可扩展建模系统[1]

V. Balaji,
地球流体实验室/ 普林斯顿大学

Zhi Liang
地球流体实验室/HPTI

1、引言：建模框架的紧迫性

在气候研究中，随着对决定着气候的各物理过程的具体表示越来越重视，一个模式的建立需要很多团队共同协作，每个团队只精通气候系统的某个部分，例如海洋环流，生物圈，陆地水循环，辐射传输以及化学等等。目前模式代码的开发需要团队能够为整个耦合系统提供分量模式，而又不要求研究人员完全掌握全部。跟前几年单个模式团队独立开发的方式相比，这可以被称作分布式开发模型。

于此同时，随着我们转向可扩展计算体系，高性能计算硬件和软件的复杂度也大大提高。可扩展体系分为几种，包括共享内存并行向量机系统，分布式内存大规模并行系统。在朝多核、很多核、协处理器和加速器发展的趋势下，单独的计算单元本身的内存层次也越来越复杂。考虑到协助在多个机构间共享代码和开发开销的问题，有必要把内部架构抽象并给不同扩展以及单处理器构架提供一种统一的编程模型。

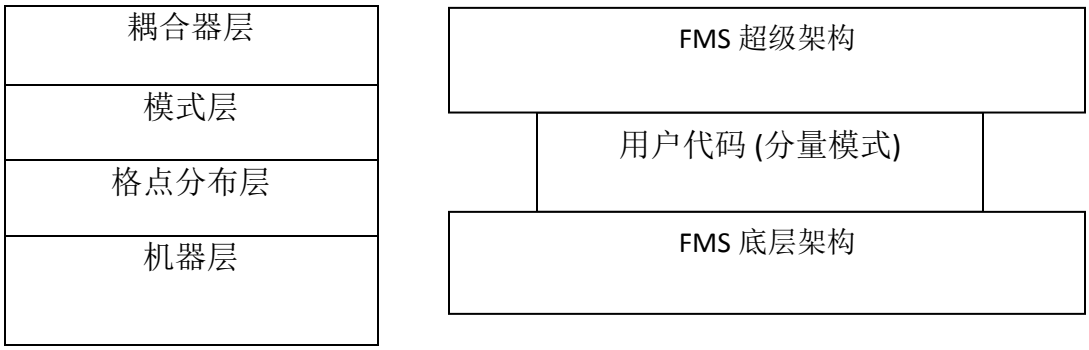

图 1.1: 扩展模型系统的架构。

[1] 原文：Balaji,V. and Liang, Z., 2010: The Flexible Modeling System。

谢歆译，何卷雄校。

GFDL 可扩展建模系统是一种模型框架的较早例子，也是一种用于建立耦合气候模式的综合编程模型和工具集。FMS 体系描述示意图见图 1.1。这种“三明治”的体系是相当普遍的。用户代码通常是表示科学算法的程序集，它们是根据标准框架的规范所写成，这些标准框架提供了很多常用的技术服务，如输入/输出，异常处理，而最重要的是分布式格点和分布式变量场上的运算。这些独立于下面硬件体系并且能够在所有平台上统一的对并行机制进行标准的高层表示是研究的热点。

本章内容安排如下。在第二部分我们描述 FMS 的上层架构。在第三部分我们简要 FMS 底层架构

2、FMS 超级构架

2.1 耦合器

FMS 耦合器被设计用于解决地球系统的不同分量，比如说大气和海洋，是如何离散的。早期几代气候模式对于所有的分量使用的是同样的离散方法或者说简单的整数精细化：因此，数据在分量之间的交换是一种相对简单的点对点交换。而任何对一个分量模式分辨率的限制不但影响其本身还会影响到其他分量。现在越来越多的分量模式使用特定适合于自己的独立的离散方法。在这种情况下，如何才能够让例如从海洋模式得到的海表面温度在大气模式中作为边界条件使用呢？

这就是对网格重新划分的问题，就地球系统模式而言，它受到以下条件的制约：

- 物理量必须能够在全球守恒：如果在界面上有某个量的通量，那么它在从一个分量被传递到另一个分量时必须守恒。这在模拟天气或者短期（季节内或者年际）气候变率时没有那么严格，但是当积分时间数量级达到 O(1000,000)-O(100,000,000)时间步长时，这在实际气候变率中就变得非常重要。
- 通量交换的数值方法必须稳定，这样每个独立的分量时间步长就不会受到边界通量计算的限制。
- 不存在对于分量模式的离散方法的限制。特别的，坐标线的分辨率和对齐不能受到外部的限制。这就意味着必须有一种高阶的内插值方案，因为低阶方案在那些倾斜分辨率比例大的格点之间效果不好。高阶方案可能会要求通过对网格重划分不仅能得到通量，而且还能得到它们的空间高阶导数。
- 要求对于时间轴的离散也是独立的：分量模式可以拥有独立的时间步长。（我们目前的确有个限制，要求耦合时间步必须是任何分量模式时间步长的整数倍数，参加交换的分量模式的时间步也不能是互为质数的）。
- 交换必须与发生在分量表面的物理过程保持一致。这个要求是必须强调的，因为发生在行星表面附近的物理过程可能很特殊，比如大气和海洋的边界层，海冰和陆面以及生物圈和水圈的边界。
- 最终，我们要求在并行化硬件上实现高效的计算：这是一种在各模式分量可扩展性的极限下对无速率限制的解决方法。分量可以在耦合事件之间被安排为串行或者并发运行。

FMS 耦合器认识到只有某些分量模式的网格是独立的，如大气，海洋表面，陆地表面以及海洋。海洋表面和海冰的表示也是一样。其他所有分量的网格都是从这些分量中得到，例如，海洋物理

