



消息传递编程接口 MPI

(一) MPI 编程基础

目录页

Contents

- 1 MPI 安装、编译与运行
- 2 MPI 数据类型
- 3 消息发送和接收
- 4 MPI 一些常用函数

1

MPI 安装、编译与运行

- 1 MPI 安装、编译与运行
- 2 MPI 数据类型
- 3 消息发送和接收
- 4 MPI 一些常用函数

MPI 介绍

MPI: Message Passing Interface

- <https://www.mpi-forum.org/>
- 消息传递编程标准，提供一个高效、可扩展、统一的并行编程环境，是目前最为通用的分布式并行编程方式。
- MPI 是一种消息传递编程模型，是一种标准或规范，MPI 实现通过提供库函数实现进程间通信，从而进行并行计算，目前所有并行机制造商都提供对MPI 的支持。
- MPI 是一个库，不是一门语言，最终目的是服务于进程间通信

The goal of the Message-Passing Interface, simply stated, is to develop a widely used standard for writing message-passing programs. As such the interface should establish a practical, portable, efficient, and flexible standard for message passing.

MPI 介绍

MPI 的目标

- 高通信性能，高可移植性，强大的功能
- Practical, Portable, Efficient, Flexible

MPI 标准和 MPI 实现

- 1994 年 MPI-1.0; 1998 年 MPI-2.0; 2012年 MPI-3.0; MPI-3.1 (2015); MPI-4.0 (2021) ; MPI-5.0 (Current Efforts)
- 支持 C 和 Fortran (目前以 Fortran 90 为主)
- MPI 实现 (免费版) : **MPICH** 和 OpenMPI
- MPI 实现 (商业版) : Intel MPI, IBM MPI, HP-MPI, MS-MPI, ...
- 所有版本都遵循 MPI 标准, 可以不加修改地运行

MPICH 下载与安装

MPICH

- <http://www.mpich.org/>
- 支持 C/C++/Fortran
- 支持编译器：GNU, Clang, Intel, PGI, XL (IBM), Sun Studio
- 安装： [apt](#) (Ubuntu 4.0.2), [yum](#) (Fedora 4.0.2)
- 手工安装：下载源代码 (4.1.1) 后编译

第一个 MPI 程序

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

int main(int argc, char * argv[])
{
    MPI_Init(&argc, &argv);

    printf("Hello World!\n");

    MPI_Finalize();
    return 0;
}
```

```
mpicc -O2 -o hello MPI_hello.c
mpirun -np 4 hello
```

程序分析

```
#include <mpi.h>
```

MPICH 相对于 C 语言的头文件

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

MPI 程序的开始

```
MPI_Finalize()
```

MPI 程序的结束。

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

第二个 MPI 程序

MPI_hello.c

