AGRIF 库介绍(Laurent Debreu et al., 2008;C&G)

1 前言

自从 Berger and Oliger (1984)将 AMR 结构网格引入求解偏微分方程以来，AMR 变得流行，AMR 技术广泛应用于各个领域。但是，实施 AMR 方法仍然是困难的，不利于其推广应用。

将 Berger and Oliger (1984)的 AMR 方法称为 BOSAMR。AMRCLAW 是基于 BOSAMR 方法，针对结构网格的 AMR 算法的 FORTRAN 77 程序，目前 AMRCLAW 已包含在 CLAWPACK 内。面向对象语言编程使 BOSAMR 方法的实施变得更为简单，已开发了很多程序包，但大多都是使用 C++语言，例如 Chombo 和 Overture。这些成熟使用 C++的类(class)处理网格定义和网格操作。

类似的 FORTRAN 90 的 AMR 程序有 PARAMESH，是基于单个单元细化的程序(one cell-based)，该方法可以有效的并行化，非常灵活地做网格细化，但是与 BOSAMR 算法不同，BOSAMR 算法是使用矩形补丁(patch-based，由若干单元组成)，在补丁网格上可实施原始均匀网格上的代码计算。

AGRIF 程序是使用 FORTRAN 90 语言实施 BOSAMR 方法的程序，特点是使用指针和充分利用 FORTRAN 77 与 FORTRAN 90 之间的完全兼容性。

2 BOSAMR 算法简介

主要思想就是：使用数学或物理的准则，局部调节数值求解的分辨率，从而在需要的位置和时刻，创建细化网格（或删除原有的网格单元）。AMR 方法采用分级的分辨率级别（a hierarchy of resolution levels），各层包含一套网格，如图 1。

每个网格都由父级别（粗的）网格覆盖，根级别网格由覆盖整个计算域的粗分辨率网格组成。当从某级别 l 过渡到细网格级别 l+1，时空分辨率除以整数 r，一般 r = 2, 3, 4。

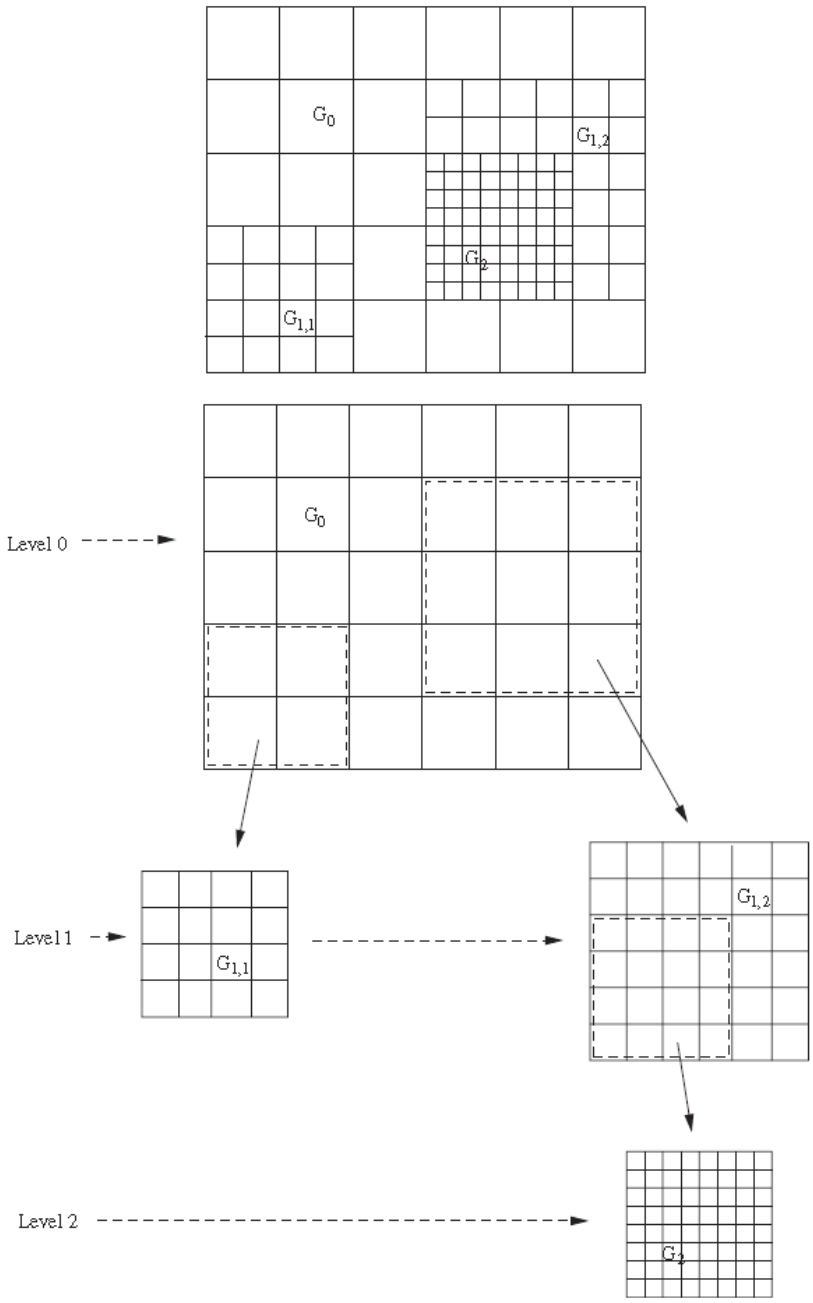


图 1 分级网格示意图：1 个根网格 G_0，有 2 个子网格 $G_{1,1}$ 和 $G_{1,2}$；在第 2 层细化网格 G_2 是 $G_{1,2}$ 的一个子网格。

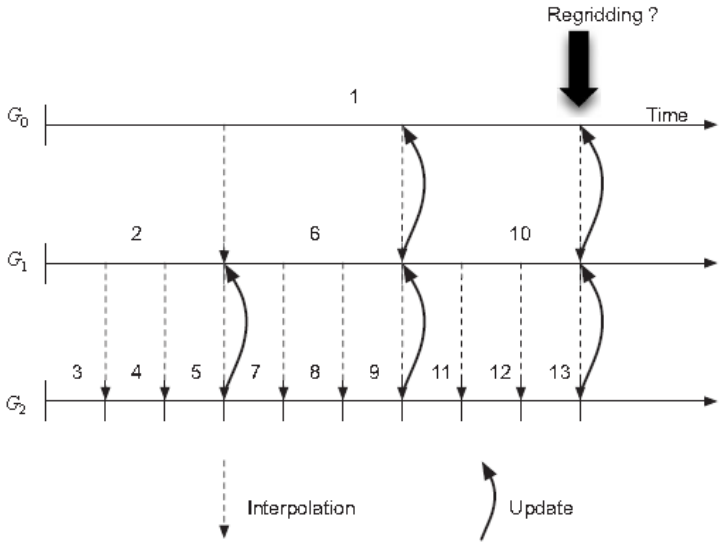
2.1 分级网格上的时间积分

分级网格的积分从根级别分层开始。根级网格上的数值解首先以粗的时间步长 Δt_0 推进。该数值解将作为 l 层网格的边界条件（通过插值得到），以时间步长 $\Delta t_1 = \Delta t_0 / r$ 推进 r 步，依次在更细化网格层递归计算。一旦在某层级网格上计算得到了数值解，将用于更新其父级别网格上的数值解。

积分算法见图 2，是在 0 层级网格上进行的递归计算步骤。

```
Recursive Procedure INTEGRATE(l)
  If l == 0 Then nbstep = 1
  Else nbstep = refinement-ratio
  Endif
  Repeat nbstep times
      Do one time step Δt_l on all grids at level l
      If level l+1 exists Then
          Compute boundary conditions at level l+1
          INTEGRATE(l+1)
          update level l
      Endif
  End Repeat
  End Procedure INTEGRATE
```

(a) 积分算法



(b) 时间细化因子为 3, 3 个网格 G1, G2, G3。图中的数字表示积分顺序

图 2 积分算法与示意图

2.2 网格插值(regridding)

在每个粗时间步或以规则时间间隔，改变网格分级。实施网格细化准则，就是侦测哪个位置的网格节点上需要细化网格或当前细网格是不必要的（粗化）。AGRIF 实施堆叠算法(clustering algorithm) (Berger and Rigoutsos, 1992)。

3 AGRIF 介绍

3.1 用 Fortran90 语言实施 AMR

AGRIF 的设计目标就是尽可能利用已有的 FORTRAN 代码。为此，使用指针的概念。指针变量不是 FORTRAN 77 语法，但是 FORTRAN 90 语法。

一个指针变量可以在计算阶段位于不同的内存位置，这是 BOSAMR 算法需

要的，当从一级网格到另一级网格计算时。AGRIF 使用一套特殊的指针，保存在公共区块(common block)，连续访问内存空间，内存位置对应不同级别网格上的变量，访问顺序就是积分算法中预设的顺序。

如图 3，令 u 和 v 是原始数值模型中的 2 个变量，定义在公共区块。在自适应版本的模型中，u 和 v 定义为指针，各级别的网格表示为一套衍生类型变量，其包含该网格上的局部变量 u 和 v。然后，在各时间步上各网格上做积分，指针 u 和 v 连接到网格上对应的局部变量。

