MPI并行编程入门培训

国家超级计算天津中心

目标
TARGET

- 了解MPI相关基础知识和概念
- 了解MPI并行程序基础框架
- 了解MPI并行程序编程思想
- 学会将简单（实例）串行程序改为MPI并行程序
- 了解并行程序的性能优势

基于MPI，实现将简单串行程序修改为并行程序

目录 Contents

1.并行思想

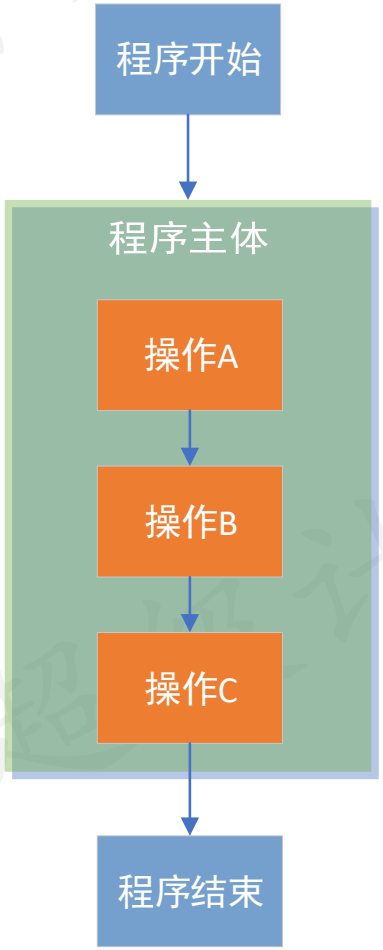
为什么要使用并行

- 逻辑简单，可读性较强；
- 低效、耗时；

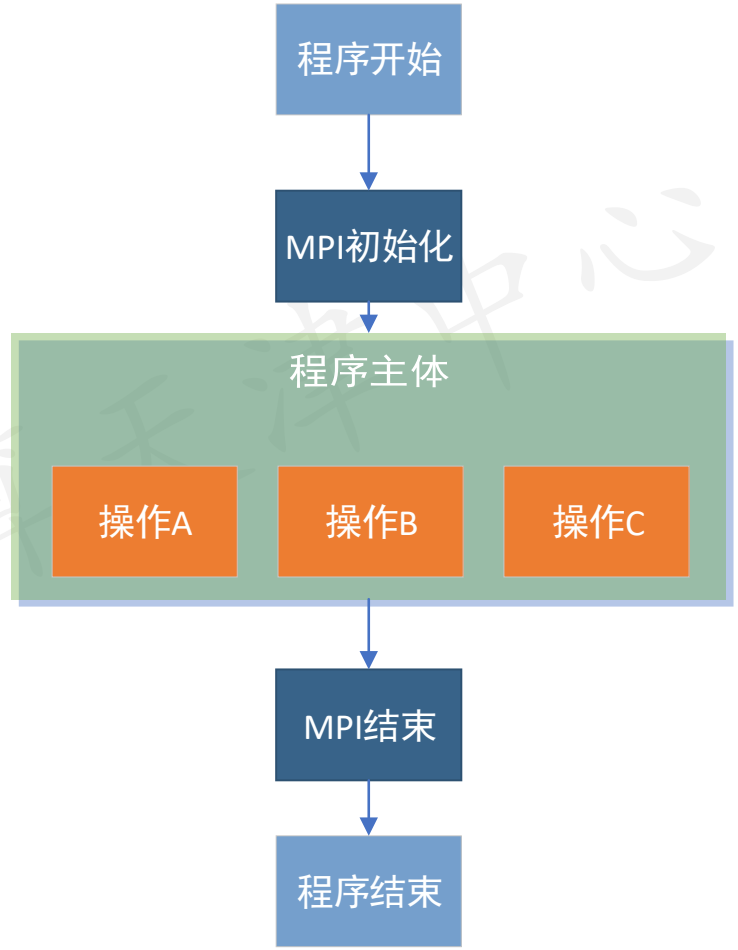
- 高效、资源利用率高；
- 同机群&超算契合程度高；
- 需要修改程序；

- 从长远角度看，
- 并行高效收益 > 并行实现代价；

串行程序



并行程序



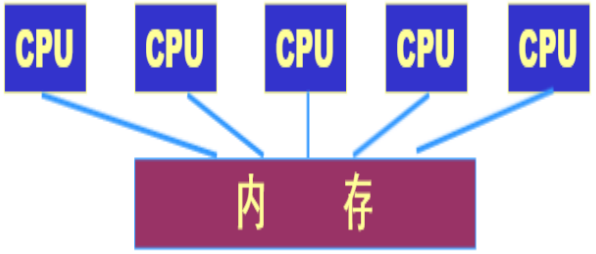
1.并行思想

什么是并行

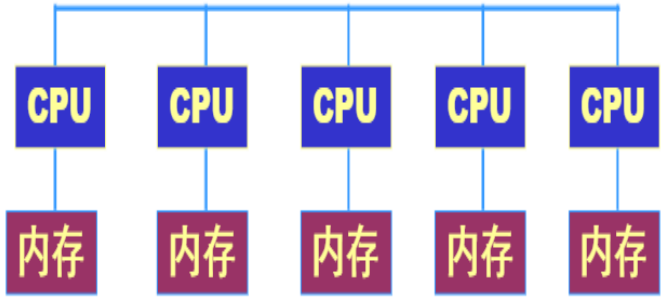
硬件并行	软件并行
多个CPU	任务并行
多个内存	数据并行

- 分解任务：
  - 简单，线性；（掰玉米1.0）
  - 逻辑，非线性；（掰玉米2.0）
- 分解数据；
- 数据通信；
- 数据存储；
- 进程标识（进程号）；

共享内存



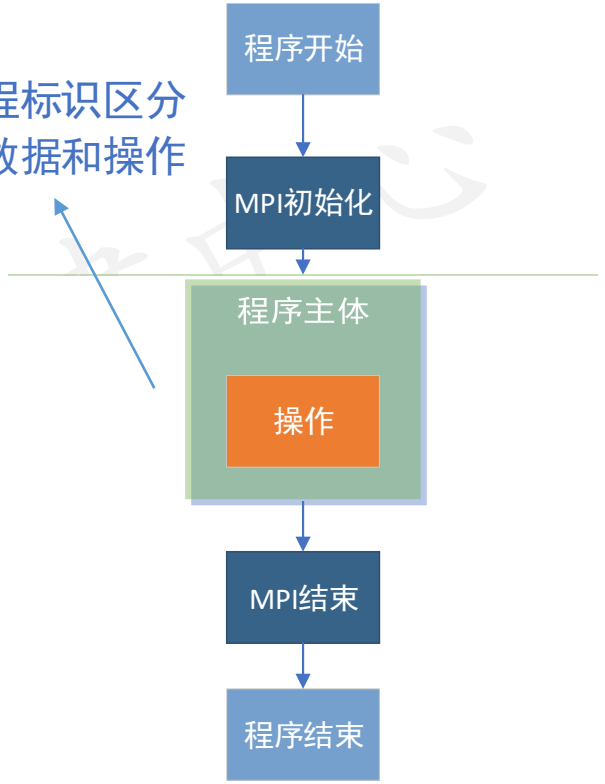
分布式内存



分布式共享内存

并行程序

通过进程标识区分
进程、数据和操作



单程序多数据（SPMD）

1.并行思想

怎样并行

- **适用于共享内存的多线程编程模型**
  - 硬件环境：支持超线程的单核CPU、多核CPU，SMP（对称多处理机）系统，三者组合
  - Win32多线程、Pthread、TBB(intel)、**OpenMP**
- **适用于分布式内存的消息传递编程模型**
  - 硬件环境：MPP（大规模并行处理机）、Cluster（集群）等分布式环境
  - PVM（并行计算机）、**MPI**
- **混合编程模型**
  - 硬件环境： SMP、MPP、集群等
  - MPI+Pthread、**MPI+OpenMP**、MPI+TBB
- **异构编程模型**
  - 硬件环境：GPU、DSP（数字信号处理器）、其他协处理器等
  - CUDA/openACC/openCL/C++ AMP（C ++加速大规模并行处理）

目录 Contents

2.基础知识

什么是MPI（Message Passing Interface）

- 是消息传递接口，一种基于信息传递的并行编程模型；
- 是针对消息传递库的跨语言标准和规范，区别于具体实现（MPICH、OPENMPI）；
- 不是一个消息传递库，而是有关该库应该是什么的规范；
- 本质是通过进程间的消息传递实现数据的封装和安全传递，达到并行；

发展历程

- MPI-1（1994）：点点通信、集合通信等
- MPI-2（1998）：非阻塞通信、并行I/O、动态进程管理
- MPI-3（2012）：完善了非阻塞集合通信、邻居概念

