实验七：MPI并行编程

郑海刚

HITSZ 实验与创新实践教育中心
Education Center of Experiments and Innovations, HITSZ

本讲概述

- 主要内容
  - MPI介绍
  - MPICH安装使用
  - MPI库函数介绍
  - MPI点对点通信
  - GEMM的MPI实现（实验内容）

多进程实现gemm

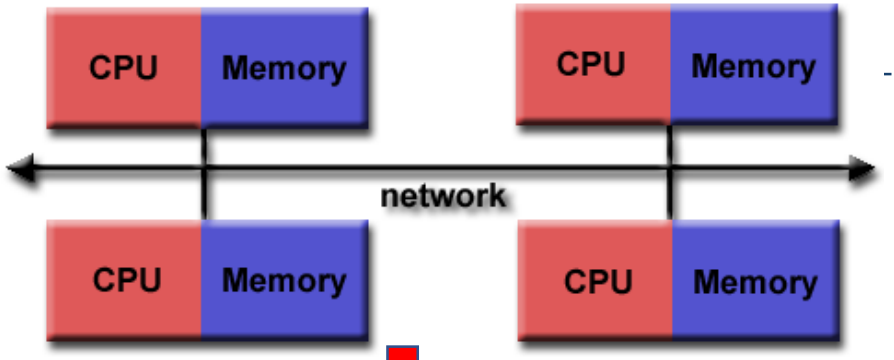
- 矩阵A、B只读不写，每个进程可数据独享
- 矩阵C需要汇总结果，多进程实现，需要进程间通信
- 进程间通信（inter-process communication，IPC）方式有：
  - 文件、套接字、管道等
- 如果多台计算机共同完成大规模矩阵的GEMM？
  - 机器之间要互联
  - 不同服务器进程间的通信：Remote procedure call（RPC）、套接字等

消息传递接口

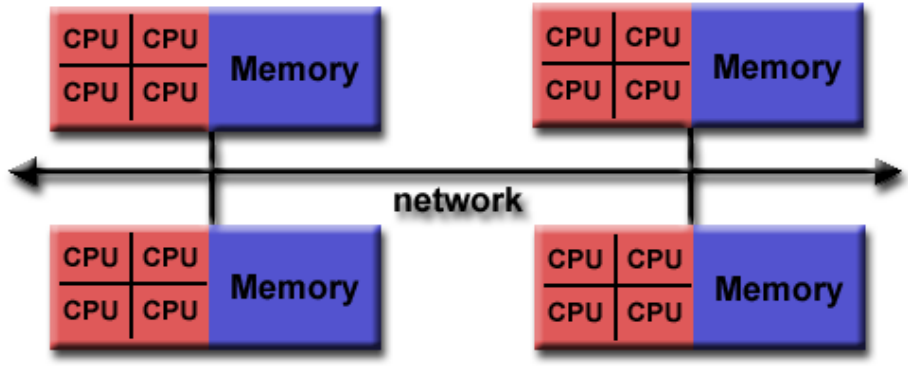
- Message Passing Interface （MPI）
  - 并行计算架构的消息传递标准
  - 定义了一组库函数：支持C/C++、Fortran
  - 主要用于分布式内存（集群系统），也能用于共享内存（单机）

MPI编程模型

- 初始目的是用于分布式并行系统（集群）
- 早期：单核CPU



- 多核时代，混合架构
- 主流的用法：MPI+OpenMP混合编程
  - 多机之间MPI通信，单台节点上使用OpenMP



MPI标准

- mpi-forum
- MPI主要版本
  - MPI-1.0 1994
  - MPI-2.0 1996
  - MPI-3.0 2012
  - MPI-4.0 2021

MPI的实现

- 开源实现
  - MPICH： www.mpich.org
  - Open MPI： www.open-mpi.org（容易与OpenMP混淆）
- 商业实现：Intel MPI，HP-MPI，MS-MPI等