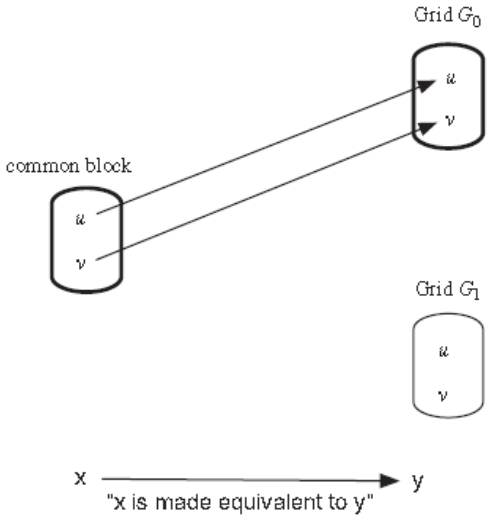


图 3 指针实例化：箭头表示实例化过程，一旦完成，访问公共块中的 u, v 就等价于访问网格 G_0 上的对应变量

3.2 实施过程概览

AGRIF 软件由 2 部分组成：model-independent 和 model-dependent。

（1）与模型不相关（独立的）部分：实施算法中的时间积分、堆叠算法、插值和更新格式等。model-independent 部分在 FORTRAN 90 modules 的库中实施。

（2）与模型相关的部分：新代码必须可以处理动态变化的内存分配、之前定义指针的实例化，等。这部分代码与原始模型的变量有关。

AGRIF 还包括一个外部程序：用 C 语言和 Yacc/Lex 语言编写的转换程序，分析用于的手写配置文件，生成 BOSAMR 的模型相关部分代码。在配置文件中，由用户提供 BOSAMR 的全局参数，包括：网格细化因子、网格分级的最大分层数、网格插值间隔，等；以及网格变量和对这些变量实施的网格操作。

每个操作都有对应的关键词(keywords)，每个关键词涉及插值或更新操作，转换程序生成 FORTRAN 子程序。具体细节参考手册说明。

编译的具体过程见图 4。在原始模型中，主要的修改就是：加入对 AGRIF 过程(prodecure)的调用，名称为 Agrif_Step，该子程序调用分级网格的时间积分算法，以规则时间间隔继续计算，直到网格插值步(regrdding step)。

编译过程为：

第①步：转换程序(CONV)生成模型相关部分（模型变量的动态空间分配、指针的实例化），修改原始模型代码的声明文件（将静态声明变量改为指针）；

第②步：编译模型相关和模型无关的部分，生成静态链接库；

第③步：使用静态链接库和第 1 步的修改声明文件，编译原始的 FORTRAN 文件，生成多分辨率模型代码。

多分辨率模型代码中，可以使用第 1 步中根据配置文件生成的插值/更新过程(procedures)。

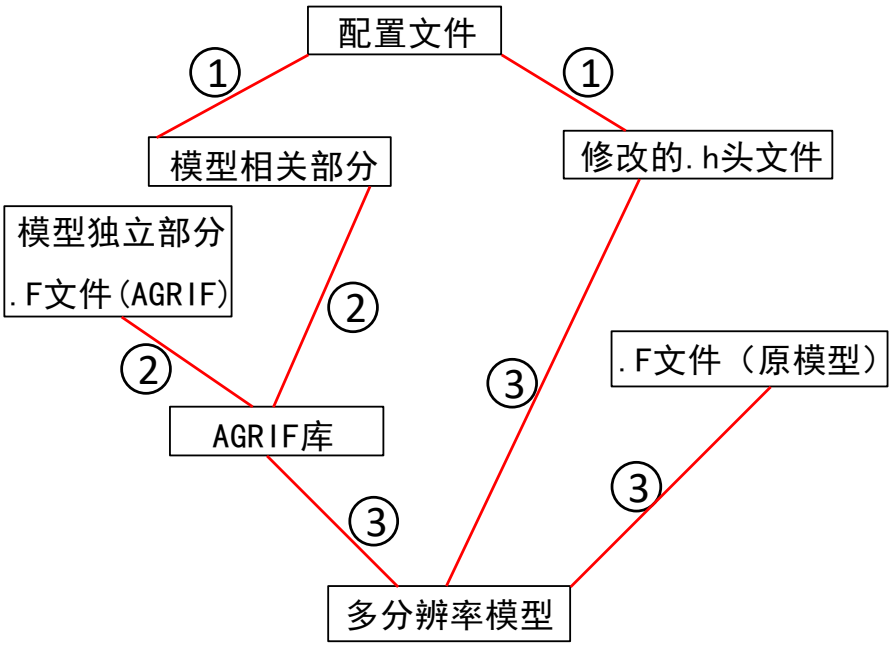


图 4 编译过程概览

AGRIF 的主要特征

主要特征有：

（1）可以实施 1D，2D 和 3D 自适应网格细化；

（2）可处理交错网格；

（3）可处理 masked 区域；

（4）可使用预定义固定位置的网格；

（5）MPI 并行化。

海洋模拟应用见 Blayo and Debreu (1999)和 Debreu et al. (2005)。

参考文献

Laurent Debreu, Christophe Vouland, Eric Blayo. 2008. AGRIF: Adaptive grid refinement in Fortran. Computers & Geosciences, 34: 8−13.

Blayo, E., Debreu, L., 1999. Adaptive mesh refinement for finite difference ocean models: first experiments. Journal of Physical Oceanography, 29: 1239−1250.

Debreu, L., Blayo, E., Barnier, B. 2005. A general adaptive multiresolution approach to ocean modelling: experiments in a primitive equation of the north Atlantic. In: Plewa, T., Linde, T., Weirs, V.G. (Eds.), Adaptive Mesh Refinement-Theory and Applications. Lecture Notes in Computational Science and Engineering, vol. 41. Springer, Berlin, pp. 303−314.

Agrif_User_Guide_v1.3-2006

1 AGRIF 介绍

AGRIF 软件的主要思想就是：将固定网格或自适应网格加密技术，引入使用 FORTRAN 语言编程的结构网格离散的已有模型。软件分两部分：

（1）一个 source-to-source 的代码转化程序(CONV)，可以将单网格模型转换为多分辨率网格(online nested grid)代码；

（2）一个静态链接库，实施网格交互。

实现 AMR 技术的可选择程序：

（1）手写代码，如 ROMS-Rutgers;

（2）RSL 软件，如 MM90, WRF, pPOM

（3）通用软件：PARAMESH (FORTRAN 90); SAMRAI (C++); Chombo (C++)

使用 AGRIF，用户需要：

（1）写一个包含模型描述的配置文件，在编译代码时，CONV 读取该配置文件；

（2）定义在哪个位置定义 fixed grid 的文件；

（3）agrif_user.F90，包含用户子程序；

（4）agrif2model.F90，链接模型与 agrif 库（该文件不能被修改）

一个典型的代码组织结构：

```
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
```

代码见手册或 tutorial_AGRIF

1.1 配置文件

（1）全局参数

网格细化维度和计算域大小

3D nx,ny,nz;

2D nx,ny;

（2）参数文件

paramfile name;

parammodule modulename;

（3）使用固定网格

USE FIXED_GRIDS

（4）使用 only fixed 网格，意味着仅使用一些固定网格，没有移动网格，定义在 Agrif_Fixedgrids.in

USE ONLY_FIXED_GRIDS;

（5）不是与网格有关的变量，这些变量在各网格上都一样，不存储在 tabvars

notgriddep variable_name;

1.2 agrif_user 子程序

在该文件中需要定义几个子程序：

- Agrif_InitWorkspace
- Agrif_InitValues
- Agrif_detect

这些子程序定义如下：

Agrif_InitWorkspace

用户需要定义所有来自细网格上的网格维度的、参与计算的变量，例如

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

endif

如果配置文件中定义了 AMR，所有 AMR 参数需要在这个子程序中定义：

Efficiency，默认 70%

Call Agrif_Set_Efficiency(0.8)

Regridding：每 N 步，做一次 Regridding

Call Agrif_Set_Regridding(10)

Space refinement factors: 空间细化因子

Agrif Set coeffref x, Agrif Set coeffref y and Agrif Set coeffref z.

