財團法人國家實驗研究院
國家高速網路與計算中心
NATIONAL CENTER FOR HIGH-PERFORMANCE COMPUTING

MPI程式撰寫與實作

鄭守成
周朝宜

2006/11/04

課程規劃

- 14:10-15:10 基本MPI 語法介紹

Basic	P2P	Group	Other
MPI_Init, MPI_Finalize, MPI_Comm_Size, MPI_Comm_Rank	MPI_Send, MPI_Recv, MPI_Sendrecv	MPI_Scatter、MPI_Gather、MPI_Reduce、MPI_Allreduce MPI_Bcast	MPI_Barrier、MPI_Wtime

- 15:30-16:30 實機練習
  - hello.c
  - pi.c
  - par_1.c

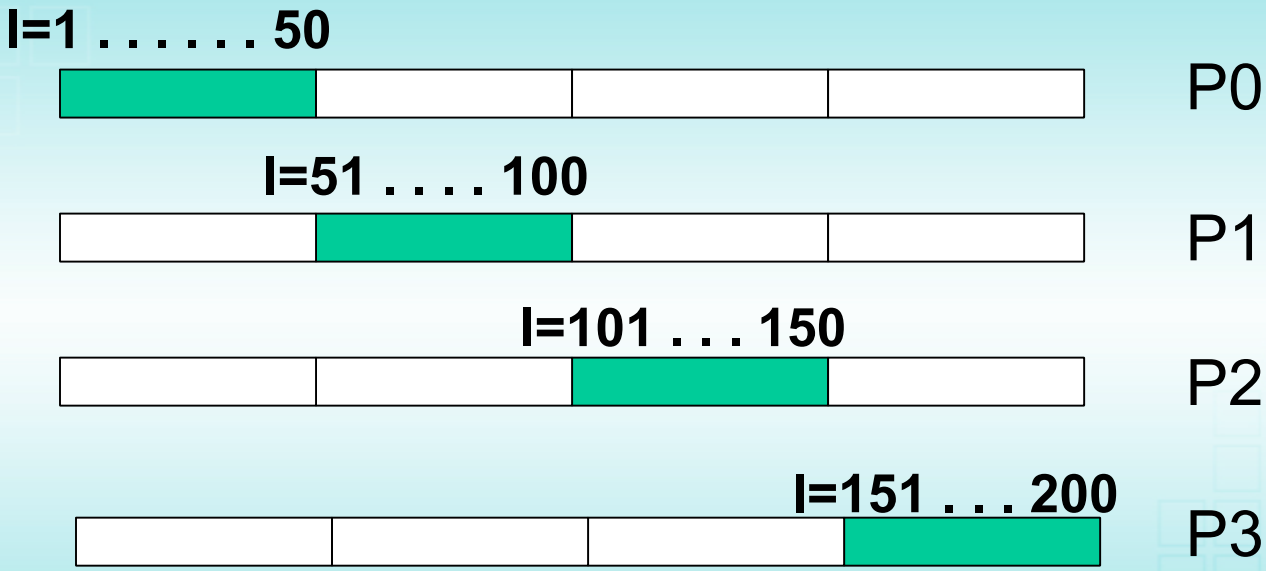
Outline

- Introduction
- MPI Basic
- Point to Point Communication
- Collective Communication
- Exercise
  - hello.c
  - pi.c
  - par_1.c

Introduction

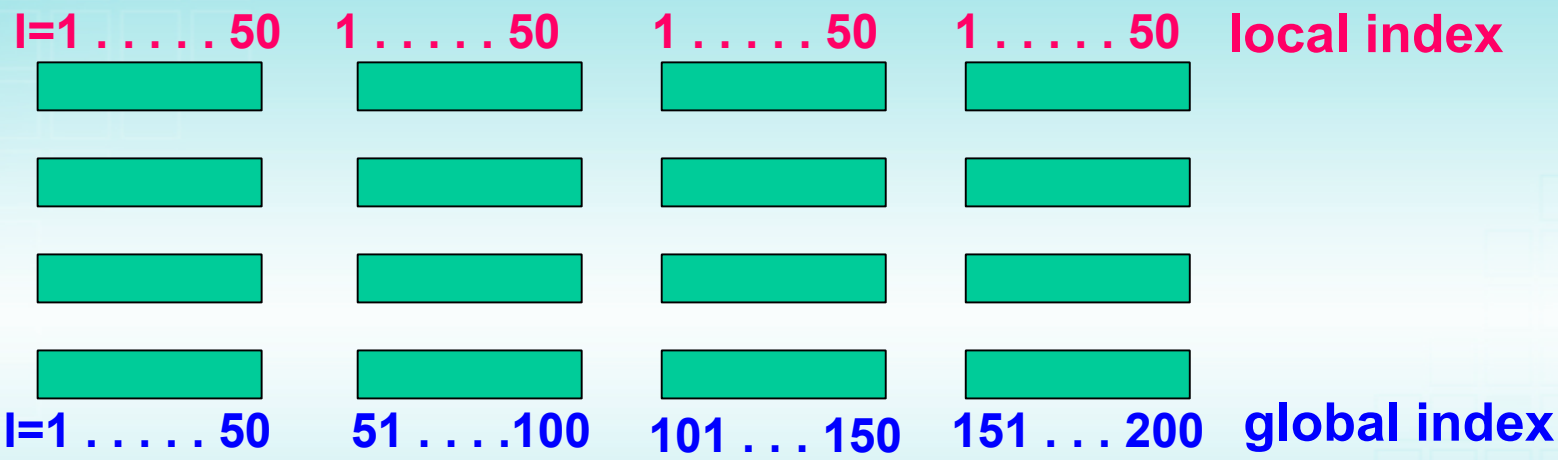
- 平行計算：將計算工作切割為 n 等份，在 n 個CPU上分工合作完成整個計算工作，每個 CPU負責其中一個等份的計算工作。
- 切割方法:
  ＊功能切割: 特殊情況下採用
  ＊資料切割: 一般情況下採用
- SPMD (Single Program Multi-Data)