MPI使用：相关命令

- mpich安装：sudo apt install mpich mpich-doc -y
- man mpirun
- man MPI_Init

```
$ ~ »mpi77
mpic++          mpicxx          mpif77          mpifort.mpich
mpicc           mpicxx.mpich    mpif77.mpich    mpirun
mpiCC           mpiexec         mpif90          mpirun.mpich
mpicc.mpich     mpiexec.hydra   mpif90.mpich    mpivars
mpichversion    mpiexec.mpich   mpifort
```

MPI : hello word

- lab7/hellow.c
  - mpicc hellow.c -o hellow
  - ./hellow 可直接运行
  - mpirun -n 4 ./hellow 启动多个进程

```c
/*
 * Copyright (C) by Argonne National Laboratory
 *     See COPYRIGHT in top-level directory
 */

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

int main(int argc, char *argv[])
{
    int rank;
    int size;

    MPI_Init(0, 0);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello world from process %d of %d\n", rank, size);
    MPI_Finalize();
    return 0;
}
```

```
$ hpc/lab7 »mpirun -n 4 ./hellow
Hello world from process 1 of 4
Hello world from process 2 of 4
Hello world from process 0 of 4
Hello world from process 3 of 4
$ hpc/lab7 »
```

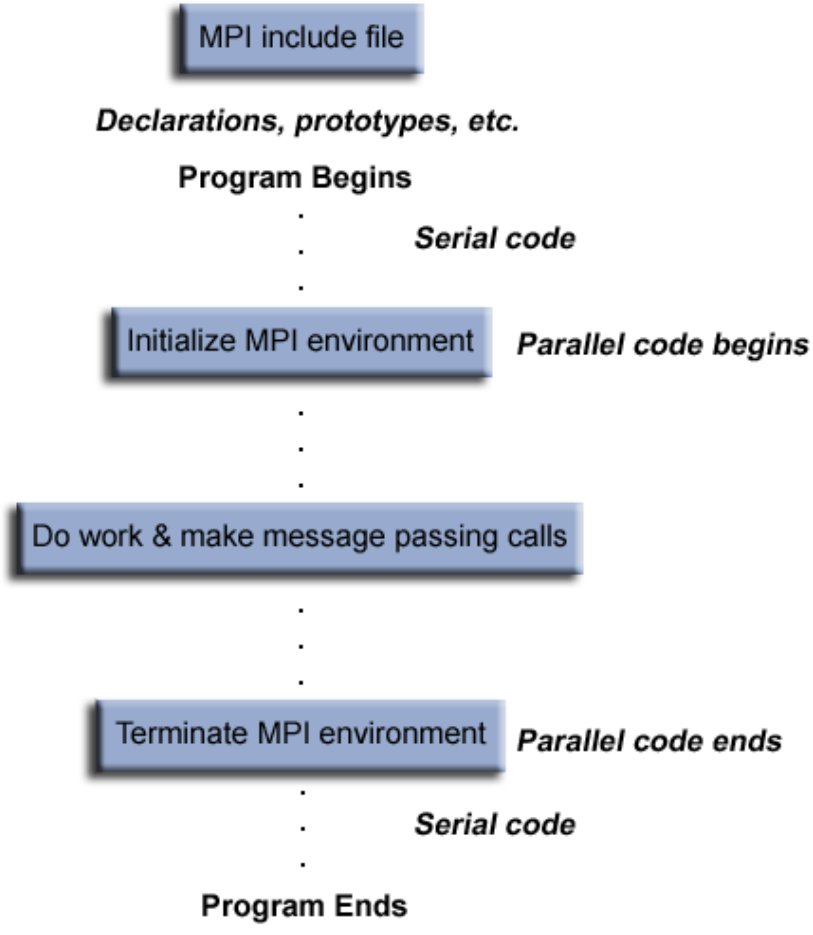
MPI常用命令

- 主要两类：编译和启动
- 实验只需到：mpicc和mpirun
- mpicc 实际是gcc的封装
  - --verbose选项

类别	命令	功能
编译	mpicc	c语言编译
	mpic++	c++编译
	mpicxx	c++编译，同mpic++
	mpif77	fortran 77编译
	mpiff90	Fortran 90编译
启动任务	mpiexec	启动任务
	mpirun	启动任务，同mpiexec