Call Agrif Set coeffref x(3) / n : integer /

Time refinement factors

Call Agrif Set coeffreft x(2)

Call Agrif Set coeffreft y(2)

Call Agrif Set coeffreft z(2)

Minimum width

Call Agrif Set minwidth(5)

Maximum number of refinements

Call Agrif Set rafmax(5)

Agrif_InitValues 子程序

该子程序在每个细网格上调用一次。

首先，我们应该调用初始化当前网格的子程序（相同的子程序允许初始化粗网格）。

Call init()

如果一个变量需要一些特殊处理（初始化、边界插值、变量 Restoration），这个变量必须“完全声明”。意思是：变量类型、计算域内的第 1 个点的标记以及计算域维度的类型，必须声明。

调用如下子程序：

1、变量的位置

通过调用 Agrif_Set_type，定义在网格上变量的存储位置。允许使用交错网格。对各空间方向，使用 1 表示单元的边界（图 1.1），使用 2 表示单元中心（图 1.2）。注意：仅当定义了空间方向，才可以在该空间方向上使用交错网格变量。

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

接下来，需要声明计算域内的变量的第一个点的编号。

最后，用户必须指明各维度的类型，使用子程序 Agrif_Set_raf：x,y,z 为空间维度，0 表示没有空间维度。

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

p1,p2,p3 表示网格变量的位置；i1,i2,i3 表示计算域内第一个点的编号。

对于一个 2D 计算域，在一个单元上定位变量有 4 种不同方式：

```
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'/))
```

图 1.3 显示在一个单元上的这些位置。

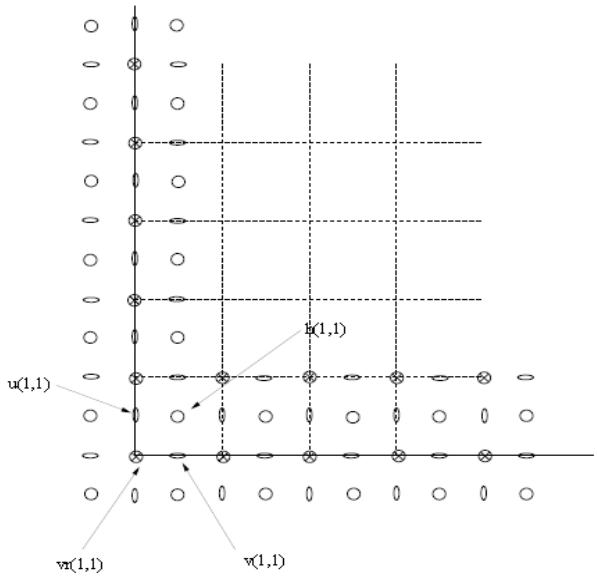


图 1.3 计算域中的第 1 个点

2、插值

AGRIF 可以超值当前网格的变量，使用子程序 Agrif_Set_interp 来声明。

```
Call Agrif_Set_interp(name, interp=method)    / method 为插值方法 /
```

例如：

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

3、边界插值

AGRIF 以相似方式，提供如果需要插值边界数值的子程序。在使用子程序 Agrif_Set_bc 之后，必须给出需要插值变量的名称，使用描述如：(ideb:ifin)或者(iind)，最后的编号指示插值到哪个位置。

对于使用 Agrif_Set_interp 已经声明的变量，边界校正中使用的默认插值方

法，与调用 Agrif_Set_interp 中使用的插值方法相同。

如果变量没有使用 Agrif_Set_interp 声明，或者如果需要对边界做其他类型的插值，需要调用 Agrif_Set_bcinterp。

Call Agrif_Set_bc(var,(/ideb:ifin/)) / ideb, ifin : integers /

Call Agrif_Set_bcinterp(var,method) / method = linear, lagrange, split /

图 1.4 表示：（在 2D 域，对于不同类型的变量），如果指定了(-1)，将校正这些位置。

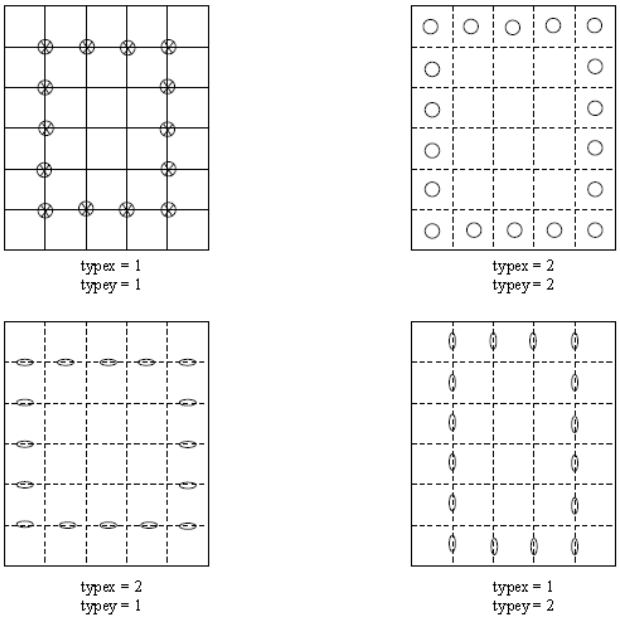
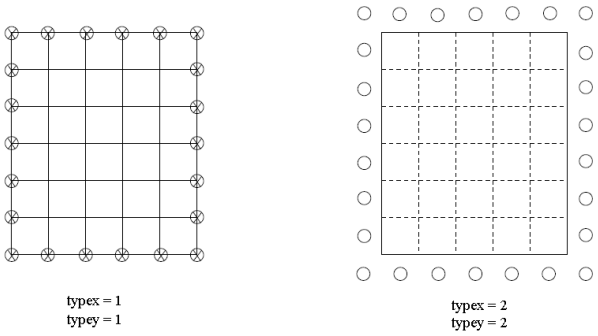


图 1.4 网格校正，BC:: var : (-1)

图 1.5 表示：（在 2D 域，对于不同类型的变量），如果指定了(0)，将校正这些位置。



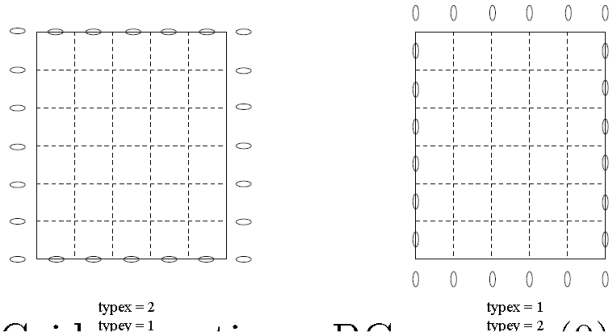


图 1.5 网格校正， BC:: var : (0)

图 1.6 表示：（在 2D 域，对于不同类型的变量），如果指定了(1)，将校正这些位置。

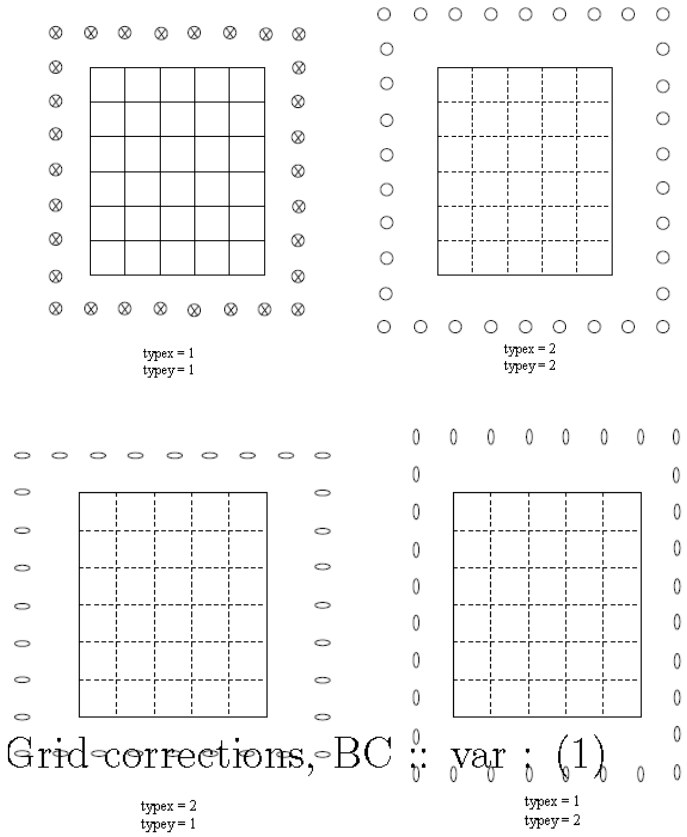


图 1.6 网格校正， BC:: var : (1)

4、更新网格变量。AGRIF 能够从父级别网格更新子网格上的变量值。

AGRIF 可以更新当前网格的父级别网格上的变量，使用 Agrif_Set_UpdateType 声明。

Call Agrif_Set_UpdateType(var, update=method)　　/ method 为更新方法 /

例如：

Call Agrif_Set_UpdateType(u, update=Agrif_Update_Average)

5、Restoration (仅 AMR)

在网格细化阶段，对每个变量，为了对新创建网格的初始化，用户可以定义必须恢复的变量（如果可能的话）。调用 Agrif_set_restore 来实现。

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

Agrif_detect 子程序

用来侦察需要细化的点。

在需要做 AMR 的位置处，执行该子程序。

1.3 Agrif2Model 子程序

CONV 程序将创建新的 FORTRAN 代码和一些文件（在 AGRIF_INC 路径下），这些文件用于完成 agrif2model 文件。

因此，在 makefile 中，应该在编译阶段结束时（在编译完最后一个文件）编译 agrif2model。

1.4 特殊操作

1.4.1 用于插值的特殊数值

在调用子程序 Agrif_Interp_VarName 或 Agrif_Bc_VarName，可以通过这些插值，为父网格和细化网格的数组定义使用一些特殊值。

父网格的必要关键词是：Agrif_UseSpecialValue 和 Agrif_SpecialValue

细网格的关键词是：Agrif_UseSpecialValueFineGrid 和 Agrif_SpecialValueFineGrid

Agrif_UseSpecialValue 和 Agrif_SpecialValueFineGrid 是两个逻辑值 (Logicals)，而 Agrif_SpecialValue 和 Agrif_SpecialValueFineGrid 是两个实数值 (Reals)。

父网格上的特殊值

如果需要使用父网格上的一个特殊值（如 98.8），则在调用 Agrif_Interp_VarName 和 Agrif_Bc_VarName 子程序之前，必须添加如下几行代码：

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

然后，如果用于细网格空间插值的父网格部分的一个值等于 98.8，该值将被

父网格上不等于 98.8 的最邻近数值代替。

细网格上的特殊值

如果需要使用细网格上的一个特殊值（如 50.0），则在调用 Agrif_Interp_VarName 和 Agrif_Bc_VarName 子程序之前，必须添加如下几行代码：