和化学分量是从大气中得到；陆地生物圈，河流和陆冰分量从陆地表面中得到；海洋地球生物化学分量是从海洋中得到。

对于上述列举的每个分量都有接口或者插槽。比如，一个海洋模式会将它的状态用一些特殊的数据结构标记，然后保存那些要跟其他分量交换的场，并把它们称作海洋边界类型和海洋数据类型。为了初始化和终止，它必须提供名为海洋模式初始化以及海洋模式结束的调用，以及一个用于更新海洋模式的程序来让模式前进一个耦合步。这些调用都有特殊的语法。每个插槽都可能含有一个空分量（如果该分量没有被用到的话），以及一个“数据”分量（比如说海洋分量可以被替换为数据）。另外，我们为调整模式提供了一个数据覆盖功能，这样模式中的某些独立场可以被数据替换掉。

代码块 1.1 显示了这样一个数据结构的例子，它是用来在冰和海洋分量之间交换数据的。类型主要包括了很多 2D 表面场，变量 xtype 用于标识交换类型。

```
type, public :: ice ocean boundary type
real :: u flux(:,:)                    ! wind stress (Pa)
real :: v flux(:,:)                    ! wind stress (Pa)
real :: t flux(:,:)                    ! sensible heat flux (W/m2)
real :: q flux(:,:)                    ! specific humidity flux (kg/m2/s)
real :: salt flux(:,:)                 ! salt flux (kg/m2/s)
real :: lw flux(:,:)                   ! long wave radiation (W/m2)
real :: sw flux vis dir(:,:)           ! direct visible sw radiation (W/m2)
real :: sw flux vis dif(:,:)           ! diffuse visible sw radiation (W/m2)
real :: sw flux nir dir(:,:)           ! direct near IR sw radiation (W/m2)
real :: sw flux nir dif(:,:)           ! diffuse near IR sw radiation (W/m2)
real :: lprec(:,:)                     ! mass flux of liquid precip (kg/m2/s)
real :: fprec(:,:)                     ! mass flux of frozen precip (kg/m2/s)
real :: runoff(:,:)                    ! mass flux of liquid runoff (kg/m2/s)
real :: calving(:,:)                   ! mass flux of frozen runoff (kg/m2/s)
real :: runoff hflx(:,:)               ! heat flux of liquid land water (W/m2)
real      :: calving hflx(:,:)         ! heat flux of frozen land water (W/m2)
real      :: p(:,:)                    ! pressure of sea ice and atmosphere (Pa)
integer :: xtype                       ! REGRID, REDIST or DIRECT
type(coupler 2d bc type) :: fluxes ! additional tracers
end type ice ocean boundary type
```

Code Block 1.1: ice ocean boundary type

耦合器是用于在独立网格上耦合大气，海洋，陆面以及海冰的驱动器。它封装了边界条件和边界通量。分量模式耦合起来，这就使得在海洋，陆地和冰模式的界面上有热量和湿度的隐式垂直扩

散。所以，大气，陆面和冰模式使用的是同一个时间步长。大气模式被分为向下和向上的调用，这跟标准的三角对角消去法的下扫和上扫对应起来。海洋界面使用隐式混合。达到海洋或者是来源于海洋的通量必须通过冰模式传递。这包括了大气通量以及从陆地到海洋的通量（径流）。这个程序包括了模式的主时间循环。主时间循环中的每一遍历都是一个耦合（慢的）时间步。使用大气模式，在这个慢时间步中含有一个快时间步，海冰和海洋在每个慢时间步交换一次。代码框 1.2 是慢时间步。快循环在慢时间步中执行并列在代码框 1.3 中。

```
do nc = 1, num_cpld_calls
    call generate_sfc_xgrid( Land, Ice )
    call flux_ocean_to_ice( Ocean, Ice, Ocean_ice_flux )
    call update_ice_model_slow_up( Ocean_ice_flux, Ice )
    !fast loop
    call update_land_model_slow(Land)
    call flux_land_to_ice( Land, Ice, Land_ice_flux )
    call update_ice_model_slow_dn( Atmos_ice_flux, Land_ice_flux, Ice )
    call flux_ice_to_ocean( Ice, Ice_ocean_flux )
    call update_ocean_model( Ice_ocean_flux, Ocean )
  enddo
```

Code block 1.2: 慢时间步的源代码

```
do na = 1, num_atmos_calls
      Time = Time + Time_step_atmos
      call sfc_boundary_layer( Atm, Land, Ice, &
      Land_ice_atmos_flux )
      call update_atmos_model_down( Land_ice_atmos_flux, Atm )
      call flux_down_from_atmos( Time, Atm, Land, Ice, &
      Land_ice_atmos_flux, Atmos_land_flux, Atmos_ice_flux )
      call update_land_model_fast( Atmos_land_flux, Land )
      call update_ice_model_fast( Atmos_ice_flux, Ice )
      call flux_up_to_atmos( Time, Land, Ice, Land_ice_atmos_flux )
      call update_atmos_model_up( Land_ice_atmos_flux, Atm )
  enddo
```

