MPI 并行程序设计平台

一、 典型并行计算环境

二、 进程与消息传递

三、 MPI 应用现状

四、 MPI 并行程序设计入门（程序示例 1）

五、 MPI 基本函数

六、 作业一

七、 MPI 并行程序示例 2（求解-Δu=f）；

八、 作业二

九、 MPI 先进函数

十、 MPI 的发展

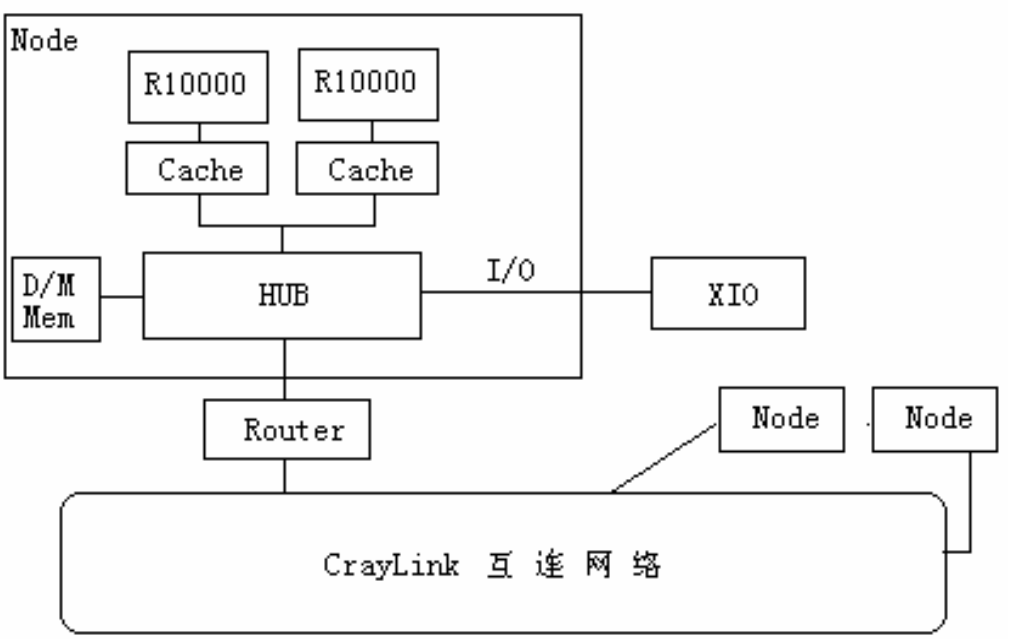
十一、 作业三

十二、 MPI 并行程序设计的高性能要求

一、典型并行计算环境

1．硬件环境（**SMP、DSM、MPP、机群系统**）：

- ●共享存储对称多处理机系统（SMP）：
  - ➢ 对称式共享存储：任意处理器可直接访问任意内存地址，且访问延迟、带宽、几率都是等价的；
  - ➢ 微处理器：8-32 个，SUN Starfare 可达 64 个，几十到几百亿次；
  - ➢ 例子：SGI Power Challenge, SUN Enterprise；
- ●分布共享存储多处理机系统（DSM）：
  - ➢ 分布共享存储：内存模块物理上局部于各个处理器内部，但逻辑上（用户）是共享存储的；
  - ➢ 基于 Cache 目录的非一致内存访问（CC-NUMA）：局部与远程内存访问的延迟和带宽不一致，3-10 倍 —> 高性能并行程序设计注意；
  - ➢ 微处理器：16-128 个，几百到千亿次；
  - ➢ 代表：SGI Origin 2000（附图一：体系结构）；



Origin 2000 体系结构

- 大规模并行计算机系统（MPP）：
  - 单一的分布内存大规模并行机（DM-MPP）：
    - 成千上万个微处理器或向量处理器通过专用高性能互连网连接，物理和逻辑上均分布内存；
    - 代表：CRAY T3E（2048）、ASCI Red（3072）、IBM SP2、IBM SP3；
  - SMP-Cluster 系统：
    - 结构：多台 SMP 或 DSM 并行机（超节点）通过专用高性能互连网连接，超节点内部共享内存，超节点之间分布或者共享内存；
    - 代表：HP/Convex SPP-2000（4 台 16-way HP SMP

超节点，64 个处理机），ASCI Blue Mountain（48 台 128-way DSM Origin 2000，6144 个处理器）；

- 机群系统：
  - 工作站或微机通过商用高性能交换机连接而成，分布存储，几台到几十台处理机，性能价格比高；
  - 代表：微机机群、工作站机群等；

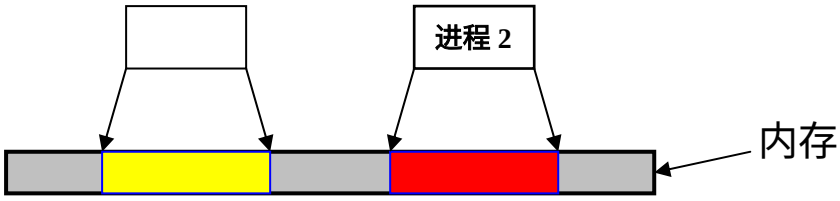
2．软件环境：

- 操作系统：UNIX、LINUX、Windows NT；
- 并行程序设计平台：
  - 共享存储 OpenMP：SMP、DSM；
  - 数据并行 HPF：SMP、DSM、MPP；
  - 消息传递 MPI、PVM：所有并行计算环境；

二、进程与消息传递

1．单个进程（process）

- 进程是一个程序，同时包含它的执行环境（内存、寄存器、程序计数器等），是操作系统中独立存在的可执行的基本程序单位；
- 通俗理解：串行应用程序编译形成的可执行代码，分为“ 指令 ”和“ 数据 ”两个部分，并在程序执行时“ 独立地申请和占有 ”内存空间，且所有计算均局限于该内存空间。



2．单机内多个进程：

- 多个进程可以**同时存在于单机内同一操作系统**：由操作系统负责调度分时共享处理机资源（CPU、内存、存储、外设等）；
- **进程间相互独立（内存空间不相交）**：在操作系统调度下各自独立地运行，例如多个串行应用程序在同一台计算机中运行；
- **进程间可以相互交换信息**：例如数据交换、同步等待，消息是这些交换信息的基本单位，消息传递是指这

些信息在进程间的相互交换，是实现进程间通信的唯一方式；

- **最基本的消息传递操作**：
  - 发送消息（send）；
  - 接受消息（receive）；
  - 进程同步（barrier）：各进程各自预先约定某个程序执行点（同步点），程序执行过程中，当且仅当所有进程均达到各自的同步点后，各进程才开始继续执行，否则空闲等待其他进程；
- 规约（reduction）：各个进程公布其某个变量的值，然后对这些值执行满足结合律和交换律的某种运算，如“加、乘、最大值、最小值、按位与、按位或、按位异或”等，也可以是程序定义的某种运算。

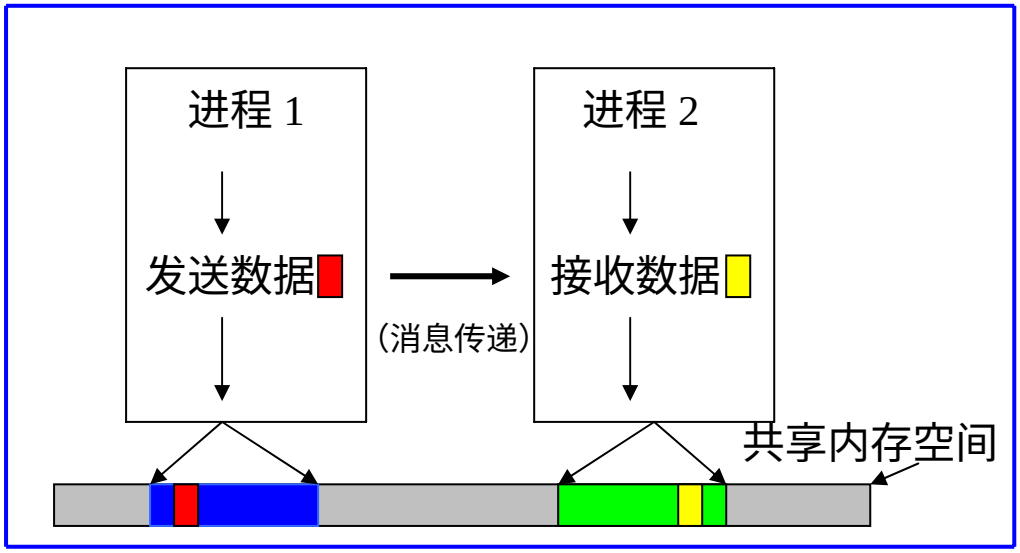


图 1　位于同一处理机的两个进程间消息传递示意图

- **消息传递的具体实现**：共享内存或信号量，用户不必关心；

3．包含于通过网络联接的不同计算机的多个进程：

- **进程独立存在**：进程位于不同的计算机，由各自独立的操作系统调度，享有独立的 CPU 和内存资源；
- **进程间相互信息交换**：消息传递；
- **消息传递的实现**：基于网络 socket 机制,用户不必关心；