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

然后，如果在细网格上的一个值等于 50.0，则该值在插值期间不会被代替。

1.4.2 得到父级别网格上的数值

有时用户需要直到当前网格的父网格上的数值（例如，用于自己程序的插值和更新）。AGRIF 提供这个 API。

如何获得父网格上的数值？

为获得父网格上的变量值，可调用 Agrif_Parent 函数：

例如：Agrif_Parent(u)

对于标量变量：

```
Real :: dtp
dtp = Agrif_Parent(dt)     ！父网格上的计算时间步长
```

对于多维变量（矢量）：

```
Real, Dimension(:,:), pointer :: parent_u
parent_u⇘⊤Agrif_Parent(u)

parent_u(i, j) = 3.0
```

细网格和父网格节点之间的对象关系是什么？

对于非交错变量，父网格上的一个节点对应子网格上的一个节点，不管空间细化因子是多少；对于交错变量，使用奇数的空间细化因子，结果也是一样的。使用偶数空间细化因子的交错变量情况，则不存在这种对应关系，如图 1.7 所示。

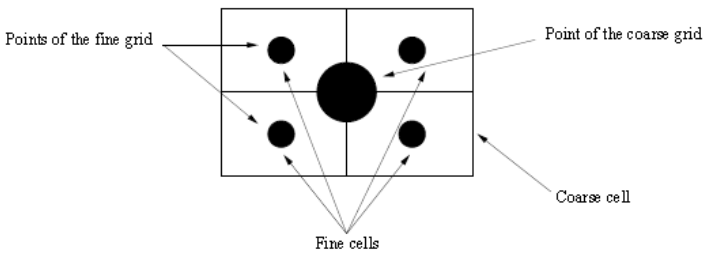


图 1.7 使用空间细化因子 2 的交错变量

细网格和粗网格节点的对应关系用下面的算法给出：

Do i = $ideb_f, \ldots, coeffraf$

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

End Do

其中，$ideb_f$ 和 $ideb_g$ 与变量的类型（交错或非交错）有关：

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

其中，ptx 是计算域内部的第一个节点的编号；ix 是当前网格在其父网格上的最小位置。

1.4.3 在 AGRIF 中使用固定网格

AGRIF 库的一个特点是，可以使用计算域内的固定细网格。

首先，用户必须在配置文件(amr.in)中插入关键词 USE FIXED_GRIDS；如果仅需要使用固定网格，无需移动（AMR）网格，还必须添加关键词 USE ONLY_FIXED_GRIDS (if !defined AGRIF_ADPTATIVE)。

接下来，必须在一个文件（AGRIF_FixedGrids.in）中声明细网格，该文件必须放置于代码运行的当前路径下。

对各细网格和各空间方向上，需要一行字符指示其在父网格上的位置、空间和时间细化因子。

语法：

•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 是 6 个整数，表示固定网格的最小和最大位置（节点编号）。该定义中，Corner 节点的位置涉及父网格折角的位置，在各方向上从 1 开始（FORTRAN 风格）。当用户写出这些位置时，必须注意：各个细网格包含在一个父级别固定网格中。

spacerefx spacerefy spcerefz timerefx timerefy timerefz 是 6 个整数，表示在 x,y,z 方向上空间和时间细化因子。如果其中一个因子等于 1，表示该空间方向上

不做网格细化。

例子：

2 维的示例如图 1.8. 根级别粗网格(G0)有 2 个子网格(G1, G2)，G1 有 1 个子网格 G3. 空间和时间细化因子在 2 个方向上都等于 2。

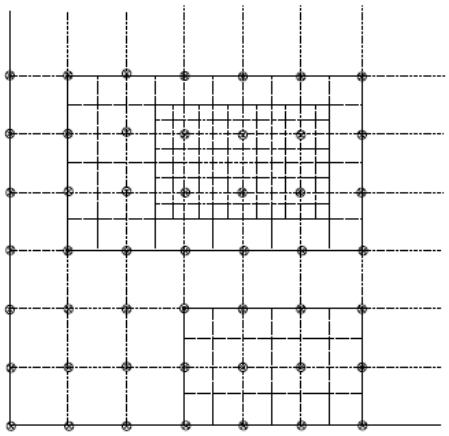


图 1.8 固定网格示例

对于该例，AGRIF_FixedGrids.in 文件必须包含如下几行文字：

```
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)
```

使用自适应加密(AMR)的固定网格

当在配置文件中使用关键词 USE FIXED_GRIDS，而不使用关键词 USE ONLY_FIXED_GRIDS，则在分级网格中同时拥有固定网格和移动网格。

2 在多维有限差分模型中使用 AGRIF

2.1 运行 AGRIF 的必要子程序

2.1.1 AGRIF 提供的子程序

Agrif_Init_Grids

该子程序创建根级别粗网格(root coarser grids)的特征量（单元数目，。。。）。在完成粗网格上的单元数的初始化后，在主程序中调用该子程序。

Agrif_Step

该子程序调用和代替用户编写的的时间积分步（**Step**）。它在所有网格上向前时间积分，以及以规则时间间隔实施网格插值（在自适应加密时）。

2.1.2 用户编写的子程序

Agrif_Initvalues

该子程序初始化细网格变量，在细化过程中调用。

Agrif_Initworkspace

该子程序，在每个空间方向上，包含在参数文件(parameter file)中声明的变量之间的连接(link)、表示计算域大小和单元数。

例如，如果 nx 是网格单元数，nxp1 是网格节点数，则在该子程序中必须编写 nxp1=nx+1 的语句。（???）

在完成网格修改之后调用 Agrif_Initworkspace，重新计算对应的数值。

Agrif_Detect

如果在自适应模式下使用 AGRIF，必须调用该子程序。它允许编写自适应网格加密准则。

2.2 如何编译 AGRIF 和你的模型？

2.2.1Makefile 的修改

创建 AGRIF key

为确定在当前模型中是否使用 AGRIF，应该创建 AGRIF key，称为 key_agrif

CCP_FGLAGS = -Dkey_agrif

该 key 允许：

（1）在预处理阶段，可以编译或删除 AGRIF 源码，例如：

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

（2）在 Makefile 中确定是否使用 AGRIF：

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

编译 AGRIF 库

AGRIF 路径应置于代码 tree 下。在 makefile 中，应首先编译 AGRIF 库，创建 libagrif.a 文件，供后面使用。编译 AGRIF 库，用户应使用 AGRIF 内的 Makefile。

使用 CONV

在对象列表的尾部增加 Agrif_User.F90 的编译。现在我们应该使用 CONV 程序创建 AGRIF 兼容的源码。

接下来，编译每个文件：

（1）应该将该文件移动到其他路径下，为了不删除原始文件情况下，编译该文件；

（2）对该文件应用 conv，创建一个新的 FORTRAN 文件；

（3）编译这个新文件。

例如，使用一个路径称为 NEW_FILES。编译每个文件，使用：

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

请参考 Makedefs.generic.AGRIF 中的命令及参数。

编译 Agrif2Model 文件

在做最后的连接之前，用户应在 AGRIF_INC 路径下由预处理产生的文件的帮助下，完成 Agrif2Model 文件。

2.2.2 模型中的修改

下面介绍原始模型代码中所必须做的修改，使其能够运行 AGRIF 程序。

修改参数文件（CROCO 见 cppdefs.h））

该文件必须包含计算域大小和与大小有关的变量。比如，在原始的 common 区，声明是：

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

修改为：

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

修改主程序

为运行 AGRIF，应将原始的时间积分过程的调用替代为'call AGRIF_Step(step)'（见 2.1.1 节）。

还应该增加一个调用 Agrif_Init_Grids（见 2.1.1 节）。该调用必须在单元个数的初始化之后完成。

输出一些过程

应该输出 Agrif_Initvalues 和 Agrif_Initworkspace 过程（见 2.1.2 节）。

2.2.3 如何使用 AGRIF 程序包

完成以上的修改后，用户还要编写配置文件（见 1.1 节）和编译程序库。

编写 AGRIF_FixedGrids.in 文件

见 1.4.3 节中介绍的文件，如果需要使用固定网格，该文件是必须的。当运行模型时，该文件必须位于主路径下。

编写配置文件

1、数一下在 X，Y 和 Z 方向上的单元数目；

2、获得参数程序名。

Agrif_User_Guide_v2.0 (Laurent DEBREU and Roland PATOUM)

第 1 章 在不同网格上运行代码

1 代码转换工具：conv

（1）原始的“单网格”代码必须重写，使其与网格无关(grid-independent)；

（2）代码转换程序：**conv**

（3）全局变量，由 Agrif 库内部 API 处理

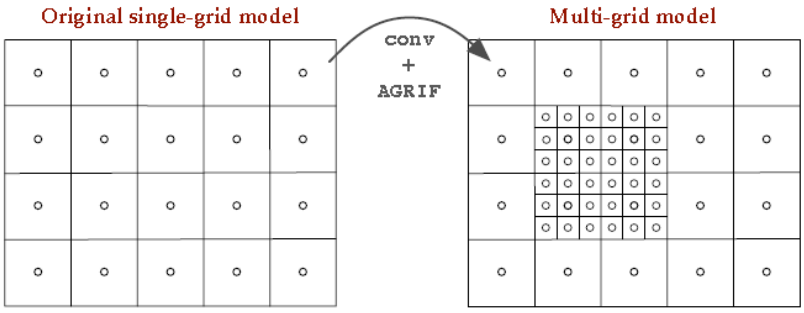


图 1.1 代码转换工具 conv

2 典型的编译顺序

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
```

第 2 章 一个 AGRIF 代码的典型结构

主程序：