目标和优势

- 高通信性
- 可移植性
- 大规模扩展性
- 多功能
- 标准化

2.基础知识

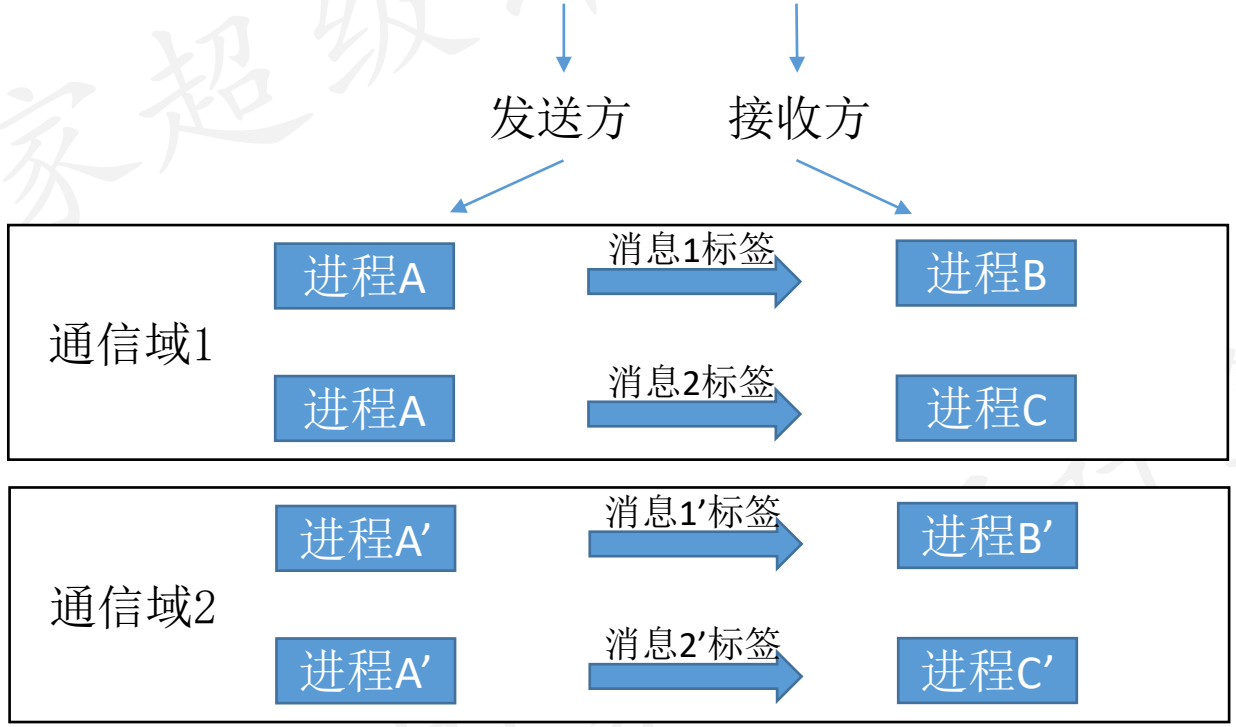
基本概念

- **进程：** MPI程序中独立参与通信的个体；
- **进程组：**
  - 部分或全部参加通信的进程的**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，即进程号**；（rank）
  - 若一共有N个进程参加通信则进程的**编号从0到N-1**；
  - 进程数在**提交并行任务时指定**；
- **通信上下文**
  - 提供相对独立的通信区域，不同的消息在不同的上下文中进行传递，将通信区分开；
  - 是隐式对象，作为通信域的一部分，无法直接操作；
- **MPI通信域**
  - 包括进程组和通信上下文；（范围和对象）
  - 分为组内通信域和组间通信域；

2.基础知识

基本概念

- **消息：**
  - 一个消息指进程间进行的一次数据交换；
  - 一个消息由通信域、源地址、目的地址、消息标签、数据个数和类型构成



- **标识：**区分同一组进程的不同通信；

2.基础知识

语言支持

- 支持C语言、C++和Fortran语言，包括77、90、03、08版本；
- C和Fortran语言的主要差异在于头文件和错误状态值；

```c
#include “mpi.h”
#include <stdio.h>
#include <math.h>
void main(int argc, char *argv[ ])
 {
      int myid, numprocs, namelen,ierr;
      char processor_name[MPI_MAX_PROCESSOR_NAME];
      MPI_Init(&argc,&argv);
      ierr = MPI_Comm_rank(MPI_COMM_WORLD,&myid);
      ierr = MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
      ierr = MPI_Get_processor_name(processor_name,&namelen);
      printf("Hello World! Process %d of %d on %s\n",myid, numprocs,
                processor_name);
      MPI_Finalize();
}
```

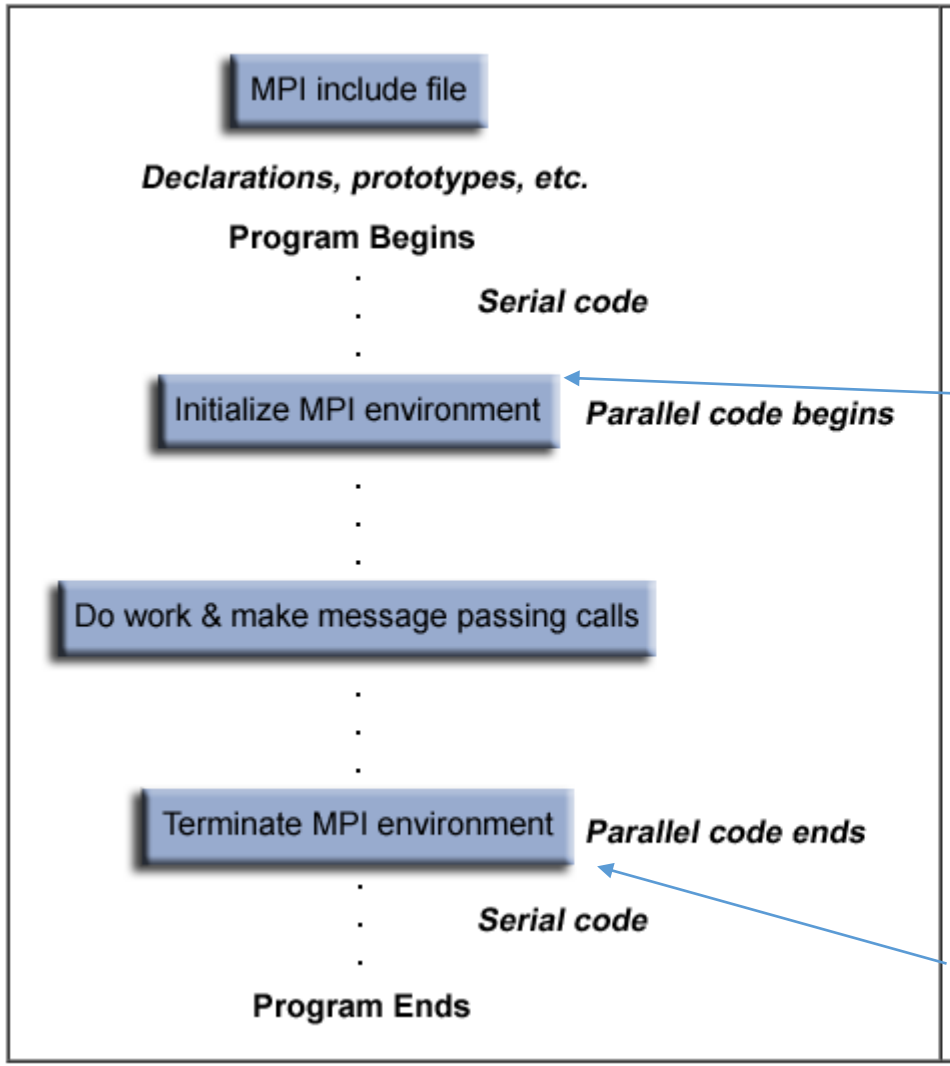
```fortran
program main
include 'mpif.h'
character * (MPI_MAX_PROCESSOR_NAME) processor_name
integer myid, numprocs, namelen, rc, ierr

call MPI_INIT( ierr )
call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
call MPI_GET_PROCESSOR_NAME(processor_name, namelen, ierr)
write(*,*) 'Hello World! Process ',myid,' of ',numprocs,' on ‘,
          processor_name
call MPI_FINALIZE(rc)
end program
```

C语言和Fortran语言 “hello world” MPI并行程序示例

2.基础知识

编程框架




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

int main (int argc, char *argv[])
{
int numtasks, rank, dest, source, rc, count, tag=1;
char inmsg, outmsg='x';
MPI_Status Stat;

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

if (rank == 0) {
  dest = 1;
  source = 1;
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  }

else if (rank == 1) {
  dest = 0;
  source = 0;
  rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, tag, MPI_COMM_WORLD, &Stat);
  rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, tag, MPI_COMM_WORLD);
  }

