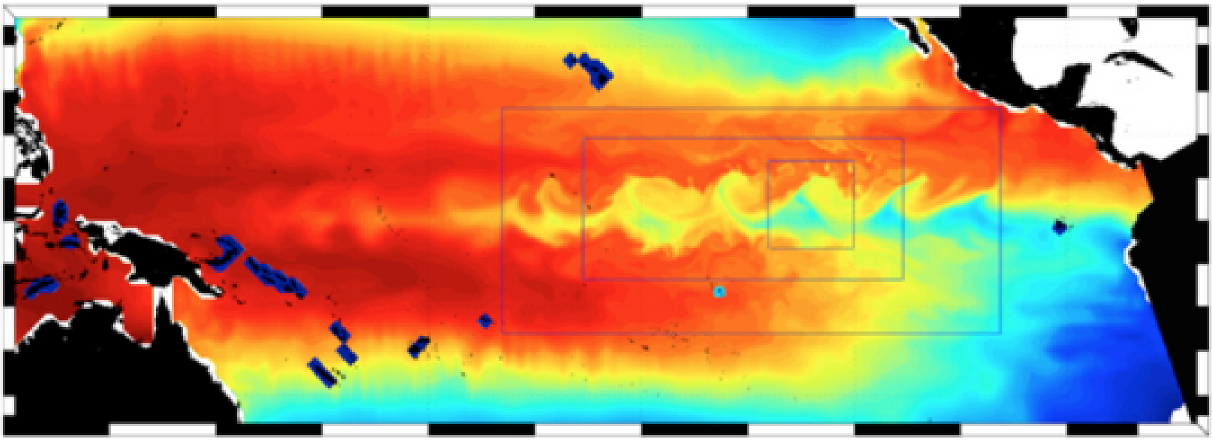
AGRIF



USER'S GUIDE

Authors:
Laurent DEBREU
Roland PATOUM

Contents

List of Figures

General introduction: AGRIF software

AGRIF software

The main idea of this software is to bring (fixed or adaptive) mesh refinement features to existing models that are written in the Fortran language and discretized on a structured grid. The software is divided in two parts:

- a source to source code converter ("CONV") that transform a unigrid code in a multigrid code
- a library that implements grids interactions.

Alternatives

- Hand coded solutions (e.g. ROMS-Rutgers)
- One code software : (e.g. RSL WRF)
- General packages : Paramesh (Fortran90); SAMRAI (C++); CHOMBO (C++)

Main contributors:

Laurent Debreu, Christophe Vouland, Cyril Mazauric, Marc Honnorat , Roland Patoum + all beta testers/occasional developers (special Thanks: R. Benshila, G. Cambon, F. Lemarie, Jean Marc Molines, Franck Vigilant)

A typical compilation organization

Let our compilation directory be organized as follows:

```
src/:
  Agrif2Model.F90    <--- glue code: should be compiled last
  Agrif_User.F90     <--- written by the user
  agrif.in           <--- config. file for 'conv'
  code.F90           <--- single-grid model
                            with Agrif directives

work/:
  AGRIF/              <--- a copy of Agrif library
  AGRIF_INC/          <--- empty directory
  AGRIF_MODEL_FILES/ <--- empty directory
```

conv configuration file

example `agrif.in` file

```
% Dimension and Number of cells in each direction %

2D nx,ny;

% Name of the common file that contains the cells variables:
% - an include file (paramfile) %
% - OR a module (parammodule) %

parammodule mod_global;

% Use only fixed grids in this example
USE ONLY_FIXED_GRIDS;
```

inside code.F90

```
module mod_global
    integer :: nx, ny
    real(8), allocatable, dimension(:,:) :: gridvar
end module mod_global
```

Chapter 1

Make the code run on different grids

1.1 Source code transformation tool: conv

- Original, 'single-grid' code must be re-written
  to be made **grid-independant**
- Source-to-source transformation tool: **conv**
- Global variables are handled internally by Agrif library

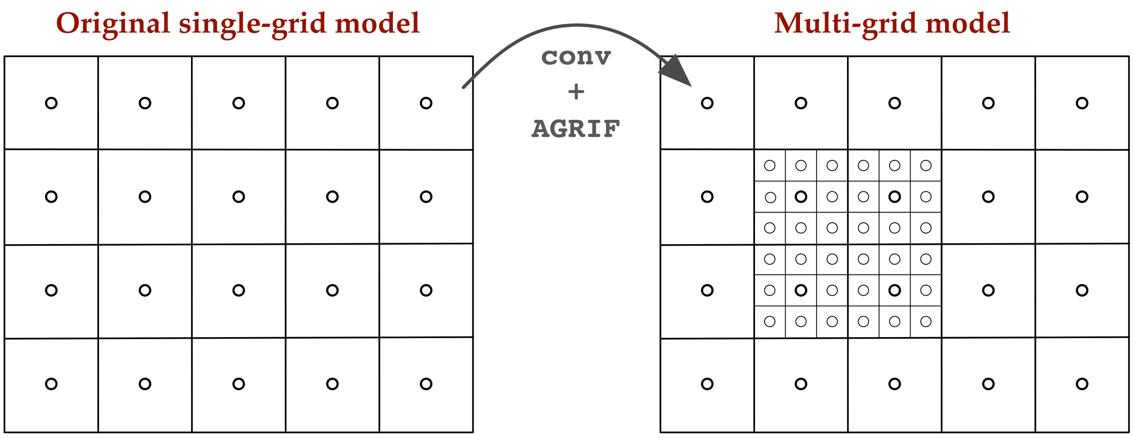


Figure 1.1: conv as transformation tool

Example : Original code

```
subroutine do_stuff ( data_in )
    use mod_global
    real(8), dimension(:,:) :: data_in
    gridvar = data_in    ! modifies global variable 'gridvar'
end subroutine do_stuff
```

- `'gridvar'` is declared global in a common or module
- `'data_in'` is function argument

Example: Conv'ed code

```
subroutine do_stuff ( data_in )
    use Agrif_Util
    use mod_global
    real(8), dimension(:,:) :: data_in
    call Sub_Loop_do_stuff(data_in, Agrif_tabvars(12) % array2)
    ! 12 ---> index of the gridvar variable
contains
    subroutine Sub_Loop_do_stuff(data_in,gridvar)
        use Agrif_Util
        real(8), dimension(:,:) :: data_in
        real(8), allocatable, dimension(:,:) :: gridvar
        gridvar = data_in   ! modifies global variable 'gridvar'
    end subroutine Sub_Loop_do_stuff
end subroutine do_stuff
```

1.2 A typical compilation organization

1. Preprocessing

This part should be done only if the original code needs to be preprocessed.

```
cpp -P -Dkey_AGRIF src/code.F90 > work/code.F90_cpp
```

2. Call to conv program

```
cd work ; ../AGRIF/conv ../src/agrif.in -rm \
     -comdirout AGRIF_MODEL_FILES \
     -convfile code.F90_cpp
```

3. Re-preprocessing

```
cd work;
cpp -P -Dkey_AGRIF -I./AGRIF_INC ./AGRIF_MODEL_FILES/code.F90_cpp > code.F90
```

4. Fortran compiltation

```
gfortran -I./AGRIF/AGRIF_OBJS -o code.o -c work/code.F90 -LAGRIF -lagrif
```

Chapter 2

General organization of an AGRIF code

Example: Main program

```
program example
    use mod_global
    integer :: i
    call Agrif_Init_Grids()
    call read_config()
    call init() ! initialize the coarse grid
    do i=1,nbsteps
        call Agrif_Step(step) ! is used to integrate the grid hierarchy which is created a
    enddo
end program example
```

2.1 Main AGRIF procedures

They have to be called :

Agrif_Init_Grids very first call in the code: initializes Agrif library.

Agrif_Step(step) integrates the model forward on all grids by calling `step()` recursively.