代码框 1.3: 快时间步的源代码

耦合器还同时支持串行耦合和并发耦合。串行耦合如图 1.2 所示使用的是耦合的向前-向后时间步。

$$A^{n+1} = A^n + f(O^n)$$
$$O^{n+1} = O^n + f(A^{n+1})$$

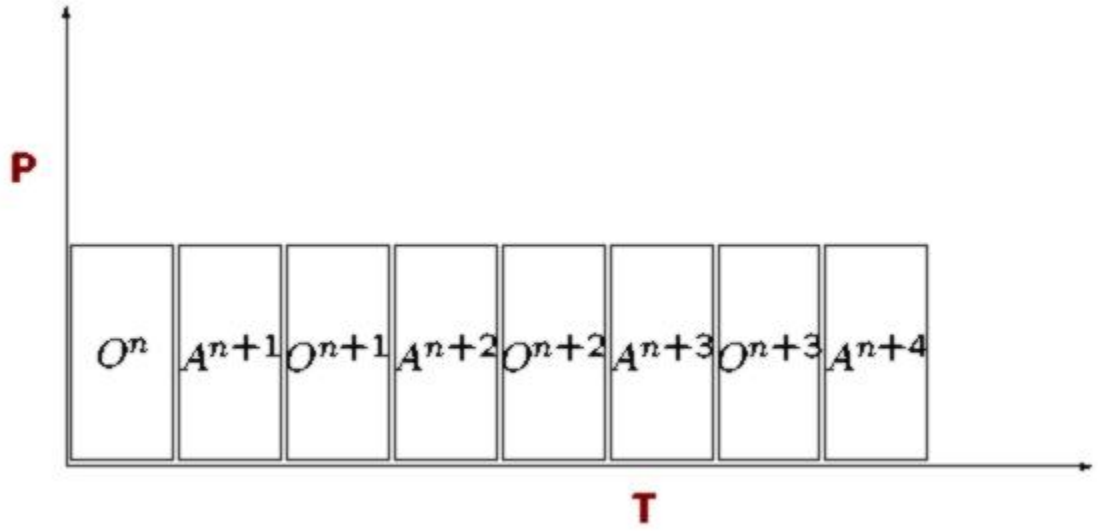

图 1.2: 串行耦合

并发耦合使用只能向前的时间步，见图 1.3。虽然这是无条件不稳定的，整个系统还是有很强的阻尼的。在串行耦合中，因为海洋被前一步的大气状态所强迫，情况也会有所改变。并行耦合的好处是它提高了可扩展性和性能。

$$A^{n+1} = A^n + f(O^n)$$
$$O^{n+1} = O^n + f(A^n)$$

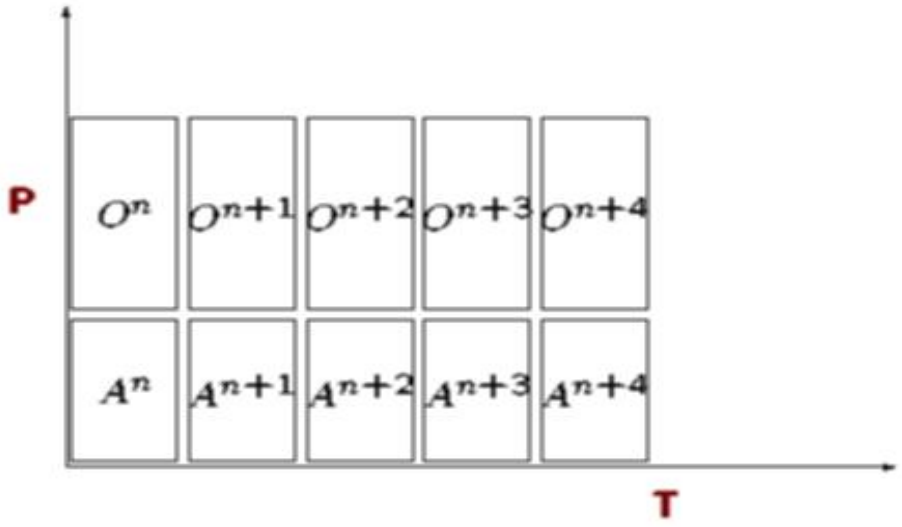

图 1.3: 并行耦合

2.2 交换网格

交换网格被定义为在离散化后的顶点连接而成的线段所定义出来的网格。给定两个网格，交换网格就是两个父网格的所有顶点合集定义出来的网格集。这可以在图 1.4 中由两个父网格（“大气”和“海洋”）看出。正如我们看到的那样，每个交换网格都是唯一的跟父网格下的一个子网格以及其所占父网格的百分比联系起来的。那些要从父网格传递到另一个网格的量会首先利用面积百分比插值到交换网格上；接着再使用其他的百分比平均到接收网格上。如果交换量的某种特殊矩是要求守恒的，那么必须用矩守恒的插值和面积百分比平均函数。这可能不仅仅要求网格平均量（零阶矩）而且要求他们的高阶矩也通过交换网格传递。

