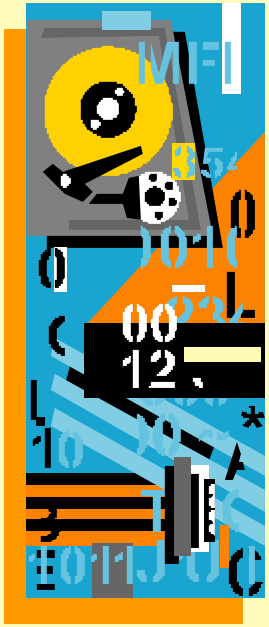
# Fortran 語言
# MPI 平行計算程式設計



鄭守成

# 課 程 大 綱

1. 前言
2. 無邊界資料交換的平行程式
3. 需要邊界資料交換的平行程式
4. 格點數不能整除的平行程式
5. 多維陣列的平行程式
6. MPI平行程式的效率提昇

# 課 程 大 綱

1. 前言
2. 無邊界資料交換的平行程式
3. 需要邊界資料交換的平行程式
4. 格點數不能整除的平行程式
5. 多維陣列的平行程式
6. MPI平行程式的效率提昇

# 前 言

- 平行計算概述
- MPI 平行計算軟體
- 國家高速電腦中心的平行計算環境
- 在IBM SP2上如何使用MPI
- 在PC Cluster上如何使用MPI

# 平行計算概述

- **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：**

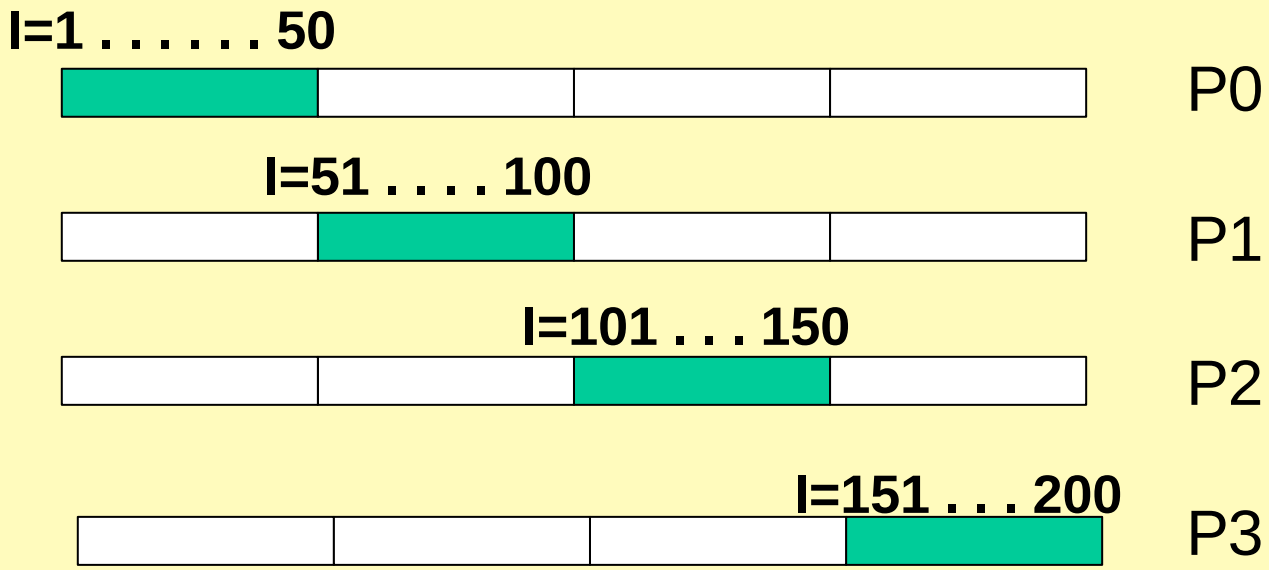
＊功能切割: 特殊情況下採用

＊資料切割: 一般情況下採用

structured grid: 等分切割

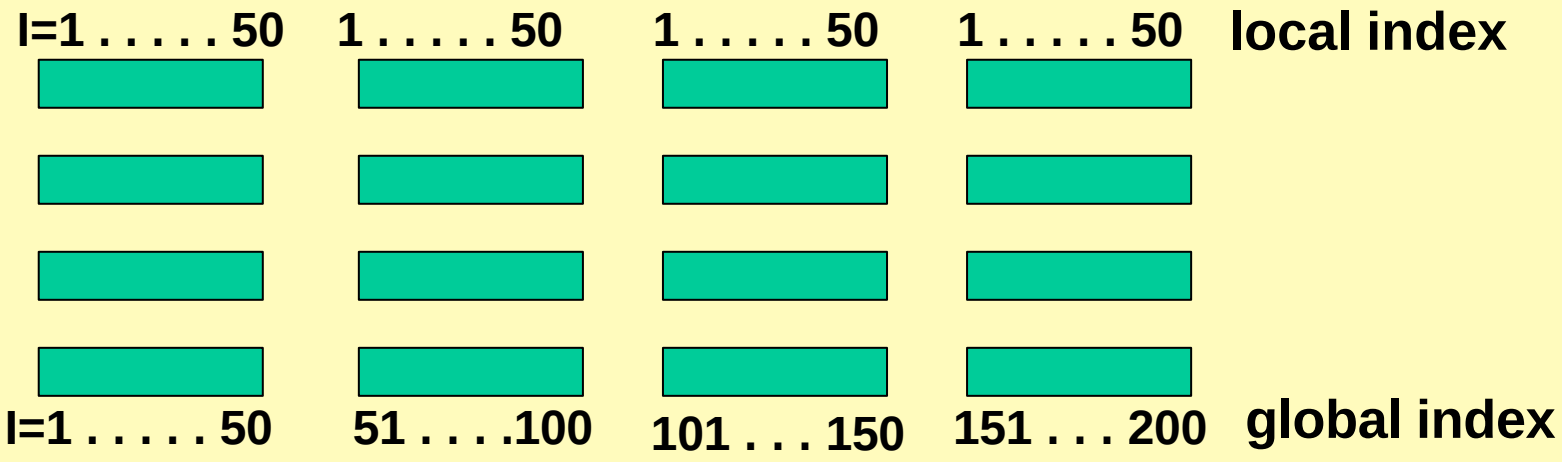
unstructured grid: 事先切割

# Parallel Processing of 1-D Arrays - without data partition



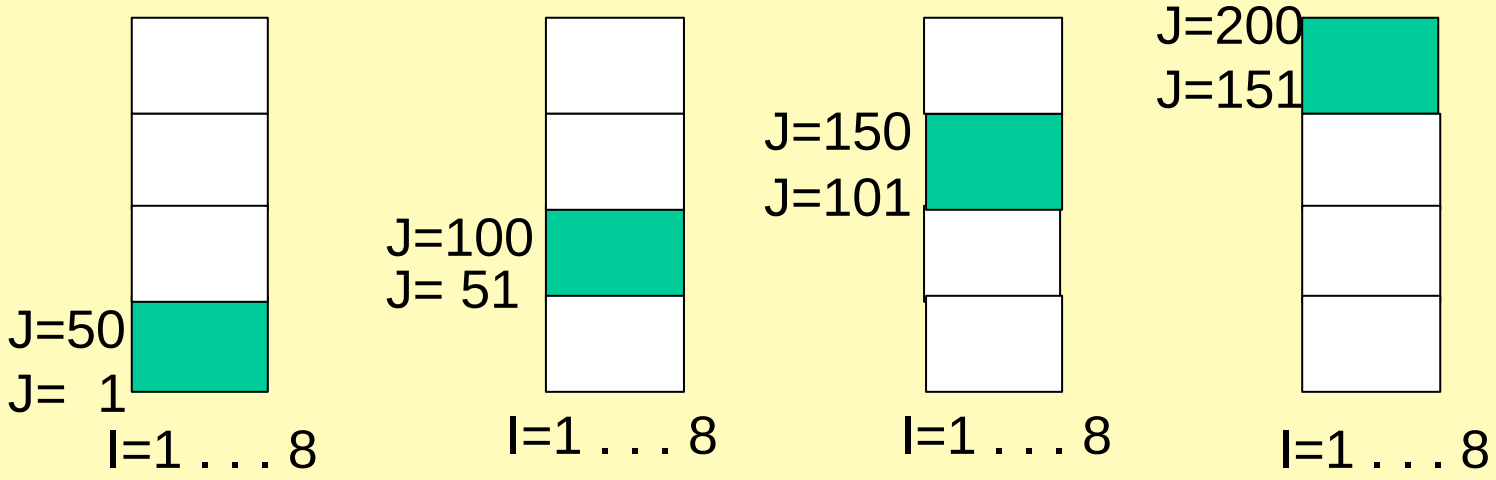

```
DO I=1,200                          DO I=ISTART, IEND
   A(I)=B(I)+C(I)*D(I)                 A(I)=B(I)+C(I)*D(I)
ENDDO                               ENDDO
```

