



第四讲

消息传递编程接口 MPI

一、MPI 编程基础

主要内容

- MPI 安装、程序编译与运行

- MPI 编程基础

 - ◆ MPI 进程 / 进程组

 - ◆ MPI 通信器

 - ◆ MPI 消息

 - ◆ MPI 程序基本结构

- MPI 程序编译与运行

- MPI 数据类型

- MPI 几个常用接口

MPI 介绍

□ Message Passing Interface

- 消息传递编程标准，目前最为通用的并行编程方式
- 提供一个高效、可扩展、统一的并行编程环境
- MPI 是一个库，不是一门语言，MPI 提供库函数 / 过程供 C/FORTRAN 调用
- MPI 是一种标准或规范的代表，并不是一个具体实现
- 所有的并行计算机制造商都提供对 MPI 的支持
- MPI 是一种消息传递编程模型，最终目的是服务于进程间通信这一目标

MPI 介绍

□ MPI 的目标

- 较高的通信性能;
- 较好的程序可移植性;
- 强大的功能

□ MPI 的实现 —— 免费版本

- 1994 年公布 MPI 1.0 标准, 1998 年公布 2.0 标准
- MPI 1.0: MPICH 1.2.7p1
- MPI 2.0: MPICH2 1.1.1p1

□ MPI 商业版本

- 一些厂商也提供商业版的 MPI 系统, 许多是在 MPICH 的基础上优化产生的

MPI 下载与安装

□ MPICH 下载

<http://www.mpich.org/>

□ MPICH 的安装

- 参考 MPICH Install Guide

□ MPICH 的使用

- 参考 MPICH User Guide

进程与通信器

□ MPI 进程

- MPI 程序中一个独立参与通信的个体

□ MPI 进程组

- MPI 程序中由部分或全部进程构成的有序集合
- 每个进程都被赋予一个所在进程组中唯一的序号(rank), 用于在该组中标识该进程, 称为进程号, 取值从 0 开始

进程的具体个数由用户在递交并行任务时指定

□ MPI 通信器 / 通信子 (Communicator)

- MPI 程序中进程间的通信必须通过通信器进行
- 通信器分为域内通信器 (同一进程组内的通信) 和域间通信器 (不同进程组的进程间的通信)

进程与通信器

- MPI 程序启动时自动建立两个通信器:

- `MPI_COMM_WORLD` : 包含程序中所有 MPI 进程

- `MPI_COMM_SELF` : 单个进程独自构成, 仅包含自己

- ◆ MPI 程序中, 一个 MPI 进程

- 由一个进程组和在该组中的进程号唯一确定; 或
由一个通信器和在该通信器中的进程号唯一确定

- ◆ 进程号是相对进程组或通信器而言的, 同一进程在不同的进程组或通信器中可以有不同的进程号

- ◆ 进程号是在进程组或通信器被创建时赋予的

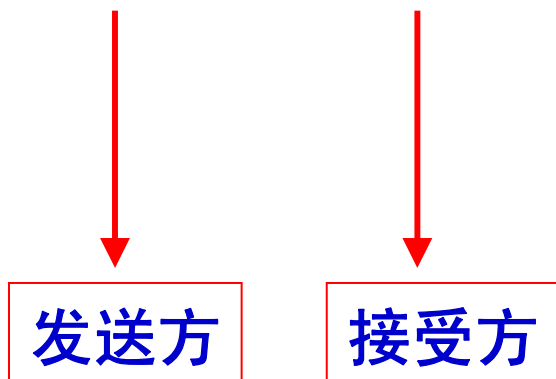
- ◆ 空进程: `MPI_PROC_NULL`

- ◆ 与空进程通信时不做任何操作

MPI 消息

□ 消息（message）

- 一个消息指进程间进行的一次数据交换
- 一个消息由
通信器、源地址、目的地址、消息标签、和数据构成



第一个 MPI C 程序

```
#include "mpi.h"
```

mpi.h 是 MPI 相对于 C 语言的头文件

```
#include <stdio.h>
```

调用 MPI 函数时必须包含 MPI 头文件

MPI 的初始化和结束

```
int main(int argc, char *argv)
```

MPI 预定义的宏：所允许的机器名字的最大长度

```
{
```

```
    int myid, np, namelen;
```

```
    char proc_name[MPI_MAX_PROCESSOR_NAME];
```

```
    MPI_Init(&argc, &argv);
```