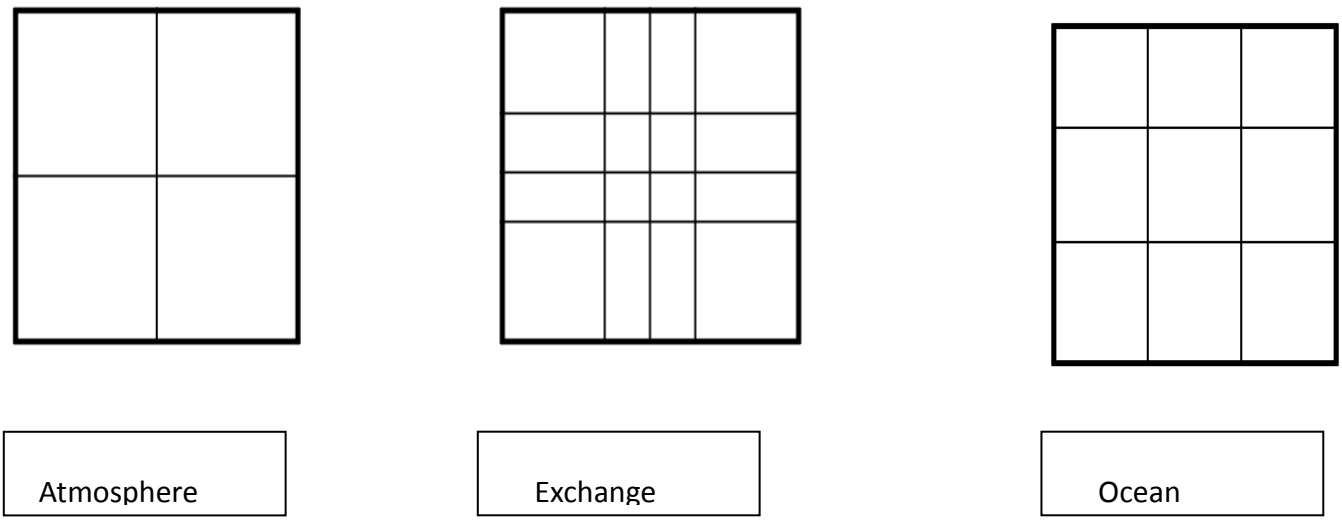

图 1.4: 大气和海洋网格之间的交换网格。

给定一个父网格的 N 个子格以及另一个父网格的 M 个子格，那么交换网格就是网格里的每个子格都和另一个网格里的子格重叠的极限情况，这样形成一个 N×M 的矩阵。实际上，一般重叠的子格很少而且交换网格是非常稀疏的。在代码中，我们通常把交换网格子格数组处理为一个紧密的一维数组，该数组的索引指向父子格。表 1.1 给出了典型气候模式分辨率下交换网格的特征。第一行是当前的 GFDL 模式 CM2.1，第二行是 CM2.4，一个更高分辨率版本。正如这里所看到的，交换网格是极为稀疏的。

大气	海洋	交换格点
144x90	360x200	79644
288x180	1080x840	895390

表 1.1:典型气候模式格点的交换网格大小。

交换网格本身的计算可能就非常费时，因为父网格可能完全是在非拱形的曲线坐标系下。实际中，网格一般会预先计算并储存起来，所以这个问题可以回避。不过，如果父网格是自适应的，那么这个问题还会再出现。

FMS 中交换网格的实现是限制在行星表面的二维网格中的。然而，这并不限制交换网格的概念在变化的三维甚至四维情况下使用。

当一个表面被分割为互补的分量时问题就会变复杂：在地球系统中，一个典型的例子就是海洋和陆地表面共同覆盖了大气以下的区域。这时就要求三个分量之间交换是守恒的：关键的变量如 CO2 在这三种介质中都有分布，总的碳含量必须是保持守恒的。

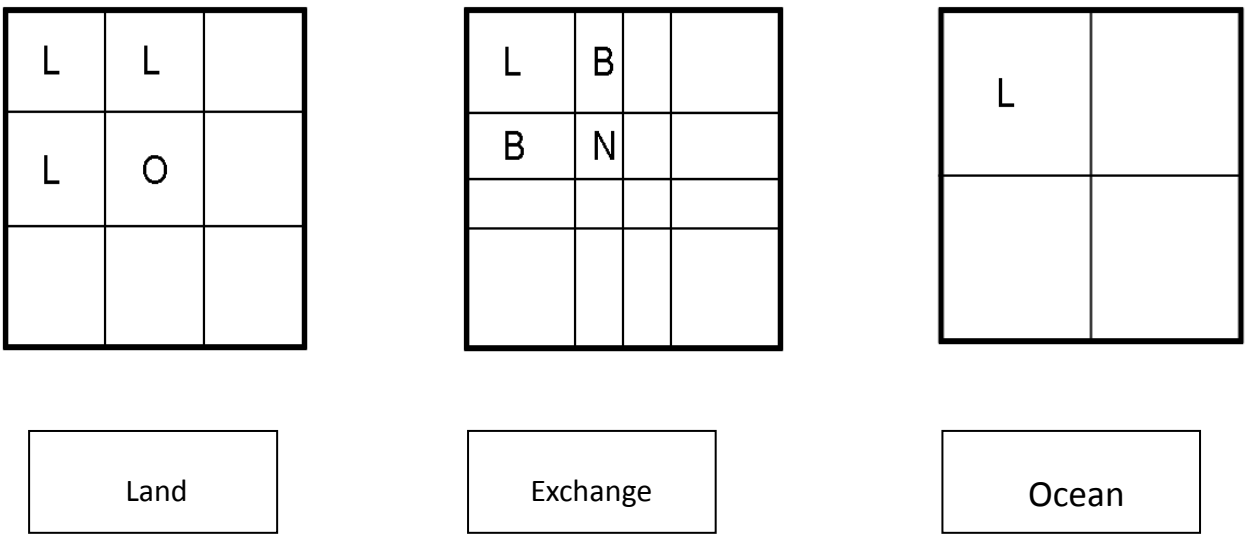

图 1.5: 蒙版问题。在左侧，陆地和大气拥有同样的格点，但右侧中，其海陆蒙版跟海洋模式不一样。中间的交换网格可以解决这些问题的：孤立的子格 N 可以被任意的指定为陆地，而那些被切出来的陆地子格区域则是被共同归于海洋和陆地的，标记为 B。

图 1.5 就展示了这样的一个大气陆地网格以及海洋网格的分辨率不同的例子。在开始两个框的黑线代表的是在两个网格上离散的蒙版，其中标记为 L 的子格属于陆地。由于分辨率的不同，某些交换网格存在模糊的状态：两个标记为 B 的子格同时被声明为陆地和海洋，而那个孤立的子格 N 则既不是海洋也不是陆地，因为它在海洋分量中被认为是陆地，而在陆地分量中被认为是海洋。

