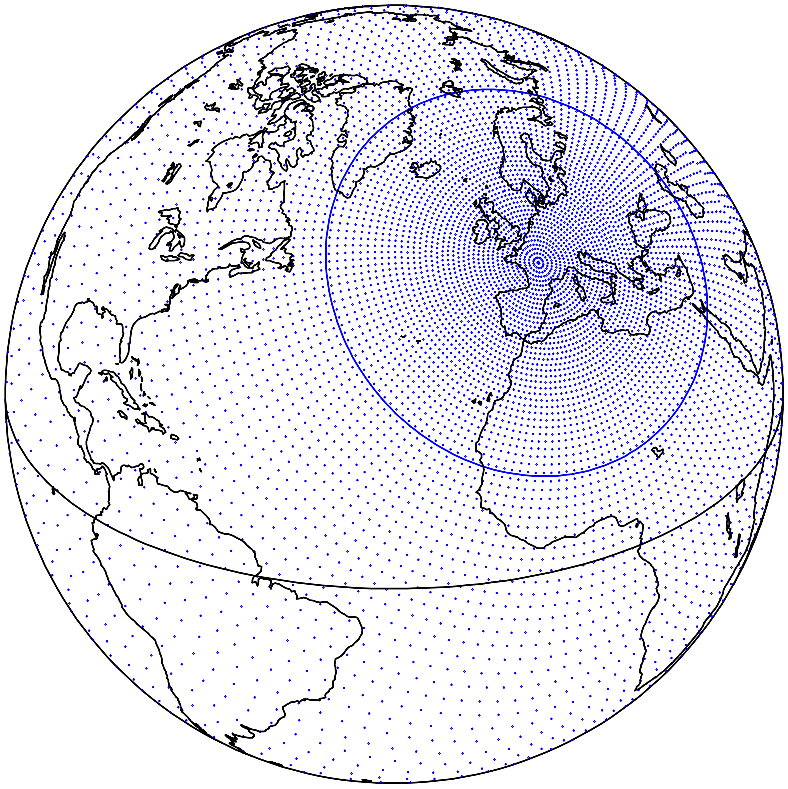
FullPOS technical guide for Arpege/Aladin cycle 25T1



R. El Khatib
METEO-FRANCE - CNRM/GMAP

October 30th, 2002

Contents

List of Figures

List of Tables

Chapter 1

Founder principles

1.1 Basic concept

FullPos is a non-independent software : it is designed to serve specifically the ARPEGE/ALADIN post-processing.
To get a post-processing fully consistant with the model itself, *FullPos* software has been completely embedded inside the ARPEGE/aladin software, in order that it can (and it should !) re-use model operators. This should also simplify a few maintenance operations[1].
The reliability of this concept has been ensured by the existence of a previous internal post-processing software (currently pointed out by the name of its leading subroutine : `POS`). However the target of this previous internal software was limited to vertical pressure interpolations for a few specific fields to be written out as spherical harmonics, while *FullPos* is designed to be a comprehensive post-processing tool.
FullPos is also designed to serve both operations (which implies : high efficiency to be run in real time situation) and research (which means : the ability to process various elaborated fields on various grids and vertical levels).

1.2 Scientific layouts

This section will list the processes that have taken part in the elaboration of *FullPos* . Presented prior to the architecture of the software, it should help understanding the conception of the code.

- Dynamical fields should be post-processable on pressure levels (P), potential vorticity levels (P_v), isentropic levels (θ), eta levels (η) including other definitions than the model originating eta levels, and on height levels above a given orography (z).
- Fields post-processed on P, P_v and θ levels should be interpolated vertically first, then horizontally. In between, it should be possible (as an option) to fit the fields in spectral space (to remove the numerical noise induced by the vertical interpolations). For a few specific fields (like geopotential, medium sea-level pressure, ...) for which the formulation of vertical interpolation induces potential inconsistancies, a filter in spectral space should be optionally performed.
- Fields post-processed on η and z levels should respect the profile of the boundary layer ; therefore they should be interpolated horizontally first, then re-ajusted with respect to the orography of the target grid.

[1] This is less and less true

- To ensure the inter-consistancy of fields interpolated on η or z levels, only the model primitive variables (U, V, T, q, P_s currently) should be horizontally interpolated : the other fields should be recomputed from the interpolated model primitive variables.
- Fields on horizontal surfaces should be homogenous ; in other words the small scale information should not pollute the interpretation of the output fields. This means that the fields which are composed of derivatives (like vorticity, divergence, vertical velocity as the integral of the divergence, but also any field on P_v levels) should be filtered in a spectral space of homogenous resolution[2]. The intensity of this filter should depend of the output grid resolution[3].
- Physical fields from the model, including cumulated fluxes and instantaneous diagnostics, should be post-processable : their interpolations require often specific treatments, like the land/sea aspect (only points of the same nature should serve the interpolations), the control of the validity domain for the output values (for instance : the interpolated land/sea mask should be either 0. or 1.), or the interdependencies of the post-processed physical fields (for instance : deep soil temperature should be interpolated as its anomaly with respect to surface temperature, which implies to interpolate surface temperature prior to deep soil temperature).
- It should be possible to interpolate "physico-dynamic" fields : these are fields defined on a surface and computed with model physical surface fields as well as upper air dynamic fields (CAPE is one of them). If computed on the model originating grid, this should be easy (because the environment is then very similar to the model gridpoint environment), but if computed on another grid, this implies to interpolate horizontally almost all the model pronostic fields (upper air as well as surface).
- It should be possible to use *FullPOS* to make full history files (for coupling, nesting, multi-incremental variational purpose, or even bogussing). This means that the fields (mainly the physical fields and the dynamic fields post-processed on η levels) should be interpolated with respect to a target orography (ie : the orography of the output grid) which should be spectrally fitted in the output spectral geometry ; and also that it should be possible to write out the dynamic fields as spectral coefficients for the target geometry in this case.
- While doing bogussing, to prevent from getting "walls" at the border of the bogussing area, the target grid should get only the interpolated increments from the source fields.
- It should be possible to use climatology data in order to interpolate with a better accuracy a few surface fields ; instead of a straightforward interpolation, we would interpolate the anomaly of a field with respect to the climatology, or even : we would impose the whole climatology field if it is a constant field (land-sea mask for instance).
- In case of gridpoint outputs on a complete ALADIN grid, the extension zone should be computed as well, taking into account the realism of the physical fields.
- In the initial design, horizontal interpolations had to be quadratic exclusively[4].
- Wind-related fields (like vorticity, divergence, etc) should be computed from the wind components so that all these fields are consistent.

[2] Actually not done for wind on PV levels

[3] Not effective for aladin. Is it a bug ? However we usually write out only one grid from aladin. Anyhow this should be harmonized with arpege

[4] Things are supposed to change in the future, so that each field could have a specific interpolation kind : quadratic, bilinear or no interpolation but the value of the model nearest point is adopted.

1.3 Technical requirements

Beside the scientific aspects, various technical aims had to be achieved :

- it should be possible to post-process during the model direct integration ("in-line post-processing") as well as after ("off-line post-processing"), both solution giving the same results. This implies that *FullPos* should not be a specific ARPEGE/ALADIN configuration but a package which could be called inside the direct model temporal loop, and that the packing of fields in history files should be considered.
- *FullPos* should benefit, as the model does, from the (super)computer hardware architecture, that is : the memory distribution today and probably OPEN-MP tomorrow. [5]
- *FullPos* should be cheap.
- *FullPos* should be modular and should not spread itself all over the code. [6]
- *FullPos* users interface should be ergonomic. This should not mean that the users interface should be restricted to a limited number of namelists parameters, but rather that the namelists should be easy to set, with meaningful parameters.
- *FullPos* should stick to the ARPEGE/ALADIN interfaces standards : namelists parameters for the users interface and ARPEGE/ALADIN files for the I/O data.
- the list of post-processing fields, levels and horizontal domains per post-processing time range should be flexible.
- The horizontal output format has been restricted to : either one gaussian grid, or one ALADIN grid, or a set of LATLON grids, or one definition of spectral coefficients.
- One should be able to pack each post-processing field on a tunable specific number of bits.

1.4 Technical limitations

Nobody is perfect, and the code is not, neither : before understanding a few technical choices concerning the conception of *FullPos* , one should remember the following technical limitations existing at the time this software has been first conceived :

- The software has been elaborated upon a Fortran 77 compiler ; Fortran 90 was not available at that time.
- To extend the possibilities of the software, ARPEGE/ALADIN system was currently using Fortran Cray extensions, like a memory manager system based on the pre-allocation of a heap ; from this heap it was possible to allocate arrays. This system was not so flexible as the "`ALLOCATE`" statement of the Fortran 90 language.
- Spectral transform were not modular.
- The ARPEGE/ALADIN was originally designed for a multi-processor vector machine with limited central memory, using multitasking and having fast I/Os ; while the architecture of today is distributed, not always vector, and with relatively slow I/Os but with a large central memory.

[5]Formerly : the multitasking and the I/O scheme.

[6]This has been a failure definitely, partly because of the technical limitations/constraints at the time of the first conception,. However things has changed so that *FullPos* is getting more concentrated and modular. Its "externalization" from the code arpege/ifs/aladin is even under consideration.

Chapter 2

General conception

2.1 Architecture

This section will describe the main subroutines involved in *FullPos* and how they are articulated between one another.

2.1.1 General implementation

FullPos is included in the configuration 001 of ARPEGE/IFS/ALADIN : this is to enable the in-line post-processing as well as the off-line post-processing (in the latter case, it is enough to run the model on zero time step but with the post-processing activated).
However it remain theoretically possible to implement *FullPos* in any other configuration. For instance, one can imagine to implement it CANARI (configuration 701).

A logical key : `LFPOS` has been implemented in order to select either *FullPos* or the previous internal post-processing. In a near future, this old post-processing should be removed, but this will not be so easy since it is also used for the so-called "movies". Note also that `LFPOS` and the binary namelist variable `N1POS` (controling the main post-processing flow) could then be merged.

The general mechanism of the code follows the "control routines cascade" (`CNT0` to `CNT4`) of the model (see the figure 2.1.1 on page 11), though all the subroutines are not used. The next list describes the purpose of each control subroutine :

MASTER : main program ; it does not do anything but calls the control subroutine `CNT0`.

CNT0 : Setup (control of level "0") ; it initializes the constants or the namelists variables as scalars or arrays.

CNT1 : Control of level "1". *FullPos* is concerned by this subroutine for two reasons. The first one is because the namelist `NAMCT1`, which contains two variables used by *FullPos* (`N1POS` and `LRFILAF`), is read below this subroutine ; the second one is because the key `LFPOS` is still used in this subroutine for a completely old-fashioned reason. These situations are not proper, and in future developments *FullPos* should no more be concerned by `CNT1`[1].

CNT2 : useless for *FullPos* .

[1]LFPOS has already been removed from CNT1 in the next cycle 26.