Parallel Processing of 1-D Arrays without data partition



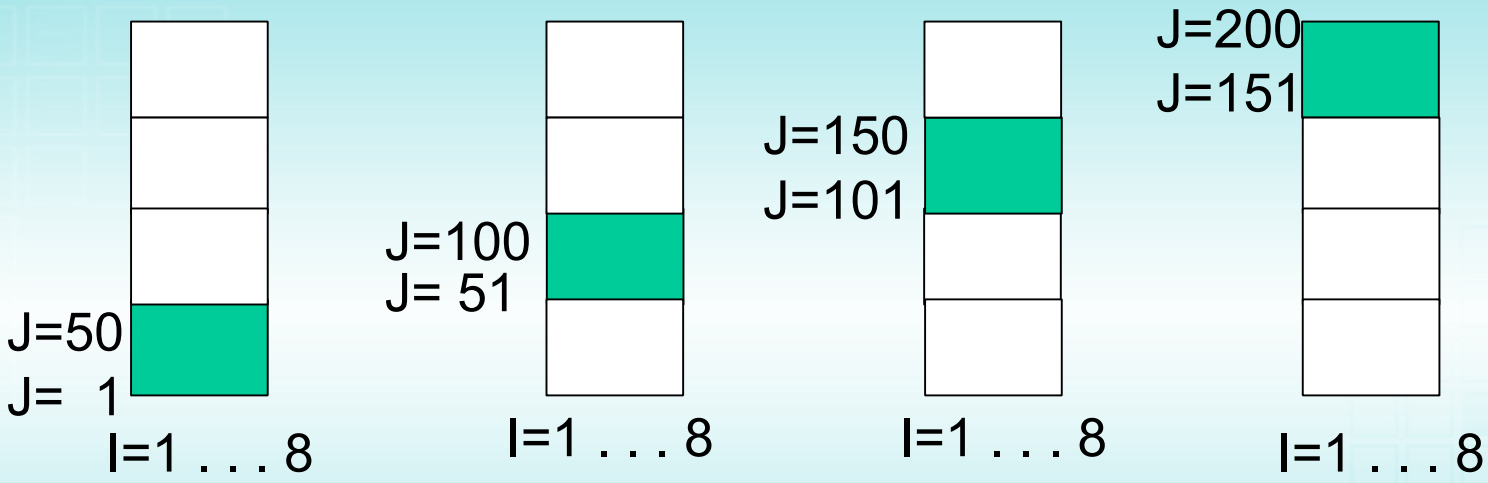

```
DO I=1,200                          DO I=ISTART, IEND
   A(I)=B(I)+C(I)*D(I)   ------>       A(I)=B(I)+C(I)*D(I)
ENDDO                               ENDDO
```

Parallel Processing of 1-D Arrays with data partition



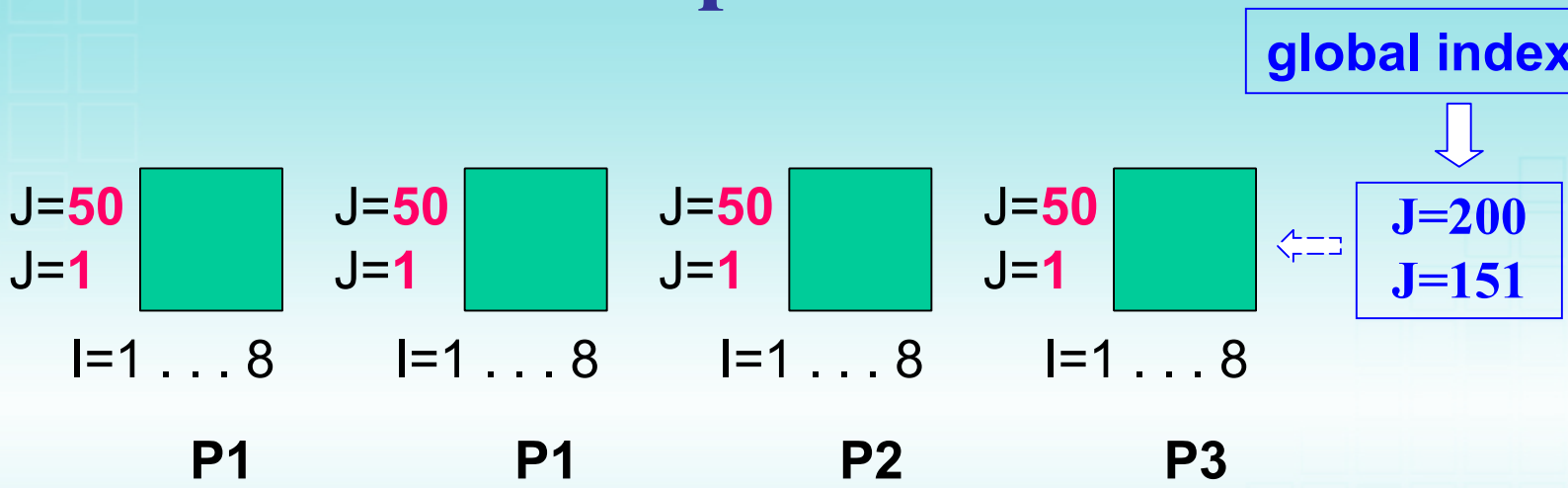

```
DO I=1,200                          DO I=ISTART, IEND
  A(I)=B(I)+C(I)*D(I)      →          A(I)=B(I)+C(I)*D(I)
ENDDO                               ENDDO
```

Parallel on 2nd index of 2-D Arrays without data partition



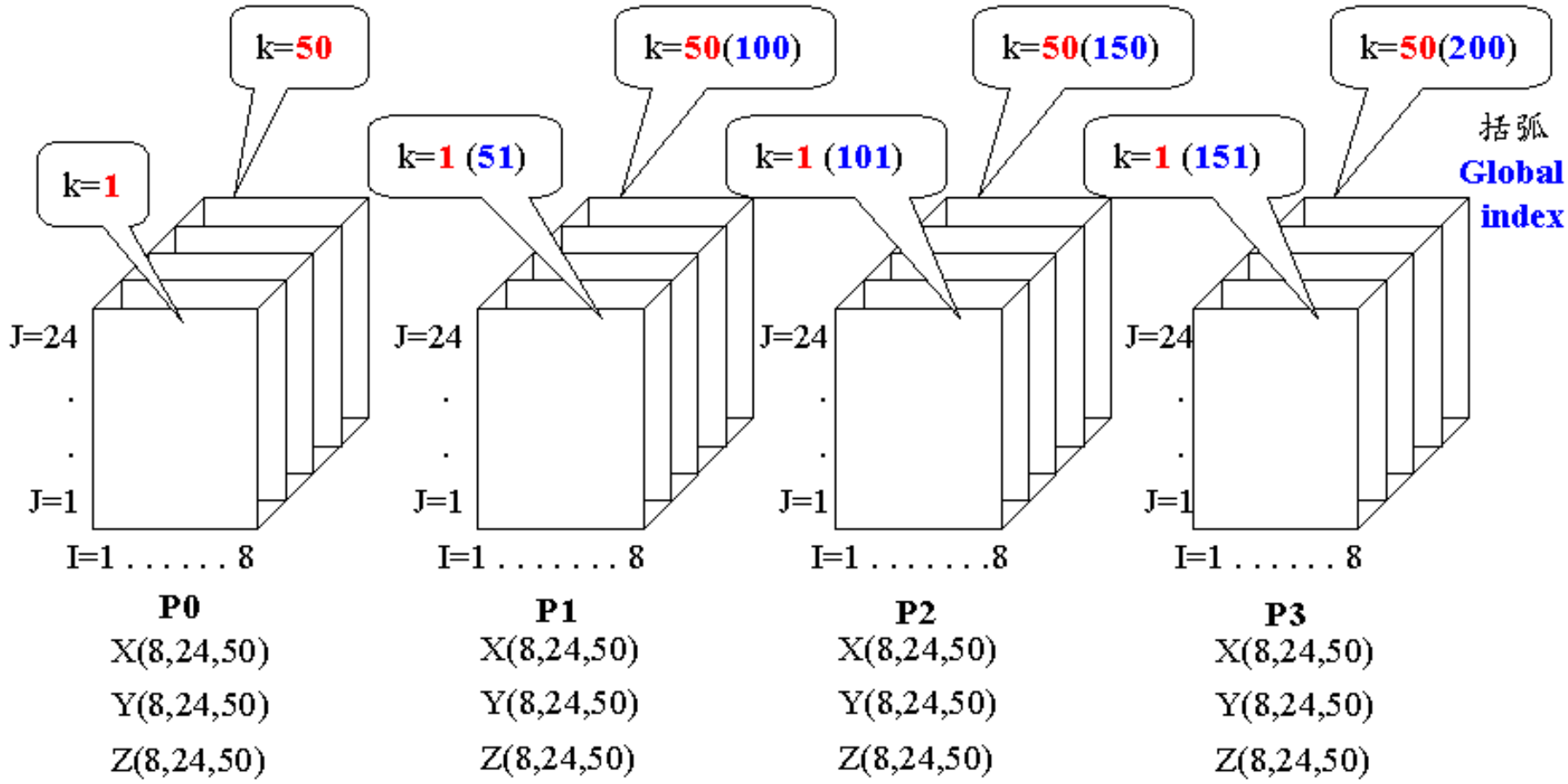

```
DO J=1,200                          DO J=JSTART, JEND
DO I=1,8                            DO I=1,8
  A(I,J)=B(I,J)+C(I,J)*D(I,J)  →    A(I,J)=B(I,J)+C(I,J)*D(I,J)
ENDDO                               ENDDO
ENDDO                               ENDDO
```

