CESM 入门及移植

v6
2011/6/10

本文档的主要内容是 CESM 的简介和移植方法，是实际工作的总结，不揣简陋，写出来供 CESM 初学者参考，希望起到抛砖引玉的作用[1]。

1 基本概念

CESM 的全称是 **C**ommunity **E**arth **S**ystem **M**odel[2]。从字面来看，CESM 是“地球系统模式”。在进一步解释“地球系统模式”这一重要概念之前，先来看 CESM 能做什么？CESM 是一个地学数值模拟软件，通俗的说就是“地球模拟”。模拟什么？过去、现在及未来的地学过程。例如：

- 过去 30 年的大气过程模拟；
- 模拟当今极地冰盖的变化；
- 模拟在当今的碳排放条件下，未来几十年的气候演变，等等。

从上面的例子可以看出，地学模式研究的问题往往是大范围的，从空间和时间来看，都具有很大的延展性，例如大洋环流，大气环流等，时间跨度则可以是几十年，上百年，上千年[3]。受地学特点的限制，在没有计算机的时代，地学研究者主要靠实际观测来研究问题，无法做人为受控的实验。与其他学科相比，地学更多的是观测，这可能是地学的一个特点。计算机数值模拟则突破了这种限制，使得地学研究也能像其他学科做实验一样，变成了可重复的，而且能够通过这种技术手段，做到以前不能做的事情，例如研究古代的气候，除了通过化石等传统手段，利用数值模拟，能够提供新视角，产生新数据和新观点。利用数值模拟可以对未来的气候演化做出合理的科学预测，所以具有很大的现实意义。例如，目前正在进行的 IPCC AR5 的评估工作中，模式的开发和科学检验是十分重要的，关于碳排放、全球变暖的预估，都是建立在计算机模拟的基础上的。

初学者心中一定会有这样的疑问：**为什么地球的变化，能用计算机进行模拟呢？**地学研究，是将地球（包括相关的地外因素，例如太阳辐射，月球引力）视作了一个物理系统（现在的研究也包含了化学、生物因素等）。这个物理系统是相当复杂的，因此，地学专家首先要做出科学的近似和抽象，将无关的或次要的因素不予考虑。例如，海洋物理学一般不考虑海洋里面的生物活动，一般不考虑地壳向海洋输送的热量，甚至可以不考虑海表高度的变化。在地学研究的问题中，海洋可被看做是一个互相联通的、巨大的、有一定盐度的、广而“薄”的水体，这个水体的运动受到大陆轮廓和洋底的限制，还会与太阳辐射、大气运动等相关。因此大尺度的海洋的动力过程，可以用物理学进行描述，而且这样的描述很好的反映了实际的海洋运动。除了海洋，大气和陆面经过一番类似的抽象和近似的功夫，实际的地球圈就被抽象成了“地球模式（earth model）”。

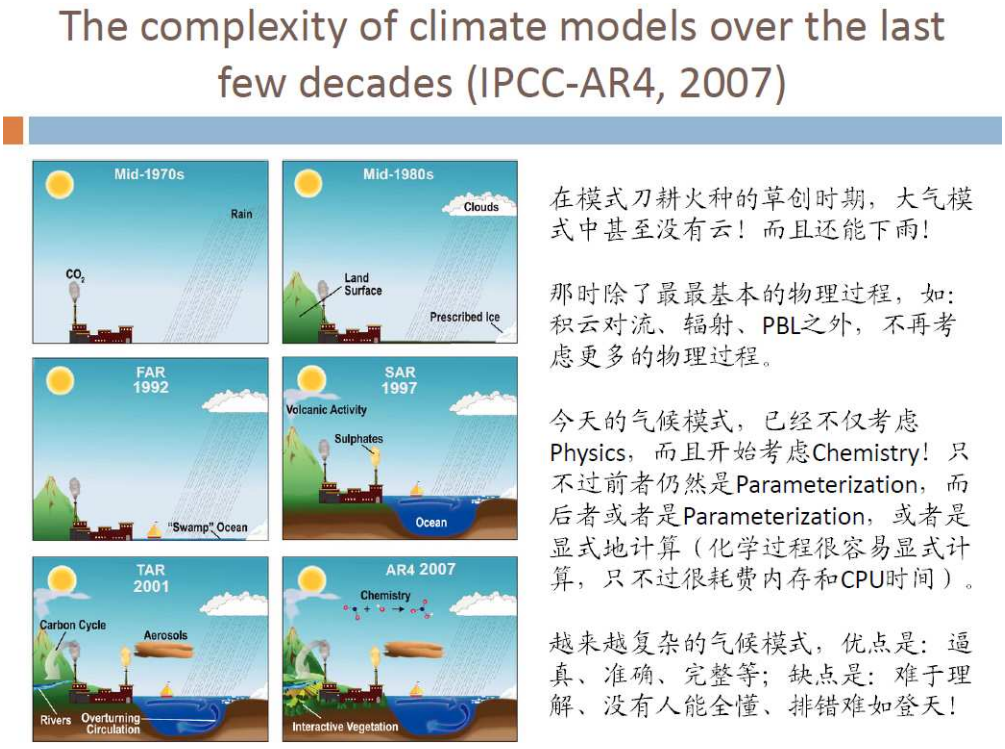
---

[1] 联系人王晖 wanghui@ncic.ac.cn 通信地址：北京 2704 信箱 中科院计算所高性能中心 邮编 100190
[2] 参考 http://www.cesm.ucar.edu/models/cesm1.0/
[3] 模拟也在小尺度上进行，例如海洋模拟，尺度可以从米级到百公里。

科学是求真的。利用地学模式模拟出来的过程，仅仅是存在于计算机中，并没有回答是否真实的问题。这个模拟世界中，与真实的地球一样有海洋和大陆，有阴晴雨雪，季节更迭，有大气环流和洋流，厄尔尼诺，温室效应，等等。利用可视化软件，还可以将数据很好的呈现出来，看到模拟的图像或动态过程。至于模拟世界能否很好的契合真实世界，以及如何去看待模拟结果，这是模式设计的核心问题之一，需要地学领域专家的分析总结。例如，参照历史观测资料，分析模式的模拟结果，哪些是很好的反映了真实的情况，哪些出现了较大的偏差。出现偏差的原因是什么，知道了误差来源，就可以对现有的物理和数学模型进行修正，对现有版本加以改进，推出新的更好的版本。从历史来看，现有的模式大都是有自己的历史渊源的，经历了长期的改进，演化，例如有的模式从最初的版本，逐渐发展出了庞大的分支，不同时期，不同国家/组织的科学家，都为模式的发展做出了贡献，使得模式能够更好的逼近真实的地球。

气候模式的出现和发展，与计算机的历史同步。世界上第一个模式是大气环流模式，1956 年创建。1963 年创建了第一个九层原始方程大气环流模式；1972 年第一个谱模式问世；二十世纪八十年代以后，逐渐形成了更大范围的气候系统模式；九十年代以后，学界开始形成地球系统模式，直到今天的格局。

地学模式领域经过半个多世纪的积累，拥有丰富的遗产和成果，有许多国家和机构都在组织、参与和建设模式领域。最初大气学家，海洋学家等都是在各自的领域研究和发明模式，例如大气模式 CAM，海洋模式 POP，这叫做独立分量。随着研究的深入，学者们发现研究地学问题往往要综合考虑各个地学领域的成果，将大气、海洋、陆面、海冰，陆冰等综合起来进行模拟，涉及各个子系统之间的交换。这样就产生了地学模式领域的耦合器。耦合器的作用就是将独立的分量模式通过耦合器连接起来，完成数据的转换和传输，以及处理同步和并发等问题。发展到今天，地学模式在朝着“全系统”的方向前进，综合考虑地球圈的各种因素，包括人类活动例如碳排放。所以，地学模式就从最初的大气、海洋等分量模式，发展到了“地球系统模式”。下图形象的显示了模式的发展历程。[4]



[4] 图片引用自 http://www.phy.pku.edu.cn/climate/class/cm2010/CM05-GCM-physics-1.pdf

有一篇 2008 年的论文可做入门材料，《地球系统模式发展展望》[5]。例如该文中提到，地球系统模式是“**全球变化研究的最重要的，不可替代的工具之一**”。该文还综述了地球系统模式以及各个分量模式存在的问题，并对未来发展趋势作出了展望。

另一方面，全球变化研究中，通过数值模拟得到的结论是否可靠？如果是模拟过去的气候，又有历史同期观测资料，那么可以与历史资料进行对照，这是评判模式好坏的一个主要的方法，如果一个模式连过去的真实过程都模拟不了，那么预测未来也就没有可信度。此外，由于时间尺度大，例如预测几十年之后的气候，这给检验带来了困难。国际上对 IPCC 的结论也存在争议，例如碳排放与全球变暖的关系。这其中既有纯科学的讨论，也有部分政治的因素[6]。气候模拟是在大尺度上，涉及因素非常多，物理的、化学的、生物的、人文的，模型是复杂的和综合的，即便是专家也要下一番功夫；国际上现有的气候模式不下百种，各有千秋，有很大的改进余地；计算机只是工具，模拟的效果如何，一方面受限于机器的计算能力，但根本上还是受限于人类的认识水平；因此，预测有失误是正常的，不能因此否定地球系统模式的价值，这也是地球系统模式工作前进的动力之一。

地学模拟数值计算，主要有以下三个要点：

- 一是要遵守基本物理定律。
  例如牛顿第二定律，质量守恒定律，热力学守恒定律，气体的有关定律，等等；
- 二是符合运动方程。
  例如连续方程，热力学方程，状态方程，水汽方程，等等；
- 三是在给定初始条件和边界条件下对方程进行求解。

例如，AGCM(Atmospheric General Circulation Model)的组成要素有：动力框架，物理过程，初值，边界条件。AGCM 更多内容请参考国家气候中心王在志的 ppt《大气模式发展与应用》。一般来说，其他模式的构建要素也都包含动力框架，物理过程，初值，边界等内容。更综合的模拟，还要考虑化学的，生物的因素。

例如，以大洋环流模式的设计过程为例。

“以基于有限差分方法的模式为例，一个比较完整的模式设计过程大致包括：

1）选择坐标系，根据一定的近似和假定写出模式的微分方程组和边界条件。

2）确定次网格尺度过程的参数化方案（详见第五讲）。

3）确定分辨率、水平网格、垂直分层、以及变量的分布方式（如‘B-grid’，‘C-grid’等），在此基础上生成模式地形、初始场和海表强迫场，这里涉及的是空间离散化和变量离散化的问题。在网格系统的选择方面，除了计算精度和便于构造差分近似方面的考虑以外，尽可能正确地描写重力波惯性波的传播过程一直是一个主要原则，可参看 Arakawa and Lamb（1977），Betteen, and Han（1981）等。

4）选择或设计具有适当精度、计算稳定的差分格式（尽可能使用简洁的格式以减小舍入误差），写出和模式微分方程组相应的差分方程组，检查其整体性质（如质量守恒，输运原理，能量守恒等），排除虚假的‘计算源汇’。

5）程序设计。作为一种初步规范的做法，可参看“LASG/IAP 气候系统海洋模式（LICOM1.0）参考手册”的第 3 部分（刘海龙等，2004）。”[7]

在具体讨论 CESM 之前，先来看一下 CESM 的历史和地位，以及这里为什么选择 CESM 作为案例。CESM 是地球系统模式家族中的一员，它的前身 CCSM 进入了 IPCC AR4，CESM 也将参加 IPCC AR5 的评估。CESM 的模式有自己的特点及优点，同时也有不足之处。我国

---

[5] 链接地址 http://www.lasg.ac.cn/UpLoadFiles/File/papers/2008/2008-wab-ztj-yyq-bw.pdf 。

[6] 参考 http://blog.sina.com.cn/s/blog_4b700c4c0100hxm1.html

[7] 摘自张学洪《大洋环流和海气相互作用的数值模拟》LASG 课程讲稿，下载链接 http://www.lasg.ac.cn/UpLoadFiles/File/misc/FGCM/2007a.doc

的地学领域研究者也开发了自己的地球系统模式，例如 LASG/IAP[8]。CESM 是一个开源项目，为各国的研究者提供了一个很好的范本。学习和研究 CESM，是为了了解国外同行的工作，取长补短，好的东西可以借鉴，帮助提升我国的地学模式研究实力，不断改进自己的地球系统模式的平台。

还需说明一点，我们此前未接触过地学专业和地球模式，凡涉及到地学，大都是借助从互联网搜集到的资料进行查阅和学习，理解肤浅，错误难免，希望求正于专家，能够聆听专家的声音。

1.1 模式和耦合器

CESM consists of five geophysical models: atmosphere (atm), sea-ice (ice), land (lnd), ocean (ocn), and land-ice (glc), plus a coupler (cpl) that coordinates the models and passes information between them. Each model may have "active," "data," "dead," or "stub" component version allowing for a variety of "plug and play" combinations.[9]

CESM 包含 5 个地学模式和 1 个耦合器。5 个地学模式包括大气（atm），海冰（ice），陆面（lnd），海洋（ocn），陆冰（glc）。1 个耦合器就是 cpl。耦合器用来协调各个模式的运行，在各个模式之间传递数据。

The active (dynamical) components are generally fully prognostic, and they are state-of-the-art climate prediction and analysis tools. Because the active models are relatively expensive to run, data models that cycle input data are included for testing, spin-up, and model parameterization development. The dead components generate scientifically invalid data and exist only to support technical system testing. The dead components must all be run together and should never be combined with any active or data versions of models. Stub components exist only to satisfy interface requirements when the component is not needed for the model configuration (e.g., the active land component forced with atmospheric data does not need ice, ocn, or glc components, so ice, ocn, and glc stubs are used).