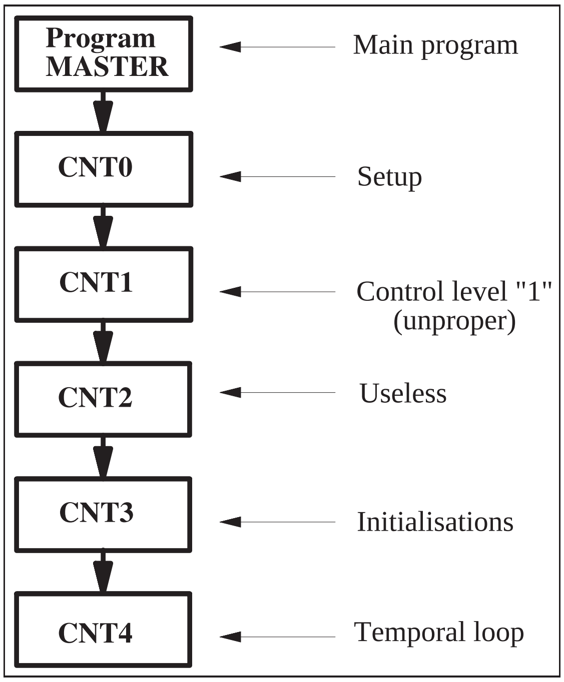


Figure 2.1: *general mechanism of the post-processing inside the model*

CNT3 : Initializations (control of level "3") ; it reads the initial conditions data (and possibly the needed climatology data) and compute the working arrays which would depend of these data.

CNT4 : Temporal loop (control of level "4") ; it initializes the time-dependent post-processing variables (like the list of fields to be post-processed for the current model step) and performs the post-processing inside the model temporal loop.

When *FullPos* is configured to make history files, things are more complicated since the code has been conceived with non-modular spectral transform : the control cascade should be invoked two times in order to change the setup of the spectral transforms.
The first control cascade will be called the *external* part, while the second control cascade will be called the *internal* part.
This mechanism is achieved by a supplementary control subroutine named `CPREP4` which is called once by `CNT4` (in the external part) and once by `CNT3` (in the internal part). The location of `CPREP4` in `CNT4` for the external part is justified by the fact that this mechanism could potentially work in the "in-line" mode, though this possibility has never been used, and has even been removed from the code. The location of `CPREP4` in `CNT3` for the internal part is justified by the fact that this part is *really* internal : it is "out of the temporal loop" ; actually this internal part is justified only because the spectral transforms were not modular at the time of this conception. At the end of the external part, `CPREP4` calls again `CNT0` after it has released all the allocated arrays. So we have to cope with a recursive cascade of subroutines (see the figure 2.1.1 on page 12). This needs supplementary control items in order to be able to leave this never-ending loop :

LFPSPEC : namelist variable which should be set to `.TRUE.` for getting *FullPos* configured to make history files. Note that this so-called "configuration" is not a real one, since the code is still implemented in the configuration 001. However, this system is well-known as "configuration (e)(e)927 for historical reasons (it has replaced the previous true configuration

926).

LFPART2 : internal key telling whether the running part is the external one (`LFPART2 = .FALSE.`) or the internal one (`LFPART2 = .TRUE.` ; also called : "part 2").

ncf927 : pilot file to control the never-ending loop on `CNT0` : its existence means that the post-processing is over.

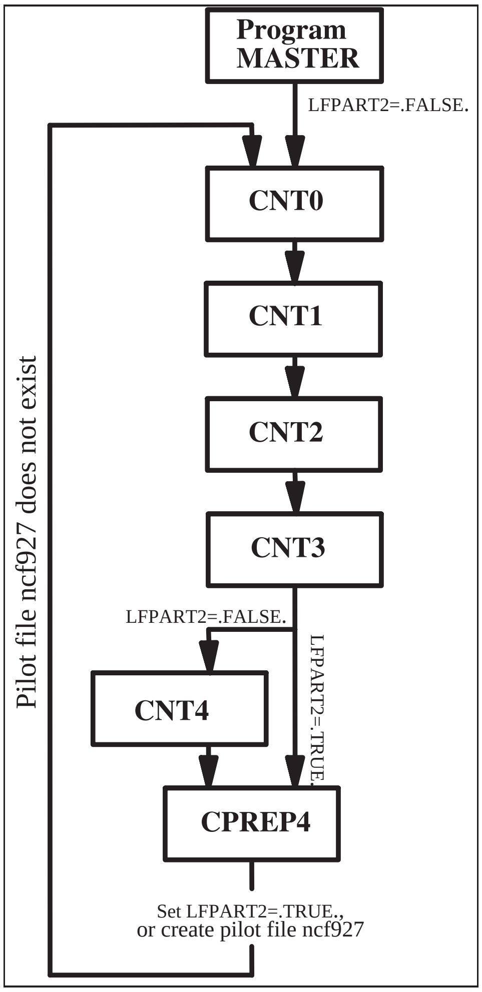


Figure 2.2: *general mechanism of the so-called configuration (e)(e)927*

2.1.2 Setup

Still following the general structure of the code ARPEGE/ALADIN, the setup of *FullP*OS is split in two main subroutines : `SUOYOMA` and `SUOYOMB`. One should remind that the original reason for splitting the setup was because of the memory management handling on the former Cray computer. Today there is no reason for such splitting, but we have to live with the past, before reforming it.
Concerning the post-processing, it was mostly important to make a hierarchical setup, in order to have an easy setup using consistent default values as much as possible. Sometimes, this become contradictory with the principle of modularity ; as a consequence, in some places, the setup of

FullPos is spread over the setup of the model. In the future this matter of fact should be reduced by a progressive reorganization of the setup of the model.

Notice : In the case of the family of configurations 927, the internal call to `CNT0` (case : `LFPART2=.TRUE.`) should be limited to what needs to be re-initialized only, because of the change of geometry. For instance the message passing should not be re-initialized. One will also notice a call to the synchornization barrier : this is because the I/O operations should be finished before starting the internal part. In short one will notice that the key `LFPART2=.TRUE.` is often spoiling the code. It would be advantageous to reconsider this "second part" so that instead of recalling the whole control subroutines cascade, it would recall only its needed parts.

SU0YOMA

This subsection will list the subroutines which are (more or less) specific to *FullPos* in the first part of the setup. Beside, the figure 2.1.2 on page 14 will show the scheme of this subroutine. One should not forget that the post-processing is also using a large amount of variables coming from the model itself (like logical unit numbers). In the scope of the externalization of *FullPos* , it will be necessary to create a data module to transfer these model variables to internal post-processing variables.

SUAFN (SetUp Arpege Field Names) : To initialize extensive descriptors about the fields which are post-processable (which fields and how they should be treated). This subroutine is interfaced with a namelist (`NAMAFN`) and consequently initializes the corresponding module `YOMAFN`. It calls three subroutines :

- **SUAFN1** : setup default values, using some model variables as a background ; so it must be called after `SUDIM1`.
- **SUAFN2** : control the users values and complete with supplementary internal control variables.
- **SUAFN3** : print out the initialized descriptors.

Remarks :

- `YOMAFN` is now used in the model to read non-hard-coded ARPEGE field names. This is embarassing because it makes both the model and *FullPos* less independent. The solution should be to create a derived type variable in the model to setup the ARPEGE field names and possibly other characteristics (GRIB code and the minimum number of bits the field should be encoded with). These derived type would then be used to setup *FullPos* .
- `SUAFN` and its relatives treat all the post-processing fields (from dynamics or physics) : this matter of fact is penalizing the maintenance. It would be better to split this ensemble in two parts : one for dynamics and one for physics ; then each part could be re-unified in one single module.

SUFPC (SetUp FullPos Computation) : originally designed to setup the list of fields to post-process as well as all scientific or technical options used for the post-processing. This subroutine as become a mess, depending on a lot of model variables. It should be split one of these days in one subroutine for the various post-processing options, and one subroutine for the list of fields to post-process.

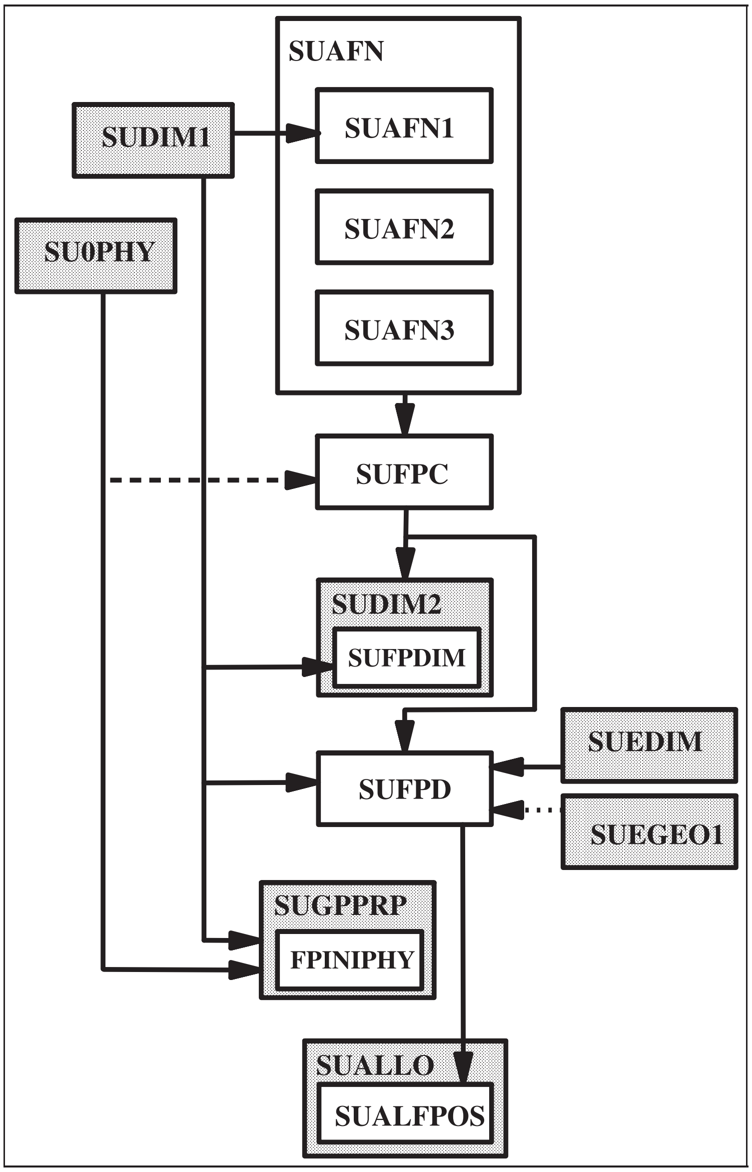


Figure 2.3: *Setup, first part : SU0YOMA. The greyish areas correspond to model subroutines.*

SUFPDIM (SetUp FullPos Dimensionning) : originally designed to setup dimensions of arrays to be shared with the model (in the spirit of re-using the model allocated arrays to save memory). This subroutine was highly related to the shared memory architecture and the internal spectral transforms, that is why it is called inside `SUDIM2`. Today it has two new purposes :

- To complete the descriptors in `YOMAFN` with informations from the model settings (more exactly : which variables are primitive) ; for that it needs to be run after `SUDIM1` and `SUFPC`.
- To compute static dimensions of data arrays now specific to *FullPos* , so it needs to be called after `SUFPC`.

Remarks :

- In the next cycle (26), this subroutine has just been removed, because it is enough and more efficient to work with dynamic dimensionning only. This was not possible at the time of the conception because the I/O scheme needed static dimensionings for the workfiles.

- There is a dirty trick in `SUDIM1` : the namelist `NAMFPC` is read there in order to initialize the variable `NFPCLI`. This is the consequence of a hurried (and hurting) split of `SUDIM` in `SUDIM1` and `SUDIM2`. The cleaning might come from the future split of `SUFPC`, but a more robust solution would be to have an independent gridpoint buffer to contain the input climatology.
- The fact that a part of a descriptor is initialized in `SUAFN*` and another part in `SUPDIM` proves that this descriptor should be split !

SUFPD (SetUp FullPos Domains) : to initialize the dimensions and bounds of the output post-processing (sub)-domains. To take advantage of default values which would be the model dimensions, this subroutine needs to be called after `SUDIM1` for ARPEGE and after `SUEDIM` and `SUEGEO1` for ALADIN. Unfortunately `SUEGEO1` is called late in `SU0YOMB` for the time being. The solution is probably to move `SUFPD` inside `SUBFPOS` (see `SU0YOMB` below).

