Agrif developers training

Agrif Documentation

Laurent Debreu[1,2] Marc Honnorat[1,2] Rachid Benshila[3]

[1]INRIA - MOISE project team, Grenoble, France

[2]Laboratoire Jean Kuntzmann

[3]CNRS, LOCEAN, Paris, France

June, 18-19, 2012

Outline

The AGRIF software

Main idea: bring (fixed or adaptive) mesh refinement features to existing models

- written in the Fortran language
- discretized on a structured grid

Alternatives

- Hand coded solutions (e.g. ROMS-Rutgers)
- One code software : (e.g. RSL WRF)
- General softwares : Paramesh (Fortran90), SAMRAI (C++), CHOMBO (C++)
  http://www.llnl.gov/casc/SAMRAI/software/software.html, http://www.physics.drexel.edu/ olson/paramesh-doc /Users manual/amr.html,http://seesar.lbl.gov/ANAG/chombo/index.html

The AGRIF software

Missing features (to be more detailed during the presentation): adaptive grid refinement and distributed memory parallelization, neighboring grids

Main contributors and main financial support

Main contributors

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

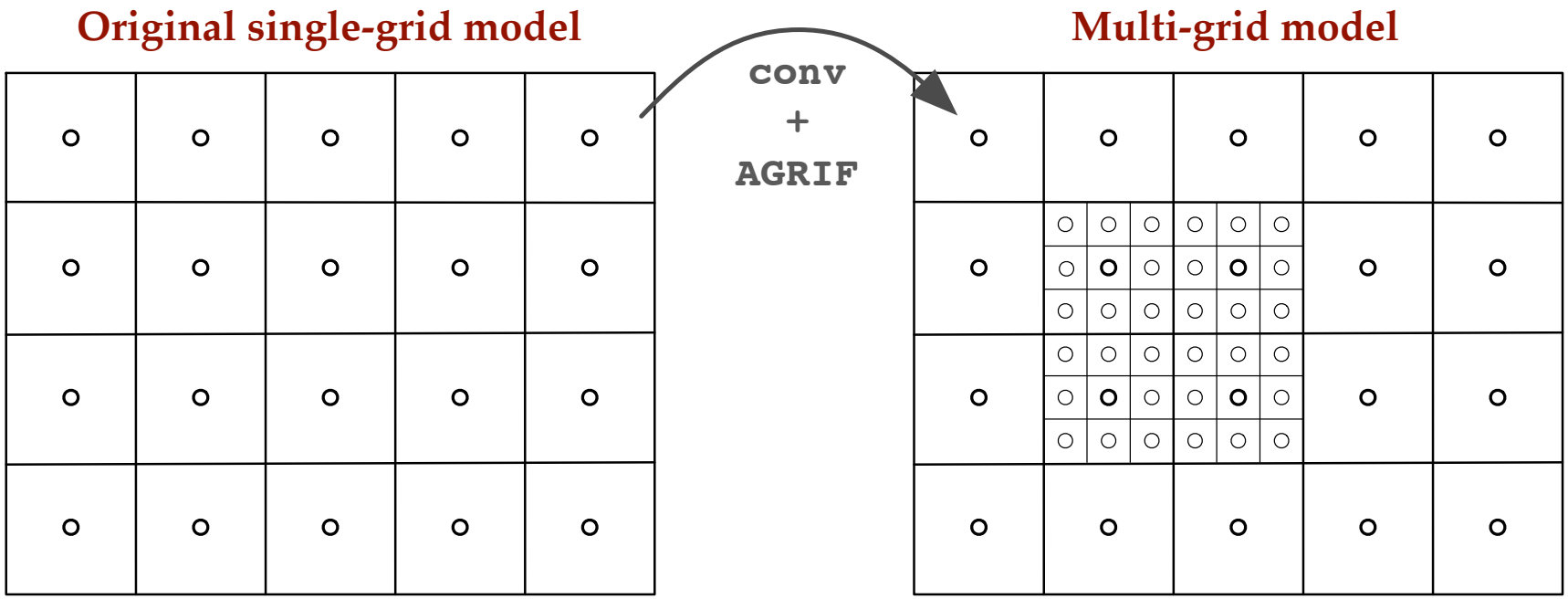
Financial support

SHOM (Initial support), INRIA (First distributed version), MERCATOR-OCEAN (Agrif and Fortran90, implementation in NEMO), IFREMER (R&D contracts), ANR (Pulsation)

Outline

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



Source code transformation tool: conv

Original code

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

- 'gridvar' is declared grid-dependant in agrif.in configuration file
- 'data_in' is function argument

Source code transformation tool: conv

Original code

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

Conv'ed code

```
subroutine do_stuff ( data_in )
    use Agrif_Util
    real(8), dimension(:,:) :: data_in
    call Sub_Loop_do_stuff(data_in, Agrif_tabvars(12)%array2)
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
```

A typical compilation organization

Let our compilation directory be organized as follows:

```
src/:
  Agrif2Model.F90     <- glue code: should be compiled last
  Agrif_User.F90
  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

agrif.in

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

% Name of the common file : an include file (paramfile) %
%                           OR a module (parammodule)   %
parammodule mod_global;

%%
USE ONLY_FIXED_GRIDS;
```

inside code.F90

```
module mod_global
    integer :: nx, ny
end module mod_global
```