```
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 creat
    enddo
end program example
```

2.1 主要的 AGRIF 过程(procedure)

必须调用的子程序：

Agrif_Init_Grids　代码中首先调用的子程序：初始化 AGRIF 库；

Agrif_Step(step)　在所有网格上向前积分模型，递归调用 step()。

选择性调用的子程序：

Agrif_Regrid()　读取 AGRIF_Fixed_Gris.in，构建网格数据结构；

Agrif_Step_Child(step) 在各网格上递归调用 step()，不使用时间细化；

Agrif_Step_Childs(step) 在当前父网格的各子级别网格上递归调用 step()，不使用时间细化；

Agrif_Integrate_ChildGrids(step) 在当前父网格的各子级别网格上递归调用 step()，使用时间细化；

用户必须编写的子程序：

Agrif_InitWorkspace

定义当前工作空间的维度。程序库需要修改当前网格时，都要调用一次。

```
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

对每个细化网格都调用一次的初始化子程序。一般用来：

（1）为插值和限制(restriction)操作声明 Agrif 参数情况(profile)；

（2）初始化变量（如：从文件读取数据，或者从粗网格插值）

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

注意：第一次调用 Agrif_Step 中调用 Agrif_Regrid 和 Agrif_InitValues 时，是作用于整个网格，初始化所有变量。

2.2 分级网格的积分

涉及分级网格积分的子程序有 4 个：Agrif_Step, Agrif_Step_Child, Agrif_Step_Childs and Agrif_Integrate_ChildGrids

（1）Agrif_Step(step)

在各级别网格上递归调用 step()，考虑时间细化因子。

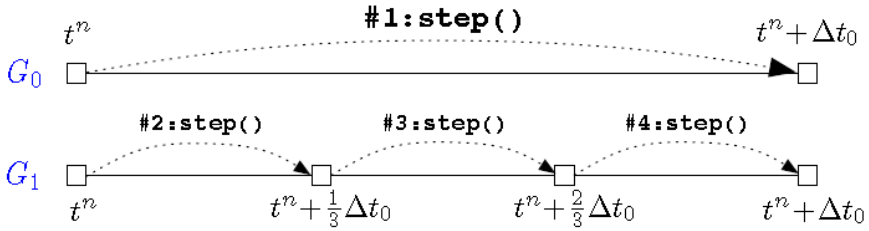


图 2.1 使用根级别网格 G0 和一个子级别网格 G1 的 Agrif_Step 子程序（时间细化因子为 3）

（2）Agrif_Step_Child(step)

实际上，可以在 step()结尾处调用 Agrif_Step_Child(step)

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

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

end subroutine step
```

（3）Agrif_Step_Childs(step)

在当前父级别网格的各子级别网格上，递归调用 step()，不做时间细化。

（4）Agrif_Integrate_ChildGrids(step)

在当前父级别网格的各子级别网格上，递归调用 step()，做时间细化。与**Agrif_Step(step)**的区别是：仅在子级别网格上调用 step()，而不在父级别网格上调用。

2.3 附属的 Agrif 函数

Agrif_Root()　指示当前网格是否是根级别网格；

Agrif_Fixed()　返回当前网格的级别数（0 表示是根级别网格）；

Agrif_IRhox()　返回当前网格的空间细化因子；

Agrif_IRhot()　返回当前网格的时间细化因子；

Agrif_Nb_Step()　返回当前网格的时间步数；

Agrif_Nbstepint()　父级别网格积分步内的子时间步数；

Agrif_Parent(X)　获取父级别网格上标量变量 X 的值。

举例：

使用 Agrif 函数设置更新变量的时间：

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

第 3 章　AGRIF 软件视角的计算网格

3.1 计算网格

3.1.1 参考网格

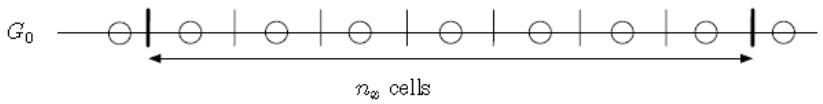


图 3.1　粗网格，包含内部 *nx* 个单元和 2 个 ghost cells

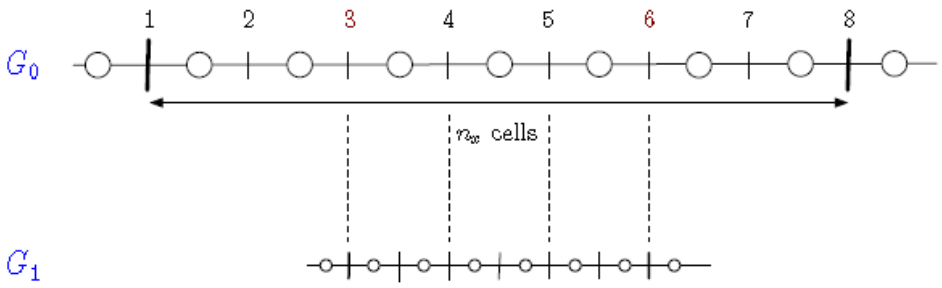


图 3.2　粗网格和细网格 G1，细化因子为 2

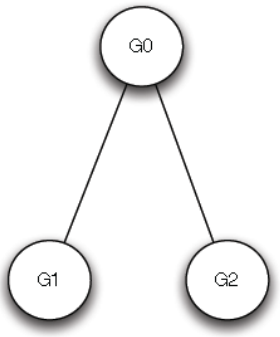
3.1.2 AGRIF 的 AGRIF_Fixed_Grids.in

该文件用来定义网格位置、时间和空间细化因子。也提供各网格的子级别网格的数目并定义他们。

文件内容举例：

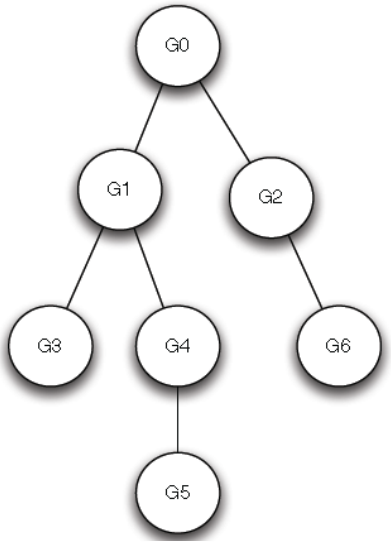
- AGRIF_Fixed_Grids.in file for the definition of one grid with no child grid in 1D:
  1
  3 6 2 2 # imin imax rhox rhot
  0
- AGRIF_Fixed_Grids.in file for the definition of one grid with no child grid in 2D:
  1
  3 6 4 8 3 2 3 # imin imax jmin jmax rhox rhoy rhot
  0

分级网格情况：



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

图 3.3 粗网格 G0，有 2 个细化网格 G1 和 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

图 3.4 粗网格 G0，有 2 个细网格 G0 和 G1，还有更细的网格

3.2 网格变量的声明

3.2.1 变量声明

为了通过使用过程名称(proc_name)，使用某变量，用于插值或更新。需要指定变量及其位置：

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

- variable_id 是一个整数（输出）；
- stag 为 1（非单元中心处）或 2（单元中心处）；
- first_index 是在参考网格中第 1 个节点的数组编号；
- dim 为 x,y,z 或 N，N 表示该方向上不做细化。

例如：

单元中心处的变量：

图 3.5 单元中心处的变量 T

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

非单元中心处的变量：

Figure 3.6: Non-centered variable U

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

注意：也在根级别网格上调用 Agrif_Declare_Variable。

3.2.2 网格上的相对位置

1、在细网格区域内第 1 个粗网格节点的编号

单元中心处变量(T)：

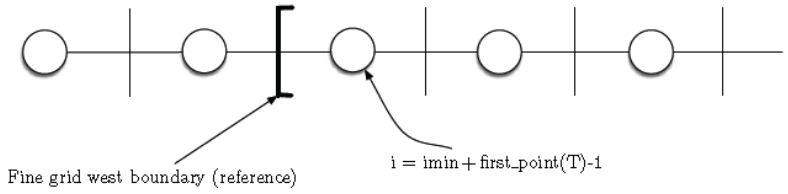


Figure 3.7: index of the first centered variable

非单元中心处变量(U)：

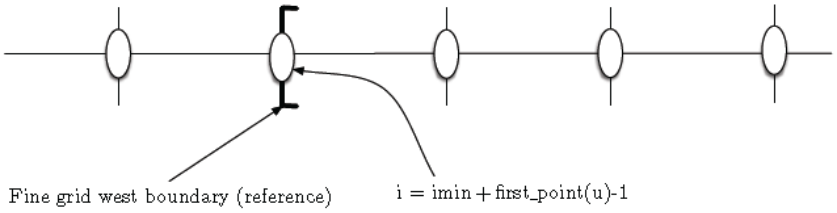


Figure 3.8: index of the first non-centered variable

imin 在文件 AGRIF_Fixed_Grids.in 中定义了。

2、更新计算 T 和 U 变量的算法