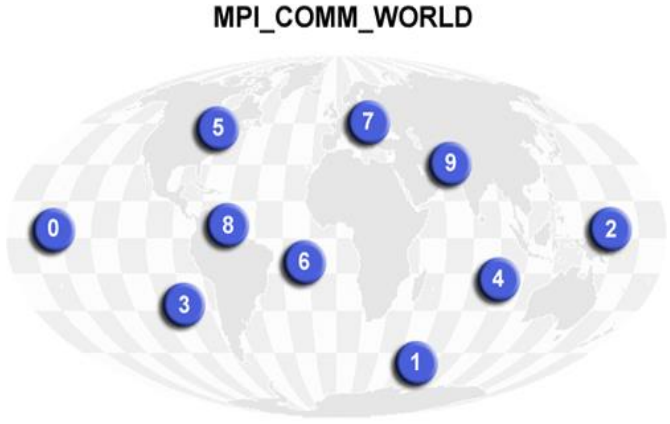
MPI 程序结构

- 头文件：mpi.h
- 变量声明
- 初始化MPI环境：MPI_Init()
- 并行化、消息传递调用
- 终止MPI环境：MPI_Finalize()
- 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_Init、MPI_Finalize

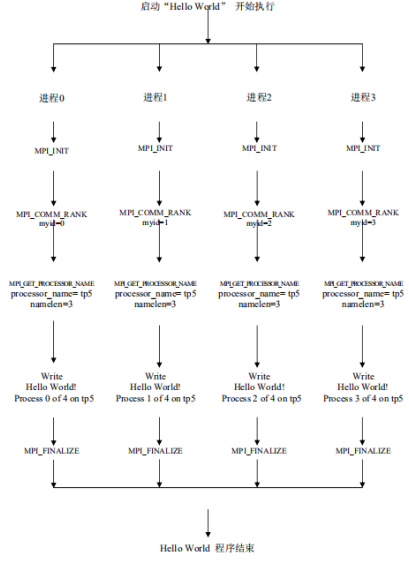
- int MPI_Init(int *argc, char ***argv)：
  - 第一个调用的MPI函数，只能调用一次
  - 环境初始化，创建全局变量、创建通信器(默认是MPI_COMM_WORLD)
  - 通信子中的每个进程都有一个ID，即秩（Rank）
- int MPI_Finalize(void)：
  - 结束MPI的运行环境
  - 必须由调用MPI_Init的进程调用



MPI_Init 并不创建进程

- lab7/nompi-hello.c
  - mpicc也能直接编译运行，也是多进程程
- MPI_init只是初始化MPI的运行环境
- MPI_Finalize并不是结束进程，只是结束MPI的运行环境
  - lab7/hellow.c：取消两行printf的注释再测试
  - MPI环境前后的代码仍是多进程运行

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello world\n");
    return 0;
}
```

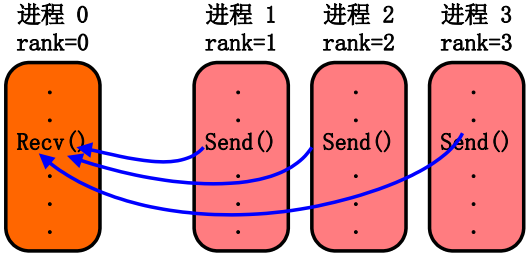


MPI 常量

- “MPI_” 开头，全部大写
- MPI's defined constants

点对点通信

- lab7/comm_demo.c




```c
#include <stdio.h>
#include <string.h>
#include "mpi.h"

int main( int argc, char *argv[] )
{
    int numprocs, myid, source;
    char message[100];
    char myid_str[2];
    MPI_Status status;