A typical compilation organization

1. Preprocessing

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

2. Call to conv program

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

3. Re-preprocessing

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

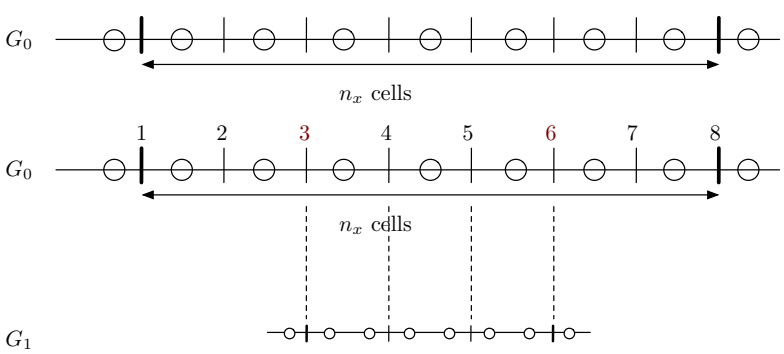
4. Fortran compiltation

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

Outline

Outline

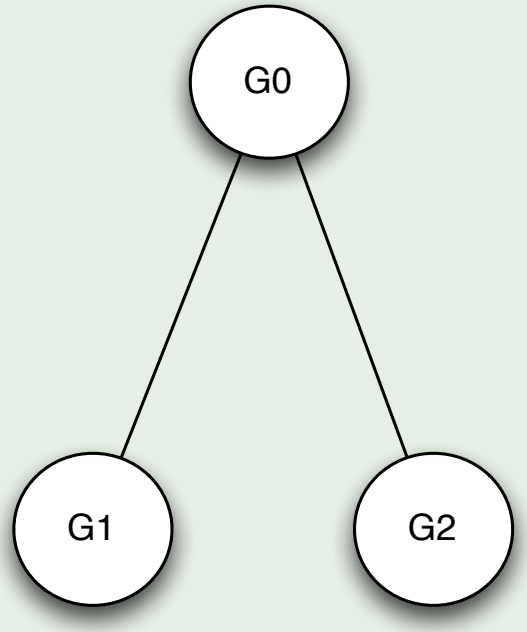
The reference grid



- Defined by its number of cells
- The reference grid for the definition of fine grids locations
- AGRIF_Fixed_Grids.in:
  1
  3 6 2 2 # imin imax rhox rhot
  0
- The reference grid for the specifications of interpolation/restriction schemes

The AGRIF_Fixed_Grids.in file

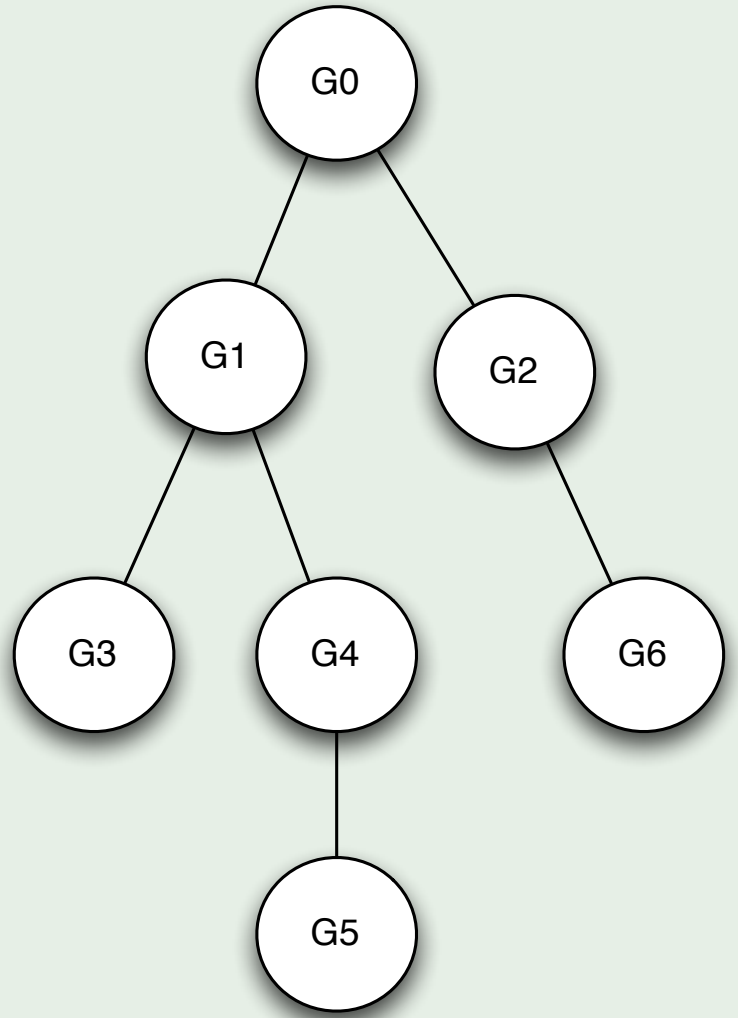
A grid hierarchy



$$
\begin{array}{l}
2 \\
\mathrm{imin}_1 \ \mathrm{imax}_1 \ \mathrm{rho}_{x,1} \ \mathrm{rho}_{t,1} \\
\mathrm{imin}_2 \ \mathrm{imax}_2 \ \mathrm{rho}_{x,2} \ \mathrm{rho}_{t,2} \\
0 \\
0
\end{array}
$$

The AGRIF_Fixed_Grids.in file

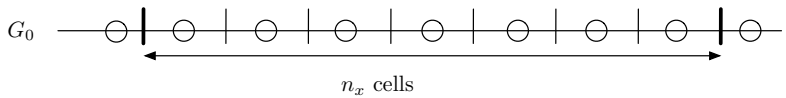
A grid hierarchy



$$
\begin{array}{l}
2 \\
\mathrm{imin}_1\ \mathrm{imax}_1\ \mathrm{rho}_{x,1}\ \mathrm{rho}_{t,1} \\
\mathrm{imin}_2\ \mathrm{imax}_2\ \mathrm{rho}_{x,2}\ \mathrm{rho}_{t,2} \\
2 \\
\mathrm{imin}_3\ \mathrm{imax}_3\ \mathrm{rho}_{x,3}\ \mathrm{rho}_{t,3} \\
\mathrm{imin}_4\ \mathrm{imax}_4\ \mathrm{rho}_{x,4}\ \mathrm{rho}_{t,4} \\
0 \\
1 \\
\mathrm{imin}_5\ \mathrm{imax}_5\ \mathrm{rho}_{x,5}\ \mathrm{rho}_{t,5} \\
0 \\
1 \\
\mathrm{imin}_6\ \mathrm{imax}_6\ \mathrm{rho}_{x,6}\ \mathrm{rho}_{t,6} \\
0
\end{array}
$$

Outline

Specification of variables that need to be interpolated/restricted



Specification of variable centering and location

Agrif_Set_type(variable,(/stag/),(/first_index/)
Agrif_Set_raf(variable,(/'dim'/))

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

Example

```
Call Agrif_Set_Type(u,(/1/),(/1/))
Call Agrif_Set_Raf(u,'x')
```

Special declarations of profiles

In order to use more complex interpolation/update (via the procnames see after) and/or to interpolate/restrict non existing variables