MPI_Finalize();
}
```

头文件

MPI环境初始化

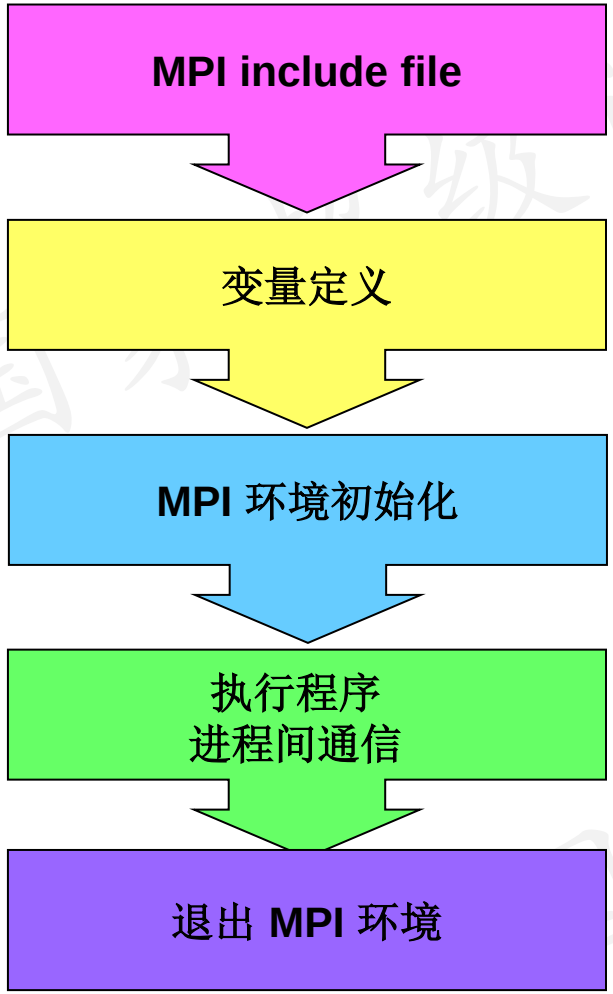
MPI进程属性

MPI通信

退出MPI

2.基础知识

编程框架



- 基本框架

```
MPI_Init(…);
MPI_Comm_size(…);
MPI_Comm_rank(…);
MPI_Send(…);
MPI_Recv(…);
MPI_Finalize();
```

- 开始与结束
  - MPI_INIT　　MPI环境初始化
  - MPI_FINALIZE　　退出MPI环境
- 进程身份标识
  - MPI_COMM_SIZE　　获取通信域内进程数目
  - MPI_COMM_RANK　　获取进程在通信域内的编号
- 发送和接受消息
  - MPI_SEND　　阻塞发送
  - MPI_RECV　　阻塞接收

2.基础知识

函数介绍

- **MPI_Init：** 初始化MPI执行环境；
  - 必须在每个MPI程序中调用此函数；
  - 必须在任何其他MPI函数之前调用此函数，且在MPI程序中只能调用一次；

编程语言	函数形式
C	MPI_Init（&argc，&argv）
Fortran	MPI_INIT（ierr）

- **MPI_Finalize：** 终止MPI执行环境；
  - 是每个MPI程序中最后一个调用的MPI例程；
  - 此后不得再调用其他MPI例程；

编程语言	函数形式
C	MPI_Finalize（）
Fortran	MPI_FINALIZE（ierr）

2.基础知识

函数介绍

- **MPI_Comm_size**：返回指定通信域中MPI进程的总数；
  - MPI默认将所有进程包含在MPI_COMM_WORLD通信域中；
  - 如comm是MPI_COMM_WORLD，则它表示应用程序可用的MPI任务数；

编程语言	函数形式
C	MPI_Comm_size(comm， &size)
Fortran	MPI_COMM_SIZE(comm， size， ierr)

参数	意义
comm	指定的通信域
size	指定通信域中包含的进程总数

- **MPI_Comm_rank**：返回指定通信域中MPI进程的序号，即进程号，或进程ID；
  - 最初，在通信域MPI_COMM_WORLD中，为每个进程分配一个介于0和任务数-1之间的唯一整数值，其通常称为任务ID；

编程语言	函数形式
C	MPI_Comm_rank（comm， &rank）
Fortran	MPI_COMM_RANK（comm， rank， ierr）

参数	意义
comm	指定的通信域
rank	该进程在指定通信域中的ID

2.基础知识

函数介绍

- **MPI_Send：** 发送数据；
  - 将本进程缓冲区buf中的datatype类型的count个数据发送给进程dest，该消息标识为tag；

编程语言	函数形式
C	MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
Fortran	MPI_SEND(BUF, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)

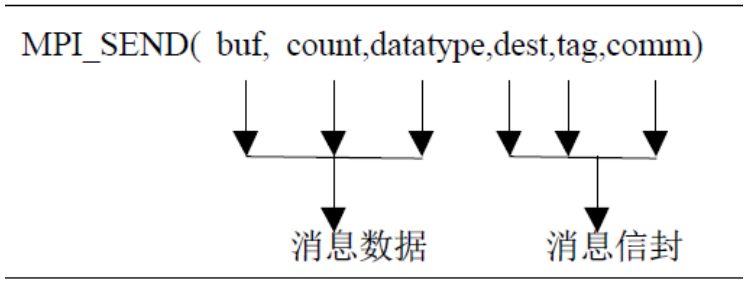
参数	意义
buf	发送缓冲区
count	将要发送的数据个数
datatype	将要发送的数据类型（MPI数据类型）
dest	目标（接收）进程号
tag	消息标签
comm	通信域

消息抬头

信封：<源/目，标识，通信域>

数据：<起始地址，数据个数，数据类型>

消息内容



2.基础知识

函数介绍

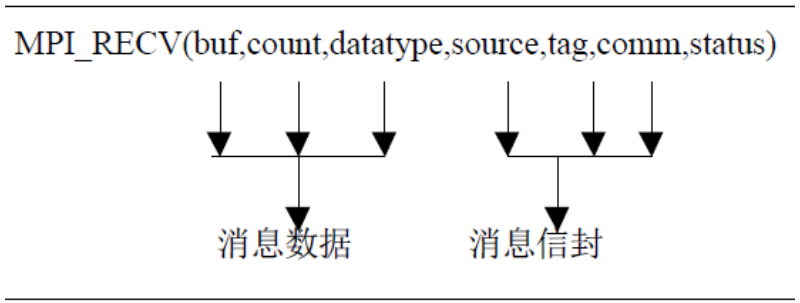
- **MPI_Recv：** 接收数据；
  - 从标识为tag的消息中接收来自source进程的datatype类型的count个数据到本进程缓冲区bug中；

编程语言	函数形式
C	MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag,MPI_Comm comm, MPI_Status *status)
Fortran	MPI_RECV(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS, IERROR)

参数	意义
buf	接收缓冲区
count	最多可接收的数据个数
datatype	接收的数据类型（MPI数据类型）
source	来源（发送）进程号
tag	消息标签
comm	通信域
status	返回状态，存放实际接收消息的状态信息，包括消息的源进程标识，消息标签，包含的数据项个数等

信封：<源/目，标识，通信域>

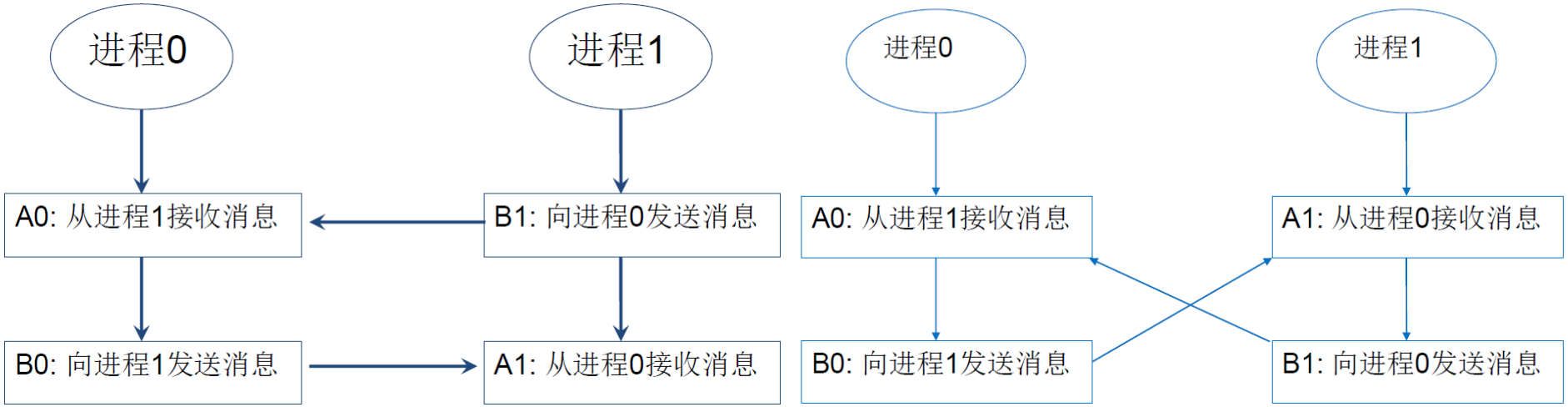
数据：<起始地址，数据个数，数据类型>



2.基础知识

函数介绍

- **MPI_SendRecv：**发送和接收数据；
  - 每一个进程都要向相邻的进程发送数据同时从相邻的进程接收数据；（如图，程序锁死）
  - 捆绑发送和接收操作，同时实现向其它进程的数据发送和从其它进程接收数据操作；
  - 由通信系统来实现系统会优化通信次序，有效避免不合理通信次序，最大限度避免死锁；



理想的通信　　通信死锁

2.基础知识

数据类型

- 预定义数据类型
  - MPI预先定义好的基础数据类型，如MPI_INT（C）、MPI_INTEGER（F）等；

MPI(C语言绑定)	C	MPI(Fortran语言绑定)	Fortran
MPI_BYTE		**MPI_ BYTE**	
MPI_CHAR	signed char	MPI_CHARACTER	CHARACTER
		MPI_COMPLEX	COMPLEX
MPI_DOUBLE	double	MPI_DOUBLE_PRECISION	DOUBLE_PRECISION
MPI_FLOAT	float	MPI_REAL	REAL
MPI_INT	int	MPI_INTEGER	INTEGER
		MPI_LOGICAL	LOGICAL
MPI_LONG	long		
MPI_LONG_DOUBLE	long double		
MPI_PACKED		MPI_PACKED	
MPI_SHORT	short		
MPI_UNSIGNED_CHAR	unsigned char		
MPI_UNSIGNED	unsigned int		
MPI_UNSIGNED_LONG	unsigned long		
MPI_UNSIGNED_SHORT	unsigned short		

2.基础知识

数据类型

- 派生数据类型
  - 可通过提供的函数接口定义派生数据类型；
  - 可为数据类型不同且地址空间不连续的数据组成；（主要针对不连续数据处理）

连续复制类型：

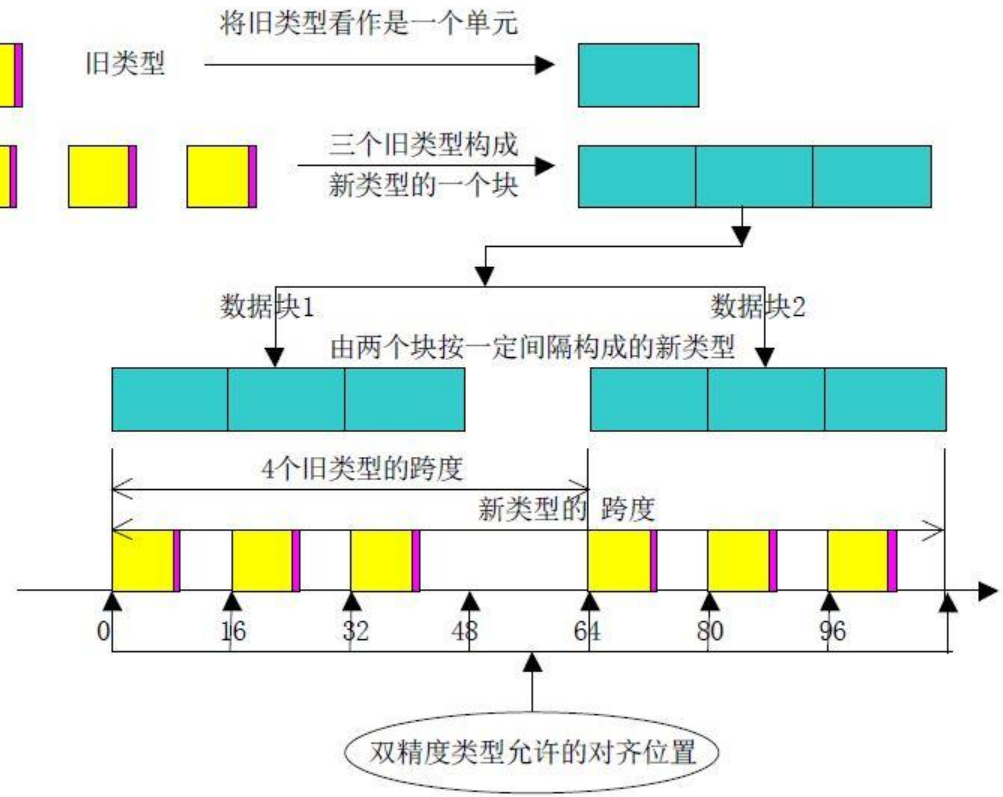
```
MPI_TYPE_CONTIGUOUS(count, oldtype, newtype)
IN      count       复制个数
IN      oldtype     旧数据类型
OUT     newtype     新数据类型
```

不连续数据类型：

