Réunion utilisateurs SURFEX
10 mars 2016

S. Faroux

Plan

- 1. Bilan du développement de la V8 et de l'implication plus forte des développeurs dans sa réalisation.
- 2. Suppression des variables globales dans la V8: explication des changements.
- 3. Transformations en cours pour la V8.1
- 4. Recensement des autres « nettoyages » pour la V8.1

1. Bilan du développement de la V8

- Rappel des changements mis en place:
  - Chaque développeur phase ses modifications sur la version en construction
  - Il teste à l'aide de la base de tests automatiques
  - Il fait une documentation:
    - Décrivant ses changements
    - Décrivant les changements dans les résultats, s'il y en a

- Ce qui a bien marché:
  - Les développeurs se sont bien prêtés au jeu, ils ont fait l'effort de phaser leur code et d'utiliser la base de tests inconnue d'eux jusqu'alors.
  - Les retours des développeurs ont été constructifs.
  - L'équipe SURFEX et certains utilisateurs sont satisfaits que les versions officielles changent un peu moins fréquemment, pour le suivi général et la stabilité.

1. Bilan du développement de la V8

- Points à améliorer pour une future version:
  - Conserver la nouvelle version en construction dans le trunk, en indiquant la dernière révision testée, afin de ne pas l'éparpiller dans de nombreuses branches (une par développeur pour la V8).
  - Dans le cas où les modifications changent les résultats des simulations, identifier précisément les changements et leur cause, et les documenter.
  - Simplifier la base de tests, donner la possibilité de l'utiliser « réduite », pour l'alléger.
  - Autres?

2.Suppression des variables globales dans la V8

- **modd_surfexn.F90**
  - Déclaration du type *SURFEX_t* qui contient tous les types du modèle (1 type = 1 module contenant des variables globales)
  - Premier niveau de regroupement des types en « super-types »:
    - Déclaration des types TEB_MODEL_t, ISBA_MODEL_t, SEAFLUX_MODEL_t, etc., qui contiennent les types associés à ces schémas
  - ➔ SURFEX_t contient les types …_MODEL_t

- **modd_off_surfexn.F90**:
  - Déclaration d'un tableau de types SURFEX_t (YSURF_LIST) correspondant aux sous-modèles Mésonh / threads OPEN-MP.
  - Déclaration du YSURF_CUR de type SURFEX_t qui correspond au sous-modèle / thread OPEN-MP courant.
  - La routine GOTO_MODEL fait pointer YSURF_CUR vers l'élément du tableau YSURF_LIST correspondant au sous-modèle / thread OPEN-MP courant.

  ➔ YSURF_CUR est passé en argument dans les drivers SURFEX et doit descendre jusqu'aux endroits où chaque module / type / variable globale du modèle est utilisé(e).

2.Suppression des variables globales dans la V8

- Implications dans les modules _n de SURFEX:
  - Exemple de modd_isban.F90:
    - Le type ISBA_t est déclaré comme avant
    - Les pointeurs THREADPRIVATE correspondant au modèle / thread OPEN-MP courant ne sont plus déclarés car on appelle désormais directement l'élément du type YSURF_CUR concerné, passé en argument depuis le driver.

    ➔ *Seul YSURF_CUR a besoin d'être THREADPRIVATE désormais.*

    - La routine ISBA_INIT s'applique aux éléments du type ISBA_t.
    - La routine ISBA_GOTO_MODEL qui fait pointer les pointeurs locaux du type vers le sous-modèle/thread OPEN-MP courant dans le tableau de types ISBA_t est supprimée.

    ➔ *La routine GOTO_MODEL de modd_off_surfexn.F90 remplace toutes les routines …_GOTO_MODEL*

- **surfex_alloc.F90**:
  - Cette routine appelle toutes les routines …_INIT pour initialiser les pointeurs des types des modules SURFEX.

2.Suppression des variables globales dans la V8

Modification dans l'arbre d'appel des routines de SURFEX.

Exemple d'init_sean.F90:

- Dans offline.F90:
  - ajout de l'argument YSURF_CUR (type déclaré dans le module MODD_OFF_SURFEX) à l'appel d'INIT_SURF_ATMn:

    CALL INIT_SURF_ATM_n(YSURF_CUR, …

- Dans init_surf_atmn.F90:
  - Ajout de l'argument YSC (nom indifférent à ce niveau) de type SURFEX_t à la déclaration de la routine
  - Utilisation de la définition du type présente dans modd_surfexn.F90 (nécessaire pour la déclaration qui suit):

    USE MODD_SURFEX_n, ONLY : SURFEX_t
  - Déclaration de l'argument YSC:

    TYPE(SURFEX_t), INTENT(INOUT) :: YSC
  - Appel d'INIT_SEA_n avec ajout en arguments des types utilisés par INIT_SEA_n et en-dessous:

    CALL INIT_SEA_n(YSC%DTCO, YSC%DGU, YSC%UG, YSC%U, USC%SM, YSC%DGL,…
  - ➔ Pour savoir à quoi se réfèrent les noms des sous-types d'YSC (YSC%…), consulter modd_surfexn.F90

2.Suppression des variables globales dans la V8

- Dans init_sean.F90:
  - Ajout des arguments : sous-types de YSC utilisés dans INIT_SEA_n et plus bas dans l'arbre d'appel:

    SUBROUTINE(INIT_SEA_n(DTCO, DGU, UG, U, SM, DGL, …

  ➔ Les arguments peuvent avoir n'importe quel nom
  - Utilisation des définitions des types présentes dans les modules associés.
    Ex: USE MODD_DATA_COVER_n, ONLY : DATA_COVER_t
  - Déclaration des arguments.