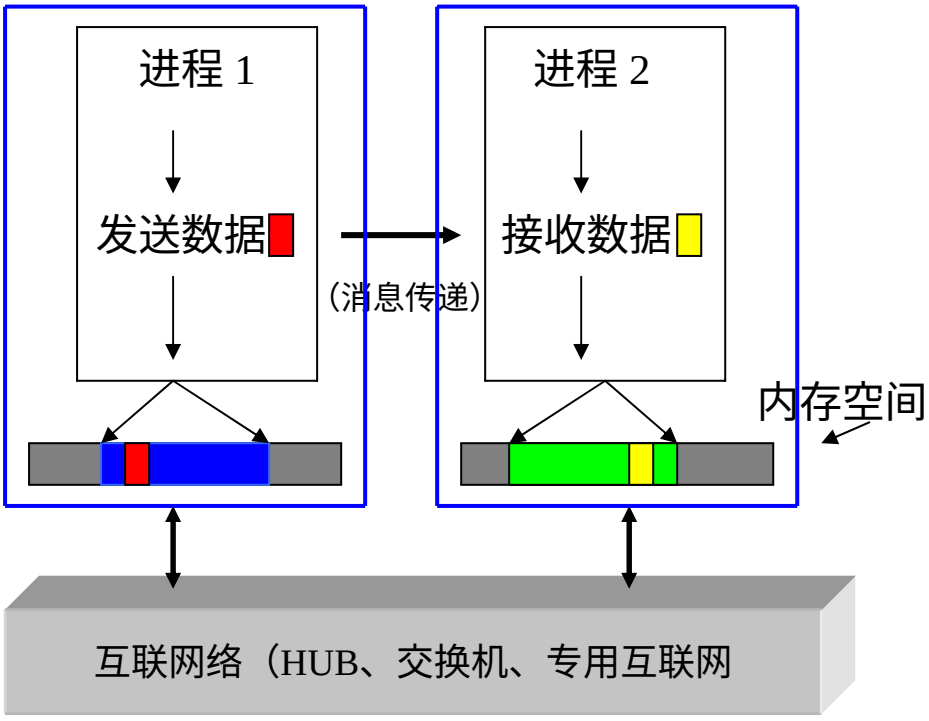


图 2　位于不同处理机的两个进程间消息传递示意图

4．消息传递库函数:

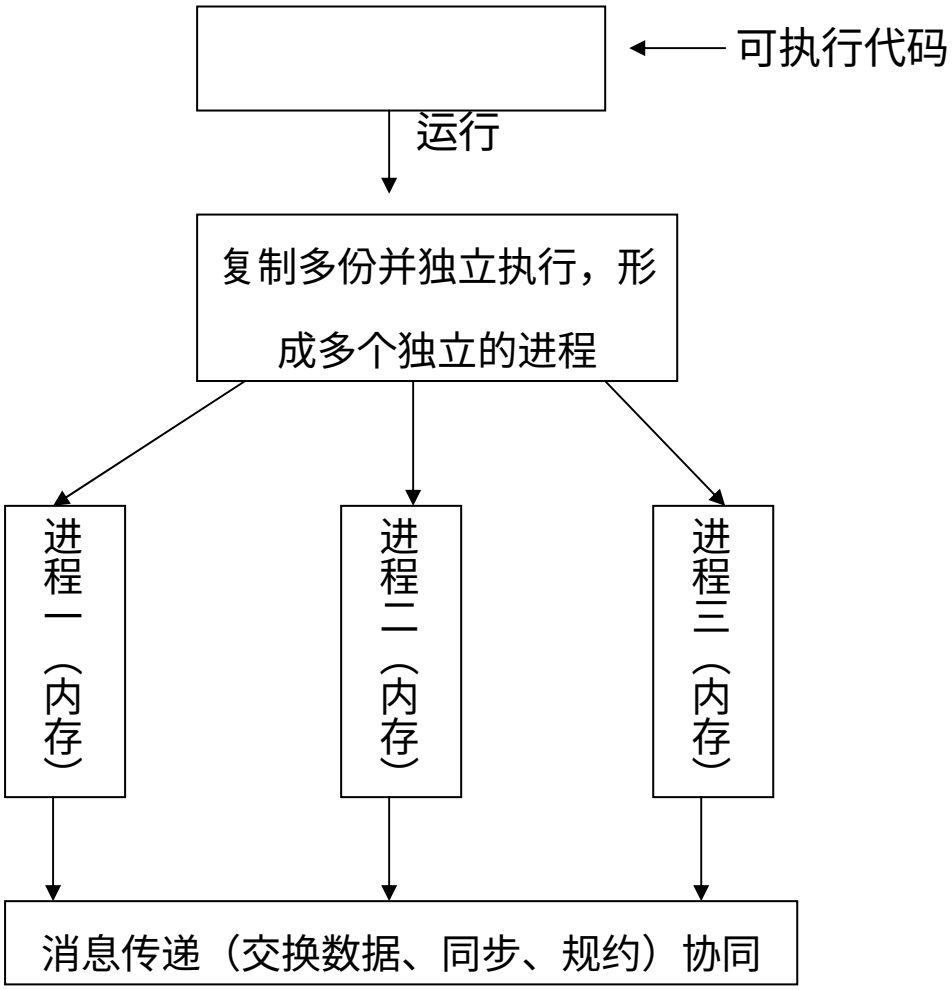
- **应用程序接口**（API）:提供给应用程序（FORTRAN、C、C++语言）的可直接调用的完成进程间消息传递的某项特定功能的函数；
- **消息传递库**:所有定义的消息传递函数编译形成的软件库，调用其内部函数的应用程序，通过与之联接，即可成为可并行执行的程序；
- **目前流行的消息传递函数库**：PVM 3.3.11、MPICH 1.2、LAMMPI 6.4 等；

5．标准消息传递界面（MPI：Message Passing Interface）：

- **MPI 标准**：根据应用程序对消息传递功能的需求，全球工业、应用和研究部门联合推出标准的消息传递界面函数，不考虑其具体实现，以保证并行应用程序的可移植性；
- **MPI 的具体实现**：消息传递库函数，目前有影响的为 MPICH 和 LAMMPI，我们注重 MPICH 系列；

6．基于消息传递的并行程序执行模式：

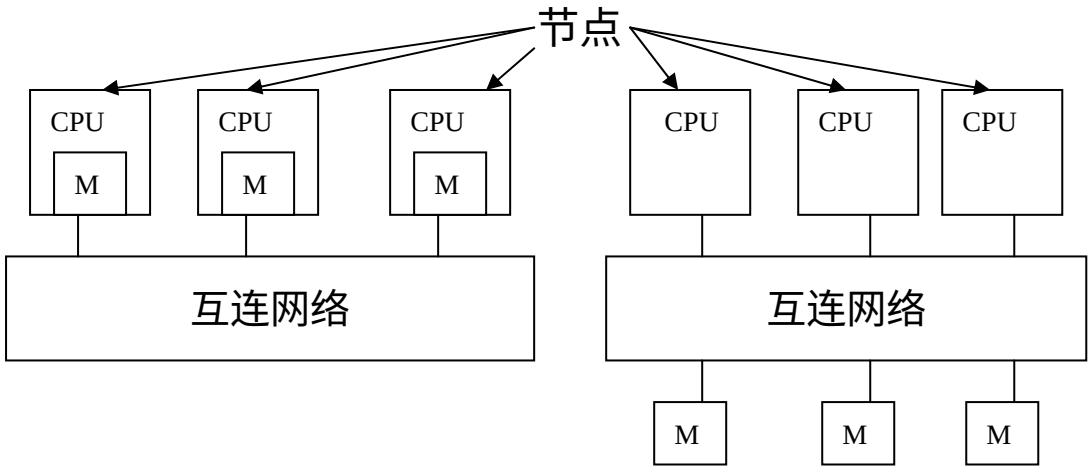
- SPMD 模式：单程序多数据流



- MPMD 模式：多程序多数据流，除初始启动多个可执行代码，其余与 SPMD 模式一致；

7．共享存储与分布式存储：

- 属于并行机体系结构的范畴，与消息传递并行程序设计平台无关；



分布式存储　　共享存储

- 消息传递是相对于**进程间通信方式**而言的，与具体并行机存储模式无关，任何支持进程间通信的并行机，均可支持消息传递并行程序设计；
- 几乎所有共享和分布存储并行计算环境均支持进程间的消息传递通信；

三、MPI 环境的应用现状

- MPI（消息传递界面）是全球工业、政府和科研部门联合推出的适合进程间进行标准消息传递的并行程序设计平台，最初版 MPI 1.0 本于 1994 年 6 月推出，目前最新的为 MPI 2.0 版，于 1998 年年低推出；
- MPI 的具体实现：MPICH 和 LAMMPI，目前均已实现 MPI 1.2 版，适用于任何并行计算平台；部分并行机已实现 MPI 2.0 版；
- MPI 是目前应用最广的并行程序设计平台，几乎被所有并行计算环境（共享和分布式存储并行机、MPP、机群系统等）和流行的多进程操作系统（UNIX、Windows NT）所支持，基于它开发的应用程序具有最佳的可移植性;
- 目前高效率的超大规模并行计算（1000 个处理器）最可信赖的平台；
- 工业、科学与工程计算部门的大量科研和工程软件（气象、石油、地震、空气动力学、核等）目前已经移植到 MPI 平台，发挥了重要作用；
- 目前，MPI 相对于 PVM：
  - 优点：功能强大，性能高，适应面广，使用方便，可扩展性好；
  - 缺点：进程数不能动态改变；

