Proposal for a new diagnostic dataflow in Arome physics

Olivier Riviere, Jean-Marcel Piriou and contributors from GMAP

21 avril 2008

1 Summary

A new coding approach is proposed for extracting diagnostics from the Arome/-MesoNH physical parametrisations. It could be used in other parts of the IFS/ARPEGE software. Physical quantities are recorded into a flexible data structure in the parametrisations, and readable by higher level routines. The data structure (a linked list of ad hoc Fortran 90 types) is automatically allocated and indexed as needed by low-level routines, so that physicists can freely choose which quantities they want to record, and how they want to process them. This technical approach will greatly simplify software clarity and maintenance. No performance penalty is expected.

The main foreseen applications are (1) to replace the existing 'DDH' diagnostic mechanism in Arome by something simpler and more powerful, and (2) to provide the ALADIN consortium with easy access to various Arome/MesoNH physical quantities at the level of the physics calling interface.

The main expected impact on the code is that (1) specific subroutine calls will be inserted into the Arome/MesoNH physics, in order to record the needed quantitites, and (2) new software will be developed to agglomerate these quantities into budget computations, e.g. to provide backward compatibility with the DDH diagnostics. The recorded quantities could be used for more general purposes like extracting 3D physical fluxes if needed.

This technical document calls for an agreement about the approach, and for a specification of the attributes that are to be stored with the extracted quantities, since it may determine what can be done with it later.

2 Introduction : motivation

The current DDH facility in Arome consists in an interface between the Meso-NH 'BUDGET' and the Arpege/Aladin 'DDH' existing facilities. In this form, the data flow is quite complicated (see figure 1) and debugging : such a structure seems hard to maintain during future evolutions of the code. Therefore a rewriting of the data flow in the DDH is suggested in order to keep the code easier to maintain and may also more efficient in terms of computational costs.

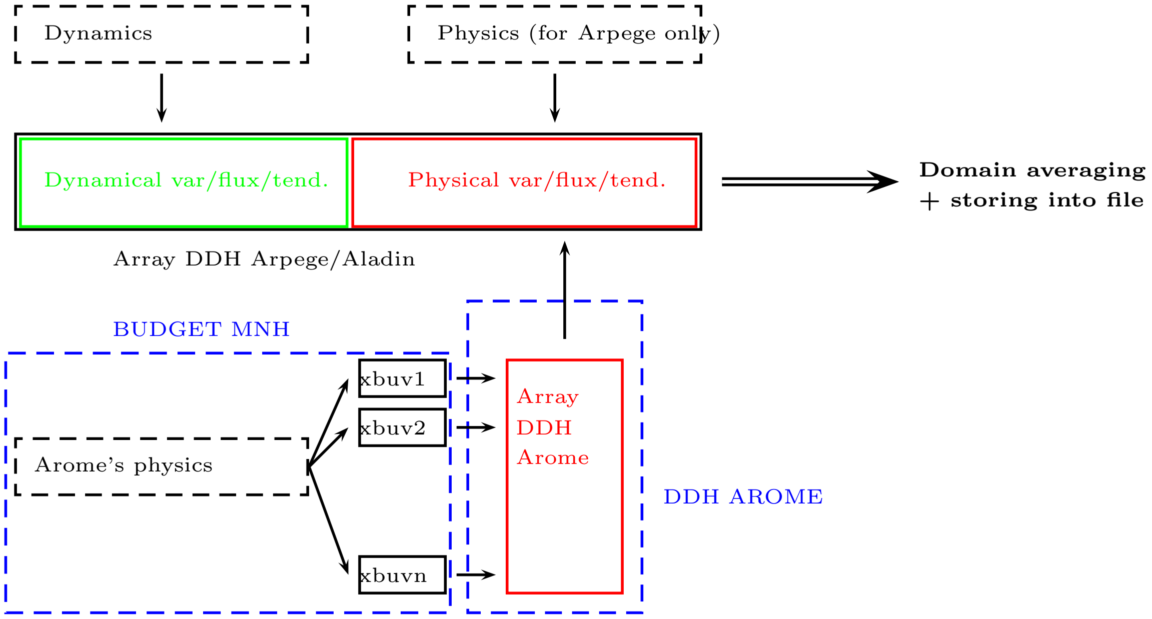

FIG. 1 – Current dataflow for DDH in Arome (each full-line rectangle represents an array where fields are stored)

The rewriting proposal that follows is based on the following requirements :

- These modifications in the data flow will remain transparent for users : DDH products should remain unchanged.
- The large amount of fluxes in the Arome/MesoNH microphysics compared to Arpege/Aladin makes obsolete the current coding approach, where arrays are pre-initialized in setups : counting before entering the physics how much fluxes/tendencies will be stored in the DDH arrays is still possible (for the moment) in Arpege but it gets difficult in Arome where the DDH output files contain around 200 items ! ! Thus it is desirable to get rid of such setups (which are duplicated in sunddh.F90 and aro_iniapft.F90).

