并行复习6 MPI编程

> ## MPI编程
>
> **MPI基本原语、阻塞通信、编程模型(对等\主从)、组通信、非阻塞通信原理、混合编程原理**
>
> 6个基本原语
>
> 非阻塞考的少知道基础的就可以 写法是I send i receive
>
> 控制、查询的方式 包括 wait cast cancel 知道这些函数可以控制
>
> 组通信是关键 一对对是什么功能
>
> 混合编程的三种方法(SINGLE不是

MPI的典型使用方式是在一台或多台计算机上启动多个MPI进程，这些进程在不同的计算节点上运行，并通过MPI的通信原语进行数据交换。MPI标准定义了一组函数和语法，程序员可以使用这些函数在不同的进程之间进行通信和同步，以实现并行计算。

mpi编程**不会有数据竞争**，但可能出现通信错误

MPI基本原语

```
MPI_Init(&argc,&argv); //初始化MPI环境
MPI_Finalize(); //终止MPI环境
MPI_Comm_size(MPI_COMM_WORLD,&myid);  //报告进程数
MPI_Comm_rank(MPI_COMM_WORLD,&numprocs);  //报告进程编号
MPI_Send();
MPI_Recv();
```

MPI程序基本结构

```
#include <mpi.h>
…
int main(int argc, char *argv[]) {
…
/* 这部分不应有MPI函数调用 */
MPI_Init(&argc, &argv);
```

```
… /* MPI程序主体 */
MPI_Finalize();
/* 这部分不应有MPI函数调用 */
return 0;
}
```

MPI消息传递

阻塞通信

阻塞通信是一种等待消息传输完成后才继续执行的通信方式。

- 阻塞发送

MPI_Send

- 阻塞接收

MPI_Recv

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

int main() {
    int rank, size;
    int data = 100;

    MPI_Init(NULL, NULL);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);

    if (size < 2) {
        printf("This program requires at least 2 processes.\n");
        MPI_Abort(MPI_COMM_WORLD, 1);
    }

    if (rank == 0) {
        // 发送数据到进程1
        MPI_Send(&data, 1, MPI_INT, 1, 0, MPI_COMM_WORLD);
        printf("Process %d sent data %d to Process 1.\n", rank, data);
    } else if (rank == 1) {
        // 接收从进程0发送来的数据
        MPI_Recv(&data, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        printf("Process %d received data %d from Process 0.\n", rank, data);
    }

```