FPINIPHY (FullPos INItialization of physics) : to control that all the model physical fields pointers which will be used by the post-processing are initialized. This is a fragile subroutine, which should be merged with parts of `SUFPC` (tests about physical aspects) and it should rather work on the physics logical keys and dimensions (from the namelists `NAMPHY` and `NAMDPHY`, actually prior to reading these namelists) rather than pointers values (we do not know what is the "undefined" value). For the time being it is called by `SUGPPRP` in order to get this "undefined" pointer value.

SUALFPOS (SetUp ALLocations of FullPOS) : to allocate various arrays. The location of this subroutine inside `SUALLO` and at the bottom of `SU0YOMA` is purely the consequence of the conception on the former Cray machine. In the next cycle (26), this subroutine has been moved inside `SUBFPOS` below `SU0YOMB` (see below). However the call to this subroutine is conditionned by the initialization of the model and the post-processing dimensionning.

SU0YOMB

In the second part of the setup, things are fortunately much more condensated as there are less interactions with the model itself.
The figure 2.1.2 on page 16 shows the scheme of this subroutine.

SUCFUFP (SetUp Cumulated FlUxes for FullPos) : To activate the model cumulated fluxes switches which will be needed by the post-processing. It is called inside `SUCFU`. This routine is typically an input-output interface between the model and *FullPos* (finally like `FPINIPHY` described above).

SUXFUFP (SetUp X —instantaneous— FlUxes for FullPos) : same as `SUCFUFP` for the instantaneous fluxes ; called inside `SUXFU`. Same remark.

SUBFPOS (SetUp part B of FullPOS) : this subroutine contains a lot of specific subroutines for *FullPos* . Here is a rough description of them :

- `SUFPG` (SetUp FullPos Geometry) : To initialize the geometric parameters of the output grids . . . and the post-processing gridpoint distribution as well !

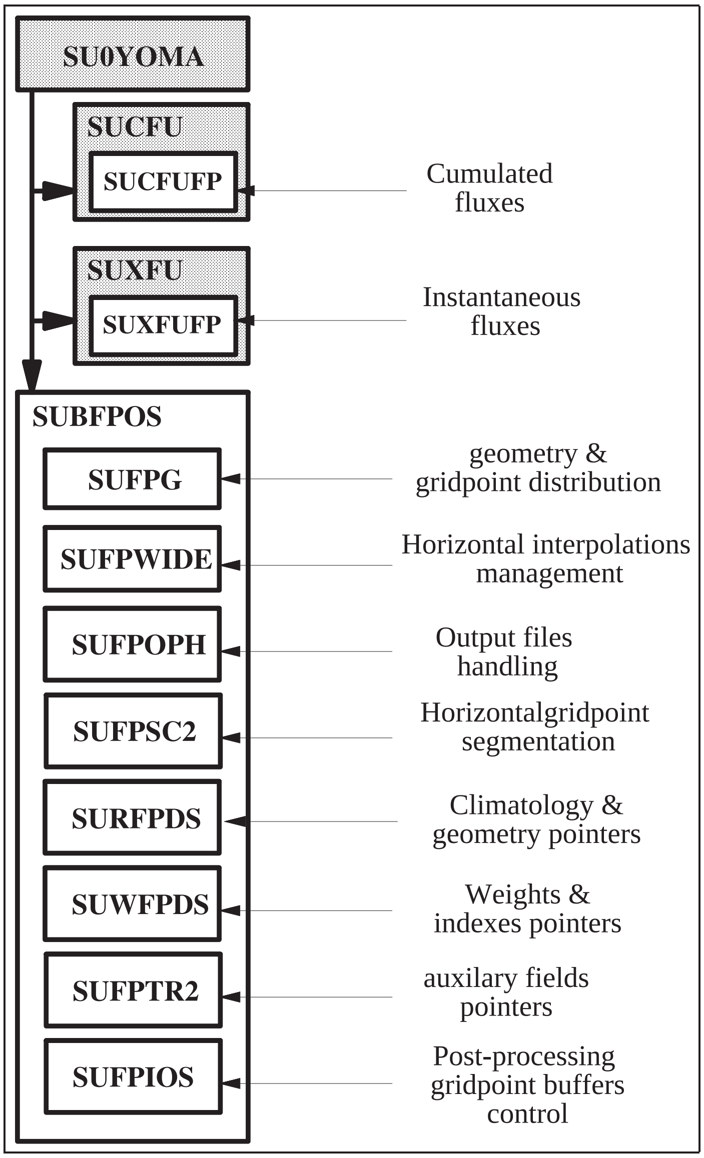


Figure 2.4: *Setup, second part : SU0YOMB. The greyish areas correspond to model subroutines.*

- `SUFPWIDE` (SetUp FullPos WIDE) : To initialize the parameters and working arrays needed for the horizontal interpolations mechanism.
- `SUFPF` (SetUp FullPos Filters) : To initialize the spectral filters parameters.
- `SUFPOPH` (SetUp FullPos Output Parameters Handling) : To setup output files names and their autodocumentation part. Note that the files logical unit numbers are initialized in the model subroutine `SULUN` (A clear problem of interfacing the model and the post-processing).
- `SUFPSC2` (SetUp FullPos SCan 2) : To initialize the horizontal segmentation in the post-processing gridpoint calculation and some corresponding control arrays.
- `SURFPDS` (SetUp Real fields FullPos DeScriptors) : To initialize fields pointers in output buffers for the target climatology and target geometry. Notice : there were no reason to put altogether these fields (climatology and geometry) but to limit I/O operations in the framework of the former I/O scheme. Today it would be advantageous to split climatology and geometry because it would improve the modularity and the

understanding of the code.

- `SUWFPDS` (SetUp Weights FullPos DeScriptors) : To initialize fields pointers in output buffers for the horizontal interpolations weights, adresses and indexes. Note that fields in this buffer are all considered as real fields, though some of them are actual integer fields. In the future it would be advantageous to build a more clever field organization (using either the Fortran function `TRANSFER` in order to save memory, or just by creating a integer gridpoint buffer : now very easy).
- `SUFPTR2` (SetUp Fullpos PoinTeR 2) : To initialize the "auxilary fields" pointers, that is the special surface fields which should be horizontally post-processed prior to the current flow of post-processed fields, because they will have to be used for the current post-processing flow.
- `SUFPIOS` (SetUp Fullpos IO scheme) : To initialize the control parameters for each "post-processing gridpoint buffer".
  Remarks :
  - This subroutine was originally designed to initialize the former I/O schemes for the post-processing gridpoint buffers, but its purpose has now completely changed.
  - This subroutine, which is reading the namelist `NAMFPIOS` containing a unique variable, should be merged with `SUFPSC2` which also reads the namelist `NAMFPSC2` containing a unique variable ; so one of these two namelists could be removed.
  - It would be more robust to setup each buffer descriptors at the time the buffer is allocated. Actually this has been already done for the next cycle (26). To go further, the next step would be to define the post-processing gridpoint buffers as derived types, together with their operators (allocation and initialization, deallocation, and even reading in and writing out).

  Notice : One will notice that in `SUBFPOS` a few variables need to be initialized even if *FullPos* is not active (`LFPOS=.FALSE.`) : this clearly shows that these variables have an active role in subroutines which are common to the model and the post-processing. In the scope of the externalization of *FullPos* , this should disappear.

2.1.3 Initializations

The figure 2.1.3 on page 18 shows the scheme of this subroutine.

The initializations part is composed of two main parts :

- The initialization of the input meteorology data (including possibly climatology data) which is common with the model : `SUINIF` for ARPEGE and `ELSAC` for ALADIN.
- `SU3FPOS` (SetUp level 3 FullPOS) : to initialize the following data buffers/arrays :
  - The buffer containing the output climatology and geometry (refer to `SURFPDS`). This is performed by the subroutine `SURFPBUF`.
  - The buffer containing the weights and indexes for horizontal interpolations (refer to `SUWFPDS`). This is performed by the subroutines `SUEHOW1`/`SUHOW1`/`SUWFPBUF`.
  - The matrixes for the spectral filters (performed by the subroutine `FPFILTER`).

  Remarks :

  - The occurence of a piece of code in `CNT3` is justified only if it depends of the initial condition fields ; that is why `SURFPBUF` could (and should !) be moved away from it, and put into `SUBFPOS` for instance.

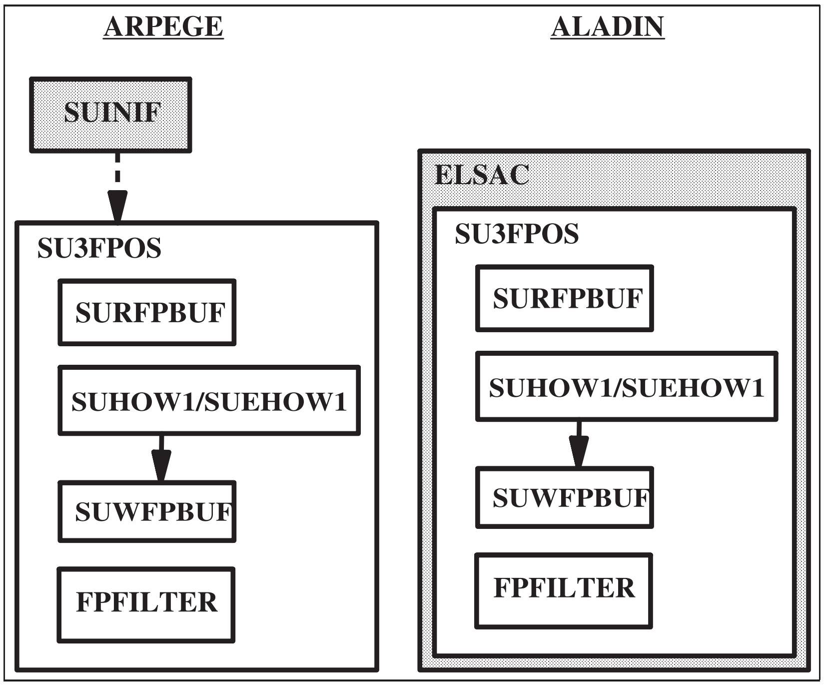


Figure 2.5: *Initializations : CNT3. The greyish areas correspond to model subroutines.*

- In the next cycle (26) `FPFILTER` has been moved away and put into the control subroutine for dynamic post-processing (see `DYNFPOS` later) because this place is the best one to limit the memory cost (special notice : the array should be allocated if necessary at the beginning of the scans "Vertical then Horizontal", and systematically deallocated at the end of this scan for an optimal memory management).
- The occurence of `SUWFPBUF` inside `CNT3` is justified because to compute the land-sea-mask-dependent interpolations weights, we need first to interpolated the model land-sea mask (and possibly the surface temperature) when the output climatology is not at disposal.
- `SU3FPOS` had to be moved inside `ELSAC` for ALADIN because the computation of weights, if it uses the model surface temperature, should be performed before the digital filters initialization of the model (true ?).
- A possibility to concentrate *FullPOS* could be just to move `SUBFPOS` inside CNT3 ?

2.1.4 Temporal loop

The figure 2.1.4 on page 19 shows the scheme of this subroutine.

The temporal loop subroutine is composed of the following parts :

MONIO (MONitoring of I/O) : To stamp the model time steps with the post-processing events ; it is called before the temporal loop. This subroutine is not specific to *FullPOS* , and thus it is penalizing its modularity. The solution would be first to move out from the model the variables `N1POS`, `NFRPOS` and `NPOSTS`, and put them into a post-processing-specific module.