– The new structure of the data flow will be compatible with the different physics available now and in the future (MESO-NH, ARPEGE, ALARO, HIRLAM...). Thus, it will facilitate the 'convergence' between the physical diagnostics.
– This structure should be able to handle different data types (fluxes, tendencies..) according to the different needs.
– This new structure could also be used to extract terms from the physics not only for DDH computations but also for other kinds of diagnostic computations. The current proposal focuses on DDHs, but one could apply its software for other purposes.

3 New flexible, self allocatable data structures

A basic prototype has been coded and found to work on the 'tori' NEC SX8R supercomputer of Météo-France. It was written for testing purposes. For simplicity the names of variables in the text below refers to this prototype.

3.1 Description

The solution was thought to be self allocatable structures similar to GFL but more flexible. This subsection describes how they are defined, the possible architecture of the code being discussed in section 4. Each extracted quantity (variable, flux, tendencies...) will be characterized through a Fortran 90 structure type (named here DDH) which defines several attributes corresponding to this quantity.

An example is given here :

```
TYPE DDH
      CHARACTER(LEN=13)::NAMEF   ! field's name
      REAL,DIMENSION(:,:),POINTER:: PFIELD  ! pointer corresponding
    !to the stored field
      INTEGER:: FIELDIND         ! position of the field in the DDH array
      CHARACTER(LEN=2)::VAR      ! variable corresponding to the stored field
      CHRACTER(LEN=1):: DDHDIAG !   = 'H' if field will be used for DDH
        !  or 'G' if only used for other diagnostics
! and a lot of open possibilities ...
END TYPE DDH
```

In this minimalist version, we use the following attributes : name of the field, position of the field in the DDH array (introduced below), the name of the variable on which the budget is applied (temperature, ...) and a string explaining which use will be made of the stored field (DDH or other diagnostics). One could also

include the name of the subroutine where the field is extracted, flags explaining the type of physics or the nature of the field (e.g. flux vs tendency)...
All these attributes can also be implicitly defined in the name of the field (for instance, a name starting with an F for a flux, with a T for a tendency...). They are important because they document the structure content itself (important for debugging purposes) and they determine which operations the extracted field will undergo at the place where it is recorded, before being stored (for instance conversion from potential temperature to temperature...)

These attributes will appear in the source code, everywhere a field is being extracted. Thus, in order to minimise future modifications to the physics code, *it is important to decide before writing the new code, which attributes one wants to store in the definition of these new structures.* Obviously, it depends on how we plan to use these fields. The various extracted fields are gathered into an allocatable array of structure of type DDH, called here DDH_DESCR and whose last dimension corresponds to the total number of extracted fields :

```
TYPE(DDH),ALLOCATABLE,DIMENSION(:):: DDH_DESCR
```

The attribute *allocatable* being forbidden inside a type structure, the field is not directly stored inside DDH_DESCR but defined through a pointer to a large array called DDH_FIELD :

```
REAL,DIMENSION(:,:,:),ALLOCATABLE,TARGET::DDH_FIELD ! target of PFIELD
! first two dims are the same as PFIELD, the third being the number of stored fiel
```

3.2 Extracting a field from the physics

When a user needs to add a field into the diagnostics, he/she only needs to call subroutine ADD_FIELD. The first argument of ADD_FIELD will be the field to store and the others will correspond to the associated attributes (for instance "call ADD_FIELD(field_to_store,'name_of_field','F','CT'....)")

ADD_FIELD performs the following tasks :

- when in the code a specific field is supplied as argument for the first time in the execution, the last dimension of arrays DDH_FIELD and DDH_DESCR is incremented in order to add space for the new field to store. The code determines if a field is encountered for the first time by testing the field's name. This reallocation of arrays may slow the code and fragment memory during the first time step, but it avoids going through complicated setups. One could also preallocate the arrays according to a first guess of the dimensions, as chosen by the user.
- at every time step the field is stored in DDH_FIELD through the pointer PFIELD