四、MPI 并行程序设计入门

1．MPI 并行程序设计平台由**标准消息传递函数及相关辅助函数**构成，多个进程通过调用这些函数（类似调用子程序），进行通信；

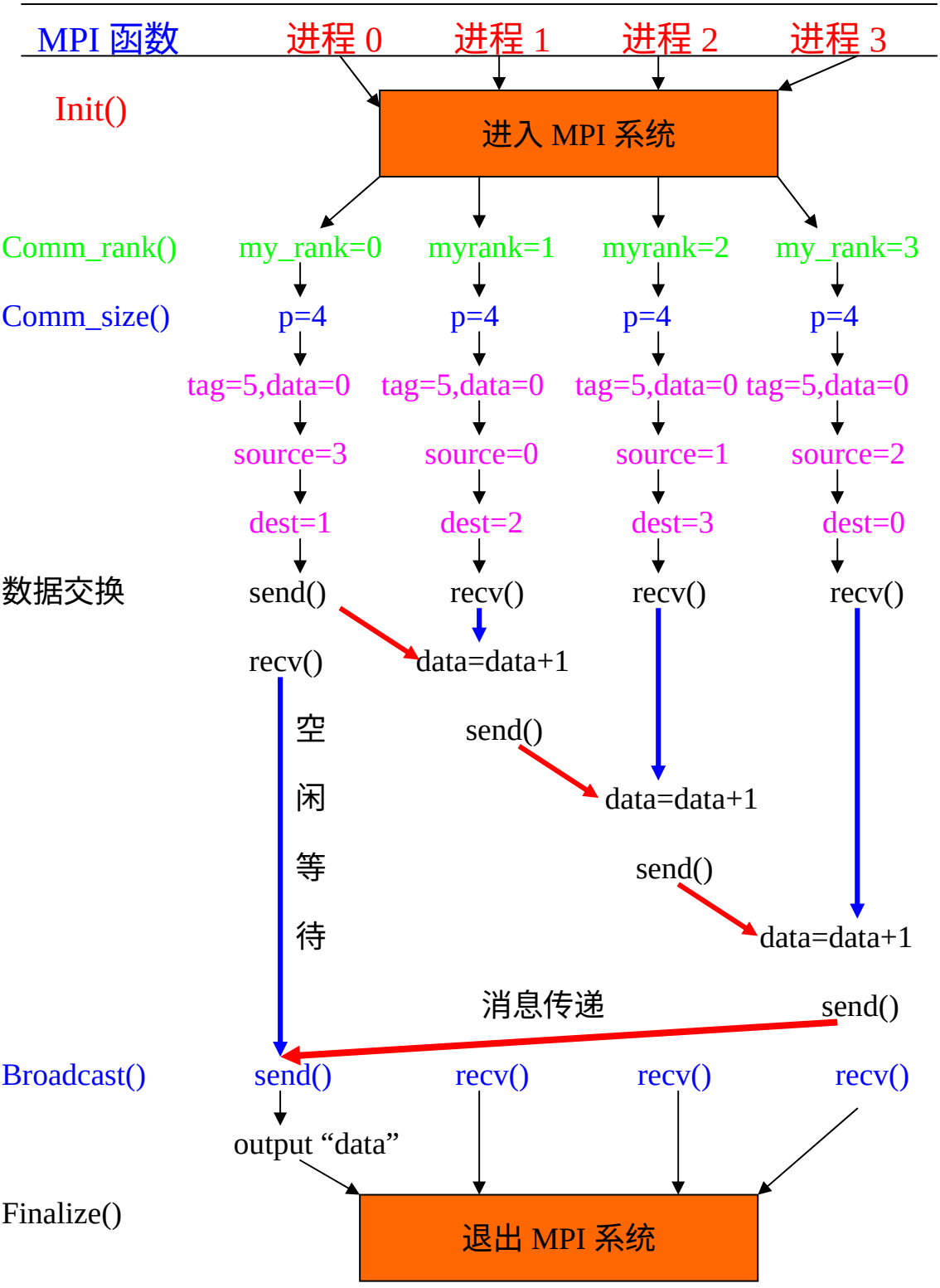
2．MPI 程序：

- SPMD 执行模式：一个程序同时启动多份，形成多个独立的进程，在不同的处理机上运行，拥有独立的内存空间，进程间通信通过调用 MPI 函数来实现；
- 每个进程开始执行时，将获得一个唯一的序号（rank）。例如启动 P 个进程，序号依次为 0，1，…，P-1；
- **MPI 程序例 1**：进程 0 发送一个整数给进程 1；进程 1 将该数加 1，传递给进程 2；进程 2 再将该数加 1，再传递给进程 3；依次类推，最后，进程 P-1 将该数传递给进程 0，由进程 0 负责广播该数给所有进程，并打印输出。

```
      program    example1
      include    “mpif.h”          !! MPI 系统头文件
      integer    status(MPI_STATUS_SIZE),my_rank,p,source,dest,tag,ierr,data
c
c-------进入 MPI 系统
      call MPI_Init(ierr)
      call MPI_Comm_rank(MPI_COMM_WORLD,my_rank,ierr)
      call MPI_Comm_size(MPI_COMM_WORLD,p,ierr)
c
c-------数据交换
      data=0
      tag    = 5
      source= my_rank-1
      if(source.eq.-1) source=p-1
      dest =my_rank+1
      if(dest.eq.p) dest=0

      if(my_rank.eq.0) then
         call MPI_Send(data,1,MPI_INTEGER,dest,tag,MPI_COMM_WORLD,ierr)
         call MPI_Recv(data,1,MPI_INTEGER,source,tag,MPI_COMM_WORLD,status,ierr)
      else
         call MPI_Recv(data,1,MPI_INTEGER,source,tag,MPI_COMM_WORLD,status,ierr)
         data=data+1
         call MPI_Send(data,1,MPI_INTEGER,dest,tag,MPI_COMM_WORLD,ierr)
      endif
c
c-------广播数据
      call MPI_Bcast(data,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
c
c------打印输出
      if(my_rank.eq.0) then
         if(data.eq.p-1) then
            print *,”Successful, data=”,data
         else
            print *,”Failure, data=”,data
         endif
      endif
c
       call MPI_Finalize(ierr)
       end
```

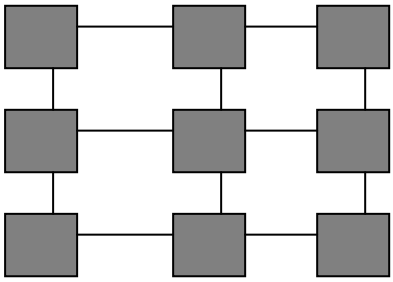
- 编译命令:　f77　-o　exam.e　example.f -lmpi
- 运行命令:　mpirun –np　4　exam.e
- 运行效果：MPI 系统选择相同或不同的 4 个处理机，在每个处理机上运行程序代码 exam.e。



3．MPI 重要概念

- **进程序号**（rank）；各进程通过函数 MPI_Comm_rank()获取各自的序号;
- **消息号**：消息的标号；
- **通信器**(Communicator)：1)理解为一类进程的集合，且在该集合内，进程间可以相互通信；类比：邮局、电话局、国际网；2)任何 MPI 通信函数均必须在某个通信器内发生；3) MPI 系统提供省缺的通信器MPI_COMM_WORLD,所有启动的MPI进程通过调用函数 MPI_Init()包含在该通信器内；4)各进程通过函数 MPI_Comm_size()获取通信器包含的(初始启动)的 MPI 进程个数;
- **消息**：分为**数据（data）**和**包装（envelope）**两个部分，其中，包装由**接收进程序号**、**发送进程序号**、**消息标号**和**通信器**四部分组成，数据包含用户将要传递的内容；
- **进程组**：一类进程的集合，在它的基础上，可以定义新的通信器；
- **基本数据类型**：对应于 FORTRAN 和 C 语言的内部数据类型（INTEGER，REAL，DOUBLE PRECISION，COMPLEX，LOGICAL，CHARACTER），MPI 系统提供已定义好的对应数据类型（MPI_INTEGER，MPI_REAL，MPI_DOUBLE_PRECISION，MPI_COMPLEX，MPI_LOGICAL，MPI_CHARACTER）；

- **自定义数据类型：**基于基本数据类型，用户自己定义的数据类型（后面介绍）；
- **MPI 对象:** MPI 系统内部定义的数据结构，包括数据类型、进程组、通信器等，它们对用户不透明，在 FORTRAN 语言中，所有 MPI 对象均必须说明为“**整型变量 INTEGER**”；
- **MPI 联接器**（handle）：联接 MPI 对象和用户的桥梁，用户可以通过它访问和参与相应 MPI 对象的具体操作；例如，MPI 系统内部提供的通信器 MPI_COMM_WORLD；在 FORTRAN 语言中，所有 MPI 联接器均必须说明为‘**整型变量 INTEGER**”；
- **进程拓扑结构**：进程组内部进程之间的一种相互连接结构，如 3×3 网格，将在后面介绍。



3 × 3 网格拓扑结构

- **静态进程个数**：进程数由命令“ mpirun –np xxx ”初始确定为 xxx 个，程序执行过程中**不能动态改变**进程的个数;
- **消息缓存区**：应用程序产生的消息包含的数据所处的内

存空间；

- **标准输入**:所有进程的标准输入 read(*,*)均省缺为当前终端屏幕，且只能由 0 号进程执行该操作，其他进程需要这些输入参数，只能由 0 号进程执行数据广播操作；
- **标准输出**:所有进程可以独立执行标准输出 write（*，*），但其省缺为当前终端屏幕；

4．MPI 函数格式：

- FORTAN 语言中，最后一个参数为该函数调用是否成功的标志：0 表示成功，其它表示各种可能的错误；
- C 语言中，该标志又函数参数返回；

  C ： ierr=MPI_Comm_rank(myrank)

  F : MPI_Comm_rank(myrank,ierr)