获取本进程的进程号

```
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
```

进程号取值范围为 0, ..., np-1

```
    MPI_Comm_size(MPI_COMM_WORLD, &np);
```

```
    MPI_Get_processor_name(proc_name, &namelen);
```

```
    fprintf(stderr, "Hello, I am proc. %d of %d on %s\n",  
            myid, np, proc_name);
```

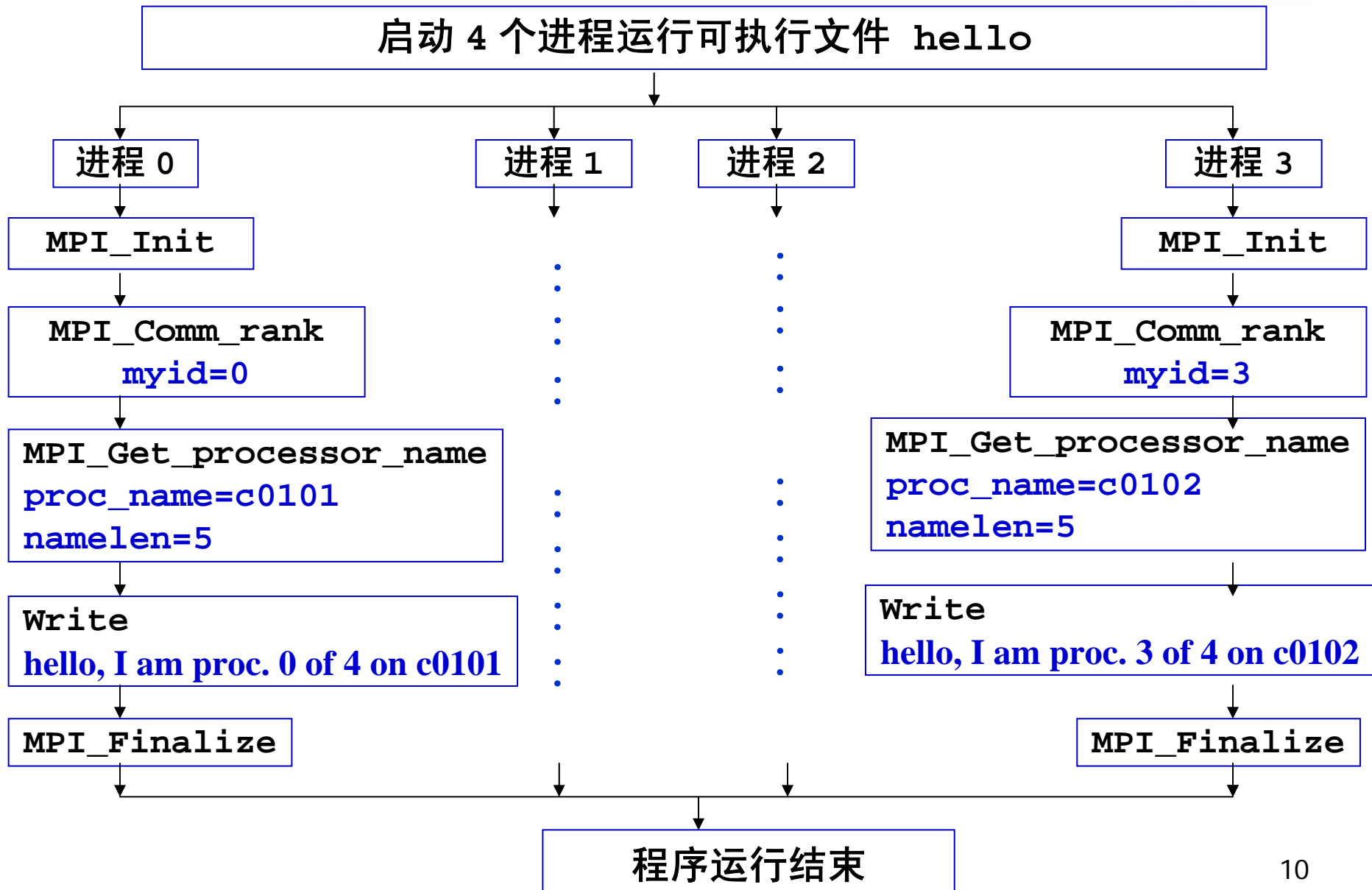
获取所有参加运算的进程的个数

```
    MPI_Finalize();
```