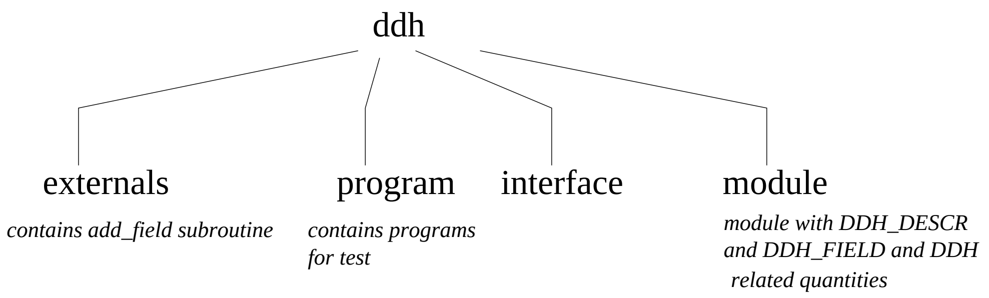
ddh
externals
program
interface
module
contains add_field subroutine
contains programs for test
module with DDH_DESCR and DDH_FIELD and DDH related quantities

FIG. 2 – Project ddh within the code's architecture

– at every time step, some transformations are done on the field according to its nature (and documented by its attributes), for instance conversion from θ to T... These operations also depend on the physics used (Meso-NH, Arpege...). Here it will be possible for users to add parts corresponding to specific needs, and to document them through attributes.

4 Architecture of the code

4.1 Creation of a project dedicated to DDH

If we want all the models to share the same tools for diagnostics it would be helpful to create within the code architecture a project dedicated to DDH in order to facilitate communications from the different physics to the DDH subroutines as shown in figure 2.

4.2 Diagnostic extraction from the physics

This will be done by inserting appropriate calls inside the physics routine, much like is already done in Arome/Meso-NH by the existing calls to routine BUDGET. One could either activate the new facility as an option inside BUDGET, or add optional calls next to every existing call to BUDGET, as preferred.

The extraction itself requires to supply, in the subroutine call, all relevant field attributes beside the field values themselves, as explained above in section 'Extracting a field from the physics'. This data will be stored into the self-allocatable structure, regardless of the nature of the field being stored.

The field may undergo some processing (e.g. variable conversion) before storage, in a way that depends on where we are in the physics. The field attributes should document where and how the field was processed. The information that is stored should be compatible with all subsequent uses that can be foreseen (e.g. not just DDH averaging).

4.3 Organization of the data flow

The DDH diagnostic facility performs some domain averaging and budget computation after the diagnostic extraction. These operations are performed at each timestep, after the physics computations, so that the raw recorded fields are accessible as NPROMA packets at the level of APLPAR/APL_AROME, where they may be used for other purposes.

For the DDH domain averaging, the Arpege subroutine cpcuddh.F90 (see DDH documentation for more details) is used and averaged fields are then written into file in ppfidh.F90 (which will be simplified since now with the self-documented structure, a loop on all elements in DDH_DESCR can generate the names of the fields to be written into the DDH file). The subroutine cpcuddh.F90 uses arrays (hdcvbx stored in module yomtddh) whose size is computed in setups (the total number of fluxes/tendencies depend on the options used for physics). Since these setups will no longer be used with the new data flow, these arrays will be either reallocated or initialized elsewhere in the code after a dummy call to the code that only computes the total size of DDH arrays (like the call to stepo from cnt4.F90 if CFU/XFU diagnostics are switched on).

Figure 3 summarizes the new data flow (which is the same for Arpege and Arome) within a time step.

4.4 MPI/OpenMP

To use this structure outside Arome in the future, it has to be not only MPI but also OpenMP compatible. This is a constraint we have to take into account because the fact that memory is either shared or distributed between processors may cause problems for OpenMP if different processors try to access simultaneously the module containing the DDH arrays. It should work with OpenMP if this module is flagged as private for OpenMP but for the moment this has not yet been tested.

5 Envisaged applications

This proposal is designed to prepare an implementation of the equivalent of DDH diagnostics in Arome, in order to allow intercomparison of physical terms with other models such as Arpege/Aladin (insofar as quantities from different physics are comparable). It is also believed to facilitate an implementation of the MAPFI proposal, by giving access at the APL_AROME level to quantities from the depths of the Arome/Meso-NH physics.