# Parallel Processing of 1-D Arrays - with data partition



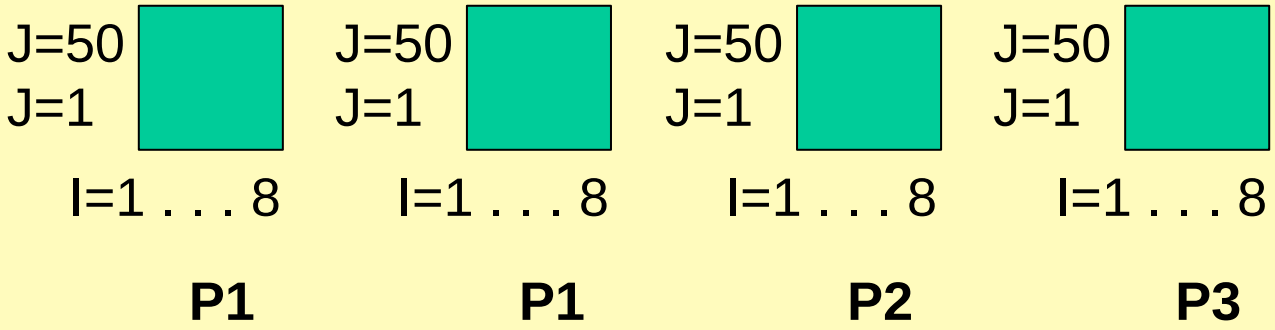

```
DO I=1,200                          DO I=ISTART, IEND
  A(I)=B(I)+C(I)*D(I)      →          A(I)=B(I)+C(I)*D(I)
ENDDO                                ENDDO
```

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



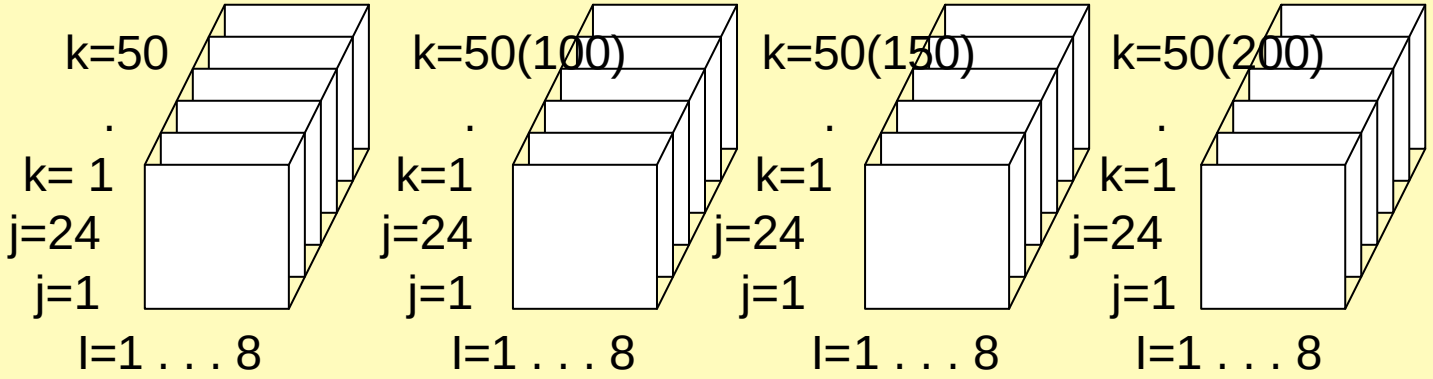

```
DO J=1,200                          DO J=JSTART, JEND
DO I=1,8                            DO I=1,8
  A(I,J)=B(I,J)+C(I,J)*D(I,J)  →      A(I,J)=B(I,J)+C(I,J)*D(I,J)
ENDDO                               ENDDO
ENDDO                               ENDDO
```

# 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




| P0 | P1 | P2 | P3 |
|---|---|---|---|
| x(8,24,50) | x(8,24,50) | x(8,24,50) | x(8,24,50) |
| y(8,24,50) | y(8,24,50) | y(8,24,50) | y(8,24,50) |
| z(8,24,50) | z(8,24,50) | z(8,24,50) | z(8,24,50) |

# Data Recursive

```
Parallelizable loops
DO J=1,N
DO I=1,M
    Y(I,J)=0.25*(X(I-1,J)+X(I+1,J)+X(I,J-1)+X(I,J+1))+H*F(I,J)
ENDDO
ENDDO

Data recursive – can not be parallelized
DO J=1,N
DO I=1,M
    X(I,J)=0.25*(X(I-1,J)+X(I+1,J)+X(I,J-1)+X(I,J+1))+H*F(I,J)
ENDDO
ENDDO
```

# 前 言

- 平行計算概述
- **MPI 平行計算軟體**
- 國家高速電腦中心的平行計算環境
- 在IBM SP2上如何使用MPI
- 在PC Cluster上如何使用MPI

# MPI 平行計算軟體

- MPI (Message Passing Interface) 是第一個標準化的 平行語言
- 可以使用在 Fortran、C、C++ 等語言
- 美國 Argonne National Lab免費提供網路下載 MPICH軟體
- http://www-unix.mcs.anl.gov/mpi/mpich
- ftp.mcs.anl.gov

# 前 言

- 平行計算概述
- MPI 平行計算軟體
- **國家高速電腦中心的平行計算環境**
- 在IBM SP2上如何使用MPI
- 在PC Cluster上如何使用MPI

# 國家高數電腦中心的平行計算環境

- 公司自備的 MPI 平行軟體 :
  - IBM p690, IBM SP2 SMP, SGI O3800, HP superdome
- MPICH 公用平行軟體 : PC cluster
- 平行環境 : IBM p690, IBM SP2 SMP, HP superdome, PC cluster
- 工作排程軟體 (job scheduler);
  - IBM : LoadLeveler
  - HP superdome : LSF
  - PC cluster : DQS like

# 前 言

- 平行計算概述
- MPI 平行計算軟體
- 國家高速電腦中心的平行計算環境
- **在IBM SP2上如何使用MPI**
- 在PC Cluster上如何使用MPI

# 在IBM SP2, p690上如何使用MPI

- mpxlf -O3 -qarch=auto -qstrict -o file.x file.f
- llclass ： CPU availability for each queue class
- llsubmit job_command_file ： submit a batch job
- llq ： job queue status
- llcancel jobid ：delete a submitted job

# IBM SP2 Job command file

```
#!/bin/csh
#@ executable = /usr/bin/poe
#@ network.mpi= css0,shared,us
#@ arguments = /working_dir/file.x
#@ output     = outp4
#@ error      = outp4
#@ job_type   = parallel
#@ class      = medium
#@ tasks_per_node = 4
#@ node = 1
#@ queue
```

# IBM p690 Job command file

- #!/bin/csh
- #@ executable = /usr/bin/poe
- #@ network.mpi= csss,shared,us
- #@ arguments = /working_dir/file.x
- #@ output = outp8
- #@ error = outp8
- #@ job_type = parallel
- #@ class = 8cpu
- #@ tasks_per_node = 8
- #@ node = 1
- #@ queue

# IBM SP2 - llclass

| Name | MaxJobCPU d+hh:mm:ss | MaxProcCPU d+hh:mm:ss | Free Slots | Max Slots |
|---|---|---|---|---|
| interactive | -1 | 0+06:00:00 | 8 | 8 |
| short | -1 | 0+12:00:00 | 4 | 6 |
| medium | -1 | 1+00:00:00 | 16 | 128 |
| bigmem | -1 | 2+00:00:00 | 12 | 16 |
| mono | -1 | 7+00:00:00 | 1 | 1 |

# IBM SP2 - llq : job queue status

| job_id | user_id | submitted | stat | prior | class | running on |
|---|---|---|---|---|---|---|
| ivory2.4371.0 | u11kcj00 | 10/2 19:25 | R | 50 | medium | ivory33 |
| ivory3.1285.0 | u32ltk00 | 10/2 23:47 | R | 50 | bigmem | ivory41 |
| ivory2.4431.0 | y19mkh00 | 10/3 07:05 | R | 50 | bigmem | ivory41 |
| ivory2.4348.0 | u00jim00 | 10/2 06:49 | R | 50 | medium | ivory13 |
| ivory2.4437.0 | u11kcj00 | 10/3 10:08 | R | 50 | short | ivory6 |
| ivory3.1269.0 | u50lun00 | 10/2 09:28 | I | 50 | medium | |
| ivory2.4368.0 | u07ish00 | 10/2 19:17 | I | 50 | medium | |

