OpenACC 应用编程接口

版本 1.0，2011 年 11 月发布

小小河 译

法律条款

1. 对本文档的任何使用都被视为完全理解并接受本文档所列举的所有法律条款。
2. 本文档的所有权利归作者所有，作者保留所有权利。
3. 未经作者书面同意，禁止任何形式的商业使用。商业使用形式包括但不限于出版、复制、传播、展示、引用、编辑。
4. 本文档允许以学术研究、技术交流为目的使用。复制、传播过程中不得对本文档作任何增减编辑，引用时需注明出处。
5. 实施任何侵权形为的法人或自然人都必须向作者支付赔偿金，赔偿金计算方法为：

   赔偿金 = 涉案人次×涉案时长（天）×涉案文档份数×受众人次×100 元人民币，

   涉案人次、涉案时长、涉案文档份数、受众人次小于 1 时，按 1 计算。
6. 举报侵权行为、提供有价值证据的自然人或法人均有权向作者索取案件实际赔偿金的 50%。
7. 涉及本文档的法律纠纷由作者所在地法院裁决。
8. 本文档所列举法律条款的最终解释权归作者所有。

译者序.....4
第 1 章　引言.....4
1.1 范围.....5
1.2 执行模型.....5
1.3 存储模型.....6
1.4 本文档组织结构.....6
1.5 参考文献.....6
第 2 章 导语.....8
2.1 导语格式.....8
2.2 条件编译.....9
2.3 内部控制变量.....9
2.4 加速器计算构件.....9
2.5 data 构件.....14
2.6 host_data 构件.....15
2.7 数据子语.....15
2.8 loop 构件.....18
2.9 Cache 导语.....21
2.10 组合导语.....21
2.11 declare 导语.....22
2.12 可执行导语.....23
第 3 章 运行时库例程.....26
3.1 运行时库的定义.....26
3.2 运行时库例程.....27
第 4 章 环境变量.....34
4.1 ACC_DEVICE_TYPE.....34
4.2 ACC_DEVICE_NUM.....34
第 5 章 词汇表.....35

译者序

2007 年以来，以 nVidia GPU 为代表的加速器并行计算风起云涌，带有加速器的超级计算机在 TOP500 中的份额逐年增加，支持加速器的主流应用软件也呈爆炸式增长，研究加速器计算的技术人员数以百万计，世界范围内的大学、研究机构竞相开设相关课程。

目前有三家厂商提供加速器产品：nVidia GPU、AMD GPU、Intel 至强 Phi 协处理器。三种加速器使用的编程语言分别为 CUDA C/CUDA Fortran、OpenCL 和 MIC 导语。加速器计算有四个技术困难：一是 CUDA/OpenCL 等低级语言编程难度大，且需要深入了解加速器的硬件结构。而大部分的用户不是专业编程人员，学习一门新的编程技术将耗费大量时间、人力、财力。二是加速器的计算模型与 CPU 差别很大，移植旧程序需要几乎完全重写。大量的旧程序在性能优化上已经千锤百炼，稳定性上也久经考验，完全重写是不可完成的任务。三是低级编程语言开发的程序与硬件结构密切相关，硬件升级时必须升级软件，否则将损失性能。而硬件每隔两三年就升级一次，频繁的软件升级将给用户带来巨大负担。四是投资方向难以选择。三种加速器均有自己独特的编程语言，且互不兼容。用户在投资建设硬件平台、选择软件开发语言时就会陷入困境，不知三种设备中哪个会在将来胜出。如果所选加速器将来落败，将会带来巨大损失；而犹豫不决又将错过技术变革的历史机遇。

OpenACC 可以克服这四个困难。OpenACC 应用编程接口的机制是，编译器根据作者的意图自动产生低级语言代码。程序员只需要在原程序中添加少量导语，无须学习新的编程语言和加速器硬件知识，可以迅速掌握。只在原始程序上添加少量导语，不破坏原代码，开发速度快，既可并行执行又可恢复串行执行。在硬件更新时，重新编译一次代码即可，不必手工修改代码。OpenACC 标准制定时就考虑了目前及将来的多种加速器产品，同一份代码可以在多种加速器设备上编译、运行，零代价切换硬件平台。

本文档将 directive 翻译为导语，将 clause 翻译为子语，将 construct 翻译为构件。directive 的作用是给编译器一些指导，指出哪些代码需要并行化、需要怎么并行化，编译器根据程序员的指导信息生成最佳的并行化代码。有人将 directive 翻译为编译制导语句、编译指导语句、指令语、指令等，意思都对，但编译制导语句、编译指导语句太长，使用不便，指令语、指令中的指令一词太普通，易混，且没有指导的含义，不太恰当。导语一词长度、意思都比较合适。clause 是导语的修饰部分，更详细地表明导语的意图。有人将 clause 翻译为子句，而子句一词含有小句子的意思，实际上 OpenACC 中的 clause 都只有一个词，很短，不能称为一个句子。子语一词既说明了它与导语的关系，又有一个相同的语字，读起来顺口。construct 翻译为构件，取自建筑名词，实际功能也很相像。

本人水平有限，错误疏漏在所难免，欢迎批评指正，感激不尽。原始英文版请到 http://www.openacc.org/下载。本文档的最新版请到 www.bytes.me/openacc/下载。

OpenACC 技术交流 QQ 群：284876008，欢迎加入，共同进步。

2013 年 1 月 6 日

第1章 引言

本文档讲述的编译器导语、库例程和环境变量共同定义了 OpenACC 应用编程接口(OpenACC API，此后简称 OpenACC 接口)。 C/C++和 Fortran 程序中，OpenACC 接口将代码从主机 CPU 卸载到一个加速器设备。对各种操作系统、各种类型的主机 CPU 和各种类型的加速器，这里介绍的编程模型都是可移植的。导语扩展了 ISO/ANSI 标准 C/C++和 Fortran 语言，它允许程序员将标准 C/C++或 Fortran 程序移迁移到加速器上。

利用本文定义的导语和编程模型，程序员能够创建的应用有能力使用加速器，而不必管理数据和主机与加速器间的程序转移，也不必费心加速器的启动与关闭. 当然了，支持 OpenACC 接口的编译器和运行环境会管理编程模型中隐含的这些细节. 编程模型允许程序员告诉编译器更多的有用信息，包括指定在加器本地存储数据、引导编译器将循环映射到加速器上，以及与性能相关的更多细节。

1.1 范围

这份 OpenACC 接口文档只涉及用户的加速器编程，由用户指定将哪些主机程序区域卸载到加速器设备上的。程序的剩余部分将在主机上执行。本文档只介绍对卸载到加速器上的循环和代码区域的详尽要求；至于主机编程的特性和限制，本文档不作介绍。

本文档不讲述使用编译器或其它工具自动侦测并将代码区域卸载到加速器的内容。也不讲述如何将循环或其它代码区域分配到与单个主机相连的多个加速器上。将来的编译器可能支持自动卸载、多个相同型号的加速器、多个不同型号的加速器，但这些特性本文档概不涉及。

1.2 执行模型

OpenACC 编译器的执行模型是主机指导加速器设备（如 GPU）的运行。主机执行用户应用的大部分代码，并将计算密集型区域卸载到加速器上执行。设备上执行并行区域和内核区域，并行区域通常包含一个或多个工作分担(work-sharing)循环，内核区域通常包含一个或多个被作为内核执行的循环。即使在加速器负责的区域，主机也必须精心安排程序的运行：在加速器设备上分配存储空间、初始化数据传输、将代码发送到加速器上、给并行区域传递参数、为设备端代码排队、等待完成、将结果传回主机、释放存储空间。大多数据时候，主机可以将设备上的所有操作排成一队，一个接一个的顺序执行。

目前大多数加速器支持二到三层并行。大多数加速器支持粗粒度并：执行单元完全并行地执行。加速器可能有限支持粗粒度并行操作间的同步。许多加速器也支持细粒度并行：在单个执行单元上执行多个线程，这些线程可以快速地切换，从而可以忍受长时间的存储操作延时。最后，大多数加速器也支持每个执行单元内的单指令多数据(SIMD)操作或向量操作。既然设备端上的执行模型包含多个并行层次，程序员就需要理解它们之间的区别。例如，一个完全并行的循环和一个可向量化但要求语句间同步的循环之间的区别。一个完全并行的循环可以用粗粒度并行执行。有依赖关系的循环要么适当分割以允许粗粒度并行，要么在单个执行单元上以细粒度并行、向量并行或串行执行。

