AGRIF

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

Tutorial

Version 1.3

27 novembre 2006

In this tutorial, we introduce the implementation of the adaptive mesh refinement method in two different finite difference models by using the AGRIF package.
The first model, a 2D shallow water model, is discretized on a C-grid.
After having briefly recalled the models equations and their discretizations, we present the implementation of the AMR with the AGRIF package.

Contents

Chapter 1

Guidelines for the use of AGRIF

1.1 Is AGRIF the right tool for your model ?

There are both technical and general reasons to use or not to use AGRIF to do adaptive mesh refinement.

1.1.1 Existing models

The main idea of AGRIF is to minimize the changes to be made in an existing model in order to integrate mesh refinement potentialities. So by nature it is mostly designed for already existing large codes.
If you begin a new project and plan to make extensive use of adaptive mesh refinement methods, you may prefer the use of a more complete package like the ones written in C++ (see for example HAMR or DAGH), or even packages that integrates the solution of partial differential equations (see for example Overture or CLAWPACK).

1.1.2 Continue to develop as before

With the use of AGRIF, users not familiar with mesh refinement can continue to develop the code as before.

1.1.3 Models written in Fortran

Of course, your model need to be written in Fortran.

1.2 Guidelines

1. The computational domain :
   One important part is to identify your computational domain. Often in

a numerical model, you have ghost cells near the boundary in order to easy implement the boundary conditions (so the question here is where the model is computing its boundary conditions). The computational domain (as seen by AGRIF) is then all the interior remaining cells.

(a) Deduce the number of cells
The number of cells in each direction of refinement of your computational domain must be a variable (even if sometimes it is already a variable of your code). In the configuration file, the corresponding variables must be indicated. These variables will be automatically changed by AGRIF during the computation so they can not be declared as parameters, they have to be declare das global global.
Concerns : **parameter file, AGRIF configuration file**

(b) Deduce all the numbers depending of the number of cells
The number of cells will change during the model integration when skipping from one grid to another. That is why all the variables that explicitely depend on these numbers must be known. The (often very few) lines that makes this dependancy must be coded in the Agrif_Initworkspace subroutine.
Concerns : **parameter file, Agrif_Initworkspace, Agrif_User.F90**

2. Isolate code corresponding to one time step (if not already done)
During the integration of the grid hierarchy, AGRIF will need to make call to a subroutine that integrate the model for one time step in time. This subroutine probably already exists in your model. If it is not the case, you should have a loop that makes directly a call to the different subroutines. Just put these calls in a special subroutine.
Note : In the current version of AGRIF, this subroutine should have no arguments.
Concerns : **main program, step program**

3. Run the code with only the root grid
At this point you should be able to run the code with only the root grid. This can be a good test as the results must not changed.
Don't forget to initialize the grid hierarchy (by a call to Agrif_Init_Grids()). The only thing to be initialized before this call is the number of cells on the root grid. (It does not matter if you call your old step routine or Agrif_Step()).
Concerns : **main program**

4. Begin with fixed grids

   (a) tell AGRIF that you are doing fixed mesh refinement.
   Several keywords have to be defined in the configuration file

   - (space and time) refinement factors
   - keyword USE ONLY_FIXED_GRIDS

   Create your 'Agrif_Fixed_Grids.in' file.
   Concerns : **AGRIF configuration file**

   (b) Grid's Initialization (and grid's outputs)
   These part is really dependent on your model and it may be the part taking most of the time in your integration of AGRIF (or any other mesh refinement tool). You will probably have the choose between

   - analytical initialization
     typically here, you will have to recompute the coordinates of your domain. This computation will probably use the Agrif_ix() and Agrif_iy() functions.
   - initialization from files
     The problem here is that you will probably like to have different file names for the different fixed grids of a grid hierarchy. (Don't forget you can use the function Agrif_Is_Fixed() to known if a grid is fixed or not). A good idea might be to concatenate to a generic name the number of the current grid (this is given by Agrif_Cfixed() witch returns a character string).

