并行计算

第三讲

MPI 编程基础

主要内容

- 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 提供库函数供 C 语言调用

MPI 编程基本概念

- 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

进程个数由用户在递交并行任务时指定

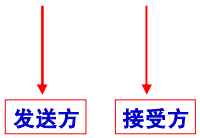
MPI 进程与通信器

- MPI 通信器 （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

- 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

mpi.h 是 MPI 相对于C语言的头文件

```
// hello_world.c
#include "mpi.h"
#include <stdio.h>

int main(argc, argv)
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();
}
```

MPI 程序分析

- 所有包含 MPI 调用的程序必须包含 MPI 头文件
- MPI_MAX_PROCESSOR_NAME 是 MPI 预定义的宏，即 MPI 所允许的机器名字的最大长度
- 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
- MPI_Comm_rank 得到本进程的进程号
- MPI_Comm_size 得到所有参加运算的进程的个数
- MPI_Get_processor_name 得到运行本进程所在的结点的主机名
- 进程号取值范围为 0, …, np-1

MPI 程序分析

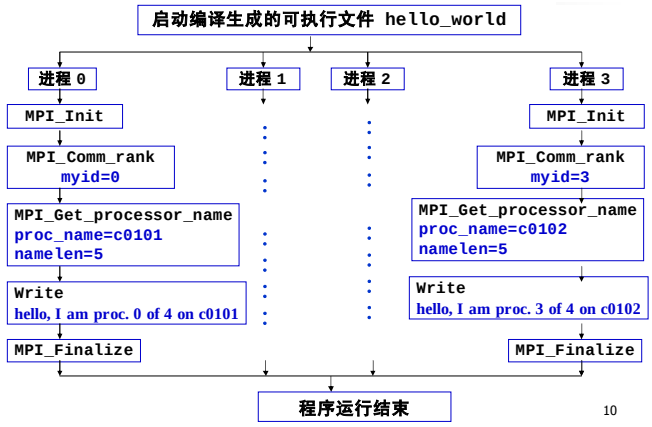
u 在单个结点(c0101)上，开 4 个进程的运行结果

```
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
```

u 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
- 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 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
- C 语言的数组的下标是从 0 开始的

MPI 程序的编译

q MPI 程序的编译

```
mpicc -o hello_world  hello_world.c
```

- -o 用于指定编译生成的可执行文件的文件名

q MPI 程序的运行

- mpirun （old，使用 rsh 启动 MPI 进程）
- mpiexec （new，使用 ssh 启动 MPI 进程，更安全）

具体使用方法见课程主页

由于系里的机群使用的是联想开发的系统管理和任务调度软件，提交任务需使用专门的命令，将在后面详细讲解。

MPI 基本数据类型

u MPI 定义了一些基本数据类型
主要用于消息传递

n 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

注：全部为大写！

MPI 数据类型 与 C 语言数据类型

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 常用的接口

- MPI_Init
- MPI_Finalize
- MPI_Comm_rank
- MPI_Comm_size
- MPI_Get_processor_name

- MPI_Send
- MPI_Recv

- MPI_Sendrecv
- MPI_Sendrecv_replace

MPI_Init

u MPI_Init：MPI 初始化

参数	无
C	int MPI_Init(int *argc, char ***argv)

l 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_Initilized）之前被调用，并且在一个 MPI 程序中，只能被调用一次

MPI_Finalize

u MPI_Finalize：结束 MPI 系统

参数	无
C	int MPI_Finalize(void)

l 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

u 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	int MPI_Comm_rank(MPI_Comm comm, int *rank)

l 该函数返回本进程在指定通信器中的进程号

MPI_Comm_size

u 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)

l 该函数返回指定通信器所包含的进程数

MPI_Get_processor_name

u 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)

l 该函数返回进程所在结点的主机名

MPI 点对点通信

n MPI 通信分类

l 点对点通信：两个进程之间的数据传递
l 聚合通信：多个进程之间的通信

n MPI 点对点通信

l 点对点通信是 MPI 通信机制的基础
l 点对点通信分：阻塞型和非阻塞型

阻塞型通信

n 阻塞型 (blocking) 通信

l 阻塞型通信函数需要等待指定的操作实际完成，或所涉及的数据被 MPI 系统安全备份后才返回
Ø 阻塞型通信是非局部操作，它的完成可能涉及其它进程
Ø 典型的阻塞型通信函数：MPI_SEND 和 MPI_RECV

l 非阻塞型通信（略）

MPI_Send 点对点通信

u 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	int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

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

MPI_Recv点对点通信

u **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	int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag,MPI_Comm comm,MPI_Status *status)

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，其中包括 MPI_SOURCE，MPI_TAG 和 MPI_ERROR。

MPI 程序示例

例：每个进程给下一个进程发送一个数据，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

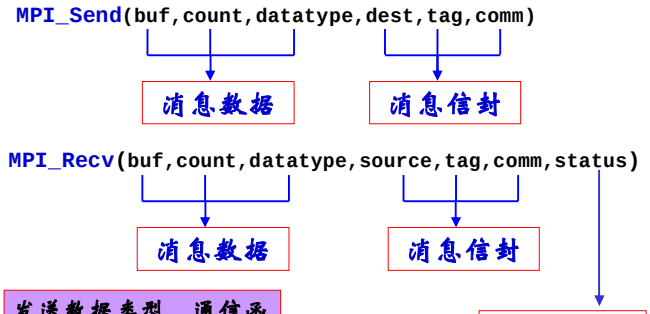
例：ex4sendrecv01.c

在使用阻塞型函数传递消息时要避免死锁！

例：ex4sendrecv02.c

- 实现方式二：作为上机作业
  奇数号进程先发送后接收，偶数号进程先接收后发送

MPI 发送接收



MPI_Sendrecv

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

参数	略
C	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)

- 这是发送和接收组合在一起的函数，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

例：ex4sendrecv03.c

MPI_Sendrecv_replace

u **MPI_Sendrecv_replace**(buf, count, datatype, dest, sendtag, source, recvtag, comm, status)

参数	略
C	int MPI_Sendrecv_replace(void *buf, int count, MPI_Datatype datatype, int dest, int sendtag, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

- 功能与 MPI_SENDRECV 类似，但收发消息使用的是同一个缓冲区

例：ex4sendrecv04.c

上机作业

- 熟悉并行程序的编译和运行
  (1) 从课程主页上下载 hello_world.c 到你的主目录中
  (2) 编译该并行程序：
  ```
  mpicc -O2 -o hello_world hello_world.c
  ```
  (3) 运行生成的可执行文件 hello_world
  ```
  mpirun -np 4 hello_world
  ```
- 每个进程给下一个进程发送一个数据，并从前一个进程接收一个数据。（要求写入实验报告纸）
  (1) 奇数号进程先发送后接收，偶数号进程先接收后发送
  (2) 使用 4 个进程
  (3) 使用 MPI_Send 和 MPI_Recv 实现
  (4) 文件取名为 myex4sendrecv.c
  (5) 参考 ex4sendrecv01.c