1.3 存储模型

一个仅在主机上运行的程序与一个在主机+加速器上运行的程序，它们最大的区别在于加速器上的内存可能与主机内存完全分离。例如目前大多数 GPU 就是这样。这种情况下，设备内存可能无法被主机直接读写，因为它没有被映射到主机的虚拟存储空间。主机内存与设备内存间的所有数据移动必须由主机完成，主机调用运行时库在相互分离的内存间显式地移动数据。数据移动通常采用直接内存访问(Direct Memory Access, DMA)技术。类似地，不能假定加速器能读写主机内存，尽管有些加速器设备支持这样的操作。

在 CUDA C 和 OpenCL 等低层级加速器编程语言中，主机和加速器存储器分离的概念非常明确，内存间移动数据的语句甚至占据大部分用户代码。在 OpenACC 模型中，内存间的数据移动是隐式的，编译器根据程序员的导语管理这些数据移动。然而程序员必须了解背后这些相互分离的内存，有包含但不限于以下理由：

- 有效加速一个区域的代码需要较高的计算密度，而计算密度的高低取决于主机内存与设备内存的存储带宽；
- 设备内存空间有限，因此操作大量数据的代码不能卸载到设备上。

在加速器一端，一些加速器(例如目前的 GPU)使用一个较弱的存储模型。特别地，它们不支持不同执行单元上操作的内存一致性；甚至，在同一个执行单元上，只有在存储操作语句之间显式地同步才能保证内存一致性。否则，如果一个操作更新一个内存地址而另一个操作读取同一个地址，或者两个操作向同一个位置存入数据，那硬件可能不保证每次运行都能得到相同的结果。尽管编译器可以检测到一些这样的潜在错误，但仍有可能编写出一个产生不一致数值结果的加速器并行区域或内核区域。

目前，一些加速器有一块软件管理的缓存，一些加速器有多块硬件管理的缓存。大多数加速器具有仅在特定情形下使用的硬件缓存，并且仅限于存放只读数据。在 CUDA C 和 OpenCL 等低层级语言的编程模型中，这些缓存交由程序员管理。在 OpenACC 模型中，编译器会根据程序员的导语管理这些缓存。

1.4 本文档组织结构

本文档后续部分组织如下：

第二章，导语，讲述 C/C++和 Fortran 导语，它们用于构建加速器区域，及提供更多信息以帮助编译器调度循环、对数据分类。

第三章，运行库例程，定义供用户调用的函数、查询加速器设备参数的库例程、控制加速器程序运行时形为的库例程。

第四章，环境变量，定义供用户设置的环境变量，它们控制加速器程序执行时的行为。

第五章，词汇表，定义本文档用到的术语。

1.5 参考文献

American National Standard Programming Language C, ANSI X3.159-1989 (ANSI C).

ISO/IEC 9899:1999, *Information Technology – Programming Languages* – C (C99).

ISO/IEC 14882:1998, *Information Technology – Programming Languages* – C++.

ISO/IEC 1539-1:2004, *Information Technology – Programming Languages – Fortran Part 1:*

Base Language, (Fortran 2003).

OpenMP Application Program Interface, version 3.1, July 2011

PGI Accelerator Programming Model for Fortran & C, version 1.3, November 2011

NVIDIA CUDA™ C Programming Guide, version 4.0, May 2011.

The OpenCL Specification, version 1.1, Khronos OpenCL Working Group, June 2011.

第2章 导语

本章讲述 OpenACC 导语的语法和形为。在 C 和 C++中，用语言本身提供的#pragam 机制来引入 OpenACC 导语。在 Fortran 中 ,用带有独特前导符的注释来引入 OpenACC 导语。如果不支持或者关闭了 OpenACC 功能，编译器将会忽略 OpenACC 导语。

限制

- Fortran 中，OpenACC 导语不能出现在 PURE 和 ELEMENTAL 过程中。

2.1 导语格式

在 C/C++中，使用#pragma 机制指定 OpenACC 导语，语法是：

```
#pragma acc 导语名字 [子语 [[,]子语]...] 换行
```

每个导语都以#pragma acc 开始。导语的其它部分都遵守 C/C++中 pragma 的使用规范。#的前后都可以使用空白字符；导语中使用空白字符来分隔各字段。#pragma 后面的预处理标记使用宏替换。导语区分大小写。一个 OpenACC 导语作用于紧接着的语句、结构块和循环。

Fortran 自由格式源文件中，用下列格式指定 OpenACC 导语：

```
!$acc 导语名字 [子语[[,]子语]...]
```

第一个注释字符(!)可以放在任意列，但它前面只能是空白字符(空格和跳格). 前导符 !$acc 必须以一个整体出现，中间不能有空白字符。每行长度、空白字符、续行符规则同样适用于导语行。导语起始行的前导符后必须有一个空白字符。待续行[1]中导语部分的最后一个非空白字符必须是连字符(&)，连字符后面仍然可以写注释；接续行中导语必需以前导符开始(前面允许有空白字符)，前导符后面的第一个非空白字符可以是续行符。导语行上也可以放注释，注释以感叹号开始，直至行尾。如果前导符后面的第一个非空白字符是一个感叹号，那么该行被忽略。

在固定格式 Fortran 源代码文件中，OpenACC 导语可以采取下列形式中的一个：

```
!$acc 导语名字 [ 子语 [[,] 子语]...]
c$acc 导语名字 [ 子语 [[,] 子语]...]
*$acc 导语名字 [ 子语 [[,] 子语]...]
```

前导符(!acc, cacc, 或*$acc)必须写在 1-5 列。固定格式的每行长度、空白字符、续行、列的规则同样适用于导语行。导语起始行第 6 列必须是空格或 0，接续行导语在第 6 列不能是空格或零. 导语行也可以添加注释，注释可以从第 7 列（包含）之后的任意列以感叹号开始，至行尾结束。

在 Fortran 中，导语不区分大小写。分写在多行的单个程序语句中间也不能混入导语，同样地，分写在多行的单个导语中间不能混入的程序语句。本文档中所有 Fortran OpenACC

[1] 如果单个程序语句被分写在多行之上，最后一行之外的每一行都称为待续行，第一行之外的每一行都称为接续行。--译者注。

导语例子都采用自由格式。

每个导语中只能有一个*导语名字*. 多个子语出现的顺序无关紧要，除非特别限定，同一子语可以重复多出现多次。有些子语有一个名为*列表*的参数；*列表*是用逗号分隔的一串变量名、数组名，有时还可能是指定下标范围的子数组。

2.2 条件编译

预定义的宏_OPENACC 的值为 yyyymm，其中 yyyy 是编译器所支持 OpenACC 版本的发布年份，mm 是月份。当且仅当 OpenACC 导语功能打开时，编译器必须定义这个宏。这里讲述的版本是 201111.

2.3 内部控制变量

一个符合 OpenACC 标准的编译器表现得好像有一些内部控制变量(Internal control variables, ICVs)控制着程序的行为。编译器通过环境变量或通过调用 OpenACC 例程将这些内部控制变量初始化。通过调用 OpenACC 例程，程序可以获得这些变量的值。

这些外部控制变量是：

- acc-device-type-var 控制使用哪种类型的加速器。
- acc-device-num-var 控制使用所选类型中的哪个加速器。

2.3.1 修改和取出内部控制变量值

下表列出了修改内部控制变量的环境变量和例程、取出内部控制变量值的例程：

内部控制变量	修改值的方法	取出值的方法
acc-device-type-var	ACC_DEVICE_TYPE acc_set_device_type	acc_get_device_type
acc-device-num-var	ACC_DEVICE_NUM acc_set_device_num	acc_get_device_num

这些变量的初值由编译器定义。赋初值之后，执行任何 OpenACC 构件、例程之前，用户设定的所有环境变量值都已经被读出，并依此修改了内部控制变量的值。此后，无论是程序还是用户对环境变量的修改都不会改变内部控制变量的值。OpenACC 构件上的子语也无法修改内部控制变量的值。

2.4 加速器计算构件

2.4.1parallel 构件

概要

这个基本构件开启加速器设备上的并行执行。

语法

在 C 和 C++中，OpenACC parallel 导语的语法为

```
#pragma acc parallel 子语, [[,]子语]...] 换行
```

结构块

在 Fortran 中的语法是

```
!$acc parallel 子语, [[,] 子语 ]...]
   结构块
!$acc end parallel
```

