A thread-safe and flexible DDH dataflow

Fabrice Voitus

CNRM/GMAP/ALGO, Météo-France

11 *September* 2019

Main objective :

- Provide the budget of the model prognostic variables on any defined domain, in order to better understand the model's dynamical and physical interactions, and thus contribute to the progress of the parametrization developments.

Main features :

- **Budgets** : mass, energy [kinetic, potential, enthalpy], momentum, entropy, water vapor, and [model surface variables], ...
- **Domains** : whole model domain, zonal bands, limited domains, and even isolated grid points.
- **Outcomes**: LFA file production at a defined time frequency, containing vertical profiles of horizontally averaged [fluxes, tendencies, variables] for each chosen domains.

DDH : Diagnostics on Horizontal Domains

General budget equation form in a mass-based η-coordinate :

For a given budget variable X,

$$\frac{1}{g}\frac{\partial}{\partial t}\left(\frac{\partial \pi}{\partial \eta}X\right) = \frac{1}{g}\frac{\partial \pi}{\partial \eta}\sum_{\mathcal{D}}\mathcal{T}_{\mathcal{D}} - \sum_{\varphi}\frac{\partial \mathcal{F}_{\varphi}}{\partial \eta}$$

- $\mathcal{T}_{\mathcal{D}}$: dynamical tendencies at model levels, e.g horizontal and vertical advection processes, ...
- $\mathcal{F}_{\varphi}$: physical fluxes at the model vertical interfaces, e.g [radiation, convection, turbulence, precipitation, ...] parametrizations.

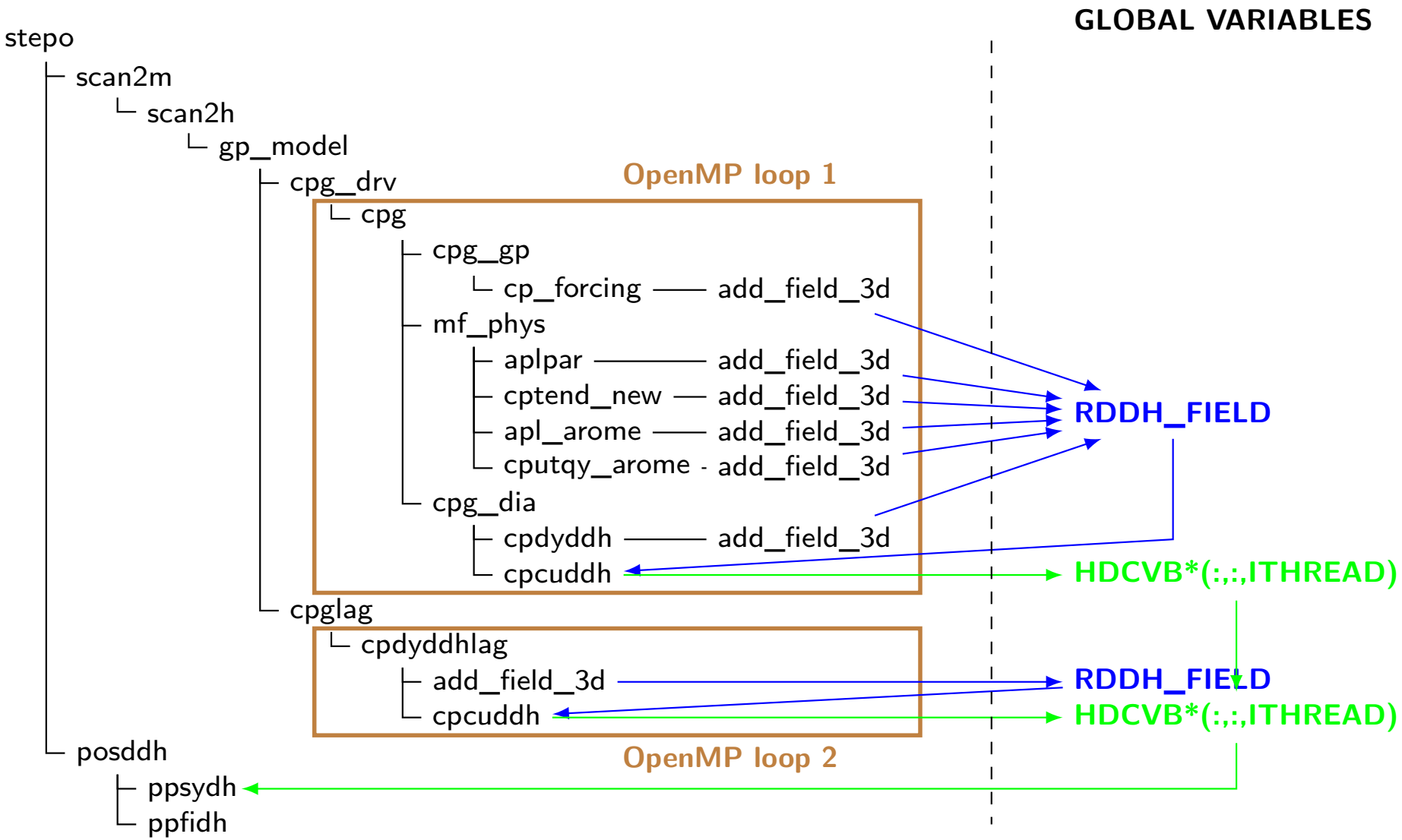
How to add new tendencies and fluxes in budgets ?