Specification of variable centering and location

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

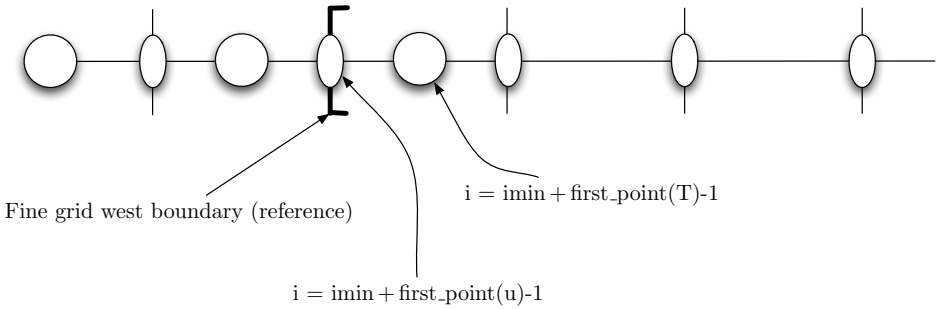
Example

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

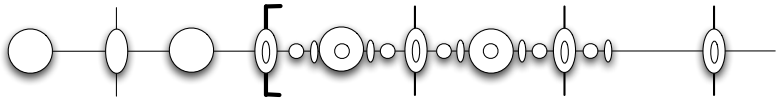
Remark

Has also to be called on the root grid

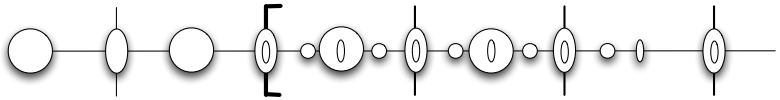
Relative Positions on the grids



Odd refinement factor (3)



Even refinement factor (2)



```
! 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
```

Outline

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 have to be written by the user:

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

Agrif_InitValues initialization routine called once for each fine grid

- compute grid indices
- declare Agrif profiles
- initialize variables
  (eg: read data from file / interpolation from coarse grid)

Main AGRIF procedures

Example

```
program example
    use mod_global
    integer :: i
    call Agrif_Init_Grids()
    call read_config()
    call init()
    do i=1,nbsteps
        call Agrif_Step(step)
    enddo
end program example
```

Main AGRIF procedures

Example Agrif_InitValues

```
subroutine Agrif_InitValues ( )
    Nnx = nx + 1
    Nny = ny + 1
    call init()
end subroutine Agrif_InitValues
```

Example Agrif_InitWorkspace

```
subroutine Agrif_InitWorkspace ( )
    use mod_global
    if ( Agrif_Root() ) then
        nx = Nnx - 1
        ny = Nny - 1
    endif
end subroutine Agrif_InitWorkspace
```

Main AGRIF procedures

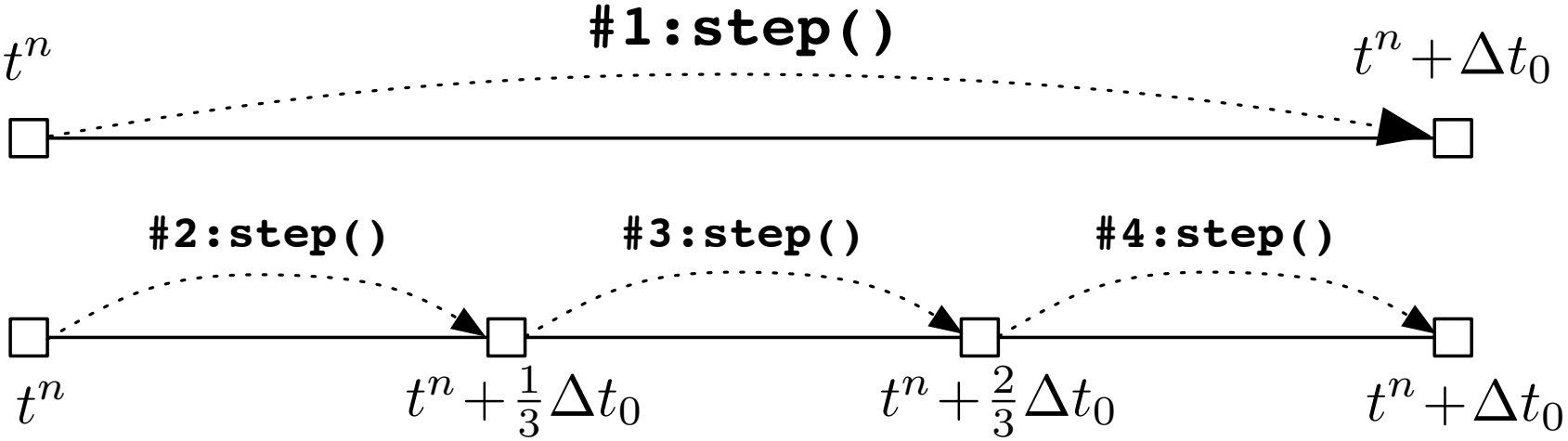
They can be called:

Agrif_Regrid	reads `AGRIF_Fixed_Grids.in` and build grids data structures (called anyway if not explicitely).
Agrif_Step_Child(func)	calls `func()` recursively for each grid (no time refinement).

$\Rightarrow$ examples in shallow-water practical session.

Integration of the grid hierarchy: `Agrif_Step`

A call to `Agrif_Step(step)` takes into account time refinement factor:



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 of integrations for current grid
Agrif_Nbstepint()	sub-step number inside parent grid integration

Example: is it time for update ?

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

Outline

Outline

Main procedures involved

Agrif_Set_Bc_Interp	Type of interpolation at boundaries
Agrif_Set_Bc	Where to interpolate
Agrif_Bc_Variable	Make a boundary interpolation
Agrif_Set_Interp	Type of interpolation
Agrif_Interp	Make an interior interpolation
Agrif_Init	Make an interior and bc interpolation

Interpolation: how to specify where to interpolate ?

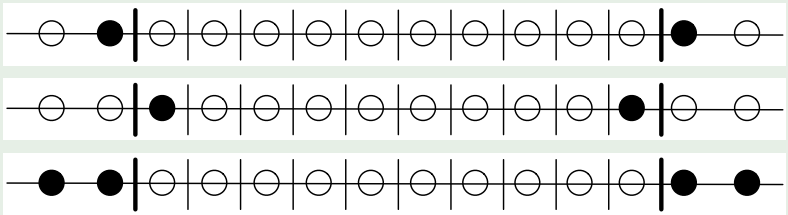
Agrif_Set_Bc

Agrif_Set_Bc(variable,point) with point=(/begin,end/)

Examples

Centered points:

point=(/0,0/)
point=(/-1,-1/)
point=(/0,1/)



Interpolation: how to specify where to interpolate ?

