AGRIF

Adaptive **G**rid **R**efinement **I**n **F**ortran

User's Guide

Version 1.3

27 novembre 2006

AGRIF is a package for the integration of adaptive mesh refinement (AMR) features within a multidimensional model written in Fortran and discretized on a structured grid.

The package mainly consists in two parts. It first provides model-independent Fortran 90 procedures implementing the usual parts of an AMR process like time integration algorithm of the grid hierarchy, clustering algorithm, refinement algorithm, interpolation and update procedures, ... In a second part, the Fortran model-dependent code is created via a program which analyses a user's written configuration file.
The model independent part is compiled alone. The model dependent part is then compiled with the current model. Then they are linked.

The first chapter deals with files which could be write in order to use AGRIF. The second one is based on the current model modification and also the way to use to compile the model and AGRIF.

Contents

Chapter 1

Description of AGRIF

To use AGRIF, the user needs

- to write a configuration file which contains model's description. This file is read by the CONV during the code compilation,
- A file where fixed grids are defined,
- The file agrif_user.F90 which contains users subroutines,
- The file agrif2model.F90, which make the link between the model and the agrif library (this file should not be modify).

All those steps are detailed below.

1.1 The configuration file

The syntax used in the configuration file is presented next. All examples given here have no real signification; practicals examples are presented in the tutorial.

1.1.1 Global parameters

Refinement dimension and domain size

1. Description

   The refinement dimension and the domain size are set with the keyword `nD` followed by one, two or three integer values corresponding to the number of **cells** in the directions of refinement.

Note that those integers must be global variables which must be defined in the parameter file and probably declared as parameters in the original model.

2. Syntax

```
nD n1[,n2[,n3]];  / n ∈ 1,2,3, n1,n2,n3 : strings /
```

3. Examples

```
3D nx,ny,nz;  / 3D refinement /
2D nx,ny;  / 2D refinement (the model can be 2D or 3D) /
```

Parameter file

1. Description

The parameter file is the usual model's file in which several parameters are declared, as the indexes delimiting the domain for example. Actually, the essential condition is that this file must contain the declaration of the cells number.
If it is an included file, its name is given after the keyword `paramfile`. If these variables are defined in a module, we should use the keyword `parammodule`.

2. Syntax

```
paramfile name;  / name : string of characters /
parammodule name;  / name : string of characters /
```

3. Example

```
paramfile parameter.h;
parammodule modulename;
```

Using fixed grids

1. Description

Keyword `FIXED_GRIDS` indicates that some fixed grids are used.

2. Syntax

```
USE FIXED_GRIDS;
```

3. Example

```
USE FIXED_GRIDS;
```

Using only fixed grids

1. Description

   Keyword `ONLY_FIXED_GRIDS` indicates that only some fixed grids are used, without moving grids. They are defined in the Agrif_Fixedgrids.in

2. Syntax

```
USE ONLY_FIXED_GRIDS;
```

3. Example

```
USE ONLY_FIXED_GRIDS;
```

not grid dependent variables

1. Description

   Keyword `notgriddep` indicates that a variable is not grid dependent. Consequently, these variable value will be the same on each grids and will not stored in the table *tabvars*.

2. Syntax

```
notgriddep name;  / name : string of characters /
```

3. Example

```
notgriddep variable_name;
```

1.2 agrif_user file

Severals subroutines should be defined in this file.

- Agrif_InitWorkspace
- Agrif_InitValues
- Agrif_detect

Those subroutines are defined below.