这就意味着蒙版定义的互补网格的边界只能使用交换网格来准确的定义。只有这样才能保证子格区域能够正好覆盖全球范围。而那些状态模糊的网格则使用某些归属规则来解决。比如说，在 FMS 交换网格中，我们一般会根据需要修改陆地模式：陆地网格子格一般比较独立，而且比较容易做这种变换。我们还会对陆地网格增加子格一直到在交换网格上不再出现孤立的子格为止，然后可以通过在陆地上切除一些分数面积来把那些双重归属的子格去掉。

3、FMS 构架

FMS 是一个用于构造和运行大气，海洋以及气候系统模式的软件架构。该架构包括了一系列处理并行，输入和输出，不同模式间格点数据交换的软件。这个架构还应该能够很大程度上把 FMS 用户从具体的机器细节中分离出来。它应该还包含了用于处理通信（3.1 节），区域划分（3.2 节），并行输入/输出（3.3 节），诊断针对以及数据覆盖的软件（3.5 节）。

3.1mpp_mod

mpp_mod 是一系列简单的调用，它们给不同的消息传递库提供了统一的接口。目前它是使用 MPI 标准实现的。

在处理器以及其内存之间的数据传输是要是基于对内存的加载和储存操作。共享内存系统（包括分布式共享内存系统）有一个单独的地址空间，任何处理器都可以在内存内通过读取和储存的操作获取任何数据。而分布式并行系统的情况就不一样了。特殊的 MPP 系统比如说 T3E 能够从远程内存中通过直接获取数据的方法来模拟共享内存。可是如果并行代码是分布在集群机器上，或者说是跨网络的，那么消息必须使用用于长距离通信的协议如 TCP/IP 来实现发送和接收。这需要分布式系统的节点间的“握手”。人们可以考虑使用协商通讯的方法（例如 MPI）进行发送和接受。

MPI 是为松散式耦合系统的分布式计算开发的一种标准，而这也造成了通信操作中软件方面带来的性能损耗。它是一个开放的工业标准。OpenMP 是在 FMS 模式中另一层级的并行实现方法，它能有效利用多核超级计算机系统。

气候和天气代码中的消息传递要求可以简化为一个相当简单的最小化集合，而这也是在任何消息传递 API 中容易实现的。mpp_mod 就提供了这个 API。mpp_mod 的特征包括：

1）简单，最小化的 API，在处理更复杂的东西时，可以自由访问底层的 API。
2）专门为气候/天气计算流体力学设计的使用设计。
3）性能不会比内建 API 降低很多。

并行计算一开始是令人生畏的，但是很快就变得让人们习惯，现在我们很多人都可以不太费劲就写出向量代码。在读、写并行代码时关键的一点是使用同样的代码抓住几个独立的并行执行流。每个你检查的量在不同的执行流中都可能有不同的取值，比如说处理器 ID 就是这样一个明显的例子。子程序和函数调用也是很有技巧的，因为仅仅从一个调用不能明显的看出它在不同执行流中是哪种同步。

因此，重要的一点是我们需要明白子程序或者函数调用的上下文环境以及它所对应的同步。这里有某些调用（比如 mpp_declare_pelist，mpp_init，mpp_malloc，mpp_set_stack_size）必须被所有 PEs 所调用。而其他的则必须被 PEs 的子集（这里称为一个 pelist）所调用，而且必须被 pelist 中的那些 PEs 所调用（比如 mpp_max，mpp_sum，mpp_sync）。仍然有一些根本就不同步的。我将突出 MPP 模块中每个调用的上下文环境，以便它所隐含的同步能够被清楚的表示。

为了照顾到性能，我们有必要尽可能的让同步限制在实现算法所允许的范围之内。比如说，两个处理器之间的一条信息仅仅意味着在所关注的 PEs 上有同步。全局的同步（或者说阻塞）很可能是很慢的，所以最好避免。

另一个使用 pelist 的原因是为了运行单独的程序，让不同的 PE 自己工作在不同的代码段上。一个典型的例子是为海洋模式和大气模式制定不同的 PE 子集，然后让他们并发而不是串行运行。MPP 模块为一个当前的 pelist 提供了一种表示方法，当一组 PEs 划分为一个子集时 pelist 就会被设置。接下来的那些调用中，如果忽略了 pelist 可选参数（见下面很多个独立的调用），那么它认为其 implied 的同步是跨过当前的 pelist 的。mpp_root_pe 和 mpp_npes 调用还会返回对于对应于当前 pelist 的值。mpp_set_current_pelist 用于设置当前的 pelist。

mpp_mod 定义的最常用的借口是用来发送信息（mpp_send）以及接受信息（mpp_recv）的。mpp_send 会发出一个非阻塞的发送。mpp_recv 同时支持阻塞和非阻塞通信。对于非阻塞通信，需要 mpp_sync_self 调用来确保安全通信。代码框 1.4 给出了使用 mpp_send 和 mpp_recv 的简单例子。

```
Call mpp_recv(recv_buffer, glen=msgsize, from_pe=from_pe, block=.false.)
Call mpp_send(send_buffer,plen=msgsize, to_pe=to_pe)
Call mpp_sync_self(check=EVENT_RECV)   ! ensure the receiving is completed
! Unpack data here
Call mpp_sync_self(check=EVEN_SEND)    ! ensure the sending is completed
```