    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &myid);
    MPI_Comm_size( MPI_COMM_WORLD, &numprocs);
    if (myid != 0)  {
        strcpy(message, "Hello World!");
        sprintf(myid_str,"%d",myid);
        strcat(message,myid_str);
        MPI_Send(message, strlen(message)+1,
                MPI_CHAR, 0, 99, MPI_COMM_WORLD);
    } else {
        for (source = 1; source < numprocs; source++) {
            MPI_Recv(message, 100, MPI_CHAR, source,
                    99, MPI_COMM_WORLD, &status);
            printf("%s\n", message);
         }
    }
    MPI_Finalize();
    return 0;
}
```

消息发送

- MPI_Send
  - 核心问题：从哪取数据发送，发送多少，发送到哪里
  - buf：代发送数据的缓冲区
  - count：发送的元素个数
  - datataype：MPI_Datatype，与C类型对应关系
  - dest：接收端进程的rank
  - tag：给信息打标签
  - comm：接收端所在通信器

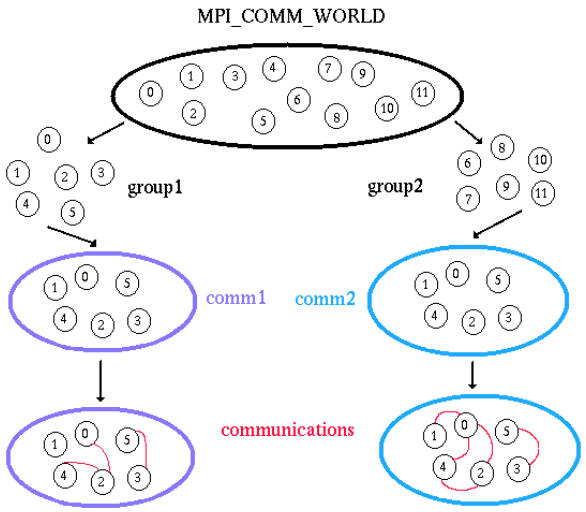
```
NAME
       MPI_Send -  Performs a blocking send

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

INPUT PARAMETERS
       buf    - initial address of send buffer (choice)
       count  - number of elements in send buffer (nonnegative integer)
       datatype
              - datatype of each send buffer element (handle)
       dest   - rank of destination (integer)
       tag    - message tag (integer)
       comm   - communicator (handle)
```

通信器

- Communicator：也叫通信域，进程通信的范围，包含上下文和进程组
  - MPI_COMM_WORLD 是MPI预定义的全局通信器
  - MPI_COMM_SELF 是MPI预定义的只包含各个进程自己的通信器

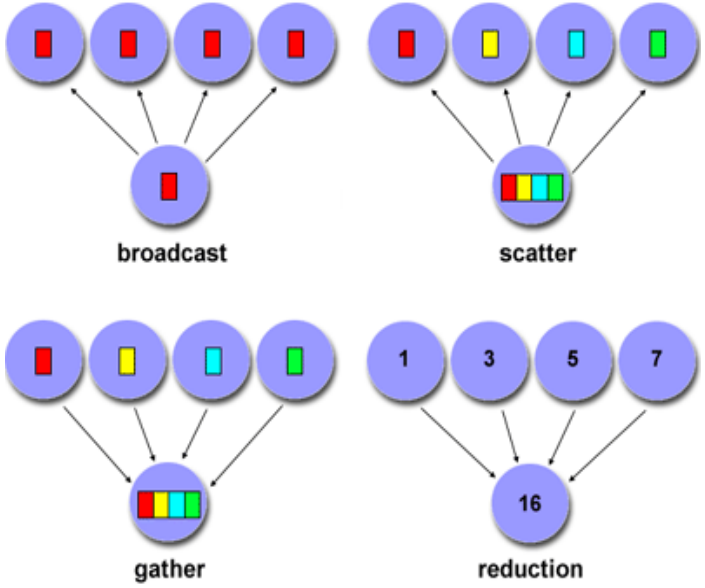
多通信器

- 进程划分多个组（group）,一个组是一个通信器
- 比如总共12个进程，6个进程找最大的数，6个进程找最小的数
- 通信器之间也可以通信
- 一个进程可以属于多个通信器



进阶

- 集合通信：
  - broadcast：将数据复制发送出去
  - scatter：将数据拆分为多段发送出去
  - gather：接收不同发送者的数据段拼接
  - reduction：接收不同发送者的数据累加
- 阻塞、非阻塞等的理解



主要参考材料

- Lawrence Livermore National Laboratory： MPI
- Wikipedia: Message Passing Interface
- mpitutorial 中文版： https://mpitutorial.com/tutorials/