SU4FPOS (SetUp level 4 FullPOS) : To setup the time-dependent variables for the post-processing (ie : the actual list of fields to post-process).

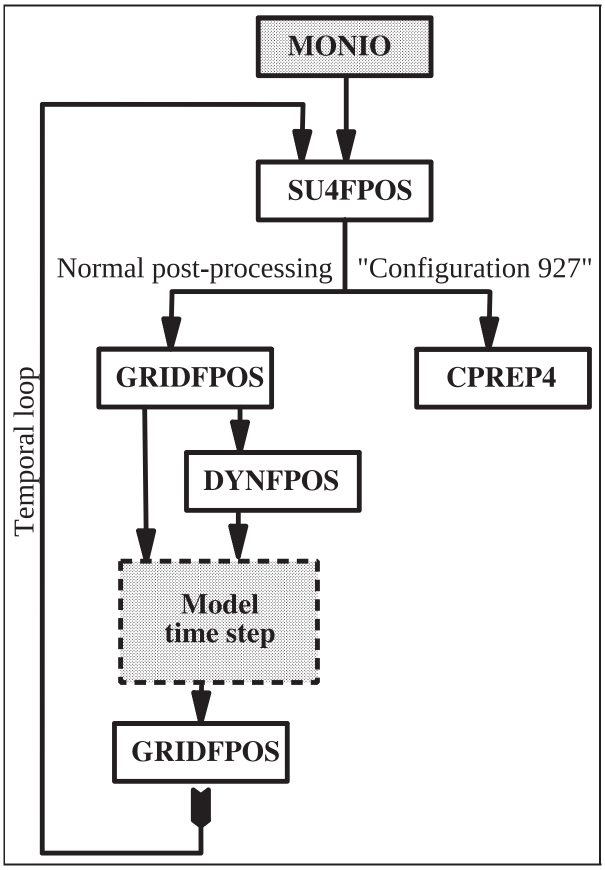


Figure 2.6: *Temporal loop : CNT4. The greyish areas correspond to model subroutines.*

GRIDFPOS (GRIDpoint FullPOS) : to perform the post-processing of physical fields (surface fields or fluxes). It can be invoked either at the beginning of the time step, or at its end (Notice : the conditions of call are not clear since ECMWF never use the cumulated fluxes buffer ; this must be the residual effects of an old-fashioned specification related to the shift between instantaneous physical fields and the fluxes, in contradiction with the specification for in-line/off-line reproductibility).

DYNFPOS (DYNamic FullPOS) : to perform the post-processing of dynamic fields. It is called at the beginning of the model time step to have the correct clock. It should be called after `GRIDFPOS` (non-lagged call) in order to have the needed surface fields already post-processed before starting the physico-dynamical post-processing. In the scope of the externalization of *FullP*OS it will be interesting to have the model `STEPO` sequences competely independent from `DYNFPOS` (no overlap on the configuration of `IOPACK`).
Note that the non-lagged call to `GRIDFPOS` could simply be put at the beginning of `DYNFPOS`.

CPREP4 (Configuration PREParatory level 4) : to control the mechanism of the so-called ”configuration 927”.

Notice : the logical key `NFPCTO` is an internal key to control the so-called ”bogussing” configuration, which will be described later. For the moment, it is enough to know that `NFPCTO` is greater or equal to 1 for ordinary post-processing.

2.1.5 Control of the configurations 927

This subroutine controls the whole family of "configurations 927", that is "927" (arpege to arpege), "e927" (arpege to aladin), "ee927" (aladin to aladin), and "bogussing" (aladin to arpege).
The figure 2.1.5 on page 20 shows the scheme of this subroutine.

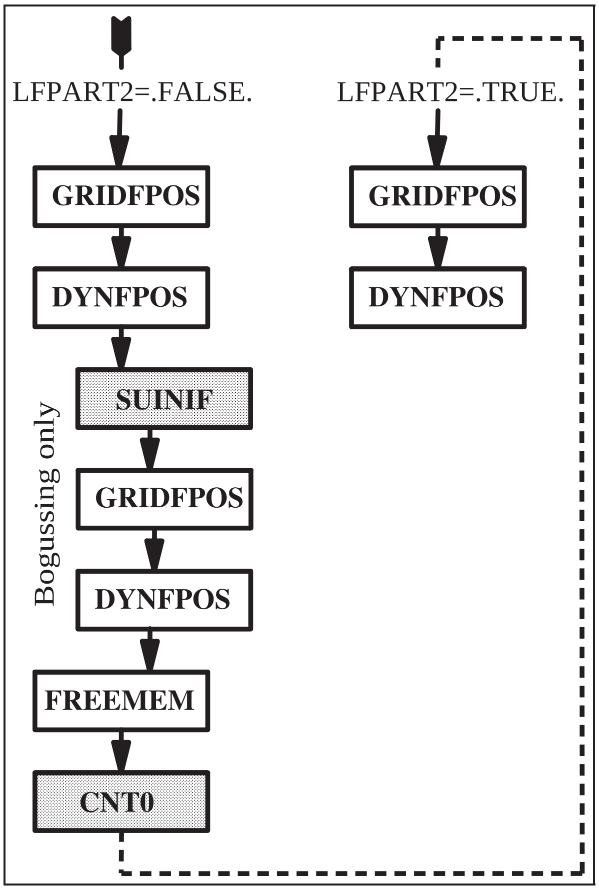


Figure 2.7: *Control of the configurations 927 : CPREP4. The greyish areas correspond to model subroutines.*

The main ingredients are again `GRIDFPOS` and `DYNFPOS`.

- In the first part (`LFPART2=.FALSE.`) the *horizontal* post-processing is performed.
- Then all the allocated arrays are released (`FREEMEM`) and the program is re-run from the (`CNT0`).
- Finally in the second part (`LFPART2=.TRUE.`) the *vertical* post-processing is performed.

Remark : again here we can guess that the key `LFPART2=.TRUE.` will complicate the setup since we need to select from the namelists the horizontal aspects first, then the vertical aspects.

The bogussing consists in a local modification of a history file ARPEGE by a file ALADIN. For that, we actually need to compute the increments to add to the ARPEGE file. Since we do not have coded yet the tangeant linear of *FullPos* , we should perform first the horizontal interpolations on two separate ALADIN files (which difference corresponds to the increments).
The bogussing is activated by the key `NFPINCR` in namelist (0 for no bogussing, 1 for bogussing). Its control is realized be the key `NFPCTO` which can have the following values :

- `NFPCT0=-2` : first part of the "external" part (for bogussing only)
- `NFPCT0=-1` : last part of the "external" part (for all)
- `NFPCT0=0` : second ("internal") part of the post-processing (for all).

However, and for preliminar test, the bogussing has been enabled out of the "927 configurations system", that is for normal post-processing, through the following options (see `CNT4` in ALADIN) :

- `NFPCT0=1` : last normal external part of the post-processing (for all)
- `NFPCT0=2` : first external part of the post-processing (for bogussing only)

2.1.6 Physical fields post-processing

The post-processing for physical fields is split in two flows :

1. Post-processing of auxilary surface fields : this is an intermediate step before the actual post-processing, as these auxilary surface fields should be re-used for the post-processing of physical fields and physico-dynamical fields.
2. Post-processing of physical fields. This step is completed by the computation of the extension zones of the post-processed fields (if the output domain includes such an area).

The figure 2.1.6 on page 21 shows the scheme of this subroutine.

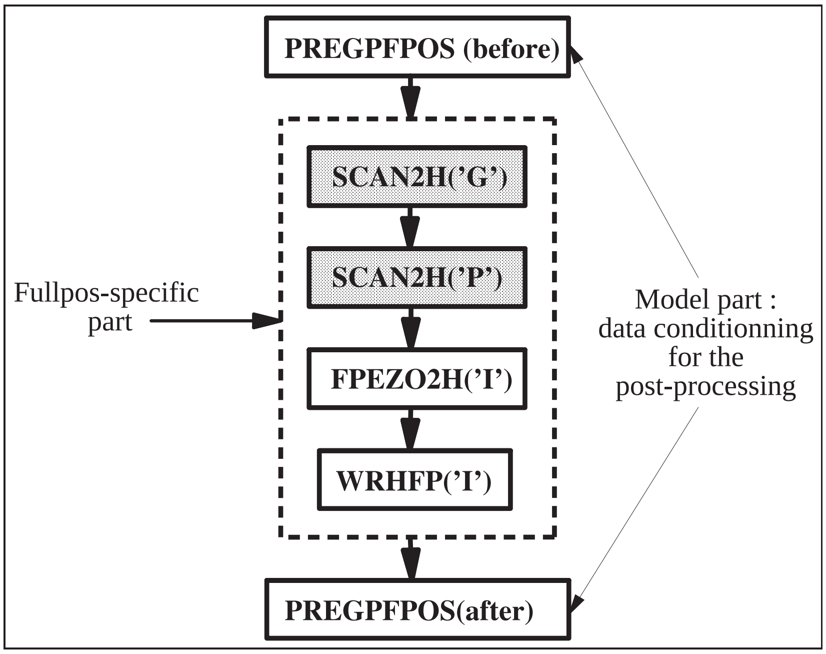


Figure 2.8: *Physical fields post-processing : GRIDFPOS. The greyish areas correspond to model subroutines.*

Following the mechanism of gridpoint calculations in the model and the horizontal interpolations for the semi-lagrangian scheme, the horizontal interpolations for the post-processing have been embedded inside the model subroutine `SCAN2H`, invoked with a specific configuration string.
The biperiodicization of post-processed fields is treated by a specific subroutine : `FPEZO2H`. Note that the content of this subroutine is "unbalanced" since both biperiodicizations for the auxilary

surface fields and the physical fields are treated within the same call, while the symetry of the code structure indicates clearly that the two biperiodicizations should be performed separately.

The writing out to files of the gridpoint post-processed fields is performed by the leading subroutine `WRHFP` (for ARPEGE/ALADIN at least), with a configuration letter.

Remark : `GRIDFPOS` is actually a model-dependent subroutine, because it contains the subroutine `PREGPFPOS` which purpose is to pack the data before, and restore them after, in order to enable the in-line/off-line reproductibility of the post-processing. All that is between the two invokation of `PREGPFPOS` is purely specific to *FullPos* , and thus it should be isolated in a specific subroutine ; while `PREGPFPOS` and the container subroutine should not be subject to externalization, unless `PREGPFPOS` would be an external module called by the post-processing (another problem of interfacing !).

Further notices about `PREGPFPOS` :

- There is no reason to have one single subroutine for both operations "before" and "after" : there should be one subroutine for "before" (`PREGPFPOS`) and one for "after" (`POSTGPFPOS` ?) ; but both subroutines could be contained in the same module.
- To avoid evident duplication of code, the subroutine should treat only one gridpoint buffer, and then called three times.

2.1.7 Dynamic and physico-dynamic post-processing

The figure 2.1.7 on page 22 describes this subroutine.