```
MPI_Type_vector(count, blocklen, stride,
                oldtype, newtype)
IN    count     不连续数据块数量
IN    blocklen            每块所含数据长度
IN    stride    各块第一个元素之间相隔的元素个数
IN    oldtype   旧数据类型
OUT   newtype   新数据类型
```

派生数据类型的提交：

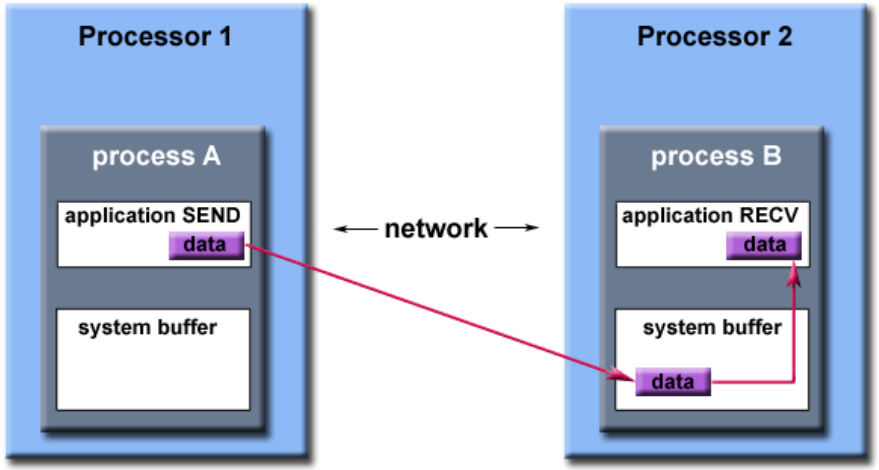
```
MPI_Type_commit(MPI_Datatype *datatype)
```



2.基础知识

缓冲区

- 对于MPI通信，即消息发送（MPI_Send）和接收（MPI_Recv）二者操作是不同步的，因此，MPI实现必须能够处理存储数据，不至于丢失数据；
- 例如
  - 发送在接收进程操作启动之前将数据送达；
  - 多个发送同时达到一个接收进程；
- 系统缓冲区：无需用户管理，自动实现，提高程序性能；
- 用户缓冲区：用户管理的发送缓冲区；



Path of a message buffered at the receiving process

2.基础知识

通信方式

- **点点通信**
  - 通常涉及在两个（只有两个）不同的MPI任务之间传递消息。一个任务正在执行发送操作，另一任务正在执行匹配的接收操作。
  - **通信机制：**阻塞和非阻塞通信；（发送和接收）
  - **通信模式：**标准通信模式、同步通信模式、缓冲通信模式、就绪通信模式；（发送）
  - 不同通信机制和通信模式互相结合，实现丰富的点对点通信；

类型	阻塞	非阻塞
标准通信模式（发送操作）	MPI_Send	MPI_Isend
同步通信模式（发送操作）	MPI_ Ssend	MPI_ Issend
缓冲通信模式（发送操作）	MPI_ Bsend	MPI_ Ibsend
就绪通信模式（发送操作）	MPI_ Rsend	MPI_ Irsend
接收操作	MPI_Recv	MPI_Irecv

2.基础知识

通信方式

- **通信机制**
  - **阻塞通信：**

  1. 发送或者接收操作完成函数才返回，否者一直等待；
  2. 针对数据而言，发送缓冲区可用，或者接收缓冲区数据可用则阻塞调用返回；

  - **非阻塞通信**

  1. 函数调用后立即返回，执行下面的操作，不管该通信是否完成；
  2. 针对消息，即将消息发送给MPI库函数，其隐式执行通信操作，用户无法预测和判断；
  3. 发送缓冲区不可复用，或者接收缓冲区不一定有数据；

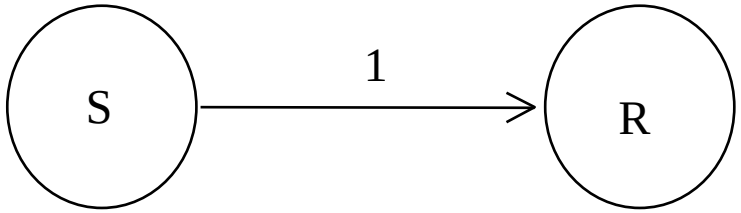