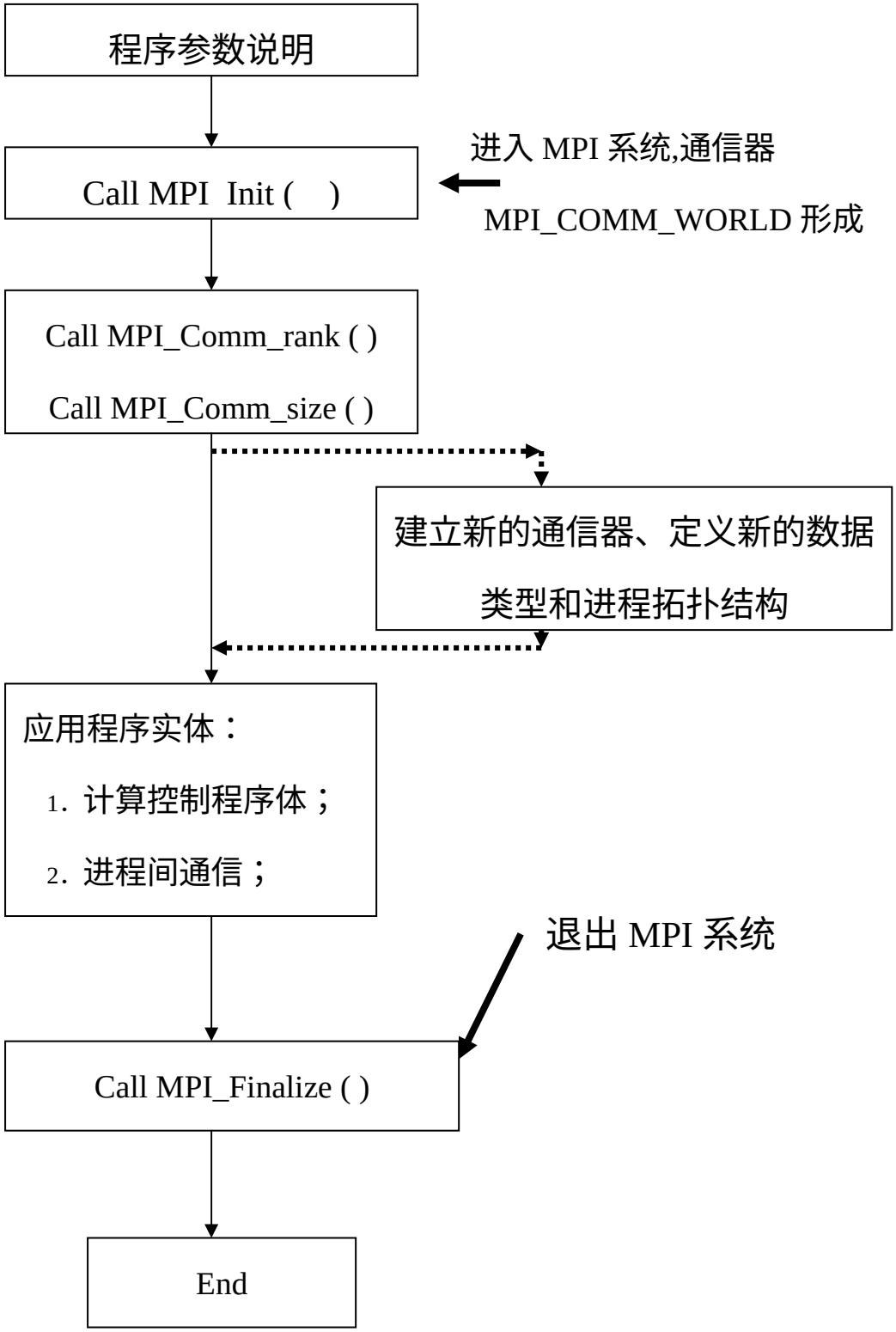
5. MPI 函数的使用查询:

- 由函数名查询： man 函数名 （ MPI_Xxxx），注意大小写，例如 man MPI_Comm_rank, man MPI_Send, man MPI_recv.

6. MPI 函数的学习与使用：

- 注重 MPI 函数的各类功能，由应用程序的通信需求出发，寻找匹配的函数类型，在查找具体函数名，采用 man 命令可以查询该函数的具体参数含义和使用方法。

- 7. 一般的 MPI 程序设计流程图

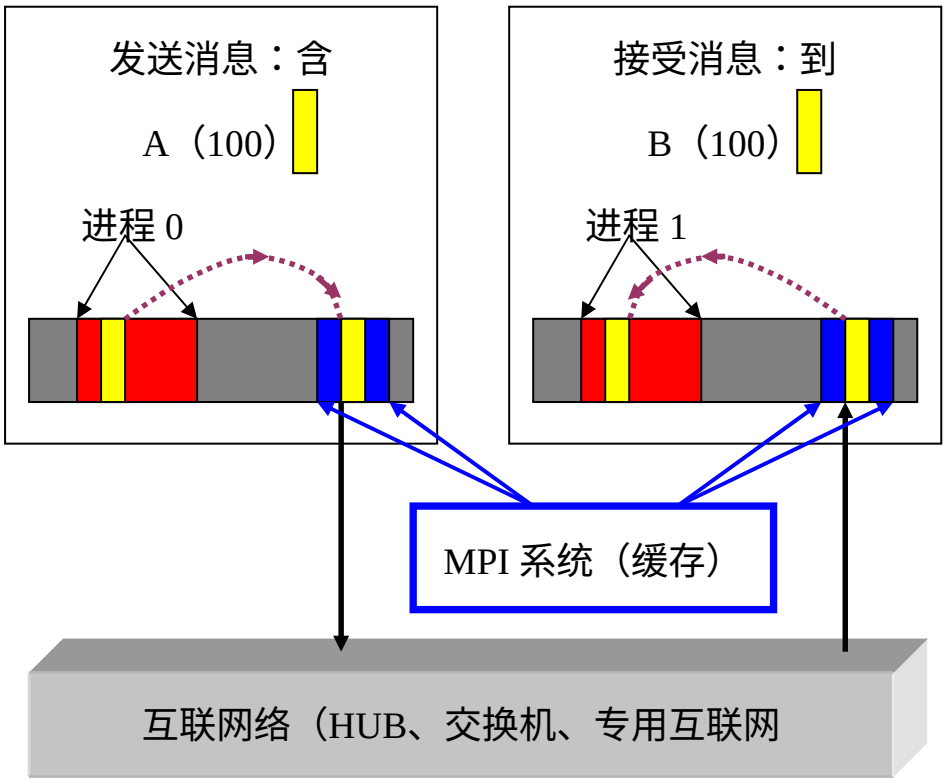


五、MPI 基本函数

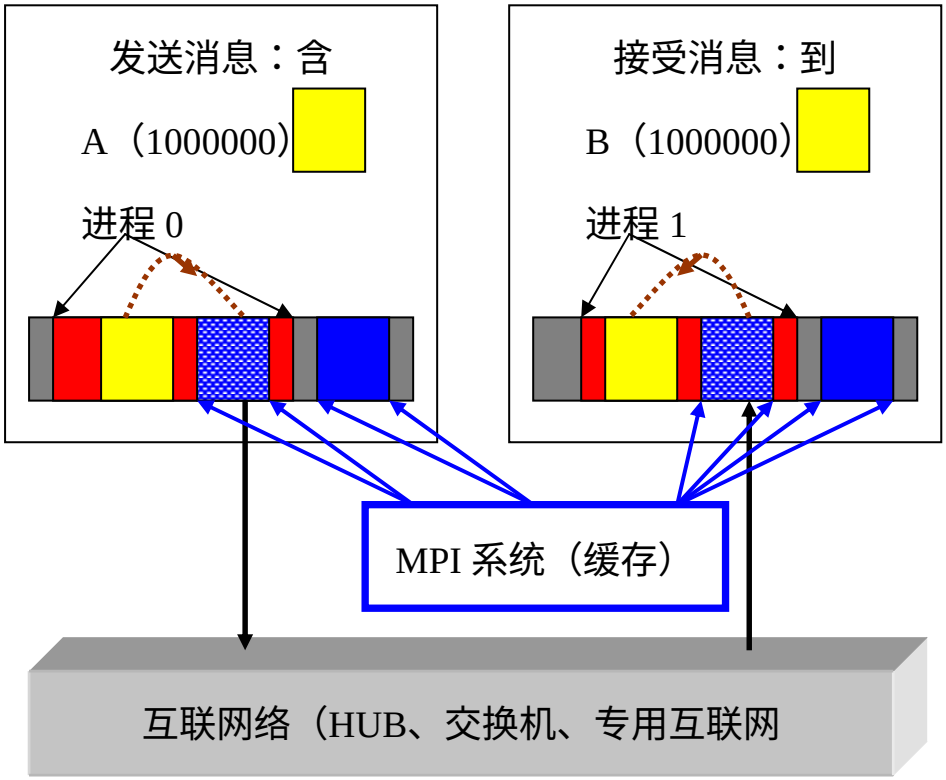
1．点对点通信（point-to-point）

- 定义：给定属于同一通信器内的两个进程，其中一个发送消息，一个接收消息；
- MPI 系统定义的所有通信方式均建立在点对点通信之上；
- 四种模式：**标准模式**、**缓存区模式**、同步模式、就绪模式；
- 原理示意图：

标准模式：**MPI** 系统内部提供对消息的缓存

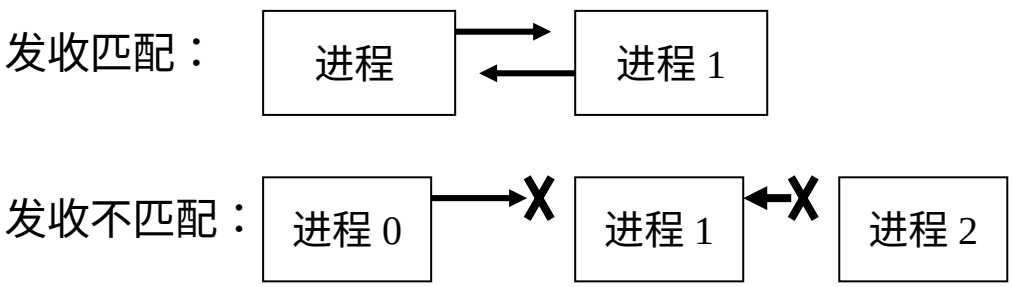


缓存区模式：大数据量消息传递（>16KB），超过 MPI 系统缓存区容量，应用程序将部分内存空间提供给 **MPI** 系统，充当消息缓存区



2．标准模式点对点通信

- 进程可以随意地发送（接收）消息，与是否存在匹配的消息接收（发送）进程无关；



- 两类：
  - **阻塞式**：消息发送函数返回，用户可以对消息缓存区进行处理，不会影响已发送的消息数据；接受函数返回，用户可以使用接受到的消息数据；
  - 非阻塞式：发送和接受函数返回后，必须调用另一类函数来确保它们的正确完成；