They can be called:

Agrif_Regrid() reads `AGRIF_Fixed_Grids.in` and build grids data structures (called anyway if not explicitly).

Agrif_Step(step) calls `step()` recursively on each grid (with time refinement).

Agrif_Step_Child(step) calls `step()` recursively on each grid (no time refinement).

Agrif_Step_Childs(step) calls `step()` recursively on each child grid of the current Parent grid (no time refinement).

Agrif_Integrate_ChildGrids(step) calls `step()` recursively on each child grid of the current Parent grid (with time refinement).

They have to be written by the user :

Agrif_InitWorkspace which defines dimensions of the current workspace. It is Called each time the library needs to change the current grid.

Example `Agrif_InitWorkspace`

```
subroutine Agrif_InitWorkspace ( )
    use mod_global
    ! compute everything that is related to array sizes
    Nnx = nx + 1
    Nny = ny + 1
end subroutine Agrif_InitWorkspace
```

Agrif_InitValues which is the initialization routine called once for each fine grid. Typically it is used to:

- declare Agrif profiles for interpolation/restriction operations
- initialize variables (eg: read data from file / interpolation from coarse grid)

Example `Agrif_InitValues`

```
subroutine Agrif_InitValues ( )
    call init()
end subroutine Agrif_InitValues
```

remark: The first call of Agrif_Step calls Agrif_Regrid and Agrif_InitValues on the whole grids and initializes all.

2.2 Integration of the grids hierarchy

There are four subroutines that involve the integration of grid hierarchy: **Agrif_Step**, **Agrif_Step_Child**, **Agrif_Step_Childs** and **Agrif_Integrate_ChildGrids**.

⇒ **Agrif_Step(step)**

It calls `step()` recursively on each grid and takes into account the time refinementfactor.

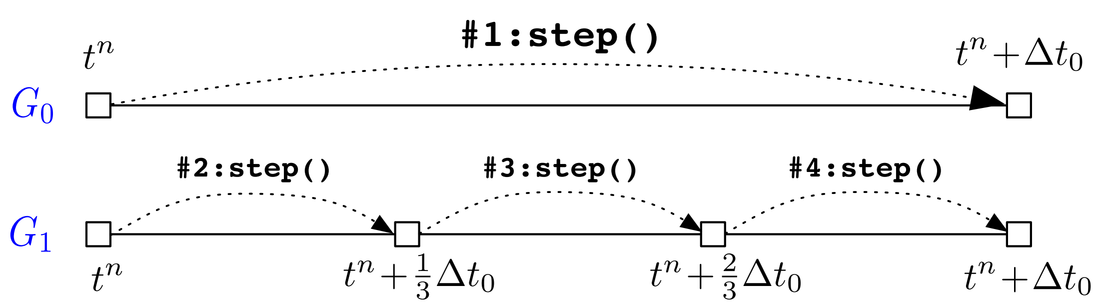


Figure 2.1: Description Agrif_Step subroutine with a root grid G0 and a child grid G1 (Time refinement factor = 3)

⇒ **Agrif_Step_Child(step)**

It calls `step()` recursively on each grid (no time refinement). This subroutine could be used instead of **Agrif_Step(step)** in order to obtain the same result. In fact, we introduce a call of **Agrif_Step_Child(step)** at the end of subroutine `step()`.

Example: subroutine step()

```
subroutine step( )
    .      .
    .      .
    .      .
    .      .

   call agrif_step_child(step)
   call agrif_update_variable(u_id,procname=Update_MyTraceur)

end subroutine step
```

⇒ **Agrif_Step_Childs(step)**

It calls `step()` recursively on each child grid of the current Parent grid (no time refinement).

⇒ **Agrif_Integrate_ChildGrids(step)**

It calls `step()` recursively on each child grid of the current Parent grid (with time refinement). The difference with **Agrif_Step(step)** is the fact that `step()` is called only on the childs grids not on the parent grid.

2.3 Auxiliary Agrif functions

Agrif_Root() indicates if the current grid is the root grid

Agrif_Fixed() returns the number of the current grid (0 for root)

Agrif_IRhox() returns the space refinement factor of the current grid

Agrif_IRhot() returns the time refinement factor of the current grid

Agrif_Nb_Step() number time steps for current grid

Agrif_Nbstepint() sub-step number inside parent grid integration

Agrif_Parent(X) gets the value of X on the parent grid where X is a scalar variable

Example: Set the time for update by using Agrif's functions

```
if ( two_way .and. (Agrif_Nbstepint() == Agrif_IRhot()-1) ) then
    call Agrif_Update_Variable(Variable_id,procname = update_MyTraceur)
endif
```

Chapter 3

The computational grid as seen by the AGRIF software

3.1 The computational grid

3.1.1 The reference grid

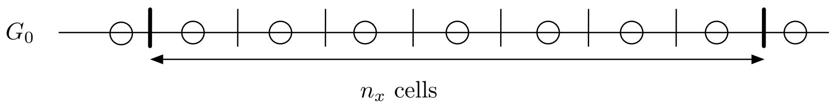


Figure 3.1: coarse grid describes with interior nx cells and two ghost cells

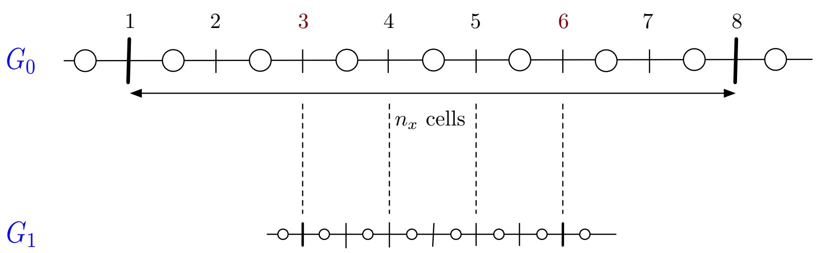


Figure 3.2: coarse grid and fine grid G1 with a refinement factor of 2

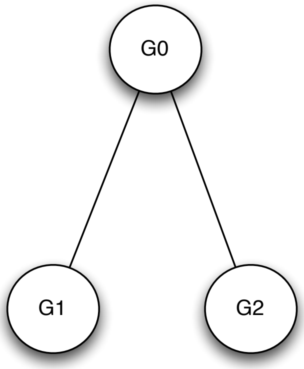
3.1.2 The AGRIF_Fixed_Grids.in file

This file is used to define grids positions, space and time refinement factors. It also gives the informations about the number of child grid for each grid and define them.

Example: The file

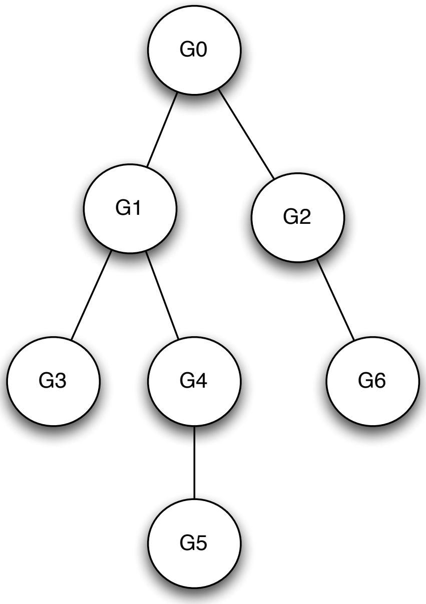
- AGRIF_Fixed_Grids.in file for the definition of one grid with no child grid in 1D:
  1
  3 6 2 2 # imin imax rhox rhot
  0
- AGRIF_Fixed_Grids.in file for the definition of one grid with no child grid in 2D:
  1
  3 6 4 8 3 2 3 # imin imax jmin jmax rhox rhoy rhot
  0

Example: Description of grid hierarchy



2
imin_1 imax_1 $\mathrm{rho}_{x,1}$ $\mathrm{rho}_{t,1}$
imin_2 imax_2 $\mathrm{rho}_{x,2}$ $\mathrm{rho}_{t,2}$
0
0

Figure 3.3: Coarse grid G0 which has 2 fine grids G1 and G2



2
imin_1 imax_1 $\mathrm{rho}_{x,1}$ $\mathrm{rho}_{t,1}$
imin_2 imax_2 $\mathrm{rho}_{x,2}$ $\mathrm{rho}_{t,2}$
2
imin_3 imax_3 $\mathrm{rho}_{x,3}$ $\mathrm{rho}_{t,3}$
imin_4 imax_4 $\mathrm{rho}_{x,4}$ $\mathrm{rho}_{t,4}$
0
1
imin_5 imax_5 $\mathrm{rho}_{x,5}$ $\mathrm{rho}_{t,5}$
0
1
imin_6 imax_6 $\mathrm{rho}_{x,6}$ $\mathrm{rho}_{t,6}$
0

Figure 3.4: Coarse grid G0 with 2 fine grids G1 and G2 which have their own fine grids

3.2 Declaration of grid variables

3.2.1 Declaration of profiles

In order to use it for interpolation/update via procnames (see after).