Actually, in a numerical Model, not all tendencies and fluxes are taken into account $\Rightarrow$ budgets are estimates upto a residual.

$$\frac{1}{g}\frac{\partial}{\partial t}\left(\frac{\partial \pi}{\partial \eta}X\right) = \frac{1}{g}\frac{\partial \pi}{\partial \eta}\sum_{\mathcal{D}'}\mathcal{T}_{\mathcal{D}'} - \sum_{\varphi'}\frac{\partial \mathcal{F}_{\varphi'}}{\partial \eta} + \mathcal{E}$$

- Be able to add any new tendencies, fluxes in a very **flexible** and tractable way, with **minor** modification of the code, in order to reduce residuals $\mathcal{E}$.

Current flexible DDH dataflow



Current flexible DDH dataflow

Dataflow

1. several calls to **add_field_3d** fill the global array RDDH_FIELD ⇒ **Manual allocations**;
2. **cpcuddh** does the horizontal averaging on RDDH_FIELD, and updates the global arrays HDCVB* in a thread-safe manner;
3. **ppsydh** does the gathering over the different MPI tasks of the arrays HDCVB*;
4. **ppfidh** writes the result to file.

Issues and Challenges

- **Thread-safety**: since different threads execute different NPROMA blocks simultaneously, they can write to the global RDDH_FIELD simultaneously ⇒ **not thread-safe**.
- **Skipping fields**: since there are two OpenMP loops [**cpg** and **cpglag**], some mechanism is necessary to tell the second call to **cpcuddh** to skip the fields coming from the first loop. Currently, this is done by adding zeroes to the existing fields ⇒ **not efficient at all**.
- **Memory usage**: all diagnostic fields (and there can be a lot of them!) are stored in memory in 3D, before the horizontal average is taken in **cpcuddh** ⇒ **unnecessary memory usage**

Thread-safety

The road to programming hell is paved with global variables, [Degrauwe D. (2016)]

Data should be transferred by arguments, instead of by global variables ⇒ two extra **structures** have been defined to achieve this requirement, [i.e, to replace RDDH_FIELD] ⇒ Take advantage of the fact that **automatic allocations** are faster than manual allocations

- A type TYP_FIELD for a single diagnosed quantity, containing the numerical data, along with some metadata [name, origin, purpose, ...].

```
TYPE TYP_FIELD
  REAL, POINTER :: RVAL(:,:)
  CHARACTER(LEN=128) :: CNAME
  !...
END TYPE TYP_FIELD
```

- A type TYP_DDH, acting as a container of TYP_FIELD's.

```
TYPE TYP_DDH
  TYPE(TYP_FIELD), POINTER :: YFIELD(:,:)
  INTEGER :: NFIELD
  !...
END TYPE TYP_DDH
```

Thread-safety

1. TYP_DDH is augmented with two components NFIELD_AUTO and RVAL, where NFIELD_AUTO is some estimate of the number of fields; RVAL is a pointer to a 3D array;
2. At the highest level of a OpenMP block-loop (i.e. **cpg** or **cpglag**), an auxiliary array ZAUX is allocated automatically with suitable dimensions : NPROMA×NLEV×TYP_DDH%NFIELD_AUTO;
3. when the new structure YLDDH (TYP_DDH) is initialized, its component RVAL points to ZAUX:

   ```
   YLDDH%RVAL=>ZAUX
   ```

4. YLFIELD (TYP_FIELD) is added to the container YLDDH (TYP_DDH) with **new_add_field**; the YLFIELD's numerical data points to the container's RVAL which points itself to the automatically allocated array ZAUX :