# 前 言

- 平行計算概述
- MPI 平行計算軟體
- 國家高速電腦中心的平行計算環境
- 在IBM SP2上如何使用MPI
- **在PC Cluster上如何使用MPI**

# 在PC Cluster上如何使用MPI

- mpif77 -O3 -o file.x file.f
- qsub32 job_command_file : submit a batch job
- qstat32 : job queue status
- qdel32 jobid : delete a submitted job

# PC Cluster : DQS job queue status

```
qstat32

user_id   jobname CPUs    jobid      job_status      submitted time
c00tch00  HUP4   hpcs01  62   0:1   r   RUNNING   02/26/99 10:51:23
c00tch00  HUP4   hpcs02  62   0:1   r   RUNNING   02/26/99 10:51:23
c00tch00  HUP4   hpcs03  62   0:1   r   RUNNING   02/26/99 10:51:23
c00tch00  HUP4   hpcs04  62   0:1   r   RUNNING   02/26/99 10:51:23

----Pending Jobs ----------------------------------------------------------

c00tch00 RAD5              70   0:2       QUEUED    02/26/99 19:24:32
```

# PC Cluster job command file

- #!/bin/csh
- #$ -l qty.eq.4,FastEthernet
- #$ -N HUP4
- #$ -A user_id
- #$ -cwd
- #$ -j y
- cat $HOSTS_FILE > MPI_HOST
- mpirun -np 4 -machinefile MPI_HOST hubksp >& outp4

# 課 程 大 綱

1. 前言
2. **無邊界資料交換的平行程式**
3. 需要邊界資料交換的平行程式
4. 格點數不能整除的平行程式
5. 多維陣列的平行程式
6. MPI平行程式的效率提昇

# 無邊界資料交換的平行程式

- 六個MPI基本指令
  MPI_INIT、 MPI_FINALIZE、
  MPI_COMM_SIZE 、 MPI_COMM_RANK、
  MPI_SEND、 MPI_RECV
- 無邊界資料交換的循序程式T2SEQ
- 資料不切割的平行程式T2CP
- MPI_SCATTER、 MPI_GATHER、
  MPI_REDUCE、 MPI_ALLREDUCE
- 資料切割的平行程式T2DCP

# mpif.h 及 MPI 基本指令 (1)

```
      INCLUDE  'mpif.h'
      REAL*8   . . .
      INTEGER  . . .
      CALL MPI_INIT(IERR)
      . . .
      CALL MPI_FINALIZE(IERR)
      STOP
      END
```

# MPI 基本指令(2)

- CALL MPI_COMM_SIZE
  (MPI_COMM_WORLD, NPROC, IERR)

  NPROC is the number of CPUs for this program

- CALL MPI_COMM_RANK
  (MPI_COMM_WORLD, MYID, IERR)

  MYID is my CPU id, count from zero

# MPI 基本指令(3)

```
PROGRAM  T2CP
PARAMETER ( . . )
INCLUDE  'mpif.h'
REAL*8   . . .
INTEGER  NPROC, MYID
CALL MPI_INIT (IERR)
CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NPROC, IERR)
CALL MPI_COMM_RANK (MPI_COMM_WORLD, MYID, IERR)
. . .
CALL MPI_FINALIZE (IERR)
STOP
END
```

# MPI 點對點通訊指令

```
CALL MPI_SEND (DATA, ICOUNT,  DATA_TYPE,
      IDEST, ITAG, MPI_COMM_WORLD, IERR)

CALL MPI_RECV (DATA, ICOUNT, DATA_TYPE,
      ISRC, ITAG, MPI_COMM_WORLD, ISTAT,
       IERR)

INTEGER   ISTAT(MPI_STATUS_SIZE)
```

# MPI 內定的資料類別

| MPI data types | Fortran data type |
| --- | --- |
| MPI_CHARACTER | CHARACTER |
| MPI_LOGICAL | LOGICAL |
| MPI_INTEGER | INTEGER |
| MPI_REAL | REAL |
| MPI_REAL8 | REAL*8 |
| MPI_COMPLEX | COMPLEX |
| MPI_COMPLEX16 | COMPLEX*16 |

# 無邊界資料交換的平行程式

- 六個MPI基本指令
  MPI_INIT、 MPI_FINALIZE、
  MPI_COMM_SIZE 、 MPI_COMM_RANK、
  MPI_SEND、 MPI_RECV
- 無邊界資料交換的循序程式T2SEQ
- 資料不切割的平行程式T2CP
- MPI_SCATTER、 MPI_GATHER、
  MPI_REDUCE、 MPI_ALLREDUCE
- 資料切割的平行程式T2DCP

# 循序程式 T2SEQ

```
PROGRAM T2SEQ
 PARAMETER (NTOTAL=200)
 REAL*8  A(NTOTAL), B(NTOTAL), C(NTOTAL), D(NTOTAL), SUMA
 OPEN(7,FILE='input.dat',STATUS='OLD',FORM='UNFORMATTED')
 READ (7) B
 READ (7) C
 READ (7) D
 SUMA=0.0
 DO I=1,NTOTAL
     A(I)=B(I)+C(I)*D(I)
     SUMA=SUMA+A(I)
 ENDDO
 WRITE(*,101)   (A(I),I=1,NTOTAL,5)
 WRITE(*,102)   SUMA
 STOP
 END
```

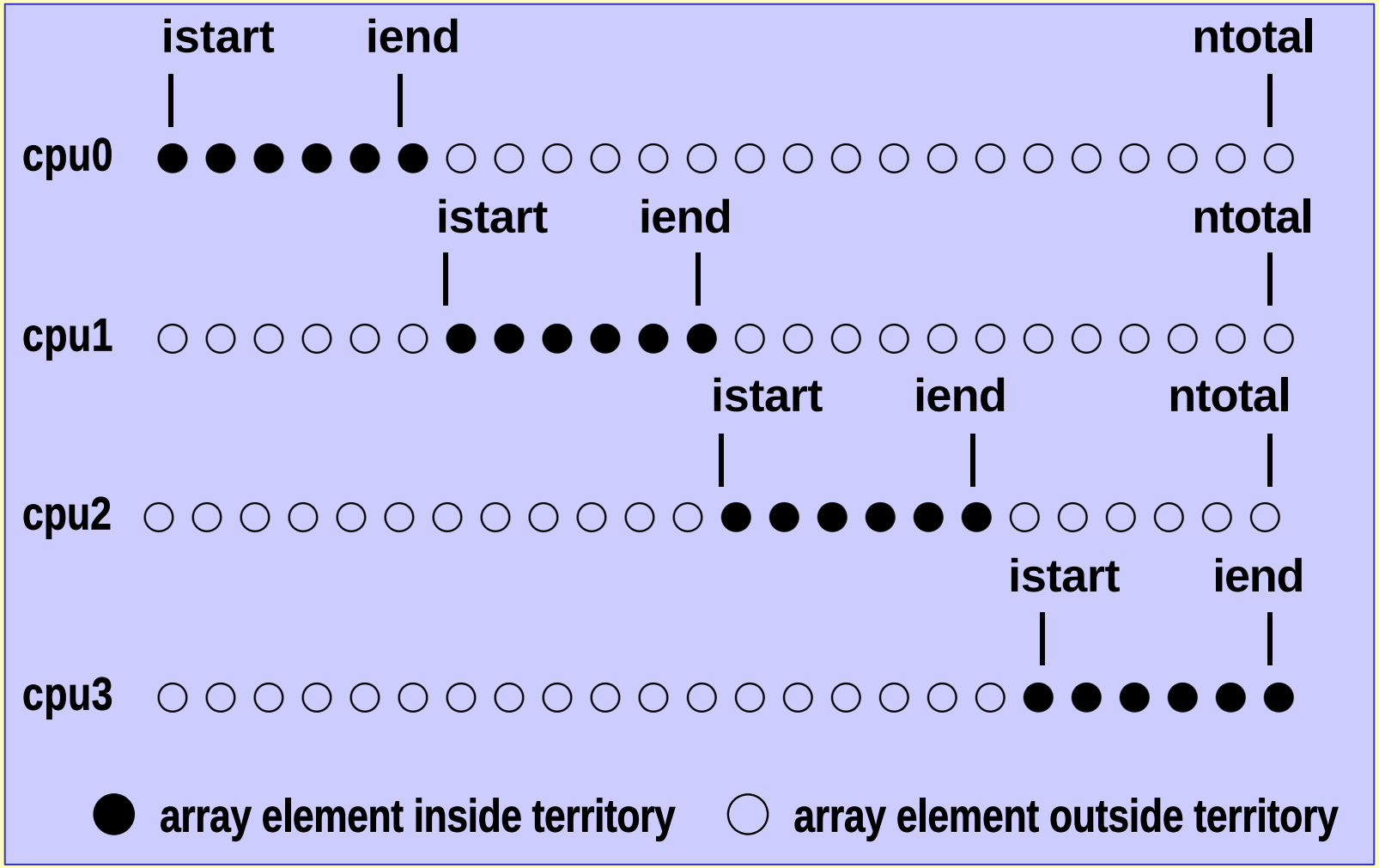
# 循序程式T2SEQ的測試結果

```
10.000   3.056   2.562   2.383   2.290   2.234   2.196   2.168   2.148   2.131
 2.118   2.108   2.099   2.091   2.085   2.079   2.074   2.070   2.066   2.063
 2.060   2.057   2.054   2.052   2.050   2.048   2.046   2.044   2.043   2.041
 2.040   2.039   2.037   2.036   2.035   2.034   2.033   2.032   2.031   2.031
SUM of array A =     .43855E+03
```