1. Specification of variable and location

   Agrif_Declare_Variable((/stag/),(/first_index/),(/'dim'/),(/ibegin/),(/iend/),variable_id)

variable_id	is an integer (output)
stag	1 (noncentered) or 2 (centered)
first_index	array index of the first point in the reference grid
dim	'x', 'y', 'z' or 'N'
	'N' : no refinement in that direction

2. Examples

   - <u>Centered variable</u>

Figure 3.5: Centered variable T

```
Call Agrif_Declare_Variable((/2/),(/1/),(/'x'/),(/0/),(/nx+1/),T_id)
```

   - <u>Noncentered variable</u>

Figure 3.6: Non-centered variable U

```
Call Agrif_Declare_Variable((/1/),(/0/),(/'x'/),(/0/),(/nx/),U_id)
```

<u>remark</u>: **Agrif_Declare_Variable** Has also to be called on the root grid.

3.2.2 Relative Positions on the grids

1. Index of the first coarse grid points inside the fine grid domain

   - Centered variable (T):

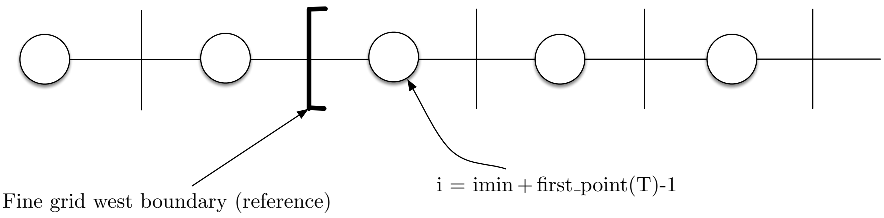


Figure 3.7: index of the first centered variable

imin is the one specified in the file **AGRIF_Fixed_Grids.in**

   - Noncentered variable (U):

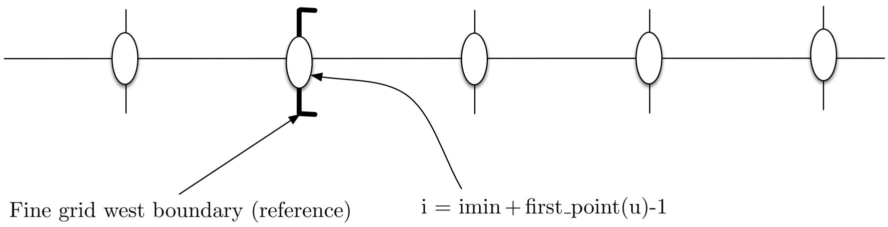


Figure 3.8: index of the first non-centered variable

2. Algorithm to update T and U variables

```
! Update of U points (copy)
fpu = first_point(U)
I_parent_U = imin+fpu-1
Do i = fpu, fpu+nx_cells, rhox
    U_parent(I_parent_U) = U(i)
    I_parent_U = I_parent_U+1
EndDo

! Update of T points (average)
fpt = first_point(T)
I_parent_T = imin+fpt-1
Do i = fpt, fpt+nx_cells-rhox+1, rhox
    T_parent(I_parent_T) = sum(T(i:i+rhox-1))/rhox
    I_parent_T = I_parent_T+1
EndDo
```

3. Odd refinement factor (3)

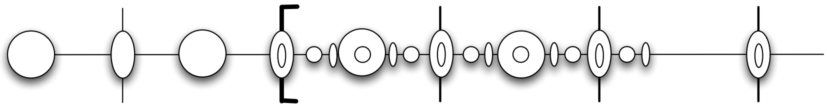


Figure 3.9: location of points for an odd refinement factor

A centered coarse point (T) is updated by either a copy or an average of fine points of the same mesh while a non-centered point (U) is updated by a copy of the matching fine point.

4. Even refinement factor (2)

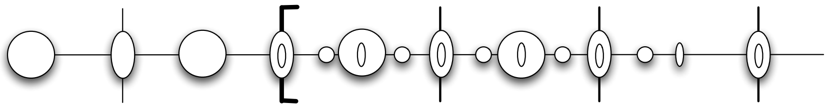


Figure 3.10: location of points for an even refinement factor

Here, centered coarse point (T) is update by an average while a copy is not possible because of no matching fine point. Non-centered coarse point (U) is updated by a copy of the corresponding fine point.

Chapter 4

Interpolation and update operations

4.1 Interpolation

4.1.1 Main procedures involved

Agrif_Set_BcInterp Type of interpolation at boundaries

Agrif_Set_Bc Where to interpolate

Agrif_Bc_Variable Make a boundary interpolation

Agrif_Set_Interp Type of interpolation within the domain

Agrif_Interp_variable Make an interior interpolation

Agrif_Init_variable To interpolate over the whole domain (within the domain and boundaries)

4.1.2 Interpolation: how to specify where to interpolate ?

1. Agrif_Set_Bc

   Agrif_Set_Bc(variable_id,point)
   with *point*=(/begin,end/)

   Examples:

   - Centered variables:

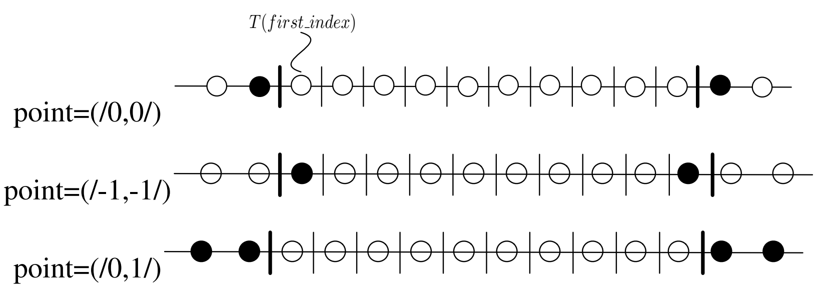


Figure 4.1: Location specification in case of interpolation of centered variable (T) in 1D

- Noncentered variables:

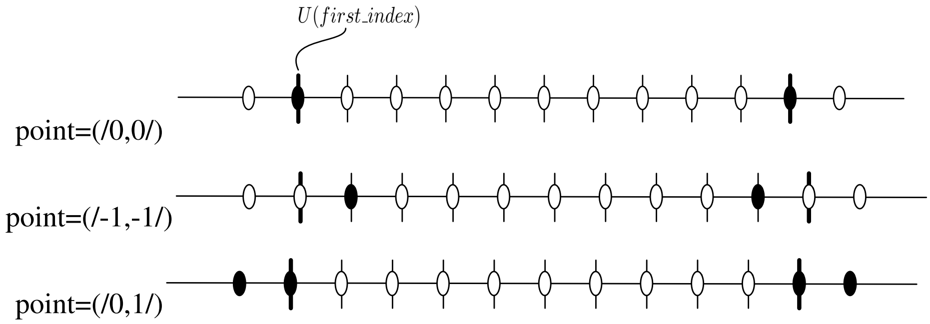


Figure 4.2: Location specification in case of interpolation of Noncentered variable (U) in 1D

2. Agrif_Set_BcInterp

Agrif_Set_BcInterp(Variable_id,interp=Agrif_Interp_Type)

The default value of Agrif_Interp_Type is *Agrif_constant*.