- **针对非阻塞通信，MPI提供了通信完成检测函数，主要的有两种：MPI_Wait函数和MPI_Test函数，用于更好的利用非阻塞通信方式；**
- **后者主要用于通信和计算的叠加，实现程序运行的高效；**

2.基础知识

通信方式

- **通信模式**
  - 标准通信：是否对发送的数据进行缓冲**由MPI的实现来决定**，而不是由用户程序来控制；

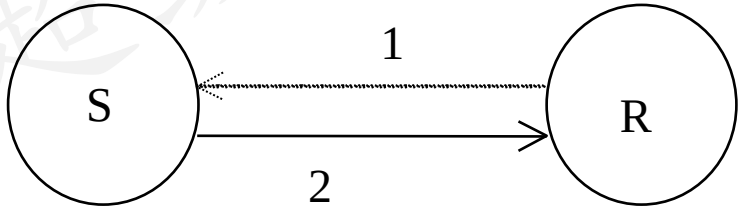
Standard



  - 就绪通信模式：只有在接收进程的**接收操作已经开始**，发送操作才进行发送；

1. 当发送操作启动而相应的接收还没有启动，发送操作将出错；
2. **接收操作必须先于发送操作启动；**

Ready

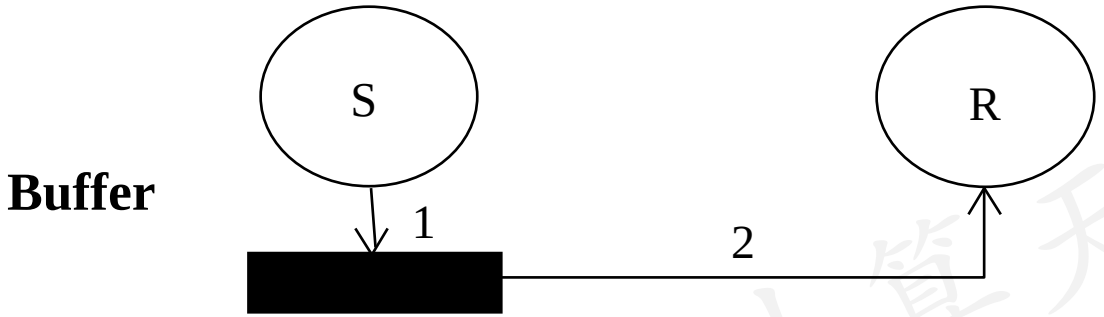


通信方式

- 通信模式

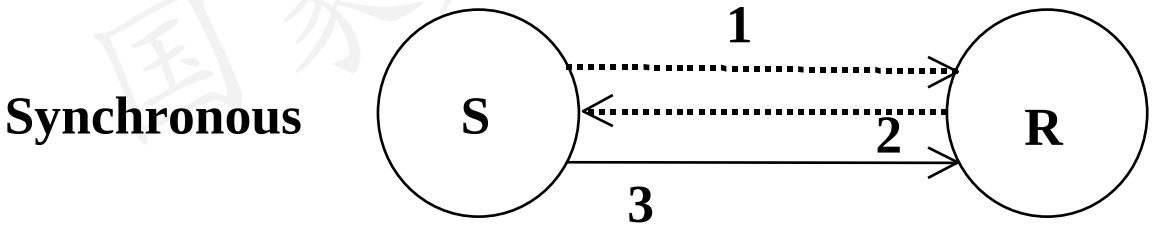
➢ 缓冲通信模式：**用户手动申请并管理缓冲区**，只将消息发送到该缓冲区就返回；

1. 需要用户事先申请一块足够大的缓冲区；
2. 通过MPI_Buffer_attch实现，通过MPI_Buffer_detach来回收申请的缓冲区；



➢ 同步通信：只有相应的**接收过程已经启动**，发送过程才正确返回；

1. 发送缓冲区中的数据已经全部**被系统缓冲区缓存**，并已开始发送到接收缓冲区；
2. **发送缓冲区可以被释放或者重新使用**；



2.基础知识

通信方式

- **集合通信**
  - 一个通信域中的所有进程都参加的全局通信操作
  - 一般实现三个功能：

通信功能完成进程组内数据的传输，如广播、分散/收集等；

聚集功能基于通信对数据完成一定的操作，如比较大小、加减等；

同步功能实现组内进程在执行进度上取得一致 ；

  - 按照通信方向的不同，可分为三种：

一对多通信：一个进程向其它所有的进程发送消息，这个负责发送消息的进程叫做Root进程

多对一通信：一个进程负责从其它所有的进程接收消息，这个接收的进程也叫做Root进程

多对多通信：每一个进程都向其它所有的进程发送或者接收消息

2.基础知识

通信方式

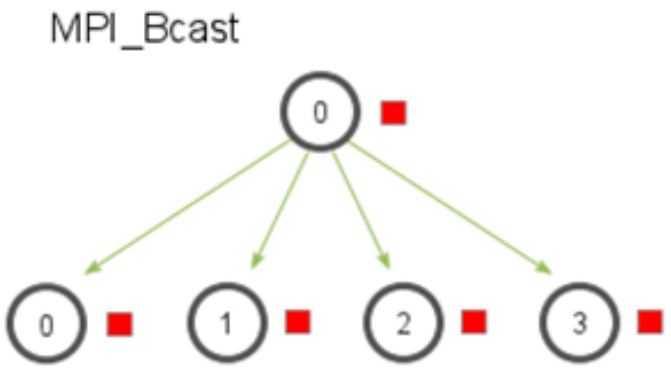
类型	函数名	含义
通信	MPI_Bcast	**一对多，广播同样的消息**
	MPI_Scatter	**一对多，散播不同的消息**
	MPI_Scatterv	MPI_Scatter的一般化
	MPI_Gather	**多对一，收集各个进程的消息**
	MPI_Gatherv	MPI_Gather的一般化
	MPI_Allgather	全局收集
	MPI_Allgatherv	MPI_Allgather的一般化
	MPI_Alltoall	**多对多，全局交换消息**
	MPI_Alltoallv	MPI_Alltoall的一般化