```
! 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、奇数的细化因子（3）

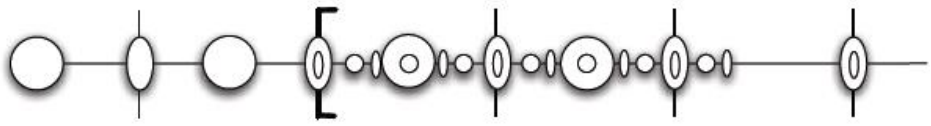


图 3.9 奇数细化因子情况的节点位置

单元中心处的粗网格节点变量（T），使用相同网格细化节点的复制或平均来更新。非单元中心节点变量（U），使用匹配的细网格节点的复制来更新。

4、偶数的细化因子（2）

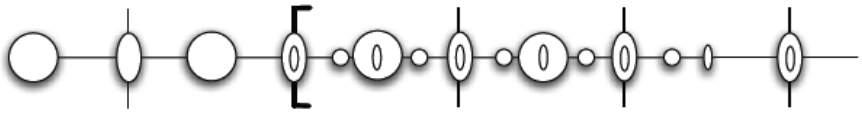


图 3.10 偶数细化因子情况的节点位置

单元中心处粗网格节点变量（T），使用平均方法来更新，而不能使用复制的方法，因为没有匹配的细网格节点。非单元中心节点变量（U），使用对应的细网格节点值的复制来更新。

第 4 章 插值和更新操作

4.1 插值

4.1.1 涉及的主要过程

- Agrif_Set_BcInterp：在边界处插值的类型；
- Agrif_Set_Bc：插值到哪个位置？
- Agrif_Bc_Variable：做一个边界插值；
- Agrif_Set_Interp：在计算域内的插值类型；
- Agrif_Interp_variable：做一个内部插值；
- Agrif_Init_variable：在整个计算域上插值（在计算域和边界上）

4.1.2 插值：如何定义插值到哪个位置？

1、Agrif_Set_Bc

Agrif_Set_Bc(variable_id, point)

其中，point=(/begin,end/)

例子：

单元中心处变量：

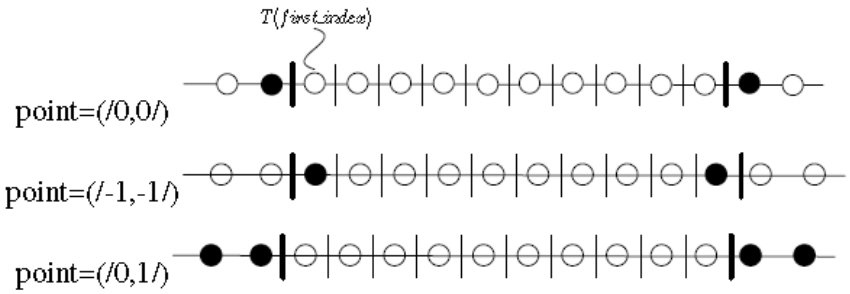


图 4.1 1D 单元中心变量（T）插值情况下的位置定义

非单元中心处变量：

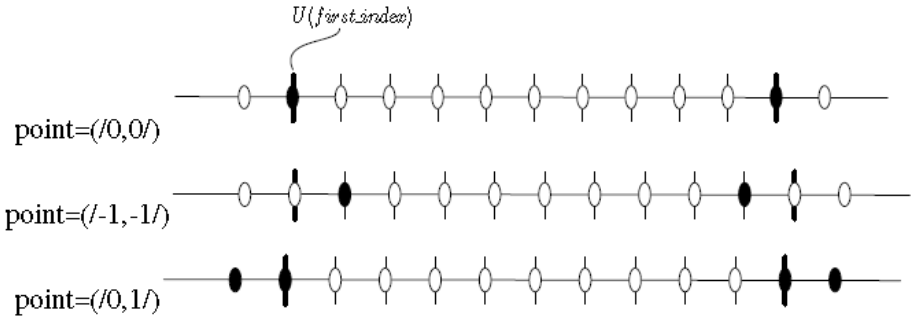


图 4.2 1D 非单元中心变量（U）插值情况下的位置定义

2、Agrif_Set_BcInterp

Agrif_Set_BcInterp(Variable_id, interp=Agrif_Interp_Type)

默认的 Agrif_Interp_Type 是 Agrif_constant

插值格式有：

	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	

例子：

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)

注意：

Interp1 表示在第 1 个维度上插值；

Interp2 表示在第 2 个维度上插值。

4.1.3 调用 Agrif_Bc_Variable

1. Agrif_Bc_Variable

Agrif_Bc_Variable(variable_id, procname)

例子：procname Interp_MyTraceur

调用 Agrif_Bc_Variable(u_id, procname=Interp_MyTraceur)

procname 的调用算法：

（1）AGRIF 在粗网格上调用 procname，使用 before=.TRUE.

输出数组：tabres

（2）AGRIF 在细网格上插值 tabres

（3）AGRIF 在细网格上调用 procname，使用 before=.FALSE.

输入数组：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
```

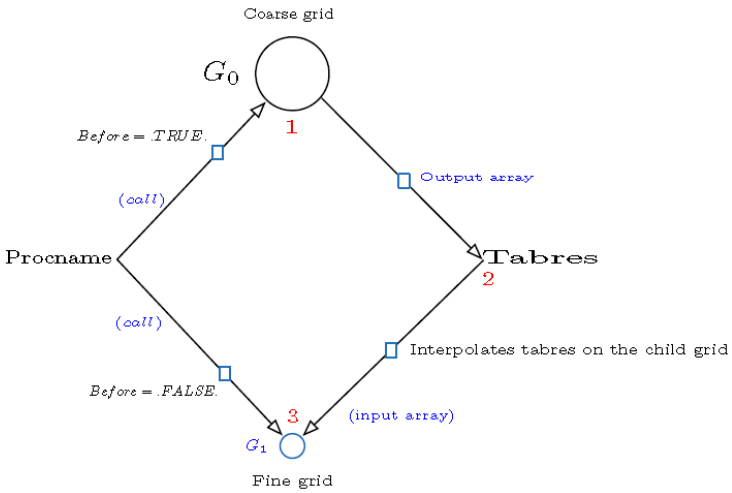


图 4.3 AGRIF 调用 procname 做插值的算法

4.1.4 时间插值

1、时间插值是如何处理的？

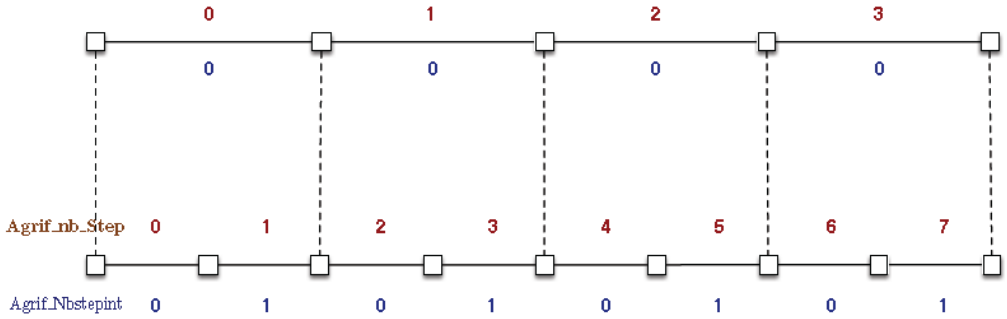


图 4.4 使用时间细化因子 2 时的网格步数 Agrif_Nb_Step 和 Agrif_Nbstepint

注意：每调用 Agrif_Step 时，Agrif_Nb_Step 发生变化。

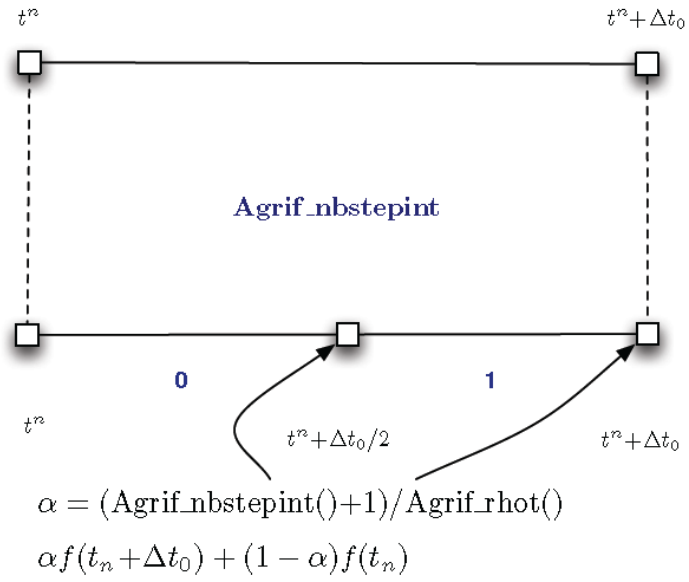


图 4.5 时间插值示意图

2、Agrif_Bc_Variable

Agrif_Bc_Variable(variable_id, procname, calledweight=α)

在 AGRIF 内，仅当父级别网格的时间编号变化时，才执行空间插值。因此，在一个父级别网格时间步内，第一次调用 Agrif_Bc_Variable 时，才执行空间插值。

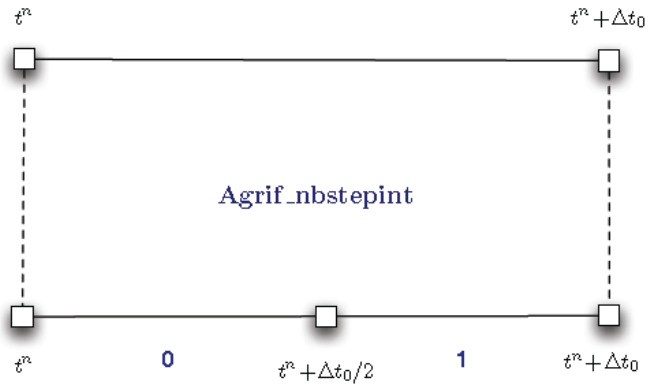


图 4.6 Agrif_Nbstepint 的介绍

注意：

（1）为了插值第一个父级别网格（场），在 Agrif_InitValues 过程中调用 Agrif_Bc_Variable 是强制性的；

（2）为实施一个插值（甚至父级别网格编号没有改变）

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

4.1.5 整个计算域上的插值

使用子程序 Agrif_Init_Variable 完成整个计算域上的插值。实际上，Agrif_Init_Variable 是通过调用 Agrif_Interp_Variable 完成计算域内的插值，以及调用 Agrif_Bc_Variable 完成边界处的插值。使用子程序 Agrif_Interp_Variable 的插值，是从计算域内的第 1 个点到计算域内的最后一个点。

值得注意的是：使用 Agrif_Set_bcinterp 做的边界处插值，指出了插值类型；计算域内部插值的情况时，是使用 Agrif_Set_interp。这 2 个子程序都使用相同的形参。

Agrif_Init_Variable = 调用 Agrif_Interp_Variable + 调用 Agrif_Bc_Variable

例子：

单元中心处变量：

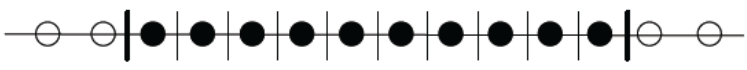


Figure 4.7: Interpolation of centered point (T)

非单元中心处变量：

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

4.1.6 掩盖(masked)域的处理

Agrif_SpecialValue，用临近数值代替 masked 数值。

必须在调用 Agrif_Bc_Variable, Agrif_Interp_Variable 之前，设置如下参数：

Agrif_UseSpecialValue= .true.
Agrif_SpecialValue = Val

例子：

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

注意：

设置 masked 数值的最大 lookup，因为这可能影响计算效率。