获取运行本进程所在的结点的主机名

```
}
```

MPI 程序执行过程



MPI 程序分析

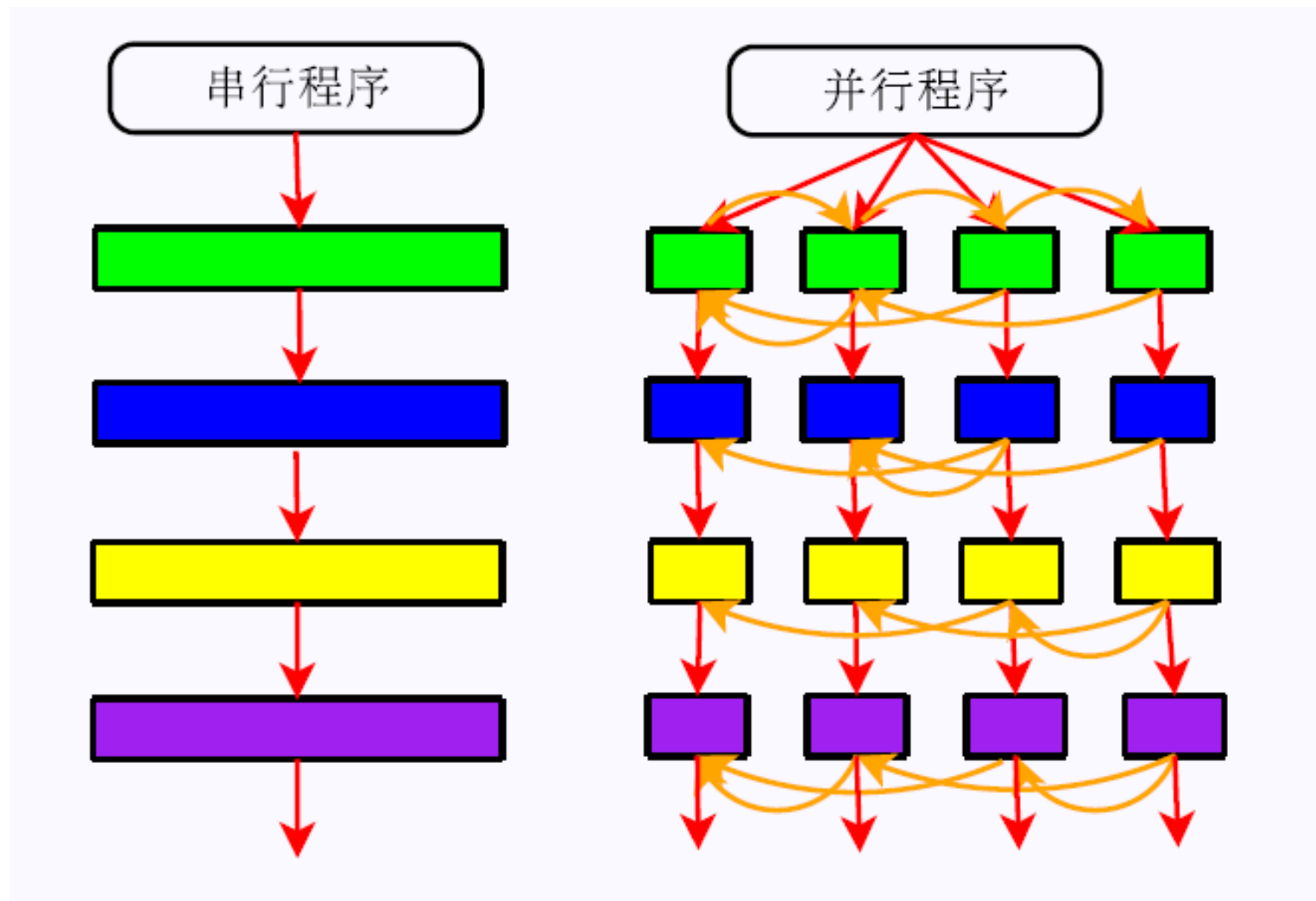
◆ 在单个结点(c0101)上，开 4 个进程的运行结果

```
Hello, I am Proc. 1 of 4 on c0101  
Hello, I am Proc. 0 of 4 on c0101  
Hello, I am Proc. 2 of 4 on c0101  
Hello, I am Proc. 3 of 4 on c0101
```