聚集	MPI_Reduce	**多对一，归约**
	MPI_Allreduce	MPI_Reduce的一般化
	MPI_Reduce_scatter	MPI_Reduce的一般化
	MPI_Scan	扫描
同步	MPI_Barrier	**同步**

简洁、高效、可读性强、已被优化

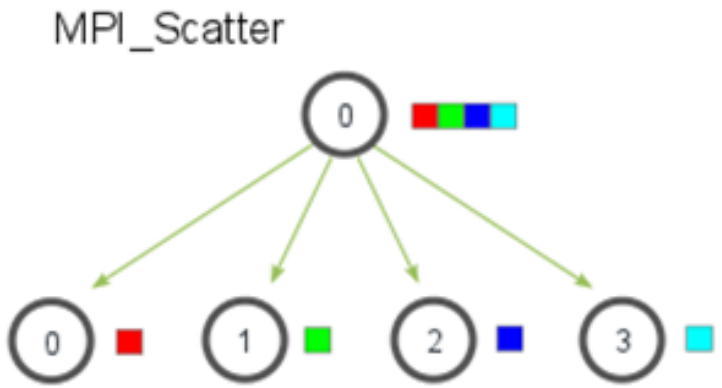
2.基础知识

通信方式

- MPI_Bcast：一对多，广播
  - 从标识为root的进程将一条消息广播发送到组内的所有其它进程，也包括它本身；



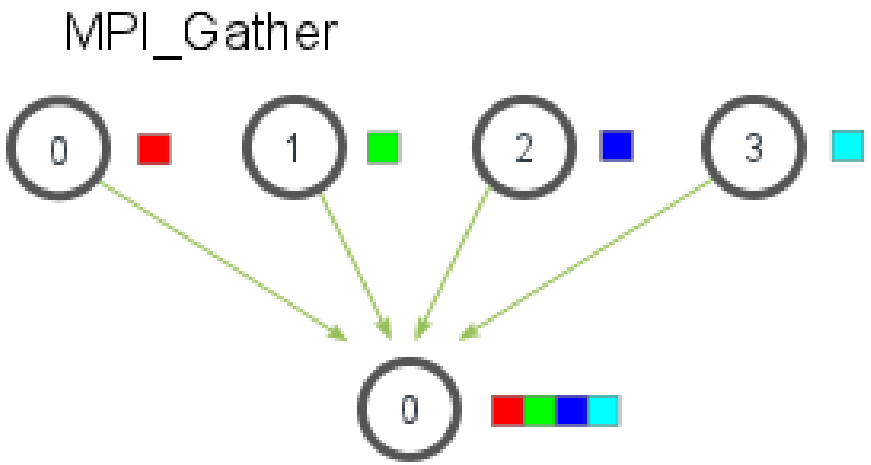
- MPI_Scatter：一对多，散播
  - 标识为root的进程可按顺序发送不同数据到组内其他进程；



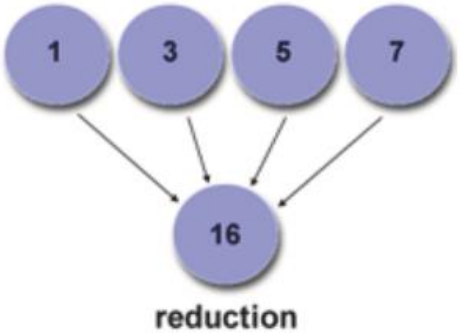
2.基础知识

通信方式

- MPI_Gather：多对一，收集
  - 和MPI_scatter刚好相反；
  - 将组内每个进程的数据按照顺序集中到标识为root的进程中；



- MPI_Reduce：多对一，归约
  - 和MPI_Gather类似；
  - 将组内每个进程的数据按给定操作进行运算并将结果返回标识为root的进程中；



2.基础知识

通信方式

运算操作符	描述	运算操作符	描述
MPI_MAX	最大值	MPI_LOR	逻辑或
MPI_MIN	最小值	MPI_BOR	位或
MPI_SUM	求和	MPI_LXOR	逻辑异或
MPI_PROD	求积	MPI_BXOR	位异或
MPI_LAND	逻辑与	MPI_MINLOC	计算一个全局最小值和附到这个最小值上的索引——可以用来决定包含最小值的进程的秩
MPI_BAND	位与	MPI_MAXLOC	计算一个全局最大值和附到这个最大值上的索引——可以用来决定包含最小值的进程的秩

MPI_Reduce约定的归约操作

2.基础知识

虚拟拓扑

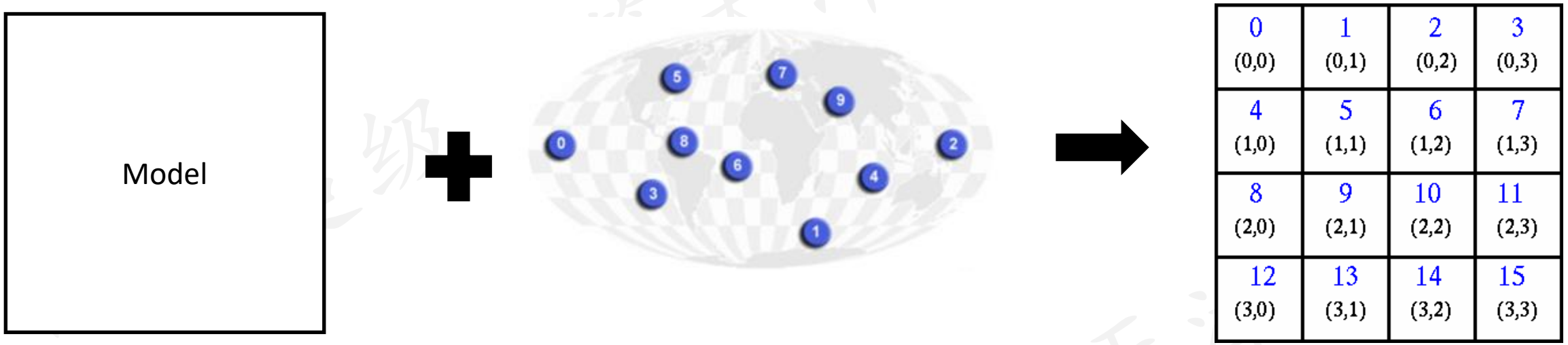
- 原因：
  - 在许多并行程序中，进程的线性排列不能充分地反映进程间逻辑上的通信模型；
  - 通常由问题几何和所用的算法决定；
  - 考虑下面问题，每个进程处理一个块的数据，上下左右块之间要通信，需要人为确定通信进程和数据；

Model

0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

2.基础知识

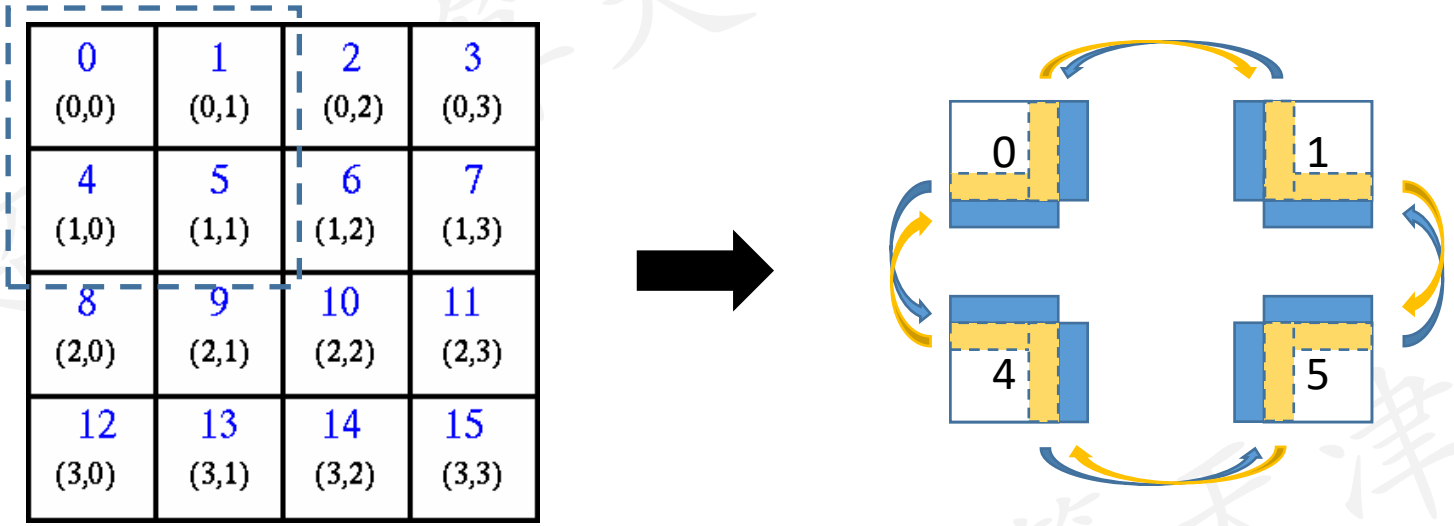
虚拟拓扑



- **定义：**
  - 提供将进程排列成二维或三维网格形式的逻辑拓扑模型功能，即虚拟拓扑；
  - 其针对进程组内通信，简化有特定拓扑要求的算法并行程序的编写；