# 無邊界資料交換的平行程式

- 六個MPI基本指令
  MPI_INIT、 MPI_FINALIZE、
  MPI_COMM_SIZE 、 MPI_COMM_RANK、
  MPI_SEND、 MPI_RECV
- 無邊界資料交換的循序程式T2SEQ
- 資料不切割的平行程式T2CP
- MPI_SCATTER、 MPI_GATHER、
  MPI_REDUCE、 MPI_ALLREDUCE
- 資料切割的平行程式T2DCP

# 資料不切割的平行計算示意圖



# 資料不切割的平行程式 T2CP(1)

```
      PROGRAM T2CP
      INCLUDE   'mpif.h'
      PARAMETER (NTOTAL=200)
      REAL*8     A(NTOTAL), B(NTOTAL), C(NTOTAL), D(NTOTAL), SUMA
      INTEGER   NPROC, MYID, ISTAT(MPI_STATUS_SIZE), ISTART, IEND,
     1                COMM, GCOUNT(0:7), GSTART(0:7), GEND(0:7)

      CALL MPI_INIT (IERR)
      CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NPROC, IERR)
      CALL MPI_COMM_RANK (MPI_COMM_WORLD, MYID, IERR)
      CALL STARTEND ( NPROC, 1, NTOTAL, GSTART, GEND, GCOUNT)
      ISTART=GSTART(MYID)
      IEND=GEND(MYID)
      PRINT *,' NPROC,MYID,ISTART,IEND=',  NPROC,MYID,ISTART,IEND
      COMM=MPI_COMM_WORLD
```

# 資料不切割的平行程式 T2CP(2)

```
IF (MYID.EQ.0) THEN
      OPEN(7,FILE='input.dat',STATUS='OLD',FORM='UNFORMATTED')
      READ (7) B
      READ (7) C
      READ (7) D
      DO IDEST=1,NPROC-1
          IST1=GSTART(IDEST)
          KNT1=GCOUNT(IDEST)
          CALL MPI_SEND (B(IST1), KNT1, MPI_REAL8, IDEST, 10,COMM,IERR)
          CALL MPI_SEND (C(IST1), KNT1, MPI_REAL8, IDEST, 20,COMM,IERR)
          CALL MPI_SEND (D(IST1), KNT1, MPI_REAL8, IDEST, 30,COMM,IERR)
      ENDDO
  ELSE
       KNT=GCOUNT(MYID)
       CALL MPI_RECV (B(ISTART), KNT, MPI_REAL8, 0,10, COMM,ISTAT,IERR)
       CALL MPI_RECV (C(ISTART), KNT, MPI_REAL8, 0,20, COMM,ISTAT,IERR)
       CALL MPI_RECV (D(ISTART), KNT, MPI_REAL8, 0,30, COMM,ISTAT,IERR)
   ENDIF
```

# 資料不切割的平行程式 T2CP(3)

```
CC  DO I=1,NTOTAL
      DO I=ISTART,IEND
          A(I)=B(I)+C(I)*D(I)
      ENDDO
      ITAG=110
      IF (MYID.NE.0) THEN
         KNT=GCOUNT(MYID)
         CALL MPI_SEND (A(ISTART), KNT, MPI_REAL8, 0, ITAG, COMM, IERR)
      ELSE
         DO ISRC=1,NPROC-1
             IST1=GSTART(ISRC)
             KNT1=GCOUNT(ISRC)
             CALL MPI_RECV (A(IST1), KNT1, MPI_REAL8, ISRC, ITAG, COMM,
                                    ISTAT, IERR)
         ENDDO
      ENDIF
```

# 資料不切割的平行程式 T2CP(4)

```
          IF(MYID.EQ.0) THEN
        WRITE(*,101) (A(I),I=1,NTOTAL,5)
        SUMA=0.0
        DO I=1,NTOTAL
            SUMA=SUMA+A(I)
        ENDDO
        WRITE(*,102) SUMA
      ENDIF
101   FORMAT(10F8.3)
102   FORMAT('SUM of array A =', E15.5)
      CALL MPI_FINALIZE (IERR)
      STOP
      END
```

# 資料不切割的平行程式 T2CP(5)

```
SUBROUTINE   STARTEND(NPROC,IS1,IS2,ISTART,IEND,ICOUNT)
INTEGER    NPROC, IS1, IS2, ISTART(0:31), IEND(0:31), ICOUNT(0:31)
ILENGTH=IS2-IS1+1
IBLOCK=ILENGTH/NPROC
IR=ILENGTH-IBLOCK*NPROC
DO ID=0,NPROC-1
    IF(ID.LT.IR) THEN
        ISTART(ID)=IS1+ID*(IBLOCK+1)
        IEND(ID)=ISTART(ID)+IBLOCK
    ELSE
        ISTART(ID)=IS1+ID*IBLOCK+IR
        IEND(ID)=ISTART(ID)+IBLOCK-1
    ENDIF
    IF(ILENGTH.LT.1) THEN
        ISTART(ID)=1
        IEND(ID)=0
    ENDIF
    ICOUNT(ID)=IEND(ID)-ISTART(ID)+1
ENDDO
END
```

# 資料不切割程式T2CP的測試結果

```
ATTENTION: 0031-408  4 nodes allocated by LoadLeveler, continuing...
 NPROC,MYID,ISTART,IEND= 4  2  101  150
 NPROC,MYID,ISTART,IEND= 4  1   51  100
 NPROC,MYID,ISTART,IEND= 4  3  151  200
 NPROC,MYID,ISTART,IEND= 4  0    1  50
 10.000   3.056   2.562   2.383   2.290   2.234   2.196   2.168   2.148   2.131
  2.118   2.108   2.099   2.091   2.085   2.079   2.074   2.070   2.066   2.063
  2.060   2.057   2.054   2.052   2.050   2.048   2.046   2.044   2.043   2.041
  2.040   2.039   2.037   2.036   2.035   2.034   2.033   2.032   2.031   2.031
SUM of array A =     .43855E+03
```

# SPMD(Single Program Multiple Data)程式

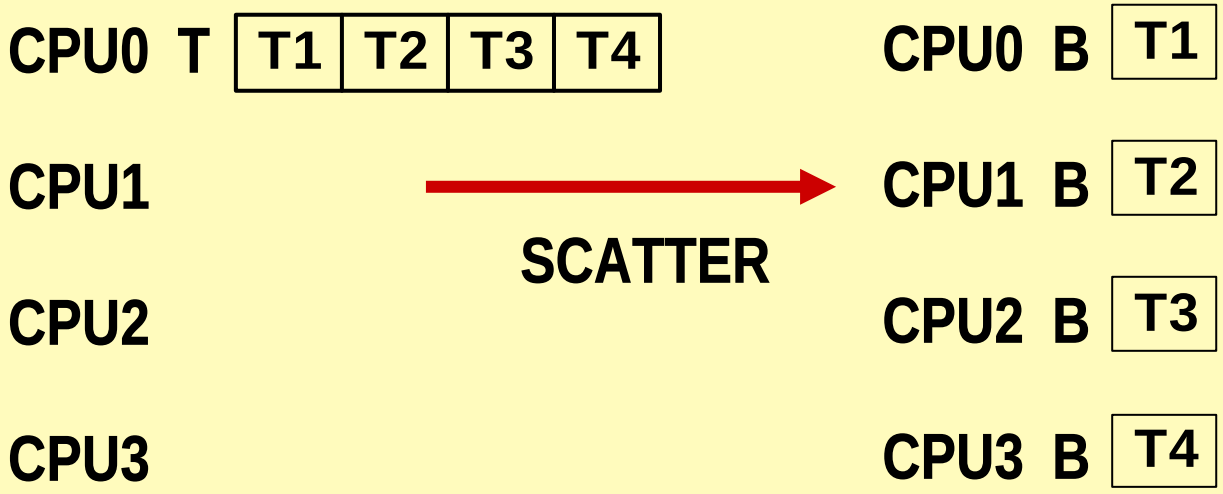
1. Use MYID to control the blocked IF statement
2. Use ISTART, IEND to compute the assigned range of arrays
3. Execute all statements otherwise

```
CALL MPI_INIT (IERR)
CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NPROC, IERR)
CALL MPI_COMM_RANK (MPI_COMM_WORLD, MYID, IERR)
CALL STARTEND (NPROC, 1, NTOTAL, GSTART, GEND, GCOUNT)
ISTART=GSTART(MYID)
IEND=GEND(MYID)
PRINT *,' NPROC,MYID,ISTART,IEND=',
NPROC,MYID,ISTART,IEND
```

