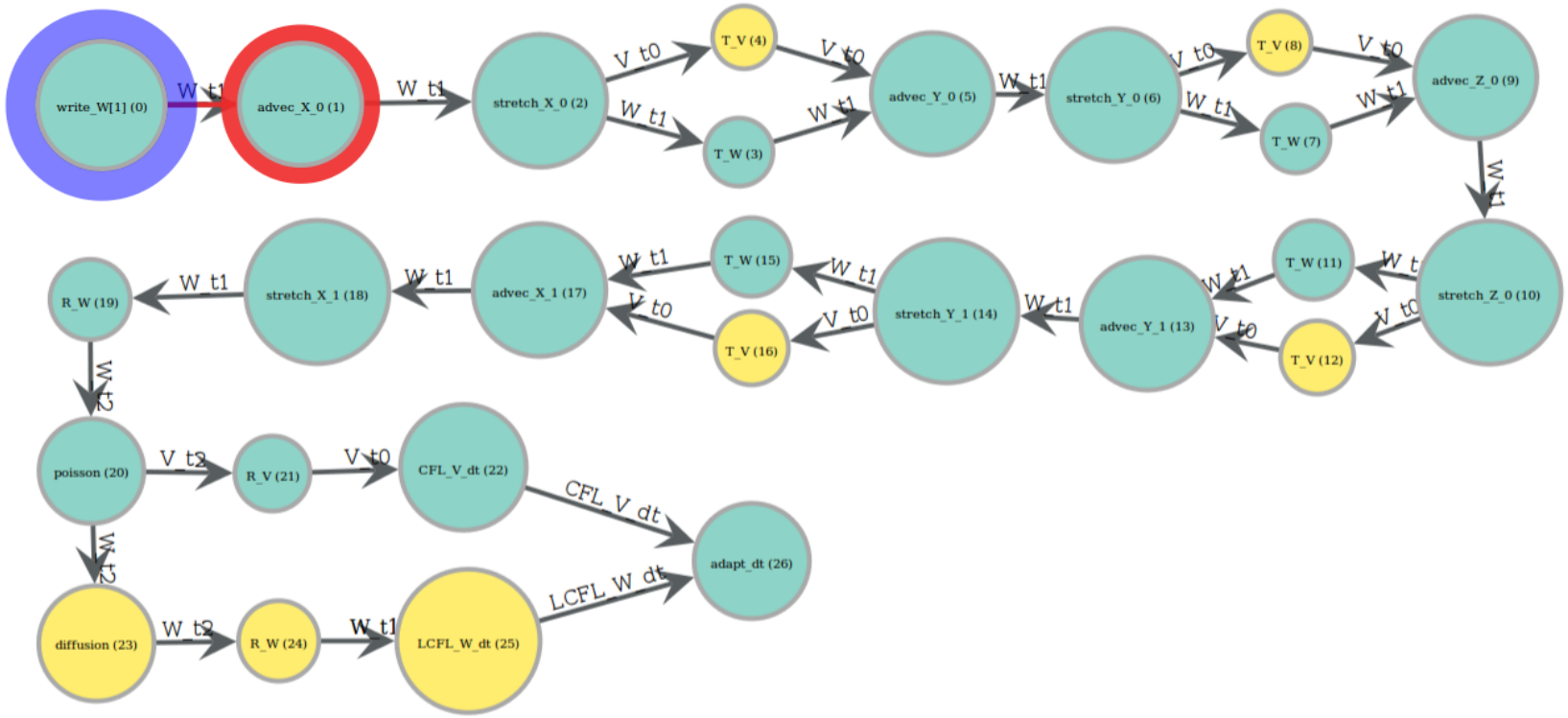
Graphes : cas de l'advection d'un scalaire en 2D



Graphes : cas de l'advection d'un scalaire en 3D



Graphes : Taylor-Green en 3D



Opérateur discret (nouvelle méthode)

Exemple de définition d'un opérateur d'advection avec la nouvelle méthode :

```
# Définit un opérateur d'advection sur GPU.
# u et theta sont deux champs continus prédéfinis .
# d3d est une discretization et non plus une topologie
# Les topologies sont crées et les variables sont discrétisées automatiquement
# durant le setup du graphe (voir slide suivante).
advec = Advection(velocity=u,
                  advected_fields=[theta],
                  variables={u:d3d, theta:d3d},
                  method={
                      TimeIntegrator: RK4,
                      Remesh: L8_4,
                      Interpolation : Linear,
                      SpaceDiscretisation : 4,
                  },
                  implementation=Implementation.OpenCL
                  )

# Cette partie n'est plus nécéssaire
# advec.discretize()
# advec.setup()
# advec.apply()
```

Enchainer des opérateurs (nouvelle méthode)

Pour enchainer les opérateurs, on va s'aider d'un objet `Problem` :

```
# Création des operateurs
op1 = ...
op2 = ...
opn = ...

# Création d'un graphe d'opérateurs
problem = Problem()
problem.insert(op1, op2, ..., opn)
problem.build()

# Résolution
simu = Simulation(tinit=0.0, tend=10.0, time_step=0.1)
problem.solve(simu)
```

On peut également insérer des graphes d'opérateurs dans d'autres graphes, ce que l'on va faire pour le splitting directionnel par exemple.

Implémenter son propre opérateur discret

Pour créer son propre opérateur discret, il faut créer un `DiscreteOperator`. Si la discrétisation est standard (étape `discretize`), et qu'il ne faut pas allouer de buffers temporaires (étape du `setup`), il suffit de redéfinir sa fonction `apply`.

Exemple : opérateur discret de pénalisation

$$\frac{\partial u(\mathbf{x},t)}{\partial t} = \chi_s(\mathbf{x})\lambda(\mathbf{u}_d - \mathbf{u}(\mathbf{x},t))$$

$$\chi_s(\mathbf{x}) = \begin{cases} 1 \ \textit{if} \ \mathbf{x} \in S \\ 0 \ \textit{if} \ \mathbf{x} \notin S \end{cases}$$

Implémenter son propre opérateur discret

Pour créer son propre opérateur discret, il faut créer un `DiscreteOperator`. Si la discrétisation est standard (étape `discretize`), et qu'il ne faut pas allouer de buffers temporaires (étape du `setup`), il suffit de redéfinir sa fonction `apply`.

Exemple : opérateur discret de pénalisation

En utilisant un schéma d'Euler implicite on obtient :

$$u_{k+1}(\mathbf{x}) = \frac{u_k(\mathbf{x})}{1+\chi_s(\mathbf{x})\lambda dt} + \frac{\chi_s(\mathbf{x})\lambda dt}{1+\chi_s(\mathbf{x})\lambda dt} u_D$$

Puis en prenant le rotationnel de l'equation précédente :

$$\omega_{k+1}(\mathbf{x}) = \omega_k(\mathbf{x}) + \nabla \times \left[\frac{-\chi_s(\mathbf{x})\lambda dt}{1+\chi_s(\mathbf{x})\lambda dt} (\mathbf{u}_k(\mathbf{x}) - \mathbf{u}_D) \right]$$