Interpolation schemes

	Order	Conservative	Monotone
Agrif_constant	0	X	X
Agrif_linear	1		
Agrif_linear_conserv	1	X	
Agrif_linear_conservlim	1	X	X
Agrif_lagrange	2		
Agrif_ppm	2	X	X
Agrif_eno	2	X	
Agrif_weno	3	X	

Examples:

```
1D
Call Agrif_Set_BcInterp(u_id,interp = Agrif_linear)

2D
Call Agrif_Set_BcInterp(u_id,interp = Agrif_linear)
or
Call Agrif_Set_BcInterp(u_id,interp2 = Agrif_PPM)
or
Call Agrif_Set_BcInterp(u_id, interp1 = Agrif_PPM, interp2 = Agrif_PPM)
```

remark:

- Interp1 indicates interpolation in the first dimension
- Interp2 indicates interpolation in the second dimension

4.1.3 The call: Agrif_Bc_Variable

1. Agrif_Bc_Variable

   Agrif_Bc_Variable(variable_id,procname)

   Example: procname Interp_MyTraceur

Call **Agrif_Bc_Variable**(u_id,procname=Interp_MyTraceur)

Algorithm of procname's call

a. AGRIF calls procname on the coarse grid with before = .TRUE.
   output array : tabres

b. AGRIF interpolates tabres on the fine grid

c. AGRIF calls procname on the fine grid with before = .FALSE.
   input array : tabres

```
Subroutine Interp_My_Traceur(tabres,i1,i2,before)
      use module_My_traceur
      real,dimension(i1:i2),intent(out) :: tabres
      Logical :: before

      if (before) then           ! on the parent grid
              tabres(i1:i2) = My_traceur(i1:i2)
      else                            ! on the child grid
              My_traceur(i1:i2) = tabres(i1:i2)
      endif
End Subroutine Interp_MyTraceur
```

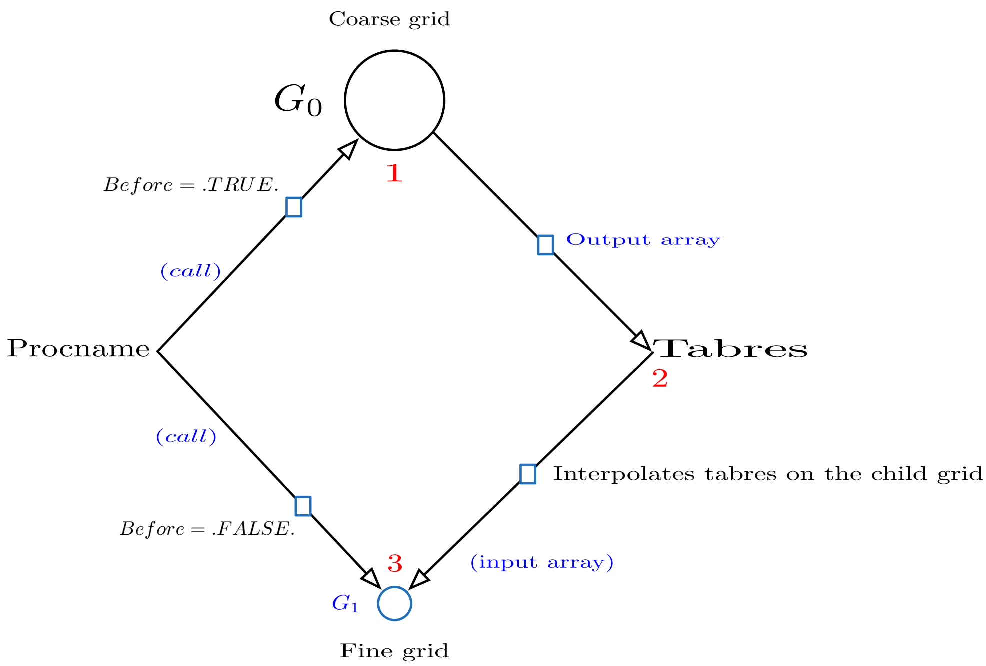


Figure 4.3: Algorithm of procname's call by AGRIF for interpolation

4.1.4 Time interpolation

1. How are time interpolations handle ?

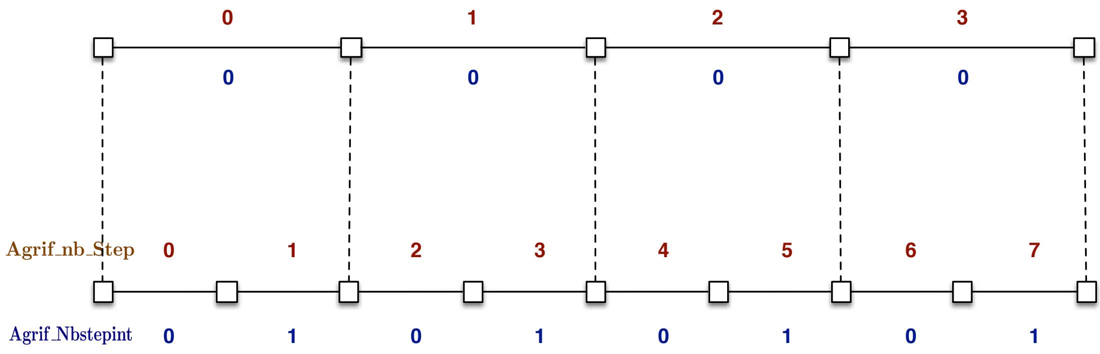


Figure 4.4: Grid step counters Agrif_Nb_Step and Agrif_Nbstepint with a time refinement factor of 2

<u>remark</u>: Agrif_Nb_Step changes every call to Agrif_Step

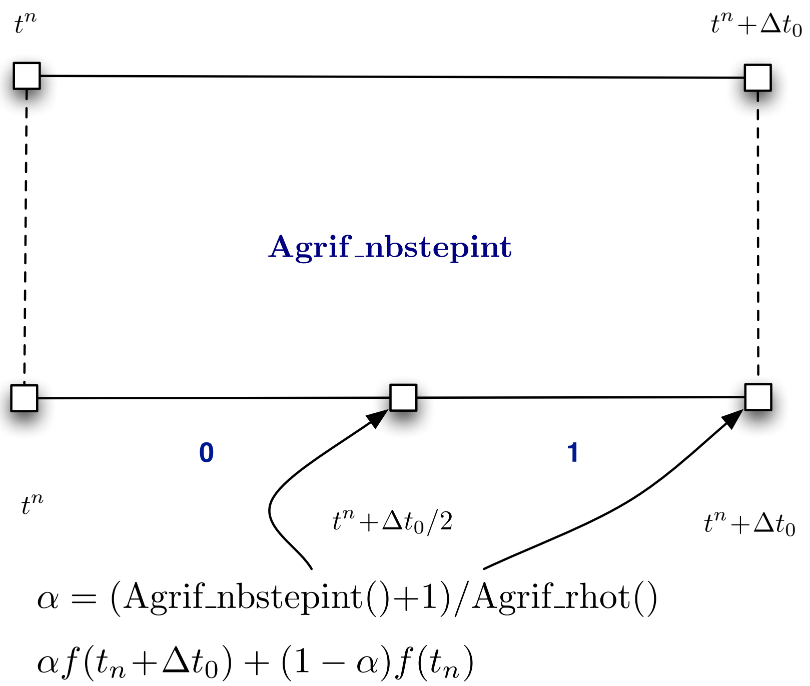


Figure 4.5: Description of time interpolation

2. Agrif_Bc_Variable

Agrif_Bc_Variable(variable_id,procname,calledweight=α)

Inside AGRIF, the spatial interpolation is done only when the time index of the parent grid changes. So, the spatial interpolation is done at the first call of *Agrif_Bc_Variable* inside a parent grid time step.