这里的子语是下列中的一个:

```
if(条件)
aysnc [(标量整数表达式)]
num_gangs(标量整数表达式)
num_workers(标量整数表达式)
vector_length(标量整数表达式)
reduction(操作符:列表)
copy(列表)
copyin(列表)
copyout(列表)
create(列表)
present(列表)
present_or_copy(列表)
present_or_copyin(列表)
present_or_copyout(列表)
present_or_create(列表)
deviceptr(列表)
private(列表)
firstprivate(列表)
```

描述

当遇到一个加速器 parallel 构件，程序就创建多个 gang 来执行这个加速器并行区域，而每个 gang 又包含多个 worker。这些 gang 一旦创建，gang 的数量和每个 gang 包含的 worker 数量在整个并行区域内都保持不变。接着，每个 gang 中的所有 worker 都开始执行构件中结构块的代码。

如果没有使用 async 子句，加速器并行区域结束处将会有一个隐式屏障，此时主机处于等待状态，直至所有的 gang 都执行完毕。

如果一个聚合数据类型的变量(或数组)在 parallel 构件中被引用，并且没有出现在该 parallel 构件的数据子语中，也没有包含在任何 data 构件中，那么它等同于出现在 parallel 构件的 present_or_copy 子语中[1]。如果一个标量变量在 parallel 构件中被引用，并且没有出现在该 parallel 构件的数据子语中，也没有包含在任何 data 构件中，那么它等同于出现在 parallel 构件的 private 子语(如果不带值进也不带值出)中或 copy 子语中。

[1]变量的缺省子语是 present_or_copy。—译者注。

限制

- OpenACC `parallel` 区域内不能包含其它 `parallel` 区域或 `kernels` 区域。
- 一个程序不能通分枝转入或跳出 OpenACC `parallel` 构件。
- 一个程序决不能依赖于子语的求值顺序或求值的副作用。
- 至多出现一个 `if` 子语。Fortran 中，`条件`的运算结果必须是一个标量逻辑值；C 和 C++中，`条件`的运算结果必须是一个标量整数值。

子语 copy, copyin, copyout, create, present, present_or_copy, present_or_copyin, present_or_copyout, present_or_create, deviceptr, firstprivate, 和 private 将在 2.7 节详细讲述。

2.4.2 kernels 构件

概要

一个程序的 kernels 构件区域将被编译成一系列在加速器设备上执行的 kernel.

语法

C 和 C++中，OpenACC kernels 导语的语法是：

```
#pragma acc kernels [子语 [[,] 子语]…] 换行
    结构块
```

在 Fortran 中的语法是

```
!$acc kernels [子语 [[,] 子语]…]
    结构块
!$acc end kernels
```

其中，`子语`是下列中的一个：

```
if(条件)
async [(标量整数表达式)]
copy(列表)
copyin(列表)
copyout(列表)
create(列表)
present(列表)
present_or_copy(列表)
present_or_copyin(列表)
present_or_copyout(列表)
present_or_create(列表)
deviceptr(列表)
```

描述

编译器将 `kernels` 区域内的代码分割为一系列的加速器 `kernel`．通常，每个循环成为一个单独的 `kernel`．当程序遇到一个 `kernels` 构件时，它在设备上按顺序启动这一系列 `kernel`．对不同的 `kernel`，`gang` 的数量、每个 `gang` 包含多少个 `worker`、`vector` 的长度以及三者的组织方式都可能不相同。

如果没有使用 `async` 子语，`kernels` 区域结束时将有一个隐式屏障，主机端程序会一

直处于等待状态，直至所有的 kernel 都执行完毕。

如果一个聚合数据类型的变量(或数组)在 kernels 构件中被引用，并且没有出现在该 kernels 构件的数据子语中，也没有包含在任何 data 构件中，那么它等同于出现在 parallel 构件的 present_or_copy 子语中。如果一个标量变量在 kernels 构件中被引用，并且没有出现在该 kernels 构件的数据子语中，也没有包含在任何 data 构件中，那么它等同于出现在 kernels 构件的 private 子语(如果不带值进也不带值出)中或 copy 子语中。

限制

- OpenACC kernels 区域内不能包含其它 parallel 区域或 kernels 区域。
- 一个程序不能通分枝转入或跳出 OpenACC kernels 构件。
- 一个程序决不能依赖于子语的求值顺序或求值的副作用。
- 至多出现一个 if 子语。Fortran 中，条件的运算结果必须是一个标量逻辑值；C/C++中，条件的运算结果必须是一个标量整数值。

数据子语 copy、copyin、copyout、create、present、present_or_copy、 present_or_copyin、present_or_copyout、present_or_create、 deviceptr 将在 2.7 节详细讲述。

2.4.3 if 子语

if 子语是 parallel 构件和 kernels 构件的可选项；没有 if 子语的时候，编译器为本区域生成加速器上执行的代码。

有 if 子语的时候，编译器为本构件生成两份代码，一份可以在加速器上执行，另一份可以在主机执行。对 C 和 C++语言，if 子语中的条件值为零时，对 Fortran 语言，if 子语中的条件值为.false.时，那份主机端代码将被执行。if 子语中条件的值在 C 和 C++语言中为非零时，或在 Fortran 语言中为.true.时，那份加速器端代码将被执行。

2.4.4 async 子语

async 子语是 parallel 构件和 kernels 构件的可选项；没有 async 子句的时候，主机进程在执行构件后面的任何代码之前，会一直等待，直到 parallel 区域或 kernels 区域运行结束。有 async 子语的时候，在主机进程继续执行区域后面代码的同时，parallel 区域或 kernels 区域会在加速器设备上异步地执行。

如果有 async 子语，async 的参数必须是一个整数表达式(C 和 C++中是 int，Fortran 中是 integer)。这个整数表达式也可以用到 wait 导语和多个运行时例程中，从而使主机进程能检测本区域是否完成，或使主机进程等待本区域完成。async 也可没有参数，这种情况下，编译器将使用一个与程序中所有 async 的显式参数都不相同的值。

具有相同参数值的两个异步活动在设备上执行的顺序，与主机进程遇到它们的顺序相同。两个具有不同参数值的异步活动在设备上执行的相对顺序是任意的。如果两个(含)以上主机线程执行或共享同一个加速器设备，具有相同参数值的两个异步活动将在设备上一个接一个的执行，执行的相对顺序未定义。

2.4.5 num_gangs 子语

parallel 构件允许使用 num_gangs 子语。整数表达式的值规定并行执行该区域的 gang 的数量。如果没有本子语，编译器将使用自定义的默认值。

2.4.6 num_workers 子语

parallel 构件允许使用 num_workers 子语。整数表达式的值规定了执行该区域的每个 gang 中所包含 worker 的数量。如果没有本子语，编译器将使用自定义的默认值；默认值可能是 1.

2.4.7 vector_length 子语

parallel 构件允许使用 vector_length 子语。整数表达式的值规定了所有 gang 中每个 worker 的 vector 长度或 SIMD 操作长度。如果没有本子语，编译器将使用自定义的默认值。向量长度将被用在 loop 导语的 vector 子语上，编译器自动向量化循环时也使用这个长度。编译器可能会限制向量长度表达式的取值范围。

2.4.8 private 子语

parallel 构件允许使用 private 子语；对列表中的每一项，每个并行 gang 都会创建一个副本。

2.4.9 firstprivate 子语

parallel 构件允许使用 firstprivate 子语；对列表中的每一项，每个并行 gang 都会创建一个副本。当遇到 parallel 构件时，程序使用主机中的对应项目来初始化 parallel 构件里的副本。

2.4.10 reduction 子语

reduction 子语可以用到 parallel 构件上。它指定一个归约操作符和一个或多个标量变量。对每个变量，每个并行运行的 gang 都会创建一个副本，并根据指定的运算符初始化这个副本。在并行区域结束时，归约运算符使用每一个 gang 里的值、变量的原始值计算出归约结果，并将该结果存入原始变量。归约结果在并行区域之后可用。

下表列出了可用的操作符及相应的初始值，初始值跟数据类型有关。对 max 和 min 归约，初始化值分别为变量所属数据类型能表示的最小值、最大值。C 与 C++(int, float, double, complex)和 Fortran(integer, real, double precision, complex)的数值数据类型都可以使用。

C 和 C++		Fortran	
操作符	初始值	操作符	初始值
+	0	+	0
*	1	*	1
max	最小值	max	最小值
min	最大值	min	最大值

&	~0	iand	所有位为 1
|	0	ior	0
^	0	ieor	0
&&	1	.and.	.true.
||	0	.or.	.false.
		.eqv.	.true.
		.neqv.	.false.