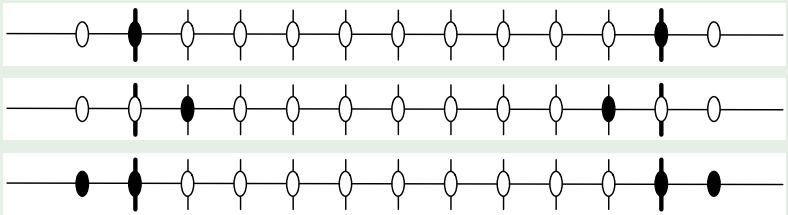
Agrif_Set_Bc

Agrif_Set_Bc(variable,point) with point=(/begin,end/)

Examples

Non centered points:

point=(/0,0/)
point=(/-1,-1/)
point=(/0,1/)



Interpolation: how to specify the type of interpolations (for each direction)

Agrif_Set_Bc_Interp

Agrif_Set_Bc_Interp(variable,interp=Agrif_Interp_Type)

Examples

```
1D
Call Agrif_Set_Bc_Interp(variable,interp = Agrif_linear)

2D
Call Agrif_Set_Bc_Interp(variable,interp = Agrif_linear)
or
Call Agrif_Set_Bc_Interp(variable,interp2 = Agrif_PPM)
or
Call Agrif_Set_Bc_Interp(variable,interp12 = Agrif_PPM, interp21 = Agrif_PPM)
```

The call: Agrif_Bc_Variable

Agrif_Bc_Variable

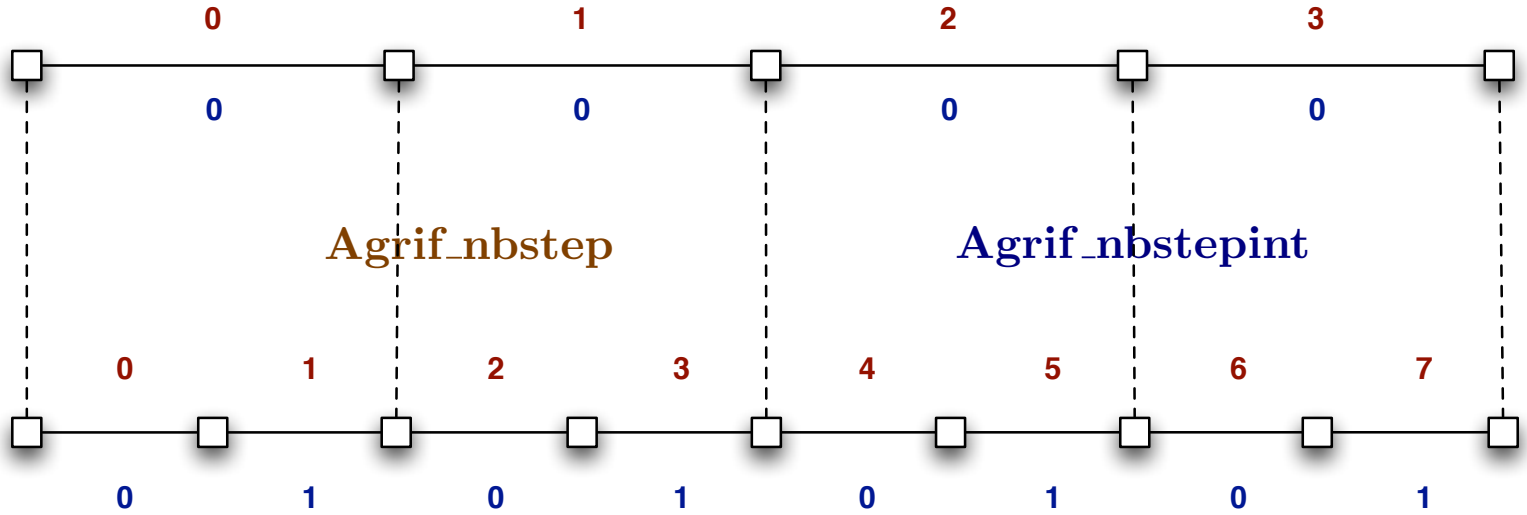
Agrif_Bc_Variable(tabres, variable)
tabres is the result of the interpolation

Example

Call Agrif_Bc_Variable(u,u)

Time interpolation

How are time interpolations handle ?