1.2.1 Agrif_InitWorkspace subroutine

In this subroutine, the user should define all variables which are computed from the grid dimension on the fine grid.
For example

```
if ( .NOT. Agrif_Root() ) then
   nx1=nx+1
   ny1=ny+1
endif
```

If the amr has been defined in the configuration file, all AMR parameters should be defined in this subroutine :

- Efficiency

  1. Description The efficiency is relative to the clustering algorithm and represents the proportion of grid points included in the newly created points which were really detected. The default efficiency is set to 70%.
     The user can change this value the subroutine `Agrif_Set_Efficiency`.
  2. Example

     ```
     Call Agrif_Set_Efficiency(0.8)
     ```

- Regridding

  1. Description The regridding operation is performed periodically, every N coarse grid time steps. This number N is set by the subroutine `Agrif_Set_Regridding`.
  2. Example

     ```
     Call Agrif_Set_Regridding(10)
     ```

- Space refinement factors

  1. Description The space refinement factors for new grids created by adaptive mesh refinement are set by subroutines `Agrif_Set_coeffref_x`, `Agrif_Set_coeffref_y` and `Agrif_Set_coeffref_z`.
  2. Syntax

     `Call Agrif_Set_coeffref_x(3)` / n : integer / `Call Agrif_Set_coeff`

- Time refinement factors

1. Description The time refinement factors for new grids created by adaptive mesh refinement are set by using subroutines `Agrif_Set_coeffreft_x`, `Agrif_Set_coeffreft_y` and `Agrif_Set_coeffreft_z`.
2. Example

```
Call Agrif_Set_coeffreft_x(2)
Call Agrif_Set_coeffreft_y(2)
Call Agrif_Set_coeffreft_z(2)
```

Minimum width

1. Description The minimum width is the minimum size of the grids (in number of cells) created during the regridding operation. This number is set with the subroutine `Agrif_Set_minwidth`.
2. Example

```
Call Agrif_Set_minwidth(5)
```

- Maximum number of refinements

1. Description The maximum number of refinement is defined for all space direction by using subroutines `Agrif_Set_rafmax`.
2. Example

```
Call Agrif_Set_rafmax(5)
```

1.2.2 Agrif_InitValues subroutine

This subroutine is called once on each fine grid.

Firstly, we should call the subroutine which allows to initialize the current grid (the same subroutine which allows to initialized the coarse grid).

```
Call init()
```

If some **specific treatments** of one variable are required (initialization, boundary interpolation, restoration), this one has to be "fully declared". This means that the type of the variable, the indices of the first point in the domain, and the type of its dimensions must be declared.
We should call following subroutines :

- Position of the variable

  1. Description We should specify the position of the variable on the grid, by calling the subroutine `Agrif_Set_type`. That allows the use of staggered grids. For each space direction, we use 1 for the boarder of a cell (figure 1.1) and 2 for the center of the cell (figure 1.2). Remember that a staggered grid variable can be used in a space direction only if this direction is refined.

Figure 1.1: Positions of a grid variable on a no staggered variable

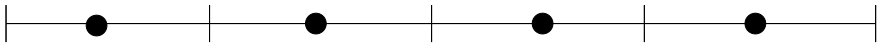

Figure 1.2: Positions of a grid variable on a staggered variable

Next, it is necessary to declare the index of the first point of the variable which is inside the computational domain.
At last, the user must indicate the type of each dimension by using the subroutine **Agrif_Set_raf** : x,y or z for a space dimension, 0 for a no space dimension.

  2. Syntax

```
Call Agrif_Set_type(u,(/p1,p2/),(/i1,i2/)) / p1,p2,p3,i1,i2,i3 : intege
Call Agrif_Set_raf(u,(/'n1','n2'/)) / n1,n2,n3 : x,y,z or N /
```

(p1,p2,p3 indicate the position of the grid variable and i1,i2,i3 the index of the first point in the domain).

  3. Example

For a 2D domain, four different ways for positionning a variable on a cell are possible.

```
Call Agrif_Set_type(vr,(/1,1/),(/1,1/))
Call Agrif_Set_type(u,(/2,1/),(/1,1/))
Call Agrif_Set_type(v,(/1,2/),(/1,1/))
Call Agrif_Set_type(h,(/2,2/),(/1,1/))
Call Agrif_Set_Raf(u,(/'y','N'/))
```

Figure (1.3) shows these positions on a cell.