Parallel on 2nd index of 2-D Arrays with data partition

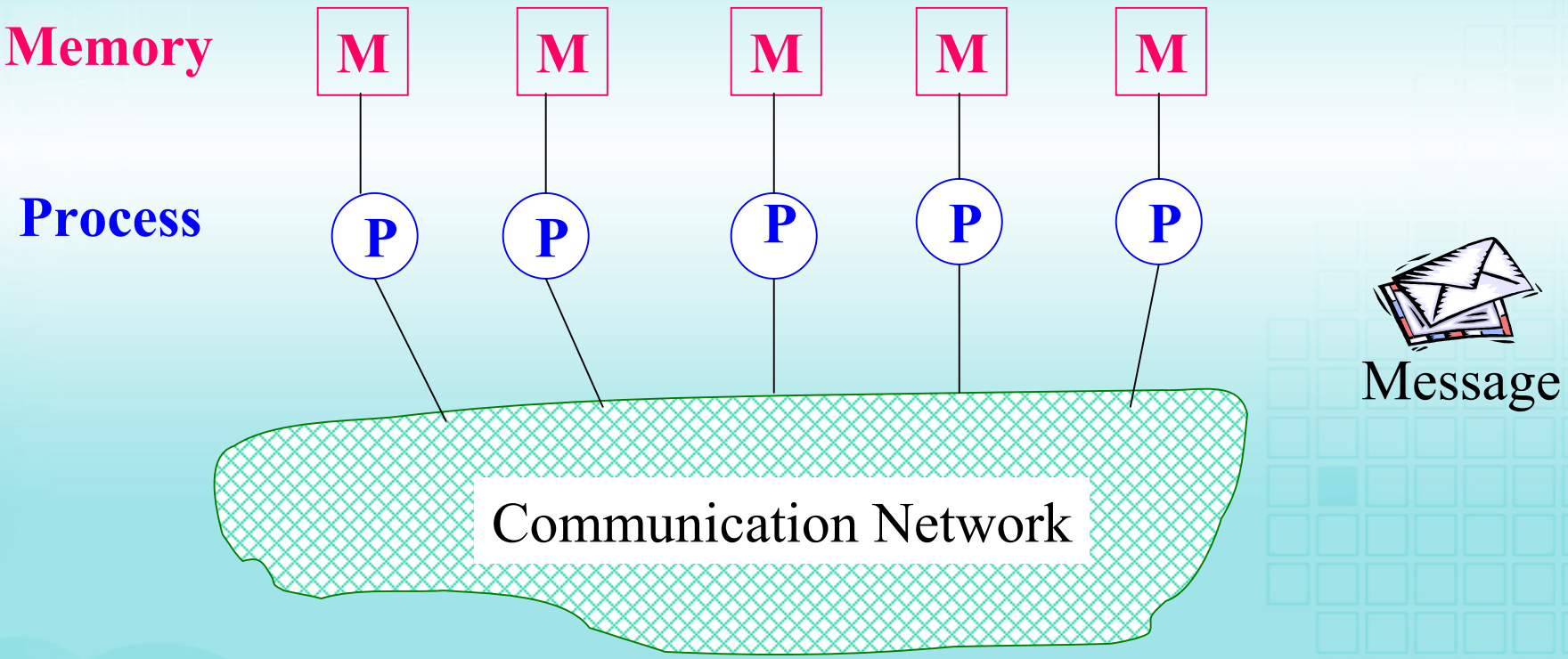



```
DO J=1,200                          DO J=JSTART, JEND
DO I=1,8                            DO I=1,8
  A(I,J)=B(I,J)+C(I,J)*D(I,J)  →    A(I,J)=B(I,J)+C(I,J)*D(I,J)
ENDDO                               ENDDO
ENDDO                               ENDDO
```

Partition on 3rd index of 3-D Arrays



Distributed Memory Architecture



Outline

- Introduction
- **MPI Basic**
- Point to Point Communication
- Collective Communication
- Exercise
  - hello.c
  - pi.c
  - par_1.c

MPI Basic

- **M**essage **P**assing **I**nterface
  - Forum define standard

    ~ 60 members

    1992/04 begin

http://www.mpi-forum.org/

- Available standards:

1994/05/05	MPI 1.0
1995/06/12	MPI 1.1
1997/07/18	**MPI 1.2**
1997/07/18	MPI 2.0

Goal

Portability : independent of Architectures and Hardware

- Free Software
  - MPICH **http://www-unix.mcs.anl.gov/mpi/mpich**
  - LAM/MPI **http://www.lam-mpi.org**

MPI Basic : C Interface

- Required header file

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

- MPI Function Format

```
error = MPI_Xxxx(parameter, ….)  ;
MPI_Xxxx(parameter, ….)  ;
```

- Initializing MPI
  - Must be the **first** routine called **and only once**

```
int MPI_Init(int *argc, char ***argv)
```

MPI Basic (3/4)

- Communicator Size

  MPI_Comm_size(MPI_Comm comm, int *size)