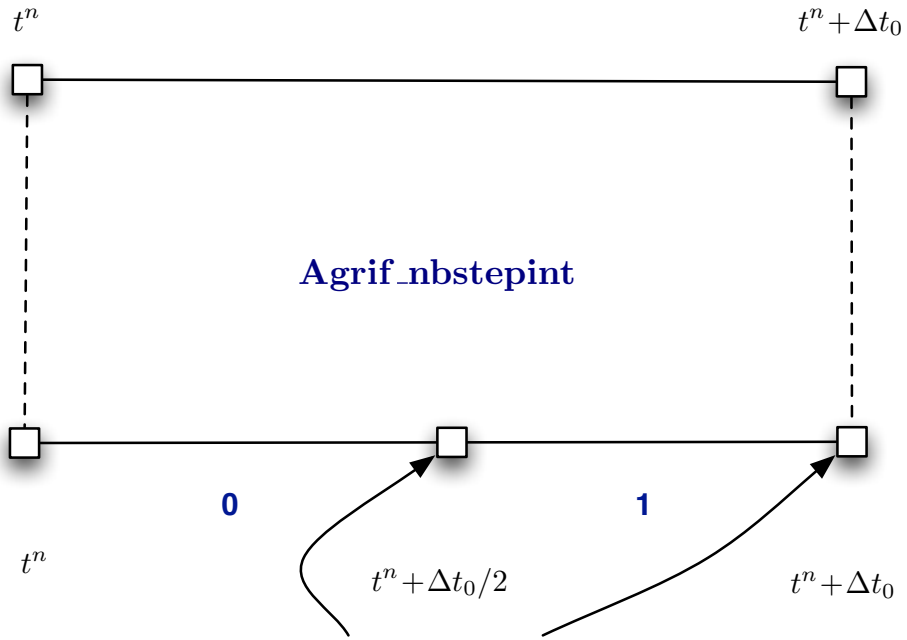


Remark

Agrif_nbstep changes every call to Agrif_Step

Time interpolation

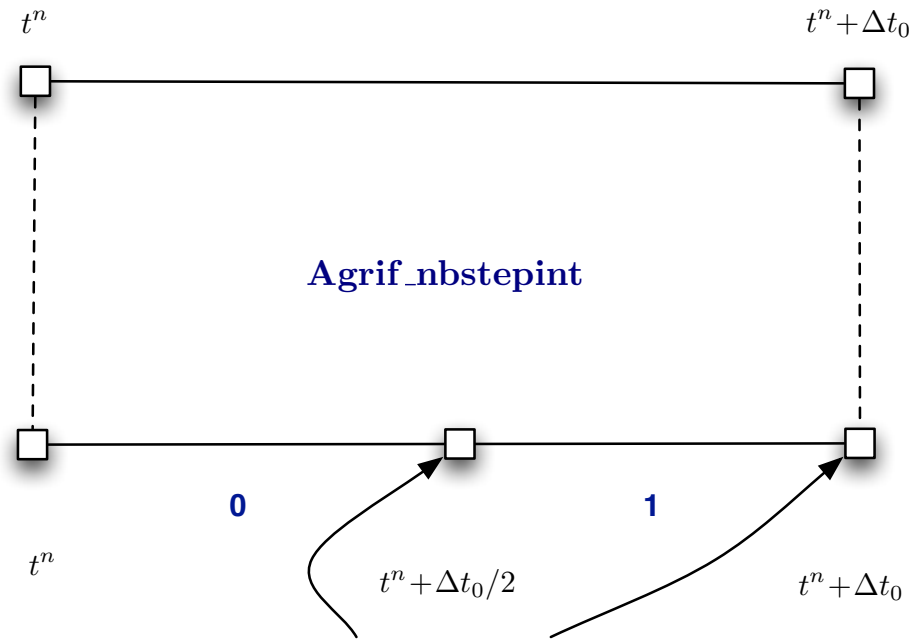
How are time interpolations handle ?



$$\alpha = (\text{Agrif\_nbstepint}()+1)/\text{Agrif\_rhot}()$$

$$\alpha f(t_n+\Delta t_0) + (1-\alpha) f(t_n)$$

Time interpolation



$$\alpha = (\text{Agrif\_nbstepint}()+1)/\text{Agrif\_rhot}()$$

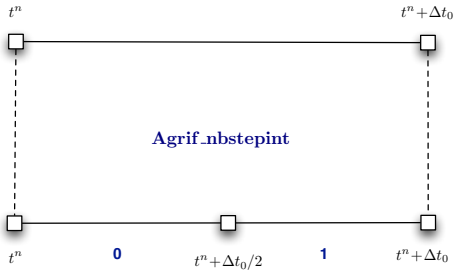
$$\alpha f(t_n+\Delta t_0) + (1-\alpha) f(t_n)$$

Agrif_Bc_Variable

Agrif_Bc_Variable(tabres, variable,callweight=α)

Time interpolation

Inside AGRIF, the spatial interpolation is done only when the time index of the parent grid changes



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,point,Interpolationshouldbemade=.TRUE.)
```

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	

The use of procnames

Agrif_Bc_Variable

Agrif_Bc_Variable(tabres, variable/variable_id,procname)

Examples

```
Call Agrif_Bc_Variable(tabres,variable,procname=Interp_MyTraceur)

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 = My_traceur(i1:i2)
else                        ! on the child grid
    My_traceur(i1:i2) = tabres
endif
End
```

Interpolation over the whole domain

Interpolation over the whole domain is done with the Agrif_Interp routine
Same arguments than those of Agrif_Bc_Variable exist
Agrif_Init = call to Agrif_Interp + call to Agrif_Bc

Treatment of masked fields

Agrif_SpecialValue, replacement of masked values by neighboring values

Agrif_SpecialValue

Agrif_UseSpecialValue= .true.
Agrif_SpecialValue = Val
must be set before the call to Agrif_Bc_Variable, Agrif_Interp

Examples

```
Agrif_UseSpecialValue=.true.
Agrif_SpecialValue=0.
Call Agrif_Bc_Variable(tabres,variable)
Agrif_UseSpecialValue=.false.
```

Remark

Setting maximum lookup of masked values (since it may affect performance)
Call Agrif_Set_MaskMaxSearch (mymaxsearch)
Defaut Value: MaxSearch = 5

Outline

Main procedures involved

Agrif_Set_Update	Type of update
Agrif_Update_Variable	Make the restriction

Restriction: how to specify the type of restriction ? for each direction

Agrif_Set_UpdateType

```
Agrif_Set_UpdateType(update = Agrif_Update_Type)
```

Examples

```
1D
Call Agrif_Set_UpdateType(update = Agrif_average)
2D
Call Agrif_Set_UpdateType(update = Agrif_average)
Call Agrif_Set_UpdateType(update1 = Agrif_Copy, update2 = Agrif_Average)
```

The call: Agrif_Update_Variable

Agrif_Update_Variable

Agrif_Update_Variable(tabtemp,variable)

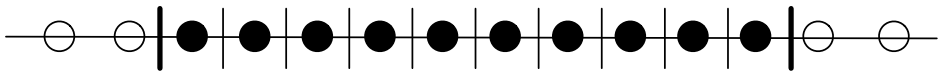
Example

Call Agrif_Update_Variable(u,u)

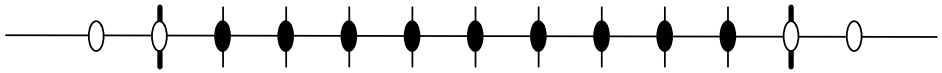
Updates: how to specify where to update ?

Default location: interior of the reference grid

Centered points:



NonCentered points:



Updates: how to specify where to update ?

Update on limited areas

Agrif_Update_Variable

Agrif_Update_Variable(variable,locupdate=)

Examples

Centered points:

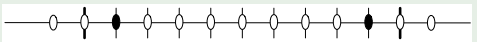
locupdate=(/0,0/) 

locupdate=(/1,1/) 

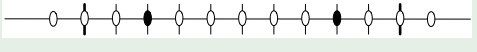
locupdate=(/0,1/) 

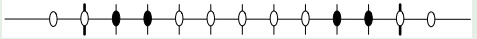
locupdate=(/1,0/) 

Non centered points:

locupdate=(/0,0/) 

locupdate=(/1,1/) 

locupdate=(/0,1/) 

locupdate=(/1,0/) 

Remark

Multidimension locupdate1=... , locupdate2 = ...

Update schemes

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

The use of procnames

Agrif_Update_Variable

Agrif_Update_Variable(tabtemp,variable/variable_id,procname)

Examples

```
Call Agrif_Update_Variable(tabtemp,variable,procname=Update_MyTraceur)

Subroutine Interp_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 = My_traceur(i1:i2)
Else                                      ! on the parent grid
   My_traceur(i1:i2) = tabres
Endif

End
```

Treatment of masked fields

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

Agrif_SpecialValue

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(tabtemp,variable)
Agrif_UseSpecialValueInUpdate=.false.
```

Outline

Distributed memory

How to make your MPI code work with Agrif ?