五个模式，包括大气、海冰、陆面、海洋、陆冰，每一个又分为几个不同的组件版本：active，data，dead，stub。

active 组件是全功能的气候预测和分析工具，运行它们需要较多的资源。命名约定：cam 大气，clm 陆面，pop 海洋，cice 海冰，cism 陆冰。这些模式都是由来已久的著名的模式，所以，CESM 的设计不是从头来过，而是继承了大量的历史成果。这样一来 CESM 软件就呈现出模块化的特点，各个模式也可以独立运行。也可以将几个模式组合起来。

data 组件主要用于测试、加速（spin-up），和参数化。命名约定：datm 大气，docn 海洋，dlnd 陆面，dice 海冰。陆冰没有 data 组件。疑问：陆冰为什么没有 data 组件？

datm （data atm）是一个纯数据组件，读入“大气强迫”。

dlnd （data land）有以下的模式：纯的径流数据模式（读入径流数据）；纯的陆面数据模式（读入耦合器历史数据：atm/land 通量（fluxes），land albedos）；或者两者的结合。

docn (data ocean)有两种操作模式：纯数据模式，从输入数据集读取海洋 SSTs（海表温度），进行差分（interpolating in space and time），然后传递给耦合器；或者，计算更新的海表温度 SSTs，这种方法是基于分层的海洋模式（slab ocean model），读入底部海洋热通量辐合（bottom ocean heat flux convergence）与边界层深度（boundary layer depths），然后与从耦

[8] 参考 http://www.lasg.ac.cn/Sylm/2011/1/md4flqfjfy.htm 。
[9] 参考页面：http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc/x42.html 。

合器获得的大气/海洋和海冰/海洋通量一起使用。

dead 组件仅仅用于技术性的系统测试，不具有科学意义。命名约定：xatm 大气，xlnd 陆面，xocn 海洋，xice 海冰。陆冰没有 dead。疑问：陆冰为什么没有 dead 组件？

stub 组件是为了满足接口的需求。例如，active land 组件与 atmospheric data 相结合，不需要 ice，ocn，glc，所以使用 ice，ocn 和 glc 的 stub 版本。命名约定：satm 大气，slnd 陆面，socn 海洋，sice 海冰，sglc 陆冰。stub 其实就是 inactive，什么也不做，只是放在那里做占位符。

耦合器 cpl 只有 active 版本。耦合器是 CESM 的核心部件。五个气候模式并不是 CESM 原生开发的，而是早在 CESM 之前就已经存在了。CESM 更像是一个框架，将五个模式集成起来，让它们能够协调一致的工作。

另一方面，五个模式的独立性也会带来一些问题。例如，五个模式最初可能是无联系的，软件的配置和编译方法不会完全相同，如何把它们统一地纳入 CESM，这是 CESM 要解决的问题。统一的工作往往需要各方的通力合作，例如 POP 海洋模式最初是独立开发和发展的，后来为了能够纳入 CESM 的体系，有针对性的对软件进行了修改。现在的很多模式还加入了对 ESMF 框架的支持，以增强接口的标准性，编程的规范性，提高互联互通能力，并能降低总体成本。

1.2 compset 和 case

compset 是一个完整的、预定义的、可重复运行的“模拟实验”。在 CESM 中运行的地球模拟程序都是以 compset 形式定义的。从 compset 出发，再加上模式分辨率，工作路径等参数，可以构造出具体的可运行实例，后者叫做 case。

CESM 对 compset 的定义是这样的：将 CESM 中的若干个组件加以联合，就构成了组件集 compset（component set）。例如，将以下的组件联合起来：xatm，xlnd，xice，xocn，sglc。也就是将所有的 DEAD 组件加上陆冰的 stub，构成了一个 compset。在 CESM 中将这个 compset 命名为 X_PRESENT_DAY，缩写为 X。[10]

严格的说，compset 不仅仅是“组合”，还包含各个组件的配置和 namelist 设置。软件的灵活性决定了 CESM 的 case 配置可以有很强的“随意性”，例如将任意几个组件联合起来，每个组件的参数随意设置，这样的一个 case 也能够运行，并得出结果。但是，并不是任意的组合都有**科学意义**。只有那些具有科学意义的配置，才是地学研究者可用的配置。

列表中可以看出，compset 考虑了各种条件，例如按照历史时间进行区分，pre-industrial，present day；考虑化学因素，例如 Carbon Nitrogen，等等。

为了帮助理解，再举两个 compset 的例子。

G_NORMAL_YEAR 是一个 compset，缩写为 G，包含的组件有大气数据 datm，陆面数据 dlnd，海冰 cice 和海洋 pop2，再加一个陆冰 stub。对这个 compset 的描述是：Coupled ocean ice with COREv2 normal year forcing。也就是说，这个 compset 是将海冰和海洋模式耦合，加上 COREv2 正常年份强迫，后者就是大气数据 datm 和陆面数据 dlnd。

F_AMIP_CAM5 是一个 compset，缩写为 FAMIPC5。包含的组件有大气模式 cam，陆面模式 clm，海冰模式 cice，以上三个 active 耦合在一起，加上数据海洋 docn（SST 海温数据），陆冰 stub。F 开头的 compset 一般来讲都是这样的组合。对这个 compset 的描述是：AMIP run for CMIP5 protocol with cam5。[11]

---

[10] CESM 目前支持的 compset 列表在下面的网址：
http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc/a3043.html 。

[11] AMIP 是大气模式比较计划，CMIP 是耦合模式比较计划（Coupled Model Intercomparison Project）。

1.3 格点和输入数据

地学的数学物理方程一般是偏微分方程。为了将其在计算机上求解，需要将数据离散化。这就涉及到数据的表示。grid 即格点的大小，代表了模拟的精细程度。例如，输入的格点数据以文件形式保存，各个模块将读入格点文件（grids file）。耦合器需要映射权重文件（mapping weights file），因为连接到耦合器的各个模块，需要的数据可能是不同的，例如数据表示方法，或者格点大小不同，在初始化的时候，各个模块将向耦合器发送格点数据；耦合器根据实际情况，利用 mapping weights file，检查数据的一致性。疑问：mapping weights file 是如何具体作用的？

CESM 支持的格点类型：

- single point 单点。
- finite volume 有限体积。大气，陆面常用。docn，dice 也支持。
- spectral 谱。大气，陆面常用。docn，dice 也支持。
- cubed sphere 只支持 cam。
- displaced pole 海洋和海冰。
- tripole 海洋和海冰。

格点文件命名约定参考 CESM 页面的“CESM Grids”部分。[12]

CESM 支持的格点格式，可参考 CESM 的相应页面。[13]

下面举例说明。在 inputdata/cpl/cpl6/ 目录下有这样的一个映射文件：map_r05_to_gx1v6_e1000r300_090226.nc 。该文件是一个 mapping weights 文件。是从 runoff (.5 degree)到 ocn，即径流到海洋。gx1v6，说明格点类型为 displaced pole，这是海洋模式采用的格点类型，1 是解析度，说明是 1-degree。6 是格点版本。疑问：e1000r300 是什么含义？再如映射文件 map_gx1v6_to_fv1.9x2.5_aave_da_090206.nc，对它的描述是：ocn to atm mapping file for states，也就是从海洋到大气的映射。fv1.9x2.5，这是 finite volume 类型的格点，大气模式常用，解析度大约是 2-degree。疑问：aave_da 是什么含义？

还需指出，这些映射文件并不是在 CESM 内部生成的，而是作为 CESM 的输入文件。实际上，这些文件需要独立下载。输入数据 inputdata 的下载点在一个独立的 svn 目录。[14] 疑问：这些文件是如何产生的？

输入数据一般是根据观测资料产生的。举个例子，北师大全球变化研究院开发了通用路面模式 CoLM。数据来源，例如 USGS(24 类)/IGBP(19 类）地表覆盖类型资料[15]，FAO(17 类)土壤质地资料。默认使用 30 角秒分辨率资料。以上面的原始资料为输入，处理之后的输出资料包括格点沙土含量、粘土含量，格点土壤表层颜色（共 8 类），格点包含的次网格类型及权重(mosaic)。[16]

1.4 CESM 模式概览

前面所讲还是有点抽象，下面以个别的程序为例，考察模式工作的流程。

---

[12] 参考 http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc/x42.html 。

[13] 参考 http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc/a3342.html 。

[14] 参考 https://svn-ccsm-inputdata.cgd.ucar.edu/trunk/inputdata/ 。

[15] IGBP 是国际地圈生物圈计划(International Geosphere-Biosphere Program，IGBP)的英文名缩写。美国地质勘探局（United States Geological Survey）简称 USGS。

[16] 参考 pdf 文档《通用陆面过程模式 CoLM 的框架和应用-纪多颖》 下载地址 http://cess.tsinghua.edu.cn/earth/index.php?option=com_jotloader&task=files.download&cid=50&Itemid=42

我们选择了一个相对简单的模式，data ocean 也就是 docn。请参考前面对 docn 的描述。docn 实际上有两个工作方式，一个是纯数据方式，另一个是有更多的耦合参与其中。[17]

CSM 通量耦合器（CSM Flux Coupler）与其他模式之间是通过定义好的接口规范来交互的，所以耦合器并不知道一个 ocean 模式是 active 的还是 inactive 的。data ocean 属于后者，它与耦合器的接口跟 active ocean 并没有区别，但是 docn 的工作只是向耦合器传送 SST 海温数据，而忽略来自耦合器的强迫数据，所以这是个“dummy”模式。docn 的实际用途主要是为了在耦合了 SST 数据的情况下观察 active 大气模式的表现。SST 数据是 docn 从数据文件读取的，原始来源则是来自气候资料，例如 Shea, Trenberth, & Reynolds (STR)。[18]

docn 的输入数据包括：

- SST 数据。
  从一个 netcdf 文件读入，包含了 12 个月的 SST 数据。2x2 degree grid。
- 域（Model domain）。
  从一个 netcdf 文件读入。主要用于与耦合器的数据交换。uses a latitude/longitude grid only, with 1d coordinate arrays x(i) & y(j), and a 2d domain mask mask(i,j)。使用的是经纬度 grid，一维数组 x(i)和 y(j)，二维掩码用不同的值代表所在位置是海洋、陆地，或者内陆区域（例如里海）。
- namelist 。
  输入参数列表。例如与耦合器的交换数据的频度。
- 从耦合器收到的数据。
  docn 直接无视这些数据，也就是不做任何的处理。[19]

  docn 的运行步骤分为两步：首先运行 setup 脚本建立一系列的文件；然后运行 run 脚本运行模式。[20]

  每一次建立一个应用，都要经历这些步骤：拷贝文件和源码文件；编译；运行。每次建立的应用存在于一个独立的目录。这些具体步骤都写在了脚本文件中，由脚本文件负责建立一个 case 的全过程。

  除了 docn 之外，还有很多其他的模式，每一个模式的构建步骤也都是类似的。因为每个模式或都曾经是一个独立的项目，所以具体细节并不完全统一。cesm 则是将这么多的模式组合起来工作，所以工作量和复杂性都有所增长。

2 CESM 移植

前面已经提到，CESM 的开发不是一时一地的，而是一个“集大成”的软件平台。CESM 包含了来自不同时期不同组织开发的模式代码。CESM 有着十分丰富的功能，例如从它支持的 compset 列表就可以看出，CESM 能够完成多种多样的模拟实验。而且，CESM 还是在不断发展的，例如可以加入新的模式，建立新的 compset。

CESM 的“综合性”的特点导致了**测试覆盖率**的问题。例如，能够运行模式 A，并不意味着运行模式 B 没有问题。因为模式 A 和模式 B 用到了不同的源码和配置。有时候，在一个模式的配置文件中修改一个 namelist 的值，也会导致源码文件列表的变化，从而引起重新

---

[17] 参考 http://www.cesm.ucar.edu/models/ocn-docn/ 。
[18] 参考 http://www.cgd.ucar.edu/cas/catalog/surface/sst/str/
[19] 参考 http://www.cesm.ucar.edu/models/ocn-docn/docn4.0/userguide.html
[20] 参考同上

编译。那么，能不能从软件配置管理的角度做一些工作，使得一次编译之后，就能够覆盖CESM 所有的文件，然后运行 make test，覆盖所有的支持的 compset？我认为这是有可能的，但是从 CESM 现状来看，那将是一个巨大的任务。