2.5 data 构件

概要

在本区域生存期内，data 构件指明的标量、数组和子数组都会在设备内存上开辟空间。进入本区域时，数据被从主机复制到设备内存；离开本区域时，数据被从设备复制到主机内存。

语法

C 和 C++中，OpenACC data 导语的语法是

```
#pragma acc data [子语 [[,] 子语]…] 换行
      结构块
```

在 Fortran 中的语法是

```
!$acc data [子语 [[,] 子语]…]
      结构块
!$acc end data
```

这里的子语为下列中的一个：

```
if(条件)
copy(列表)
copyin(列表)
copyout(列表)
create(列表)
present(列表)
present_or_copy(列表)
present_or_copyin(列表)
present_or_copyout(列表)
present_or_create(列表)
deviceptr(列表)
```

描述

数据将设备内存上开辟空间。根据需要，数据从主机内存复制到设备，或者复制回来。数据子语将在 2.7 节讲述。

2.5.1 if 子语

if 子语是可选项；没有 if 子语的时候，编译器生成的代码将在加速器设备上开辟存储

空间，并在设备与主机间移动数据。

有 if 子语的时候，程序在某些条件下才在设备上开辟内存空间，并将数据移动到设备上或将数据从设备上取回。当 if 子语中条件的值为零（对 C 和 C++）或 .false.（对 Fortran）时，不会开辟设备内存空间，也不会移动数据。当条件的值为非零（对 C 和 C++）或 .true.（对 Fortran）时，程序将按照指定的方式为数据开辟空间，并移动数据。

2.6 host_data 构件

概要

host_data 构件使设备上数据的地址在主机上可用。

语法

C 和 C++中，OpenACC host_data 导语的语法为

```
#pragma acc host_data [子语 [[,] 子语]…] 换行
    结构块
```

在 Fortran 中的语法为

```
!$acc host_data [子语 [[,] 子语]…]
     结构块
!$acc end host_data
```

这里只有一个子语只可用：

```
use_device(列表)
```

描述

本构件用来使设备上数据的地址可以主机代码中使用。

2.6.1 use_device 子语

use_device 告诉编译器，对列表中的任何变量和数组，在本构件代码中使用它们的设备端内存地址。特别地，本子语可以用来传递变量或数组的设备内存地址给用低层 API 优化过的例程。在包含本构件的数据区域中，列表中的变量或数组必须已经存在于加速器内存之中。

2.7 数据子语

这些数据子语可以出现在 parallel 构件、kernels 构件和 data 构件上。每个子语的参数列表是一串用逗号分隔的变量名字、数组名字或给定下标范围的子数组。在所有情形下，编译器都会在设备内存上为这些变量或数组创建、管理一个副本，即为变量和数组创建一个可见设备副本。

这样做是为了支持那些在物理内存和逻辑内存上都与主机相分离的加速器。然而，如果加速器可以直接访问主机内存，编译器可能通过简单地使用主机内存来避免开辟内存空间和数据移动。所以，如果一个程序在主机端使用并为数据赋值，在设备上也使用相同的数据并为之赋值，且没有使用 update 导语来管理两个副本的一致性，那么使用不同的编译器、

不同的加速器都可能会得到不同的结果。

C 和 C++中，子数组就是数组名后跟一个方括号，方括号内的起始下标和长度用来指定数组范围，例如

```
arr[2:n]
```

下边界的缺省值为零。如果没有指定长度且数组的尺寸已知，那么默认用数组的声明尺寸与下边界的差值来代替；否则必须指定长度。子数组 `arr[2:n]`表示元素 `a[2], a[3],…, a[2+n-1].`

Fortran 中，子数组就是一个数组名后跟一个指定范围的圆括号，圆括号内用逗号分隔上、下边界，例如

```
arr(1:high,low:100)
```

没有指定上边界或下边界的时候，如果知道数组在声明或开辟空间时的边界，就使用这些边界。

限制

- Fortran 中，*不定尺寸形参数组*[1]的最后一个维度必须指定上边界。
- C 和 C++中，必须显式指定动态分配数组的长度。
- 如果一个数据子语中指定了一个子数组，编译器可能选择仅为该子数组分配加速器上的内存。
- 为对齐内存和提高程序性能，编译器可能会填充加速器上数组的维度。
- Fortran 中，可以在子语中指定指针，但设备内存中不再保留指针的指向关系。
- 数据子语中的任何数组、子数组、Fortran 数组指针，都必须是一块连续的内存。
- C 和 C++中，如果指定了一个变量、结构数组、或者类，那么该结构或类的成员都会被酌情分配空间、复制数据。如果结构或类的一个成员是指针类型，那么不会隐式复制该指针指向的数据。
- C 和 C++中，可以指定结构或类的一个成员，也可能是成员的一个子数组。然而，该结构本身必须具有 `automatic`、`static` 或 `global` 属性，即不需要通过指针就能访问。不能指定结构子数组的成员和类子数组的成员。
- Fortran 中，如果指定了一个派生类型的变量或数组，那么会酌情为该派生类型的所有成员分配空间和复制数据。对任何有 `allocatable` 或 `pointer` 属性的成员，通过该成员访问的数据都不会被复制。
- Fortran 中，可以指定派生类型变量的成员，或者成员的一个子数组。然而，派生类型变量不能有 `allocatable` 或 `pointer` 属性。不能指定派生类型子数组的成员。

2.7.1 deviceptr 子语

`deviceptr` 子语用来声明*列表*中的指针是设备指针，因此不必再为该指针指向的数据

[1] assumed-size dummy array，这种数组会显式地声明除了最后一个维度以外的所有维度的长度，最后一个维度的长度用星号来代替。它是 Fortran 语言早期版本的一种延续，新程序不应该再使用它。--译者注。

分配空间，也不必在主机与设备间移动数据。

C 和 C++中，*列表*中的变量必须是指针。

Fortran 中，*列表*中的变量必须是虚设参数（数组或标量），并且不能具有 Fortran `pointer`、`allocatable` 或 `value` 属性。

2.7.2 copy 子语

`copy` 子语用来声明*列表*中的变量、数组或子数组存在于主机内存中，需要将它们复制到设备内存，在加速上为它们赋值后再复制回主机。如果声明的是一个子数组，那么只需复制该数组的子数组。进入本区域之前，将数据复制到设备内存，本区域结束时，再将数据复制回主机内存。

2.7.3 copyin 子语

`copyin` 子语用来声明*列表*中的变量、数组或子数组位于主机内存中的值需要复制到设备内存。如果*列表*中包含子数组，那么该数组中只有指定的子数组部分需要复制。一个变量、数组或子数组出现在 `copyin` 中意味着，即使它们的值在加速器上被改变了，也不需将它们从设备内存复制回主机内存。将数据复制到设备内存的操作发生在进入构件区域时。

2.7.4 copyout 子语

`copyout` 子语用来声明，*列表*中的变量、数组或子数组将会在设备内存中被赋值或在设备内存中保存数值，需要在加速器区域结束时将它们复制回主机内存。如果声明了一个子数组，那么只复制该数组的子数组。一个变量、数组或子数组出现在 `copyout` 中意味着，即使在加速器上会用到它们的值，也不需要将它们从主机内存复制到设备内存。退出本区域时，数据被复制回主机内存。

2.7.5 create 子语

`create` 子语用来声明*列表*中的变量、数组或子数组需要在设备内存上开辟空间（创建），但加速器不需它们在主机内存上的值，主机也不需要它们在加速器上计算得到的值或被赋的值。这个子语中，没有数据需要在主机内存与设备内存之间复制。

2.7.6 present 子语

`present` 子语用来告诉编译器*列表*中的变量或数组已经存在于加速器内存中，因为已经有数据区域包含着本区域，而数据区域可能来自于本构件所在例程的主调例程。编译器将会发现并使用这些已经存在的加速器数据。如果没有数据区域将这些变量、数组中的任何一个放置至加速器上，那么程序将报错中止。

如果包含本构件的数据区域指定的是一个子数组，那么 `present` 子语必须指定相同的子数组，或指定该数据区域子数组的一个适当子集构成的子数组[1]。如果 `present` 子语中的子数组的某些元素不包含在数据区域的子数组中，那么它是一个运行时错误(`runtime`

[1] 就是子数组的子数组。--译者注。

error)。

2.7.7 present_or_copy 子语

present_or_copy 子语用来告诉编译器去检测*列表*中的每一个变量或数组是否已经存在于加速器内存中。如果已经存在，就使用加速器数据。如果不存在，就在设备内存上开辟数据空间，并在进入区域时将数据从主机复制到设备，离开区域时将数据复制回主机，就像 copy 子语一样。这个子语可以缩写为 pcopy。present 子语中对子数组的限制同样适用于本子语。