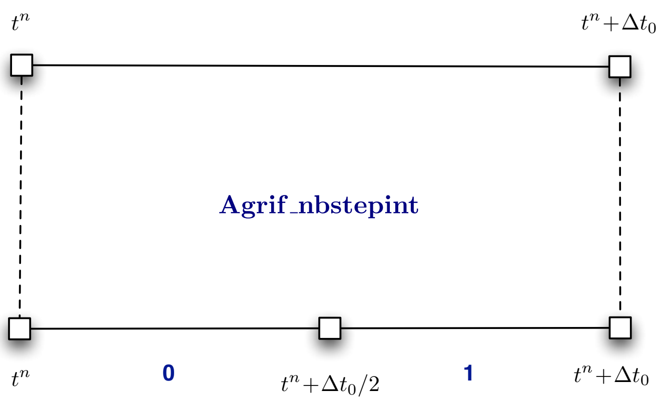


Figure 4.6: Description of Agrif_Nbstepint

remark:

- In order to interpolate the very first parent field, a call to *Agrif_Bc_Variable* in the *Agrif_InitValues* procedure is mandatory
- To force an interpolation (even if the parent grid index has not changed)

```
Call Agrif_Set_Bc(variable_id,point,Interpolationshouldbemade=.TRUE.)
```

4.1.5 Interpolation over the whole domain

The interpolation over the whole domain is done through the subroutine ***Agrif_Init_Variable***. In fact, *Agrif_Init_Variable* represents an addition of interpolation within the domain made through a call to the subroutine***Agrif_Interp_Variable*** and interpolation at boundaries made through a call to ***Agrif_Bc_Variable***. Interpolation with the subroutine *Agrif_Interp_Variable* is made from the first point within the domain to the last point within the domain.

It is important to notice that for boundary interpolation we use the subroutine ***Agrif_Set_bcinterp*** to indicate the type of interpolation and the subroutine ***Agrif_Set_interp*** in case of interpolation within the domain. All those two subroutines use the same arguments.

Agrif_Init_Variable = call to ***Agrif_Interp_Variable*** + call to ***Agrif_Bc_Variable***

Example:

- Centered points:

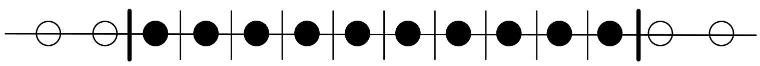


Figure 4.7: Interpolation of centered point (T)

- NonCentered points:

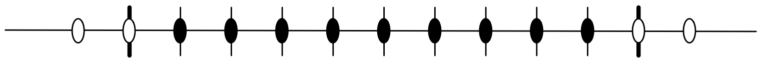


Figure 4.8: Interpolation of non-centered point (U)

4.1.6 Treatment of masked fields

Agrif_SpecialValue, replacement of masked values by neighboring values

Agrif_UseSpecialValue= .true.

Agrif_SpecialValue = Val

must be set before the call to Agrif_Bc_Variable, Agrif_Interp_Variable

Examples:

```
Agrif_UseSpecialValue=.true.
Agrif_SpecialValue=0.
Call Agrif_Bc_Variable(variable_Id,procname)
Agrif_UseSpecialValue=.false.
```

remark

Setting maximum lookup of masked values (since it may affect performance)

Call ***Agrif_Set_MaskMaxSearch***(mymaxsearch)

Defaut Value: MaxSearch = 5

4.2 Updates

4.2.1 Main procedures involved

Agrif_Set_UpdateType Type of update

Agrif_Update_Variable Make the restriction and specify where to update

4.2.2 Update: how to specify the type of update ?

The type of update has to be indicated through the subroutine **Agrif_Set_UpdateType**

1. Agrif_Set_UpdateType

   Agrif_Set_UpdateType(variable_id,update = Agrif_Update_Type)

   Agrif_Update_Type should be either *Agrif_Update_Copy*, *Agrif_Update_Average* or *Agrif_Update_Full_Weighting*

2. Update schemes

	First Order	Second Order
Agrif_Update_Copy	∞	0
Agrif_Update_Average	2	1
Agrif_Update_Full_Weighting	2	2

Examples:

```
1D
Call Agrif_Set_UpdateType(variable_id,update = Agrif_average)
2D
Call Agrif_Set_UpdateType(variable_id,update = Agrif_average) ! same update scheme in all directions
Call Agrif_Set_UpdateType(variable_id,update1 = Agrif_Copy, update2 = Agrif_Average)
```

Examples: Space refinement of 3

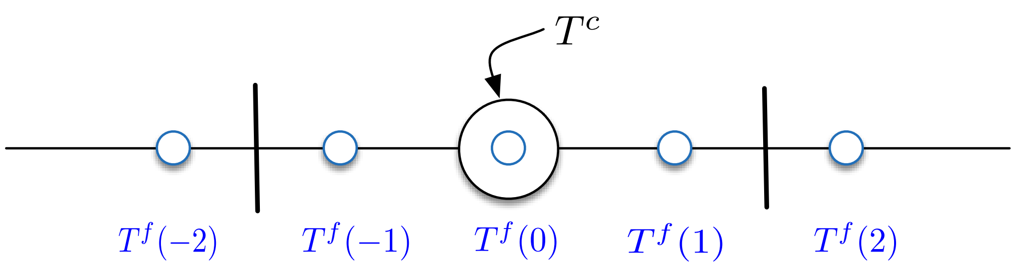


Figure 4.9: Relative positions of coarse and fine points with space refinement of 3

copy: $$T^c = T^f(0)$$

Average: $$T^c = \frac{1}{3}\left[T^f(-1) + T^f(0) + T^f(1)\right]$$

FW: $$T^c = \frac{1}{9}\left[T^f(-2) + 2T^f(-1) + 3T^f(0) + 2T^f(1) + T^f(2)\right]$$

4.2.3 The call: Agrif_Update_Variable

Agrif_Update_Variable(variable_id,locupdate,procname)

Example:

Call **Agrif_Update_Variable**(u_id,locupdate=(/0,0/), procname=Update_MyTraceur)

remark: locupdate is an optional keyword that indicates the location the update takes place.

4.2.4 Specify the locations that you want to update

1. Without locupdate keyword

   Agrif_Update_Variable(variable_id,procname)

   Default location without locupdate keyword: interior of the reference grid

   - Centered variable:

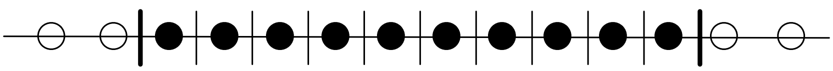
   

   Figure 4.10: Update the centered variable without locupdate keyword

   - Noncentered variable:

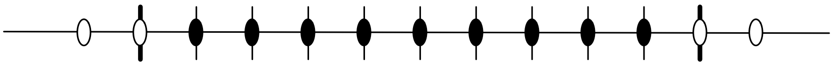
   

   Figure 4.11: Update the Noncentered variable without locupdate keyword

2. With locupdate keyword

   Update on limited areas

   Agrif_Update_Variable(variable_id,locupdate,procname)

Examples

- Centered points:

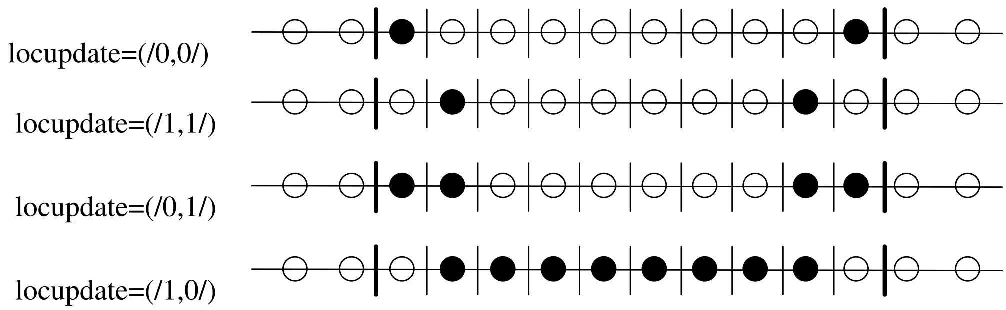


Figure 4.12: Update of centered points(T) with locupdate keyword

- Non centered points:

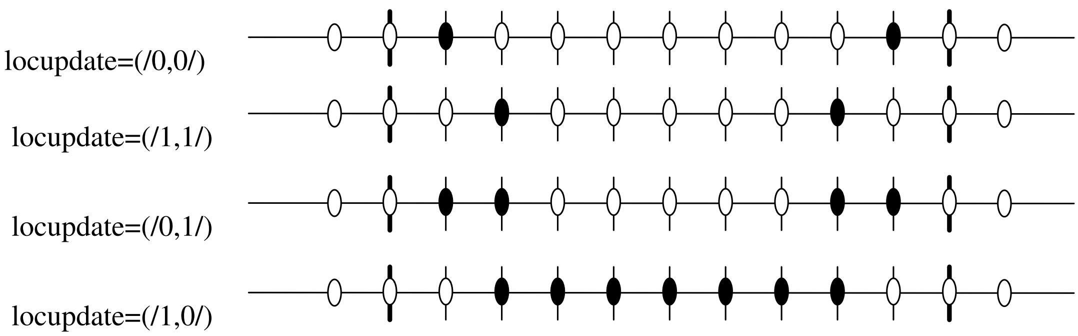


Figure 4.13: Update of non-centered points (U) with locupdate keyword

remark: Multidimension
locupdate1=... , locupdate2 = ...

4.2.5 The use of procnames

1. Agrif_Update_Variable

   Agrif_Update_Variable(variable_id,procname)

   Examples

   Call Agrif_Update_Variable(variable_id,procname=Update_MyTraceur)

   Algorithm of procname's call

   a AGRIF calls procname on the child grid with before = .TRUE.
   output array : tabres

   b AGRIF updates tabres on the parent grid

   c AGRIF calls procname on the coarse grid with before = .FALSE.
   input array : tabres

```
Subroutine Update_My_Traceur(tabres,i1,i2,before)
use module_My_traceur
real,dimension(i1:i2),intent(inout) :: tabres
logical :: before

  If (before) then             ! on the child grid
   tabres(i1:i2) = My_traceur(i1:i2)
  Else                         ! on the parent grid
   My_traceur(i1:i2) = tabres(i1:i2)
  Endif
End Subroutine Update_My_Traceur
```

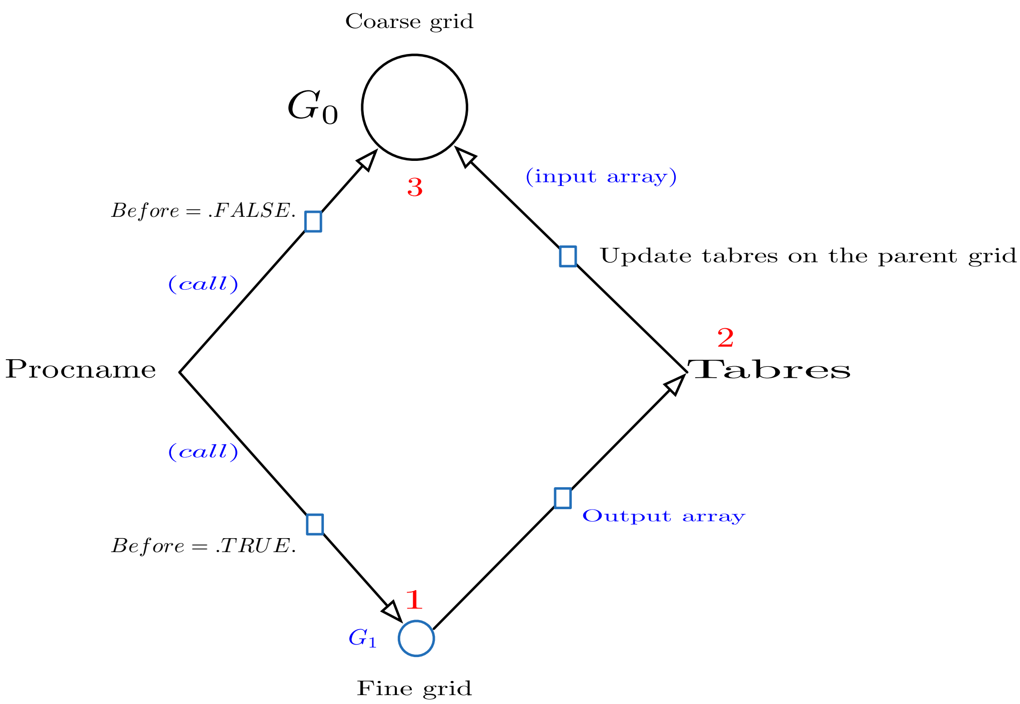


Figure 4.14: Algorithm of procname's call by AGRIF for update

4.2.6 Treatment of masked fields

Agrif_SpecialValue_InfineGrid, the masked values are not taken into account in the restriction operation

Agrif_UseSpecialValueInUpdate = .true.

Agrif_SpecialValueFineGrid = Val

must be set before the call to Agrif_Update_Variable

Examples

```
Agrif_UseSpecialValueInUpdate=.true.
Agrif_SpecialValueFineGrid=0.
Call Agrif_Update_Variable(variable_id,procname=Update_MyTraceur)
Agrif_UseSpecialValueInUpdate=.false.
```

Chapter 5

Distributed and shared memory parallelization

5.1 Distributed memory

How to make your MPI code work with Agrif ?

Example:

In your Makefile

```
CPPFLAGS += -DAGRIF_MPI
```

MPI initilization

```
use Agrif_Mpp
call MPI_INIT(status)
call Agrif_MPI_Init()
```

You have to tell Agrif how to convert local indices (on a given proc) into global workspace:

Example: `Agrif_Invloc`

```
subroutine Agrif_Invloc ( indloc, procnum, dir, indglob )

    integer, intent(in)  :: indloc   ! local index  (input)
    integer, intent(in)  :: procnum  ! current MPI proc id
    integer, intent(in)  :: dir      ! direction (1,2,3)
    integer, intent(out) :: indglob  ! global index (output)

    select case( dir )
    case(1)   ;   indglob = indloc + ishiftmpi(procnum)
    case(2)   ;   indglob = indloc + jshiftmpi(procnum)
    case(3)   ;   indglob = indloc
    end select

end subroutine Agrif_Invloc
```

Currently, all grids are integrated sequentially:

$$\begin{array}{r|ccccccc} \text{sequence} & i_1 & & i_2 & & i_3 & & \\ \text{grid} & G_0 & \rightarrow & G_1 & \rightarrow & G_2 & \rightarrow & \ldots \end{array}$$

For each grid, all processors are used.