	阻塞式	非阻塞式
	INTEGER A A=100 MPI_Send(A,1,….) A=200	INTEGER A A=100 MPI_Isend(A,1,…) A=200
消息数据:	A=100	A=100 或 A=200
		MPI_Isend(A,1,…flag,…) MPI_Wait(flag,…) A=200
消息数据:	A=100	A=100

3．点对点通信函数举例

- 阻塞式标准消息发送函数

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

Real*8（integer，…）buf ：消息发送缓存区起始地址

(Fortran, 用户的待发送的第一个数据)

integer count ：buf 起始的数据单元个数

integer datatype ：数据类型（基本或用户定义的）

integer dest ：接收进程序号

integer tag ：消息的标号

integer comm ： 通信器

integer ierr ： 函数调用返回错误码

```
      real *8    a(100,100)
      integer    b(60,60)
c-----发送 50 个双精度数 “ a(5,20) : a(54,20)”到 2 号进程
      call MPI_Send( a (5,20),50,MPI_DOUBLE_PRECISION,2,
     &                    99999,MPI_COMM_WORLD,ierr )
c-----发送 20 个整型数 “ b(20,40) : b(39,40)”到 5 号进程
      call MPI_Send( b (20,40),20,MPI_INTEGER,5,
     &                    6666,MPI_COMM_WORLD,ierr )
```

- 阻塞式标准消息接收函数

MPI_Recv（buf，count，datatype，dest，tag，comm，status, ierr）

Real*8（integer，…）buf ：消息接收缓存区起始地址

(Fortran, 用户用于接受的第一个数据)

integer count ：buf 起始的数据单元个数

integer datatype ：数据类型（基本或用户定义的）

integer dest ： 发送进程序号

integer tag ： 消息的标号

integer comm ： 通信器

integer status(MPI_STATUS_SIZE)： 接收状态数组;

integer ierr ： 函数调用返回错误码

```
      real *8    a(100,100)
      integer    b(60,60)
c-----从 2 号进程接收 50 个双精度数到 “ a(5,20) : a(54,20)”
      call MPI_Recv( a (5,20),50,MPI_DOUBLE_PRECISION,2,
     &                    99999,MPI_COMM_WORLD,status,ierr )
c-----从 5 号进程接收 20 个整型数到 “ b(20,40) : b(39,40)”
      call MPI_Recv( b (20,40),20,MPI_INTEGER,5,
     &                    99999,MPI_COMM_WORLD,status,ierr )
```

- 其他点对点通信函数：参考手册；

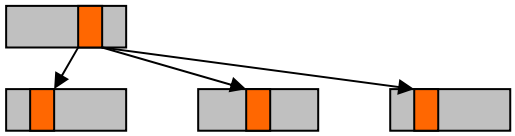
4．聚合通信（Collective Communication）

- 定义：属于同一通信器的所有 MPI 进程均必须参与的通信操作；
- 参与方式：调用同一聚合通信函数；
- 函数类型：
  - 同步通信函数：所有进程在某个程序点上同步；

    MPI_Barrier (comm , ierr)
  - 全局通信函数:
    - 广播: MPI_Bcast(buf,count,dtype,**root**,comm,ierr)

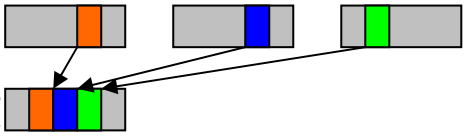
      root 发送

      other 接受

      
    - 收集：MPI_Gather(bufs,bufr,count,dtype,root,comm,ierr)

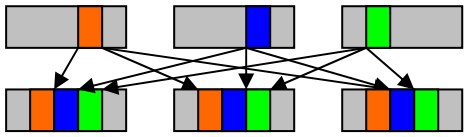
      all 发送大小一致的数据块

      root 接受并按序号连续存放

      
    - 全收集：MPI_Allgather()

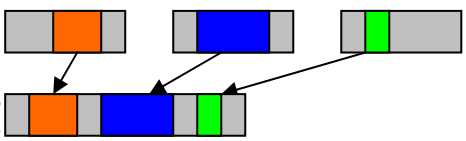
      all 发送

      all 接受

      
    - 索引收集:MPI_Gatherv()

      all 发送大小不等的数据块

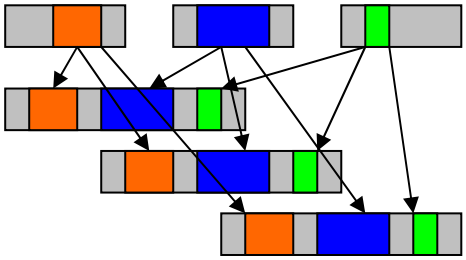
      root 接受并按索引间断存放

- 索引全收集:MPI_Allgatherv()

  all 发送大小不等的数据块

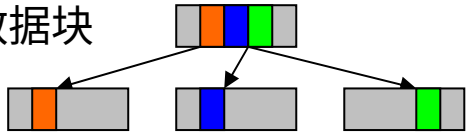
  All 接受并按索引间断存放



- 分散:MPI_Scatter(bufs,bufr,count,dtype,root,comm,ierr)

  root 发送连续的大小一致数据块

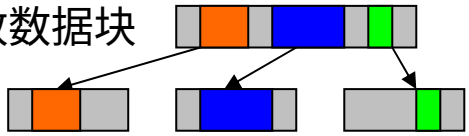
  All 接受



- 索引分散: MPI_Scatterv()

  root 发送间断的大小不一致数据块

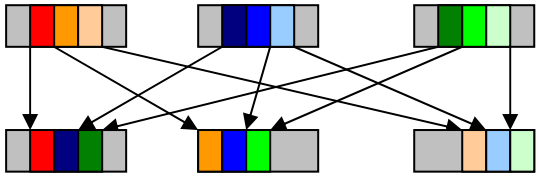
  All 接受



- 全交换: MPI_Alltoall()

  All 发送大小一致数据块到各进程

  All 接受大小一致数据块并按序号连续存放



- 索引全交换: MPI_Alltoallv()

➢ 全局规约(global reduction)函数:

- 规约: MPI_Reduce(sbuf,rbuf,count,dtype,**op**,**root**, comm,ierr);

  规约操作类型 **op** : MPI_SUM, MPI_MIN, MPI_MAX, MPI_PROD 等 12 种;

例子: 求两个向量的内积,并将结果返回进程 0

```
      subroutine par_blas1(m,a,b,c,comm)
      real    a(m),b(m)          ! local slice of array
      real    c                  !    result
      real    sum
      integer    m,comm,i,ierr
c
c     !local sum
        sum=0.0d0
        do    i=1,m
              sum=sum+a(i)*b(i)
        enddo
c     ! global sum
      call MPI_Reduce(sum,c,1,MPI_REAL,MPI_SUM,0,
     &                comm,ierr)
```

- 全规约: MPI_Allreduce(), 除要求将结果返回到所有进程外,与 MPI_Reduce()一致;
- 规约分散: MPI_Reduce_scatter(),将规约结果分散到各进程;
- 并行前缀计算:MPI_Scan()

六、作业一

P 个进程，第 *i* 个进程将其包含 100 个双精度数据的数组 A（100）传送给第（*i* +1）mod *P* 个进程，同时从第（*i* -1）mod *P* 个进程接受 100 个双精度数据到另一个数组 B（100）中，令数组 C（1：100）=A（1：100）+B（1：100），然后求数组 C（1：100）各元素的类加和，最后在将该和全部累加到 0 号进程，打印并输出该和。

提示：可在例 1 的基础上修改，编制。

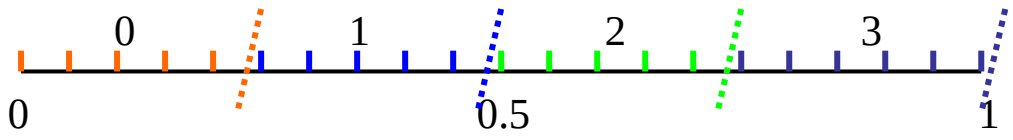
七、MPI 并行程序示例 2

1．一维 Dirichlet 问题：

$$\begin{cases} -\Delta u(x) = f(x) & x \in \Omega = (0,1) \\ u(0)=0.0,\ u(1)=0.0,\ f(x)=4.0 \end{cases}$$