5. Boundary interpolations
   Find the parts of your code where the boundary conditions are needed. For each variable to be interpolated at the boundary, use the subroutine Agrif_Set_bc in the file Agrif_User.F90. You will probably make use of the function Agrif_Root() to differentiate boundary conditions on the root and the refined grids.

6. One way interaction
   It is known time to try one way interaction. It should always be the first step, since going directly to two way will most of the time hide some numerical problems that are best to be solved.

7. Two way fixed grid refinement
   Write your update subroutines or call the AGRIF ones.

8. Adaptive Mesh refinement

- If you like to do adaptive mesh refinement, the keyword USE ONLY_FIXED_GRIDS should be removed. (Replace it by USE FIXED_GRIDS if you like to keep some fixed grids in the domain)
- specify the frequency of refinement, the maximum number of refinement, the minimum size of the grids and the efficiency of the clustering algorithm in the subroutine Agrif_Initvalues
- write your own error detection routine (the AGRIF_detect routine)

Chapter 2

A shallow water model

2.1 Introduction

This chapter deals with an example of the coupling between a shallow water model and AGRIF. This example can be download in the AGRIF web site under the name sw_agrif1.3.tar.gz.
In order to run this example. The file sw_agrif1.3.tar.gz should be unzip. In the directory sw_agrif/AGRIFSWFIXED there is a Makefile written in order to compile this example with ifort compiler.
To create the executable, just type **make**, and execute ./ipxqg.

2.2 The model

The following model describes a 2D shallow water problem. The velocity and the surface elevation are the grid variables. Moreover, a mask is applied in each of these variables allowing to represent the land (where the velocity is null and the surface elevation constant).
In this example, the computationnal domain size is 2000 by 2000 km, the cell size is 25 by 25 km (i.e. 80 cells in each space direction) on the coarse grid.
The time step is equal to 1800 secondes on the root grid.

The model equations are :

$$\left\{\begin{array}{rcl} \dfrac{\partial u}{\partial t} & = & (\xi+f)\,v - \dfrac{\partial B}{\partial x} + \dfrac{\tau_x}{\rho_0 h} + A\Delta u - ru \\ \dfrac{\partial v}{\partial t} & = & -(\xi+f)\,u - \dfrac{\partial B}{\partial y} + A\Delta v - rv \\ \dfrac{\partial h}{\partial t} & = & -\dfrac{\partial}{\partial x}(hu) - \dfrac{\partial}{\partial y}(hv) \end{array}\right.$$

where the state variables are u and v (horizontal velocities) and h (surface elevation). $B = \frac{1}{2}(u^2+v^2)$ is the Bernoulli function, $\xi = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$ the relative vorticity, $f = 2\Omega \sin\Psi$ the coriolis parameter. A is the laplacian diffusivity coefficient and r the bottom friction.

The model is discretized with centered second order finite difference schemes in space and a leap frog time scheme. The equations are discretized on an Arakawa C-grid. Figure (2.1) shows the mesh and the position of the variables on the C-grid.