  - **How many processes are contained within a communicator?**
- **Process Rank**
  - **Process ID number within the communicator**
  - Starts with zero and goes to (n-1) where n is the number of processes requested
  - **Used to identify the source and destination of messages**

  MPI_Comm_rank(MPI_Comm comm, int *rank)

MPI Basic (4/4)

- Exiting MPI
  - Must be called last by “all” processes

  MPI_Finalize()

- 程式基本架構

```
#include <mpi.h>
void main(int argc, char *argv[]) {
int rank, size;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
/* … your code here … */
MPI_Finalize ();
}
```

hello.c

```
/*The Parallel Hello World Program*/
#include <stdio.h>
#include <mpi.h>
main(int argc, char **argv)
{
    int rank, size;
    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    printf("Hello World from Node %d of %d \n", rank, size);
    MPI_Finalize();
}
```

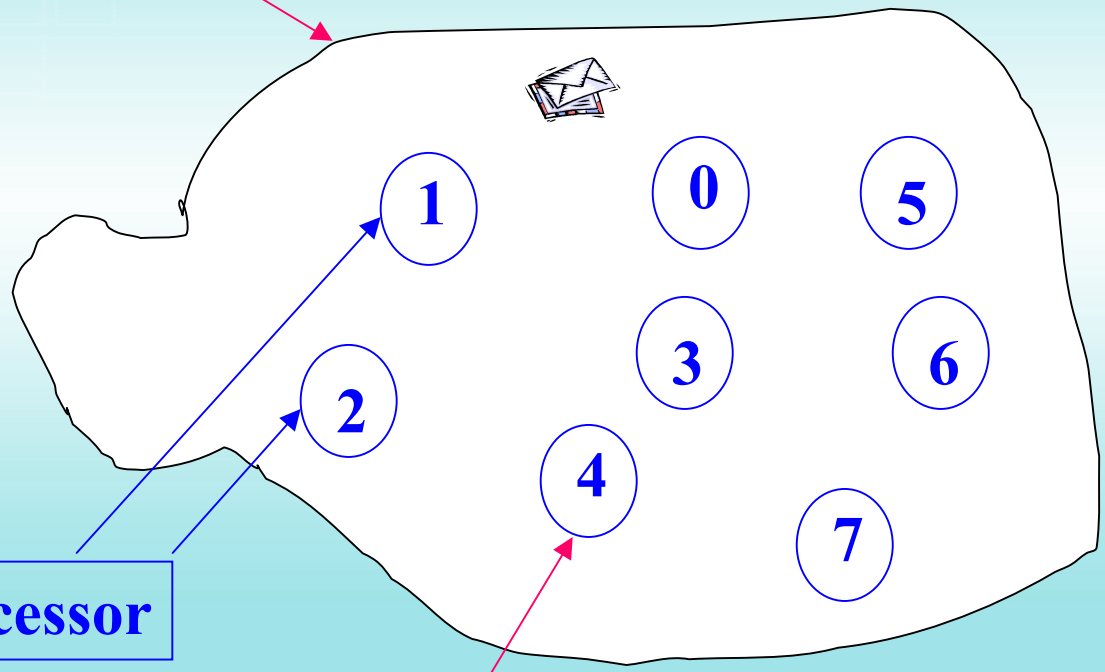
```
Hello World from Node 0 of 2
Hello World from Node 1 of 2
```

Communicator and Rank

Communicator
MPI_COMM_WORLD

Point to Point
Collective

Communication



Source
Destination

Processor

Rank

```
if (myid == 0)
  { do some…}
else{….}
```

MPI Data Type

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	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Outline

- Introduction
- MPI Basic
- **Point to Point Communication**
- Collective Communication
- Exercise
  - hello.c
  - pi.c
  - par_1.c

MPI_Send

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

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)

Performs a basic send

```
int buf[10];
MPI_Send(buf, 10, MPI_INT, 1, 123, MPI_COMM_WORLD);
```

MPI_Recv

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

Basic receive

buf initial address of **receive** buffer (choice)

Output Parameter

count number of elements in **receive** buffer (nonnegative integer)

datatype datatype of each **receive** buffer element (handle)

source rank of source (integer)

tag message tag (integer)

comm communicator (handle)

status status object (Status)

```
MPI_Recv(buf, 10, MPI_INT, 0, 123, MPI_COMM_WORLD,
&status );
```

Outline

- Introduction
- **MPI Basic**
- Point to Point Communication
- **Collective Communication**
- Exercise
  - hello.c
  - pi.c
  - par_1.c

MPI_Bcast

```
int MPI_Bcast
(
    void  *buf,
    int     count,
    MPI_Datatype    datatype,
    int     root,
    MPI_Comm   comm
) ;
```

buf initial address of send buffer (choice)

count number of elements in send buffer (nonnegative integer)

datatype datatype of each send buffer element (handle)

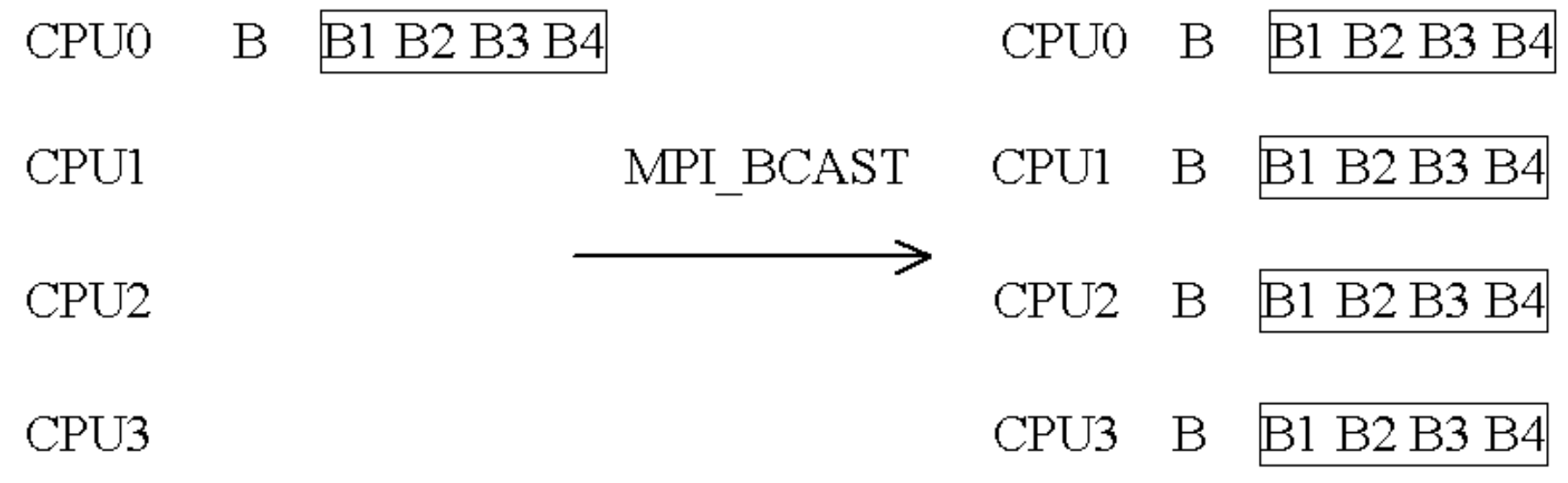
root rank of broadcast root (integer)

comm communicator (handle)