2.7.8 present_or_copyin 子语

present_or_copyin 子语用来告诉编译器去检测*列表*中的每一个变量或数组是否已经存在于加速器内存中。如果已经存在，就使用加速器数据。如果不存在，就在设备内存上开辟数据空间，并在进入区域时将数据从主机复制到设备，就像 copyin 子语一样。这个子语可以缩写为 pcopyin。present 子语中对子数组的限制同样适用于本子语。

2.7.9 present_or_copyout 子语

present_or_copyout 子语用来告诉编译器去检测*列表*中的每一个变量或数组是否已经存在于加速器内存中。如果已经存在，就使用加速器数据。如果不存在，就在设备内存上开辟数据空间，并在离开区域时将数据复制回主机，就像 copyout 子语一样。这个子语可以缩写为 pcopyout。present 子语中对子数组的限制同样适用于本子语。

2.7.10 present_or_create 子语

present_or_create 子语用来告诉编译器去检测*列表*中的每一个变量或数组是否已经存在于加速器内存中。如果已经存在，就使用加速器数据。如果不存在，就在设备内存上开辟数据空间，就像 create 子语一样。这个子语可以缩写为 pcreate。present 子语中对子数组的限制同样适用于本子语。

2.8 loop 构件

概要

OpenACC loop 导语作用于紧跟该导语的一个循环。loop 导语可以描述执行这个循环的并行类型，还可以声明循环的私有变量、数组和归约操作。

语法

C 和 C++中，loop 导语的语法是

```
#pragma acc loop [子语 [[,] 子语]...] 换行
    for 循环
```

Fortran 中，loop 导语的语法是

```
!$acc loop [子语 [[,] 子语]…]
    do 循环
```

这里的子语是下列中的一个：

```
collapse(n)
gang [(标量整数表达式)]
worker [(标量整数表达式)]
vector [(标量整数表达式)]
seq
independent
private(列表)
reduction(操作符:列表)
```

一些子语只可用于 parallel 区域环境，而一些只可用于 kernels 区域环境；详述见下文。在 parallel 区域内，一个不带 gang、worker 或 vector 子语的导语，允许编译器自动选择是否使用多个 gang 来的执行该循环、每个 gang 中有多少个 worker、是否以向量操作执行。编译器可能选择使用传统的自动向量化技术来执行任何没有 loop 导语的循环。

2.8.1 collapse 子语

collapse 子语用来指定有多少层紧密嵌套的循环与 loop 构件关联。collapse 子语的参数必须是一个常量正整数表达式。如果没有 collapse 子语，只有紧接着的循环才与 loop 导语相关联。

如果有一个以上的循环与 loop 构件关联，那么被关联循环中的所有循环体都会按照剩余的子语来调度。与 collapse 子语相关联的所有循环的步长必须是可计算的，并在所有循环中保持不变。

编译器自行决定是否将导语的 gang、worker 或 vector 子语应用于每一个循环、线性化迭代空间。

2.8.2 gang 子语

在一个加速器 parallel 区域中，gang 子语要求将循环中的迭代步分摊到 parallel 构件创建的多个 gang 上，从而实现并行执行。不允许有参数。循环的所有迭代步必须是数据独立的，但在 reduction 子语中指定的变量除外。

在一个加速器 kernels 区域中，gang 子语指明，关联循环的迭代步需要在为内核创建的所有 gang 上并行执行。如果指定了一个参数，该参数指明使用多少个 gang 来执行本循环的迭代步。

2.8.3 worker 子语

在一个加速器 parallel 区域内，worker 子语指明，关联循环的迭代步将被分散给单个 gang 的多个 worker 并行执行。不允许使用参数。循环的所有迭代步必须是数据独立的，但 reduction 子句中指定的变量除外。一个带有 worker 子语的循环是否能包含一个带有 gang 子语的循环，由编译器决定。

在一个加速器 kernels 区域内，worker 子语要求，关联循环的迭代步在为内核创建

的所有 gang 的所有 worker 上并行执行。如果指定了一个参数，该子语指明了使用多少个 worker 来执行本循环的迭代步。

2.8.4 seq 子语

seq 子语指明关联循环需要在加速器上串行地执行；这是一个加速器 parallel 区域内的默认执行方式。本子语将阻止编译器的自动并行化、向量化。

2.8.5 vector 子语

在一个加速器 parallel 区域内，vector 子语指明，关联循环的迭代步以向量模式或 SIMD 模式执行。向量的长度由本子语指定或由编译器为本 parallel 区域自动选择。编译器决定一个带 vector 子语的循环是否能够包含一个带有 gang 或 worker 子语的循环。

在一个加速器 kernels 区域，vector 子语指明，关联循环的迭代步以向量模式或 SIMD 模式执行。如果指定了一个参数，迭代步将以这个长度的向量执行；如果没有指定参数，编译器将自行选择一个合适的向量长度。

2.8.6 independent 子语

kernels 区域中的 loop 导语允许使用 independent 子语，这个子语告诉编译器该循环的迭代步是彼此数据独立的。因此允许生成并行执行这些迭代步的代码，且不需要同步。

限制

- 如果有一个迭代步向一个变量或数组写数据，而另外一个迭代步也读或写相同的变量或数组元素，那么使用 independent 子语就是一个编程错误但 reduction 子语中的变量除外。

2.8.7 private 子语

loop 导语上的 private 子语明确要求，对*列表*中的每一项，关联循环的每一个迭代步都要创建一个副本。

2.8.8 reduction 子语

reduction 子语允许用在带有 gang、worker 或 vector 子语的 loop 构件上。它列出一个操作符和一个或多个标量变量。对每一个归约变量，关联循环的每一个迭代步都会创建一个私有副本，并根据操作符进行初始化；详情见 2.4.10 节的表格。在循环结束处，用归约操作符将每一个迭代步的值组合起来，得到的结果在 parallel 区或 kernels 区域结束时存入原始变量。

在 parallel 区域中，如果 reduction 子语用在带有 vector 或 worker 子语（并且不带 gang 子句）的 loop 构件上，并且标量变量出现在 parallel 构件的 private 子语中，那么该标量的私有副本的值将在循环退出时更新。其它情况下，parallel 区域内 loop 构件的 reduction 子语中出现的变量直到该区域结束时才更新。

2.9 Cache 导语

概要

cache 导语可以出现在一个循环的顶部（循环内）。它指定哪些数组元素或子数组需要为循环体而预取到最高层级的缓存中。

语法

C 和 C++中，cache 导语的语法是

```
#pragma acc cache(列表) 换行
```

Fortran 中，cache 导语的语法是

```
!$acc cache(列表)
```

列表中的条目必须为数据元素或者简单子数组。C 和 C++中，简单子数组就是一个数组名字后跟一个标明数组范围的方括号，方括号有起始下标和长度，例如

```
arr[下界:长度]
```

这里的下界可以是一个常量、循环内不变量、循环变量加减一个常量或加减一个循环内不变量，长度是一个常量。

Fortran 中，简单子数组就是一个数组名后跟一个圆括号，圆括号内是一个用逗号分隔的范围列表，范围用下界下标与上界下标指定，例如

```
arr(下界:上界,下界 2:上界 2)
```

下界必须是一个常量、循环内不变量、循环变量加减一个常量或加减一个循环内不变量；更多地，上界与相应下界的差值必须是一个常量。

2.10 组合导语

概要

如果 parallel 构件或 kernels 构件内立即嵌套一个 loop 导语，就可以组合起来简写为 OpenACC 的 parallel loop 构件或 kernels loop 构件。它的含义等同于在 parallel 导语或 kernels 导语内显式地包含一个 loop 导语。任何可以用在 parallel 或 loop 导语上的子语都可以用在 parallel loop 导语上。任何可以用在 kernels 或 loop 导语上的子语都可以用在 kernels loop 导语上。

语法

C 和 C++中，parallel loop 导语的语法是

```
#pragma acc parallel loop [子语 [[,] 子语]...] 换行
        for 循环
```

Fortran 中，parallel loop 导语的语法是

```
!$acc parallel loop [子语 [[,] 子语]…]
        do 循环
[!$acc end parallel loop]
```

紧接着导语的循环就是关联结构块。parallel 或 loop 的任意子语，只要可以用在 parallel 区域，就可以出现在这里。

C 和 C++中，kernels loop 导语的语法是

```
#pragma acc kernels loop [子语 [[,] 子语]...] 换行
        for 循环
```