  - 拓扑还可以辅助运行时系统将进程映射到实际的硬件结构之上；
- **分类：**
  - 笛卡尔拓扑和图拓扑；

2.基础知识

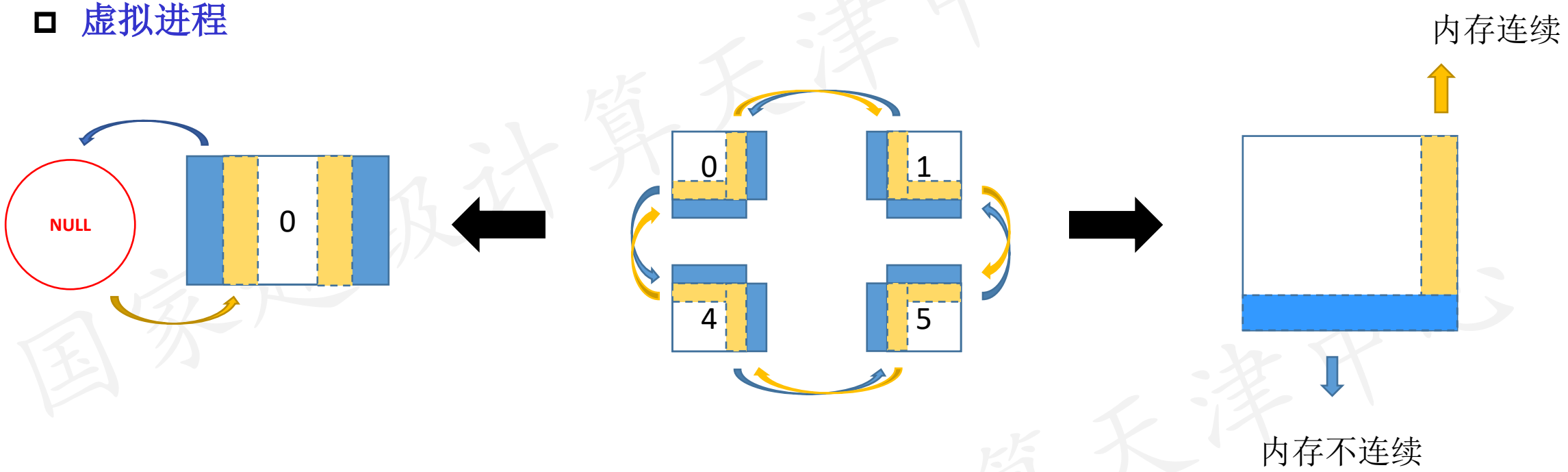
虚拟拓扑



- 笛卡尔拓扑方法：
  - 通过MPI_Cart_create函数即可定义笛卡尔拓扑结；
  - MPI_Cart_cords　根据rank号获取坐标号；
  - MPI_Cart_shift　根据rank号获取相邻进程rank号；

2.基础知识

虚拟进程

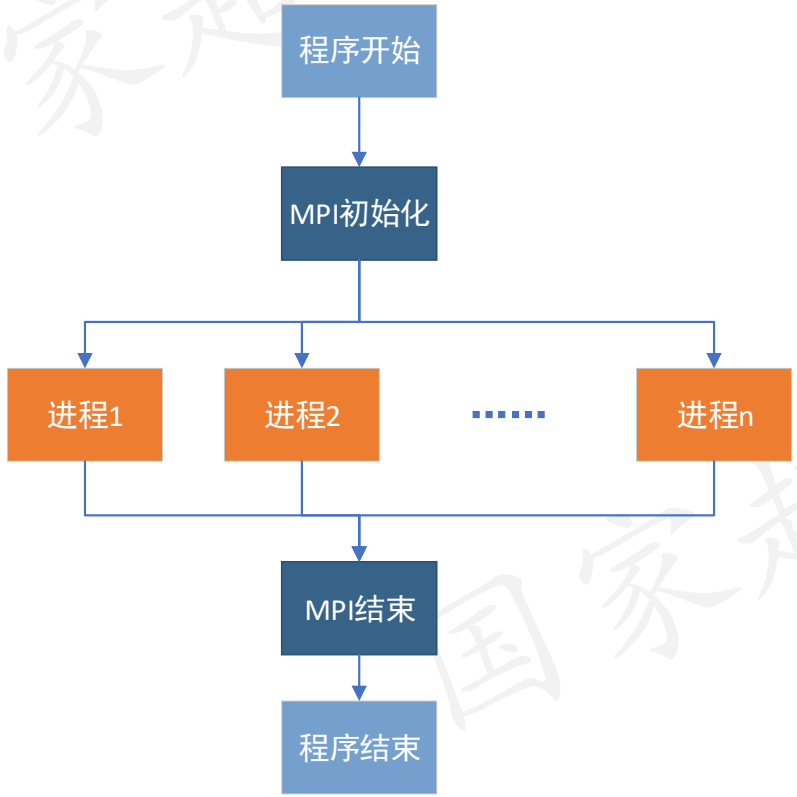


- **虚拟进程（MPI_PROC_NULL）**
  - 是不存在的假想进程；
  - 充当真实进程通信的目的或源，为了方便在某些情况下编写通信语句；
  - 当一个真实进程向一个虚拟进程发送数据或接收数据时，该进程会立即正确返回，如同执行了一个空操作；
- **数据不连续：派生数据类型；**

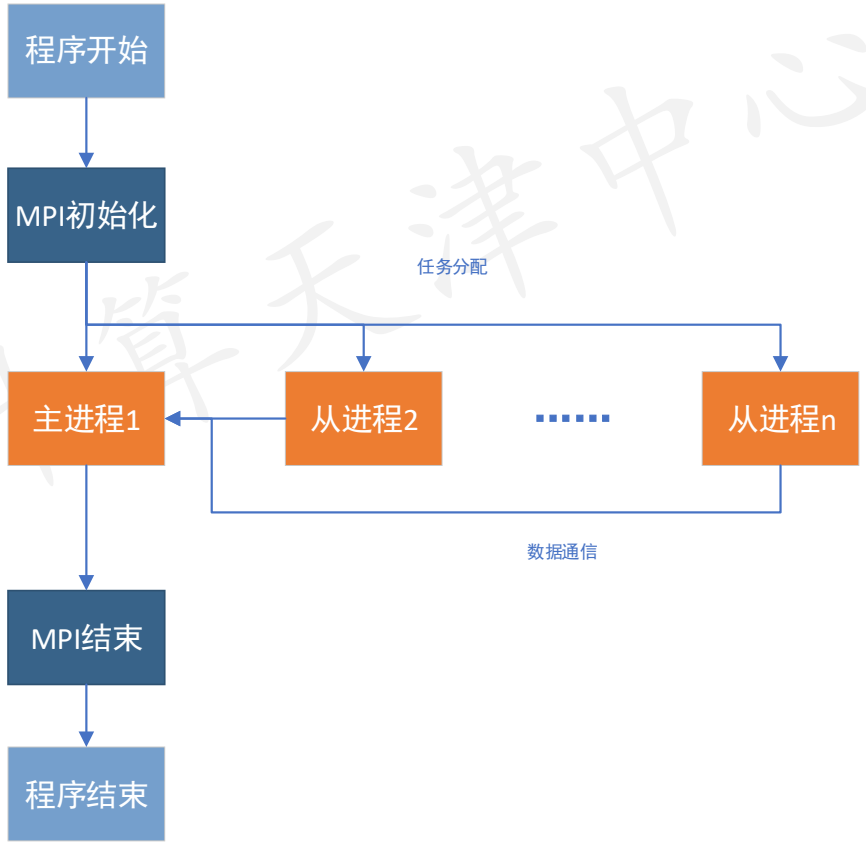
2.基础知识

并行编程模式

- 对等模式：每个进程之间功能和代码基本一致，只是处理的数据或对象不同；



- 主从模式：进程分主进程和从进程，主进程接收从进程的处理结果，并进程汇总处理；



2.基础知识

MPI并行程序设计特点

- 每个进程有独立的地址空间，用于存储数据；
- 每个进程相互之间不能直接访问，必须通过消息传递实现；
- 需要用户显示处理消息，即发送和接收，以实现进程间的数据通信；
- 并行编程的重点在于对问题的分解和不同进程间的数据交换组织；
- 并行粒度较大，适用于大规模可扩展并行算法；
- 适用于超算集群等大规模并行集群系统，可充分发挥集群性能，达到较高效率；

目录 Contents

3.编程实例

程序名称	功能
helloworld_0	串行helloworld程序
helloworld_1	并行helloworld程序
helloworld_2	并行helloworld程序，进程号
sendrecv_0	Send和recv基本功能实现程序
sendrecv_1	Sendrecv捆绑函数程序
sendrecv_0_error_0	Send和recv基本功能实现程序，锁死情况
sendrecv_0_error_1	Send和recv基本功能实现程序，消息标识不对应
jacobi_s	串行jacobi迭代程序
jacobi_p	并行jacobi迭代程序

3.编程实例

Jacobi迭代算例

- **问题来源：**
  - 数值求解二维区域上的Laplace方程，即扩散问题；
  - 采用中心差分格式离散方程；