代码框 1.4: 使用 mpp_send 和 mpp_recv 的简单例子

mpp_mod 提供了用来做性能分析的接口。在接口 mpp_clock_begin 和 mpp_clock_end 附近，它会把代码的运行时间打印出来。每个时钟都会通过接口 mpp_clock_id 来初始化。代码框 1.5 提供了跟踪子程序 update_ocean_model 的运行时间的例子

```
id_ocean = mpp_clock_id('Ocean')
Call mpp_clock_begin(id_ocean)
Call update_ocean_model
Call mpp_clock_end(id_ocean)
```

代码框 1.5: 跟踪运行时间的例子。

3.2 mpp_domains_mod

mpp_domains_mod 是一系列用于区域分解以及在直线方块网格上做区域更新的简单调用。有限差分的可扩展实现通常基于模式的区域分区，而分出来的子区域分布在处理器上。这样，这些区域将负责在它的边界上交换数据。就像在谱变换中一样，如果数据有向外的依赖，那么需要从全局区域中获取信息，否则如果数据依赖是相邻的则不需要。区域分区是并行代码开发的关键操作。

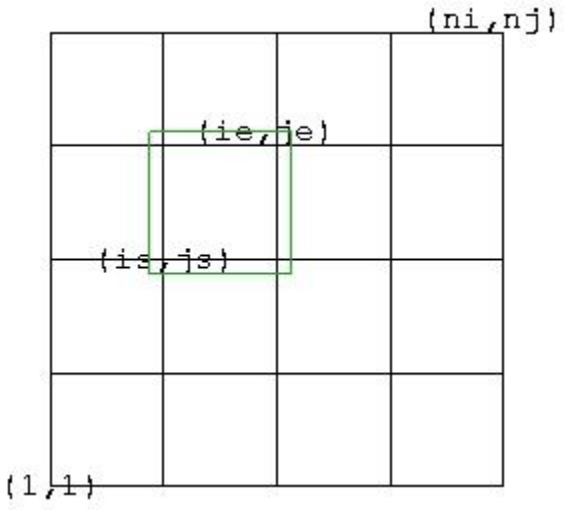

图 1.6: 在 16 个处理器上使用布局（4,4）的定义域分区。在每个处理器上，全局定义域是（1：ni，1：ni），计算定义域是（is:ie, js:je）。

一般假设区域分解主要是在两个水平空间维中进行，这两个空间维度的指标通常是变化最快的。也存在一种分别对最快变化指标上的一维分解实现，以及对第二个最快变化指标上的一维分解的方法，它被当做二维分解的一种特殊情况。我们把区域定义为与一个任务相联系的一套格点。而我们把计算区域定义为被一个任务计算的一系列网格点集，而把数据区域定义为计算任务所需要的一系列点集。尽管区域的数量却跟处理器数量经常是一样的，每个 PE 可能会有超过一个任务，，我们把全局定义域定义为整个模式的全局计算定义域（就像在单个处理器上运行的计算定义域一样）。二维区域是使用继承类型 domain2D 来定义的。图 1.6 提供了简单定义域分区的例子。

mpp_domains_mod 提供了在直线网格上进行区域分解和区域更新的 API，它是建立在 mpp_mod 的消息传递 API 上的。mpp_domains_mod 的特性包括：

- 简单，最小化 API，对于更复杂的操作可以自由访问底层的 API。
- 专门为常用气候/天气计算流体力学代码设计。

在 mpp_domains_mod 中最常使用的接口是 mpp_define_domains 和 mpp_update_domains。mpp_define_domains 被用于设置区域分解。mpp_update_domains 则用来对每个处理器上的区域分解数组的环状区进行更新。代码框 1.3 定义了一个网格尺寸为（nx,ny）的双周期性边界区域。其布局可以通过 mpp_define_layout 来定义或者由用户来选择（就像图 1.6 中 layout=(4,4)）。mpp_get_compute_domains 用于获取计算区域尺寸而 mpp_get_data_domains 则用于重试数据区域尺寸。计算是在计算区域上完成的，而数据则位于数据区域上。代码框 1.3 提供了 mpp_update_domains 的标量和向量版本。

```
call mpp_define_layout( (/1,nx,1,ny/), npes, layout )
call mpp_define_domains( (/1,nx,1,ny), npes, domain, xhalo=1, yhalo=1,  &
                xflags=CYCLIC_GLOBAL_DOMAIN, yflags=CYCLIC_GLOBAL_DOMAIN )
call mpp_get_compute_domains(domain, isc, iec, jsc, jec)
call mpp_get_data_domains(domain, isd, ied, jsd, jed)
allocate( x(isd:ied,jsd:jed) )
call mpp_update_domains(x, domain)
call mpp_update_domains(x, y,  domain, gridtype=BGRID_NE)
```

代码框 1.6: 使用 mpp_domains 的 API 的例子。

3.3 mpp_io_mod

mpp_io_mod 是一组分布式系统上对并行 I/O 的简单调用。它是为把数据写为 netCDF 格式而设计的。

在大规模并行环境中，一个常见的困难问题是磁盘上文件数据的读写。mpp_io_mod 尝试使用简单的 API 来完成各种需要的 I/O 操作。它不像 MPI 那样尽可能包括了所有的标准，当然如果需要的话它也可以在 MPI 中实现。因此在厂家给定的 APIs 基础上给 mpp_io_mod 增加并行 I/O 也是一样简单的，这样就给用户代码增加了一个隔离层。

mpp_io_mod 并行 I/O API 是建立在 mpp_domains_mod 和 mpp_mod API 上用于区域分解以及消息传递的。mpp_io_mod 的特性包括：