# 無邊界資料交換的平行程式

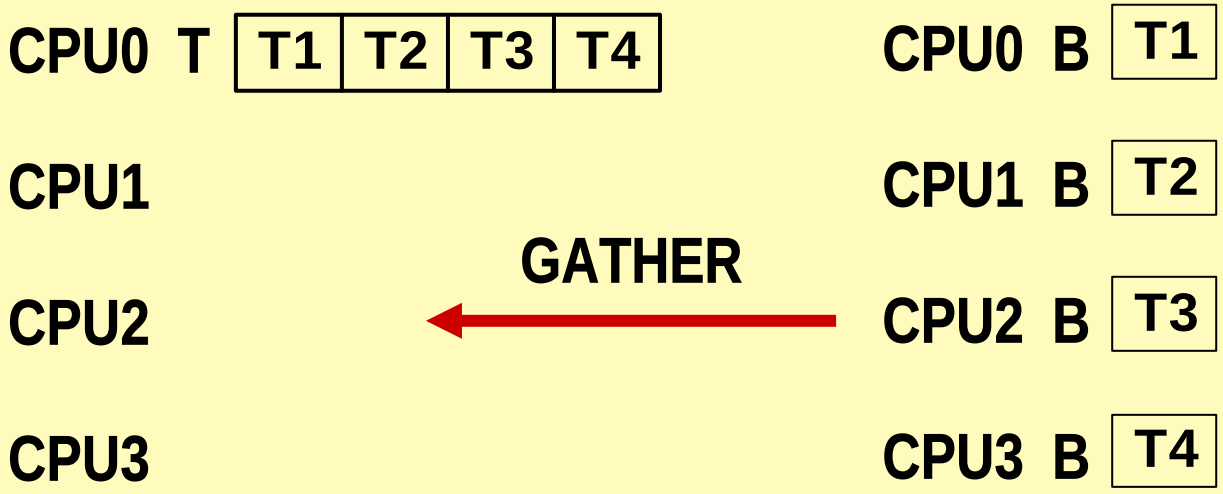
- 六個MPI基本指令
  MPI_INIT、 MPI_FINALIZE、
  MPI_COMM_SIZE 、 MPI_COMM_RANK、
  MPI_SEND、 MPI_RECV
- 無邊界資料交換的循序程式T2SEQ
- 資料不切割的平行程式T2CP
- MPI_SCATTER、 MPI_GATHER、
  MPI_REDUCE、 MPI_ALLREDUCE
- 資料切割的平行程式T2DCP

# 集體通訊指令 MPI_SCATTER



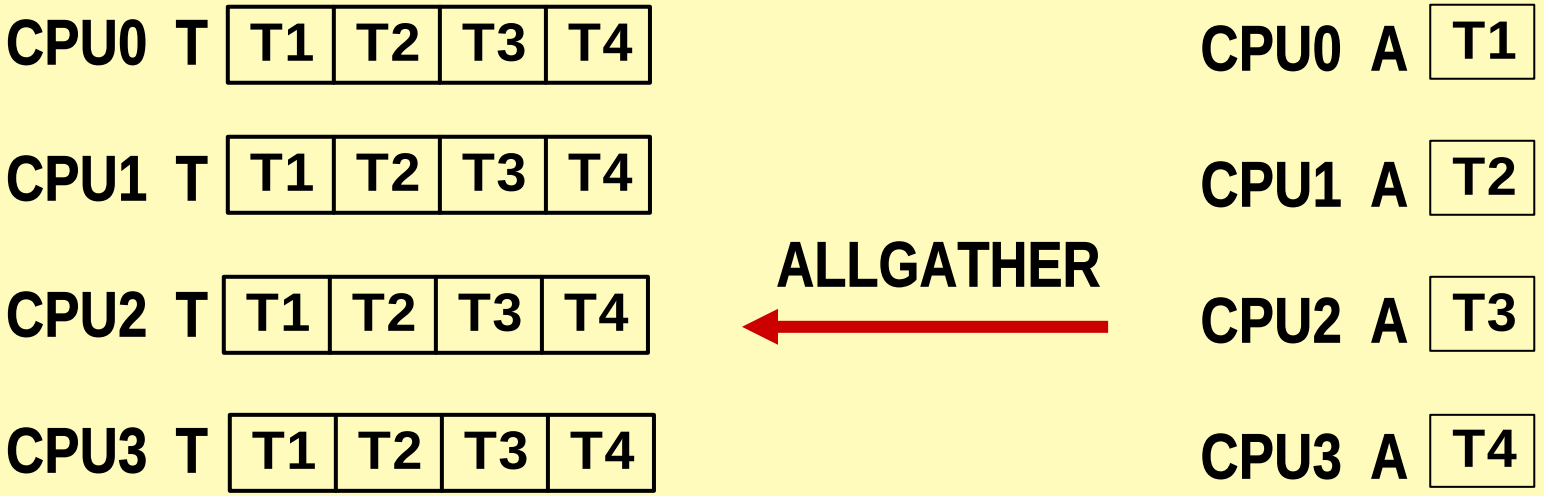

```
IROOT = 0
CALL MPI_SCATTER ( T, N, MPI_REAL8,  B, N, MPI_REAL8,
&                                    IROOT, MPI_COMM_WORLD, IERR)
```

# 集體通訊指令 MPI_GATHER



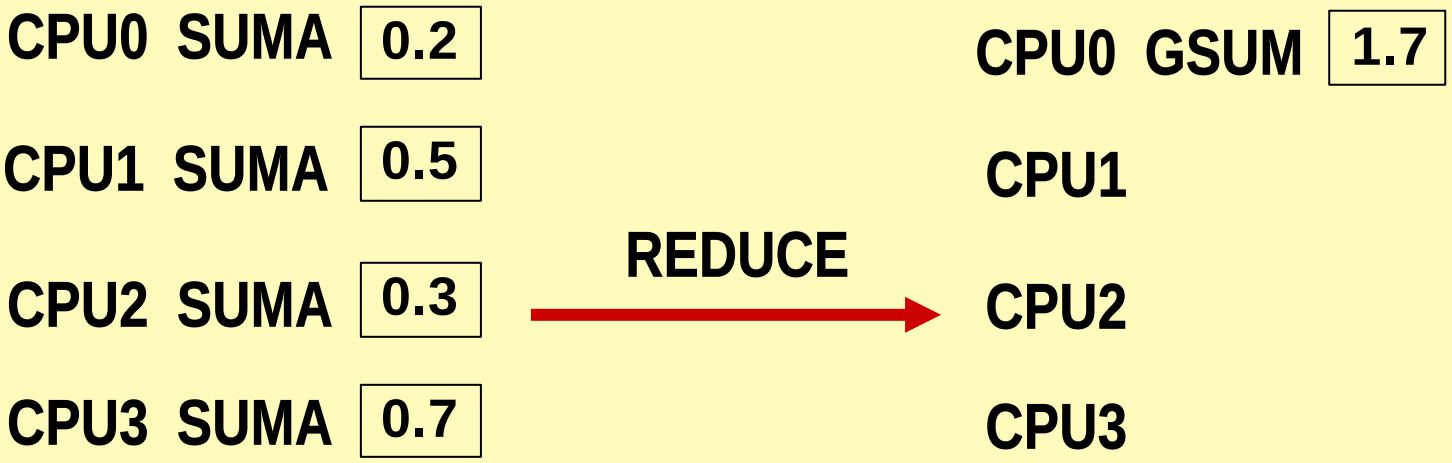

```
IDEST = 0
CALL MPI_GATHER ( A, N, MPI_REAL8,  T, N, MPI_REAL8,
&                                   IDEST, MPI_COMM_WORLD, IERR)
```

# 集體通訊指令 MPI_ALLGATHER




```
CALL MPI_ALLGATHER ( A, N, MPI_REAL8,  T, N, MPI_REAL8,
&                                  MPI_COMM_WORLD, IERR)
```