```
#include <mpi.h>
#include <stdio.h>
#include <math.h>
int main(int argc, char * argv[])
{
    int myid, np;
    int namelen;
    char proc_name[MPI_MAX_PROCESSOR_NAME];
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    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);
    MPI_Finalize();
}
```

程序分析

- `MPI_MAX_PROCESSOR_NAME`

预定义的宏，即 MPI 所允许的机器名字的最大长度

- `MPI_Comm_rank`

函数，返回到本进程的进程号

- `MPI_Comm_size`

函数，返回所有参加运算的进程的个数

- `MPI_Get_processor_name`

函数，返回运行本进程所在的结点的主机名

MPI 通信器

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)
- 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** : 有单个进程独自构成, 仅包含自己

MPI 进程

MPI 进程

- MPI 程序中一个独立参与通信的个体；
- MPI 进程组：由部分或全部进程构成的有序集合；
- 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
- 进程号取值范围为 $0, \dots, np-1$
- 空进程：MPI_PROC_NULL
- 与空进程通信时不做任何操作

程序运行结果

□ 在单个结点(c22)上，开 4 个进程的运行结果

Hello, I am Proc. 1 of 4 on c22

Hello, I am Proc. 0 of 4 on c22

Hello, I am Proc. 2 of 4 on c22

Hello, I am Proc. 3 of 4 on c22

□ 在四个结点上，开 4 个进程的运行结果

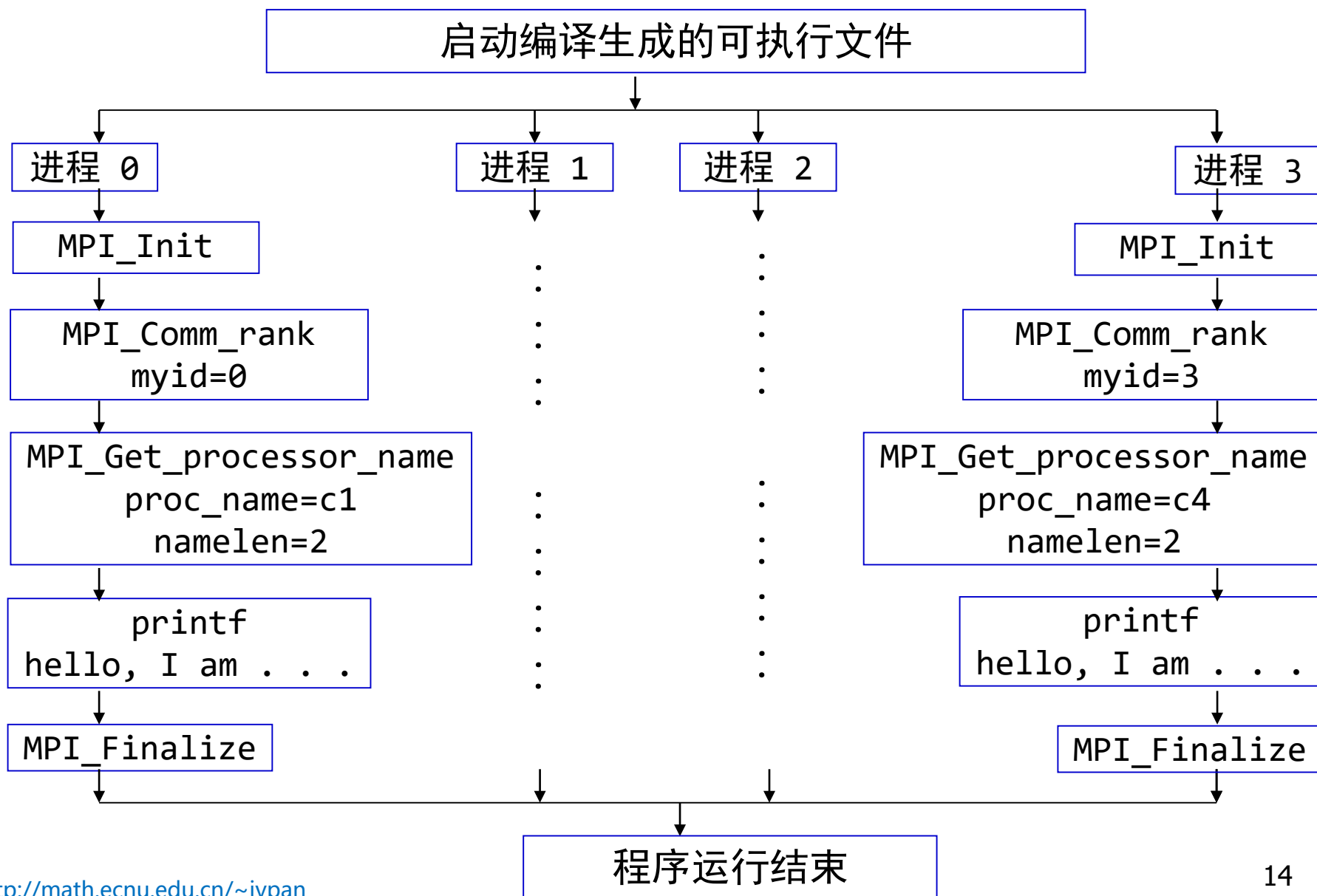
Hello, I am Proc. 1 of 4 on c1

Hello, I am Proc. 0 of 4 on c2

Hello, I am Proc. 2 of 4 on c4

Hello, I am Proc. 3 of 4 on c3

程序运行过程



MPI 编程的一些惯例

- MPI 的所有常量、变量与函数/过程均以 `MPI_` 开头
- MPI 的 C 语言接口为函数，FORTRAN 接口为 SUBROUTINE，且对应接口的名称相同
- 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
- 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 编程的一些惯例

- 由于 C 语言的函数调用机制是值传递，所以 MPI 的所有 C 函数中的输出参数用的都是指针
- MPI 是按进程组 (Process Group) 方式工作：
所有 MPI 进程在开始时均是在通信器 `MPI_COMM_WORLD` 所对应的进程组中工作，之后用户可以根据自己的需要，建立其它的进程组
- 所有 MPI 的通信一定要在通信器中进行
- FORTRAN 的数组下标缺省以 1 开始，而 C 语言的数组是以 0 开始

2

MPI 数据类型

- 1 MPI 安装、编译与运行
- 2 MPI 数据类型
- 3 消息发送和接收
- 4 MPI 一些常用函数

MPI 数据类型

MPI 除了提供函数外，还定义了一组常量和数据类型

- MPI 常量命名规则：全部大写
- MPI 变量数据类型命名规则：
以 `MPI_` 开头，后面跟 C 语言或 FORTRAN 原始数据类型名称
- MPI 数据类型包含 原始数据类型 和 自定义数据类型

† 注意：MPI 的数据类型只用于消息传递！

MPI 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	