Fortran 中，kernels loop 导语的语法是

```
!$acc kernels loop [子语 [[,] 子语]…]
        do 循环
[!$acc end kernels loop]
```

紧接着导语的循环就是关联结构块。kernels 或 loop 的任意子语，只要可以用在 kernels 区域，都可能出现在这里。

限制

- 这个组合导语不能出现在加速器 parallel 区域或 kernels 区域之内。
- parallel、kernels、loop 构件的限制同样适用。

2.11 declare 导语

概要

一个 declare 导语用在 Fortran 子例程、函数或模块的声明部分，或者跟在 C/C++变量声明之后。它指定的变量或数组需要在设备内存上开辟空间，在一个函数、子例程或程序的隐式数据区域的生存期之内，这些变量一直驻留设备内存中。它还能指定是否需要在隐式数据区域的入口处将数据值从主机传递到设备内存，是否需要在隐式数据区域的出口处将数据值从设备传递到主机内存。这个导语为变量或数组创建一个可见设备副本。

语法

C 和 C++中，declare 导语的语法是：

```
#pragma acc declare 子语 [[,] 子语]... 换行
```

Fortran 中，declare 导语的语法是：

```
!$acc declare 子语 [[,] 子语]...
```

这里的子语是下列中的一个：

```
copy(列表)
copyin(列表)
copyout(列表)
create(列表)
present(列表)
present_or_copy(列表)
present_or_copyin(列表)
present_or_copyout(列表)
present_or_create(列表)
```

deviceptr(*列表*)
device_resident(*列表*)

该导语所在函数、子例程或程序的隐式数据区域是其关联区域。如果该导语出现在 Fortran MODULE 的子程序中，整个程序的隐式数据区域就是其关联区域。其它情形下，这些子语的使用效果，与显式地使用带相同子语的数据构件完全相同。数据子语的详细讲述在 2.7 节。

限制

- 一个函数、子例程、程序或模块中，一个变量或数组在所有 `declare` 导语的所有子语中只能出现至多一次。
- 子数组不允许出现在 `declare` 导语中。
- 如果一个变量或数组出现在 `declare` 导语中，那么该变量可见区内的任何构件的数据子语中都不能出现该变量或数组。
- Fortran 中，不定尺寸形参数组[1]不能出现在 `declare` 导语中。
- 为了对齐内存和提升性能，编译器可能对加速器上的数组进行维度填充。
- Fortran 中，可以指定指针数组，但设备内存中不再保留指针的指向关系。

2.11.1 device_resident 子语

概要

`device_resident` 明确要求指定的变量需要在加速器设备内存上开辟空间，不在主机内存上开辟空间。

C 和 C++中，这意味着主机不能访问这些变量。对一个函数，*列表*中的变量必须是一个函数的文件静态变量或局部变量。

Fortran 中，如果变量具有 `allocatable` 属性，当主机程序执行到这个变量的 `allocate` 或 `deallocate` 语句时，这个变量的存储空间将在加速器设备内存上开辟和释放。如果变量具有 Fortran 指针属性，它可能在加速器设备内存上分配和释放；如果指针赋值语句的右手边变量出现在一个 `device_resident` 子语中，那么它也可以出现在指针赋值语句的左手边。如果变量既不没有 `allocatable` 属性也没有 `pointer` 属性，那么它必须是一个子程序的本地变量。

2.12 可执行导语

2.12.1 update 导语

概要

`update` 导语用在显式或隐式数据区域中，使用设备内存中数组的值来更新主机内存中相应数组的全部值或部分值，或者使用主机内存中数组的值来更新设备内存中相应数组的全部值或部分值。

语法

[1]见 2.7 节脚注。

C 和 C++中，update 导语的语法是：

```
#pragma acc update 子语 [[,] 子语]... 换行
```

Fortran 中，update 导语的语法是：

```
!$acc update 子语 [[,] 子语]...
```

这里的子语是下列中的一个：

```
host(列表)
device(列表)
if(条件)
async[(标量整数表达式)]
```

update 子语的参数列表是一串用逗号分隔的变量名、数组名或限定范围的子数组。列表中可以出现同一个数组的多个子数组。对 update host，update 子语的效果是将加速器设备内存中的数据复制到主机内存；对 update device，update 子语的效果是将主机内存中的数据复制到加速器设备内存。数据的更新顺序与它们在子语中出现的顺序相同。出现在 host 或 device 子语的变量和数组必须有一份可见设备副本。host 子语和 device 子语必须至少出现一个。

2.12.1.1 host 子语

host 子语明确要求将列表中的变量、数组或子数组从加速器设备内存复制到主机内存。

2.12.1.2 device 子语

device 子语明确要求将列表中的变量、数组或子数组从主机内存复制到加速器设备内存。

2.12.1.3 if 子语

if 子语是可选项；没有 if 子语的时候，编译器将生成无条件更新的代码。有 if 子语的时候，编译将生成有条件更新的代码，只有当条件的值在 C/C++中为非零或在 Fortran 中为.true.时才进行更行。

2.12.1.4 async 子语

async 子语是可选项；没有 async 子语的时候，在执行 update 导语后的任何代码之前，主机进程会一直等待直至所有更新完成。如果有 async 子语，在主机进程继续执行导语后面代码的同时，进行异步地更新。

如果 async 子语有一个参数，那么这个参数必须是一个整数变量（C/C++中的 int，

Fortran 中的 integer）的名字。该变量可以用到 wait 导语中和多个运行时例程中，使主机进程能检测更新是否完成，或使主机进程等待更新的完成。async 子语也可没有参数，这种情况下，编译器将使用一个与程序中所有 async 的显式参数都不相同的值。

具有相同参数值的两个异步活动在设备上执行的顺序，与主机进程遇到它们的顺序相同。两个具有不同参数值的异步活动在设备上执行的相对顺序是任意的。如果两个（含）以上主机线程执行或共享同一个加速器设备，具有相同参数值的两个异步活动将在设备上一个接一个的执行，执行的相对顺序未定义。

限制

- update 导语是可执行的。在 C 和 C++中，它不准出现在紧跟 *if*、*while*、*do*、*switch*、*label* 的语句位置上；在 Fortran 中，它不准出现在紧跟逻辑 *if* 的语句位置上。
- 出现在 update 导语*列表*中的变量、数组必须有一份可见设备副本。

2.12.2 wait 导语

概要

wait 子语导致主机等待一个异步活的完成，例如一个加速器 parallel 区域、kernels 区域或 update 导语。

语法

C 和 C++中，wait 导语的语法是：

```
#pragma acc wait [( 标量整数表达式 )]  换行
```

Fortran 中，wait 导语的语法是：

```
!$acc wait [(标量整数表达式)]
```

如果指定了参数，参数必须是一个整数表达式（C/C++中的 int，Fortran 中的 integer）。主机线程一直等待，直至带有相同参数的 async 子语引发的所有异步活动完成。

如果没有指定参数，主机进程将等待，直至所有异步活动完成。

如果有两个或两个以上线程执行和共享同一个加速器设备，一个 wait 子语将导致主机线程至少等待它自己发起的所有异步活动都完成。但不能保证其它线程发起的类似异步活动也都完成。

第3章 运行时库例程

本章讲述供程序员使用的 OpenACC 运行时库例程。在不支持 OpenACC 接口的系统上使用这些例程可能会影响移植性。利用预处理变量_OPENACC 进行条件编译可以保持移植性。

本章包括两部分：

- 运行时库的定义
- 运行时库例程

限制

Fortran 中，没有任何一个 OpenACC 运行时库例程可以在 PURE 或 ELEMENTAL 中调用。

3.1 运行时库的定义

对 C 和 C++，本章讲述的运行时库例程的原型保存在一个名为 openacc.h 的头文件中。所有的库例程都是用“C”连接的 extern 函数。这个文件中定义[1]：

- 本章中所有例程的原型。
- 这些原型中使用的所有数据类型，包括一个描述加速器类型的枚举类型。

对 Fortran，接口声明都保存在一个名为 openacc_lib.h 的 Fortran 包含文件中和一个名为 openacc 的模块文件中。这两个文件中定义：

- 本章中所有例程的接口。
- 整数参数 openacc_version，参数值为 *yyyymm*，其中 *yyyy* 和 *mm* 分别是年份和月份，表示所支持的加速器编程模型的版本。这个值与预处理变量_OPENACC 的值相同。
- 用以定义这些例程的整数参数长度的整数。
- 用以描述加速器类型的整数参数。