   ```
   YLFIELD%RVAL=>YDDDH%RVAL(:,:,JFLD)
   ```

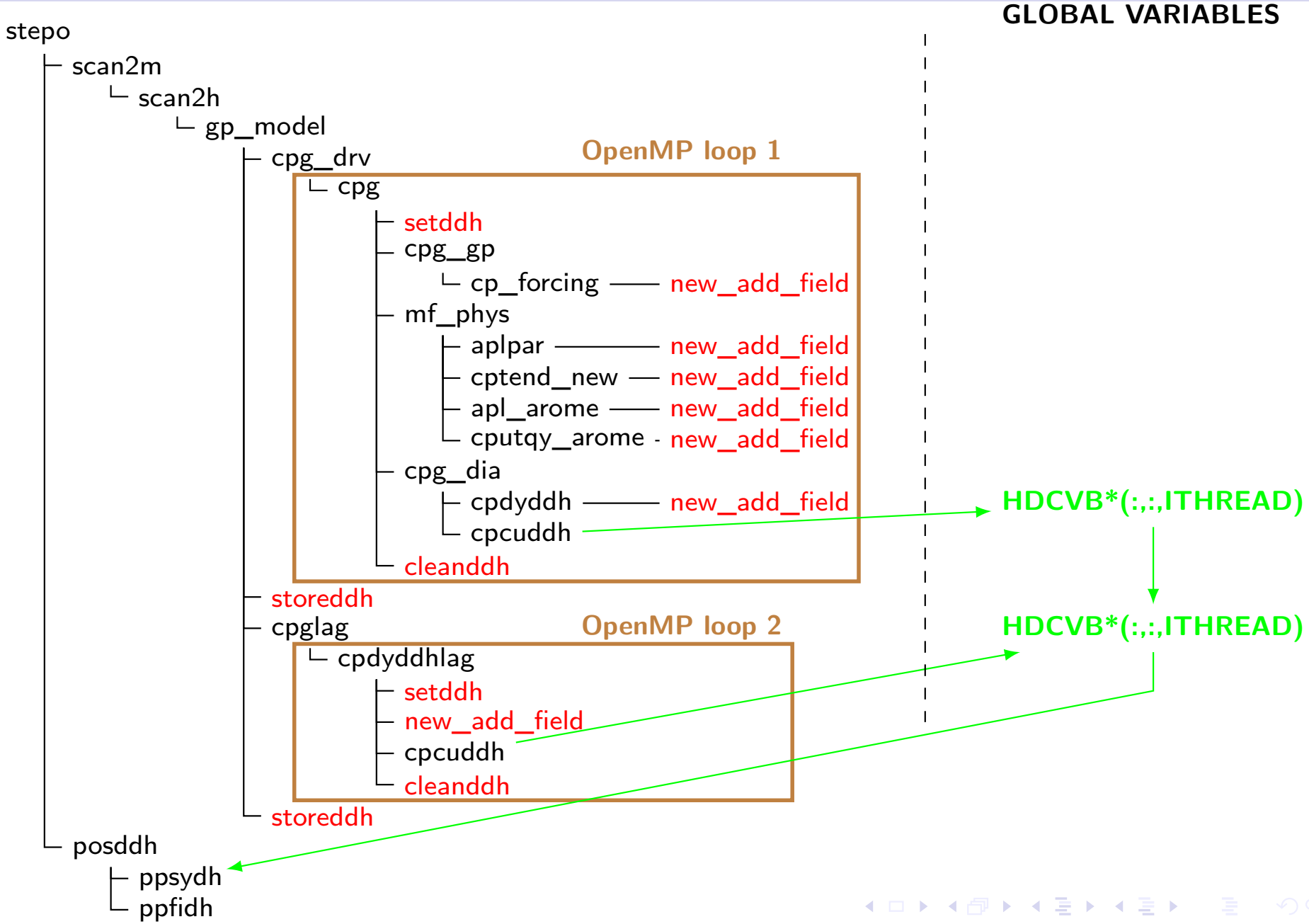
   If the number of fields exceeds the initial estimate (third dimension of ZAUX), then the array is allocated manually:

   ```
   ALLOCATE(YLFIELD%RVAL(KPROMA,KLEV))
   ```

5. At the end of the NPROMA block, the estimate of number of fields is updated, so that manual allocation is avoided for the following iterations of the loop.

```
CPG_DRV
  | TYPE(TYP_DDH) :: YLDDH
  | !$OMP PARALLEL DO FIRSTPRIVATE(YLDDH) LASTPRIVATE(YLDDH)
  └ CALL CPG(YLDDH)
      | REAL, TARGET :: ZAUX(KPROMA,KLEV,YLDDH%NFIELD_AUTO)
      | YLDDH%RVAL=>ZAUX
      ├ CALL MF_PHYS(YLDDH)
      |   └ CALL CPTEND_NEW(YLDDH)
      |       └ CALL NEW_ADD_FIELD(PVAL,CNAME,YLDDH)
      |             TYPE(TYP_FIELD), POINTER :: YLFIELD
      |             ! increment YLDDH%NFIELD (+1) ...
      |             ! extend YLDDH%YFIELD ...
      |             YLFIELD=>YLDDH%YFIELD(NFIELD)
      |             IF (NFIELD<=NFIELDAUTO) THEN
      |                 YLFIELD%RVAL=>YLDDH%RVAL(:,:,NFIELD)
      |             ELSE
      |                 ALLOCATE(YLFIELD%RVAL(KPROMA,KLEV)
      |             ENDIF
      |             ! fill YLFIELD%RVAL and YLFIELD%CNAME ...
      | YLDDH%NFIELDAUTO=MAX(YLDDH%NFIELD,YLDDH%NFIELD_AUTO)
      └ CALL CPG_DIA(YLDDH)
          └ CALL CPCUDDH(YLDDH)
                DO JFLD=1,YLDDH%NFIELD
                  DO JROF=1,KPROMA
                    HDCVB(JFLD,:,ITHREAD)=HDCVB(JFLD,:,ITHREAD) &
                      & +YLDDH%YFIELD(JFLD)%RVAL(JROF,:)*WEIGHT(JROF)
                  ENDDO
                ENDDO
  !$OMP END PARALLEL DO
```

New thread-safe flexible DDH dataflow



New DDH subroutines

- **setddh** : Called at the beginning of an NPROMA block for initializing a TYP_DDH structure with correct dimensions etc.
- **cleanddh** : Called at the end of an NPROMA block to stores some metadata (field names, types, ...) in global variables, and performs deallocations for one NPROMA block.
- **storeddh** : Called after the OpenMP loops in order to store some data in global variables.
- **new_add_field** : Called throughout the gridpoint calculations in replacement of the current add_field.

Skipping fields mechanism :

Actually, the number of fields at the end of the first block-loop is the only information that needs to be exchanged with the second block-loop ⇒

- An NFIELD_OFFSET integer component is added to the TYP_DDH structure. This value is used in **cpcuddh** to write the fields of the current block-loop to the correct position in HDCVB* global arrays.
- Before the first Block-loop, NFIELD_OFFSET is set to zero; Then before the second block-loop, it is set to the total number of stored fields from the first block-loop in **storeddh**.

⇒ **Let's move to some practical work now !!!**

```
NUMBER OF STORED DDH FIELDS   :           10

flx_phy1  *
flx_phy2  !
flx_phy3  !
flx_phy4  !
flx_tke   !   First NPROMA block-loop
flx_turb  !
flx_dyn1  !
flx_dyn2  !
flx_dyn3  !
flx_dyn4  *
```

```
NUMBER OF STORED DDH FIELDS   :           14

flx_phy1  *
flx_phy2  !
flx_phy3  !
flx_phy4  !
flx_tke   !   First NPROMA block-loop
flx_turb  !
flx_dyn1  !
flx_dyn2  !
flx_dyn3  !
flx_dyn4  *
flx_lag1  *
flx_lag2  !  Second NPROMA block-loop
flx_lag3  !
flx_lag4  *
```

Save memory usage

Perform averaging, before storing :

To **reduce memory usage**, the horizontal average is already taken inside these new thread-safe **new_add_field** subroutines ⇒ averaging weight and domain indexes need to be also included in TYP_DDH structure ⇒ Initialisation is set in **setddh** subroutine.

```
SUBROUTINE NEW_ADD_FIELD(PVAL,CDNAME,YDDDH)
....
! local pointer
TYPE(TYP_FIELD), POINTER       :: YLFIELD
....
! point to the newly added field
YLFIELD=>YDDDH%YFIELD(YDDDH%NFIELD)
....
! set numerical data (average)
      YLFIELD%RVAL(:,:)=0.
      DO JROF=YDDDH%KST,YDDDH%KEND
         YLFIELD%RVAL(1:SIZE(PVAL,2),YDDDH%NDDHI(JROF)) = &
         & YLFIELD%RVAL(1:SIZE(PVAL,2),YDDDH%NDDHI(JROF)) &
         & +PVAL(JROF,:)*YDDDH%WEIGHT(JROF)
      ENDDO
....
END  SUBROUTINE NEW_ADD_FIELD
```

Summary

TYP_DDH is a new super-structure which serves multifold purposes:

1. Safely transport information from **new_add_field** to **cpcuddh**. This is the reason why YLDDH is made threadprivate. The components that are required for this data transport are NFIELD, YFIELDS, RVAL.
2. Pass information from one loop iteration to the next. For example the number of fields NFIELD_AUTO that should be allocated dynamically in ZAUX is determined in the first iteration, then used in the following iterations.
3. Pass information from the first loop **cpg** to the next loop **cpglag**. This is the case for the NFIELD_OFFSET component.
4. Save memory usage by performing weighted horizontal averaging directly inside the calls to **new_add_field**.