算法:　　均匀网格有限差分离散,Jacobi 迭代求解。

区域分解：nproc=4, n = 21, ns = (n-1)/nproc+1 =6




```
      program    example
      implicit real*8 (a-h,o-z)
      include “mpif.h”
      parameter ( n = 21, nproc = 4, ns = (n-1)/nproc+1)
      parameter ( errtol=1.e-4,nitmax=100)
      dimension    u(ns), f(ns), solution(ns),uold(0:ns+1)
                   ! uold(0)       :    left dummy grid point
                   ! uold(ns+1) :      right dummy grid point
      integer      status(MPI_STATUS_SIZE),size
c
c          enter into MPI
           call    MPI_Init(ierr)
           call    MPI_Comm_rank(MPI_COMM_WORLD,myrank,ierr)
           call    MPI_Comm_size(MPI_COMM_WORLD,size,ierr)
           if(size.ne.nproc) then
              print *,”+++ errors for number of process distributions +++”
              goto 888
           endif
c
c          assign initial values
           h    = 1.0d0/(n-1)                    !    discrete step size
           xst = myrank*(ns-1)*h          !    start x-coordinate of local domain
           nlocal=ns-1                                    ! local number of grid point
```

```
      ist=1
      if(myrank.eq.0) ist=ist+1
      do i=1,ns
         f(i)=4.0d0
         u(i)=0.0d0
         uold(i)=0.0d
         xauxi=xst+(i-1)*h
         solution(i)= -2*xauxi*(xauxi-1.0d0)
      enddo
      uold(0)=0.0d0
c
      nlp=myrank-1              ! left process
      nrp=myrank+1              ! right process
      if(nlp.lt.0)           nlp=MPI_PROC_NULL
      if(nrp.gt.nproc-1)    nrp=MPI_PROC_NULL
                        ! null process, null communication operation
c
c     Jacobi iterating
      nit    =0                          ! current iterations
c
   10 continue
c     swapping dummy elements along pseudo-boundary
      call MPI_Send(uold(1),1,MPI_DOUBLE_PRECISION,
     &              nlp,nit, MPI_COMM_WORLD,ierr)
      call MPI_Send(uold(nlocal),1, MPI_DOUBLE_PRECISION,
     &              nrp,nit,MPI_COMM_WORLD,ierr)
      call MPI_Recv(uold(nlocal+1),1,MPI_DOUBLE_PRECISION,
     &              nrp,nit, MPI_COMM_WORLD,status,ierr)
      call MPI_Recv(uold(0),1, MPI_DOUBLE_PRECISION,
     &              nlp,nit, MPI_COMM_WORLD,status, ierr)

c     iterating and convergenc checking
      error=0.0d0
      do    i=ist,nlocal
            u(i)=(h*h*f(i)+uold(i-1)+uold(i+1))/2
            xauxi= dabs(u(i)-solution(i))
            if(dabs(xauxi).gt.error)    error=xauxi
      enddo
c
c     maximum error
      call MPI_Allreduce(error,xauxi,1,MPI_DOUBLE_PRECISION,MPI_MAX,
     &                   MPI_COMM_WORLD,ierr)
      error=xauxi
      if(error.gt.errtol) then
            do i=ist,nlocal
```

```
                uold(i)=u(i)
              enddo
              if(nit.lt.nitmax) then
                nit=nit+1
                if(myrank.eq.0) print *,"nit=",nit," error=",error
                goto 10
              endif
            endif
c
            if(myrank.eq.0) then
              if(nit.le.nitmax.and.error.le.errtol) then
                write(*,100) nit
 100            format(1x, “Successfully touch the exact solution after nit = ”,
     &               i4, “  iterations”)
              else
                write(*,200) nit
 200            format(1x, “Fail to touch the exact solution after nit = ”,
     &               i4, “  iterations”)
              endif
            endif
c
  888   call MPI_Finalize(ierr)
c
        end
```

---

1. 编译

LINUX : mpif77 –o exam.e example.f

2．运行

mpirun -np 4 exam.e

八、作业二

（任意选择其中之一）

1．矩阵乘：C（M，L）=A（M，N）*B（N，L）

数据分割：

- 矩阵 A，C ： 按行等分块存储于各进程中；
- 矩阵 B ： 按列等分块存储于各进程中；

算法：

- 初始矩阵形成 :各进程独立并行地对它所拥有的矩阵块元素按某种规律赋值；
- 矩阵 B 的各子块在各进程中间循环移动，并完成相应的矩阵子块乘操作；
- 结果：矩阵 C 按行存储在各进程中，并与预期正确结果比较，结果正确，赋标志为 0，否则为 1，并将这些标志规约到 0 号进程，由 0 号进程打印输出矩阵乘是否成功完成的信息。

注：如有可能，请比较调用和不调用 BLAS 库程序的并行性能差别。

2．求解二维规则区域上 Dirichlet 问题- Δ u = f，有限差分离散，沿两个方向的等网格区域分解，Jacobi 迭代求解。

九、MPI 先进函数

1．自定义数据类型

- 定义：在 MPI 系统已定义的基本数据类型（MPI_INTEGER,MPI_REAL,MPI_DOUBLE_PRECISION,MPI_CHARACTER 等）基础上，用户根据需求，自己定义的数据类型；

---

real a(1000)

发送： a(5:9)

call MPI_Send(a(5), 5, MPI_ REAL,…..) ← OK

发送： a(5),a(7),a(9),a(11),a(13),a(15)

```
do i=5, 15, 2
    call MPI_Send(a(i),1,MPI_REAL,….)   ← OK
enddo
```

缺点: 多次发送，效率低，程序设计繁琐

改进： 用户定义新的数据类型

```
call  MPI_Type_vector(6,1,2,MPI_REAL,newtype,ierr)
call  MPI_Type_commit(newtype , ierr)   ← 提交
call  MPI_Send(a(5), 1, newtype,….)
call  MPI_Type_free(newtype,ierr)       ← 释放
```

- 在用户已定义好的数据类型基础上，还可以进一步定义新的数据类型；
- 用户定义的数据类型，必须由函数

MPI_Type_Commit()提交给 MPI 系统；此后，就可以象基本数据类型一样，在消息传递函数中重复使用；并由函数 MPI_Type_free()释放；

- 具体自定义数据类型函数，请参考手册；

（按手册函数讲解）

/*　　自定义数据类型创建函数,创建一个新的数据类型,由连续个原数据类型组成.

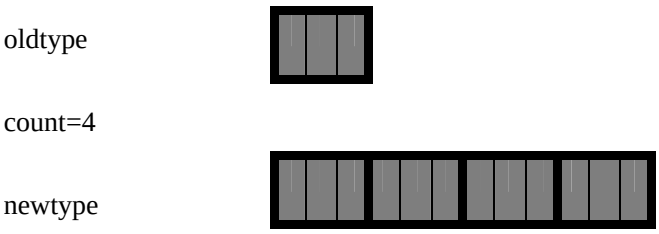
MPI_TYPE_CONTIGUOUS（count，oldtype，newtype）

IN	**count**	oldtype 数据单元个数
IN	**oldtype**	原数据类型
OUT	**newtype**	新数据类型

C　　int　MPI_Type_contiguous（int count，MPI_Datatype oldtype，
MPI_Datatype *newtype）

Fortran　MPI_TYPE_COTIGUOUS（COUNT，OLDTYPE，NEWTYPE，IERROR）
INTEGER COUNT，OLDTYPE，NEWTYPE，IERROR

MPI_TYPE_CONTIGUOUS是最简单的自定义数据类型创建函数。它将count个oldtype类型的数据（包括基本数据类型和已提交给 MPI 系统的自定义数据类型）复制到连续的位置，形成新的自定义数据类型 newtype。



函数 MPI_TYPE_CONTIGUOUS 示意图

/*　　自定义数据类型创建函数,创建一个新的数据类型,由间隔的多个数据块组成.

MPI_TYPE_VECTOR（count，blocklength，stride，oldtype，newtype）

IN	**count**	数据块个数
IN	**blocklength**	每数据块包含数据单元的个数
IN	**stride**	以数据单元为单位，连续两个数据块起始地址之间的间距
IN	**oldtype**	原数据类型
OUT	**newtype**	新数据类型

C　　int MPI_Type_vector（int count，int blocklength，int stride，MPI_Datatype oldtype，MPI_Datatype *newtype）

Fortran　　MPI_TYPE_VECTOR（COUNT，BLOCKLENGTH，STRIDE，OLDTYPE，NEWTYPE，IERROR）

INTEGER　COUNT，BLOCKLENGTH，STRIDE，OLDTYPE，NEWTYPE，IERROR

MPI_TYPE_VECTOR 创建的新数据类型 newtype 由 count=3 个块组成，其中每块含 blocklength=2 个 oldtype 数据单元,且连续块起始地址之间相差 stride=3 个 oldtype 数据单元。

oldtype 

count=3，blocklength=2，stride=3

newtype 

函数 MPI_TYPE_VECTOR 示意图

/*　　自定义数据类型创建函数,创建一个新的数据类型,由间隔的多个数据块组成.

MPI_TYPE_HVECTOR（count，blocklength，stride，oldtype，newtype）

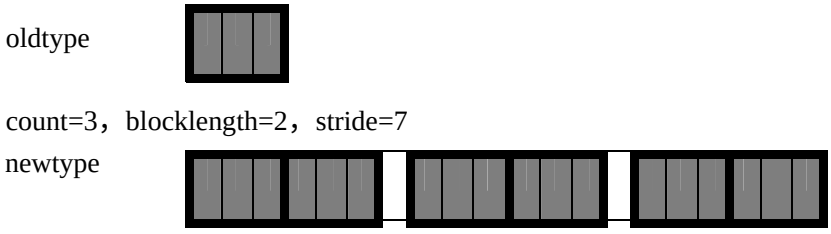
IN	**count**	数据块个数
IN	**blocklength**	每数据块包含数据单元个数
IN	**stride**	以字节为单位，连续两个数据块起始地址之间的间距
IN	**oldtype**	原数据类型
OUT	**newtype**	新数据类型

C　　int MPI_Type_hvector（int count，int blocklength，int stride，MPI_Datatype oldtype，MPI_Datatype *newtype）

Fortran　　MPI_TYPE_HVECTOR（COUNT，BLOCKLENGTH，STRIDE，OLDTYPE，NEWTYPE，IERROR）

INTEGER　COUNT，BLOCKLENGTH，STRIDE，OLDTYPE，NEWTYPE，IERROR

MPI_TYPE_HVECTOR 与 MPI_TYPE_VECTOR 类似，唯一不同的是，连续两个数据块起始地址之间的间距不是以 oldtype 数据单元为单位，而是以字节为单位，以便更紧凑地使用内存空间。