MPI 数据类型主要用于数据的传递。

3

消息发送和接收

- 1 MPI 安装、编译与运行
- 2 MPI 数据类型
- 3 消息发送和接收
- 4 MPI 一些常用函数

MPI 消息

MPI 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

MPI 常用的六个接口

- ❑ MPI_Init
- ❑ MPI_Finalize
- ❑ MPI_Comm_rank
- ❑ MPI_Comm_size

- ❑ MPI_Send
- ❑ MPI_Recv

MPI_INIT

MPI_Init: MPI 初始化

参数	无
C	<code>int MPI_Init(int *argc, char *argv[])</code>
F77	<code>MPI_INIT(IERR)</code> <code>INTEGER IERR</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（除 `MPI_Initialized`）之前被调用，并且在一个 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>
F77	<code>MPI_FINALIZE(IERR)</code> <code>INTEGER IERR</code>

该函数清除 MPI 环境的所有状态。即一但它被调用，所有MPI 函数都不能再调用，其中包括 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>	
F77	<pre>MPI_COMM_RANK(COMM, RANK, IERR) INTEGER COMM, RANK, IERR</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	int MPI_Comm_size(MPI_Comm comm, int *size)		
F77	MPI_COMM_SIZE(COMM, SIZE, IERR) INTEGER COMM, SIZE, IERR		

该函数返回指定通信器所包含的进程数

MPI_SEND 点对点通信

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>		
F77	<pre>MPI_SEND(BUF,COUNT,DATATYPE,DEST,TAG,COMM, IERR) <type> BUF(*) INTEGER COUNT, DATATYPE, DEST, TAG, COMM, IERR</pre>		

MPI_SEND 点对点通信

`MPI_Send(buf, count, datatype, dest, tag, comm)`

- 阻塞型消息发送接口
- **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**。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。其中 **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 点对点通信

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

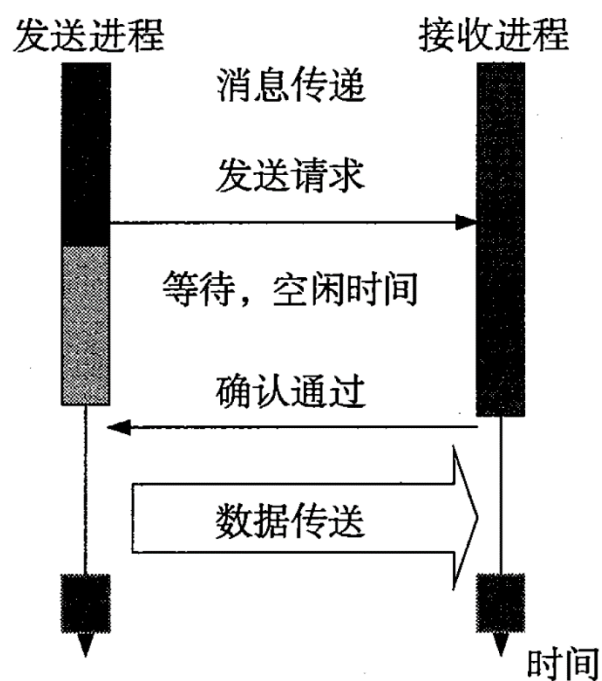
参数	OUT	buf	接收消息数据的首地址
	IN	count	接收数据的最大个数
	IN	datatype	接收数据的数据类型
	IN	source	发送消息的进程的标识号
	IN	tag	消息标签
	IN	comm	通信器
	OUT	status	返回状态
C	<pre>int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)</pre>		
F77	<pre>MPI_RECV(BUF,COUNT,DATATYPE,SOURCE,TAG,COMM,STATUS,IERR) <type> BUF(*) INTEGER COUNT, DATATYPE, SOURCE, TAG, COMM, IERR, STATUS(MPI_STATUS_SIZE)</pre>		