Broadcasts a message from the process with rank "root" to all other processes of the group

```
MPI_Bcast( &size, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

MPI_Bcast 示意圖



MPI_Barrier

```
int MPI_Barrier
(
   MPI_Comm   comm
) ;
```

comm communicator (handle)

```
MPI_Barrier(comm);
```

Blocks until all process have reached this routine

MPI_Wtime

double MPI_Wtime()

Returns an elapsed time in seconds on the calling processor

```
double start_time, elap_time;
start_time = MPI_Wtime()
……
 elap_time = MPI_Wtime() - start_time
```

MPI_Reduce

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

sendbuf address of send buffer (choice)

recvbuf address of receive buffer (choice, significant only at root)

count number of elements in send buffer (integer)

datatype datatype of each send buffer element (handle)

op reduce operation (handle)

root rank of root process (integer)

comm communicator (handle)

Reduces values on all processes to a single value

```
MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
MPI_COMM_WORLD);
```

Predefined Reduce Operators in MPI

Operation Name	Meaning
MPI_SUM	sum 累加
MPI_PROD	product 乘積
MPI_MAX	maximum 最大值
MPI_MIN	minimum 最小值
MPI_MAXLOC	max value and location
MPI_MINLOC	min value and location
MPI_LAND	logical AND
MPI_LOR	logical OR
MPI_LXOR	logical exclusive OR
MPI_BAND	binary AND
MPI_BOR	binary OR
MPI_BXOR	binary exclusive OR

MPI_Allreduce

```
int MPI_Allreduce
(
    void            *sendbuf,
    void            *recvbuf,
    int             count,
    MPI_Datatype    datatype,
    MPI_Op          op,
    MPI_Comm        comm
)
```

sendbuf address of send buffer (choice)

recvbuf address of receive buffer (choice, significant only at root)

count number of elements in send buffer (integer)

datatype datatype of each send buffer element (handle)

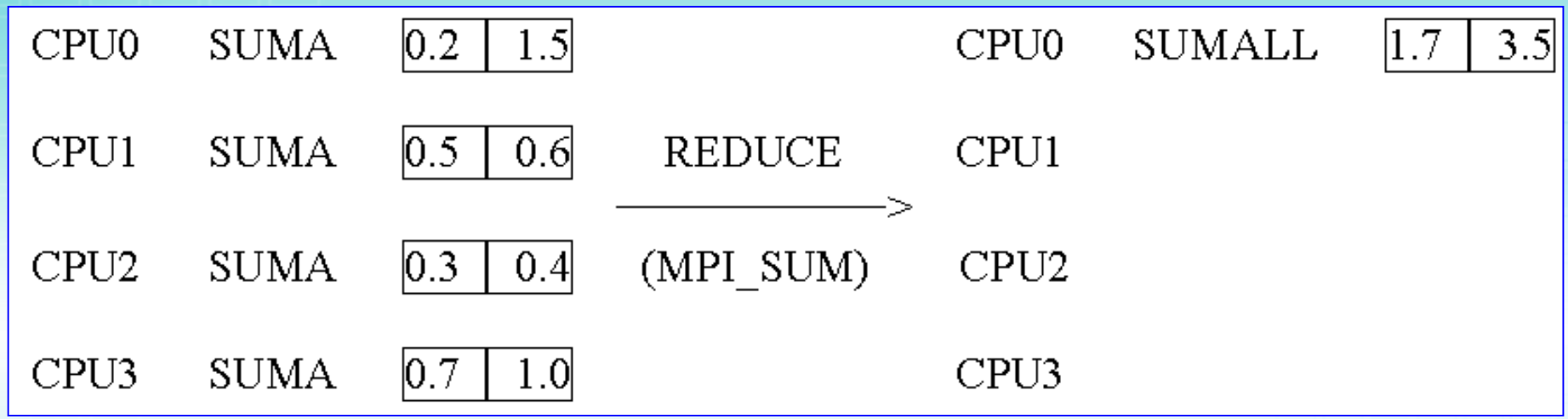
op reduce operation (handle)

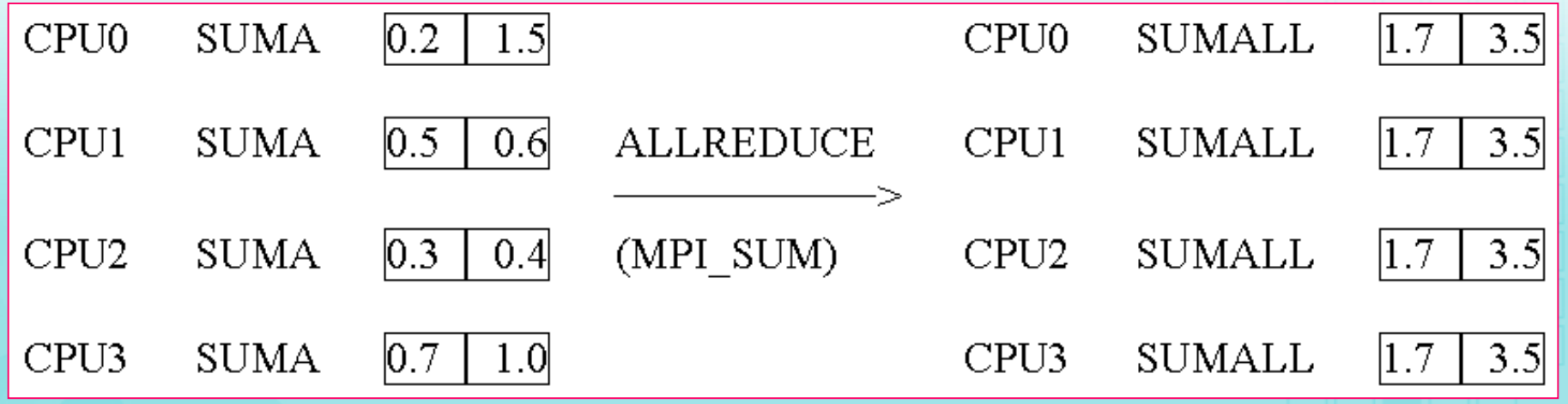
comm communicator (handle)

Combines values from all processes and distribute the result back to all processes

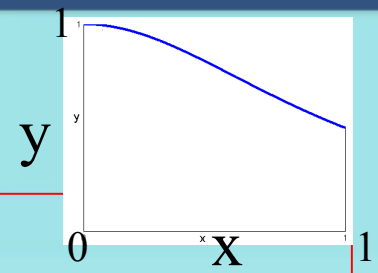
```
MPI_Allreduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM,
MPI_COMM_WORLD);
```

MPI_Reduce *vs.* MPI_Allreduce 示意圖





pi.c (1/3)



$$\int_0^1 \frac{1}{1+x^2}dx = tan^{-1}\, x\Big[_{x=0}^{x=1} = \frac{\pi}{4}$$

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
double f( double );
double f( double a ){
    return (4.0 / (1.0 + a*a)); }
int main( int argc, char *argv[]){
    int n, myid, numprocs, i;
    double PI25DT = 3.141592653589793238462643;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    startwtime = MPI_Wtime();
  MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
```