  Ex: TYPE(DATA_COVER_t), INTENT(INOUT) :: DTCO

- Et ainsi de suite dans l'enchaînement des appels de routines.

➔ Conséquences immédiates: lorsqu'on code, toutes les variables de modules doivent être passées en argument, on ne peut plus utiliser des commandes du style:

USE MODD_DATA_COVER_n, ONLY : XSEA

3. Transformations en cours pour la V8.1

- A. Nettoyage du passage des types en arguments des routines:
  - Observations:
    - parfois on passe en arguments un type provenant d'un module + des tableau issus de ce type ➔ *on passe 2 fois l'info, c'est superflu*
    - Parfois on passe en arguments tant de variables d'un type qu'il serait beaucoup plus léger de passer le type en entier et de faire référence aux tableaux du type dans la routine appelée
  - Travail réalisé:
    - Au cas par cas, on modifie les appels de routines dans tout le code pour limiter le nombre d'arguments total des routines.
    - La lisibilité du code devrait également être améliorée.
    - Cela induit l'introduction de nouveaux TYPE% dans les routines de calcul
  - A vérifier:
    - D'après Juan, le compilateur IFORT ne vectorise pas les boucles qui contiennent des variable du type TYPE%.
    - Des tests de performances vont donc devoir être réalisés prochainement pour vérifier que la modification du passage d'arguments ne pénalise pas l'efficacité du code.

3. Transformations en cours pour la V8.1

- B: Utilisation de types génériques pour les modules qui se ressemblent:
  - modules concernés :
    - Modules ISBA_n / GARDEN_n / GREENROOF_n
    - Modules DIAG_ISBA_n, DIAG_TEB_n, DIAG_SEAFLUX_n, etc
    - Modules _CANOPY_n et _SBL_n
    - Modules _GRID_n
  - Transformation réalisée (cf modd_surfexn.F90):
    - On a un type générique, par exemple GRID_t
    - Différentes instances de ce type sont déclarées pour chaque schéma qui le nécessite: IG (ISBA grid), TG (TEB grid), etc.

    ➔ Si dans certains schémas, certains tableaux de types ne sont pas utilisés, ils ne sont pas alloués et n'utilisent pas d'espace mémoire.
  - Intérêt:
    - Moins de modules à modifier, moins de fichiers presque identiques
    - On se rapproche de la programmation de type objet
    - Si un type générique est passé en argument, la routine qui l'appelle peut être elle aussi utilisé dans des contextes différents

3. Transformations en cours pour la V8.1

- C: Découpage du module MODD_ISBA_n en sous-modules, en s'inspirant de ce qui avait été fait pour GARDEN:
  - ISBA_OPTIONS_t, ISBA_PGD_t, ISBA_INIT_PGD_t, ISBA_INIT_t, ISBA_PARAM_t, ISBA_PROG_t
  - MODD_ISBA_n est très gros et ce découpage permet de ne passer en argument que la partie utile du module
  - La référence aux variables du type ISBA_t nécessite de savoir dans quel sous-type elles se trouvent

3. Transformations en cours pour la V8.1

- D: Dans les diagnostics ISBA, un tableau de dimension (NDIM, NPATCH) devient un tableau de dimension NDIM dans un type de dimension NPATCH:

Ex: ***DGI%XRN_PATCH(JI,JPATCH) => DGIP%AL(JPATCH)%XRN***

➔ Il n'est alors plus besoin de déclarer les tableaux moyens sur les patchs et les tableaux par patchs dans le type DIAG_t:

```
TYPE DIAG_PATCH_t
    TYPE(DIAG_t), DIMENSION(: ) : : AL
END TYPE DIAG_PATCH_t
…
TYPE DIAG_ISBA_t
    TYPE(DIAG_t) :: DGI
    TYPE(DIAG_PATCH_t) :: DGIP
END TYPE DIAG_ISBA_t
…
ALLOCATE(DGI%XRN(NDIM))
ALLOCATE(DGIP(NPATCH))
DO JPATCH=1,NPATC
  ALLOCATE DGIP%AL(JPATCH)%XRN(NDIM)
ENDDO
```

3. Transformations en cours pour la V8.1

- Inconvénients de la transformation D:
  - A la lecture et à l'écriture
  - Possibilité à l'avenir d'écrire / lire un champ par patch plutôt qu'un champ unique pour tous les patchs?
    - Permettrait de limiter la taille unitaire des champs lus / écrits (demandé par Méso-NH également)➔ *à discuter*
    - Rendrait difficile la reproductibilité ascendante en cas de lecture des diags au restart
    - Pourrait nécessiter d'adapter des post-traitements des fichiers de sorties existants

- Extension à étudier de la transformation D:
  - Si on trouve un moyen d'étendre cette transformation des DIAG d'isba vers les variables du module MODD_ISBA_n elles-mêmes, on pourrait dimensionner chaque champ à la taille effective des patchs et se débarrasser du PACK / UNPACK.
  - Mais pour le moment c'est impossible car hors de la boucle sur les patchs de coupling_isban, le reste des calculs ont été pensé pour des tableaux comportant la dimension « PATCH ».

4. Recensememement des autres « nettoyages » requis pour la V8.1

- Y a-t-il des options qui peuvent être supprimées de SURFEX (plus utilisées du tout)?
  - AGS, NIT, etc…
  - Autre?
- Autres nettoyages qui paraîtraient utiles aux utilisateurs?
- Quelle documentation pour les options de namelist?
- La V8.1 contiendra aussi:
  - La parallélisation et l'optimisation des phases PGD / PREP
  - La suppression des zones parallèles OPEN-MP du driver OFFLINE
  - Le serveur d'I/O XIOS (à confirmer par Stéphane Sénési)