# 集體通訊指令 MPI_REDUCE




```
IROOT = 0
KOUNT = 1
CALL MPI_REDUCE ( SUMA, GSUM, KOUNT, MPI_REAL8,
&                 MPI_SUM, IROOT, MPI_COMM_WORLD, IERR)
```

# 集體通訊指令 MPI_ALLREDUCE


```
KOUNT=1
CALL MPI_ALLREDUCE ( SUMA, GSUM, KOUNT, MPI_REAL8,
&                                MPI_SUM, MPI_COMM_WORLD, IERR)
```

# MPI Reduction Function

| MPI 指令 | Operation | Data type |
|---|---|---|
| MPI_SUM<br>MPI_PROD | sum 累加<br>product乘積 | MPI_INTEGER, MPI_REAL, MPI_REAL8,<br>MPI_COMPLEX, MPI_COMPLEX16 |
| MPI_MAX<br>MPI_MIN | maximum最大值<br>minimum最小值 | MPI_INTEGER, MPI_REAL,<br>MPI_REAL8 |
| MPI_MAXLOC<br>MPI_MINLOC | max value & location<br>min value & location | MPI_2INTEGER, MPI_2REAL,<br>MPI_2REAL8 |
| MPI_LAND<br>MPI_LOR<br>MPI_LXOR | logical AND<br>logical OR<br>logical exclusive OR | MPI_LOGICAL |
| MPI_BAND<br>MPI_BOR<br>MPI_BXOR | binary AND<br>binary OR<br>binary exclusive OR | MPI_INTEGER, MPI_BYTE |

# 資料切割的平行程式 T2DCP(1)

```
      PROGRAM T2DCP
C
C     Data & Computational Partition Using MPI_SCATTER, MPI_GATHER
C     NP=4 must be modified when run on other than 4 processors
C
      PARAMETER (NTOTAL=200, NP=4, N=NTOTAL/NP)
      INCLUDE  'mpif.h'
      REAL*8   A(N), B(N), C(N), D(N), T(NTOTAL), SUMA, GSUM

      INTEGER  NPROC, MYID, ISTAT(MPI_STATUS_SIZE), ISTART,
     IEND,COMM

      CALL MPI_INIT (IERR)
      CALL MPI_COMM_SIZE ( MPI_COMM_WORLD, NPROC, IERR)
      CALL MPI_COMM_RANK ( MPI_COMM_WORLD, MYID, IERR)
      PRINT *,' NPROC,MYID=', NPROC, MYID
```

# 資料切割的平行程式 T2DCP(2)

```
      IF (MYID.EQ.0) THEN
         OPEN(7,FILE='input.dat',STATUS='OLD',FORM='UNFORMATTED')
         READ (7) T                                ! read array B
      ENDIF
      IROOT=0
      CALL MPI_SCATTER (T, N, MPI_REAL8,  B, N, MPI_REAL8,
     &                                IROOT, MPI_COMM_WORLD, IERR)
      IF(MYID.EQ.0) THEN
        READ (7) T                                 ! read array C
      ENDIF
      CALL MPI_SCATTER (T, N, MPI_REAL8,  C, N, MPI_REAL8,
     &                                IROOT, MPI_COMM_WORLD, IERR)
      IF(MYID.EQ.0) THEN
        READ (7) T                                 ! read array D
      ENDIF
      CALL MPI_SCATTER (T, N, MPI_REAL8,  D, N, MPI_REAL8,
     &                                IROOT, MPI_COMM_WORLD, IERR)
C
```

# 資料切割的平行程式 T2DCP(3)

```
      SUMA=0.0
C     DO I=1,NTOTAL
      DO I=1,N
           A(I)=B(I)+C(I)*D(I)
           SUMA=SUMA+A(I)
      ENDDO
      IDEST=0
      CALL MPI_GATHER (A, N, MPI_REAL8,  T, N, MPI_REAL8,
     &                                IDEST, MPI_COMM_WORLD, IERR)
      CALL MPI_REDUCE (SUMA, GSUM, 1, MPI_REAL8, MPI_SUM,
     &                                IDEST, MPI_COMM_WORLD, IERR)
      IF (MYID.EQ.0) THEN
          WRITE(*,101) (T(I),I=1,NTOTAL,5)
          WRITE(*,102) GSUM
      ENDIF
101   FORMAT(10F8.3)
102   FORMAT('SUM of array A =', E15.5)
      CALL MPI_FINALIZE (IERR)
      STOP
      END
```

# 平行程式 T2DCP 的測試結果

```
ATTENTION: 0031-408  4 nodes allocated by LoadLeveler, continuing...
  NPROC,MYID= 4  0
  NPROC,MYID= 4  1
  NPROC,MYID= 4  2
  NPROC,MYID= 4  3
10.000   3.056   2.562   2.383   2.290   2.234   2.196   2.168   2.148   2.131
  2.118   2.108   2.099   2.091   2.085   2.079   2.074   2.070   2.066   2.063
  2.060   2.057   2.054   2.052   2.050   2.048   2.046   2.044   2.043   2.041
  2.040   2.039   2.037   2.036   2.035   2.034   2.033   2.032   2.031   2.031
SUM of array A =     .43855E+03
```

# 課 程 大 綱

1. 前言
2. 無邊界資料交換的平行程式
3. **需要邊界資料交換的平行程式**
4. 格點數不能整除的平行程式
5. 多維陣列的平行程式
6. MPI平行程式的效率提昇

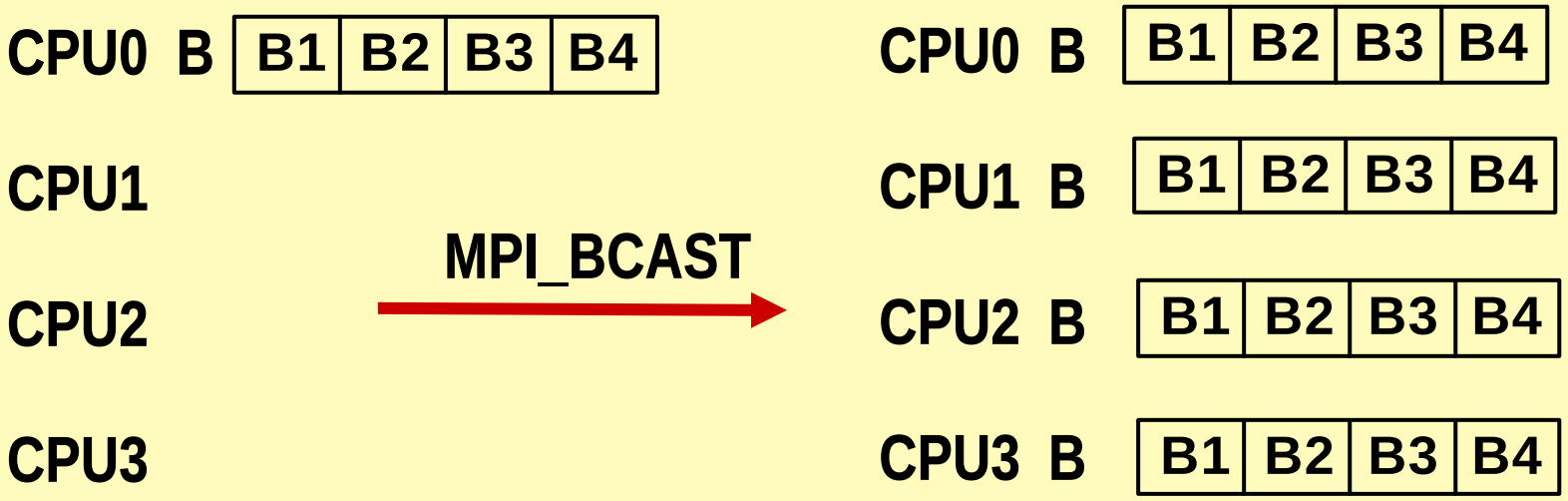
# 需要邊界資料交換的平行程式

- MPI_SENDRECV、 MPI_BCAST
- 需要邊界資料交換的循序程式T3SEQ
- 資料不切割的平行程式T3CP
- 交換一個陣列元素的資料切割平行程式 T3DCP_1
- 交換兩個陣列元素的資料切割平行程式 T3DCP_2

# 點對點指令 MPI_SENDRECV

```
ITAG=110
CALL MPI_SENDRECV
     (B(IEND),       ICOUNT,DATA_TYPE,IDEST,ITAG,
      B(ISTARTM1),ICOUNT,DATA_TYPE,ISRC,ITAG,
      MPI_COMM_WORLD, ISTATUS, IERR)
```

# 集體通訊指令 MPI_BCAST




```
IROOT = 0
CALL MPI_BCAST ( B, ICOUNT, DATA_TYPE, IROOT,
&                          MPI_COMM_WORLD, IERR)
```