- 简单，最小化 API，对于更复杂的操作可以自由访问底层的 API。
- 自描述文件：在文件中含有完整的头文件信息（元数据）。
- 更注重并行写操作的性能：气候模式通常设计为读最少量的数据（一般在运行开始的时候），但是会在运行过程中写大量的数据。我们同样提供了读的接口，不过其性能还没有得到优化。
- 集成 netCDF 的功能：netCDF 是一种在气候/天气模式社区中广泛使用的数据格式。netCDF 是 mpp_io_mod 数据储存的主要方式。我们还支持纯无格式 FORTRAN I/O，而这是为了防止由于没有 netCDF，以及因不合适或者性能差的而不能采用 netCDF 的情况而加入的。

写出的数据的内部表示默认是实数类型，可以是 4 或者 8 字节。为了避免使用单位秒是在气候尺度上造成溢出，时间数据通常以 8 字节写出。

Mpp_io_mod 专为对模式性能至关重要的 I/O 操作设计，I/O 操作一般是在运行过程中每隔一个期间在整个模式的格点体上对大量数据进行写操作。设想一个三维格点体，其模式数组表示为(i,j,k)。区域分解一般是沿着 i 或者 j 的：因此在往磁盘上写整个全局数据时，数据的各个分区块就一定会是非连续的。因此如果我们尝试让所有的处理器向同一个文件写数据时，因为涉及到数据重排操作，性能就会收到严重的影响。可能的解决方式是可以让一个处理器获得所有数据然后将数据写出，或者让所有处理器独立的写文件，最后再离线合并。但在运行大量处理器时，性能也可能受到严重影响。

按照 mpp_io_mod 里的说法，根据线程和文件集这两个参数有三种操作模式：单线程 I/O：一个单处理器获得所有数据并写出。多线程，单文件集 I/O：很多处理器写进一个文件中。多线程，多文件集 I/O：很多处理器写到独立的文件中。这也叫做分布式 I/O。

当高分辨率模式在大量处理器（超过 1000 个）上运行时，分布式 I/O 同样存在性能问题。最新的方法是定义一个拥有 io_layout 的 IO 区域进而对所有处理器分组。每个组的第一个处理器从它的组中获得所有的数据并写到单个文件中（每个组的文件都会不同）。写入读出的文件数量和数据大小之间必须保持一定的平衡。io_domain 的布局也将依赖于网格区域的布局。图 1.7 给出了一个 IO 区域的例子。

18	19	20	21	22	23
12	13	14	15	16	17
6	7	8	9	10	11
0	1	2	3	4	5

图 1.7: 使用 IO 区域来控制 IO。网格区域的布局时(6,4)而 io_layout 是(2,2)。处理器 0 到处理器 23 被分为四个组：(0,1,2,6,7,8), (3,4,5,9,10,11), (12,13,14,18,19,20) and (15,16,17,21,22,23)。(0,3,12,15)这四个处理器将写入四个独立的文件中。

mpp_io_mo 提供了接口用于打开和关闭文件，从文件中读取元数据和数据，以及把数据和元数据写入文件中。代码框 1.7 给出了如何使用 mpp_io_mod 接口读取数据的例子。代码框 1.8 给出了如何使用 mpp_io_mod 接口写数据的例子

```
call mpp_open( unit, “foo.nc”, action=MPP_RDONLY,  &
            form=MPP_NETCDF, threading=MPP_MULTI, fileset=MPP_SINGLE )
 call mpp_get_info( unit, ndim, nvar, natt, ntime )
 allocate( vars(nvar)
 call mpp_get_fields ( unit, vars(:) )
 call mpp_read( unit, vars(1), domain, data)
 call mpp_close(unit)
```

代码框 1.7： 读取数据的例子。

```
call mpp_open( unit, “foo.nc”, action=MPP_OVERWR, &
          form=MPP_NETCDF, threading=MPP_MULTI, fileset=MPP_MULTI )
call mpp_write_meta( unit, id_x, 'X', 'km', 'X distance', 'X', domain=xdom, data=(/(i,i=1,nx)/) )
 call mpp_write_meta( unit, id_y, 'Y', 'km', 'Y distance', 'Y', domain=ydom, data=(/(i,i=1,ny)/) )
 call mpp_write_meta( unit, id_f, (/x,y/), 'Data', 'metres', 'Random data', pack=1 )
 call mpp_write( unit, x )
 call mpp_write( unit, y )
 call mpp_write( unit, data, domain, data)
 call mpp_close(unit)
```

代码框 1.8: 写数据的例子。

3.4 data_override

在运行的时候，大气，海冰，陆面以及/或者海洋分量模式可以通过 namelist 的选项 do_ice，do_land，do_ocean 来关闭。FMS 内部构件提供了在运行时覆盖分量边界场的方法（接口

data_override）。data_override_init 必须在调用 data_override 之前调用（只需要调用一次）。边界场可以用某个常数覆盖进去或者从指定的 netCDF 文件中获得。如果有必要将数据转换到模式的网格和时间上，那就需要使用空间和时间插值。这一信息有数据重载表（data override table）提供。在使用 data_override 之前，一个 data_table 必须被生成，它含有以下参数：gridname，fieldname_code，fieldname_file，file_name，ongrid 以及 factor。举个例子，如果来自海冰模式传递给海洋的海表面温度可使用观测的海表面温度数据覆盖，那么它的表格项如下：

```
cat > data_table <<EOF
   'OCN', 'sst_obs', 'sst', 'INPUT/sst_obs.nc', .false., 0.01
 EOF
```

图 1.2: data_override 表格的例子.