另一方面，交给科学家使用的 CESM 应该是一个经过“充分测试”的，随时可用的软件。例如不允许出现“运行 A 没问题，运行 B 总是出错”。也要避免繁琐的配置选项和参数调整。目前的解决办法就是所谓的“移植”。移植就是让 CESM 中所有的 compset 都能够在某个特定的软硬件平台上跑起来，如果都运行成功了，就是“移植成功”。或者最低限度要保证使用者感兴趣那部分模式能够跑起来。此外，移植也要考虑到性能问题，优化问题等等。至少目前来看，CESM 开发组还在维持现状：对一个新的平台来说，移植工作是不可避免的。

CESM 内置了对一些**特定软硬件平台**的支持。如下表所示[21]。

```
MACHINES:    name (description)
   bluefire (NCAR IBM p6, os is AIX, 32 pes/node, batch system is LSF)
   brutus_po (Brutus Linux Cluster ETH (pgi/9.0-1 with open_mpi/1.4.1), 16 pes/node, batch system LSF, added by UB)
   brutus_pm (Brutus Linux Cluster ETH (pgi/9.0-1 with mvapich2/1.4rc2), 16 pes/node, batch system LSF, added by UB)
   brutus_io (Brutus Linux Cluster ETH (intel/10.1.018 with open_mpi/1.4.1), 16 pes/node, batch system LSF, added by UB)
   brutus_im (Brutus Linux Cluster ETH (intel/10.1.018 with mvapich2/1.4rc2), 16 pes/node, batch system LSF, added by UB)
   edinburgh_lahey (NCAR CGD Linux Cluster (lahey), 8 pes/node, batch system is PBS)
   edinburgh_pgi (NCAR CGD Linux Cluster (pgi), 8 pes/node, batch system is PBS)
   edinburgh_intel (NCAR CGD Linux Cluster (intel), 8 pes/node, batch system is PBS)
   franklin (NERSC XT4, os is CNL, 4 pes/node, batch system is PBS)
   hadley (UCB Linux Cluster, os is Linux (ia64), batch system is PBS)
   hopper (NERSC XT5, os is CNL, 8 pes/node, batch system is PBS)
   intrepid (ANL IBM BG/P, os is BGP, 4 pes/node, batch system is cobalt)
   jaguar (ORNL XT4, os is CNL, 4 pes/node, batch system is PBS)
   jaguarpf (ORNL XT5, os is CNL, 12 pes/node, batch system is PBS)
   kraken (NICS/UT/teragrid XT5, os is CNL, 12 pes/node)
   midnight (ARSC Sun Cluster, os is Linux (pgi), batch system is PBS)
   prototype_atlas (LLNL Linux Cluster, Linux (pgi), 8 pes/node, batch system is Moab)
   prototype_columbia (NASA Ames Linux Cluster, Linux (ia64), 2 pes/node, batch system is PBS)
   prototype_frost (NCAR IBM BG/L, os is BGL, 8 pes/node, batch system is cobalt)
   prototype_nyblue (SUNY IBM BG/L, os is BGL, 8 pes/node, batch system is cobalt)
   prototype_ranger (TACC Linux Cluster, Linux (pgi), 1 pes/node, batch system is SGE)
   prototype_schirra (NAS (NASA) IBM p5+, os is AIX, 2 pes/node, batch system is PBS)
   prototype_ubgl (LLNL IBM BG/L, os is BGL, 2 pes/node, batch system is Moab)
   generic_ibm (generic ibm power system, os is AIX, batch system is LoadLeveler, user-defined)
   generic_xt (generic CRAY XT, os is CNL, batch system is PBS, user-defined)
   generic_linux_pgi (generic linux (pgi), os is Linux, batch system is PBS, user-defined)
   generic_linux_lahey (generic linux (lahey), os is Linux, batch system is PBS, user-defined)
   generic_linux_intel (generic linux (intel), os is Linux, batch system is PBS, user-defined)
   generic_linux_pathscale (generic linux (pathscale), os is Linux, batch system is PBS, user-defined)
```

为了让一个具体的 CESM 地学模式应用，在一个特定的软硬件平台上跑起来，需要完成以下的两项工作：

1 系统环境建立和验证。包括编译环境和运行时环境。首先应当着眼于编译环境的建立，这是为了得到自己平台的 CESM 可执行程序 ccsm.exe[22]。只需在一个节点上建立编译环境；然后，为了在自己的平台上运行 CESM 的 case，还要对运行时环境进行设置。运行时环境一般是指 MPI 运行时环境，以及作业管理系统。运行时环境的建立涉及到所有的计算节点。

2 CESM 代码移植和验证。涉及具体代码的“修改-编译-试运行-调试-再修改-…”，这个过程可能有多次的迭代，直到最后完全成功。

---

[21] 参考 cesm 官方页面：http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc/c2161.html 。

[22] ccsm.exe 虽然有.exe 后缀，但其实是一个标准的 ELF 可执行程序。不同的 CESM case 有各自的构建目录，最后生成的主程序都叫做 ccsm.exe。

2.1 下载 CESM

CESM 的代码和数据是用 SVN 管理的。代码和数据要分别下载。下载前需要先在 CESM 网站注册一个账号。注册是免费的。

2.1.1 源代码

CESM 代码的下载，目前最新版为 cesm1_0_2。下载方法

```
# svn co https://svn-ccsm-release.cgd.ucar.edu/model_versions/cesm1_0_2/
```

或者用图形化 svn 客户端下载，例如 windows 上的 TortoiseSVN。

2.1.2 输入数据

CESM 输入数据的下载地址在：

```
https://svn-ccsm-inputdata.cgd.ucar.edu/trunk/inputdata/
```

CESM 目前的数据集全部加起来超过 1TBytes，而且今后必然还会增长。所以下载输入数据文件，都是“按需下载”，只下载必要的输入数据文件。在 build case 这一步，构建脚本根据文件名列表，检查输入数据文件是否齐全，如果缺少数据文件，自动调用 svn 下载；如果下载出错，将提示用户自行下载。

输入数据文件的大小不一，小的仅仅数 kB，大的超过 1GB。数据文件的格式有几种，例如 ascii 文本格式，netcdf 格式，二进制数据格式。其中 netcdf 格式是与平台无关的，二进制数据格式则要注意字节序的问题。CESM 一般使用 big endian 存放二进制数据。

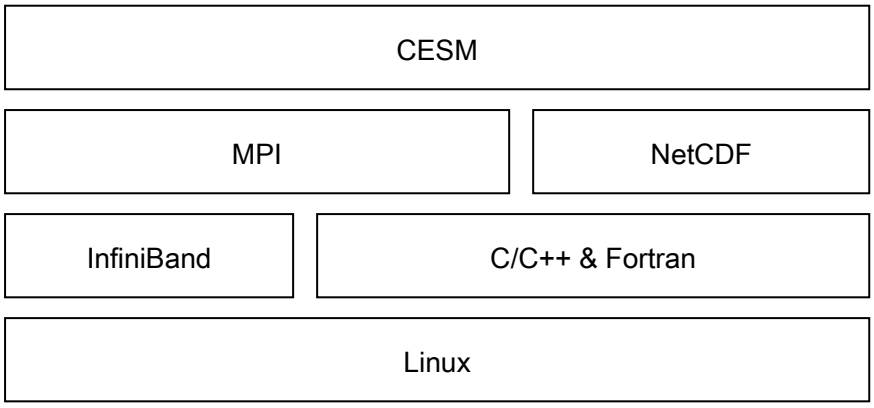
输入数据的下载，推荐用下载工具，例如“迅雷”[23]，传输协议是 https。使用迅雷可以方便下载管理，比 svn 好用；更重要的是比较稳定，长时间的、大文件下载不会出错。相比之下，浏览器下载可能不够稳定，例如有时候显示下载完成，但是得到的文件是不完整的，这样的数据放到 CESM 里面，开始也能够运行起来，系统不会检查数据文件的完整性，结果导致运行时出错。

还要注意一个重启文件兼容的问题。经我们测试，CCSM4 的重启文件不能直接用于 cesm1_0_2。例如，其中 CICE 模块的.nc 重启文件中缺少某些 cesm1_0_2 重启需要的量。

- 建议用下载工具下载输入数据文件。
- 全部的数据集很大，应当在 build case 这一步自动下载根据提示“按需下载”。
- 下载的大文件，要检查是否完整。例如，根据文件字节数进行判断。
- CESM 的纯二进制数据文件一般都是大端字节序。
- CCSM4 的重启文件与新版的 CESM 不兼容。

[23] http://dl.xunlei.com/

2.2 编译环境



CESM 的编译环境如上图所示。从下至上依次说明。工作顺序不能颠倒。例如，先要确定 c/c++和 Fortran 编译器，下一步才能够编译 MPI 库。

2.2.1 操作系统

操作系统的选择并无特别的讲究，以稳定为上，兼顾效率。生产性的 HPC 环境已经安装了操作系统，用户没有选择的余地。如果在自己搭建的实验环境运行 CESM 过程中，内核有报错，那么可以尝试更换内核或操作系统。例如，我们曾发现在某个 xen 内核上运行 CESM 报错，更换为对应的普通内核之后，问题消失，该问题是否与 xen 有关？事后未予深究。总之，当运行时莫名其妙的出错或退出，要留意系统日志/var/log/messages 是否含有关于 ccsm.exe 的报错。操作系统版本，我们用过 openSUSE 11.2，CentOS 5.3，CentOS 5.5 等，都没有问题。

- 最好不用 xen 虚拟化内核。

2.2.2 InfiniBand

InfiniBand 的有无并不影响 CESM 移植，但是对 CESM 的实际运行效率有很大的影响。在生产性的 HPC 环境中，很可能安装了 InfiniBand 环境，例如 OFED 驱动。请向系统管理员咨询 InfiniBand 的情况。如果机器上没有 InfiniBand，那就用以太网接口好了，后者只是性能差，并不影响程序的移植和功能的完整性、正确性。

2.2.3 编译器

C/C++和 Fortran 编译器。机器上可能安装有不同版本的编译器，需要选择其中的一个，或者需要重新安装编译器。我们在移植过程中分别使用过 GNU 编译器 gcc/gfortran 和 Intel 编译器 icc/ifort。需要注意的问题：

- 系统可能默认没有安装任何 fortran 编译器。所以，先检查一下你的系统，看看已经安装了哪些编译器，有没有自己需要的版本。例如在 suse 平台上安装 gfortran，需要安装三个 rpm 包：libgfortran44，gcc44-fortran，gcc-fortran。
- 要明确自己用的是 32 位版本的编译器，还是 64 位版本的编译器。例如，有的 64 位 cpu 实际跑的是 32 位的系统，与 64 位的环境相比，性能一定会有不同。
- c/c+和 fortran 编译器的版本号要完全一致。例如运行以下命令检查版本号是否完全一致。

```
# gfortran --version
# gcc -version
```

Intel 编译器的命令行是 icc 和 ifort。

- 如果用 gfortran，建议使用 4.4 及以上的版本。4.3 版本在编译 CESM 的时候可能出错，不建议使用。以下命令查询系统上安装的 fortran 软件。注意输出没有包含从源码安装的 fortran 编译器。

```
# rpm --queryformat "%{NAME}-%{VERSION}.%{ARCH}\n" -qa |grep fortran
```

2.2.4 MPI 库

MPI 库的实现版本有很多，例如 OpenMPI，MPICH，MVAPICH 等。在同一个机器上可能安装了多个 MPI 的实现。所以主要问题是，需要确定自己使用哪个 MPI 的实现。我们在移植过程中分别使用过 OpenMPI 和 MVAPICH。需要注意的问题：

- 在二进制安装的时候避免重复安装。向系统中安装某个 MPI 之前，首先检查系统中是否已经安装过了。例如，不要安装两个不同版本 openmpi 的 rpm。
- 从源码安装，在编译 MPI 之前，设置好 InfiniBand 的相关选项。不同 MPI 实现的 IB 支持选项的设置方法不同，请参考各自的安装文档。例如，mvapich 在编译之前，先编辑 make.mvapich.gen2 文件，如下图所示。然后运行 make.mvapich.gen2 进行编译。

```
# Mandatory variables.  All are checked except CXX and F90.
IBHOME=${IBHOME:-/usr/local/ofed}
IBHOME_LIB=${IBHOME_LIB:-/usr/local/ofed/lib64}
PREFIX=${PREFIX:-/home/wanghui/mvapich-nocoll}
export CC=${CC:-/opt/intel/bin/icc}
export CXX=${CXX:-/opt/intel/bin/icc}
export F77=${F77:-/opt/intel/bin/ifort}
export F90=${F90:-/opt/intel/bin/ifort}
```

- 从源码安装，在编译 MPI 之前，还要选择正确的编译器。举例如上图所示。编译 MPI 注意观察 configure 的输出或日志文件，看看编译器的选择是否自己所期望的。
- 实验环境下，可以将 MPI 直接安装在自己的个人目录下面。例如上面的例子，用户 wanghui 将 mvapich 安装在了/home/wanghui/ mvapich-nocoll/目录。这是一个很好的实践，可以避免与系统的和别人的 MPI 发生冲突。
- 可以仿效 mpi-selector 的做法，给自己的 bash 命令行设置好环境变量。仍然以上面的 mvapich 安装为例，用户 wanghui 安装完成 mvapich 之后，编辑自己的 ~/.bash_profile，内容如下图所示。注意 LD_LIBRARY_PATH 是有下划线的。脚本的写法可以参考 mpi-selector 的目录/var/mpi-selector/data/中的脚本文件。其