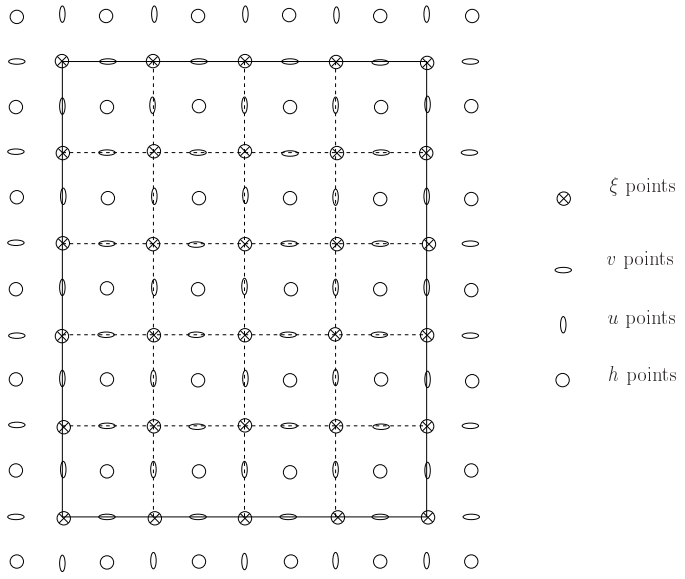

Figure 2.1: The shallow water model grid

2.3 The parameter file (parameter.F90)

We define by nx and ny the number of cells in the X and Y directions (eg on figure (2.1), nx=4 and ny=5); $nxp1 = nx + 1$ and $nyp1 = ny + 1$ are the numbers of grid points.

```
module parametre

integer nx = 30
integer nxp1 = nx + 1
integer ny = 30
integer nyp1 = ny + 1
real dtt,ds2

real A,r,rho0,g,beta,f0,tau0,h0,gamma
integer isauv = 5000
integer noslip

REAL, PARAMETER ::  L= 2e6
INTEGER, PARAMETER ::  itdeb = 1
INTEGER, PARAMETER ::  itfin = 100000

end module parametre
```

2.4 The common file (commodel.F90)

The state variables are declared as follow :

$$\begin{cases} u(1:nxp1, 0:nyp1) & \text{horizontal velocity, X-direction} \\ v(0:nxp1, 1:nyp1) & \text{horizontal velocity, Y-direction} \\ h(0:nxp1, 0:nyp1) & \text{surface elevation} \end{cases}$$

The additional work arrays are:

$$\begin{cases} vr(1:nxp1, 1:nyp1) & \text{relative vorticity} \\ fv(2:ny) & \text{coriolis parameter, u-points} \\ fu(1:ny) & \text{coriolis parameter, v-points} \\ taux(1:ny) & \text{wind stress, X-direction} \end{cases}$$

The arrays (um, uc, up), (vm, vc, vp), (hm, hc, hp) are declared in the same way as u, v and h for the time integration scheme.

Finally, the corresponding file is :

```
      module commodel
      USE parametre
      real dt,ds

      real up(1:nxp1,0:nyp1),u(1:nxp1,0:nyp1),
     &         um(1:nxp1,0:nyp1),uc(1:nxp1,0:nyp1)
      real vp(0:nxp1,1:nyp1),v(0:nxp1,1:nyp1),
     &         vm(0:nxp1,1:nyp1),vc(0:nxp1,1:nyp1)
      real hp(0:nxp1,0:nyp1),h(0:nxp1,0:nyp1),
     &         hm(0:nxp1,0:nyp1),hc(0:nxp1,0:nyp1)
      real fu(1:ny),fv(2:ny),taux(1:ny)
      real vr(1:nxp1,1:nyp1)
      integer istok

      real y0
      integer nbstep

      end module commodel
```

2.5 Writing the configuration file called agrif_sw.in

The user needs several informations before writing the configuration file :

1. the dimension of the problem is 2 (2D)
2. the number of cells in the X and Y directions are nx and ny

The configuration file corresponding to these informations is the following :

```
% Number of cells in each direction %
 2D nx,ny;

USE ONLY_FIXED_GRIDS;
```

2.6 Writing the user AGRIF routines

Agrif_Initworkspace

A few lines must be added in the routine *Initworkspace* called at each change of grid. It concerns the relations between the grid size (number of cells) and the number of grid points. For the shallow water model, those lines are:

```
Subroutine Agrif_InitWorkspace()

      Use AGRIF_Util
      USE parametre
      USE commodel
      Implicit none
!
!
      if ( .NOT. Agrif_Root() ) then
         dtt = 2. * dt
         ds2 = ds * ds
         nxp1 = nx + 1
         nyp1 = ny + 1
      else
         nx = 30
 ny = 30
         nxp1 = nx + 1
 nyp1 = ny + 1
 isauv = 5000
      endif
!
      Return


End Subroutine Agrif_InitWorkspace
```