pi.c (2/3)

```
    h   = 1.0 / (double) n; sum = 0.0;
    for (i = myid + 1; i <= n; i += numprocs){
        x = h * ((double)i - 0.5);
        sum += f(x);  }
    mypi = h * sum;
    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE,
MPI_SUM, 0, MPI_COMM_WORLD);
    endwtime = MPI_Wtime();
    if (myid == 0)    {
        printf("pi is approximately %.16f,
Error is %.16f\n", pi, fabs(pi - PI25DT)); }
    printf("myid = %d, wall clock time = %f\n",
myid, endwtime-startwtime);
    MPI_Finalize();
    return 0; }
```

Output of pi.c (3/3)

```
    The number of intervals = 100000
pi is approximately 3.1415926535981167, Error
is 0.0000000000083236
myid = 0, wall clock time = 0.003906
myid = 1, wall clock time = 0.003906
myid = 2, wall clock time = 0.003906
myid = 3, wall clock time = 0.003906
```

11th

- Results pi = 3.1415926535981167
- PI25DT = 3.141592653589793238462643
- Double Precision only 15 digits

MPI_Scatter

```
int MPI_Scatter
(
    void            *sendbuf,
    int             sendcnt,
    MPI_Datatype    sendtype,
    void            *recvbuf,
    int             recvcnt,
    MPI_Datatype    recvtype,
    int             root,
    MPI_Comm        comm
)
```

Sends data from one task to all other tasks in a group

sendbuf address of send buffer (choice, significant only at root)

sendcnt number of elements sent to each process (integer, significant only at root)

sendtype data type of send buffer elements (significant only at root) (handle)

recvbuf address of receive buffer (choice)

recvcount number of elements in receive buffer (integer)

recvtype data type of receive buffer elements (handle)

root rank of sending process (integer)

comm communicator (handle)

MPI_Gather

```
int MPI_Gather
(
   void              *sendbuf,
   int                sendcnt,
   MPI_Datatype       sendtype,
   void              *recvbuf,
   int               recvcount,
   MPI_Datatype       recvtype,
   int                root,
   MPI_Comm           comm
 )
```

Gathers together values from a group of processes

sendbuf address of send buffer (choice)

sendcnt number of elements sent to each process (integer)

sendtype data type of send buffer elements (handle)

recvbuf address of receive buffer (choice)

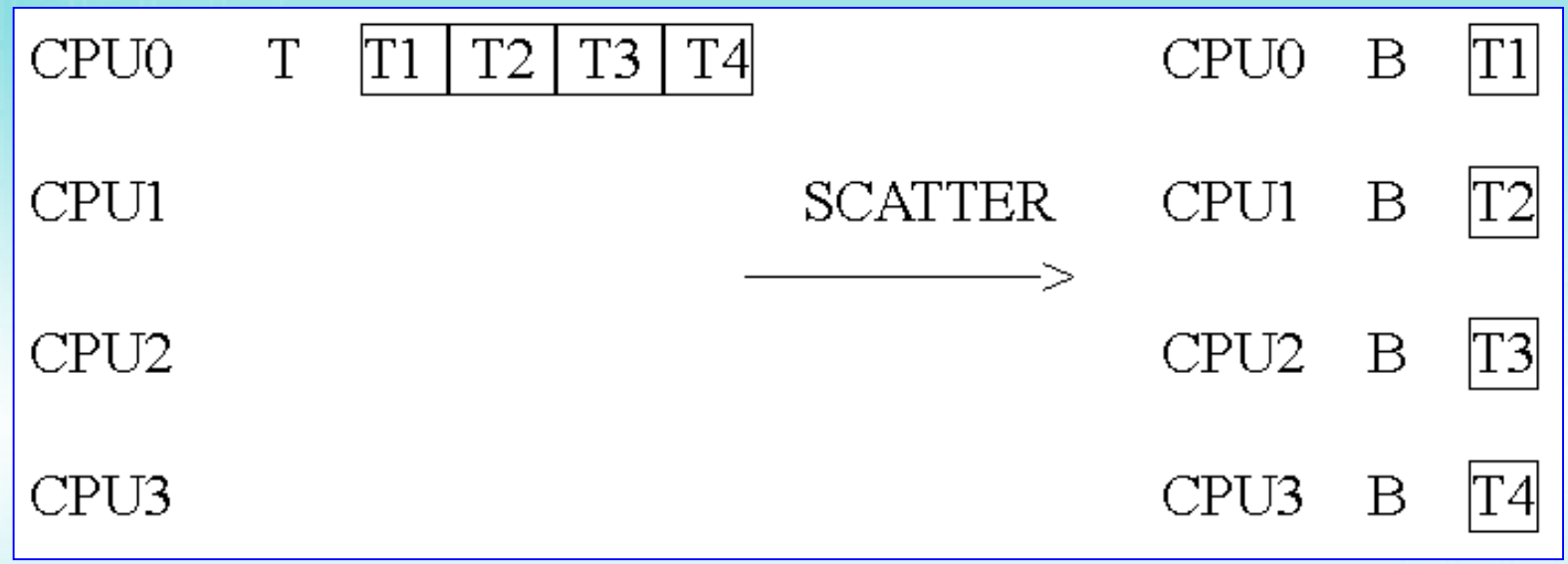
recvcount number of elements in receive buffer (integer, significant only at root)