函数 MPI_TYPE_HVECTOR 示意图

矩阵转置： B（100，100）=A（100，100）^{T}。

首先，我们以行序（row-major）创建一个描述矩阵单元分布的数据类型；然后，发送该数据类型，并以列序（column-major）接收。具体程序如下：

```
      REAL a（100,100）, b（100,100）
      INTEGER row,xpose,sizeofreal,myrank,ierr
      INETGER status（MPI_STATUS_SIZE）
C
C     transpose matrix a into b
      CALL  MPI_COMM_RANK（MPI_COMM_WORLD, myrank, ierr）
      CALL  MPI_TYPE_EXTENT（MPI_REAL, sizeofreal, ierr）
C
C     create datatype for one row
C     （vector with 100 real entries and stride 100）
      CALL  MPI_TYPE_VECTOR（100, 1, 100, MPI_REAL, row, ierr）
C
C     create datatype for matrix in row-major order
C     （ one hundred copies of the row datatype, strided one word
C     apart; the successive row datatypes are interleaved ）
      CALL  MPI_TYPE_HVECTOR（100,  1,  sizeofreal, row, xpose, ierr）
      CALL  MPI_TYPE_COMMIT（xpose,ierr）
C
C     send matrix in row-major order and receive in column major order
      CALL  MPI_SENDRECV（a, 1, xpose, myrank, 0, b, 100*100, MPI_REAL,
     &                    myrank, 0, MPI_COMM_WORLD, status, ierr）
```

/*　　自定义数据类型创建函数,创建一个新的数据类型,由自索引的多个数据块组成.

MPI_TYPE_INDEXED（count，array_of_blocklengths，
array_of_displacements，oldtype，newtype）

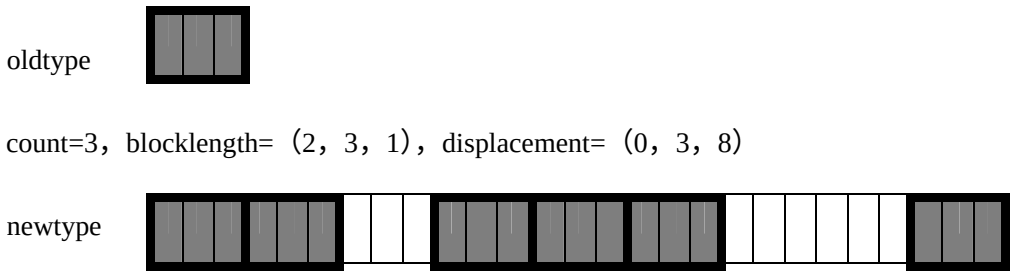
IN	**count**	数据块个数
IN	**array_of_blocklengths**	数组，含每数据块拥有的单元个数
IN	**array_of_displacements**	数组，含每数据块的初始位置（单元为单位）
IN	**oldtype**	原数据类型
OUT	**newtype**	新数据类型

C　　int MPI_Type_indexed（int count，int *array_of_blocklengths，int *array_of_displacements，MPI_Datatype oldtype，MPI_Datatype *newtype）

Fortran　MPI_TYPE_INDEXED（COUNT，ARRAY_OF_BLOCKLENGTHS，ARRAY_OF_DISPLACEMENTS，OLDTYPE，NEWTYPE，IERROR）

INTEGER　COUNT，ARRAY_OF_BLOCKLENTHS（*），ARRAY_OF_DISPLACEMENTS（*），OLDTYPE，NEWTYPE，IERROR

MPI_TYPE_INDEXED 更自由地复制 oldtype 类型数据到 count 个数据块中，形成自定义数据类型 newtype。其中，array-of-displacements（i）和 array-of-blocklengths（i）以 oldtype 单元为单位，分别给出第 i 个数据块的起始地址和大小。



函数 MPI_TYPE_INDEXED 示意图

/*　　自定义数据类型创建函数,创建一个新的数据类型,由自索引的多个数据块组成.

MPI_TYPE_HINDEXED（count，array_of_blocklengths，
array_of_displacements，oldtype，newtype）

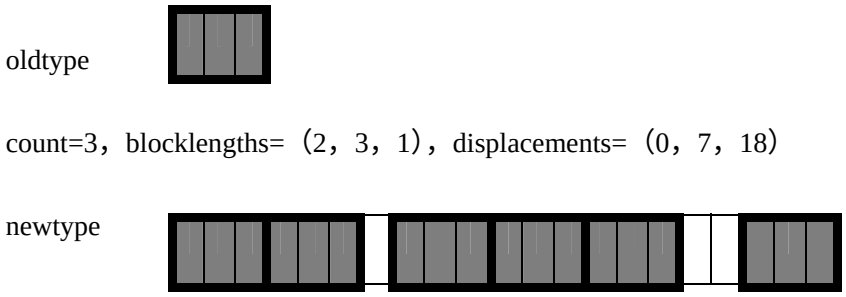
IN	**count**	数据块个数
IN	**array_of_blocklengths**	数组，包含每数据块拥有的数据单元个数
IN	**array_of_displacements**	数组，包含每数据块的初始位置（字节为单位）
IN	**oldtype**	原数据类型
OUT	**newtype**	新数据类型

C　　int MPI_Type_hindexed（int count，int *array_of_blocklengths，int *array_of_displacements，MPI_Datatype oldtype，MPI_Datatype *newtype）

Fortran　MPI_TYPE_HINDEXED（COUNT，ARRAY_OF_BLOCKLENGTHS，ARRAY_OF_DISPLACEMENTS，OLDTYPE，NEWTYPE，IERROR）

INTEGER　COUNT，ARRAY_OF_BLOCKLENTHS（*），ARRAY_OF_DISPLACEMENTS（*），OLDTYPE，NEWTYPE，IERROR

除了 array-of-placements 中元素以字节为单位，MPI_TYPE_HINDEXED 与 MPI_TYPE_INDEXED 完全一致。



函数 MPI_TYPE_HINDEXED 示意图

/*　　自定义数据类型创建函数,创建一个新的数据类型,由多种不同类型数据组成.

MPI_TYPE_STRUCT（count，array_of_blocklengths，
array_of_displacements，array_of_types，newtype）

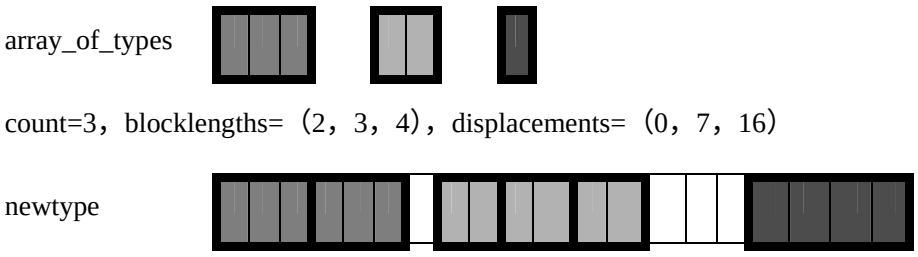
IN	**count**	数据块个数
IN	**array_of_blocklengths**	数组，包含每数据块拥有的数据单元个数
IN	**array_of_displacements**	数组，包含以字节为单位的每数据块初始位置
IN	**array_of_types**	数组，包含每数据块拥有的数据单元类型
OUT	**new datatype**	新数据类型

C　　int MPI_Type_struct（int count，int *array_of_blocklengths，MPI_Aint *array_of_displacements，MPI_Datatype *array_of_types，MPI_Datatype *newtype）

Fortran　　MPI_TYPE_STRUCT（COUNT，ARRAY_OF_BLOCKLENGTHS，ARRAY_OF_DISPLACEMENTS，ARRAY_OF_TYPES，NEWTYPE，IERROR）

INTEGER　COUNT，ARRAY_OF_BLOCKLENGTHS（*），ARRAY_OF_DISPLACEMENTS（*），ARRAY_OF_TYPES（*），NEWTYPE，IERROR

MPI_TYPE_STRUCT 是最普遍的自定义类型创建函数,它推广 MPI_TYPE_HINDEXED 的功能，允许每个数据块复制不同的数据类型。



函数 MPI_TYPE_STRUCT 示意图

2．数据的封装与拆卸

MPI 系统提供了函数 MPI_PACK，将多个不同类型和数量的数据单元***封装*** 到一个显示的应用程序提供的内存空间（缓存区）中，形成一个数据单元，其类型为 MPI_PACKED，充当 MPI 任何通信函数的数据类型参数。因此，我们可以通过该函数，将大量零碎的数据封装在一个消息中，参与消息传递，提高并行计算性能。

反之，MPI 系统也提供了函数 MPI_PACK 的逆函数 MPI_UNPACK，它能***拆卸*** 一个接收到的类型参数为 MPI_PACKED 的消息，即解散该消息，并将消息包含的零碎数据存储到不同的内存空间。

/*　　数据封装函数，封装数据到应用程序显示提供的缓存区中。

MPI_PACK（inbuf，incount，datatype，outbuf，outsize，position，comm）

IN	**inbuf**	输入缓存区起始地址
IN	**incount**	输入单元个数
IN	**datatype**	输入单元数据类型
OUT	**outbuf**	输出缓存区起始地址
IN	**outsize**	输出缓存区大小（字节为单位）
INOUT	**position**	输出缓存区当前位置指针（字节为单位）
IN	**comm**	通信器

C　　int　MPI_Pack（void* inbuf，int incount，MPI_Datatype datatype，void *outbuf，int outsize，int *position，MPI_Comm comm）