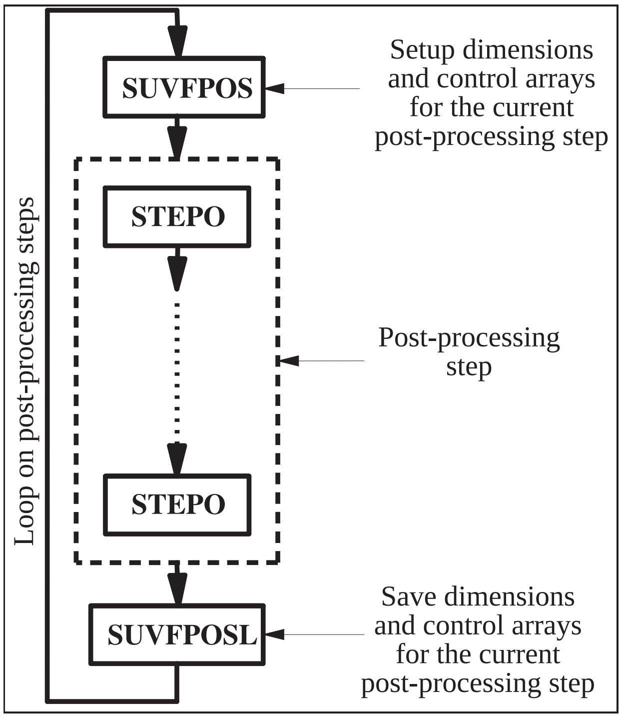


Figure 2.9: *Dynamical and physico-dynamical post-processing : DYNFPOS.*

The dynamic and physico-dynamic post-processing is based on the following concepts :

- It uses the model subroutine `STEPO` to combine spectral transforms, gridpoint calculations, spectral calculations and I/O operations.

- A "post-processing step" is composed of a succession of calls to `STEPO`, since one single call to this subroutine would not be enough to treat at once the vertical interpolations, the horizontal ones, the spectral fit and the spectral filters.
- To each kind of post-processing level corresponds a "post-processing step".

There are three kinds of post-processing steps :

The post-processing on P, P_v or θ levels : it is composed of three calls to `STEPO` :

1. vertical interpolations and spectral filters
2. horizontal interpolations
3. outputs

The post-processing on z or η levels : it is composed of four calls to `STEPO` :

1. computation of the model primitive fields on the model levels
2. horizontal interpolations of the model primitive fields on the model levels
3. computation and vertical adjustment of post-processing fields
4. outputs

Notice : the first two sequences have been separated because such a conception was easier and faster. However they could now be merged : the idea would be to pipe the output array from `POS` with the input core array for horizontal interpolations. Anyhow this must not lead to duplication of code (`VPOS` to be re-visited ?).

The post-processing on the originating model primitive fields and levels : this is an internal post-processing step dedicated to the external part of the "configuration 927", and composed of three calls to `STEPO` :

1. computation of the model primitive fields on the model levels
2. horizontal interpolations of the model primitive fields on the model levels
3. outputs

Notice : this post-processing step is very close from the beginning of the previous one, except that the data flow in output is different (this is justifying another configuration sequence).

Remarks :

- The mechanism of `STEPO` is so that the writing out of the post-processing fields, for a given post-processing step, is always "postponed" to the beginning of the next post-processing step. This is complicating the code because the descriptors control arrays, which are used to drive a given post-processing step, are overwritten (by the following post-processing step) before the current step is over ; that is why (a part of) these descriptors must be saved before starting a new post-processing step.
- In the scope of the externalization and the simplification of *FullPOS* , the model subroutine `STEPO` should be abandonned for the post-processing, and a new specific one should be written, with a well-adapted configuration string, like the following one :

0. Model fields (adapted) inverse transforms

1. Vertical interpolator
2. Direct spectral transforms
3. Spectral filters
4. Inverse spectral transforms
5. Horizontal interpolator
6. Residual computations and vertical adjustments
7. Biperiodicizator
8. Output fields preconditioner
9. Outputs

The management of the I/Os is still wide open : what kind and conditionment for the input data and the output data : should the model inverse transform or the writing out to files be parts of the post-processing package or not ?

- The output configuration letters from the subroutine `SUFPCONF` should be arrays dimensionned to the maximum number of scans, stored in a module and initialized in the setup (`SUBFPOS`).
- The double call to `SUVFPOS` is odd : there should be one (simplified ?) subroutine to find out whether or not the sequence on model fields and levels should be performed or not.
- The loop on post-processing steps should be split in two parts : one for the P, P_v, θ levels and internal sequences ; the other one for the z and η levels. In between, spectral matrixes arrays should be released in order to save memory.

The table 2.1.7 on page 25 shows the STEPO configuration letters used for the post-processing.

However, all the combinations are not possible : they are build in function of the horizontal output format and the vertical kind of levels, as described in the table 2.1.7 on page 26. Note that whenever the configuration letter "P" occurs, it can be replaced by "0" if the spectral fit for the current post-processing level has been switched off. But once the spectral fit is active, the program will always run through the spectral computations (even if the filters are inactive) because there happen arrays transfers.
Notice : the "configurations 927" (corresponding to `LFPSPEC=.TRUE.`) are normally performed only for η levels outputs.
The figure 2.1.7 on page 27 shows the ingredients of the subroutine `STEPO` used for the post-processing.

WRHFP, WRSFP (WRite Horizontal fullPos, WRite Spectral FullPos) : To write out post-processed data. These subroutines are embedded inside `IOPACK` but are specific to *FullPos* .

ESPFP : to precondition some spectral fields before writing them out. This subroutine is specific to ALADIN in *FullPos* : it is used in the case of plane geometry to go from the reduced variables to the geographical ones.

FPEZO2H : to perform the biperiodicization of the fields if needed (see `GRIDFPOS` above).

Sequence	Function	Possible values
(1)	Outputs	"E" : P levels "U" : z levels "Y" : P_v levels "M" : θ levels "Z" : η levels
(2:3)	Inverse spectral transforms	"A" : on model fields "P" : on post-processing fields "0" : No inverse transforms
(4)	Not used	"0"
(5)	Gridpoint computations	"B" : vertical interpolations on P levels "V" : vertical interpolations on P_v levels "T" : vertical interpolations on θ levels "H" : vertical interpolations on z levels (above model surface) "S" : vertical interpolations on η levels (above model surface) "Z" : vertical interpolations on z levels (above output surface) "E" : vertical interpolations on η levels (above output surface) "M" : computation of the model fields on the model levels "A" : horizontal interpolations of vertically post-processed fields "G" : horizontal interpolations of auxilary surface fields "P" : horizontal interpolations of physical fields "I" : horizontal interpolations of model fields
(6)	Not used	"0"
(7:8)	Direct spectral transforms	"P" : on post-processing fields "0" : No inverse transforms
(9)	Spectral computations	"P" : active "0" : not active

Table 2.1: *STEPO configuration letters used for the post-processing*

	Gridpoint	Spectral (`CFPFMT='MODEL'`)	"927" (`LFPSPEC=.TRUE.`)

P	`. A A 0 B 0 P P P`
	`0 P P 0 A 0 0 0 0`
	`E . . . . . . . .`
		`. A A 0 B 0 P P P`
		`E . . . . . . . .`
			`. A A 0 M 0 0 0 0`
			`0 0 0 0 A 0 0 0 0`
			`Z 0 0 0 0 0 0 0 0`
			`0 A A 0 B 0 P P P`
			`E 0 0 0 0 0 0 0 0`

P_v	`. A A 0 V 0 P P P`
	`0 P P 0 A 0 0 0 0`
	`Y . . . . . . . .`
		`. A A 0 V 0 P P P`
		`Y . . . . . . . .`
			`. A A 0 M 0 0 0 0`
			`0 0 0 0 A 0 0 0 0`
			`Z 0 0 0 0 0 0 0 0`
			`0 A A 0 V 0 P P P`
			`Y 0 0 0 0 0 0 0 0`

θ	`. A A 0 T 0 P P P`
	`0 P P 0 A 0 0 0 0`
	`M . . . . . . . .`
		`. A A 0 T 0 P P P`
		`M . . . . . . . .`
			`. A A 0 M 0 0 0 0`
			`0 0 0 0 A 0 0 0 0`
			`Z 0 0 0 0 0 0 0 0`
			`0 A A 0 T 0 P P P`
			`M 0 0 0 0 0 0 0 0`

z	`. A A 0 M 0 0 0 0`
	`0 0 0 0 I 0 0 0 0`
	`0 0 0 0 Z 0 0 0 0`
	`U . . . . . . . .`
		`. A A 0 H 0 P P P`
		`U . . . . . . . .`
			`. A A 0 M 0 0 0 0`
			`0 0 0 0 A 0 0 0 0`
			`Z 0 0 0 0 0 0 0 0`
			`0 A A 0 H 0 P P P`
			`U 0 0 0 0 0 0 0 0`

η	`. A A 0 M 0 0 0 0`
	`0 0 0 0 I 0 0 0 0`
	`0 0 0 0 E 0 0 0 0`
	`Z . . . . . . . .`
		`. A A 0 S 0 P P P`
		`Z . . . . . . . .`
			`. A A 0 M 0 0 0 0`
			`0 0 0 0 A 0 0 0 0`
			`Z 0 0 0 0 0 0 0 0`
			`0 A A 0 Z 0 P P P`
			`Z 0 0 0 0 0 0 0 0`

Table 2.2: *Possible combinations of STEPO sequences*

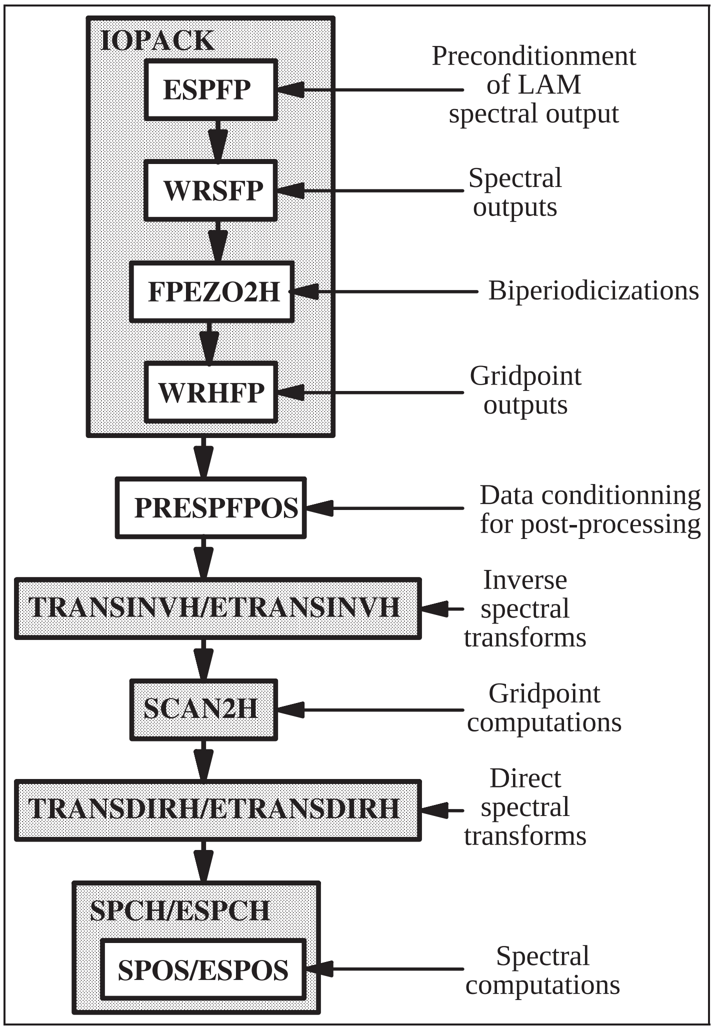


Figure 2.10: *Elements of STEPO used for the post-processing. The greyish areas correspond to model subroutines.*

PRESPFPOS (PREpare SPectral FullPOS) : to pack the spectral data and the surface fields before, and restore them after, in order to enable the in-line/off-line reproductibility of the post-processing. Like the subroutine `PREGPFPOS`, it should rather be considered as a model-specific subroutine.

TRANSINVH/ETRANSINVH : Inverse spectral transforms. Note that the data can be either the model trajectory or a post-processing flow (externalization to be prospected).