很多例程都接受或返回一个表示加速器设备类型的值。C 和 C++中，表示设备类型值的数据类型是 acc_device_t；Fortran 中，这个数据类型是 integer(kind=acc_device_kind)。具体用什么值表示设备类型由编译器决定，C 和 C++中，这些值在头文件 openacc.h 中列出；Fortran 中，这些值在 Fortran 包含文件 openacc_lib.h 和 Fortran 模块 openacc_lib 中列出。所有的编译器都支持四个值：acc_device_none，acc_device_default，acc_device_host 和 acc_device_not_host。若想了解其它的值，请查看编译器包含的相应文件，或者阅读编译器文档。acc_device_default 不会是任何函数的返回值；用作输入参数时，它告诉运行时库使用编译器的默认设备类型。

[1] 编译器可能会在头文件添加额外的定义，例如 PGI 编译器就添加了一些自行定义的例程和宏。--译者注。

3.2 运行时库例程

3.2.1 acc_get_num_devices

概要

例程 acc_get_num_devices 返回主机上指定类型的加速器设备数量。

格式

C 或 C++：

```
int acc_get_num_devices( acc_device_t );
```

Fortran:

```
integer function acc_get_num_devices( devicetype )
integer(acc_device_kind) devicetype
```

描述

例程 `acc_get_num_devices` 返回主机上指定类型的加速器设备数量。输入参数说明对哪种类型的设备计数。

3.2.2 acc_set_device_type

概要

例程 `acc_set_device_type` 告诉运行时环境使哪种类型的设备来执行加速器 parallel 区域和 kernels 区域。当编译器允许被编译程序在一种以上类型的加速器上运行时，这个例程很有用。

格式

C 或 C++：

```
void acc_set_device_type ( acc_device_t );
```

Fortran:

```
subroutine acc_set_device_type ( devicetype )
integer(acc_device_kind) devicetype
```

描述

在所有可用的设备类型中，例程 `acc_set_device_type` 告诉运行时环境使用哪种类型的设备。为更高效，这个例程应该在任何加速器数据区域、parallel 区域、kernels 之前调用，或在调用 `acc_shutdown` 之后调用。

限制

- 在加速器 parallel 区域、kernels 区域、数据区域内不可调用本例程。
- 如果指定的加速器类型不可用，例程的行为将由编译器决定；特别地，程序可能中止。
- 如果本例程以不同的设备类型参数调用一次以上，且中间没有调用 `acc_shutdown`，调用效果由编译器自行定义。
- 如果某些加速器区域仅为某一种设备类型编译，那么以不同的设备类型调用本例程

可能产生未定义的行为。

3.2.3 acc_get_device_type

概要

如果已经选定了一种设备类型，例程 acc_get_device_type 将告诉程序运行下一个加速器区域时使用的设备类型。当编译器允许将代码编译成能在一种以上加速器上运行的程序时，本例程有用。

格式

C 或 C++：

```
acc_device_t acc_get_device_type ( void );
```

Fortran:

```
function acc_get_device_type ()
integer(acc_device_kind) acc_get_device
```

描述

如果已经选定了一种设备类型，例程 acc_get_device_type 返回一个值，告诉程序运行下一个加速器区域时使用什么类型的设备。选定设备类型的方法可能是调用 acc_set_device_type、使用一个环境变量，或程序默认行为。本例程仅对那些能在一种以上类型的加速器设备上运行的加速区域有效。

限制

- 在加速器 parallel 区域、kernels 区域内不可调用本例程。
- 如果还没有选定任何设备类型，返回值为 acc_device_none.

3.2.4 acc_set_device_num

概要

acc_set_device_num 告诉运行时环境使用哪一个设备。

格式

C 或 C++:

```
void acc_set_device_num( int, acc_device_t );
```

Fortran:

```
subroutine acc_set_device_num( devicenum, devicetype )
integer devicenum
integer(acc_device_kind) devicetype
```

描述

在指定类型的所有设备中，acc_set_device_num 告诉运行时库环境用哪一个设备。如果 devicenum 的值为零，运行时环境将恢复到编译器自行定义的默认行为。如果第二个参数是零，选定的设备编号将应用到所有的加速器类型。

限制

- 在加速器 parallel 区域、kernels 区域、数据区域内不可调用本例程。

- 对指定的设备类型，如果 devicenum 的值大于 acc_get_num_devices 的返回值，那么例程行为由编译器定义。
- acc_set_device_num 会隐式调用 acc_set_device_type，使用设备类型作为参数。

3.2.5 acc_get_device_num

概要

acc_get_device_num 例程返回指定设备类型的一个设备编号，该设备将运行下一个加速器 parallel 或 kernels 区域。

格式

C 或 C++：

```
int acc_get_device_num( acc_device_t );
```

Fortran:

```
integer function acc_get_device_num( devicetype )
integer(acc_device_kind) devicetype
```

描述

acc_get_device_num 例程返回一个与指定类型设备编号相对应的整数，该设备将执行下一加速器 parallel 区域或 kernels 区域。

限制

- 在加速器 parallel 区域、kernels 区域内不可调用本例程。

3.2.6 acc_async_test

概要

acc_async_test 例程检测所有关联的异步活动是否完成。

格式

C 和 C++：

```
int acc_async_test( int );
```

Fortran:

```
logical function acc_async_test( arg )
integer(acc_handle_kind) arg
```

输入参数必须是一个整数表达式。如果这个值出现在一个或更多个 async 子语中，并且这些异步活动均已完成，那么 acc_async_test 例程返回一个非零值或.true.。如果某些异步活动尚未完成，acc_async_test 例程将返回零或者.false.。如果有两个或两个以上主机线程共享同一个加速器，仅当本主机线程发起的所有匹配异步活动都已经完成，acc_async_test 例程才返回零或.false.；但不保证由其它主机线程发起的所有匹配异步活都已经完成。

限制

- 本例程不能在加速器 parallel 区域或 kernels 区域内调用。

3.2.7 acc_async_test_all

概要

acc_async_test_all 例程等待所有异步活动的完成。

格式

C 和 C++：

```
int acc_async_test_all( );
```

Fortran:

```
logical function acc_async_test_all( )
```

描述

如果所有异步活动都已经完成，acc_async_test_all 将返回一个非零值或.true.如果某个异步活动尚未完成，acc_async_test_all 将返回零或.false.。 如果两个或两个以上主机线程共享同一个加速器，仅当本主机线程发起的所有异步活动均已完成，acc_async_test_all 才返回零或.false.；不保证其它主机线程发起的异步活动均已完成。

限制

- 本例程不能在加速器 parallel 区域或 kernels 区域内调用。

3.2.8 acc_async_wait

概要

acc_async_wait 等待所有关联的异步活动完成。

格式

C 和 C++：

```
void acc_async_wait( int );
```

Fortran:

```
subroutine acc_async_wait( arg )
integer(acc_handle_kind) arg
```

描述

输入参数必须是一个整数表达式。如果这个值出现在一个或更多的 async 子语中，acc_async_wait 将一直等待，直到最后一个异步活动完成才返回。如果两个或两个以上主机线程共享同一个加速器，本主机线程发起的所有匹配异步活动都完成以后，acc_async_wait 例程才返回；不保证其它主机线程发起的所有匹配异步活动都已完成。

限制

- 本例程不能在加速器 parallel 区域或 kernels 区域内调用。

3.2.9 acc_async_wait_all

概要

acc_async_wait 等待所有异步活动完成。

语法

C 或 C++:

```
void acc_async_wait_all( );
```

Fortran:

```
subroutine acc_async_wait_all( )
```

描述

acc_async_wait_all 一直等待，直到最后一个异步活动完成才返回。如果两个或两个以上主机线程共享同一个加速器，本主机线程发起的所有异步活动都完成以后，acc_async_wait_all 例程才返回；不保证其它主机线程发起的所有异步活动都已完成。

限制

- 本例程不能在加速器 parallel 区域或 kernels 区域内调用。

3.2.10 acc_init

概要

acc_init 例程告诉运行时环境将指定设备类型的运行时环境初始化。统计性能数据时，本例程用于分离初始化时间和计算时间。

格式

C 和 C++：

```
void acc_init ( acc_device_t );
```

Fortran:

```
subroutine acc_init ( devicetype )
integer(acc_device_kind) devicetype
```