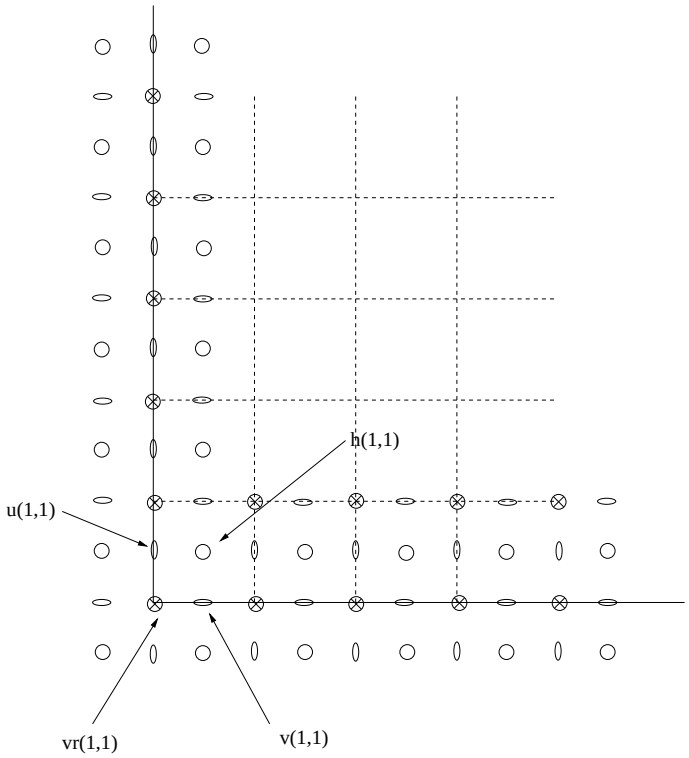

Figure 1.3: First points in the domain

- Interpolation

  1. Description AGRIF can interpolate variables of the current grid which are declared with the subroutine `Agrif_Set_interp` in this subroutine.

  2. Syntax

     `Call Agrif_Set_interp(name,interp=method)` / method of interpolation

  3. Example

     ```
     Call Agrif_Set_interp(u,interp=Agrif_linear)
     ```

- Boundary interpolation

  1. Description AGRIF can, in a similar way, provide subroutines for the interpolation of boundary values if necessary. The name of the concerned variables must be given after with the subroutine `Agrif_Set_bc`, followed by a description such as (ideb:ifin) or (iind), where these last indices indicate where to interpolate.
     For variables which have been also declared by with the subroutine `Agrif_Set_interp`, the default interpolation type used in the

boundary corrections is the same that the one used in the `Agrif_Set_interp` called.
If the variable is not declared by using the subroutine `Agrif_Set_interp` or if another kind of interpolation for the boundary is desired, the subroutine `Agrif_Set_bcinterp` should be called.

2. Syntax

```
Call Agrif_Set_bc(var,(/ideb:ifin/))   / ideb,ifin : integers /
Call Agrif_Set_bcinterp(var,method)    / method = linear, lagrange, spl
```

Figure (1.4) represents (in 2D and for the different types of variables) the positions which will be corrected if you specify (-1).
Figure (1.5) represents (in 2D and for the different types of variables) the positions which will be corrected if you specify (0).
Figure (1.6) represents (in 2D and for the different types of variables) the positions which will be corrected if you specify (1).

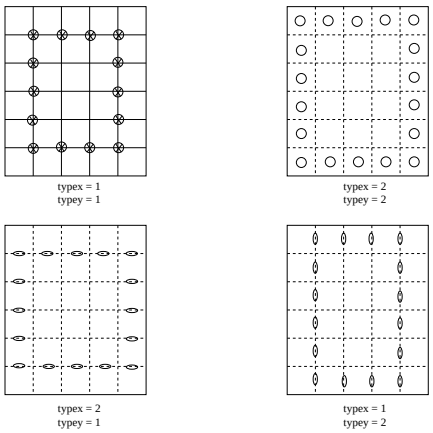

Figure 1.4: Grid corrections, BC :: var : (-1)

- Update of the grid variables The AGRIF package is able to update grid variables from their parent grid.