```
Agrif_Set_MaskMaxSearch( mymaxsearch )
```

默认值：MaxSearch = 5

4.2 更新

4.2.1 涉及的主要过程

Agrif_Set_UpdateType: 更新的类型；

Agrif_Update_Variable：做出限制，定义更新哪个位置

4.2.2 如何定义更新的类型？

必修使用子程序 Agrif_Set_UpdateType，指出更新的类型。

1、Agrif_Set_UpdateType

```
Agrif_Set_UpdateType(variable_id, update = Agrif_Update_Type)
```

Agrif_Update_Type 应该是 Agrif_Update_Copy, Agrif_Update_Average 或 Agrif_Update_Full_Weighting

2、更新方法

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

例子：

1D

```
Call Agrif_Set_UpdateType(variable_id, update = Agrif_average)
```

2D

Call Agrif_Set_UpdateType(variable_id,update = Agrif_average)！在所有方向上都使用相同的更新方法

Call Agrif_Set_UpdateType(variable_id,update1 = Agrif_Copy, update2 = Agrif_Average)！在 x 方向(1)和 y 方向(2)上，分别使用不同的更新方法

例子：空间细化因子为 3

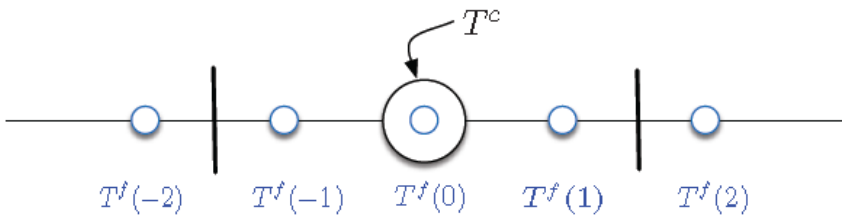


图 4.9 使用空间细化因子 3 时，粗网格和细网格节点的相对位置

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 调用 Agrif_Update_Variable

Agrif_Update_Variable(variable_id, locupdate, procname)

例子：

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

注意：locupdate 是一个可选的关键词，指示执行更新的位置。

4.2.4 定义需要更新的位置

1、不使用 locupdate 关键词

Agrif_Update_Variable(variable_id, procname)

不使用 locupdate 关键词时，默认的插值位置是：参考网格的内部。

单元中心处变量：

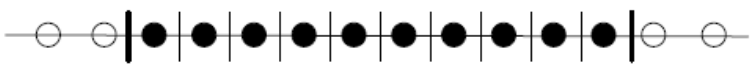


图 4.10 不使用 locupdate 关键词时更新单元中心处变量

非单元中心处变量：

图 4.11 不使用 locupdate 关键词时更新非单元中心处变量

2、使用 locupdate 关键词

更新有限区域：

Agrif_Update_Variable(variable_id, locupdate, procname)

例子：

单元中心点：

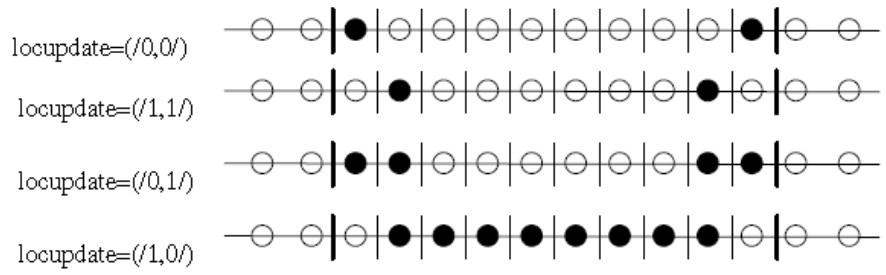


图 4.12 使用 locupdate 关键词，更新单元中心点（T）

非单元中心点：

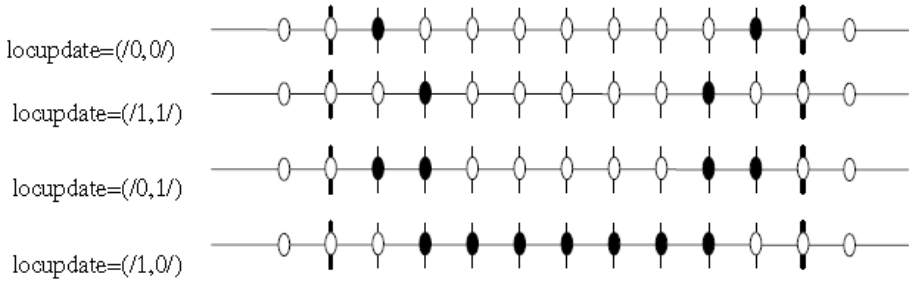


图 4.13 使用 locupdate 关键词，更新非单元中心点（U）

注意：多维情况，locupdate1=... , locupdate2 = ...

4.2.5 使用 procnames

1. Agrif_Update_Variable

Agrif_Update_Variable(variable_id, procname)

例子：

Call Agrif_Update_Variable(variable_id, procname=Update_MyTraceur)

调用 procname 的算法：

（1）AGRIF 在子级别网格上调用 procname，使用 before=.TRUE.

输出数组：tabres

（2）AGRIF 在父级别网格上更新 tabres

（3）AGRIF 在粗网格上调用 procname，使用 before=.FALSE.

输入数组：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
```

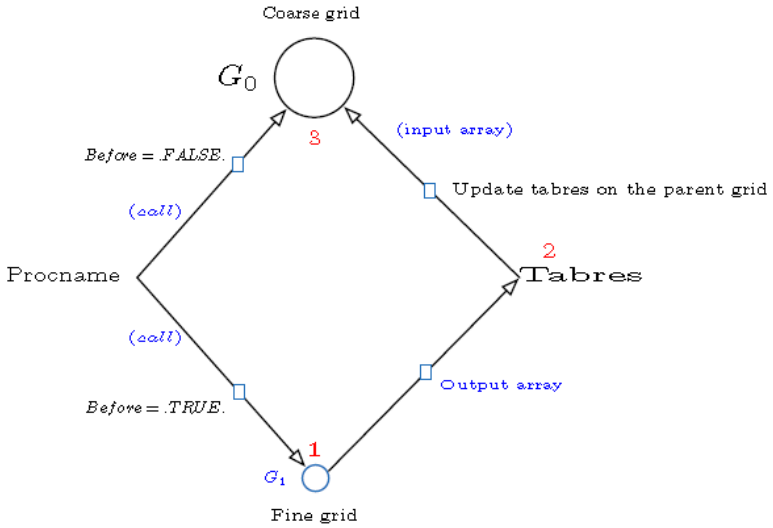


图 4.14 AGRIF 调用 procname 做更新计算的算法

4.2.6 掩膜(masked)区域处理

Agrif_SpecialValue_InfineGrid，在限制操作中不考虑掩膜值；

Agrif_UseSpecialValueInUpdate = .true.