中，下图中的/opt/intel/bin 是 Intel 编译器 icc 和 ifort 等程序所在目录；mvapich 的可执行程序，例如编译器 mpicc 和 mpif90 等程序所在目录为/home/wanghui/mvapich-nocoll/bin；mvapich 的库文件 libmpich.a 等文件所在目录则是/home/wanghui/mvapich-nocoll/lib。用户 wanghui 要退出 bash 再次登录，之后按照后面的方法进行验证，确认 MPI 安装正确。

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
       . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH

export PATH

alias mpirun_my='/home/wanghui/mvapich-nocoll/bin/mpirun_rsh'

## intel
if test -z "`echo $PATH | grep /opt/intel/bin`"; then
   PATH=/opt/intel/bin:${PATH}
       export PATH
fi

# my MPI PATH
if test -z "`echo $PATH | grep /home/wanghui/mvapich-nocoll/bin`"; then
   PATH=/home/wanghui/mvapich-nocoll/bin:${PATH}
       export PATH
fi

# my MPI LD_LIBRARY_PATH
if test -z "`echo $LD_LIBRARY_PATH | grep /home/wanghui/mvapich-nocoll/lib`"; then

LD_LIBRARY_PATH=/home/wanghui/mvapich-nocoll/lib${LD_LIBRARY_PATH:+:}${LD_LIBRARY_PATH}
   export LD_LIBRARY_PATH
fi

# my MPI  MANPATH
if test -z "`echo $MANPATH | grep /home/wanghui/mvapich-nocoll/man`"; then
   MANPATH=/home/wanghui/mvapich-nocoll/man:${MANPATH}
   export MANPATH
fi

```

- 安装完成之后，确认安装正确。例如 openmpi，运行 ompi_info 命令，确认 openmpi 的设置都正确。再如，运行 mpicc --version 及 mpif90 --version 确定 MPI 的编译器版本是否正确和一致。还可以运行 env 命令检查环境变量$PATH 以及$LD_LIBRARY_PATH 是否正确设置了。下图所示，用户 wanghui 的 mpicc 和 mpif90 都是正确的，分别指向 Intel 编译器 icc 和 ifort，而且版本号相同 。

```
[wanghui@gh87 ~]$ which mpicc
~/mvapich-nocoll/bin/mpicc
[wanghui@gh87 ~]$ which mpif90
~/mvapich-nocoll/bin/mpif90
[wanghui@gh87 ~]$ mpicc --version
icc (ICC) 12.0.4 20110427
Copyright (C) 1985-2011 Intel Corporation.  All rights reserved.
[wanghui@gh87 ~]$ mpif90 --version
ifort (IFORT) 12.0.4 20110427
Copyright (C) 1985-2011 Intel Corporation.  All rights reserved.
```

2.2.5 NetCDF

NetCDF 是 CESM 用来存储科学数据的文件格式，netcdf 的文件后缀是.nc，二进制文件，而且与平台无关。我们安装的是 netcdf-4.1.1，从源码编译安装。特别需要观察 configure 的输出，看看编译器是否是自己想要的。例如，用户 wanghui 以下面的选项进行配置，显式指定了编译器为 Intel 的 icc 和 ifort。

```
[wanghui@gh87   netcdf-4.1.1]$      ./configure   --prefix=/usr/local
--disable-docs CC=icc CXX=icc F77=ifort
[wanghui@gh88 netcdf-4.1.1]$ make
[wanghui@gh88 netcdf-4.1.1]$ make check
注意如果不能访问大网，个别测试可能会有失败
[wanghui@gh88 netcdf-4.1.1]$ sudo make install
[wanghui@gh87 netcdf-4.1.1]$ nc-config --all

This netCDF 4.1.1 has been built with the following features:

  --cc       -> icc
  --cflags    ->  -I/usr/local/include
  --libs     -> -L/usr/local/lib -lnetcdf -L/usr/kerberos/lib64 -lcurl
-lgssapi_krb5 -lkrb5 -lk5crypto -lcom_err -lresolv -ldl -lidn -lssl
-lcrypto -lz

  --cxx      -> icc
  --has-c++   -> yes

  --fc       -> ifort
  --fflags    -> -g -I/usr/local/include
  --flibs     -> -L/usr/local/lib -lnetcdf -L/usr/kerberos/lib64 -lcurl
-lgssapi_krb5 -lkrb5 -lk5crypto -lcom_err -lresolv -ldl -lidn -lssl
-lcrypto -lz
  --has-f77   -> yes
  --has-f90   -> yes

  --has-dap   -> yes
  --has-nc2   -> yes
  --has-nc4   -> no
  --has-hdf5  -> no
  --has-hdf4  -> no
  --has-szlib -> no

  --prefix    -> /usr/local
  --includedir-> /usr/local/include
  --version   -> netCDF 4.1.1
```

归纳 netcdf 安装的注意问题。

- 使用正确的编译器。可以在 configure 命令行选项显式指定。
- 安装完成，运行 nc-config 命令检查安装是否正确。
- HDF5 是可选项。我们安装的的 netcdf 没有支持 HDF5，不影响 CESM 的编译和运行。

CESM 的编译环境设置，放到后面讲述。

2.3 运行时环境

CESM 运行时环境，涉及两个方面的内容：并行应用的运行时环境；机群作业管理系统。

2.3.1 并行应用环境

并行应用的运行时环境的建立，主要是创建用户，给用户指定一个机群范围内都能访问的个人目录，让用户能够在机群的节点间无密码 ssh 登录。涉及以下的一些工作步骤。

1. 一个所有节点共享的/home 路径。一般机群系统都是这么做的。
2. 在机群上创建一个用户。例如，wanghui。有的机群没有建立统一的账号管理，例如 NIS，LDAP 等，这种情况需要在各个节点上分别创建 wanghui 用户。
3. 创建 SSH 钥匙对。用户 wanghui 在一个节点上运行 ssh-keygen -t rsa 命令，创建自己的 SSH 钥匙对。不同的 SSH 套件可能程序的名字略有差别，例如可能不叫 ssh-keygern，但也有类似的命令行。密钥的保护口令留空，可实现无密码登录。
4. 机群内无密码登录。用户 wanghui 运行 ssh-copy-id 或类似命令。结果是将自己的公钥拷贝到了~/.ssh/authorized_keys 文件。手工拷贝公钥也是可以的。既然/home 路径是机群范围内共享的，这样就实现了用户 wanghui 在所有节点间的无密码 ssh 登录。
5. 给 wanghui 设置 sudo 权限。超级管理员编辑/etc/sudoers 以及/etc/group。这一步可选。

以上这些工作完成之后，可以尝试编译和运行一个 MPI 并行应用。举例如下图所示。

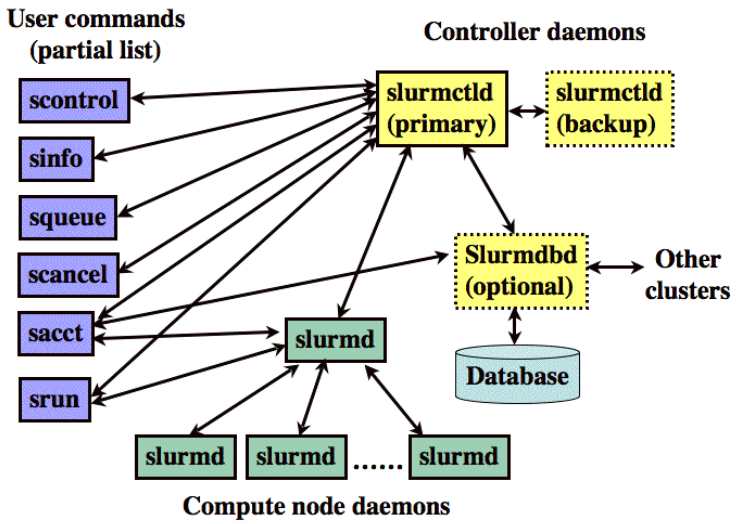
```
[wanghui@gh87 installtest]$ pwd
/home/wanghui/mvapich-1.2rc1/installtest
编译一个并行程序。该示例程序源码 pi3f90.f90 位于 mvapich 的源码目录。
[wanghui@gh87 installtest]$ mpif90  -O2 pi3f90.f90 -o  pi3
注意编辑默认的 host 列表文件 ~/.mpirun_hostfile
然后运行这个程序。
[wanghui@gh87 installtest]$ mpirun_rsh -np 12 ./pi3
 Process         10  of         12  is alive
 Process          0  of         12  is alive
 Process          1  of         12  is alive
 Process          5  of         12  is alive
 Process          8  of         12  is alive
 Process          2  of         12  is alive
 Process          3  of         12  is alive
 Process          6  of         12  is alive
 Process          7  of         12  is alive
 Process          9  of         12  is alive
 Process         11  of         12  is alive
Enter the number of intervals: (0 quits)
 Process          4  of         12  is alive
1000000
  pi is approximately: 3.1415926535898784  Error is: 0.0000000000000853
Enter the number of intervals: (0 quits)
```

到了这一步，可以说，CESM 需要的“外部环境”基本搭建完成。接下来的工作就是移植 CESM 本身了。在简单讲述作业管理之后，我们将结束本节。

2.3.2 作业管理系统

作业管理系统主要讲一下 SLURM。对于 CESM 的移植和测试来说，实验用的机群系统规模一般都不大，用户不多，计算任务不繁忙，因此手工运行程序就足够了。当把移植好的 CESM 放到百万亿次、千万亿次甚至更快更强的超级计算机上运行的时候，这些系统都有专业的团队负责日常维护和管理，通过作业管理软件，向系统提交用户的计算任务，作业管理软件采取某些定义好的策略，管理提交的所有作业。例如，可以为作业分配不同的资源，设定启动和停止时间，为作业设定优先级，记录作业的实际执行情况，等等。

CESM 的设计考虑了与作业管理软件的结合。这里简要介绍 SLURM[24]与 CESM 的结合问题。由于作业管理问题不是本文的重点，所以这里直接结合一个具体案例来讲解。涉及到 CESM 的 case 生成的内容，后面还会有详细讲述。



上图是 SLURM 的组成结构图。其中 slurmd 在每个节点上都要安装，slurmctld 是管理 daemon，在主控节点上安装。scontrol，sinfo，squeue 等是用户命令行。在我的测试环境中，对上图进行了简化，例如不包含 slurmdbd 和 slurmstld(backup)。测试环境包含两个主机，主机 quarter01 运行 slurmd，主机 quarter02 运行 slurmd 和 slurmctld。

[24]参考 https://computing.llnl.gov/linux/slurm/quickstart.html

```
quarter02:~ # sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
debug*       up   infinite     2   idle quarter[01-02]
quarter02:~ # scontrol show nodes
NodeName=quarter01 Arch=i686 CoresPerSocket=1
   CPUAlloc=0 CPUErr=0 CPUTot=8 Features=(null)
   OS=Linux RealMemory=1 Sockets=8
   State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1
   Reason=(null)
NodeName=quarter02 Arch=i686 CoresPerSocket=1
   CPUAlloc=0 CPUErr=0 CPUTot=8 Features=(null)
   OS=Linux RealMemory=1 Sockets=8
   State=IDLE ThreadsPerCore=1 TmpDisk=0 Weight=1
   Reason=(null)
下面运行 hostname 命令
quarter02:~ # srun -N2 hostname
quarter01
quarter02
```

slurm 如何与 CESM 结合？我们的做法是在 CESM 的配置文件 config_machines.xml 中，为特定的 Machine 指定 BATCHSUBMIT 为 SLURM 的作业提交命令 sbatch。

例如，如下图所示，利用 create_newcase 脚本生成一个 CESM 的 case，该 case 属于 X compset，模式分辨率为 f45_g37，case 目录为 test_X/。

```
quarter02:/shared/CESM/cesm1_0_2/scripts # ./create_newcase  \
   -case test_X \
   -res f45_g37 \
   -compset X_PRESENT_DAY \
   -mach quarter \
   -scratchroot /ptmp/username
quarter02:/shared/CESM/cesm1_0_2/scripts/test_X # ./configure -case
```

configure -case 将生成四个脚本文件

- test_X.quarter.build
- test_X.quarter.clean_build
- test_X.quarter.run
- test_X.quarter.submit

其中，test_X.quarter.build 和 test_X.quarter.clean_build 分别是 build 和 clean 脚本，与作业管理无关。test_X.quarter.submit 是作业提交脚本，在这个脚本中，简单的通过 SLURM 的提交命令 sbatch 提交作业 test_X.quarter.run。

```
quarter02:/shared/CESM/cesm1_0_2/scripts/test_X # cat
test_X.quarter.submit
#! /bin/csh -f