1. Description AGRIF can update the variables of the parent grid of the current grid which are declared with the subroutine `Agrif_Set_UpdateType` .

2. Syntax

```
Call Agrif_Set_UpdateType(var,update=method) / method of interpolati
```

3. Example

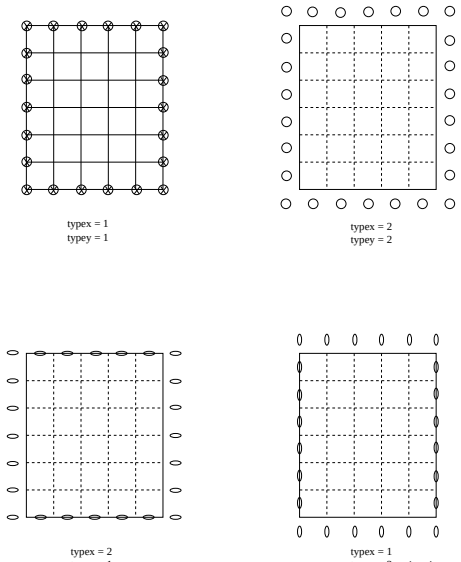

Figure 1.5: Grid corrections, BC :: var : (0)

```
Call Agrif_Set_UpdateType(u,update=Agrif_Update_Average)
```

- Restoration (adaptive mesh refinement only)

  1. Description During a step of mesh refinement and for each variable, the user can specify that its value must be restored (if possible) for the initialization of the newly created grids. This is accomplished by calling the subroutine `Agrif_set_restore`.
  2. Syntax

     ```
     Call Agrif_set_restore(var);
     ```

  3. Example

     ```
     Call Agrif_set_restore(u)   Call Agrif_set_restore(v)
     ```

1.2.3 Agrif_detect subroutine

This subroutine is used in order to detect points where raffinement is required.
This subroutine should be write when an adaptative mesh raffinement is necessary.

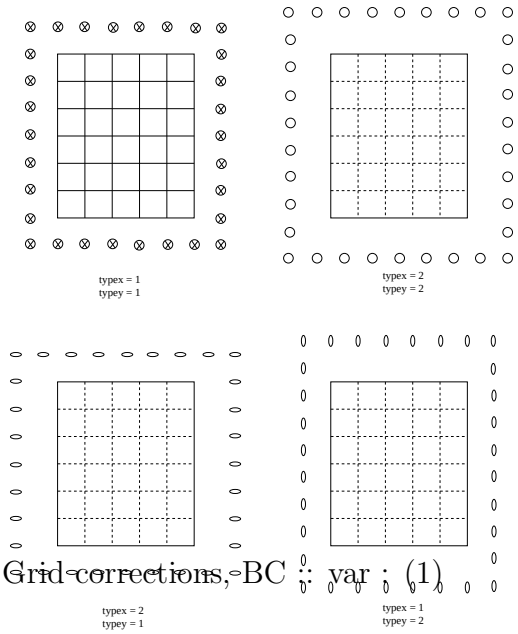

Figure 1.6: Grid corrections, BC : var ; (1)

1.3 Agrif2Model file

The CONV program allows to create a new fortran code and also files (in AGRIF_INC directory) which are used to complete the agrif2model file.
So, in the Makefile we should compile the agrif2model file at the end of the compilation just after the compilation of the last model file.