◆ 在四个结点上，开 4 个进程的运行结果

```
Hello, I am Proc. 1 of 4 on c0101  
Hello, I am Proc. 3 of 4 on c0102  
Hello, I am Proc. 2 of 4 on c0104  
Hello, I am Proc. 0 of 4 on c0103
```

MPI 程序执行过程



MPI 编程惯例

- MPI 的所有常量、变量与函数均以 `MPI_` 开头
- MPI 的 C 语言接口为函数
- 在 C 程序中，所有常数的定义除下划线外一律由大写字母组成，在函数和数据类型定义中，接 `MPI_` 之后的第一个字母大写，其余全部为小写字母，即 `MPI_Xxxx_xxx` 形式
- MPI 程序的开始和结束必须是 `MPI_Init` 和 `MPI_Finalize`，分别完成 MPI 的初始化和结束工作
- 除 `MPI_Wtime` 和 `MPI_Wtick` 外，所有 C 函数调用之后都将返回一个错误信息码

MPI 编程惯例

- MPI 是按进程组(**Process Group**) 方式工作:
所有 MPI 程序在开始时均被认为是在通信器 **MPI_COMM_WORLD** 所拥有的进程组中工作,
之后用户可以根据需要, 建立其它的进程组
- 所有 MPI 的通信一定要在**通信器**中进行

编译与运行

□ MPI 程序的编译

- C 编写的 MPI 程序

```
mpicc -o hello hello.c
```

□ MPI 程序的运行

- `mpirun`

- `mpiexec` （使用 `ssh` 启动 MPI 进程）

MPICH2 中的 `mpirun` 和 `mpiexec` 是同一个命令

MPI 编程初步

数据类型与基本接口

MPI 数据类型

◆ MPI 定义了一些基本数据类型

主要用于消息传递

■ MPI 数据类型命名规则

以 MPI_ 开头，后面跟 C 语言原始数据类型名

全部为大写！

MPI 数据类型分：

原始数据类型和自定义数据类型

| MPI datatype | C datatype |
|--------------------|--------------------|
| MPI_CHAR | signed char |
| MPI_SHORT | signed short int |
| MPI_INT | signed int |
| MPI_LONG | signed long int |
| MPI_UNSIGNED_CHAR | unsigned char |
| MPI_UNSIGNED_SHORT | unsigned short int |
| MPI_UNSIGNED_INT | unsigned int |
| MPI_UNSIGNED_LONG | unsigned long int |
| MPI_FLOAT | float |
| MPI_DOUBLE | double |
| MPI_LONG_DOUBLE | long double |
| MPI_BYTE | |
| MPI_PACKED | |