./check_case || echo "check_case failed" && exit -99

sbatch test_X.quarter.run

set sdate = `date +"%Y-%m-%d %H:%M:%S"`
echo "run submitted $sdate" >>& CaseStatus
```

sbatch 将运行脚本 test_X.quarter.run 提交到 SLURM 作业管理系统。运行脚本 test_X.quarter.run 同样是一个 csh 脚本，内容涉及三部分工作：

1）开始部分首先给出作业选项，就是那些以“#SBATCH”开头的注解行；

2）然后要检查和设置运行程序需要的环境变量；

3）最后调用 mpirun 或类似程序，启动 CESM 主程序 ccsm.exe。

2.4 创建 CESM Machine

从 CESM 的官网资料来看，移植是一个尝试和纠错的过程，从最简单的 case 开始，然后逐步推广和验证，经过反复多次的迭代，最终就可以确认，CESM 在这个特定平台上移植成功了，可以正式的投入使用。这个尝试和纠错的过程，不管对新手还是专家都是必经之路。例如，从一个初始的模板创建 case，然后 build case，通常都会报错，问题诸如编译器的路径，编译器版本，或者库没有正确设置，或者是编译器的选项设置不当，以及环境变量的问题，等等。每个平台都有自己的设置，要根据平台的情况来修改 CESM 的相关的配置文件。前面列出了 CESM 目前支持的所有 Machines。CESM 的移植，就是要针对自己的计算平台，创建一个新的 CESM Machine。

```
1. Macros.xxxx
2. mkbatch.xxxx
3. env_machopts.xxxx
4. config_machines.xml
```

其中，xxxx 代表新的 CESM Machine 的名字，例如我们移植的平台有：quarter，gbnode，ghnode。前三个文件 Macros.xxxx，mkbatch.xxxx，env_machopts.xxxx 是新创建的，格式都是纯文本。最后一个 config_machines.xml 则是需要修改的，向其中添加关于新 Machine 的内容。

移植 CESM 到一个新的平台，涉及到的文件只有上面的四个，这四个文件都位于 CESM 源码的一个子目录：**scripts/ccsm_utils/Machines/**。

Macro 文件主要包含基本的编译选项；mkbatch 文件是作业批处理相关的内容；env_machopts 包含 CESM Machine 的环境变量配置；config_machines 则包含基本的配置信息，例如 case 目录、输入数据所在目录、批处理命令等。

下面主要以 ghnode Machine 为例，讲述移植的时候要注意的问题。ghnode 平台是位于我们机房的一组完全同配置的机器，基本的配置情况见后面的“CESM 测试”一章。

与 ghnode 有关的四个文件是：Macros.ghnode，mkbatch.ghnode，env_machopts.ghnode，

config_machines.xml。其中前面的三个文件是新创建的，再配合 config_machines.xml 文件，就为 CESM 增加了一个 Machine。运行脚本 create_newcase –list 可以看到 ghnode。

```
[wanghui@gh87 scripts]$ ./create_newcase -list|grep ghnode
   ghnode (node with IB in ncic machine room)
```

因为系统都是 linux，所以移植无需从零开始，首先以 CESM 自带的三个文件为模板：env_machopts.generic_linux_intel，mkbatch.generic_linux_intel，Macros.generic_linux_intel，以它们为基础进行修改，得到符合 ghndoe 的配置。

- 以 CESM 自带的 Machine 文件为模板，修改得到自己平台的 Machine 文件。
- 移植 CESM 到某个新的平台，涉及到四个文件：创建三个文件，修改一个文件。

2.4.1 Macros 文件

Macros 文件的内容是，在特定的 CESM Machine 上编译 CESM 所需的宏定义和编译选项。下面主要讲述我们认为在移植过程中要注意的问题。更详细的内容请直接参考文件 scripts/ccsm_utils/Machines/Macros.ghnode。

首先，指定 MPI 编译器。

```
34 FC          := mpif90
35 CC          := mpicc
```

第二，指定 netcdf 路径。

```
50 NETCDF_PATH   := /usr/local
51 INC_NETCDF    := $(NETCDF_PATH)/include
52 LIB_NETCDF    := $(NETCDF_PATH)/lib
53 MOD_NETCDF    := $(NETCDF_PATH)/include
```

其中 INC_NETCDF 是 netcdf.h 和 netcdf.inc 所在路径；LIB_NETCDF 是 libnetcdf.a 所在路径；MOD_NETCDF 是 netcdf.mod 所在路径。

第三，指定 MPI 路径和 MPI 库的名称。

```
55 INC_MPI      :=  /home/wanghui/mvapich-nocoll/include
56 LIB_MPI      :=  /home/wanghui/mvapich-nocoll/lib
57 MPI_LIB_NAME  := mpich
```

我们没有指定 PNETCDF_PATH，INC_PNETCDF，LIB_PNETCDF。

第四，指定 FFLAGS。这一步很关键，如果指定的编译选项不合适，后面实际编译的时候就会报错。我们指定的编译选项是：

```
65 FIXEDFLAGS    := $(CPPDEFS) -g -132 -fp-model precise -convert
big_endian -assume byterecl -ftz -traceback
66 FREEFLAGS     :=
67 FFLAGS        := $(CPPDEFS) -g -fp-model precise -convert big_endian
-assume byterecl -ftz -traceback
```

其中，-132 是指定每行的最大长度，该选项只对固定格式的 fortran 源码有效；-convert big_endian 是指定从外部读取数据的时候按照大端字节序；-fp-model 用来控制浮点计算的语义，这里取值 precise；-assume byterecl 是假定打开 unformatted 文件时的记录长度按照 byte 计算；

再举一个 FFLAGS 的例子供参考。我们在 gbnode 上做 CESM 移植，编译器为 gfortran。其中-fconvert=big-endian 是保证二进制数据外部读取按照大端字节序；-ffree-line-length-none 是为了解除 132 个字符的行长度限制。

```
# gbnode 的 FFLAGS 选项。编译器为 gfortran。
FFLAGS        := $(CPPDEFS) -g -fconvert=big-endian
-ffree-line-length-none -fno-range-check -fcray-pointer -std=gnu
-D__G95__
```

第五，为 mct 和 pio 传入 configure 参数。由于 mct 和 pio 是独立编译的，所以这里传入了很多 configure 参数，见粗体字部分。如果不传入这些参数，构建可能会出现意想不到的问题。

```
156 # GENERIC_USER
157 # Options passed to the mct and pio build are set here
158
159 ifeq ($(MODEL),mct)
160   #add arguments for mct configure here
161   CONFIG_ARGS += CC="$(CC)" FC="$(FC)" F90="$(FC)" INCLUDEPATH="-I$(INC_MPI)"
FFLAGS="$(FFLAGS)"
162 endif
163
164 ifeq ($(MODEL),pio)
165   ifneq ($(strip $(PIO_CONFIG_OPTS)),)
166     CONFIG_ARGS += $(PIO_CONFIG_OPTS)
167   endif
168   CONFIG_ARGS += CC="$(CC)" F90="$(FC)" NETCDF_PATH="$(NETCDF_PATH)"
LIB_NETCDF="$(LIB_NETCDF)" MPI_INC="-I$(INC_MPI)" FFLAGS="$(FFLAGS)"
169 endif
```

2.4.2 mkbatch 文件

mkbatch 主要是为了生成 run 脚本。在 run 脚本中有作业管理的内容。我们在移植过程中，没有使用作业管理系统，所以与作业管理相关的内容，没有添加到 mkbatch.ghnode。运行脚本的部分，主要是添加 mpirun 或类似命令。以 ghnode 为例。mpirun_rsh 是 MVAPICH 的加载命令；-np 120 是加载 120 个进程；主机列表文件这里没有显式指定，说明将读取默认的主机列表~/.mpirun_hostfile。如下图所示。更详细的内容请直接参考文件

scripts/ccsm_utils/Machines/mkbatch.ghnode。

```
 61 cat >> ${CASEROOT}/${CASE}.${MACH}.run << EOF1
 62 # ----------------------------------------------------------------------
 63 # Run the model
 64 # ----------------------------------------------------------------------
 65
 66 sleep 5
 67 echo "cd \$RUNDIR"
 68 cd \$RUNDIR
 69 echo "\`date\` -- CSM EXECUTION BEGINS HERE"
 70
 71
#===============================================================================
 72 # GENERIC_USER
 73 # Launch the job here.  Some samples are commented out below
 74
#===============================================================================
 75
 76 setenv OMP_NUM_THREADS 1
 77 echo "Remember edit ~/.mpirun_hostfile."
 78 echo "mpirun_rsh -np 120 ./ccsm.exe >&! ccsm.log.\$LID"
 79 mpirun_rsh -np 120 ./ccsm.exe >&! ccsm.log.\$LID
 80
 81 wait
 82 echo "\`date\` -- CSM EXECUTION HAS FINISHED"
 83
 84 EOF1
```

2.4.3 env_machopts 文件

env_machopts.ghnode 是一个 csh 脚本文件，内容十分简单。这个文件就是为了设置某些 shell 环境变量。我们的 env_machopts.ghnode 实际只包含如下几行内容。

```
 1 #! /bin/csh -f
22 setenv MPI_PATH /home/wanghui/mvapich-nocoll
23 setenv MVAPICH_LIB_PATH /home/wanghui/mvapich-nocoll/lib
24 setenv MVAPICH_INCLUDE /home/wanghui/mvapich-nocoll/include
26 setenv PATH ${MVAPICH_LIB_PATH}:${MVAPICH_INCLUDE}:${PATH}
```

猜测：可以将这个文件置空。因为环境变量的设置事实上在 Macros 文件中都完成了，这里的设置其实是重复的，没有任何必要。

- env_machopts 文件很可能没有用处。

2.4.4 config_machines.xml

config_machines.xml 是 CESM 的 Machine 配置文件。该文件有很多个<machine/>字段组成，每个 machine 字段描述了一个 CESM Machine。与 ghnode 有关的字段如下所示。其中，EXEROOT 为 case 指定工作目录；DIN_LOC_ROOT_CSMDATA 指定输入数据所在目录；BATCHSUBMIT 指定作业提交命令；MAX_TASKS_PER_NODE 指定没节点最大的任务数，这里设定为与每节点 CPU core 的数目相同。

```
<machine MACH="ghnode"
         DESC="node with IB in ncic machine room"
         EXEROOT="/home/wanghui/tmp/$CASE"
         OBJROOT="$EXEROOT"
         LIBROOT="$EXEROOT/lib"
         INCROOT="$EXEROOT/lib/include"
         DIN_LOC_ROOT_CSMDATA="/home/wanghui/inputdata"
         DOUT_S_ROOT="UNSET"
         DOUT_L_HTAR="FALSE"
         DOUT_L_MSROOT="UNSET"
         OS="Linux"
         BATCHQUERY="squeue"
         BATCHSUBMIT="sbatch"
         GMAKE_J="1"
         MAX_TASKS_PER_NODE="12"
         MPISERIAL_SUPPORT="FALSE" />
```

其中 DIN_LOC_ROOT_CSMDATA 是 inputdata 目录；EXEROOT 是 case 运行的根目录；DOUT_S_ROOT 是“短期 case”保存实验数据和 restart 文件的目录，默认值为”UNSET”，这个目录位于 EXEROOT 目录下；GMAKE_J 是 build 的时候 gmake 的线程数，在多核 cpu 上可以设置为一个不大于 cpu 核数的数字。不过，如果发现 build 有问题，可尝试将 GMAKE_J 设置为“1”。

- GMAKE_J=”1”是最稳妥的设置。

2.5 修订 CESM 源码

经过上面的所有工作之后，CESM 移植是否已经成功了？还没有。接下来需要运行各个 compset，以及 CESM 定义的基准测试程序，确定 CESM 的各个 case 都能正常运行，无论是最简单的 X compset，还是 all active 的 B compset 等；要确定各项功能都表现正常，例如，特别是 restart 功能。

我们在移植过程中，逐渐的发现了 CESM 源码的一些问题。这些问题，不能简单的通过调整编译参数，或修改某个脚本来解决，只能直接修改 Fortran 源码。

CESM 基本是用 fortran 语言实现的。修改源码一定要慎重。如果不是地学专家或者地学模式的开发者，不要轻易去改动 fortran 源码，特别是绝对不能修改算法。我们移植过程中对 fortran 源码做了少量的修改，但都是在不涉及算法的情况下。下面罗列一下我们做的主要的修改。更详细的修改情况请直接参考我们修改之后的 CESM 源码。

还需指出，我们发现错误往往与编译器相关。所以，如果在你的平台上没有出现相同的错误，那也是不奇怪的。另一方面，这也反映了 CESM 的代码还需要进一步的锤炼。代码应该做到尽量与编译器版本不相关。

2.5.1 指针初始化

- 错误类别：运行时错误。
- 问题文件：models/atm/cam/src/chemistry/utils/tracer_data.F90。
- 错误描述：指针未赋初值，指针没有指向合法的地址，然后在 ccsm.exe 运行时，错误的调用释放函数，导致 segfault。现象为在 F_AMIP_CAM5 (FAMIPC5) 测试

中，出现“Address not mapped”的错误。

- 解决办法：文件中凡是定义 pointer 的地方，都赋初始值 null。例如将