```
    MPI_Finalize();
    return 0;
}

```

进程0发送数据到进程1。MPI_Send函数在进程0中发送数据，然后进程1中的MPI_Recv函数接收数据。

发送和接收的操作是阻塞的，进程0将等待直到数据被成功发送，进程1将等待直到数据被成功接收。

阻塞通信模式

- MPI有一个默认缓冲区, 执行Send时, 先将数据放入缓冲区, 只要缓冲区中的数据被发送完毕, 就认为发送完毕, 可以继续执行其余代码
- 接收方一定是等到数据全部接收完毕才执行其余代码
- 缺点是可能导致死锁

消息传递两种编程模型

- 对等式：地位平等，功能相近
- 主从式：地位不同，功能不同

组通信

- 一个进程组（通信域）内的所有进程同时参加通信
- 所有参与进程的函数调用形式完全相同
- 三个功能：通信，同步，计算
- 在组通信中不需要消息标志参数
- 操作是成对的：互为逆操作

广播和归约

- one to all　broadcast 广播

一个进程向其他所有进程发送相同数据

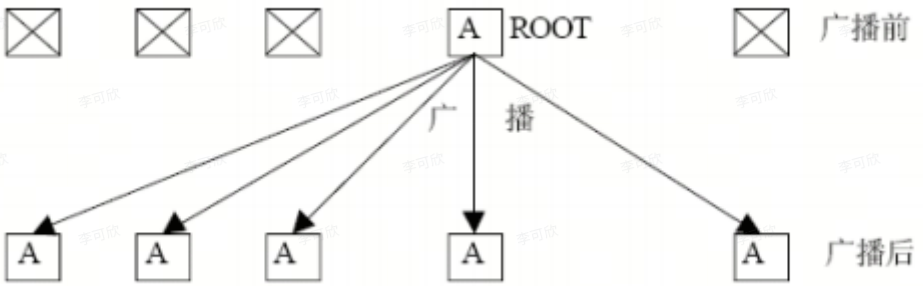
- all to one　reduction 归约

其他所有进程的相同数据经过计算得到一份数据，传送到目的进程

MPI广播

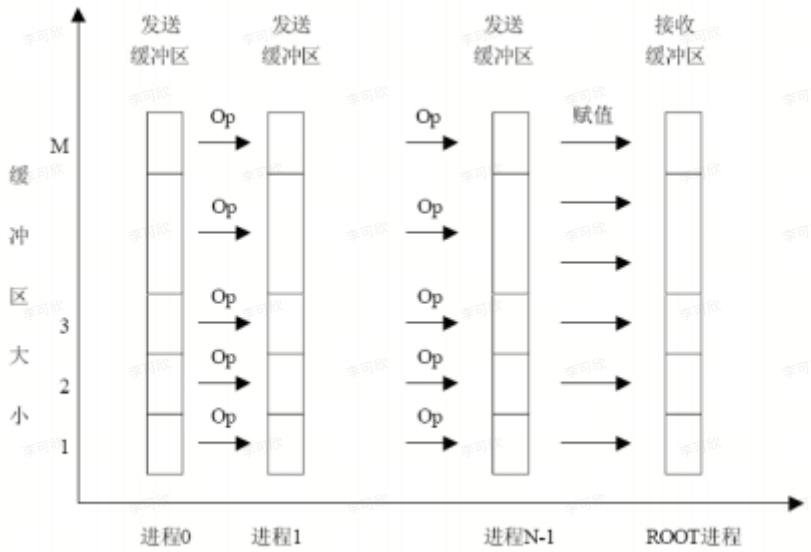
```
int MPI_Bcast(void* buffer,int count,MPI_Datatype
```

```
                datatype,int root, MPI_Comm comm)
root：发送广播的进程
count：待广播的数据个数
```



MPI归约

```
int MPI_Reduce(void* sendbuf, void* recvbuf, int count,
               PI_Datatype datatype,MPI_Op op, int root,
                MPI_Comm comm)
```



非阻塞通信

- 发送和接收数据的过程中，CPU可以执行别的事情
- 查询命令MPI_Wait查看是否完成

混合编程

- MPI描述了进程（独立地址空间）间并行
- *线程*并行则是提供了一个进程内的共享内存模型
- 多核集群的常见编程方法
  - 纯MPI
    - 节点内和跨节点都采用MPI实现进程并行
    - 节点内MPI内部是采用共享内存来通信的
  - MPI + OpenMP
    - 节点内使用OpenMP，跨节点使用MPI
  - MPI + Pthreads
    - 节点内使用Pthreads，跨节点使用MPI
- 后两种方法被称为“混合编程”

MPI+Threads

```
//SINGLE模式并不算混合编程
MPI_THREAD_SINGLE:默认应用中只有一个线程
MPI_THREAD_FUNNELED：多线程，但只有主线程会进行MPI调用（调用MPI_Init_thread的那个线程）
MPI_THREAD_SERIALIZED：多线程，但同时只有一个线程会进行MPI调用
MPI_THREAD_MULTIPLE：多线程，且任何线程任何时候都会进行MPI调用（有一些限制避免竞争条件）
```

```
MPI定义了一个MPI_Init的替代API
MPI_Init_thread(requested,provided)
requested：应用给出它希望的级别；MPI_THREAD_XXXXX
 provided：MPI实现返回它支持的级别
```