# 需要邊界資料交換的平行程式

- MPI_SENDRECV、 MPI_BCAST
- 需要邊界資料交換的循序程式T3SEQ
- 資料不切割的平行程式T3CP
- 交換一個陣列元素的資料切割平行程式 T3DCP_1
- 交換兩個陣列元素的資料切割平行程式 T3DCP_2

# 邊界資料交換的循序程式T3SEQ(1)

```
   PROGRAM T3SEQ
C
   PARAMETER (NTOTAL=200)
   REAL*8   A(NTOTAL), B(NTOTAL), C(NTOTAL), D(NTOTAL), AMAX

    OPEN(7,FILE='input.dat',STATUS='OLD',FORM='UNFORMATTED')
    READ (7) B
    READ (7) C
    READ (7) D
    AMAX=-1.D12
    DO I=2,NTOTAL-1
       A(I)=C(I)*D(I)+(B(I-1)+2.0*B(I)+B(I+1))*0.25
        AMAX=MAX(AMAX,A(I))
     ENDDO
```

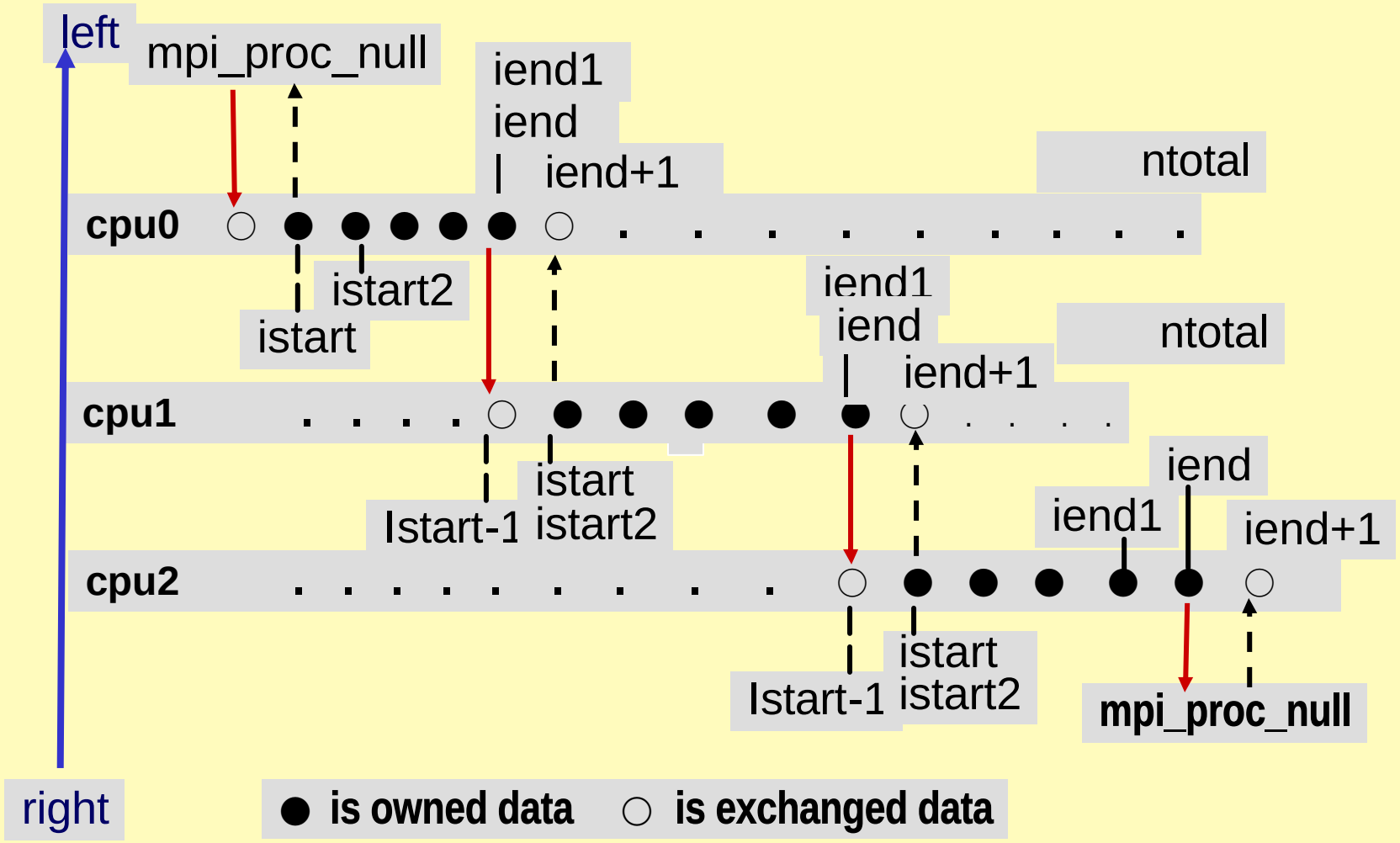
# 邊界資料交換的循序程式T3SEQ(2)

```
      WRITE(*,101) (A(I),I=1,NTOTAL,5)
       WRITE(*,102) AMAX
101  FORMAT(10F8.3)
102  FORMAT('MAXIMUM VALUE OF A ARRAY is', E15.5)
      STOP
      END
```

# 需要邊界資料交換的平行程式

- MPI_SENDRECV、 MPI_BCAST
- 需要邊界資料交換的循序程式T3SEQ
- 資料不切割的平行程式T3CP
- 交換一個陣列元素的資料切割平行程式 T3DCP_1
- 交換兩個陣列元素的資料切割平行程式 T3DCP_2

# 資料不切割的邊界資料交換示意圖



# 資料不切割的loop index(1)

```
AMAX=-1.D12
DO I=2,NTOTAL-1
    A(I)=C(I)*D(I)+(B(I-1)+2.0*B(I)+B(I+1))*0.25
    AMAX=MAX(AMAX,A(I))
ENDDO
```

- index I是從2到NTOTAL-1。在切割後，只有CPU0是從2開始，其他的CPU都是從 istart開始， 所以必須設一個變數istart2來解決這個問題 。

# 資料不切割的loop index(2)

```
ISTART2=ISTART
IF(MYID.EQ.0) ISTART2=2
```

- 而loop的終點只有最後一個CPU是到NTOTAL-1，也就是iend-1，其他的CPU 都是到iend，所以必須再設一個變數iend1來解決這個問題。

```
IEND1=IEND
IF(MYID.EQ.NPROC-1) IEND1=IEND-1
```

# 資料不切割的邊界資料交換

```
      ISTARTM1=ISTART-1
      IENDP1=IEND+1

      L_NBR=MYID-1
      R_NBR=MYID+1
      IF(MYID.EQ.0)            L_NBR=MPI_PROC_NULL
      IF(MYID.EQ.NPROC-1)  R_NBR=MPI_PROC_NULL

      ITAG=110
      CALL MPI_SENDRECV (B(IEND),        1, MPI_REAL8, R_NBR, ITAG,
     1                   B(ISTARTM1), 1, MPI_REAL8,  L_NBR, ITAG,
     2                   MPI_COMM_WORLD, ISTATUS, IERR)
      ITAG=120
      CALL MPI_SENDRECV (B(ISTART),1, MPI_REAL8,  L_NBR, ITAG,
     1                   B(IENDP1), 1, MPI_REAL8,  R_NBR, ITAG,
     2                   MPI_COMM_WORLD, ISTATUS, IERR)

      CALL MPI_ALLREDUCE ( AMAX, GMAX, 1, MPI_REAL8, MPI_MAX,
     1                     MPI_COMM_WORLD, IERR )
```

# 資料不切割的平行程式T3CP(1)

```
PROGRAM     T3CP
PARAMETER (NTOTAL=200)
INCLUDE  'mpif.h'
REAL*8   A(NTOTAL), B(NTOTAL), C(NTOTAL), D(NTOTAL), AMAX, GMAX

 INTEGER  NPROC, MYID, ISTAT(MPI_STATUS_SIZE), ISTART, IEND,
 1     L_NBR, R_NBR, COMM, GSTART(0:31), GEND(0:31), GCOUNT(0:31)
 CALL MPI_INIT (IERR)
 CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NPROC, IERR)
 CALL MPI_COMM_RANK (MPI_COMM_WORLD, MYID, IERR)
 CALL STARTEND (NPROC, 1, NTOTAL, GSTART, GEND,GCOUNT)
 ISTART=GSTART(MYID)
 IEND=GEND(MYID)
 PRINT *,' NPROC,MYID,ISTART,IEND=', NPROC,MYID,ISTART,IEND
```