```
real(r8), pointer, dimension(:) :: curr_data_times
```

修改为

```
real(r8), pointer, dimension(:) :: curr_data_times => null()
```

- 问题总结：一般来说指针引起的错误，最终报错点可能出现在多个不同的位置，要根据函数调用栈进行排查，修改的位置并不是一目了然的。所以在编码的时候就要养成良好的习惯，可避免类似的问题。也许有的 fortran 编译器支持给指针自动赋初值 0 的选项，但是最好不要依赖编译器，以保证代码的可移植性。

2.5.2 引用空指针

- 错误类别：运行时错误。
- 问题文件：models/lnd/dlnd/dlnd_comp_mod.F90
  models/utils/mct/mct/m_GlobalSegMap.F90
- 错误描述：运行时退出。调用栈信息举例如下。

```
[gb03:24293] [ 0] /lib64/libpthread.so.0 [0x3e8dc0eb10]
[gb03:24293] [ 1] ./ccsm.exe(__m_globalsegmap_MOD_lsize_+0xa1) [0x825fc9]
[gb03:24293] [ 2] ./ccsm.exe(__dlnd_comp_mod_MOD_dlnd_comp_init+0x29fe) [0x4c896e]
[gb03:24293] [ 3] ./ccsm.exe(MAIN__+0xbe16) [0x4757c6]
[gb03:24293] [ 4] ./ccsm.exe(main+0x2a) [0x98980a]
[gb03:24293] [ 5] /lib64/libc.so.6(__libc_start_main+0xf4) [0x3e8d01d994]
[gb03:24293] [ 6] ./ccsm.exe [0x4695c9]
[gb03:24293] *** End of error message ***
```

- 解决办法。在引用指针的时候需要预先判断，如果指针处于悬空状态，那么不能引用。如下图所示。从逻辑上讲，只有 GSMap%pe_loc 合法，才能够进一步引用 GSMap%pe_loc(n)。所以，这里的修改不会改变算法。

```
--- a/models/utils/mct/mct/m_GlobalSegMap.F90
+++ b/models/utils/mct/mct/m_GlobalSegMap.F90
@@ -1554,11 +1555,15 @@

   local_size = 0

-  do n=1,ngseg
-     if(GSMap%pe_loc(n) == myID) then
-       local_size = local_size + GSMap%length(n)
-     endif
-  end do
+  if( ASSOCIATED (GSMap%pe_loc) ) then
+     do n=1,ngseg
+       if(GSMap%pe_loc(n) == myID) then
+          local_size = local_size + GSMap%length(n)
+       endif
+     end do
+  endif

--- a/models/lnd/dlnd/dlnd_comp_mod.F90
+++ b/models/lnd/dlnd/dlnd_comp_mod.F90
@@ -401,23 +401,28 @@
     if (my_task == master_task) write(logunit,F00) ' initialize gsmaps'
     call shr_sys_flush(logunit)

-    lsize_s = mct_gsmap_lsize(gsmap_s,mpicom)
+    lsize_s = 0
+    if( ASSOCIATED (gsMap_s) ) then
+       lsize_s = mct_gsmap_lsize(gsMap_s,mpicom)
+    else
+       write(logunit,F00) ' WARNING: gsMap_s is null pointer '
+    endif
```

2.5.3 变量类型匹配

- 错误类别：编译错误。
- 问题文件：ncdio.F90。
- 错误描述：见下图。

```
cesm1_0_2/models/lnd/clm/src/main/ncdio.F90:5059.21:
      cols(:)    = nan
                   1
Error: Conversion of an Infinity or Not-a-Number at (1) to INTEGER
```

- 解决办法：查看 cols 的定义：

```
integer :: cols(maxpatch)              ! grid cell columns for scam
```

出错语句的作用，是给数组 cols 赋一个不合理的值，作为数组的初始值，这是常见的做法，但是赋值的时候，出现了类型不匹配（浮点到整型）的低级错误？查看相关代码，-1 是 cols 数组的一个不合理值，因此可以作为初始值。源码修改为

```
 cols(:)    = -1
```

2.5.4 类型未定义

- 错误类别：编译错误。
- 问题文件：solar_data.F90 等多个文件。
- 错误描述：提示没有找到 pio_types 中定义的类型。这个错误在用 Intel 编译器的时候出现，涉及到多个文件。gfortran 没有报告此类错误。
- 解决办法：增加 use pio_types 语句。举一个例子。

```
--- a/models/atm/cam/src/chemistry/utils/solar_data.F90
+++ b/models/atm/cam/src/chemistry/utils/solar_data.F90
@@ -6,6 +6,7 @@ module solar_data
   use spmd_utils,   only: masterproc
   use abortutils,   only: endrun
   use pio
+  use pio_types, only: file_desc_t
   use cam_pio_utils, only : cam_pio_openfile
   use cam_logfile,  only: iulog
```

2.5.5 符号未找到

- 错误类别：编译错误。
- 问题文件：models/csm_share/shr/shr_sys_mod.F90
- 错误描述：链接时提示“undefined reference to `system_'”。gfortran 报错。Intel 编译器未报错。
- 解决办法：见下图所示。不过，我们不了解这样修改的负面作用。从运行情况来看，没有发现问题。

```
--- a/models/csm_share/shr/shr_sys_mod.F90
+++ b/models/csm_share/shr/shr_sys_mod.F90
@@ -44,7 +44,7 @@ SUBROUTINE shr_sys_system(str,rcode)
    integer(SHR_KIND_IN),external    :: ishell ! function to envoke shell
command
 #endif
 #if (defined OSF1 || defined SUNOS || (defined LINUX && !defined __G95__)
|| (defined LINUX && !define
d CATAMOUNT))
-   integer(SHR_KIND_IN),external    :: system ! function to envoke shell
command
#endif

    !----- local -----
@@ -80,7 +80,8 @@ SUBROUTINE shr_sys_system(str,rcode)
 #endif

 #if (defined OSF1 || defined SUNOS || defined LINUX && !defined CATAMOUNT
|| defined __G95__)
-   rcode = system(str)
+   call system(str,rcode)
#endif
```

2.5.6 判断字符串相等

- 错误类别：运行时错误。
- 问题文件：models/drv/shr/seq_infodata_mod.F90
- 错误描述：见下图所示。由于 rest_case_name 可能包含多余的\x0，trim()不能正确得到 rest_case_name 字符串，导致 47 行的判断总是为真。
- 解决办法：在 if 判断之前，将\x0 的部分填充为空格，见 40-45 行。trim()能够正确处理空格。

```
26 --- a/models/drv/shr/seq_infodata_mod.F90
27 +++ b/models/drv/shr/seq_infodata_mod.F90
28 @@ -1563,6 +1563,7 @@ subroutine seq_infodata_Check( infodata )
29     integer :: lastchar                    ! Last character index
30     integer :: rc                          ! Return code
31
32 +   integer :: i               ! loop count
33  !-------------------------------------------------------------------------------
34  ! Notes:
35  !-------------------------------------------------------------------------------
36 @@ -1611,6 +1612,12 @@ subroutine seq_infodata_Check( infodata )
37 call shr_sys_abort(subname//': start_type invalid = '//trim(infodata%start_type))
38     end if
39
40 +   do i=1,SHR_KIND_CS
41 +      if ( infodata%rest_case_name(i:i) == CHAR(0) ) then
42 +         infodata%rest_case_name(i:i) = ' '
43 +      end if
44 +   enddo
45 +
46     if ((trim(infodata%start_type) == seq_infodata_start_type_cont ) .and.  &
47         (trim(infodata%case_name)  /= trim(infodata%rest_case_name))) then
48   write(logunit,'(10a)') subname,' case_name =',trim(infodata%case_name),':', &
```

2.5.7 字符串长度匹配

- 错误类别：编译错误。
- 问题文件：ocn/source/passive_tracers.F90
- 错误描述：见下图。Intel 编译器报错。

```
ocn/source/passive_tracers.F90(312): error #7938: Character length
argument mismatch.  [TADVECT_CTYPE_PASSIVE_TRACERS]

tadvect_ctype_passive_tracers(ecosys_ind_begin:ecosys_ind_end), &
------------------------^
```

- 解决办法：不要直接以 tadvect_ctype_passive_tracers()作为 ecosys_init()的传入参数，因为前者的返回值与后者要求的输入参数类型不匹配。因此，先定义一个符合 ecosys_init()参数要求的 local 变量 buf_loc，长度为 char_len。buf_loc 专门用来存放 tadvect_ctype_passive_tracers()的返回值，因为只是长度不一样，而且 buf_loc 的缓冲区更大，所以这里的自动类型转换一定是安全的。Intel 编译器在这个问题上比较严格，gfortran 不报错。

```
188 --- a/models/ocn/pop2/source/passive_tracers.F90
189 +++ b/models/ocn/pop2/source/passive_tracers.F90
190 @@ -198,6 +198,8 @@
191    character (char_len) :: sname, lname, units, coordinates
192    character (4) :: grid_loc
193
194 +  character (char_len), dimension(ecosys_tracer_cnt) :: buf_loc
195 +
196  !-----------------------------------------------------------------------
197  !  register init_passive_tracers
198  !-----------------------------------------------------------------------
199 @@ -303,10 +305,11 @@
200  !-----------------------------------------------------------------------
201
202    if (ecosys_on) then
203 +     buf_loc = tadvect_ctype_passive_tracers(ecosys_ind_begin:ecosys_ind_end)
204       call ecosys_init(init_ts_file_fmt, read_restart_filename, &
205                        tracer_d(ecosys_ind_begin:ecosys_ind_end), &
206                        TRACER(:,:,:,ecosys_ind_begin:ecosys_ind_end,:,:), &
207 -                      tadvect_ctype_passive_tracers(ecosys_ind_begin:ecosys_ind_end), &
208 +                      buf_loc, &
209                        errorCode)
```

2.5.8 数据读取格式

- 错误类别：运行时错误
- 问题文件：models/ocn/pop2/source/overflows.F90
- 错误描述：提示错误信息为“Fortran runtime error: Constant string in input format”。gfortran 编译器有这个问题。Intel 编译器没报错。
- 解决办法：将 format()修改为更通用的形式。见下图所示。

```
--- a/models/ocn/pop2/source/overflows.F90
+++ b/models/ocn/pop2/source/overflows.F90
@@ -2215,16 +2215,16 @@
      108 format(2x,1PE27.18)
 ! kmt changes, if any
      read(mu,1090) ovf(n)%num_kmt
-     1090 format(2x,i10,20x,'! number of kmt changes')
+     1090 format(2x,i10,20x) ! number of kmt changes
      do m=1,ovf(n)%num_kmt
        read(mu,1091) ovf(n)%loc_kmt(m)%i
-       1091 format(2x,i10,20x,'! i grid box index for kmt change')
+       1091 format(2x,i10,20x) ! i grid box index for kmt change
        read(mu,1092) ovf(n)%loc_kmt(m)%j
-       1092 format(2x,i10,20x,'! j grid box index for kmt change')
+       1092 format(2x,i10,20x) ! j grid box index for kmt change
        read(mu,1093) ovf(n)%loc_kmt(m)%korg
-       1093 format(2x,i10,20x,'! korg  original grid box k index')
+       1093 format(2x,i10,20x) ! korg  original grid box k index
        read(mu,1094) ovf(n)%loc_kmt(m)%knew
-       1094 format(2x,i10,20x,'! knew  new     grid box k index')
+       1094 format(2x,i10,20x) ! knew  new     grid box k index
      end do
```

2.5.9 模块重启失败

- 错误类别：运行时错误。
- 问题文件：不详。应该是那些与 restart 有关的文件，特别是与 CICE 模块重启相关的代码。从 CESM 网站的资料来看，CICE 的 netcdf 格式重启功能支持，应该是后来加入的。我们在运行 B1850CN 的时候，发现 restart 功能不正常，而且是 CICE 模块报错。估计是写入 CICE restart 文件的数据有误，但确切原因不详。
- 错误描述：B1850CN restart 之后，运行时出错，导致模式运行终止。错误属于模式内部逻辑报错，程序本身的运行没有异常，例如没有报告 fortran 运行时错误，程序段错误之类。出错信息如下。