自定义数据类型以后再介绍

Fortran 77 原始数据类型

| MPI datatype | Fortran 77 datatype |
|----------------------|---------------------|
| MPI_INTEGER | INTEGER |
| MPI_REAL | REAL |
| MPI_DOUBLE_PRECISION | DOUBLE PRECISION |
| MPI_COMPLEX | COMPLEX |
| MPI_DOUBLE_COMPLEX | DOUBLE COMPLEX |
| MPI_LOGICAL | LOGICAL |
| MPI_CHARACTER | CHARACTER*1 |
| MPI_BYTE | |
| MPI_PACKED | |

MPI 常量

■ MPI 定义一组常量

● MPI 常量命名规则：全部大写

MPI_MAX_PROCESSOR_NAME

MPI_PROC_NULL

MPI_COMM_WORLD

MPI_COMM_SELF

MPI_COMM_NULL

MPI_ANY_SOURCE

MPI_ANY_TAG

MPI_TAG_UB

MPI_LB

MPI_UB

MPI_BOTTOM

... ..

MPI 常用接口

- MPI_INIT
- MPI_FINALIZE
- MPI_COMM_RANK
- MPI_COMM_SIZE
- MPI_SEND
 - MPI_SENDRECV
 - MPI_SENDRECV_REPLACE
 - MPI_GET_COUNT
 - MPI_ABORT
 - MPI_WTIME
 - MPI_GET_PROCESSOR_NAME
- MPI_RECV

MPI_INIT

MPI_INIT: MPI 初始化

| | |
|----|--|
| 参数 | 无 |
| C | <code>int MPI_Init(int *argc, char ***argv)</code> |

- 该函数初始化 MPI 并行程序的执行环境
- 它必须在调用所有其它 MPI 函数之前被调用
(除 `MPI_INITIALIZED`)
- 在一个 MPI 程序中，只能被调用一次

MPI_FINALIZE

MPI_FINALIZE: 结束 MPI 系统

| | |
|----|-------------------------------------|
| 参数 | 无 |
| C | <code>int MPI_Finalize(void)</code> |

- 该函数清除 MPI 环境的所有状态
- 它被调用后，所有MPI 函数都不能再调用，包括 MPI_INIT

MPI_COMM_RANK

MPI_COMM_RANK(comm, rank)

| | | | |
|----|--|------|--------------------|
| 参数 | IN | comm | 通信器 |
| | OUT | rank | 本进程在通信器 comm 中的进程号 |
| C | <pre>int MPI_Comm_rank(MPI_Comm comm, int *rank)</pre> | | |

- 该函数返回本进程在指定通信器中的进程号

MPI_COMM_SIZE

MPI_COMM_SIZE(comm, size)

| | | | |
|----|---|------|-----------------|
| 参数 | IN | comm | 通信器 |
| | OUT | size | 该通信器 comm 中的进程数 |
| C | <pre>int MPI_Comm_size(MPI_Comm comm, int *size)</pre> | | |

- 该函数返回指定通信器所包含的所有进程个数

获取结点主机名

MPI_GET_PROCESSOR_NAME(name, namelen)

| | | |
|----|--|--------|
| 参数 | OUT name | 结点主机名 |
| | OUT namelen | 主机名的长度 |
| C | <pre>int MPI_Get_processor_name(char *name, int *namelen)</pre> | |

- 该函数返回进程所在结点的主机名

MPI_ABORT

MPI_ABORT(comm, errorcode)

| | | | |
|-----|--|-----------|-----|
| 参 数 | IN | comm | 通信器 |
| | OUT | errorcode | 错误码 |
| C | int MPI_Abort(MPI_Comm comm, int errorcode) | | |