SCAN2H : the model gridpoint head subroutine is invoked with a specific configuration string (see `GRIDFPOS` above).

TRANSDIRH/ETRANSDIRH : Direct spectral transforms. They are performed on post-processing data exclusively. Post-processing-specific subroutines are embedded inside.

SPOS/ESPOS : (Spectral POSt-processing) : to perform computation in spectral space for the post-processing. These routines are not only used to filter fields, but also for other operations. They are embedded inside the model subroutines `SPCH` and `ESPCH`.

2.1.8 Gridpoint calculations

This section will describe the mechanism of post-processing computations in gridpoint space, which is embedded inside the model subroutines `SCAN2H` and `SCAN2MDM`. In the scope of the externalization of *FullPos* , it will be necessary to leave these interfaces.

Horizontal interpolations

Following the mechanism of "semi-lagrangian buffers" at the time of the shared memory architecture, the horizontal interpolations have been conceived as a succession of two subroutines, respectively :

HPOS (Horizontal POSt-processing) : to copy the fields to be interpolated in a "core-array"

HPOSLAG : (Horizontal POSt-processing, LAGged part) : to interpolate the fields (the suffix "LAG" recalls the former shared memory architecture where the call to this subroutine had to be synchronized with the corresponding call to `HPOS`).

The figure 2.1.8 on page 28 describes this general mechanism.

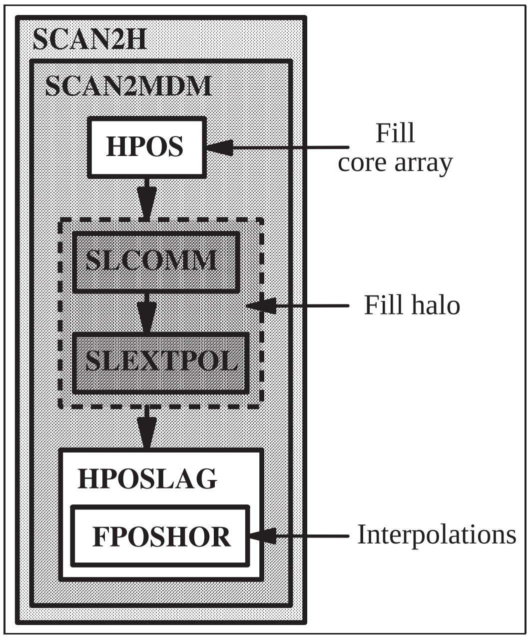


Figure 2.11: *General mechanism of horizontal interpolations. The greyish areas correspond to model subroutines.*

Between these two subroutines, the "core-array" is surrounded by a "halo" of data from the neighbouring processors. This part is common with the model. In the scope of the externalization of *FullPos* , it is supposed to be externalized as well ; it could be the opportunity to write a specific interpolator, well-adapted to an irregular string of output points.

Today `HPOSLAG` has become a useless interface which calls a main subroutine for the management of horizontal interpolations : `FPOSHOR` (FullPOS HORizontal).
Notice : in a new modular framework, the general organization should be composed of three successive subroutines :

1. `HPOS` : to fill the core array

2. `HPOSLAG` : to fill the halo

3. `FPOSHOR` : to interpolate.

The subroutine `FPOSHOR`, as shown on the figure 2.1.8 on page 29, contains the following elements :

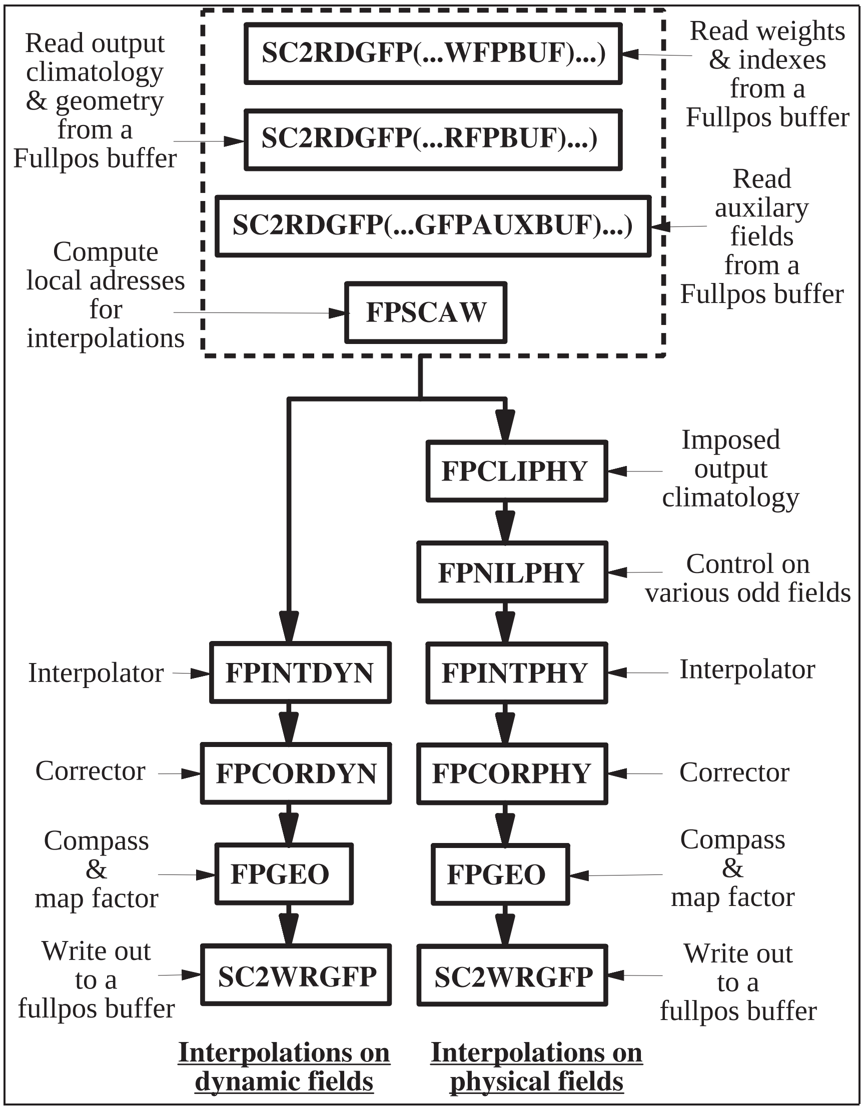


Figure 2.12: *Horizontal interpolations management : FPOSHOR.*

SC2RDGFP : To extract data from a "fullpos buffer". Three of them are used here :

1. The weights and indexes for interpolations (refer to `SUWFPDS` and `SUWFPBUF`).
2. The output climatology and geometry (refer to `SURFPDS` and `SURFPBUF`).
3. The auxilary (pre-interpolated) surface fields (refer to `SUFPTR2` and `SCAN2H('G')`).

FPSCAW : To compute the exact adresses of the neighbouring input points for interpolations.

FPINTDYN/FPINTPHY : Fields basic interpolators, respectively for dynamics and physics.

FPCORDYN/FPCORPHY : Fields basic correctors after interpolations, respectively for dynamics and physics.

FPGEO : To convert wind-related fields to the output compass and map factor.

SC2WRGFP : To write out the interpolated data in a "fullpos buffer".

FPCLIPHY : To use the output climatology rather than interpolate (for the appropriate fields only !)

FPNILPHY : A strange subroutine which controls that the remaining fields are proper for interpolations, and which perform the approriate computations for those which should not be interpolated.

Remark : to enable the use of all this mechanism for the pre-interpolation of the model land-sea mask (instead of duplicating the code — refer to `CPCLIMI` —), the buffer containing the weights and indexes should be split : one for the land-sea-dependent fields and one for the "standard" fields.

Vertical interpolations, re-ajustements and physico-dynamic calculations

The vertical interpolations, which are performed on the model grid, are simply controlled by a post-processing-specific interface named `VPOS` ; as shown on the figure 2.1.8 on page 31, it contains the following main subroutines :

POS : To perform vertical interpolations.

PHYMFPOS (PHYsical Meteo-France POStprocessing) : To compute physico-dynamic fields on the model grid (used only if the output grid is the model grid).

Notice :

- The conditions of calls to `POS` and `PHYMFPOS` are unproper : they should be limited to the occurence of any field subject to vertical interpolations for `POS`, and to the occurence of any field subject to physico-dynamic computations on the model grid for `PHYMFPOS`. `QFPTYPE` to be extended in this sense ?
- The interface between `VPOS` and the model has become odd : while the physical fields are read inside `VPOS`, the dynamic ones are obtained via the subroutine interface. This should be harmonized (all the model data to be read inside).

The vertical re-ajustements, which are performed on the output grids (because they have to be computed after the horizontal interpolations of the model primitive fields), are simply controlled by a post-processing-specific interface named `ENDVPOS` ; as shown on the figure 2.1.8 on page 31, it contains the following subroutines :

SC2RDGFP : To extract data from a "fullpos buffer". Three of them are used here :

1. The horizontally interpolated model dynamic fields.
2. The output climatology and geometry (refer to `SURFPDS` and `SURFPBUF`).
3. The auxilary (pre-interpolated) surface fields (refer to `SUFPTR2` and `SCAN2H('G')`).

ENDPOS (END POSt-processing) : To perform vertical re-ajustements and physico-dynamic calculations

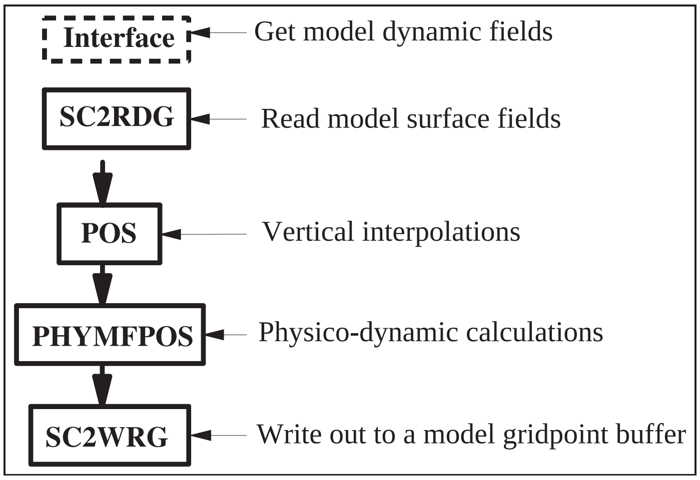


Figure 2.13: *Vertical interpolations and physico-dynamic calculations on input grid : VPOS.*

SC2WRGFP : To write out the interpolated data in a ”fullpos buffer”.

Notice :

- In `ENDVPOS` the loop on subrows should be put outside the subroutine to make easier a further distribution (OPEN-MP).
- `POS`/`PHYMFPOS` on one side, and `ENDPOS` on the other side are similar. In the scope of a further harmonization of the code, one should investigate how to re-organize all these subroutines.