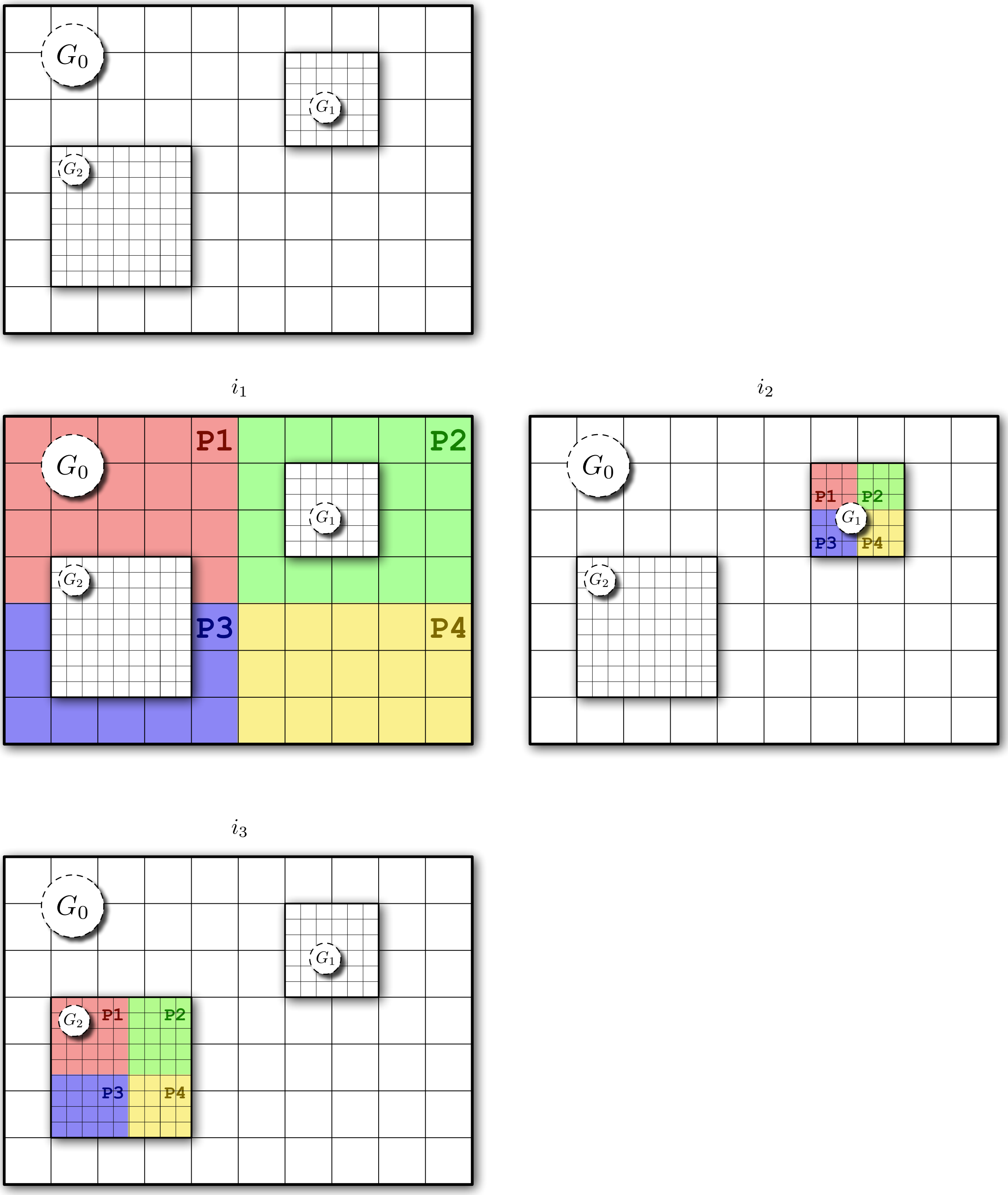


Figure 5.1: Processors utilisation for each grid when grids are integrated sequently

Sister grids (same parent) could be integrated concurently:

$$\begin{array}{r|ccccc} \text{sequence} & i_1 & & i_2 & & \\ \text{grid} & G_0 & \rightarrow & G_1 \,//\, G_2 & \rightarrow & \dots \end{array}$$

For each sequence, processors are distributed on the grids of same level.

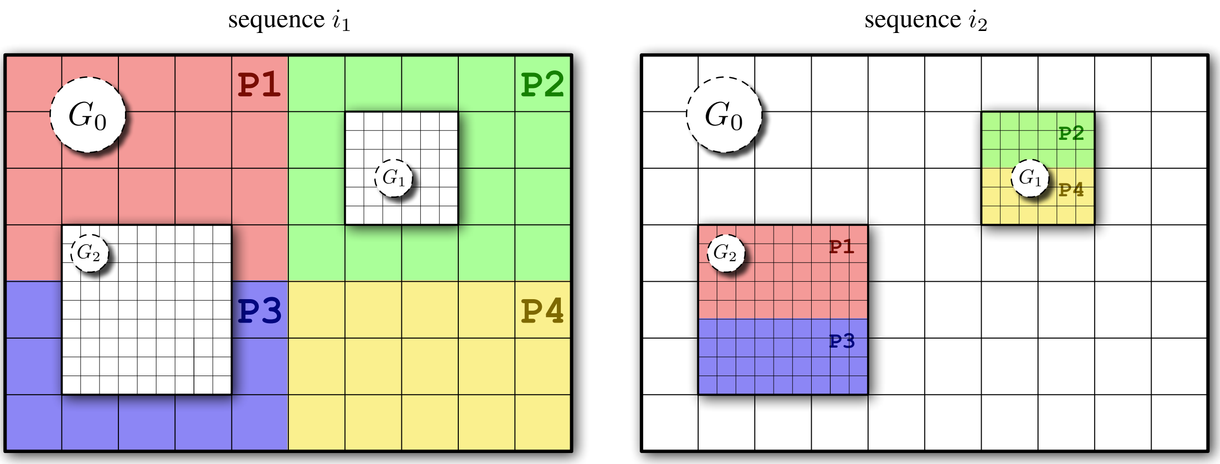


Figure 5.2: Distribution of processors on the grids of same level

5.2 Shared Memory

Potentially less efficient (currently all the interpolation/restriction work is done by one thread), potential use of nested parallelism is being studied

Example:

```
!$OMP MASTER
  Call Agrif_Bc_Variable(u_id,procname=Interp_MyTraceur)
!$OMP END MASTER
```

Chapter 6

Adaptive grid refinement

6.1 Parameters of adaptive refinement

- **Parameters & Main functions**

Agrif_Set_Rafmax(nlevel)	Maximum Level of refinement
Agrif_Set_Regridding(nbsteps)	Reconstruction the hierarchy every nbsteps (root) grid steps
Agrif_Set_Minwidth(minwidth)	Minimun size of the grid

Example:

```
Call Agrif_Set_Regridding(30)
Call Agrif_Set_Minwidth(18)
Call Agrif_Set_Rafmax(3)
```

6.2 Refinement Criterion

- **Agrif_Detect**

Agrif_Detect(taberr,size_taberr)
taberr = 1
Where error is detected, taberr values are located at grid cell corners of the reference grid (including boundaries)

Example:

```
      SUBROUTINE Agrif_detect(taberr,sizexy)
      implicit none
# include "ocean2d.h"
      Integer, Dimension(2) :: sizexy
      Integer,Dimension(sizexy(1),sizexy(2))   :: taberr
      real vort(GLOBAL_2D_ARRAY)

      do j=1,Mm+1
      do i=1,Lm+1
        vort(i,j) = (v(i,j)-v(i-1,j))-(u(i,j)-u(i,j-1))
      enddo
      enddo
      crit = maxval(abs(vort))
      taberr=0
      where abs(vort)>0.8*crit
          taberr=1
      end where
      End Subroutine Agrif_detect
```

remark:

Agrif_detect has to be written even without adaptive mesh refinement

6.3 Variable restoring

Restoring of grid variable from one grid hierarchy to another

Declaration: Call **Agrif_Declare_Variable**(...,variable_id,restore=.true.)