In your Makefile

```
CPPFLAGS += -DAGRIF_MPI
```

MPI initilization

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

Distributed memory

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

Agrif_Invloc

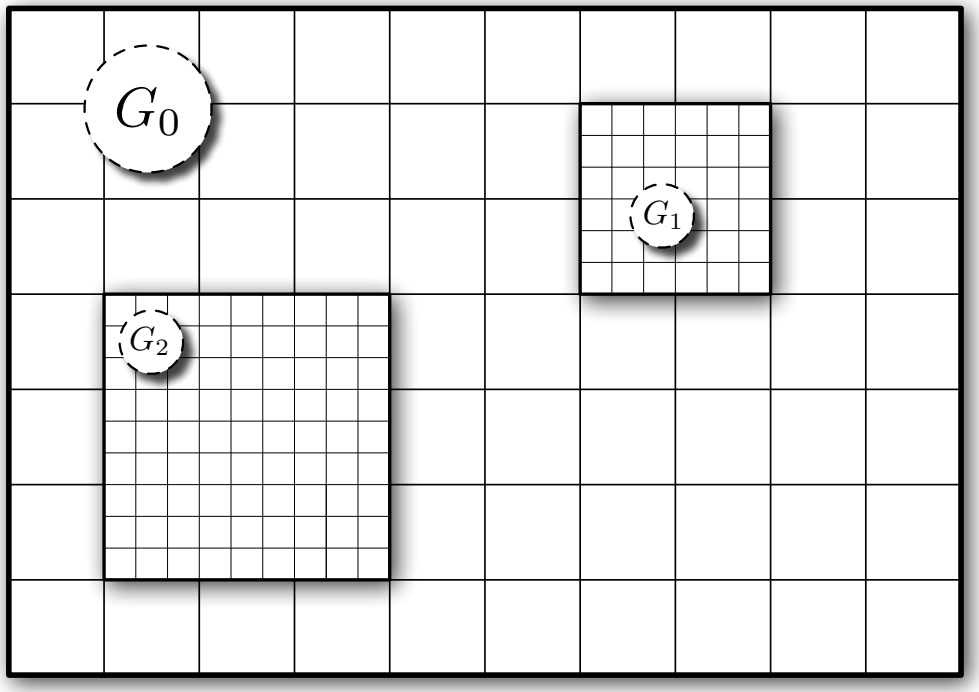
```
subroutine Agrif_Invloc ( indloc, procnum, dir, indglob )

    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
```

Distributed memory

Currently, all grids are integrated sequentially:

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

For each grid, all processors are used.



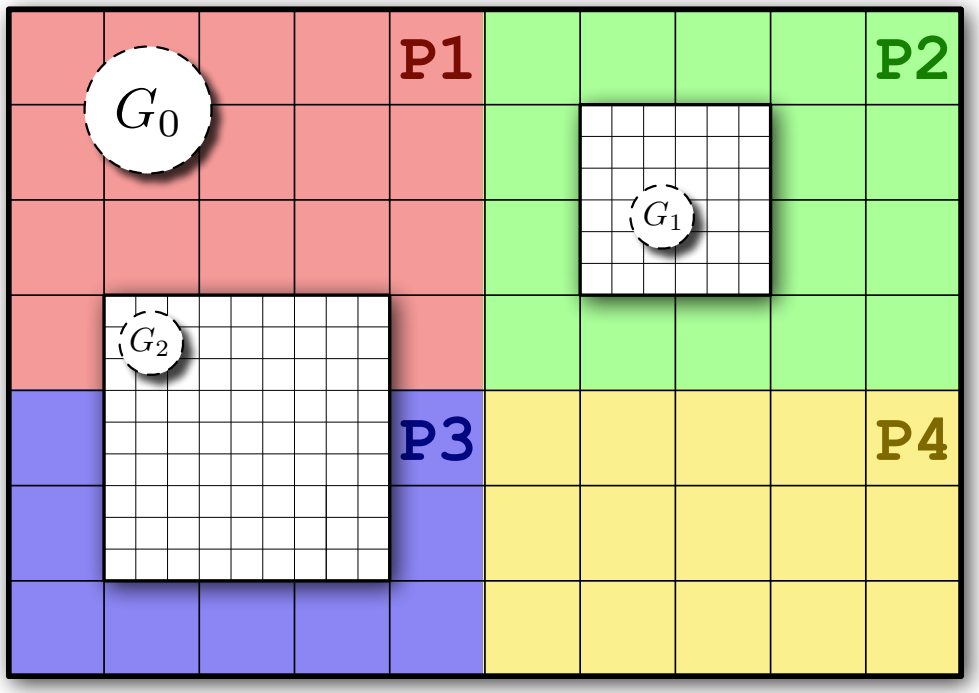
Distributed memory

Currently, all grids are integrated sequentially:

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

For each grid, all processors are used.

i_1 :