Agrif_Initvalues

This routine allows to initialize the fine grids and to calculate their space and time refinement factors (ds and dt).

```
Subroutine Agrif_Initvalues
```

```
      Use AGRIF_UTIL
      USE parametre
      USE commodel
      Implicit none
!
      REAL hptemp(0:nxp1,0:nyp1)
      REAL htemp(0:nxp1,0:nyp1)
      REAL vtemp(0:nxp1,1:nyp1)
      REAL vptemp(0:nxp1,1:nyp1)
      REAL utemp(1:nxp1,0:nyp1)
      REAL uptemp(1:nxp1,0:nyp1)
!
!     Calculation of the space and time steps for the fine grids
!
      Call Agrif_Set_type(up,(/1,2,0/),(/1,1,0/))
      Call Agrif_Set_type(u ,(/1,2,0/),(/1,1,0/))
      Call Agrif_Set_type(vp,(/2,1,0/),(/1,1,0/))
      Call Agrif_Set_type(v ,(/2,1,0/),(/1,1,0/))
      Call Agrif_Set_type(hp,(/2,2,0/),(/1,1,0/))
      Call Agrif_Set_type(h ,(/2,2,0/),(/1,1,0/))
!
!     Space directions for each variables
!
      Call Agrif_Set_raf(up,(/'x','y','N'/))
      Call Agrif_Set_raf(u ,(/'x','y','N'/))
      Call Agrif_Set_raf(vp,(/'x','y','N'/))
      Call Agrif_Set_raf(v ,(/'x','y','N'/))
      Call Agrif_Set_raf(hp,(/'x','y','N'/))
      Call Agrif_Set_raf(h ,(/'x','y','N'/))
!
!     type of interpolation
!
      Call Agrif_Set_interp(up,interp=AGRIF_linear)
      Call Agrif_Set_interp(u ,interp=AGRIF_linear)
      Call Agrif_Set_interp(vp,interp=AGRIF_linear)
      Call Agrif_Set_interp(v ,interp=AGRIF_linear)
      Call Agrif_Set_interp(hp,interp=AGRIF_linear)
      Call Agrif_Set_interp(h ,interp=AGRIF_linear)
```

```
!
      Call Agrif_Set_bcinterp(up,interp=AGRIF_linear)
      Call Agrif_Set_bcinterp(u ,interp=AGRIF_linear)
      Call Agrif_Set_bcinterp(vp,interp=AGRIF_linear)
      Call Agrif_Set_bcinterp(v ,interp=AGRIF_linear)
      Call Agrif_Set_bcinterp(hp,interp=AGRIF_linear)
      Call Agrif_Set_bcinterp(h ,interp=AGRIF_linear)
!
!     Location of interpolation
!
      Call Agrif_Set_bc(up,(/0,0/))
      Call Agrif_Set_bc(u ,(/0,0/))
      Call Agrif_Set_bc(vp,(/0,0/))
      Call Agrif_Set_bc(v ,(/0,0/))
      Call Agrif_Set_bc(hp,(/0,0/))
      Call Agrif_Set_bc(h ,(/0,0/))
!
!     Calculation of the masks on the fine grids
!
      call Initgrille()
!
      Call AGRIF_Init_variable(u,u)
      Call AGRIF_Init_variable(v,v)
      Call AGRIF_Init_variable(h,h)
!
      Agrif_UseSpecialvalue=.true.
      Agrif_Specialvalue=0.
      Call AGRIF_BC_variable(up,up,calledweight=1.)
      Call AGRIF_BC_variable(vp,vp,calledweight=1.)
      Call AGRIF_BC_variable(hp,hp,calledweight=1.)
      Agrif_UseSpecialvalue=.false.
!
      Return
!
End Subroutine Agrif_Initvalues
```

2.7 Modifying the model

The model is composed of two files called main.f and step.f. These ones must be a few modified in order to allow the use of the AGRIF library.

The main.f file

- calling the `Agrif_Init_Grids` procedure.
- calling the `Agrif_Step` procedure.

The step.f file

- calling the `Agrif_Bc_variable` for up and vp routines allowing the calculation of the boundary conditions.