Fortran　MPI_PACK（INBUF，INCOUNT，DATATYPE，OUTBUF，OUTSIZE，POSITION，COMM，IERROR）

<type>　INBUF（*），OUTBUF（*）

INTEGER　INCOUNT，DATATYPE，OUTSIZE，POSITION，COMM，IERROR

MPI_PACK 从位置 position（字节为单位）开始，将输入数据缓存区（inbuf，incount，datatype，comm）封装到应用程序显示提供的缓存区（outbuf，outsize）中，且令 position=position+extent（datatype）×incount。其中，输入缓存区可为 MPI_SEND 所允许的任何消息缓存区，输出缓存区为包含 outsize 个字节的连续内存空间，且两个缓存区互不相交。

消息发送和接受函数调用形式：

MPI_SEND(**outbuf , 1 , MPI_PACKED**, dest,tag,comm,ierr)

MPI_RECV(**recvbuf, 1, MPI_PACKED**, src,tag,comm,status,ierr)

/*　　消息拆卸函数，从应用程序显示提供的缓存区中拆卸数据到各接收缓存区。

MPI_UNPACK（inbuf，insize，position，outbuf，outcount，datatype，comm）

IN	**inbuf**	输入缓存区起始地址
IN	**insize**	输入缓存区大小（以字节为单位）
INOUT	**position**	拆卸的数据在输入缓存区的起始位置（以字节为单位）
OUT	**outbuf**	输出缓存区起始地址
IN	**outcount**	被拆卸的数据单元个数
IN	**datatype**	每个拆卸数据单元的类型
IN	**comm**	通信器

C　　int　MPI_Unpack（void* inbuf，int insize，int *position，void *outbuf，int　outcount，MPI_Datatype datatype，MPI_Comm comm）

Fortran　MPI_UNPACK（INBUF，INSIZE，POSITION，OUTBUF，OUTCOUNT，DATATYPE，COMM，　IERROR）

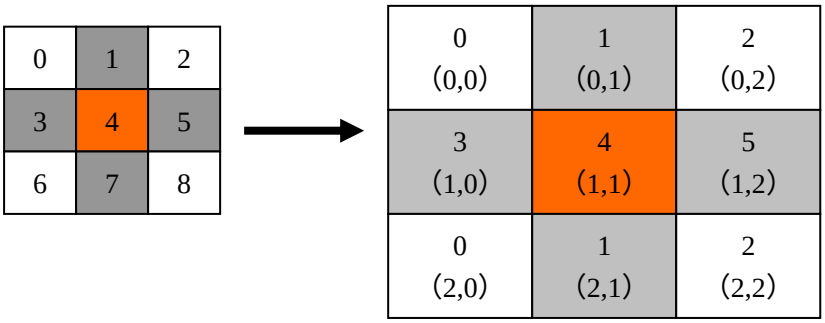
<type>　INBUF（*），OUTBUF（*）

INTEGER　INCOUNT，DATATYPE，OUTSIZE，POSITION，COMM，IERROR

MPI_UNPACK 是函数 MPI_PACK 的逆。在 MPI 系统中，任何消息均可用数据类型 MPI_PACKED 来接收，然后调用 MPI_UNPACK 拆卸消息中数据到各接收缓存区，且每次拆卸完毕，令 position=position+extent（datatype）×outcount。

3．进程拓扑结构

- 定义：根据应用程序的特征，在进程间建立的一种虚拟拓扑连接方式，以方便并行程序设计和提高并行计算性能；
- 例：二维规则区域，3*3 区域分解，9 个进程，建立 Cartesion 坐标，进程（i,j）的相邻进程为 (i-1,j), (i+1,j), (i,j-1), (i,j+1)；



3．并行 I/O

- 各进程可以类似于串行程序独立地读/写不同的文件；
- MPICH 1.2 以上版本支持所有进程并行读写同一个文件；

十、作业三

给定双精度型数组 A（M，N，L）和 B（L，N，M），要求使用 MPI “ 自定义数据类型函数 ” 和 “ 消息传递函数 ” 实现矩阵转置操作，即 $B=A^T$。

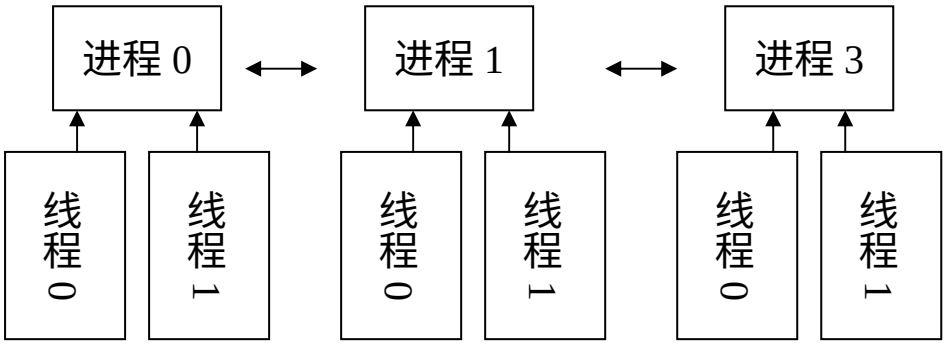
十一、MPI 环境的发展

1．版本更新：

- V 1.0⟶V1.2 ：已完成，增加了并行 I/O 功能；
- V1.2 ⟶ V2.0 ：正进行，将增加**进程数动态可调、进程间内存数据直接访问、多个独立的并行应用程序之间的动态互操作性等** 3 项主要功能；

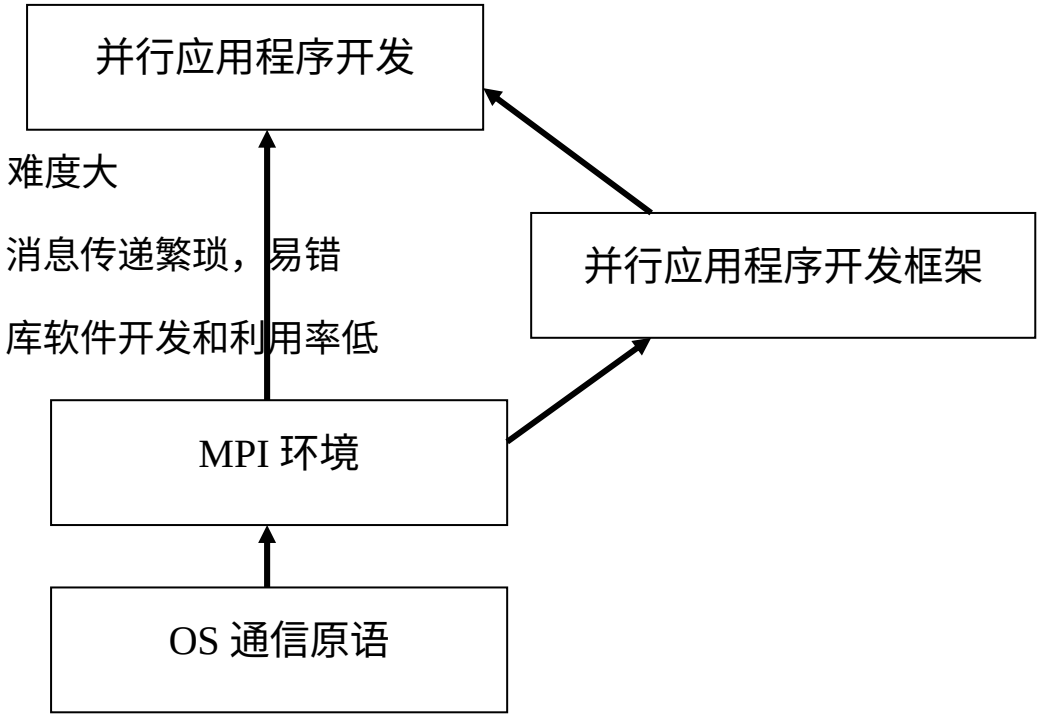
2．相容共享存储并行程序设计标准 OpenMP ：

- 在共享存储并行机上，进程与线程相容；



- MPI 与 OpenMP 的混合编程模式：
  - **可能**比较适合“ SMP 机群系统 ”+ 物理上分区计算的应用问题：即 SMP 之间用 MPI 处理各物理区，SMP 内部用 OpenMP 并行物理区内部的计算，综合 MPI 的可扩展性与 OpenMP 的简单性，以较小的代价获取可接受的性能；
  - **可能**要求用户成为 MPI 和 OpenMP 两个方面的专家;

3．特定领域建立“ 并行应用程序开发支持框架（工具箱）”：



- 采用面向对象技术，围绕矩阵向量（数组）数据结构：
  - 屏蔽消息传递，用户只需调用一个函数即可完成；
  - 集成各类核心库软件，但又提供面向对象的简单、统一的用户界面；
  - 对特定领域的具有公用性的问题，研究成熟后，可形成高性能的特殊功能部件，集成到该框架中；
  - 高性能细节对用户屏蔽，由专家负责；
  - 用户只需集中于基于区域分解的并行计算方法设计，然后通过调用框架提供的功能部件来完成并行应用程序的开发，框架能保证该并行应用程序的高性能、可扩展；

- 强调软件的可重用性、可继承性、可维护性和可验证性，缩短应用程序开发周期；

- 美国能源部 NERSC ODE2000 工程从 90 年代中期开始，重点支持了各个领域的约 20 个该类科学计算工具箱的建立，目前比较成功的为 Argonal 国家重点实验室开发的 PETSc（Parallel Extensible Tooltiks for Scientific Computing）,由于在LLNL的三维无结构网格NS应用程序的成功应用（ASCI Red 3072 台处理机，95%的并行效率，25%的峰值浮点性能），获得 SC’99 Gondar Bell 最佳应用成就奖。（公开初步的源代码，但高性能的核心部件不提供）
- 与 MPI 系统兼容;