```
istep1:        0    idate:    10106    sec:        0
 Thermo energy conservation error
 istep1, my_task, i, j:          1          0          2          5
 Flux error (W/m^2) =  4.64076121648152647E-003
 Energy error (J) =   8.3533701896667480
 Initial energy = -394609593.44356591
 Final energy =   3070534184.9539928
 efinal - einit =   3465143778.3975587
 Input energy =   3465143770.0441885
 istep1, my_task, iblk =          1          0          1
 Global block:          1
 Global i and j:          1          4
 Lat, Lon:  -74.950465454514614      -36.700000000015606
(shr_sys_abort) ERROR: ice: Vertical thermo error
(shr_sys_abort) WARNING: calling shr_mpi_abort() and stopping
```

- 解决办法：开启 CICE 模块的 binary 重启选项。涉及的文件有以下 6 个：

  cice/src/drivers/cpl_mct/ice_comp_mct.F90

  cice/src/drivers/cpl_share/ice_restart.F90

  cice/bld/build-namelist

  cice/bld/namelist_files/namelist_defaults_cice.xml

  cice/bld/namelist_files/namelist_definition.xml

  scripts/ccsm_utils/Tools/st_archive.sh

  具体请参考我们修改的 CESM 源码。

3 CESM 测试实例

本章主要讲述我们在移植和测试 CESM 过程中用到的平台，测试的方法及过程，主要测试结果和分析。

3.1 测试平台

我们先后在三个平台上移植和测试了 CESM，创建了三个 CESM Machine，分别叫做 quarter，gbnode，ghnode。

3.1.1 quarter

最初的测试环境 quarter，只包含两个节点，通过一个千兆交换机相连。节点配置如下。注意到，实验环境的两个测试节点总计有 CPU 核 2node x 2 way/node x 4core/way = 16 core。

主机名	quarter01	quarter02
角色	计算节点	IO 节点 & 计算节点
IP 地址	10.10.102.206	10.10.102.217
硬件	双路 AMD Opteron 4 核 CPU，主频 2.2GHz； 内存 6G	双路 AMD Opteron 4 核 CPU，主频 2.2GHz； 内存 4G
网络	千兆以太网	千兆以太网
操作系统	SUSE Linux Enterprise Server 11 (i586)	openSUSE 11.2 "Emerald"
内核版本	2.6.27.19-5-pae #1 SMP 2009-02-28 04:40:21 +0100 i686 athlon i386 GNU/Linux	2.6.31.5-0.1-desktop #1 SMP PREEMPT 2009-10-26 15:49:03 +0100 i686 athlon i386 GNU/Linux
Fortran	gcc-fortran-4.3-62.198	gcc-fortran-4.4-4.2.i586
MPI	openmpi-1.4.3 安装路径/usr/local	openmpi-1.4.3 安装路径/usr/local
NetCDF	netcdf-4.1.1 安装路径/usr/local	netcdf-4.1.1 安装路径/usr/local
SLURM	slurm- 2.1.14 安装路径 /usr/local 运行 slurmd	slurm- 2.1.14 安装路径 /usr/local 运行 slurmctld slurmd
CESM 目录	直接 mount 来自 IO 节点的目录： mount –t nfs quarter02:/shared/CESM /shared/CESM	作为 IO 节点，对外共享 NFS 路径：/shared/CESM 该目录下包含三个子目录： cesm1_0_2/ 源码目录，包括脚本和输出 inputdata/ 输入数据集，下载得到。 run/ 运行目录。

3.1.2 gbnode

在移植取得初步经验之后，为了测试计算规模比较大的 case，又在一个 7 节点的 cluster 环境下移植了 CESM，并进行测试。这 7 个节点是相同型号的曙光服务器，基本配置是完全相同的。

主机名	gb03,gb04,gb05,gb07,gb08,gb09	gb01
角色	计算节点	IO 节点
IP 地址	10.10.102.[345789]	10.10.102.1
硬件	2 路 4 核 Intel(R) Xeon(R) CPU E5620 开了 HT 内存 12G	同左
网络	千兆以太网	千兆以太网
操作系统	CentOS release 5.5 (Final)	同左
内核版本	2.6.18-194.el5 #1 SMP Fri Apr 2 14:58:14 EDT 2010 x86_64 x86_64 x86_64 GNU/Linux	同左
Fortran	GNU Fortran (GCC) 4.4.0 20090514 (Red Hat 4.4.0-6)	N/A
MPI	openmpi-1.4-4.el5 安装路径/usr/lib64/openmpi/1.4-gcc/	N/A
NetCDF	netcdf-4.1.1 安装路径/usr/local	N/A
CESM inputdata	10.10.102.1:/home 448G 178G 247G 42% /home 10.10.102.1:/mnt/nfs 448G 178G 247G 42% /mnt/nfs	/home 和 /mnt/nfs 位于同一个物理分区

3.1.3 ghnode

ghnode 是一组同构的机器，配置如下表所示。移植到 ghnode 一方面是因为这些节点配置了 InfiniBand；另一方面还因为 ghnode 也是 GPU 加速平台，可为今后算法优化提供必要的硬件支持。另外，ghnode 的规模还可以扩大。

主机名	gh80,gh84,gh87,gh88,gh89	gh01
角色	计算节点	IO 节点
IP 地址	10.10.108.8[04789]	10.10.108.1
硬件	2 路 6 核 Intel(R) Xeon(R) CPU X5650 12 Cores/节点 内存 24G NVIDIA Tesla	同左
网络	千兆以太网 InfiniBand: Mellanox Technologies MT26428 [ConnectX IB QDR, PCIe 2.0 5GT/s] (rev b0)	同左
操作系统	CentOS release 5.3 (Final)	同左
内核版本	2.6.18-128.el5 #1 SMP Wed Jan 21 10:41:14 EST 2009 x86_64 x86_64 x86_64 GNU/Linux	同左

Fortran	ifort (IFORT) 12.0.4 20110427	N/A
MPI	mvapich-1.2rc1 安装路径 /home/wanghui/mvapich-nocoll/	N/A
NetCDF	netcdf-4.1.1 安装路径 /usr/local	N/A
CESM inputdata	10.10.108.1:/home 11T 4.7T 5.6T 46% /home Inputdata 位于 /home/wanghui/inputdata	

3.2 配置 case

一个 CESM 的 case，需要经过创建（create），配置（configure），构建（build），运行（run）几个环节。每一个环节用到不同的 xml 配置文件。这些 xml 文件包含了大量的配置参数，这些参数的意义请参考 case 目录下的一个文件 README/readme_env。

当一个 xml 配置文件被应用到一个 case 的某个环节之后，该文件中的信息被读取出来并写入了其他的文件中，因此这个 xml 文件可能会被 CESM“锁定”，即不能再被修改。如果要做修改，则必须清除先前的工作。

这些配置文件的情况见下表。

配置文件名	文件主要作用	锁定时刻
env_case.xml	包含 case 的基本信息，例如 case 的名字，case 路径，case 的 grid，compset，machine。这些信息都是只读的。	调用 create_newcase 之前。 通常没有修改的需要。
env_conf.xml	包含 case 的运行类型，开始日期，参考 case；定义各个模块的运行模式和基本配置。	调用 configure –case 之前。 如果后来修改了这个文件，先执行 configure –cleannamelist，然后执行 configure -case
env_mach_pes.xml	包含对 PE Layout 的描述。因为不同的 PE Layout 对一个 CESM case 的运行效率是有不可忽略的影响的，所以这个文件的内容需要经过仔细的调配，使得 case 运行效率最大化。	调用 configure –case 之前。 如果后来修改了这个文件，需要先执行 configure -cleanmach，再 configure -case
env_build.xml	包含与构建相关的内容。例如，构建路径设置，上次构建是否成功等。	执行 build 脚本之前。
env_run.xml	包含运行设置，例如运行时间，运行类型等。	该文件从不被锁定。 任何时候都可以修改这个文件，然后运行 run 脚本。

下面给出一些常见的配置。更详细的信息请参考 CESM 用户指南[25]。

	修改文件	修改变量	取值范围
设置运行时间	env_run.xml	STOP_OPTION，STOP_N	
设置自动重启	env_run.xml	RESUBMIT	>=0
设置重启时间	env_run.xml	REST_OPTION，REST_N	
设置运行方式	env_run.xml	CONTINUE_RUN	TRUE, FALSE
设置运行类型	env_conf.xml	RUN_TYPE	startup，branch，hybrid[26]
设置运行拓扑	env_mach_pes.xml	涉及文件中的多个变量	

3.2.1 设置运行时间

变量：STOP_OPTION，STOP_N

说明：默认的运行时间是模拟 5 天。这个数值仅仅是为了测试而设的；通常的运行时间要比 5 天长得多，例如模拟 20 年，50 年等等。

如果实际运行时间很长，就很容易超出作业管理系统设置的作业时间限制。这种情况一般将运行时间设置为一个合理的时间，然后利用自动重启，运行多个轮次。

3.2.2 设置自动重启

变量：RESUBMIT

说明：如果大于 0，那么每一轮运行结束之后将再次自动提交，重复提交 RESUBMIT 次，也就是总计运行 RESUBMIT+1 次。

3.2.3 设置重启时间

变量：REST_OPTION，REST_N

说明：这里是设置重启文件产生的频度。一般将重启时间设置为与运行时间相同。这样，重启文件将在运行结束的时候产生一次。

[25] 参考 http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc/c796.html
[26] 参考 http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc/x924.html

3.2.4 设置运行方式

变量：CONTINUE_RUN

说明：如果是 restart，那么设置为 TRUE；如果是第一次运行，设置为 FALSE，具体的运行类型还要看 RUN_TYPE 的取值。

case 的重启，就是从包含若干个 restart 文件的“检查点”恢复启动，继续运行，好像从未停止过运行。这种恢复是精确的。例如，运行两个月，与运行一个月停止-再重启运行一个月，两者得到的最终数据完全相同。

3.2.5 设置运行类型

变量：RUN_TYPE

说明：通常不需要修改这个变量的值。一般情况下，RUN_TYPE 的默认设置是“startup”。当 CONTINUE_RUN 为 FALSE 的时候，RUN_TYPE 这个变量才有意义。

startup run 意味着是一次“全新”的启动。各个组件分别初始化，初始化的方式可以各不相同。例如，数据来源有：initial files；restart files；external observed data files；internal initialization（cold start）。耦合器 CPL 不需要输入数据。OCN 模块是延迟启动的（第二天）。

branch run 在启动的时候，各个组件读取重启文件进行初始化。这一组重启文件是从之前的某个运行实例抓下来的。这个历史实例被称为参考 case。branch run 的运行开始日期是从这一组重启文件得到的，所以不能被某个配置文件设置和修改。branch run 的名字可以与参考 case 不同或相同。branch run 的定义需要设置 RUN_REFCASE 和 RUN_REFDATE 变量。另外，branch run 有一个准备数据的过程，需要将重启文件预先拷贝到运行目录。

hybrid run 则是综合了 startup 和 branch 两种方式。例如，有的模块是从 initial files 启动，有的模块则是从重启文件启动。

这里提出一个问题：branch run 和 restart run 是否等价？第一种情况是 CONTINUE_RUN 为 FALSE，同时 RUN_TYPE 设置为“branch”；第二种情况，设置变量 CONTINUE_RUN 为 TRUE。我们的理解是，这两种运行方式是非常类似的，都具备以下特点：从一组一致的重启文件启动；都是精确重启（bit-for-bit），即程序好像没有停止过运行一样。主要不同是，branch run 是从一个参考 case 启动，历史文件是外部产生的；restart run 的重启文件是自己产生的，也就是重启自己，接着自己上次运行的结果运行下去。另外，当 branch run 的第一轮运行结束之后，一般将变量 CONTINUE_RUN 设置为 TRUE，让此后的运行成为重启自身的 restart run。

3.2.6 设置运行拓扑

env_mach_pes.xml 文件包含对 PE Layout 的描述。因为不同的 PE Layout 对一个 CESM case 的运行效率是有不可忽略的影响的，所以这个文件的内容需要经过仔细的调配，使得 case 运行效率最大化。

每一个组件，包括 CPL 的布局都是可配置的。可配置项包括：MPI task 的数目；每个 MPI task 包含的线程数；起始的 MPI task 号。下面示例说明主要的概念[27]。

---

[27] http://www.cesm.ucar.edu/models/cesm1.0/cesm/cesm_doc/x924.html

```
<entry id="NTASKS_ATM" value="16" />
<entry id="NTHRDS_ATM" value="4"  />
<entry id="ROOTPE_ATM" value="0"  />
<entry id="NTASKS_OCN" value="64" />
<entry id="NTHRDS_OCN" value="1"  />
<entry id="ROOTPE_OCN" value="16" />
```

以上示例说明，ATM 模块总计有 16 个 MPI task，每个 task 有 4 线程，总计占用 16x4=64 个 PE，root MPI task 是 0 号 task；OCN 的 task 有 64 个，每个 task 是单线程，root task 是 16 号 task；布局如下

task	0					16			
PE	0	…	3		63	64			127
	ATM					OCN			

下面将 ROOTPE_OCN 改为 64，如下所示。

```
<entry id="NTASKS_ATM" value="16" />
<entry id="NTHRDS_ATM" value="4"  />
<entry id="ROOTPE_ATM" value="0"  />
<entry id="NTASKS_OCN" value="64" />
<entry id="NTHRDS_OCN" value="1"  />
<entry id="ROOTPE_OCN" value="64" />
```

布局如下

task	0					16		64			
PE	0	…	3		63	64		112			175
	ATM					空闲		OCN			

当打开线程支持之后，MPI task 和实际的 PE 不是一一对应的，所以看起来有点复杂。

当运行 CESM 的 all active case 的时候，我们设置 PE 的一般经验是：

1. 将 OCN 与其他模块互斥。
2. CPL，LND，ICE，ATM 都可“序列”化，即 PE 的设置“重叠”起来。
3. 给 GLC 分配很少的几个 PE 就够了。

3.2.7 检查 inputdata

手工检查 inputdata 可以通过脚本 check_input_data。例如下图所示。

