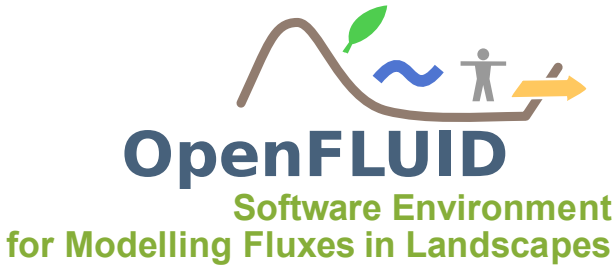

TP8 : Encapsulation d'un code existant

Objectifs:	Transformer un code existant, Fortran ou C, en une fonction de simulation OpenFLUID
Pré-requis:	TP1 à TP5

1 Introduction

Dans ce TP, deux codes existants (Fortran90 et C) sont considérés. Ils permettent de représenter l'évolution de l'humidité d'une colonne de sol en fonction de sa profondeur par analogie réservoir. Il s'agit de transformer un de ces codes de calculs (au choix Fortran90 ou C) en fonctions de simulation OpenFLUID et de les coupler à d'autres fonctions de simulations au sein d'un même modèle.

2 Présentation des codes existants

2.1 Principe du modèle réservoir

Le principe de ce modèle est de considérer une colonne de sol comme un empilement de réservoirs. Quand une quantité d'eau est apportée en surface du sol, la première couche de sol se remplit jusqu'à atteindre sa capacité au champ (`th_cc` dans le code), puis le surplus est drainé vers la seconde couche. Le processus se répète tant qu'il reste un volume d'eau à transmettre. Le volume restant en fond de colonne correspond au drainage sous la colonne (voir figure **??**).

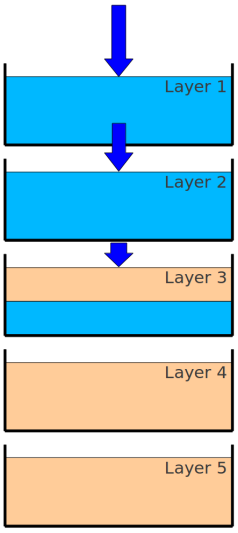

Figure 1 – illustration des principes du modèle réservoir. Ici, les 5 couches ont toutes la même capacité et l'apport d'eau correspond à la moitié de la capacité totale de la colonne : la première et la deuxième couche se remplissent jusqu'à atteindre leur capacité maximale et le reste de l'apport d'eau draine dans la troisième couche.

Le modèle réservoir est parmi les modèles les plus simples représentant l'infiltration d'une quantité d'eau dans le sol. Il est développé dans un contexte 1D, n'est pas spatialisé et ne dépend pas directement du temps (le temps intervient indirectement au travers de chroniques d'apport d'eau en surface). Les fonctionnalités d'OpenFLUID seront exploitées pour distribuer ce modèle sur différentes unités spatiales (SU) et les faire évoluer dans un contexte temporel plus rigoureux.

2.2 Compilation et lancement des codes sources

Les codes sources sont disponibles dans les jeu de données fourni :

- dans `original/src/modele_reservoir.f90` pour le code Fortran,
- dans `original/src/modele_reservoir.c` pour le code C.

Ils lisent en entrée les fichiers :

- `original/data/in` renseignant les caractéristiques du sol et
- `original/data/Pluvio_3_1997_06_05.txt` renseignant la chronique des apports d'eau dans le temps.

Ils produisent en sortie le fichier :

- `original/data/out` décrivant l'humidité du sol et le drainage cumulé en fin de simulation.

Dans un premier temps, compiler les codes grâce au `Makefile` à la racine du répertoire `original` en appelant `make compile_f` ou `make compile_c` selon le langage de programmation retenu. Les exécutables `modele_reservoir_f` ou `modele_reservoir_c` seront créés dans le répertoire `_build` et devront être lancés à partir de cet emplacement.

2.3 Analyse des codes existants

Afin de permettre une communication cohérente entre les différentes fonctions à coupler, OpenFLUID prend en charge la gestion des boucles de temps, la distribution des actions de simulations sur les unités spatiales, la lecture et l'écriture des données d'entrée et de sortie. Tous ces éléments doivent donc être éliminés du code à encapsuler et remplacer par des appels et des déclarations OpenFLUID (voir figure **??**).