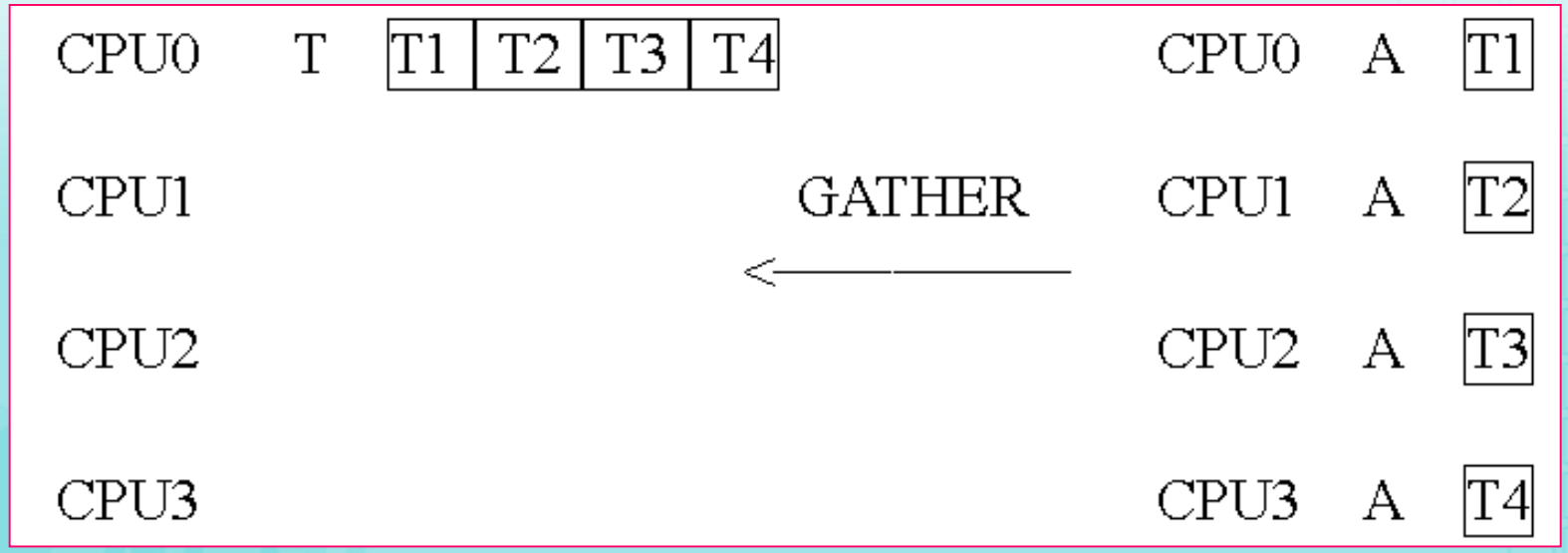
recvtype data type of receive buffer elements (handle) (significant only at root)

root rank of receiving process (integer)

comm communicator (handle)

MPI_Scatter vs. MPI_Gather 示意圖





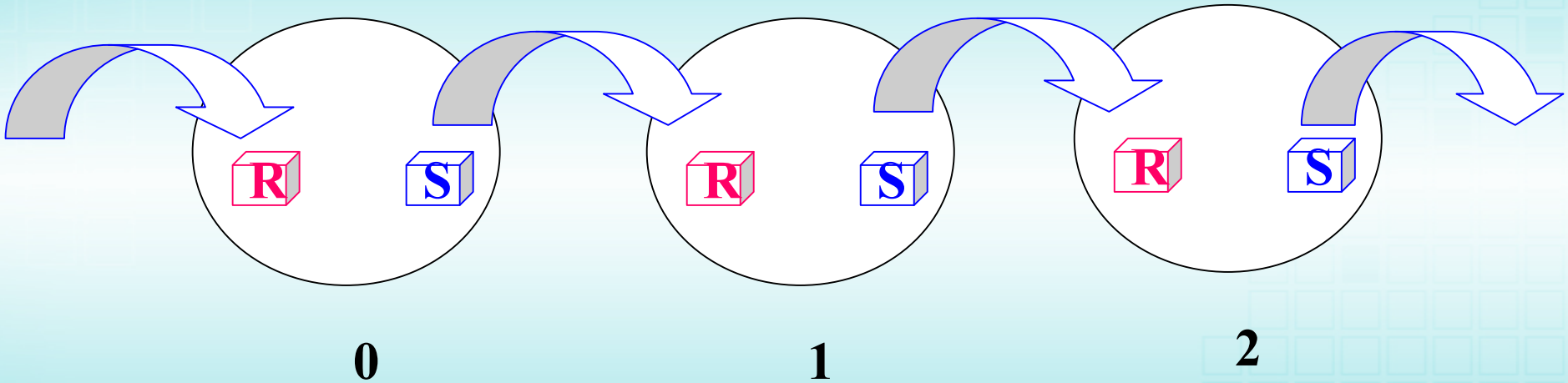
MPI_Sendrecv

```
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
) ;
```

Sends and receives a message

sendbuf initial address of send buffer (choice)
sendcount number of elements in send buffer (integer)
sendtype type of elements in send buffer (handle)
dest rank of destination (integer)
sendtag send tag (integer)
recvbuf initial address of receive buffer (choice)
recvcount number of elements in receive buffer (integer)
recvtype type of elements in receive buffer (handle)
source rank of source (integer)
tag receive tag (integer)
comm communicator (handle)
status status object (Status)

MPI_Sendrecv 示意圖



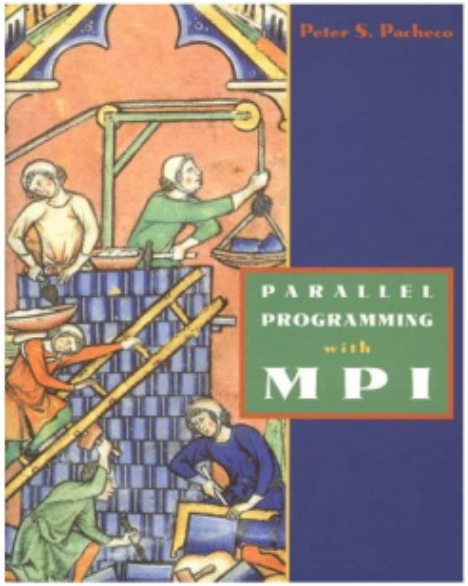
MPI_Sendrecv(**buf, 10, MPI_INT, R_NBR, 123,** ⇦ **SEND**
recvbuf, 10, MPI_INT, L_NBR, 123, ⇦ **RECV**
MPI_COMM_WORLD, &status);

Summary

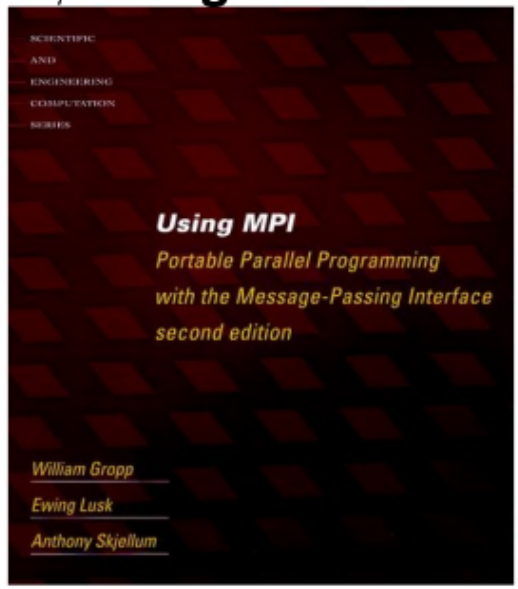
Basic	P2P	Group	Other
MPI_Init, MPI_Finalize, MPI_Comm_Size, MPI_Comm_Rank	MPI_Send, MPI_Recv, MPI_Sendrecv	MPI_Scatter、MPI_Gather、MPI_Reduce、MPI_Allreduce MPI_Bcast	MPI_Barrier、MPI_Wtime