MPI_RECV 点对点通信

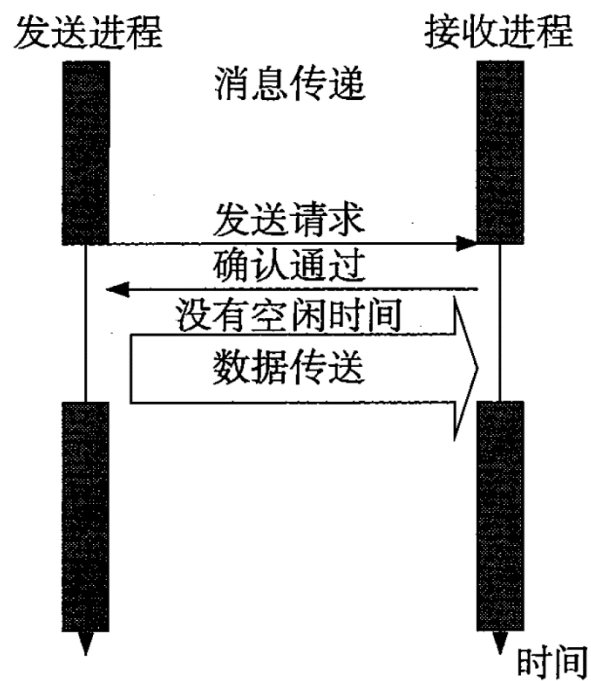
`MPI_Recv(buf, count, datatype, source, tag, comm, status)`

- 阻塞型消息接收接口
- 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`。这里 `source` 的取值范围为 `0 ~ np-1`，或 `MPI_ANY_SOURCE`，或 `MPI_PROC_NULL`，`tag` 的取值为 `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，在 FORTRAN 中是用数组定义的，其中包括 `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，需要用到后面给出的函数 `MPI_GET_COUNT`

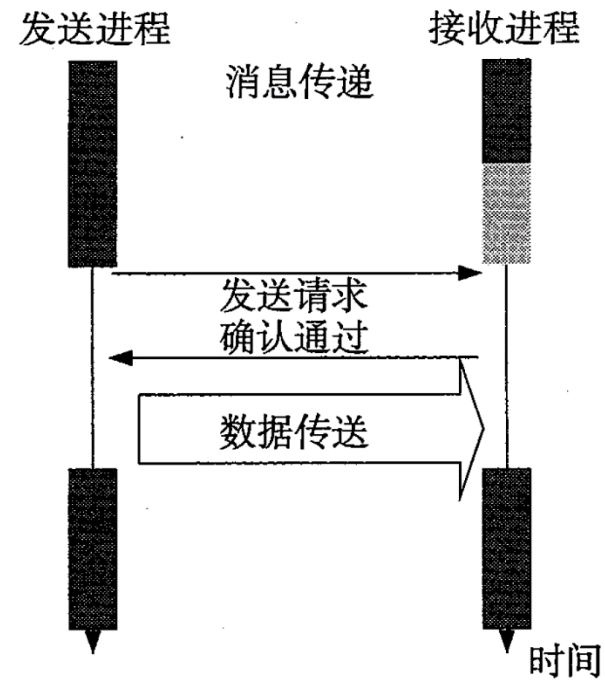
阻塞通信：握手模式



a)



b)



c)

MPI 程序示例

例：将每个进程中某个数据发送给下一个进程，并从前一个进程接收一个数据，即 0 号进程给 1 号进程发送一个数据，并从 np-1 号进程接收一个数据，1 号进程从 0 号进程接收一个数据，并向 2 号进程发送一个数据，以此类推。

- **实现方式一：** 0 号进程先发送后接收，其它进程先接收后发送

MPI_Sendrecv_01.c

MPI_Sendrecv_02.c

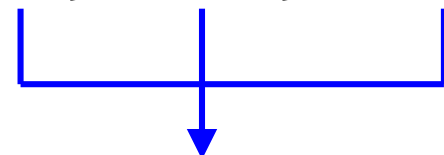
† 在使用阻塞型函数传递消息时要避免死锁！

- **实现方式二：** 奇数号进程先发送后接收，偶数号进程先接收后发送

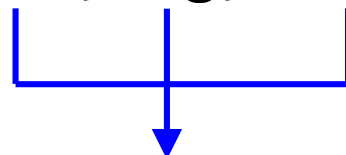
留作课堂练习

MPI 发送接收

`MPI_Send(buf, count, datatype, dest, tag, comm)`

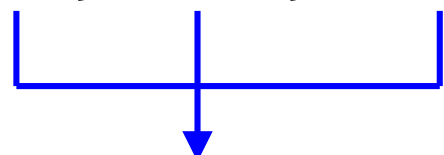


消息数据

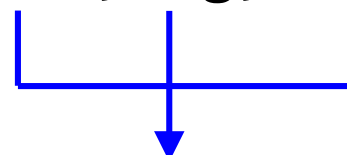


消息信封

`MPI_Recv(buf, count, datatype, source, tag, comm, status)`



消息数据



消息信封

✦ 发送数据类型、通信函数
中的数据类型、接收的数
据类型要一致！

C: 结构

status.MPI_SOURCE
status.MPI_TAG
status.MPI_ERROR

FORTTRAN: 数组

status(MPI_SOURCE)
status(MPI_TAG)
status(MPI_ERROR)

发送与接收组合进行

**MPI_Sendrecv(sendbuf, sendcount, sendtype, dest, sendtag,
recvbuf, recvcount, recvtype, source, recvtag, comm, status)**

参数	略
C	<pre>int MPI_Sendrecv(void *sendbuf, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void *recvbuf, int recvcount, MPI_Datatype recvtype, int source, int recvtag, MPI_Comm comm, MPI_Status *status)</pre>
F77	<pre>MPI_SENDRCV(SENDBUF, SENDCOUNT, SENDTYPE, DEST, SENDTAG, RECVBUF, RECVCOUNT, RECVTYPE, SOURCE, RECVTAG, COMM, STATUS, IERR)</pre> <type> SENDBUF(*), RECVBUF(*) INTEGER SENDCOUNT, SENDTYPE, DEST, SENDTAG, RECVCOUNT, RECVTYPE, SOURCE, RECVTAG, COMM, IERR, STATUS(MPI_STATUS_SIZE)

MPI_SENDRECV