```
 [wanghui@gh87 test_b1850cn_f09_g16.ghnode]$ ./check_input_data
-check -inputdata /home/wanghui/inputdata
Input Data List Files Found:
./Buildconf/cam.input_data_list
./Buildconf/pop2.input_data_list
./Buildconf/cice.input_data_list
./Buildconf/clm.input_data_list
./Buildconf/cpl.input_data_list
File status unknown:
b40.1850.track1.1deg.006.cam2.i.0863-01-01-00000.nc
```

如果出现“File status unknown”提示，请检查这几个 input_data_list 文件中的列表，往往是因为文件路径没有写全。例如上例中，将 cam.input_data_list 等文件中的

```
ncdata = b40.1850.track1.1deg.006.cam2.i.0863-01-01-00000.nc
```

修改为全路径

```
ncdata =
$DIN_LOC_ROOT/ccsm4_init/b40.1850.track1.1deg.006/0863-01-01/b40.18
50.track1.1deg.006.cam2.i.0863-01-01-00000.nc
```

- 文件路径要写全。
- 下载得到的 ccsm4 历史文件可能与 cesm 不兼容。参考 2.1.2。

3.2.8 修改 run 脚本

run 脚本是在 configure –case 之后创建的。

有时候需要手工修改这个脚本。例如，在试验环境下，往往没有那么多的节点供测试使用，所以，可用的节点达不到 Machine 文件给出的 np 值，这时就需要手工修改一下 run 脚本中的 mpirun 或类似的命令行的-np 参数。有时候为了调试，想把 ccsm.log 直接输出，这也可以通过直接修改 run 脚本实现。

3.3 测试数据

我们主要完成了以下 compset 的测试，见下图打勾的测试。

模式	f45_g37	f19_f19	f09_g16
X_PRESENT_DAY	✓		
A_PRESENT_DAY	✓		

B_1850	✓		
B_2000	✓		
C_NORMAL_YEAR	✓		
F_2000	✓		
F_AMIP_CAM5		✓	
B1850CN	✓		✓
ERU.B1850CN	✓		✓

表 1：B1850CN.f45_g37.ghnode 实验多次取最佳												
物理节点数		1		2		2		2		3		3	
模式		PEs	根 PE	PEs	根 PE	PEs	根 PE	PEs	根 PE	PEs	根 PE	PEs	根 PE
CPL	cpl	12	0	24	0	12	0	20	0	24	0	24	0
GLC	sglc	8	0	8	0	8	0	8	0	8	8	8	12
LND	clm	12	0	24	0	12	0	20	0	24	0	24	0
ICE	cice	8	0	20	0	8	0	20	0	20	0	20	0
ATM	cam	12	0	24	0	12	0	20	0	24	0	24	0
OCN	pop2	8	4	20	4	8	12	20	0	20	16	8	28
Seconds/day		18.070		10.796		13.553		10.569		9.416		8.433	

表 2：B1850CN.f45_g37.ghnode 实验多次取最佳												
物理节点数		3		3		3		3		3		3	
模式		PEs	根 PE	PEs	根 PE	PEs	根 PE	PEs	根 PE	PEs	根 PE	PEs	根 PE
CPL	cpl	36	0	24	0	16	0	24	0	24	0	24	0
GLC	sglc	8	0	8	24	8	0	4	0	4	0	8	12
LND	clm	36	0	24	0	16	0	6	18	4	20	16	0
ICE	cice	20	0	20	0	16	0	20	4	20	4	8	16
ATM	cam	36	0	24	0	16	0	24	0	24	0	24	0
OCN	pop2	20	16	20	16	20	16	8	28	8	28	8	28
Seconds/day		9.181		10.765		11.145		10.422		11.056		8.938	

表 3：B1850CN.f45_g37.ghnode 实验多次取最佳											
物理节点数		3		3		5		5		5	
模式		PEs	根 PE	PEs	根 PE	PEs	根 PE	PEs	根 PE	PEs	根 PE
CPL	cpl	24	0	24	0	40	0	40	0	40	0
GLC	sglc	8	0	8	0	8	0	8	0	8	0
LND	clm	12	0	24	0	32	0	20	0	40	0
ICE	cice	8	16	20	0	8	32	20	20	40	0
ATM	cam	24	0	24	0	40	0	40	0	40	0
OCN	pop2	8	28	8	28	20	40	20	40	20	40
Seconds/day		8.858		7.756		7.600		5.552		4.861	

表 4：B1850CN.f45_g37.gbnode 实验多次取最佳							
物理节点数		1		1		3	
模式		PEs	根 PE	PEs	根 PE	PEs	根 PE
CPL	cpl	16	0	8	0	28	0
GLC	sglc	8	0	8	0	8	0
LND	clm	16	0	8	0	28	0
ICE	cice	16	0	8	0	20	0
ATM	cam	16	0	8	0	28	0
OCN	pop2	16	0	8	8	20	28
Seconds/day		30.748		40.490		29.647	

表 5：B1850CN.f09_g16.ghnode 实验多次取最佳			
物理节点数		5	
模式		PEs	根 PE
CPL	cpl	40	0
GLC	sglc	8	0
LND	clm	40	0
ICE	cice	40	0
ATM	cam	40	0
OCN	pop2	20	40
Seconds/day		65.610	

给出最后一个实验（表 5）的 timing 文件局部。

```
---------------- CCSM TIMING PROFILE ---------------------

  Case       : test_b1850cn_f09_g16.ghnode_ERU.ghnode
  LID        : 110607-115408
  Machine    : ghnode
  Caseroot   : /home/wanghui/cesm1_0_2/scripts/test_b1850cn_f09_g16.ghnode_ERU.ghnode
  Timeroot   : /home/wanghui/cesm1_0_2/scripts/test_b1850cn_f09_g16.ghnode_ERU.ghnode/Tools
  CCSM User  : wanghui
  CCSM Tag   : cesm1_0_2  (best guess)
  Curr Date  : Tue Jun 7 12:25:12 CST 2011

  grid       : 0.9x1.25_gx1v6
  compset    : B_1850_CN (B1850CN)
  run_type   : hybrid, continue_run = TRUE (inittype = FALSE)
  stop_option : nmonths, stop_n = 1
  run_length  : 28 days (28 for ocean)

  component      comp_pes    root_pe   tasks  x threads (stride)
  ---------       ------    -------   ------   ------   ------
  cpl = cpl       40         0       40     x 1      (1     )
  glc = sglc      8          0       8      x 1      (1     )
  lnd = clm       40         0       40     x 1      (1     )
  ice = cice      40         0       40     x 1      (1     )
  atm = cam       40         0       40     x 1      (1     )
  ocn = pop2      20         40      20     x 1      (1     )

  total pes active          : 60
  pe count for cost estimate : 60

  Overall Metrics:
    Model Cost:           399.13   pe-hrs/simulated_year (scale= 1.00)
    Model Throughput:       3.61   simulated_years/day

    Init Time   :     18.295 seconds
    Run Time    :   1837.073 seconds      65.610 seconds/day
    Final Time  :      0.003 seconds

    Actual Ocn Init Wait Time     :      0.006 seconds
    Estimated Ocn Init Run Time   :      0.000 seconds
    Estimated Run Time Correction :      0.000 seconds
      (This correction has been applied to the ocean and total run times)

Runs Time in total seconds, seconds/model-day, and model-years/wall-day
CPL Run Time represents time in CPL pes alone, not including time associated with data exchange
with other components
    TOT Run Time:    1837.073 seconds      65.610 seconds/mday        3.61 myears/wday
    LND Run Time:     161.509 seconds       5.768 seconds/mday       41.04 myears/wday
    ICE Run Time:     210.820 seconds       7.529 seconds/mday       31.44 myears/wday
    ATM Run Time:    1407.823 seconds      50.279 seconds/mday        4.71 myears/wday
    OCN Run Time:    1378.323 seconds      49.226 seconds/mday        4.81 myears/wday
    GLC Run Time:       0.000 seconds       0.000 seconds/mday        0.00 myears/wday
    CPL Run Time:      34.973 seconds       1.249 seconds/mday      189.52 myears/wday
```

3.4 测试分析

从运行情况来看，CESM 是典型的计算密集型应用。下面我们重点考察 CESM 对资源的需求。

在运行期间，CPU 利用率保持在 100%。这是典型的计算密集型程序。

内存不足将严重影响性能，甚至不能完成测试。但是一般来说，现在的服务器内存跑 CESM 是够用的。在我们所有的实验中，仅出现过一次例外，是在利用 quarter 平台运行 F_AMIP_CAM5，运行了默认的 5days，总计用时 1h25m。测试完成，但在运行中显著的出现了磁盘 swap。quarter 平台仅有两台节点，若要消除内存瓶颈，比较好的方法是增加两个与 quarter01 相同配置的节点。在 gbnode 和 ghnode 平台上，从未出现将内存耗尽的情况。一个 ccsm.exe 进程的内存占用的典型值，VIRT 大约 300~400M，RES 大约 100~250M，SHR 大约 15~20M。如果模拟分辨率进一步提高，那么问题空间又会增长，内存需求必然也会增长。

磁盘 IO 也不是瓶颈。磁盘读写密集出现在开始阶段的读入数据，结束阶段对数据进行

存盘，在中间的计算阶段，CPU 利用率一般都达到百分之百。特别是在较低分辨率下，不会有特别大的数据量，用普通的 nfs 共享目录就可以了。但是，随着数据量增大或运行方式的不同，IO 性能可能会成为不得不考虑的问题。例如当模式网格分辨率很精细的时候，输入的数据量会有显著的增长。在初始化阶段，就要读入大量的数据。例如，B1850CN.f09_g16 做一次重启文件写盘，总计占用大约 4GBytes 的磁盘空间。但是，这些操作所占的比重并不大，而且出现在运行的“一头一尾”，是一种“额外”的消耗。总之，CESM 的磁盘 IO 操作不会很频繁，而是“间歇性”的。当然，如果 CESM 不是独占系统的存储带宽，而是与某个 IO 密集型应用在争夺资源，那么会对 CESM 应用有不利的影响。

对存储空间的要求。首先是输入数据集占用的磁盘空间，大约有几十 GBytes 的体量。第二是当运行时间很长，产生的数据量也大，例如积累下来的重启文件，每隔一个规定的时间就会产生重启文件，积累下来的总数据量比较大。

以上对存储的分析也是建立在目前的分辨率基础上。如果再进一步提高分辨率，那么可能会从量变走向质变。例如 B1850CN，f45_g37 的重启文件只有大约 500MBytes，而 f09_g16 就增长为 4GBytes。如果随着模拟更加精细，存储的文件尺寸变为几十甚至上百 GBytes，优化工作就值得一做了。

CESM 对底层通信的要求。主要是消息延迟敏感。例如，注意观察上一节表 4 所列的在 gbnode 上的实验 B1850CN.f45_g37，会发现它的加速比很不理想。而在 ghnode 上，则获得了预期的计算加速。例如，B1850CN 在 f45_g37 分辨率下，在 ghnode 平台上，当节点数从 1→2→3→5，模拟的速率（秒/天）分别从 18→11→8→5。gbnode 是一个千兆以太网的配置，而 ghnode 则配备了 InfiniBand 作为 MPI 通信的底层硬件。在运行时，两者的显著区别还体现在 CPU 的利用率特征。观察 CPU 占用情况，在 IB 平台上基本达到 99.x%us，0.x%sy。而在 gbnode 平台上，长期维持的典型值是 15%us，85%sy。

我们没有进一步检验实验得到的的数据是否全部正确。目前，我们判断运行成功的依据有两个：

1. 无明显报错。屏幕打印“SUCCESSFUL TERMINATION OF CPL7-CCSM”提示。
2. 有结果文件。例如，timing/目录下能够找到相应的 timing 文件[28]。

在本节的最后，给出我们的几个结论：

- CESM 是计算密集型应用。
- CESM 应用对消息延迟敏感。
- 模式分辨率是 CESM 应用的“需求放大器”。随着分辨率的提高，目前看来还不成为问题的地方，将来可能都是不得不考虑的问题。

4 移植总结

CESM 移植出现的问题主要有三类，首先是软件环境，包括软件安装，环境变量、编译选项的设置；第二是输入数据集，要保证输入数据的完整，且被程序正确的读取；第三是 fortran 源码的问题，例如指针未赋初值等。这些问题在本文的第二章做了较为详细的讲述。

在移植成功之后，为了研究运行的效率问题，在配置并行计算机软硬件环境的基础上，

[28] 参考 http://www.cesm.ucar.edu/models/cesm1.0/timing/

需要认真阅读 CESM 用户手册，研究 case 的运行配置参数（例如给各个分量模式分配节点），将任务合理分解，然后通过测试确定最佳的配置；进一步的优化，还需要理解并行算法的实现机制，能够改进现有的并行算法（例如改进格点分配方案）和计算模式（例如引入 GPU 加速）。地学专家则可以从改进数理模型入手（例如简化现有模型），优化现有的模式算法。

总之，计算机专业和地学专业既有分工又要密切合作。设想如下图所示。