1.4 Special Operations

1.4.1 Special values with interpolations

Before a call to the Agrif_Interp_VarName or Agrif_Bc_VarName subroutines, you can specify the use of some special values for the arrays of the parent and fine grids concerned by these interpolations.
The necessary keywords are `Agrif_UseSpecialValue` and `Agrif_SpecialValue` for the parent grids, `Agrif_UseSpecialValueFineGrid` and `Agrif_SpecialValueFineGr` for the fine grids.
Agrif_UseSpecialValue and Agrif_UseSpecialValueFineGrid are two **logicals** whereas Agrif_SpecialValue and Agrif_SpecialValueFineGrid are two **reals**.

Special values on the parent grids

If you want to use a special value for the parent grid (for example 98.8 here), you must add the following lines before any call to an Agrif_Interp_VarName

or Agrif_Bc_VarName subroutine :

```
AGRIF_UseSpecialValue = .TRUE.
AGRIF_SpecialValue = 98.8
```

Then, if a value on the part of the parent grid used for the space interpolation of the fine grid is equal to 98.8, this value will be replaced, if possible, by the nearest value of the parent grid different of 98.8.

Special values on the fine grids

If you want to use a special value for the fine grid (for example 50.0 here), you must add the following lines before any call to an Agrif_Interp_VarName or Agrif_Bc_VarName subroutine :

```
AGRIF_UseSpecialValueFineGrid = .TRUE.
AGRIF_SpecialValueFineGrid = 50.0
```

Then, if a value on the fine grid is equal to 50.0, this value won't be replaced during the interpolation.

1.4.2 Getting the values on the parent grid

Sometimes, the user need to know the values on the parent grid of the current grid (to write its own routines of interpolation and update for example). AGRIF offers this possibility.

How to get the values on the parent grid ?

In order to get the value on the parent grid we should call the function Agrif_Parent.

Example : `Agrif_Parent(u)`

- For a scalar variable :

  Real :: dtp
  dtp = Agrif_Parent(dt)

- For a multidimensionnal variable :

  Real, Dimension(:,:), pointer :: parent_u

parent_u ⇒ Agrif_Parent(u)
parent_u(i,j) = 3.

What is the correspondance between points on fine and parent grids ?

For a no staggered variable, a point on a parent grid always corresponds with a point on the child grid, whatever the space refinement factors are; the same holds for a staggered variable with an odd space refinement factor. Concerning a staggered variable with an even space refinement factor, this correspondence does not exist as it is shown on the figure (1.7).

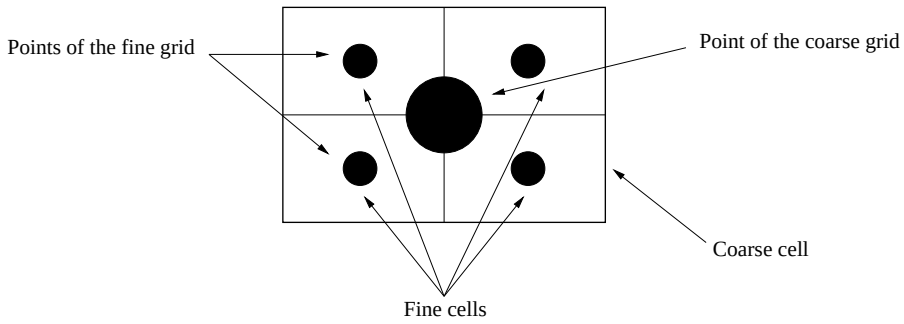

Figure 1.7: Staggered variable with a space refinement factor equal to 2

The correspondances between the fine and coarsed grids points are given by the following algorithm :

Do i = $ideb_f, \ldots, coeffraf$
$Var_G(ideb_g + (\frac{i-ideb_f}{coeffraf})) = Var_f(i)$
End Do

where the values of $ideb_f$ and $ideb_g$ depend on the type of the variable (staggered or no staggered) :

	$ideb_f$	$ideb_g$
No staggered variable	ptx	$ix + ptx - 1$
Staggered variable	$ptx + \frac{coeffraf-1}{2}$	$ix + ptx - 1$

where ptx is the index of the first point inside the domain and ix the minimum position of the current grid on its parent grid.