```
MPI_Sendrecv(sendbuf, sendcount, sendtype,  
              dest, sendtag,  
              recvbuf, recvcount, recvtype,  
              source, recvtag, comm, status)
```

- 这是发送消息和接收消息组合在一起的一个函数，好处是不用考虑先发送还是先接收消息，从而可以避免消息传递过程中的死锁
- **sendbuf** 和 **recvbuf** 必须指向不同的缓冲区

[MPI_Sendrecv_03.c](#)

MPI_SENDRECV_REPLACE

```
MPI_Sendrecv_replace(buf, datatype, dest,  
                      sendtag, source, recvtag,  
                      comm, status)
```

- 功能与 MPI_SENDRECV 类似，但收发消息使用的是同一个缓冲区

MPI_Sendrecv_04.c

MPI 程序示例

例：计算 π 的值。

$$\pi = \int_0^1 \frac{4}{x^2 + 1} dx \approx h \sum_{i=1}^n f(x_i)$$

- 采用等步长中矩形公式，其中 n 为积分区间数， $h = 1/n$ 为步长， $x_i = (i + 0.5)h$ 为积分区间的中点
- 假设用 p 的进程计算，各自计算其中的一部分，然后再将结果加起来。

MPI_pi.c

4

MPI 一些常用函数

- 1 MPI 安装、编译与运行
- 2 MPI 数据类型
- 3 消息发送和接收
- 4 MPI 一些常用函数

MPI_GET_COUNT

MPI_GET_COUNT(status, datatype, count)

参数	IN status	接收操作的返回状态
	IN datatype	接收操作使用的数据类型
	OUT count	接收到的数据个数
C	<pre>int MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count)</pre>	
F77	<pre>MPI_GET_COUNT(STATUS,DATATYPE,COUNT,IERR) INTEGER DATATYPE,COUNT,IERR, STATUS(MPI_STATUS_SIZE)</pre>	

查询实际接收到的数据信息

MPI 异常终止

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)		
F77	MPI_ABORT(COMM, ERRORCODE, IERR)		
	INTEGER	COMM, ERRORCODE, IERR	

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 计时函数

MPI_WTIME()

参数	无
C	double MPI_Wtime(void)
F77	DOUBLE PRECISION MPI_WTIME()

该函数返回当前的墙钟时间

MPI 计时函数

MPI_WTICK()

参数	无
C	double MPI_Wtick(void)
F77	DOUBLE PRECISION MPI_WTICK()

该函数返回 MPI_WTIME 的时钟精度，单位为秒，值为 10^{-9}

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	int MPI_Get_processor_name(char *name, int *namelen)	
F77	MPI_GET_PROCESSOR_NAME(NAME, NAMELEN, IERR) CHARACTER(*) NAME INTEGER NAMELEN, IERR	

该函数返回进程所在结点的主机名

获取 MPI 版本号

MPI_MPI_GET_VERSION(version, subver)

参数	OUT version	主版本号
	OUT subver	次版本号
C	<pre>int MPI_Get_version(int *version, int *subver)</pre>	
F77	<pre>MPI_GET_VERSION(VERSION, SUBVER, IERR) INTEGER VERSION, SUBVER, IERR</pre>	

该函数返回 MPI 的版本号