Specify what to restore: Agrif_Before_Regridding, Agrif_Save_ForRestore

Example:

```
Call Agrif_Declare_Variable(...,zeta_id,restore=.true.)
Subroutine Agrif_Before_Regridding()
#include "ocean2d.h"
   Call Agrif_Save_ForRestore(Zt_avg1,zeta_id)
End Subroutine Agrif_Before_Regridding()
```

remark

Agrif_Before_Regridding has to be written even without adaptive mesh refinement

Chapter 7

Example of Advanced use

7.1 Flux correction

1. Flux correction (I) (ROMS)

   How to implement a flux correction procedure ?

$$\frac{\partial T}{\partial t} + \frac{\partial F^x}{\partial x} + \frac{\partial F^y}{\partial y} = 0$$

$$T_{i,j}^{n+1} = T_{i,j}^{n} - \frac{\Delta t}{\Delta x}\left(F_{i,j}^{x} - F_{i-1,j}^{x}\right) - \frac{\Delta t}{\Delta y}\left(F_{i,j}^{y} - F_{i,j-1}^{y}\right)$$

   Store the fluxes on the fine and coarse grid and use the differences to correct the coarse grid points near the interface.

2. Flux correction (II) (ROMS)

   (a) Declare flux arrays (F_x, F_y), compute then and make the summation on the fine grids over a parent time step

   (b) Declare the profiles corresponding to these fluxes

   Call Agrif_Declare_variable((/1,2/),(/1,1/),(/'x','y'/),(/0,1/),(/nx,ny/),$F_{x\mathrm{id}}$)

   Call Agrif_Declare_variable((/2,1/),(/1,1/),(/'x','y'/),(/1,0/),(/nx,ny/),$F_{y\mathrm{id}}$)

   (c) Set the update type

   Call Agrif_Set_UpdateType($F_{x\mathrm{id}}$,update1=Agrif_Update_Copy, update2=Agrif_Update_Average)

   Call Agrif_Set_UpdateType($F_{y\mathrm{id}}$,update1=Agrif_Update_Average, update2=Agrif_Update_Copy)

3. Make the flux correction inside a call to Agrif_Update

   Call Agrif_Update_Variable($F_{x\mathrm{id}}$, procname = Correct_Flux_x)

   Call Agrif_Update_Variable($F_{y\mathrm{id}}$, procname = Correct_Flux_y)

4. Flux correction (III) (ROMS)

Example:Correct_Flux_x

```
Subroutine Correct_Flux_x(tabres,i1,i2,j1,j2,before,nb,ndir)
logical :: western_side, eastern_side

if (before) then
   tabres = Fx(i1:i2,j1:j2)
else
   western_side  = (nb == 1).AND.(ndir == 1)
   eastern_side  = (nb == 1).AND.(ndir == 2)
   if (western_side) then
      do j=j1,j2
        My_traceur(i1,j)   = My_traceur(i1,j)   -(dt/(dx))(tabres(i1,j)-Fx(i1,j))
      enddo
   endif
   if (eastern_side) then
      do j=j1,j2
        My_traceur(i2+1,j) = My_traceur(i2+1,j) +(dt/(dx))(tabres(i2,j)-Fx(i2,j))
      enddo
   endif
endif
End Subroutine Correct_Flux_x
```

7.2 Wetting and drying

1. Wetting and drying (I) (MARS)

   How to specify if we are in land or water through the SpecialValue ?

   Example:Interp_Traceur

```
Agrif_Use_SpecialValue = .TRUE.
Agrif_SpecialValue = -999.
Call Agrif_Bc_Variable(tabtemp, my_Traceur_id,procname = Interp_my_Traceur)

Subroutine Inter_my_Traceur(tabres,i1,i2,j1,j2,before)

if (before) then
   where ((h0(i1:i2,j1:j2)+ssh(i1:i2,j1:j2)) < min_depth)
       tabres = Agrif_SpecialValue
   elsewhere
      tabres = my_Traceur(i1:i2,j1:j2)
   endwhere
else
   where (tabres /= Agrif_SpecialValue)
     My_traceur = tabres
   endwhere
endif
End Subroutine Inter_my_Traceur
```

7.3 Vertical refinement

1. Different vertical grids (I) (NEMO)

   The grids have different vertical meshes. The vertical remapping between the two grids are done in the procnames routines.

   Example:Update with different vertical grids

```
Subroutine Update_My_Traceur(tabres,i1,i2,k1,k2,before)
real,dimension(i1:i2,k1:k2) :: tabres
logical before

if (before) then
   ! on the child grid
   tabres = My_Traceur(i1:i2,k1:k2)
else
   ! on the parent grid (NB: tabres has a vertical dimension corresponding to the one of fine grid)
   do i=i1,i2
     call remap(tabres(i,:),My_traceur(i,:)) ! remap is a vertical remapping procedure
   enddo
endif
End Subroutine Update_My_Traceur
```

2. Different vertical grids (II) (NEMO)

   The grids have different vertical mesh. The vertical remapping between the two grids are done in the procnames routines.

   Example:Interpolation with different vertical grids

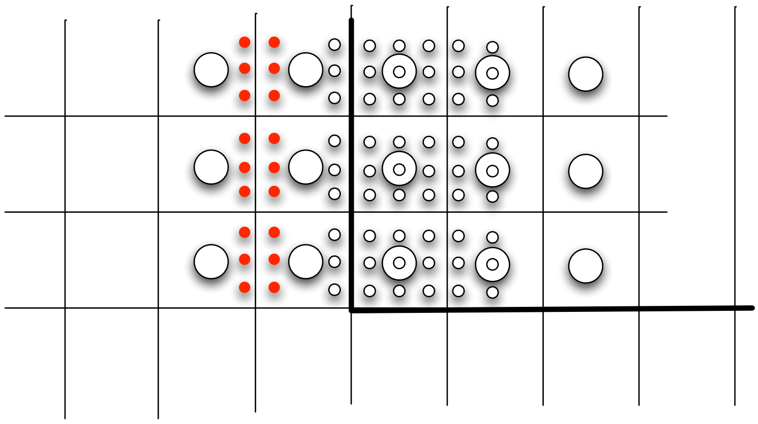
```
Subroutine Interp_My_Traceur(tabres,i1,i2,k1,k2,before)
real,dimension(i1:i2,k1:k2) :: tabres
logical before

if (before) then
   ! on the parent grid
   tabres = My_Traceur(i1:i2,k1:k2)
else
   ! on the child grid (NB: tabres has a vertical dimension corresponding to the one of parent grid)
   do i=i1,i2
     call remap(tabres(i,:),My_traceur(i,:)) ! remap is a vertical remapping procedure
   enddo
endif
End Subroutine Interp_My_Traceur
```

7.4 Interpolation with fine grid values

1. Tracer interpolation (I) (NEMO)

   We want to use internal fine grid values during the interpolation



2. Tracer interpolation (II) (NEMO)

   - Profiles

```
decal = Agrif_irhox()+1\\
Call Agrif_Declare_Profiles((/2,2/),(/1,1/),(/-decal,-decal/),(/nx+decal,ny+decal/),My_Traceur_id)
Call Agrif_Set_Bc(My_traceur_id,(/decal-1,decal/))
! NB: if not specified interp type is Agrif_constant
Call Agrif_Set_Bcinterp(My_traceur_id,interp12=Agrif_ppm,interp21=Agrif_ppm)
```

3. Tracer interpolation (III) (NEMO)

   - Interpolation procnames

```
Call Agrif_Bc_Variable(tabtemp, My_traceur_id,procname = Interp_My_Traceur)

Subroutine Interp_My_Traceur(tabres,i1,i2,j1,j2,before,nb,ndir)
if (before) then
 tabres = My_Traceur(i1:i2,j1:j2)
else
 western_side  = (nb == 1).AND.(ndir == 1)
 if (western_edge) then
  do j=j1,j2
   if (u(1,j) <0.) then
    My_Traceur(0,j)=a1*tabres(-Agrif_irhox(),j)  + b1*My_Traceur(1,j) + (1.-a1-b1)*My_Traceur(2,j)
   else
    My_Traceur(0,j)=a2*tabres(-Agrif_irhox()-1,j)+b2*tabres(-Agrif_irhox(),j)+(1.-a2-b2)*My_Traceur(1,j)
   endif
  enddo
 endif
endif
End Subroutine Interp_My_Traceur
```