1.4.3 Using fixed grids in AGRIF

One of the features of the AGRIF package is its ability to deal with the use of fixed fine grids in the domain.

First, the user must insert the keyword `USE FIXED_GRIDS` in the configuration file; he must add the keyword `USE ONLY_FIXED_GRIDS` if he want to use only fixed grids and no moving grids.

Next, he must declared the fine grids in a file called ***AGRIF_FixedGrids.in*** which has to be in the current directory where the code runs.
For each fine grid and each space direction, a line indicates its position on its parent grid and its space and time refinement factors.

Syntax

- 1D
  `imin imax spacerefx timerefx`

- 2D
  `imin imax jmin jmax spacerefx spacerefy timerefx timerefy`

- 3D
  `imin imax jmin jmax kmin kmax spacerefx spacerefy spcerefz timerefx timerefy timerefz`

Imin,imax,jmin,jmax,kmin,kmax are six integers indicating the minimum and maximum positions of the fixed grids; in this definition, the positions of the corner points refer to the positions **of the parent grid corners**, starting with the **value 1** in each space direction. When the user writes these positions, he has to take care that each fine grid is contained in a parent fixed grid.

Spacerefx, spacerefy, spcerefz, timerefx, timerefy and timerefz are six integers indicating space and time refinement factors in the x,y and z directions.
If one of these factors are equal to 1, it means there is no refinement in the concerned space direction.

Example

Figure (1.8) gives an example in dimension 2. The root coarse grid (G_0) has two child grids G_1 and G_2, and G_1 has one child grid G_3. Space and

time refinement factors are equal to 2 in both space directions.

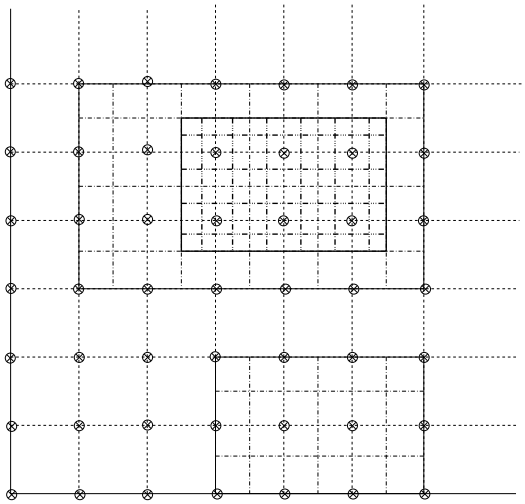

Figure 1.8: Fixed grids example

For this example, the *AGRIF_FixedGrids.in* file must contain the following lines:

```
2                                          (G0 has 2 child grids : G1 and G2)
2    7    4    7    2    2    2    2 (positions and refinement factors fo
4    7    1    3    2    2    2    2 (positions and refinement factors fo
1                                          (G1 has 1 child grid : G3)
4   10    2    6    2    2    2    2 (positions and refinement factors fo
0                                          (G3 has no child grid)
0                                          (G2 has no child grid)
```

Fixed grids with adaptive mesh refinement

When the keyword `USE FIXED_GRIDS` is in the configuration file without the keyword `USE ONLY_FIXED_GRIDS`, you have at the same time in the grid hierarchy fixed grids and moving grids.

Chapter 2

Using AGRIF in a multidimensional finite difference model

2.1 Necessary subroutines to run AGRIF

2.1.1 Subroutines provided by AGRIF

Agrif_Init_Grids

This routine creates the characteristics of the root coaser grid (number of cells, ...). It is called in the main file after the initialization of the number of cells on the coarser grid.

Agrif_Step

This routine calls and replaces the user's time integration procedure (named **Step**) the user has to write. It integrates the model forward on all grids and performs a regridding at regular interval (in case of adaptive refinement).

2.1.2 Subroutines to be written by the user

Agrif_Initvalues

This routine allows to initialize the fine grid variables. It is called in the procedure of refinement.