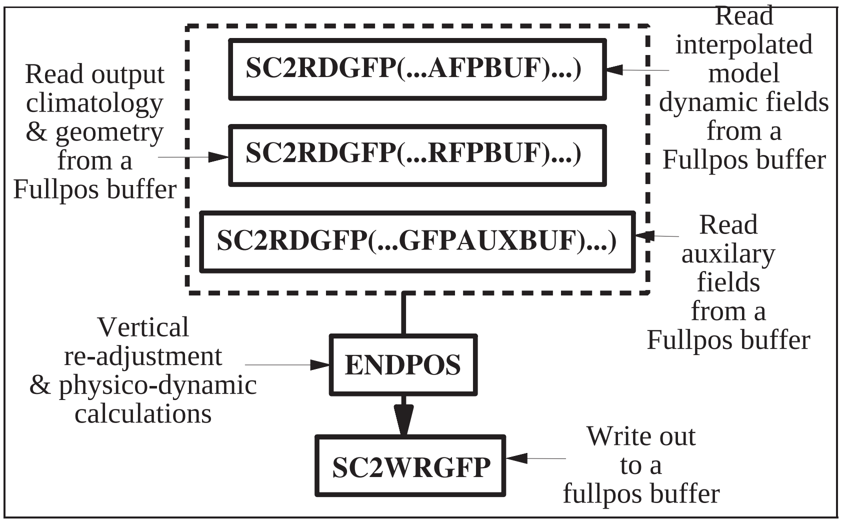


Figure 2.14: *Re-ajustements and physico-dynamic calculations on output grids : ENDVPOS.*

Biperiodicization

The biperiodicization is performed in the same spirit as the horizontal interpolations : somehow the ”interpolations” are replaced by ”extrapolations”.
The control subroutine, as seen before, is named `FPEZO2H`. The figure 2.1.8 on page 32 describes its mechanism.

EPOS (Extension zone POSt-processing) : To fill a "fullpos buffer" with fields to biperiodicize. This subroutine is the counterpart for `HPOS`.

ENDEPOS (END Extension zone POSt-processing) : To correct the extended fields afer the biperiodicization. This subroutine is the counterpart of a part of `FPOSHOR`.

FPEZONE (FullPos Extension ZONE) : To perform the biperiodicization itself. This subroutine has the specificity that its distribution is odd : since the calculation itself is not (yet ?) distributed, the distribution is performed on the global fields.

Notice :

- `FPEZO2M` is nothing but an interface to control the former multitasking system. It should be removed now.
- `FPEZONE` is supposed to be externalized, or more exactly : its main subroutine `FPBIPER`. This externalization should naturally fit the spectral transforms ... and their distribution ! Therefore the distribution of `FPBIPER` should be investigated : then *FullPos* would be adapted.
- For the clarity of the code, it would be nice to have a straight symetry between the interpolations and the biperiodicization ! But this depends of the biperiodicization distribution.

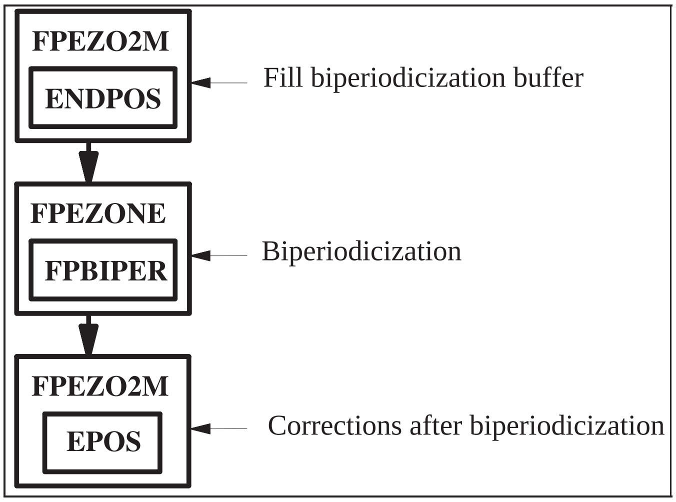


Figure 2.15: *Biperiodicization : FPEZO2H.*

2.2 Data flow

This section will describe the principal data arrays used in *FullPos* .

2.2.1 Spectral arrays

FullPos is (indirectly) using the model spectral arrays (`SPA2` and `SPA3`) as the input of the model inverse spectral transforms. It uses also specific spectral arrays which will contains the spectrally fitted fields to be filtered or written out to files. As *FullPos* uses the `tfl`/`tal` spectral transforms packages, these arrays are all shaped and distributed in the same manner.
Open question : in the scope of the externalization of *FullPos* , should we consider that the

model inverse transforms is an external subroutine used inside the post-processing package ? This could be a solution to improve the portability of the externalized *FullPos* in other applications (with 100% gridpoint input data for instance).

There are three specific spectral arrays :

SPAFP : it contains the vertically post-processed fields to be spectrally fitted which are not "derivative fields" to be filtered in the homogeneous high resolution space. `SPAFP` is output from `TRANSDIRH` and input to `SPOS/ESPOS`.

SPDFP : it contains the vertically post-processed fields to be spectrally fitted which are "derivative fields" to be filtered in the homogeneous high resolution space. `SPDFP` is output from `TRANSDIRH` and input to `SPOS/ESPOS`.

SPBFP : it contains the vertically post-processed spectral fields which are "derivative fields" filtered in the homogeneous high resolution space. In this array, for each field, there is one spectrum per field and per subdomain. `SPBFP` is output from `SPOS/ESPOS` and input to `TRANSINVH`.

(See the figure 2.2.1 on page 33).

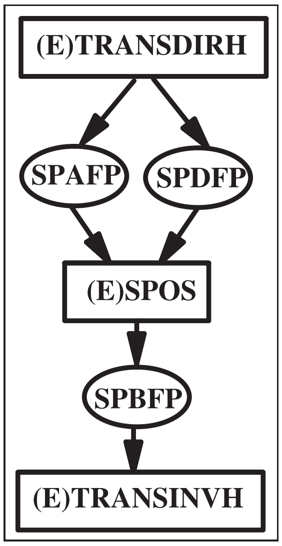


Figure 2.16: *Spectral post-processing data flow.*

Remarks :

- Though there were no reasons to do it, the spectral data flow in ALADIN has been split like in ARPEGE to stick to the same code structure.
- `SPAFP` and `SPDFP` had been split for technical reasons, after an old conception. In the next cycle (26), all these arrays have been unified in one single array `SPBFP`, thanks to a more clever monitoring of the post-processed fields.

2.2.2 Gridpoint buffers

By "gridpoint buffers" one should understand the model gridpoint buffers, that is those which are shaped to fit the model geometry and its distribution.

These buffers (or arrays) are used to interface :

- the model and the post-processing in general (`GPPBUF`, `GFUBUF`, `XFUBUF` for the physical fields ; `GPP` and `GPUABUF` for the dynamic fields ; various geographical data arrays from the module `YOMGC`),

- the vertical post-processing and the spectral transforms (`GPP` is re-used as output from `VPOS`),

- the inverse spectral transforms and the horizontal post-processing (`GPP` is used as output from the inverse spectral transforms and input to `HPOS`),

- the vertical post-processing and the horizontal interpolations (`GAUXBUF` used for the fields which are not concerned by the spectral fit).

The figure 2.2.2 on page 34 gives more informations about the relations between the main subroutines and these buffers/arrays. Note that the model array GPP is re-used as input/output for *FullP*os : this is embarassing for the externalization.

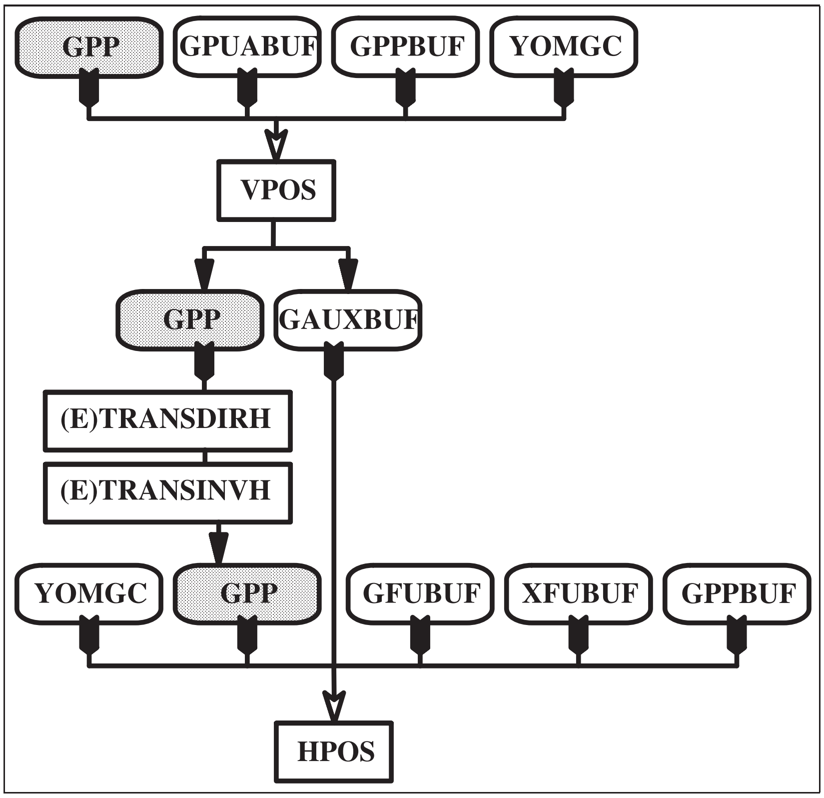


Figure 2.17: *Gridpoint buffers interactions.*

2.2.3 Fullpos buffers

By "Fullpos buffers" we shall point out the gridpoint buffers specially designed for the post-processing output.

How are they shaped ?
The figure 2.2.3 on page 35 helps to understand the problem :
Given the model gridpoint area and its distribution, it appears that from one processor to another, the number of "output points" (ie : the points where to the model fields will be interpolated) can

be quite different. This is even more obvious when the input model is stretched and the target grid is not, or when the output grid is centered on the pole of interest of the input model (which is typically the case of the ARPEGE/ALALDIN coupling).

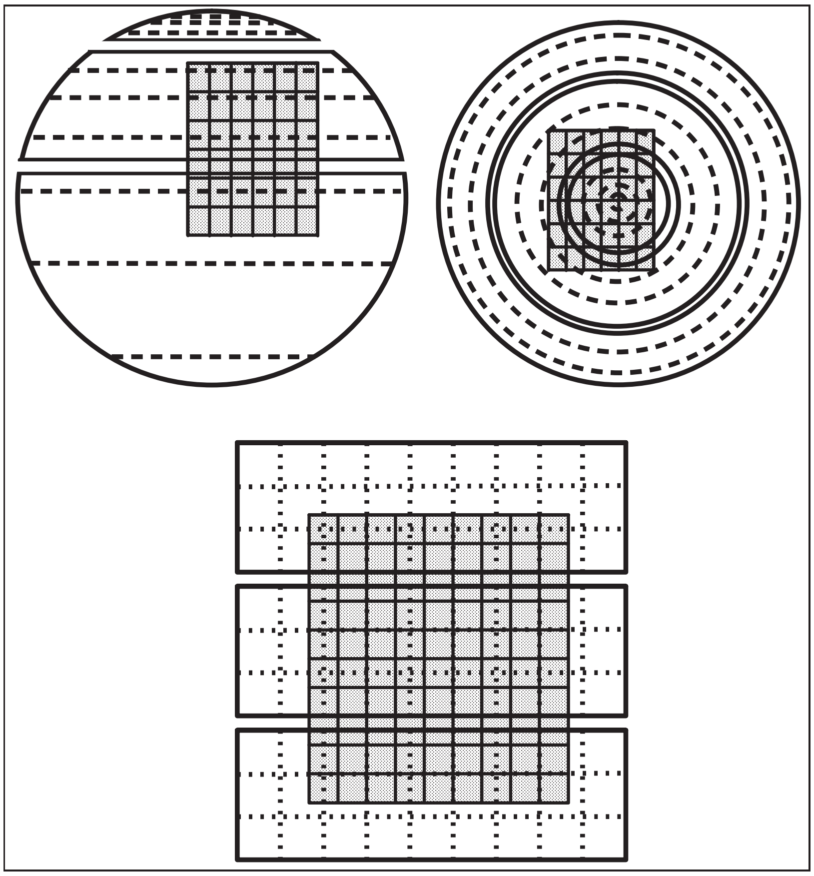


Figure 2.18: *Gridpoints repartitions betwen model and output grids : the greyish areas corresonds to an ouput grid ; the background areas corresponds to the input model, with a physical separation between processors. Top left represents a typical coupling from a streched grid, Top right a typical coupling from a rotated grid, bottom a typical nesting.*

At the time of the shared memory code and the I/O scheme, this was really a complication because to a model "row" (with a fixed size `NPROMA`) we had to affect a post-processing "row" of points with a variable size. To solve this problem we had to consider that the post-processing packets were composed of subrows of fixed size `NFPROMAG`.
Things have been much easier to concieve since the removal of the I/O scheme and the new distributed architecture : the "Fullpos buffer" is very close to the model "Gridpoint buffer", since only the characteristics have different numerics : the size is `NFPRGPL` instead of `NGPTOT` ; the segmentation is `NFPROMAG` instead of `NPROMA` (however, this last distinction has been purely formal since the distributed memory architecture, while it was of prime importance for the parallelism in the shared memory architecture).
Remarks :

- The code is still containing a lot of items which were used for the shared memory architecture and which have become completely useless today (cleaning to be done !).
- "Fullpos buffers" and "Gridpoint buffers" should remain formal different structures to make

the externalization of *FullPos* easier. "Fullpos buffers" should even be able to setup "Gridpoint buffers"-like structures on demand, in order to help the externalization (so that buffers like `GAUXBUF`, which are internal to *FullPos* , would not need the model structure). Another possibility would be to create an external common structure for the model and the post-processing (to be archived in `xrd` library ?).