必须在调用 Agrif_Update_Variable 之前，设置 Agrif_SpecialValueFineGrid = Val

例子：

```
Agrif_UseSpecialValueInUpdate=.true.
Agrif_SpecialValueFineGrid=0.0
Call Agrif_Update_Variable(variable_id, procname=Update_MyTraceur)
Agrif_UseSpecialValueInUpdate=.false.
```

第 5 章 共享内存和分布内存并行化

5.1 分布式内存

如何使用 Agrif 使你的代码 MPI 并行化？

例子：

在 Makefile 中添加（在 jobcomp 的 Line 244）：

```
CPPFLAGS += -DAGRIF_MPI
```

MPI 初始化：

```
use Agrif_Mpp              ! CROCO 没有使用这个 module 了
call MPI_INIT(status)    ! CROCO 的 main.F 中，使用 Agrif_MPI_Init()就不用 MPI_INIT(status)
call Agrif_MPI_Init()     ！ 必须的
```

用户必须告诉 Agrif 如何转换局部编号（在某进程上）为全局工作空间：

例子：Agrif_Invloc　　（在 zoom.F 中）

```
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
```

目前，所有网格都串行(sequentially)积分，如图 5.1：

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

对每个网格，使用所有进程。

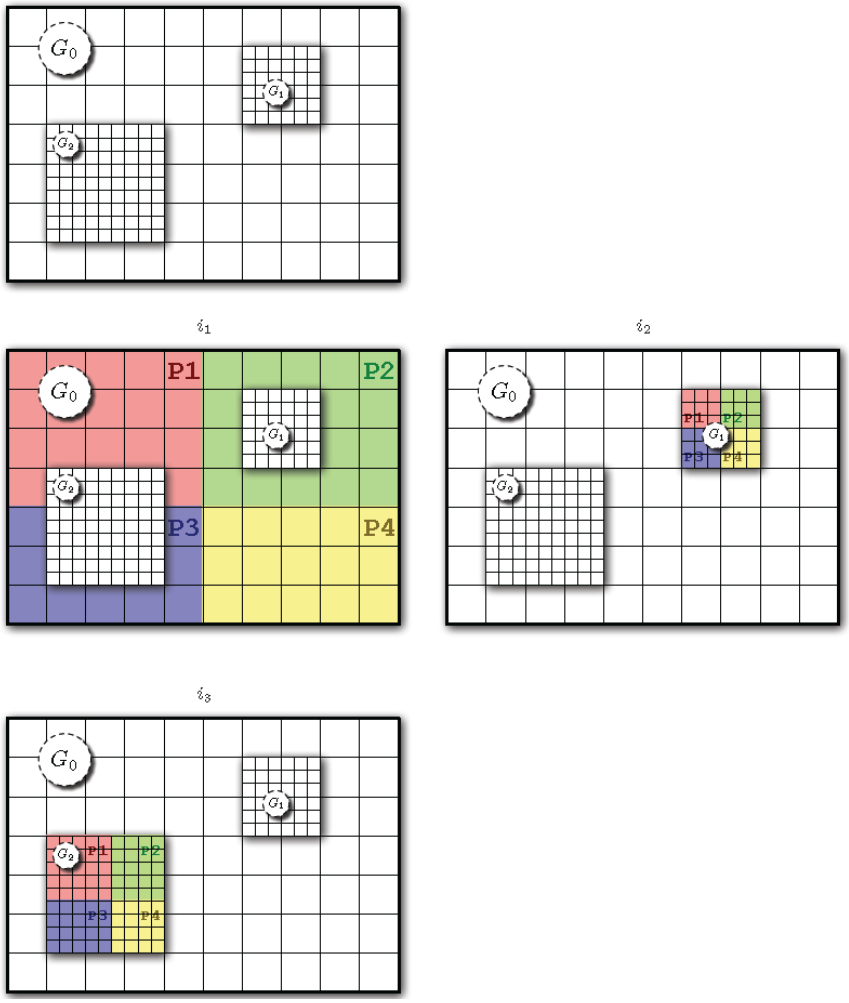


图 5.1 当网格是串行积分时，对每个网格使用所有进程

Sister 网格（相同的父网格）可以并行(Concurrently)积分，如图 5.2：

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

对每个顺序执行，进程分配在相同级别的网格上。

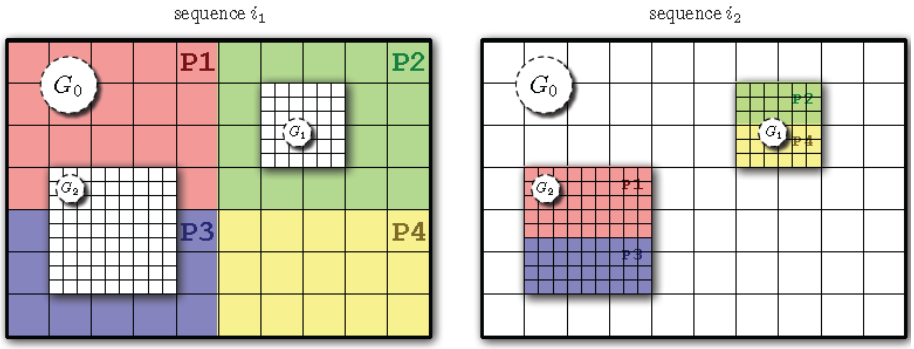


图 5.2 相同级别网格上进程的分配

5.2 共享式内存

目前，所有的插值和限制操作，都是由 1 个线程执行，计算效率低，使用 nested 并行化在研究中。

例子：

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

第 6 章 自适应网格细化

6.1 自适应细化的参数

参数与主要函数：

Agrif_Set_Rafmax(nlevel) 最大细化层数

Agrif_Set_Regridding(nbsteps) 每 nbsteps（根级别）网格步，重构网格分级

Agrif_Set_Minwidth(minwidth) 网格的最小宽度

例子：

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

6.2 自适应准则

```
Agrif_Detect
Agrif_Detect(taberr, size_taberr)
```

当侦察到错误，taberr = 1，taberr 值位于参考网格的单元角落上（包括边界）。

例子：

```
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
```

注意：不做自适应网格细化时，也要调用 Agrif_Detect

6.3 变量恢复

从一个分级网格，恢复(restore)网格变量到另一级别网格。

声明：调用 Agrif_Declare_Variable(..., variable_id, restore=.true.)

定义恢复到哪个位置：Agrif_Before_Regridding, Agrif_Save_ForRestore

例子：

```
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()
```

注意：不使用自适应网格细化时，也要调用 Agrif_Before_Regridding。

第 7 章 高级应用范例

7.1 通量校正(ROMS)

1、通量修正(I) (ROMS)

如何实施通量修正过程？

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

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

在细网格和粗网格上存储通量，使用两者之差来修正界面附近粗网格节点上(的变量值)。

2、通量修正(II) (ROMS)

（a）声明通量数组(F_x, F_y)，计算，然后在父时间步、在细网格上求和；

（b）声明对应这些通量的 profile:

Call Agrif_Declare_variable((/1,2/),(/1,1/),(/'x','y'/),(/0,1/),(/nx,ny/), F_{xid})
Call Agrif_Declare_variable((/2,1/),(/1,1/),(/'x','y'/),(/1,0/),(/nx,ny/), F_{yid})

（c）设置更新类型

Call Agrif_Set_UpdateType(F_{xid},update1=Agrif_Update_Copy, update2=Agrif_Update_Average)
Call Agrif_Set_UpdateType(F_{yid},update1=Agrif_Update_Average, update2=Agrif_Update_Copy)

3、在调用 Agrif_Update 内部做通量修正

Call Agrif_Update_Variable(F_{xid}, procname = Correct_Flux_x)
Call Agrif_Update_Variable(F_{yid}, procname = Correct_Flux_y)

4、通量修正(III) (ROMS)

例子：Correct_Flux_x　（CROCO 见 update2D.F 的 Updateubar 子程序）

```
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   ! not before
    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 干湿演变

1、干湿演变(I) (MARS)

如何通过 SpecialValue 定义是陆地还是水域？

例子: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 垂向细化

1、不同的垂向网格(I) (NEMO)

网格有不同的垂向网格。2 个网格之间的垂向映射在 procname 中完成。

例子：使用不同的垂向网格更新。

```
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、不同的垂向网格(II) (NEMO)

使用不同的垂向网格做插值。　　（代码貌似与 Update 的一样啊？）

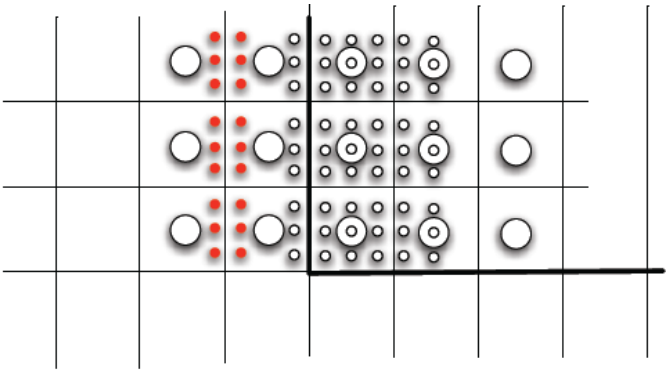
```
Subroutine Interp_My_Traceur(tabres,i1,i2,k1,k2,before)
real,dimension(i1:i2,k1:k2) :: tabres
logical before
if (before) then
   ! on the parent grid
   tabres = My_Traceur(i1:i2,k1:k2)
else
   ! on the child grid (NB: tabres has a vertical dimension corresponding to the one of parent grid)
   do 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 用细网格值做插值

1、标量插值(I) (NEMO)

在插值期间需要使用内部细网格的数值。



2、标量插值(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、标量插值(III) (NEMO)

- 插值 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
```