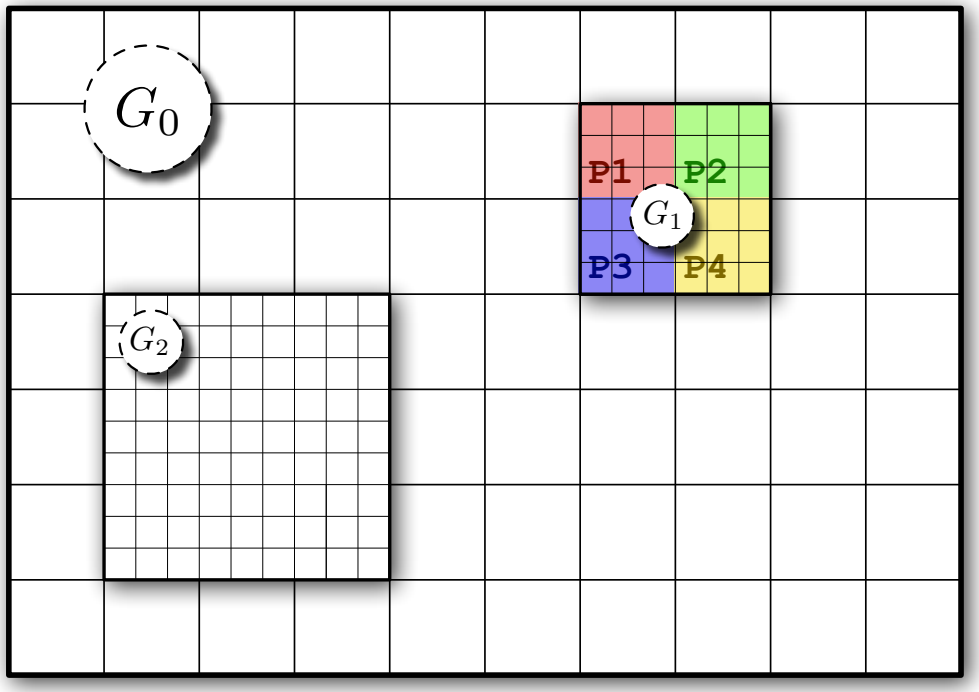
Distributed memory

Currently, all grids are integrated sequentially:

sequence	i_1		i_2		i_3		
grid	G_0	$\rightarrow$	G_1	$\rightarrow$	G_2	$\rightarrow$	$\ldots$

For each grid, all processors are used.

i_2 :



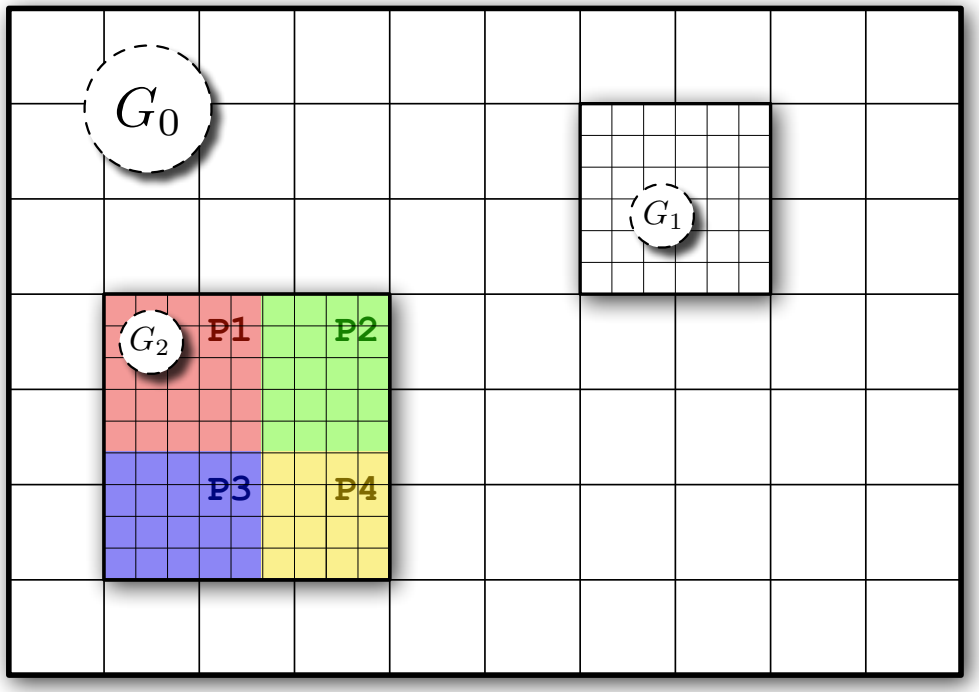
Distributed memory

Currently, all grids are integrated sequentially:

sequence	i_1		i_2		i_3		
grid	G_0	$\rightarrow$	G_1	$\rightarrow$	G_2	$\rightarrow$	$\ldots$

For each grid, all processors are used.

i_3 :



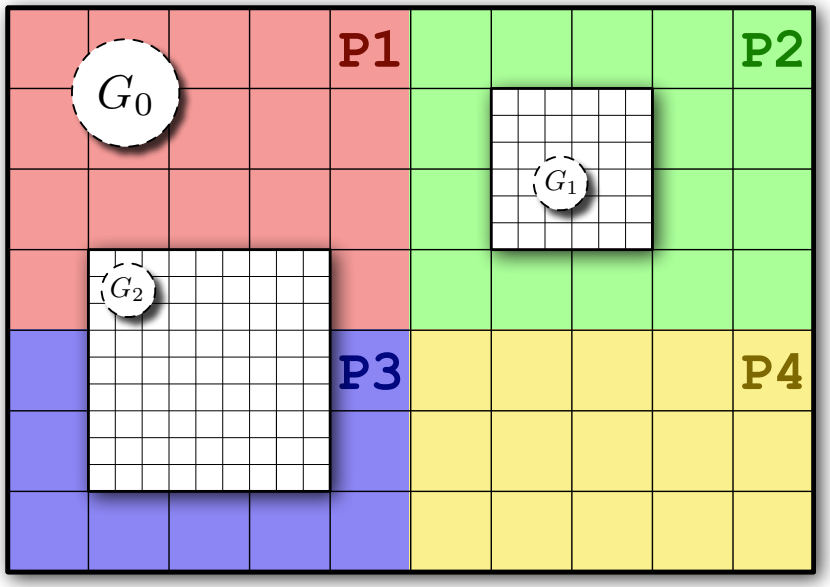
Distributed memory

Sister grids (same parent) could be integrated concurently:

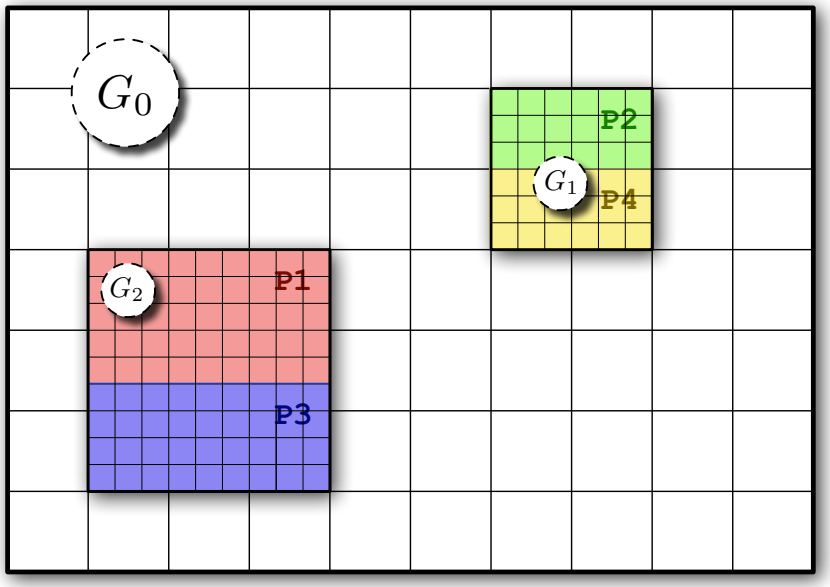
$$\begin{array}{r|lcl} \text{sequence} & i_1 & & i_2 \\ \text{grid} & G_0 & \rightarrow & G_1 \;//\; G_2 \quad \rightarrow \quad \ldots \end{array}$$

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

sequence i_1



sequence i_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

```
!$OMP MASTER
  Call Agrif_Bc_Variable(u,u)
!$OMP END MASTER
```

Outline

Outline

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)
```

Outline

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
```

Outline

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

Outline

Outline

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.

Flux correction (II) (ROMS)

How to implement a flux correction procedure ?

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

1. Declare flux arrays (F_x, F_y), compute then and make the summation on the fine grids over a parent time step
2. Declare the profiles corresponding to these fluxes
   Call Agrif_Declare_Profile((/1,2/),(/1,1/),(/'x','y'/),(/0,1/),(/nx,ny/),$F_{x\mathrm{id}}$)
   Call Agrif_Declare_Profile((/2,1/),(/1,1/),(/'x','y'/),(/1,0/),(/nx,ny/),$F_{y\mathrm{id}}$)
3. 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)
4. Make the flux correction inside a call to Agrif_Update
   Call Agrif_Update_Variable(tabtemp, $F_{x\mathrm{id}}$, procname = Correct_Flux_x)

Flux correction (III) (ROMS)

How to implement a flux correction procedure ?

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

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
```

Outline

Wetting and drying (I) (MARS)

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

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
```

Outline

Different vertical grids (I) (NEMO)

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

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
```

Different vertical grids (II) (NEMO)

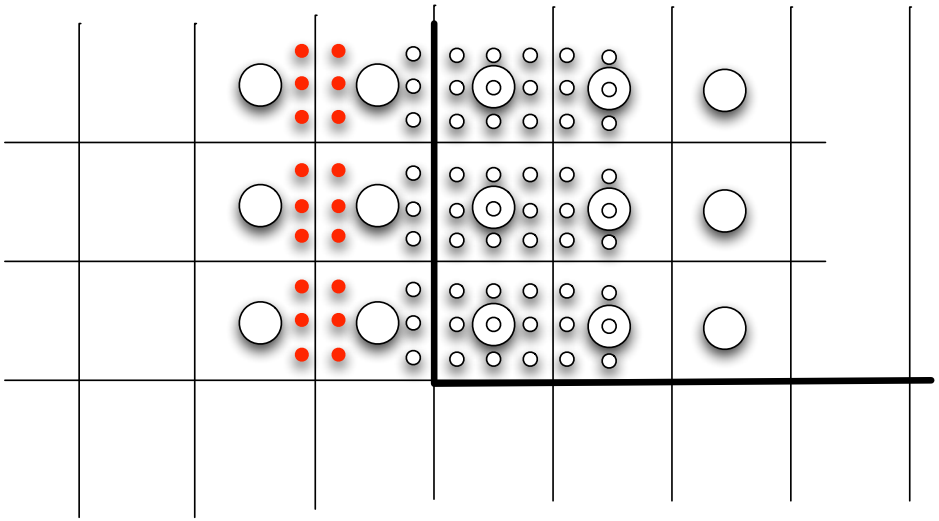
The grids have different vertical mesh. The vertical remapping between the two grids are done in the procnames routines.

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
```

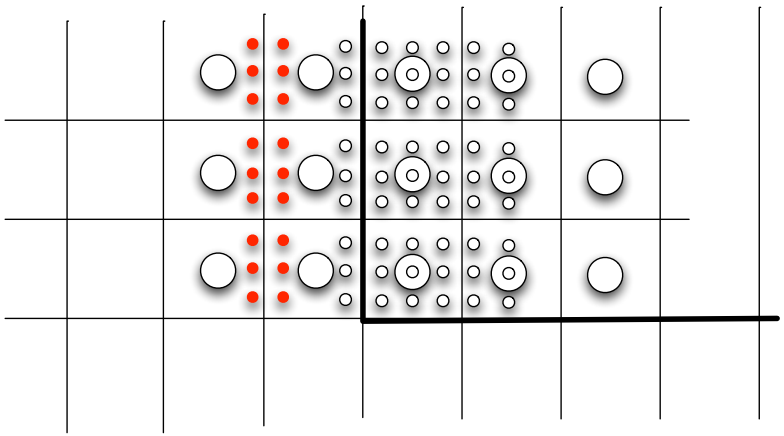
Outline

Tracer interpolation (I) (NEMO)



I want to use internal fine grid values during the interpolation

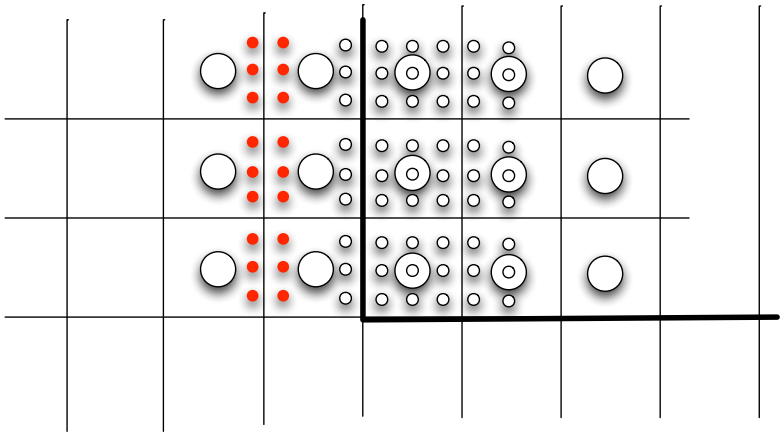
Tracer interpolation (II) (NEMO)



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)
```

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
```