- "Fullpos buffers" are also already used to handle the biperiodicization, with different characteristics : `NFPEL` instead of `NFPRGPL` and `NFPROMEL` instead of `NFPROMAG`.
- Distribution on horizontally interpolated data may be better balanced. But how to do it is a big deal ! The solution would be to distribute as equally as possible the geographically sorted string of post-processing points, and then to adapt the construction of the cores and halos. This leads to a re-distribution of the model gridpoints after the vertical interpolations and before the horizontal ones, and thus to a post-processing-specific horizontal interpolator system (good for externalization !) : see the figure 2.2.3 on page 36.

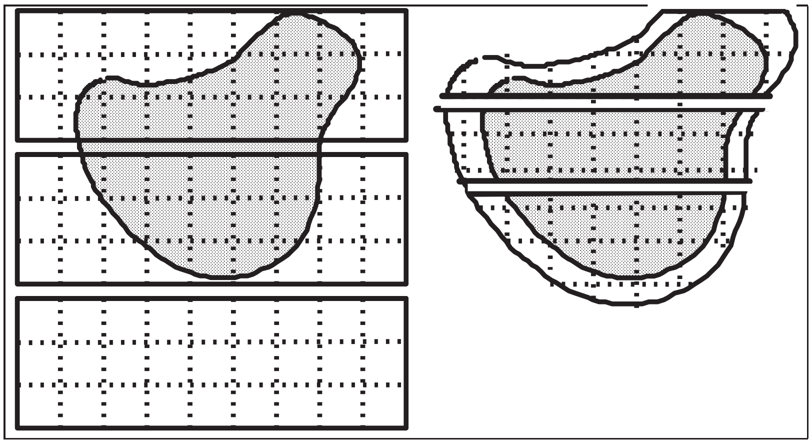


Figure 2.19: *"Next generation" fullpos horizontal interpolator : the cores and halos are re-built to fit the post-processing distribution.*

As shown in the figure 2.2.3 on page 37, these buffers are used to interface the vertical interpolations (`FPOSHPOR`, according to its configuration), the vertical re-adjustment and physico-dynamic calculations on the target grids (`ENDVPOS`), and finally the writing out to files (`WRHFP`).

The used buffers are :

WFPBUF : weights and indexes for interpolations

RFPBUF : output climatology and geometry

GFPAUXBUF : auxilary surface fields

GPFPBUF : output physical fields

GDFPBUF : output dynamic fields

AFPBUF : horizontally interpolated model fields

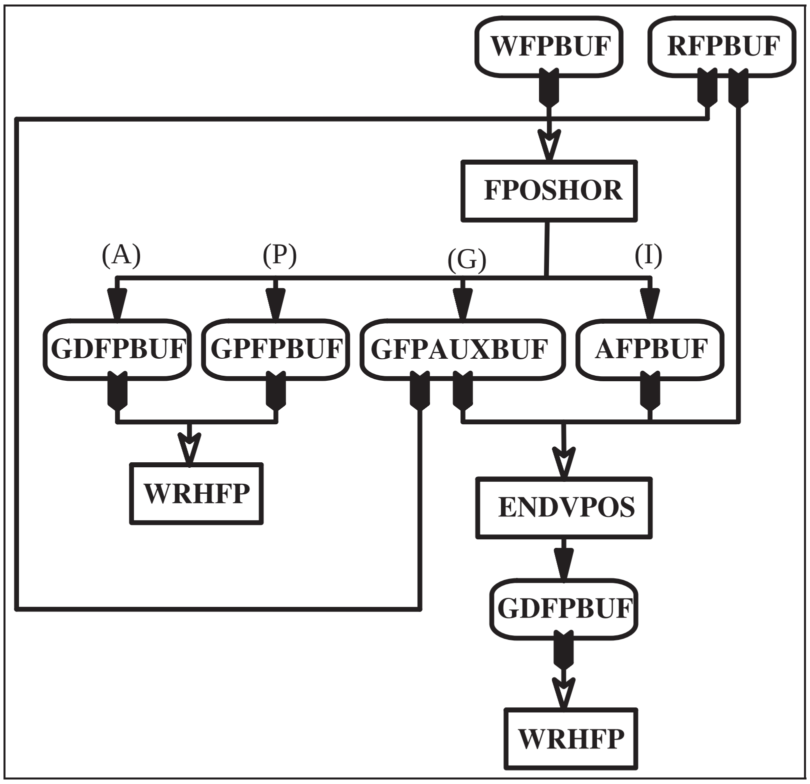


Figure 2.20: *Fullpos buffers interactions in the horizontal interpolations.*

Notice : GPFPBUF and GDFPBUF are never used simultaneously ; so they should be merged in a single buffer (FFPBUF like "Final FullPos BUFfer" ?).

As shown in the figure 2.2.3 on page 37, these buffers are also used to interface the biperiodicizations ; the used buffer, named EZOBUF, contains either the extensions zone of the physical fields or the extensions zones of the dynamic fields (according to the configuration of FPEZO2H).

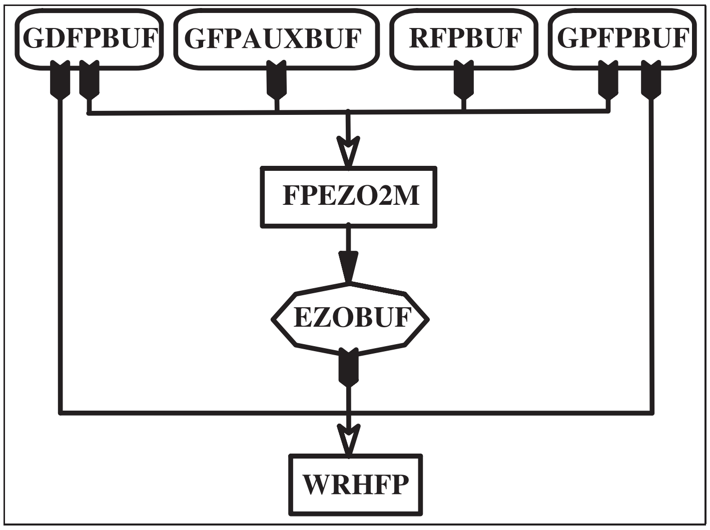


Figure 2.21: *Fullpos buffers interactions in the biperiodicizations.*

2.3 Monitoring

This section will describe how the post-processed fields are monitored through the software.

2.3.1 Physical fields

The monitoring of the physical fields is rather easy, since there are only horizontal interpolations (at least for the time being, since the upper air fluxes are not yet post-processable). However it has become old-fashionned with regards to the Fortran 90 facilities, and it is heavy to use. This monitoring should be revisited to make the maintenance easier and to enable the treatment of upper air fluxes.

The characteristics are the following :

- The fields are stored in the following order :

  1. Surface fields (from `GPPBUF`)
  2. Cumulated fluxes (from `GFUBUF`)
  3. Instantaneous fluxes (from `XFUBUF`)

  This rule should be followed through the whole code in order to locate the fields.

- The fields are characterized by :

  - The ARPEGE/ALADIN field name.
  - The number of bits for packing before writing out to file.

  The setup is performed in `YOMAFN`, `NAMAFN`, `SUAFN1`, `SUAFN2` and `SUAFN3` (Note that the tables numering is fragile). Then the fields will be always recognized by their ARPEGE/ALADIN name.

The steps to be followed in order to add a new field are then the next ones :

- For a given post-processing field, all the needed model fields should be at disposal : `FPINIPHY`/`SUFPCFU`/`SUFPXFU` according to the field kind.

- Each field is associated to one or more model fields to fill the interpolations buffer : `HPOS`.

- Each field is associated to one or more interpolated fields to fill the biperiodicization buffer : `FPFILLB`.

- Interpolations and control of interpolations may occurs in the following subroutines : `FPINTPHY`, `FPNILPHY`, `FPCLIPHY`, `FPCORPHY`, `FPGEO`.

2.3.2 Dynamic fields

The monitoring of the dynamic fields now uses complex derived types from the Fortran 90 language. The aim is to setup all the characteristics wich will be used to monitor the fields, so that the fields could be handled in an almost blind way.

The monitoring is realized with two kinds of specific derived types (both declared in the module file fullpos_descriptors.F90)

1. **fullpos_descriptor** :
   a static derived type, describing the fixed characteristics through the post-processing for all the fields which can be requested. This type is set in `YOMAFN`, `NAMAFN`, `SUAFN1`, `SUAFN2` and `SUAFN3`.
   Notice : the item %ILMOD is set in `SUFPDIM` ; this item, together with %LLGP and %IBIT are a bit odd since they depend on the model namelist (to be revisited). This type is partly external, as it is partly accessible from the namelist `NAMAFN`.
   There is one type for each field, and one type array containing all of them (named : `TFP_DYNDS`). The fields are then recognized by their rank in this type array (it plays the role of an internal code).

2. **fullpos_request** :
   a *dynamic* derived type named `QFPTYPE`, describing the changing characteristics of each field along the post-processing. This type is set in `CPVPOSPR`, `SUVPOS` and `UPDVPOS` (in cycle 26 : only in `CPVPOSPR` and `SUVPOS`). It is then used in a large amount of subroutines. It is a purely internal variable type.
   The purpose of `SUVPOS` is to set the fields characteristics, while `CPVPOSPR` initializes the pointer of each field in each space. This last routine is crucial to the dataflow between the gridpoint space and the spectral space : it enables the gridpoint fields to be stored in the proper order for the spectral transforms.

The steps to be followed in order to add a new field are then the next ones :

- To add a new "fullpos_descriptor"-type for the new field and set it in `YOMAFN`, `NAMAFN`, `SUAFN1`, `SUAFN2` and `SUAFN3`.
- To possibly modify `SUFPDIM` if the field should be considered as a new pronostic field.
- To compute the field in `POS`/`ENDPOS` using the "fullpos_request"-type `QFPTYPE` and the individual "fullpos_descriptor"-types.
- To possibly control the horizontal interpolation of this field in `FPCORDYN`, using the individual "fullpos_descriptor"-types.