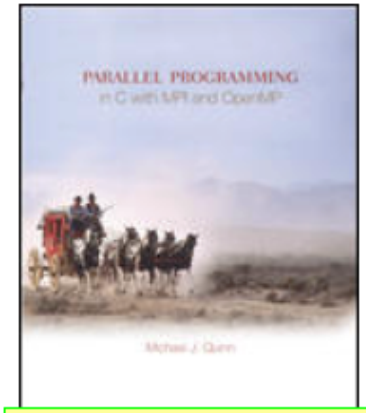
Reference

Peter S. Pacheco, ***Parallel Programming with MPI***

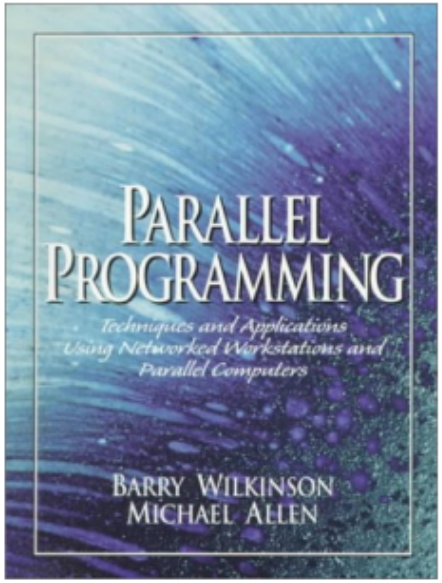


- W. Gropp, E. Lusk, and A. Shjellum, ***Using MPI***

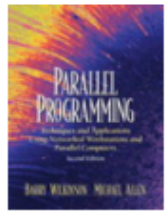




Michael J. Quinn, "*Parallel Programming in C with MPI and OpenMP*," McGraw-Hill, 2004



Barry Wilkinson and Michael Allen, "*Parallel Programming*," Prentice-Hall, 1999



Introduction to Message Passing and Parallel Programming With MPI

Dr. Georg Hager, Dr. Gerhard Wellein (RRZE)

A. Skjellum, P. Bangalore, S. Herbert,

R. Rabenseifner

MISSISSIPPI STATE UNIVERSITY
HIGH PERFORMANCE COMPUTING LAB
NSF ENGINEERING RESEARCH CENTER

H L R S

Parallel Programming with MPI

Science & Technology Support
High Performance Computing

Ohio Supercomputer Center
1224 Kinnear Road
Columbus, OH 43212-1163

Writing Message Passing Parallel Programs with MPI

A Two Day Course on MPI Usage

Course Notes

Version 1.8.2

Neil MacDonald, Elspeth Minty, Joel Malard, Tim Harding, Simon Brown, Mario Antonioletti

Parallel Programming for Clusters Using MPI: An Introduction

University of Ontario Institute of Technology
Parallel Programming Workshop

2006/11/28 ~ 29

■ 教育訓練課程 http://www.nchc.org.tw more

- 11.28-29 MPI平行計算程式設計基礎訓練課程 ... New!

Outline

- Introduction
- MPI Basic
- Point to Point Communication
- Collective Communication
- **Exercise**
  - **hello.c**
  - **pi.c**
  - **par_1.c**

基本指令介紹 (1/2)

- How to compiler ?
  - mpicc -O3 -o hello hello.c
  - which mpicc
- How to run ?
  - mpirun -np 4 hello
- How to submit job ?
  - qsub job.4

基本指令介紹 (2/2)

- How to monitor job status ?
  - qstat
  - qdel job_id

Monitor
Ganglia (LIVE)
PBS on line (LIVE)

Ganglia :
http://formosa.nchc.org.tw/

- How to modify job script file ?
  - vi job.4
- How to generate a profile of sequential code ?
  - cc -p –o seq_1 seq_1.c
  - ./seq_1 108
  - gprof seq_1

Exercise 1. Hello.c

- Copy source directionary
  - ls /work/pi
  - cp -r /work/pi .
- Compile and submit this job to
  - 2 processes
  - 4 processes
  - 8 processes
- Answer the following questions:
  - *What MPI functions were used in this code ?*
  - *Is the order in which processes perform their send/receives significant?*

Exercise 2. pi.c

- Compile and submit this job
  - $n=10^5$ for 2, 4, and 8 processes to compare the elapsed time.
    - mpicc -o pi pi.c
    - mpirun -np 2 pi 100000
  - 4 processes for $n=10^2$, 10^5, and 10^8 to compare the error.
- Answer the following question:
  - *What MPI functions were used in this code ?*
  - *Is the order in which processes perform their send/receives significant?*
  - *Compare the speed on case 1.*
  - *Compare the error on case 2.*

Exercise 3. seq_1.c to par_1.c

- Run 1080 digits for seq_1.c and par_1.c on 4 processors
  - Compare the elapsed time and error
- How to re-write seq_1.c into par_1.c ?
- How to fix the imbalance problem in the par_1.c?
- How to improve the efficiency of the code ?

$$\pi = \sum_{k=1}^{\infty} \frac{k2^k}{\binom{2k}{k}} - 3 = \sum_{k=1}^{\infty} \frac{k\,2^k\,(\mathrm{k}!)^2}{(2\mathrm{k})!} - 3$$

S. Plouffe, "On the computation of the n'th decimal digit of various transcendental numbers," November 1996

The output of the formula

K	Value
1	1.0000000000000000000000000000000
2	1.3333333333333333333333333333333
3	1.2000000000000000000000000000000
4	0.9142857142857142857142857142857
5	0.6349206349206349206349206349206
…	…
101	0.0000000000000000000000000007105
102	0.0000000000000000000000000003605
103	0.0000000000000000000000000001829
104	0.0000000000000000000000000000928
105	0.0000000000000000000000000000471

	pi
Eq(1)	3.1415926535897932384626433832795
東京大學金田康正	3.1415926535897932384626433832795

Subroutine startend

```
int startend(int nproc,int is1,int is2,int gstart[16],int gend[16],int
gcount[16])
{
    int ilength, iblock, ir, i;
    ilength=is2-is1+1;   iblock=ilength/nproc;
    ir=ilength-iblock*nproc;
    for ( i=0; i < nproc; i++ ) {
      if(i < ir) {  gstart[i]=is1+i*(iblock+1); end[i]=gstart[i]+iblock; }
      else { gstart[i]=is1+i*iblock+ir;     gend[i]=gstart[i]+iblock-1;  }
      if(ilength < 1) {       gstart[i]=1;      gend[i]=0;    }
      gcount[i]=gend[i]-gstart[i] + 1;
    }
    return 0;
}
```

Thank You!