$$\begin{cases} \Delta h(x,y) = \dfrac{\partial^2 h(x,y)}{\partial^2 x} + \dfrac{\partial^2 h(x,y)}{\partial^2 y} = 0 \qquad x,y \in [0,1] \\ h(0,y) = h(1,y) = h(x,0) = h(x,1) = 8 \end{cases}$$

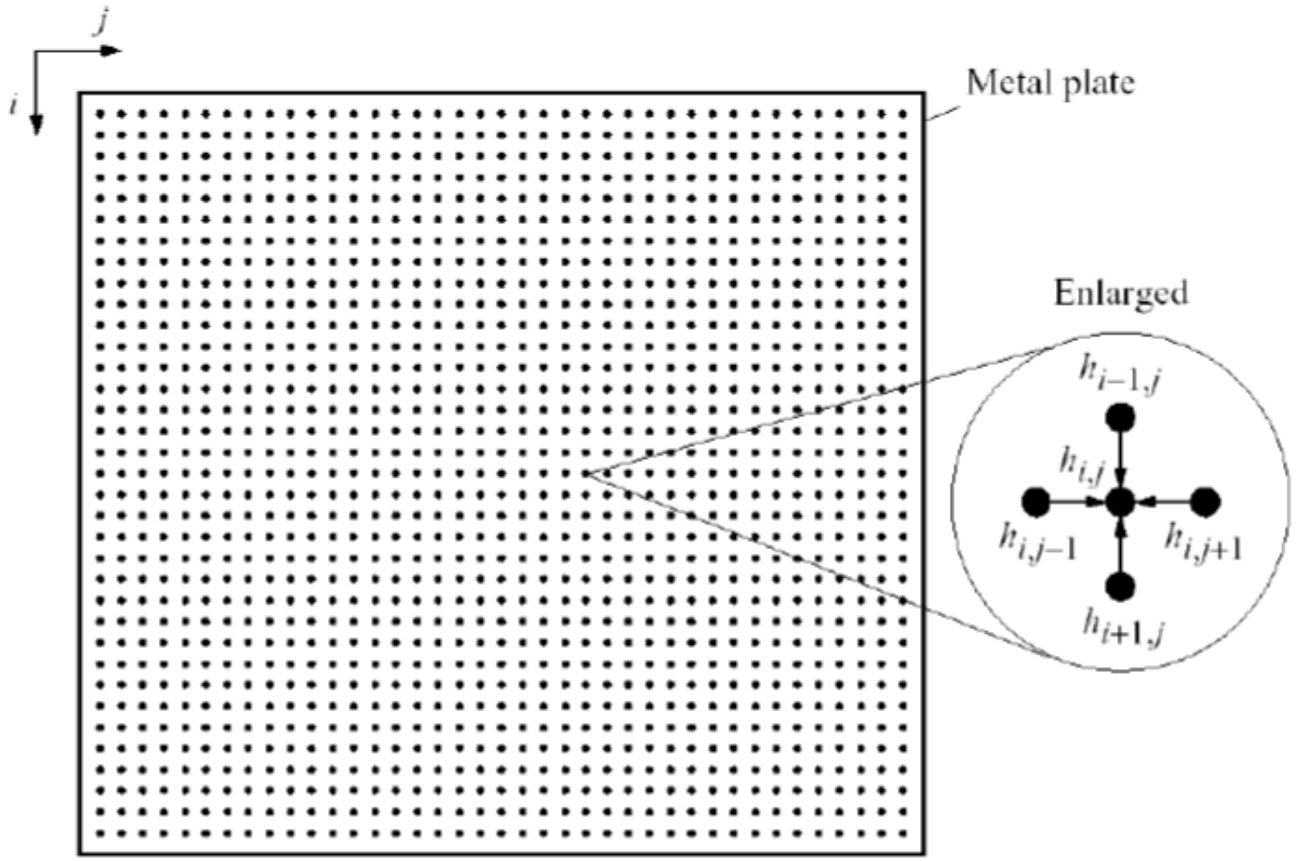
- **核心算法：jacobi迭代法**
  - 从任意已知初始值开始，反复带入方程计算，多步迭代；
  - 直到前后两次数值结果差值在设置误差范围内，即得到数值解；

解 Ax=b 的基本迭代公式为

$$\begin{cases} x^{(0)} = (x_1^{(0)}, \cdots, x_n^{(0)}), \\ x_i^{(k+1)} = (b_i - \sum\limits_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(k)}) / a_{ii} (i = 1,2,\cdots,n)(k = 0,1,\cdots). \end{cases}$$

3.编程实例

Jacobi迭代算例



- 已知当前步的值，计算下一迭代步的值；
- i,j点需要周围四个点的值；

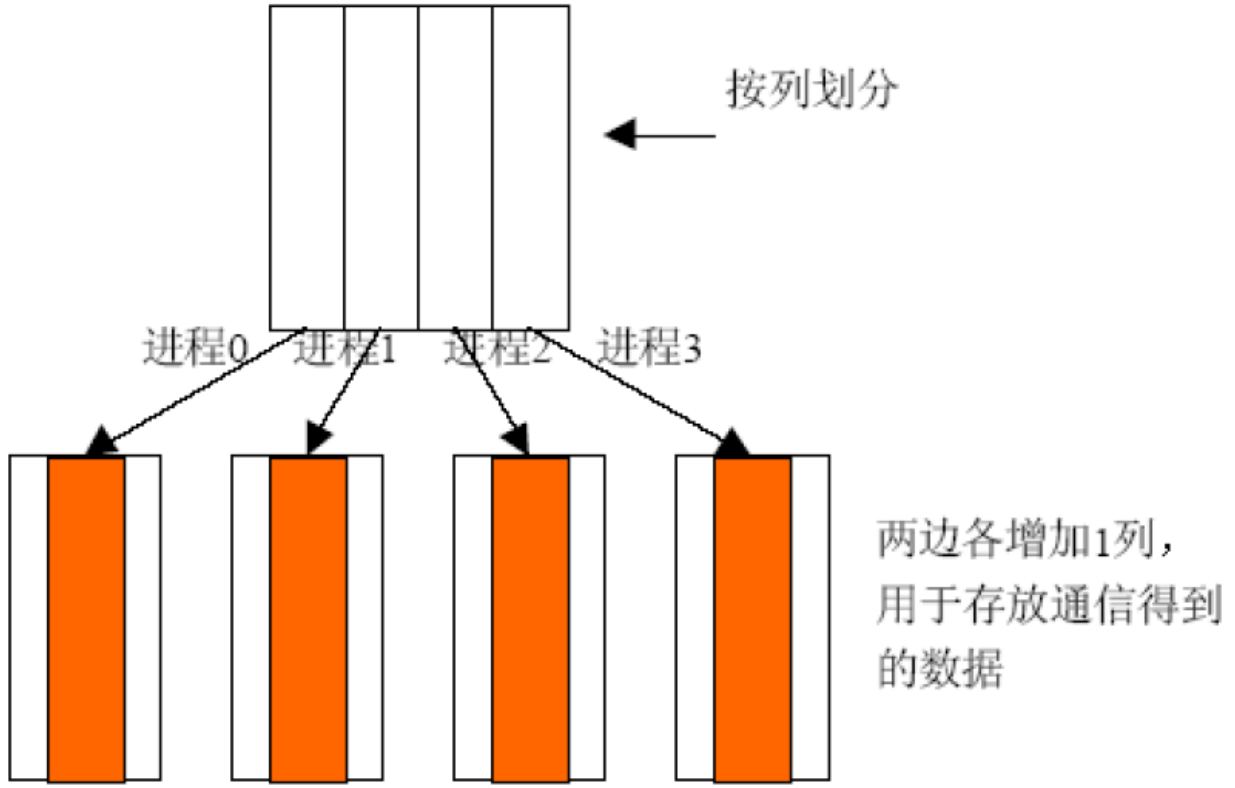
$$h_{i,j} = \frac{h_{i-1,j} + h_{i+1,j} + h_{i,j-1} + h_{i,j+1}}{4}$$

3.编程实例

Jacobi迭代算例

虚点+通信

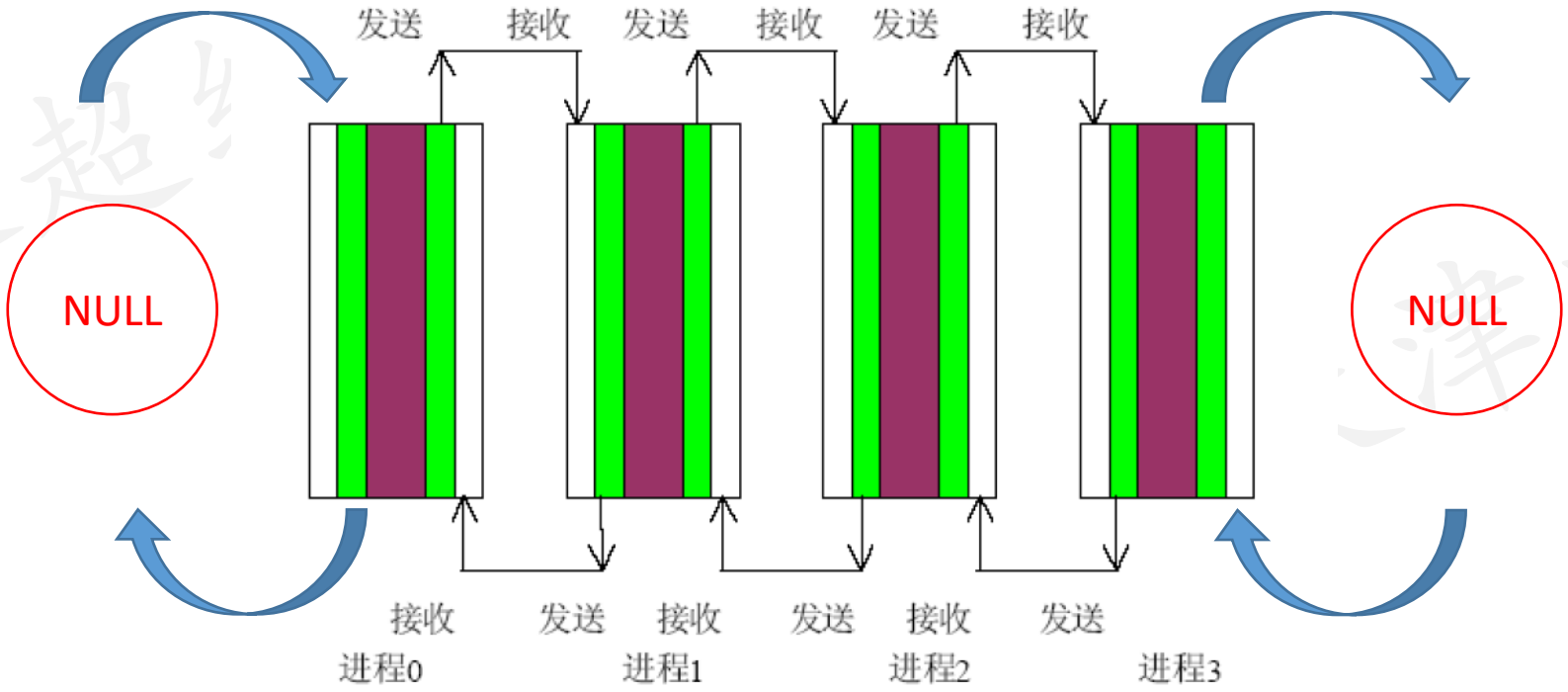
- 简单将列按进程数划分成块；
- 内部点边界值计算需要相邻块的值；
- 物理边界不计算，只赋值；
- 为格式统一简便，边界单元也扩展；



并行数据划分示意图

3.编程实例

- Jacobi迭代算例



数据通信示意图

目录 Contents

4.性能测试

- 测试情况：
  - 迭代步数为10000步；
  - 并行测试进程数为25；

测试时间	矩阵规模				
	500	1000	2000	4000	8000
串行（s）	2.609	10.483	49.369	340.760	1305.236
并行（s）	4.157	8.266	22.854	128.465	513.22
加速比	0.628	1.268	2.160	2.653	2.543

- 改进优化：
  - 可采用虚拟拓扑方法，将数据两个方向均划分成块；
  - 可采用派生类型方法，定义不连续的数据边界；
  - 按照差值量级判断是否收敛；

0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

谢 谢！

www.nscc-tj.cn

电话：022-65375561

邮箱：service@nscc-tj.cn