- 异常终止 MPI 程序的执行，MPI 系统会尽量设法终止通信器中的所有进程

MPI_WTIME

MPI_WTIME()

| | |
|----|-------------------------------------|
| 参数 | 无 |
| C | <code>double MPI_Wtime(void)</code> |

- 该函数返回当前的墙钟时间（系统时间）

例: `cpi.c`

MPI_WTICK

MPI_WTICK()

| | |
|----|-------------------------------------|
| 参数 | 无 |
| C | <code>double MPI_Wtick(void)</code> |

- 该函数返回 MPI_WTIME 的时钟精度，单位为秒

MPI_WTIME 和 MPI_WTICK 是 MPI 对于 C 语言的仅有的两个返回双精度值而非整型错误码的 MPI 函数！

MPI_GET_VERSION

`MPI_GET_VERSION(version, subver)`

| | | |
|----|--|------|
| 参数 | OUT version | 主版本号 |
| | OUT subver | 次版本号 |
| C | <code>int MPI_Get_version(int *version, int *subver)</code> | |

- 该函数返回 MPI 的版本号

MPI_SEND

MPI_SEND(buf, count, datatype, dest, tag, comm)

| | | | |
|-----|--|----------|-------------|
| 参 数 | IN | buf | 所发送消息的首地址 |
| | IN | count | 将发送的数据的个数 |
| | IN | datatype | 发送数据的数据类型 |
| | IN | dest | 接收消息的进程的标识号 |
| | IN | tag | 消息标签 |
| | IN | comm | 通信器 |
| C | <pre>int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)</pre> | | |

- 阻塞型消息发送接口
- 最基本的点对点通信函数之一

MPI_SEND

MPI_SEND(buf, count, datatype, dest, tag, comm)

- MPI_SEND 将缓冲区中 count 个 datatype 类型的数据发给进程号为 dest 的目的进程
 - 这里 count 是元素个数，即指定数据类型的个数，不是字节数，数据的起始地址为 buf
 - datatype 是 MPI 数据类型
 - 本次发送的消息标签是 tag，使用标签的目的是把本次发送的消息和本进程向同一目的进程发送的其它消息区别开来
 - dest 的取值范围为 0~np-1 (np 表示通信器 comm 中的进程数) 或 MPI_PROC_NULL, tag 的取值为 0~ MPI_TAG_UB
 - 该函数可以发送各种类型的数据，如整型、实型、字符等
- 点对点通信是 MPI 通信机制的基础

MPI_RECV

MPI_RECV (buf , count , datatype , source , tag , comm , status)

| | | | |
|-----|---|----------|-------------|
| 参 数 | OUT | buf | 接收消息数据的首地址 |
| | IN | count | 接收数据的最大个数 |
| | IN | datatype | 接收数据的数据类型 |
| | IN | source | 发送消息的进程的标识号 |
| | IN | tag | 消息标签 |
| | IN | comm | 通信器 |
| | OUT | status | 返回状态 |
| C | <pre>int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)</pre> | | |

- 阻塞型消息接收接口
- 最基本的点对点通信函数之一

MPI_RECV

MPI_RECV (buf, count, datatype, source, tag, comm, status)

- 从指定的进程 source 接收不超过 count 个 datatype 类型的数据，并把它放到缓冲区中，起始位置为 buf，本次消息的标识为 tag
- source 取值范围为 0 ~ np-1，或 MPI_ANY_SOURCE，或 MPI_PROC_NULL，tag 取值为 0~MPI_TAG_UB 或 MPI_ANY_TAG
- 接收消息时返回的状态 STATUS，在 C 语言中是用结构定义的，可供查询的成员包括 MPI_SOURCE，MPI_TAG 和 MPI_ERROR。
- 此外，STATUS 还包含接收消息元素的个数，但它不是显式给出的，需要调用函数 MPI_GET_COUNT 查询