描述

acc_init 例程也会调用 acc_set_device. 为更有效率，本例程应在所有加速区域之前调用，或者在调用 acc_shutdown 之后调用。

限制

- 在加速器 parallel 区域、kernels 区域内不可调用本例程。
- 如果指定的加速器类型不可用，例程的行为将由编译器决定；特别地，程序可能中止。
- 如果本例程被以不同的设备类型参数调用一次以上，且中间没有调用 acc_shutdown，调用效果由编译器自行定义。
- 如果某些加速器区域仅为某一种设备类型而编译，那么以不同的设备类型调用本例程可能产生未定义的行为。

3.2.11 acc_shutdown

概要

acc_shutdown 告诉运行时环境关掉与指定加速器设备的连接，释放所有的运行时资

源。如果程序可运行在不同类型的设备上，本例程可以用来连接一个不同的设备。

格式

C 或 C++：

```
void acc_shutdown ( acc_device_t );
```

Fortran:

```
subroutine acc_shutdown ( devicetype )
integer(acc_device_kind) devicetype
```

描述

`acc_shutdown` 例程断开程序与加速器设备的连接。

限制

- 在加速器 `parallel` 区域、`kernels` 区域、数据区域内不可调用本例程。

3.2.12 acc_on_device

概要

`acc_on_device` 例程告诉程序它是否正在一个特定的设备上执行。

格式

C 或 C++：

```
int acc_on_device ( acc_device_t );
```

Fortran:

```
logical function acc_on_device ( devicetype )
integer(acc_device_kind) devicetype
```

描述

根据是否正在主机或某个加速器上运行，`acc_on_device` 例程可以用来执行不同的路径。 如果有一个编译时常量参数，`acc_on_device` 例程在编译阶段就返回一个常量。参数必须为一个已定义的加速器类型。如果参数是 `acc_device_host`，那么在加速器并行区域与内核区域的外部，和在主机处理器上执行的加速器并行区域与内核区域内部，本例程都将返回一个非零整数（对 C 和 C++）或 `.true.`（对 Fortran）；其它情况下，本例程返回零（对 C 和 C++）或 `.false.`（对 Fortran）。

3.2.13 acc_malloc

概述

`acc_malloc` 例程在加速器设备上分配内存。

格式

C 或 C++:

```
void* acc_malloc ( size_t );
```

描述

`acc_malloc` 例程可以用来在加速器设备上分配内存。本函数得到的指针可以用在

deviceptr 子语中，告诉编译器这个指针的目标已经驻留在加速器上。

3.2.14 acc_free

概述

acc_free 例程释放加速器设备上的内存空间。

格式

C 或 C++：

```
void acc_free ( void* );
```

描述

acc_free 例程将释放先前在加速器设备上分配的内存空间；输入参数必须是调用 acc_malloc 返回的指针。

第4章 环境变量

本章讲述用以改变加速器区域行为的环境变量。环境变量的名字必须大写。赋给环境变量的值是区分大小写的，值的前后都可以有空白字符。在程序开始之后，如果环境变量的值发生了改变，即使是程序自己修改的这些值，修改后形为仍有编译器定义。

4.1 ACC_DEVICE_TYPE

如果程序被编译成可以在一种以上类型的设备上执行的版本，环境变量 ACC_DEVICE_TYPE 控制执行加速器 parallel 区域或 kernels 区域时使用的默认设备类型。这个环境变量允许使用的值由编译器定义。请在发行说明里查看本环境变量当前支持哪些值。

例子：

```
setenv ACC_DEVICE_TYPE NVIDIA
export ACC_DEVICE_TYPE=NVIDIA
```

4.2 ACC_DEVICE_NUM

环境变量 ACC_DEVICE_NUM 控制着执行加速器区域时使用的默认设备编号。这个环境变量的值必须是一个非负整数，取值范围下限是零，上限是主机上所用类型设备的数量。如果值为零，使用编译器定义的默认设备。如果值超过主机上的设备数量，行为由编译器定义。

例子：

```
setenv ACC_DEVICE_NUM 1
export ACC_DEVICE_NUM=1
```

第5章 词汇表

清晰一致的术语对讲述任何编程模型都很重要。你必须理解我们在这里定义的名词，以便有效利用本文档及文档中的编程模型。

加速器（accelerator） – 一个挂载在 CPU 上的特殊功能协处理器，CPU 可以将数据和计算核心卸载到它上面进行计算密集型运算。

屏障(barrier) – 一种同步，在任何并行执行单元或线程被允许通过此屏障之前，所有的并行执行单元或线程必须到达该屏障；就像赛马场起点上的障。

计算密度(compute intensity) – 对一个给定的循环、区域或程序单元，计算密度就是在数据上完成的算术操作次数除以在两个内存层级间的内存存取次数得到的比值。

构件（construct） - 一个导语及其关联的语句、循环或结构块（如果有）。

计算区域（compute region） - 一个 `parallel` 区域或一个 `kernels` 区域。

CUDA – 来自英伟达的 CUDA 环境是一个类 C 编程环境，它用来显式地控制英伟达 GPU 和在英伟达 GPU 上编程。

数据区域（data region） - 用加速器 `data` 构件定义的一个区域，或者是包含加速器导语的例程、程序的一个隐式数据区域。`data` 构件通常会请求开辟设备内存空间，在入口处将数据从主机复制到设备内存，在出口处将数据从设备复制回主机内存并释放设备内存。数据区域可以包含其它数据区域和计算区域。

设备(device) – 所有类型加速器的统一称谓。

设备内存(device memory) – 加速器的附属内存，在物理上与逻辑上都与主机内存相分离。

导语(directive) – 在 C 或 C++中，一个`#program`；在 Fortran 中，一个特殊格式的注释语句。编译器能从导语中得到控制程序形为的额外信息。

DMA – 直访（直接内存访问，Direct Memory Access），在物理上相互分离的内存间移动数据的一种方法；它通常由一个与主机 CPU 相分离的直访引擎来完成，直访引引擎可以像访问一个 IO 设备一样访问主机物理内存或其它物理内存。

GPU – 图形处理单元（Graphics Processing Unit）；加速器设备的一种。

GPGPU – 图形处理单元上的通用计算（General Purpose）。

主机（host） - 本文档中，主机是具有一个附属加速器设备的 CPU。主机 CPU 将程序区域和数据加载到设备上，并在设备上运行。

内核（kernel） - 被加速器并行执行的嵌套循环。通常循环被划分为一个并行区块，循环的主体也就成为内核的主体。

内核区域（kernel region） - 一个加速器 kernels 构件定义的区域。内核区域就是一个为加速器编译的结构块。编译器将内核区域内的代码分割为一系列内核；通常，每个循环成为一个独立的内核。一个内核区域可能会要求开辟设备内存空间，在区域入口处将数据从主机复制到设备，离开区域时将数据从设备内存复制到主机内存。在这个版本的标准中，内核区域不能包含其它的内核区域。

循环次数（loop trip count） - 某个特定的循环被执行的次数。

MIMD – 多指令多数据（Multiple Instruction, Multiple Data），一种并行执行方法。不同的运算单元或线程相互间异步地执行不同的指令流。

OpenCL – 开放计算语言（Open Compute Language）的缩写，一个发展中的、可移植的类 C 语言编程标准，它能在 GPU 和其它加速器上进行低层通用编程。

并行区域（parallel region） - 加速器 parallel 构件定义的一个区域。并行区域就是专为加速器编译的结构块。一个并行区域通常包含一个或多个工作分担(work-sharing)循环。一个并行区域可能会要求开辟设备内存空间，在区域入口处将数据从主机复制到设备，离开区域时将数据从设备内存复制到主机内存。在这个版本的标准中，并行区域不能包含其它的计算区域。

区域（region） - 一个构件的执行实例中遇到的所有代码。一个区域包括被调用例程的所有代码，也可以认为是一个构件的动态上下文。它可能是 `parallel` 区域、`kernels` 区域、`data` 区域或隐式 `data` 区域。

SIMD – 单指令多数据（Single-Instruction, Multiple-Data），一种并行执行方法，将单个指令同时用到多个数据单元上。

SIMD 操作（SIMD operation） - 一个用 SIMD 指令实现的向量操作。

结构块（structured block） - C 或 C++中，一个可执行语句，也可以是复合语句，该语句顶部有唯一入口，底部有唯一出口。Fortran 中，顶部有唯一入口且底部有唯一出口的一块可执行语句。

向量操作（vector operation） - 一致地作用于一个数组全部元素的单个操作或一系列操作。

可见设备副本（visible device copy） - 变量、数组、子数组在设备内存中的一个副本，直到编译时，这些副本对程序可见。