Further applications could include more detailed diagnostics in Arome (e.g. as inspired by the BUDGET facility of Meso-NH), the computation of new kinds of budgets (by using new extracted fields), and a conversion of the DDH mechanism

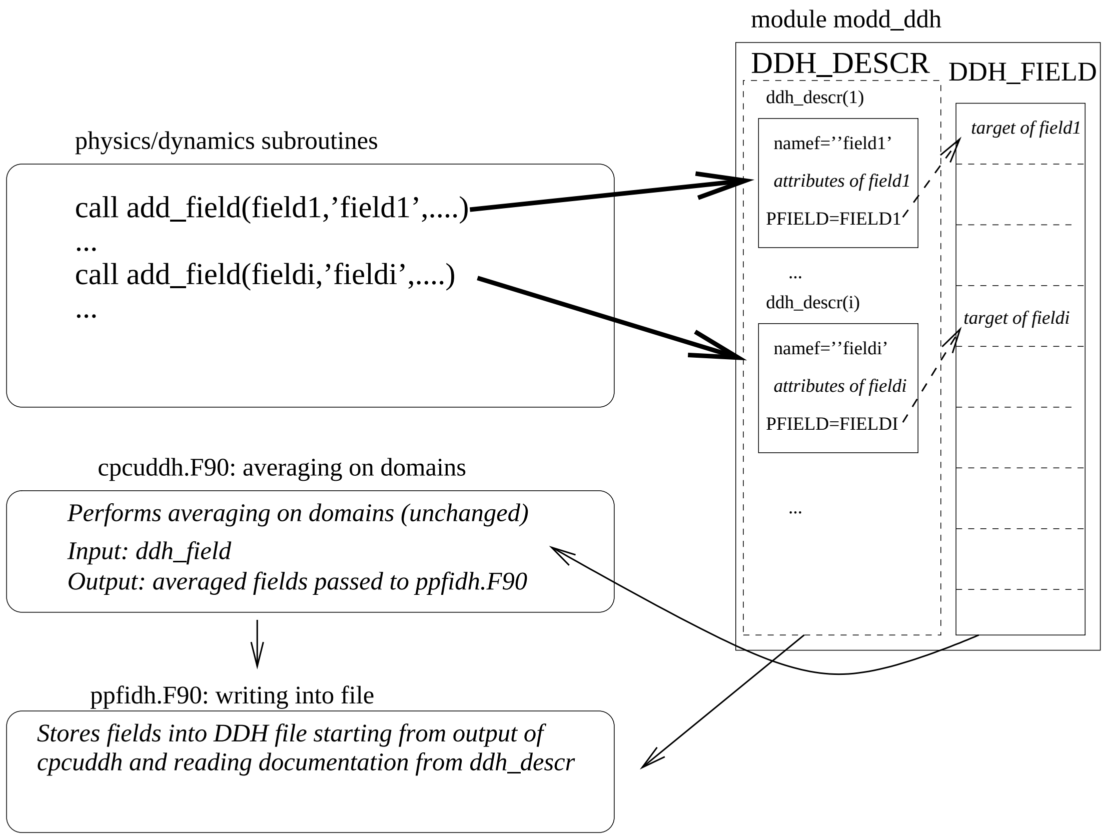
module modd_ddh
DDH_DESCR
DDH_FIELD
ddh_descr(1)
namef=''field1'
attributes of field1
PFIELD=FIELD1
target of field1
...
ddh_descr(i)
namef=''fieldi'
attributes of fieldi
PFIELD=FIELDI
target of fieldi
...
physics/dynamics subroutines
call add_field(field1,'field1',....)
...
call add_field(fieldi,'fieldi',....)
...
cpcuddh.F90: averaging on domains
Performs averaging on domains (unchanged)
Input: ddh_field
Output: averaged fields passed to ppfidh.F90
ppfidh.F90: writing into file
Stores fields into DDH file starting from output of
cpcuddh and reading documentation from ddh_descr

FIG. 3 – Organization of the data flow within a time step. Subroutine ADD_FIELD stores the field and the associated description into DDH_DESCR after possible transformations (bold arrows). Averaging on the domains is performed as in Arpege in cpcuddh.F90, the output being written into file in ppfidh.F90 using the description of the fields stored as attribute in DDH_DESCR.

of other models (e.g. Arpege/Aladin) under the new dataflow, if agreed by the relevant users. Until then, the new dataflow will be developed as a new option beside the older diagnostic facilities, in order to preserve backward compatibility.

6 Conclusion

A prototype with a large part of the above mentioned ideas has been successfully tested. This new version of the dataflow offers not only more facilities to add new quantities in the diagnostics but also more flexibility in terms of possible uses of these diagnostics. For developers, since the new code is considerably smaller and readable than the current one in Arome, it will be easier to debug and maintain when physics evolve in the future. We also expect an increase in the code's performance for Arome's DDH since the Meso-NH budgets part of the code (with a lot of unused (in Arome) options slowing the code) will be skipped. An another important aspect is that this tool, after being successfully implemented in Arome could be used in Arpege/Aladin [1]. Before going on with further work to upgrade this prototype version towards a beta version, discussion between the different possible users of this type of diagnostics is needed in order to raise possible new issues and needs regarding what different users would like these structures to offer.

[1]In fact it is easier to implement in Arpege than in Arome