'OCN'　　--- 用于标识目标分量模式。
'sst_obs'　--- 在调用 data_override 时对应边界场的名字。
'sst'　　--- 在 netCDF 数据文件中场的名字。
'INPUT/sst_obs.nc'　---- 包含 SST 数据的文件名。
.false.　　--- 数据是否（true 或者 false）在模式网格上。
0.01　　--- 放缩比例。

接口 data_override 是用于在模式代码中覆盖边界数据的。data_override_init（只需要调用一次）必须在调用任何 data_override 前调用。代码框 1.9 给出了覆盖海表面温度和洋面风应力的例子

```
call data_overide_init()
call data_override('OCN','sst_obs',sst,Time)
call data_override('OCN','u_flux',u_flux,Time)
call data_override('OCN','v_flux',v_flux,Time)
```

代码框 1.9：使用 data_override 的源代码。

3.5 diag_manager

diag_manager 是一系列在分布式系统上并行诊断的简单调用。它是为把数据写成 netCDF 格式专门设计。diag_manager_mod 提供了一种可方便把数据写入磁盘的接口。它基于 FMS 代码的并行 I/O 接口。运行时的诊断设定是通过诊断表格来输入的。

从零维数组（标量）一直到三维数组以及带有时间蒙版的场的输出时间平均，diag_manager 都可以输出。默认情况下，一个场将被输出到全局格点上，用户也可以在 diag_table 中指定只输出一个区域的场。为了检查 diag_table 是否被正确的设置，用户应该把 diag_manager_nml 变量设定为 debug_diag_manager=true。这样，diag_table 的内容就会以标准输出打印出来。一个场仅仅在当它在模式代码中被注册而且在 diag_table 中指定的时候才会被输出出来。

代码框 1.10 给出了如何使用 diag_manager 接口的一个例子。diag_manager_init 会在初始化 diag_manager_mode 时被调用。而 diag_manager_end 则需要最后调用来把数据写入到磁盘上。diag_manager_init 和 diag_manager_end 在一个程序中只会被调用一次。

```
Call diag_manager_init
id_axis(1) = diag_axis_init('lon', lon, 'degrees_E', 'x', long_name='longitude',Domain2=Domain)
id_axis(2) = diag_axis_init('lat', lat, 'degrees_N', ‘y', long_name='latitude', Domain2=Domain)
id_eta_t  = register_diag_field ('ocean_model', 'eta_t', id_axis, time, 'surface height on T cells',
              'meter',  missing_value=-10.0, range=(/-10.0,10.0/))
used = send_data (id_sea_level, eta_t , time, rmask=mask)
call diag_manager_end
```

代码框 1.10: 使用 diag_manager_mod 接口的例子。

diag_table 包含了三个部分：全局，文件以及场。全局部分是表格的前两行包括了实验标题和基准日期。基准日期是用于作为时间单位的参考时间。基准日期必须大于或者等于模式的启动时间。日期包含了六个以空格隔开的整数：年，月，日，小时，分钟和秒。

文件部分的每一样都指定了一个输出文件。它包括六个项：“file_name”，output_freq，“output_freq_units”，format，“time_units”，“time_long_name”。“file_name”是将被输出的文件名。当 output_freq=0 时，每个时间步都会输出；当 output_freq=-1 时，会在运行结束前输出；当 output_freq>0，以 output_freq_units 为单位的频率进行输出。当 format=1 时，输出为 netCDF 格式，这也是唯一支持的格式。“time_units”是用来标记时间轴的单位。“time_long_name”

场部分的每一行都指定了一个输出的场。它包含 8 个项：“module_name”，“field_name”，“output_name”，“file_name”，“time_sampling”，“time_avg”，“other_opts”，“packing”。“module_name”是模块名字，比如“ocean_model”。“file_name”是通过 register_diag_field 注册的场名称。“output_name”是输出文件里场的名称，“file_name”是文件输出文件名。“time_avg”可以为.true.（时间平均）或者.false.（瞬时）。当“packing=1”时，输出是双精度的；当“packing=2”时，输出是浮点精度的，当“packing=4”时，输出被包装成 16 位整数。

表格 1.3 给出 一个诊断表格的例子。

```
cat >> diag_table <<EOF
CM2.1U_Control-1990_E1.M_3A
1990 1 1 0 0 0
"ocean_month",    1,  "months", 1, "days", "time",
“ocean_daily",     24,  "hours",  1, "days", "time",
"ocean_model","eta_t","eta_t","ocean_month","all",.true.,"none",2
"ocean_model","eta_t","eta_t","ocean_daily","all", .false.,"none",2
EOF
```

表格 1.3 一个诊断表格的例子。

综上所述，本章对整个可扩展建模系统做了一个概述。这主要包括 GFDL 可扩展建模系统的关键特征以及 FMS 构架的重要模块。大气，陆地表面，海洋表面和海洋分量利用标准耦合接口上的插槽在交换网格的表面边界层上耦合起来。所有的分量都包含在一个可执行程序中，但是它们可以串行调度或者并发运行。目前的代码已经可扩展到 O（1000）的处理器数量级，其中在每个大气时间步长中耦合快表面过程（通常 15 分钟），每个海洋时间步耦合慢过程（通常一个小时）。耦合一直到二阶精度都是守恒的。

FMS 架构提供了支持并行，输入/输出，对分量模式和耦合器的进行诊断以及为其提供边界强迫数据的接口。该架构应该可以很大程度上把模式开发者从机器相关的细节中脱离开。它同时还把 FMS 用户从消息通信库（MPI）和科学数据格式库（netCDF）中脱离。它还提供了不同分量模式的常用接口，同时对不同分量模式的诊断进行了标准化。