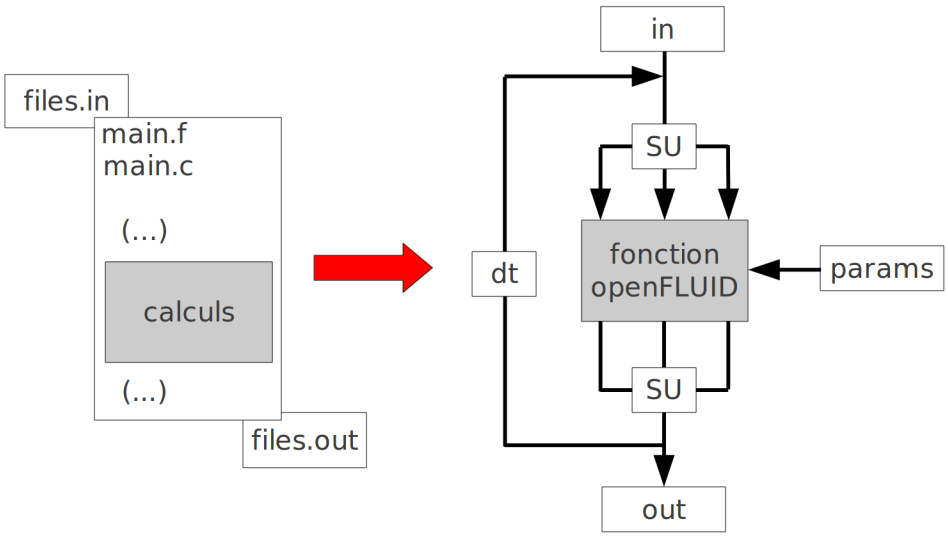

Figure 2 – le code original (à gauche) est analysé et décomposé de manière à isoler la partie "calculs". Seule cette partie "calculs" du code original est conservée dans la fonction de simulation OpenFLUID, le reste du code original (pré-, post-traitement, initialisation, boucle sur le temps et les unités spatiales,...) est confié au contexte OpenFLUID (à droite).

Cette analyse est l'étape primordiale de l'encapsulation, elle permet d'isoler la boucle de calcul et de définir les paramètres, les variables requises et produites de la fonction de simulation OpenFLUID.

3 Création de la fonction de simulation OpenFLUID

3.1 Principes généraux

Une fois l'analyse du code effectuée, créer une nouvelle fonction de simulation OpenFLUID, déclarer ses paramètres, variables requises et produites comme il a été vu dans les TP précédents.

Pour encapsuler un code écrit en C, il est possible d'insérer le code initial directement dans la méthode `runStep()` de la fonction de simulation OpenFLUID ou de déclarer une nouvelle méthode prenant comme arguments les variables nécessaires et produites par le calcul.

Pour encapsuler un code écrit en Fortran, il faut redéfinir une subroutine externe dans un fichier indépendant et l'appeler dans la méthode `runStep()` avec les arguments adéquates.

Note: l'encapsulation d'un code Fortran nécessite des déclarations et une syntaxe spécifiques qui sont décrites plus loin dans ce document.

3.2 Stratégie retenue pour l'encapsulation

Différentes stratégies peuvent être envisagées pour réaliser cette encapsulation. Une approche simple est privilégiée ici.

spatialisation du modèle : le modèle est distribué sur 7 unités surfaciques. Les fichiers décrivant leurs caractéristiques sont donnés dans les répertoires `OPENFLUID.IN.*`.

lecture des apports d'eau : la lecture des apports d'eau s'effectue grâce à une fonction de simulation dédiée d'OpenFLUID. Il s'agit de la fonction `water.atm-surf.rain-su.files` accessible sur OpenFLUID-Market. La documentation liée à cette fonction de simulation décrit son utilisation. Elle fait également appel à des fichiers d'entrée présents dans les répertoires `OPENFLUID.IN.*`.

manipulation des vecteurs : le modèle réservoir utilise des vecteurs pour décrire l'humidité et l'épaisseur des couches de sol. Dans le cas présent, ce nombre de couches de sol est de 5. Il est considéré comme fixe par la suite. Ainsi, la lecture de l'humidité initiale ou des épaisseurs des couches de sol n'est pas effectuée grâce à une variable vectorielle de taille 5, mais grâce à 5 variables scalaires. Les vecteurs au sein des méthodes de la fonction de simulation peuvent être simplement traités en déclarant un vecteur OpenFLUID spatialisé :

```
openfluid::core::IDVectorValueMap vector;
```

Note: OpenFLUID proposent des solutions moins restrictives et plus élégantes du point de vue algorithmique que celle qui a été retenue. Cependant, le but ici est de se focaliser sur la technique d'encapsulation afin d'être capable de la reproduire sur des exemples plus complexes par la suite et l'introduction d'autres nouveaux concepts a été évitée.

3.3 Particularités pour l'encapsulation d'un code Fortran

Pour encapsuler un code Fortran, des déclarations spécifiques sont utilisées :

- ajout d'une librairie dans la liste des headers :

```
#include <openfluid/tools/FortranCPP.hpp>
```

- déclaration d'une fonction externe, à insérer après la déclaration de la signature :

```
BEGIN_EXTERN_FORTRAN
        EXTERN_FSUBROUTINE(<nom de la subroutine>)
                (<listes des types et des arguments,ex:FINT *n,...>);
END_EXTERN_FORTRAN
```

- syntaxe spécifique pour appeler dans la méthode `runStep()` une subroutine Fortran (n'utilisant aucun module) :

```
CALL_FSUBROUTINE(<nom de la subroutine>)
        (<liste des arguments, ex:&n,...>);
```

- déclaration dans `CMake.in.config` du fichier source de la subroutine externe :

```
# list of Fortran files, if any
SET(FUNC_FORTRAN <nom du fichier source Fortran>)
```

- passage de vecteur en argument d'une subroutine.
  L'indexation des vecteurs est différente en Fortran et en C. Il faut donc passer en argument à la subroutine Fortran le premier élément du vecteur C déréférencé.

  ```
  CALL_FSUBROUTINE(<nom de la subroutine>)(<édbut du vecteur C, ex:&vect[0]>);
  ```

  De plus, la taille des vecteurs doit être déclarée de manière explicite dans l'entête de la subroutine Fortran.

  ```
  subroutine <nom de la subroutine> (<liste des arguments, ex:n,vect>)
  integer,intent(in)      :: n
  real(kind=8),intent(in) :: vect(n)
  ```