Agrif_Initworkspace

This routine contains, for each space direction, the links between variables declared in the parameter file and representing the size of the domain and the number of cells.
For example, if `nx` is the number of cells and `nxp1` the number of grid points, then the sentence `nxp1 = nx + 1` must be written in this routine.

`Agrif_Initworkspace` is called after each change of grid to recalculate the corresponding values.

Agrif_Detect

This routine is necessary if AGRIF is used in adaptative mode.
It allows to write the adaptative mesh raffinement criterion.

2.2 How to compile AGRIF (*and your model with AGRIF*) ?

2.2.1 Changes in the Makefile

The Makefile should be modify in order to use AGRIF library

Create a AGRIF key

In order to determine if AGRIF is used or not in the current model, we should created an AGRIF key. Let's say key_agrif.

```
CCP_FGLAGS = -Dkey_agrif
```

This key allows to :

- Compile or remove the AGRIF source code during the preprocessing. For example in the source code :

```
#if defined key_agrif
   code for AGRIF
#else
   code without agrif
#endif
```

- Determine in the Makefile if AGRIF should be used

```
ifneq (,$(findstring key_agrif,$(CPPFLAGS)))
    Makefile with AGRIF
else
    Makefile without AGRIF
endif
```

Compile the AGRIF library

The AGRIF directory should be place in code tree.
In the makefile we should begin with AGRIF library compilation in order to create a libagrif.a file which could be used later. To compile it, the user should use the Makefile presents in AGRIF.

Use CONV

Add the compilation of the file Agrif_User.F90 at the end of the object list. Now we should create a source code compatible with AGRIF using the CONV program.
Consequently, for each file compilation :

- we should move this file in an other directory in order to work on the file without removing the original one.
- apply the conv on this file which allows to create a new fortran file.
- compile this new file

For example, let's use a directory call NEW_FILES. Foe each file compilation we should have :

```
$(CPP) $(CPPFLAGS) $(*).F90 > NEW_FILES/$(*).F90
(cd NEW_FILES; ./conv agrif.in -comdirin ./ -comdirout AGRIF_MODELFILES -
$(CPP) $(CPPFLAGS) -INEW_FILE/AGRIF_INC NEW_FILE/AGRIF_MODELFILES/$(*).F9
$(F_C) NEW_FILE/$(*).F90
```

Compile Agrif2Model file

Just before the final link, the user should complete the Agrif2Model file with the help of files presents in AGRIF_INC directory by preprocessing.

2.2.2 Changes in the model

This section describes the few changes the user has to make in his original code to be able to run the AGRIF package.

Modifying the parameter file

This file must contain the domain sizes and variables that depend on those sizes. For example if you had in your original common, a declaration like

```
        integer nx,ny,nz
parameter (nx=100,ny=100,nz=30)
        integer,parameter :: nxp1=nx+1,nyp1=ny+1,nzp1=nz+1
```

change it into

```
integer nx=100,ny=100,nz=30
integer nxp1=nx+1,nyp1=ny+1,nzp1=nz+1
```

Modifying the main program

For running AGRIF, you should replace the call to your original time integration procedure by **'call AGRIF_STEP'** (see section 2.1.1).
You should also add a call to the `Agrif_Init_Grids` (see section 2.1.1) subroutine. This call must be done after the initialization of the number of cells.

Writing some procedures

You should write the Agrif_Initvalues and Agrif_Initworkspace procedures (see section 2.1.2).

2.2.3 How to use the AGRIF package

After making the changes described in the previous section, the user has to write the configuration file (see section 1.1) and to compile the library.

Writing the AGRIF_FixedGrids.in file

This file (described in section 1.4.3) is necessary if you want to use some fixed grids. It must be positioned in the main directory where the model runs.

Writing the configuration file

Here, we try to present how the user could process to get an idea of what he should put in the configuration file :

1. count the numbers of cells in the X,Y and Z directions

2. get the parameter file name