# 資料不切割的平行程式T3CP(2)

```
COMM=MPI_COMM_WORLD
LASTP=NPROC-1
ISTARTM1=ISTART-1
IENDP1=IEND+1
ISTART2=ISTART
IF(MYID.EQ.0) ISTART2=2
IEND1=IEND
IF(MYID.EQ.LASTP) IEND-=IEND-1
L_NBR=MYID-1
R_NBR=MYID+1
IF(MYID.EQ.0)              L_NBR=MPI_PROC_NULL
IF(MYID.EQ.NPROC-1) R_NBR=MPI_PROC_NULL

IF(MYID.EQ.0) THEN
   OPEN(7,FILE='input.dat',STATUS='OLD',FORM='UNFORMATTED')
```

# 資料不切割的平行程式T3CP(3)

```
   READ (7) B
   READ (7) C
   READ (7) D
   DO IDEST=LASTP
      IST1=GSTART(IDEST)
      KNT1=GCOUNT(IDEST)
      CALL MPI_SEND (B(IST1), KNT1, MPI_REAL8, IDEST, 10, COMM, IERR)
      CALL MPI_SEND (C(IST1), KNT1, MPI_REAL8, IDEST, 20, COMM, IERR)
      CALL MPI_SEND (D(IST1), KNT1, MPI_REAL8, IDEST, 30, COMM, IERR)
   ENDDO
ELSE
   KNT=GCOUNT(MYID)
   CALL MPI_RECV (B(ISTART), KNT, MPI_REAL8, 0, 10,COMM, ISTAT, IERR)
   CALL MPI_RECV (C(ISTART), KNT, MPI_REAL8, 0, 20,COMM, ISTAT, IERR)
   CALL MPI_RECV (D(ISTART), KNT, MPI_REAL8, 0, 30,COMM, ISTAT, IERR)
 ENDIF
```

# 資料不切割的平行程式T3CP(4)

```
   CALL MPI_SENDRECV (B(IEND),        1, MPI_REAL8,  R_NBR, 110,
  1                   B(ISTARTM1), 1, MPI_REAL8,  L_NBR, 110,
  2                   COMM, ISTAT, IERR)
   CALL MPI_SENDRECV (B(ISTART), 1, MPI_REAL8,  L_NBR, 120,
  1                   B(IENDP1), 1, MPI_REAL8,  R_NBR, 120,
  2                   COMM, ISTAT, IERR)
     AMAX=-1.0D12
C    DO I=2,NTOTAL-1
     DO I=ISTART2,IEND1
        A(I)=C(I)*D(I)+(B(I-1)+2.0*B(I)+B(I+1))*0.25
        AMAX=MAX(AMAX,A(I))
     ENDDO
```

# 資料不切割的平行程式T3CP(5)

```
ITAG=110
IF(MYID.NE.0) THEN
   KNT=GCOUNT(MYID)
   CALL MPI_SEND (A(ISTART), KNT, MPI_REAL8, 0, ITAG, COMM, IERR)
ELSE
  DO ISRC=1, LASTP
      IST1=GSTART(ISRC)
      KNT1=GCOUNT(ISRC)
      CALL MPI_RECV (A(IST1), KNT1, MPI_REAL8, ISRC, ITAG,
                              COMM, ISTAT, IERR)
   ENDDO
ENDIF
```

# 資料不切割的平行程式T3CP(6)

```
      CALL MPI_REDUCE (AMAX, GMAX, 1, MPI_REAL8, MPI_MAX,
     1                 COMM, IERR)
        IF(MYID.EQ.0) THEN
          WRITE(*,101) (A(I),I=1,NTOTAL,5)
          WRITE(*,102) GMAX
        ENDIF
101   FORMAT(10F8.3)
102   FORMAT('MAXIMUM VALUE of ARRAY A is', E15.5)
        CALL MPI_FINALIZE(IERR)
        STOP
        END
```

# 資料不切割平行程式T3CP的測試結果

```
ATTENTION: 0031-408  4 nodes allocated by LoadLeveler, continuing...
  NPROC,MYID,ISTART,IEND= 4  3  151  200
  NPROC,MYID,ISTART,IEND= 4  2  101  150
  NPROC,MYID,ISTART,IEND= 4  1   51  100
  NPROC,MYID,ISTART,IEND= 4  0    1   50
    .000   3.063   2.563   2.383   2.290   2.234   2.196   2.168   2.148   2.131
   2.118   2.108   2.099   2.091   2.085   2.079   2.074   2.070   2.066   2.063
   2.060   2.057   2.054   2.052   2.050   2.048   2.046   2.044   2.043   2.041
   2.040   2.039   2.037   2.036   2.035   2.034   2.033   2.032   2.031   2.031
MAXIMUM VALUE OF ARRAY A is      .57500E+01
```

# 資料不切割的輸入資料分送

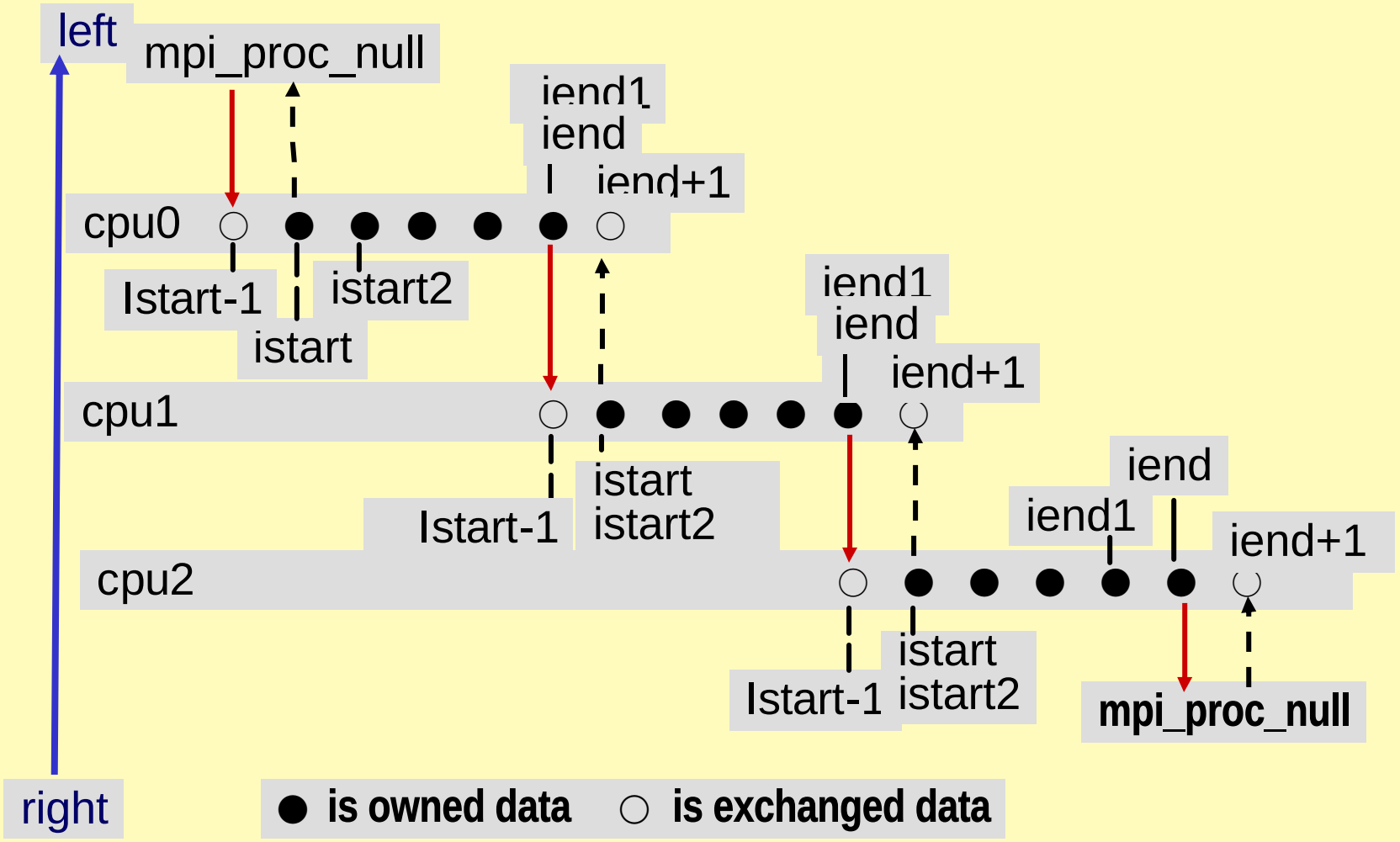
- 在陣列不切割的平行程式裏，CPU0讀入資料之後也可以使用MPI_BCAST把整個陣列傳送給各個CPU。

```
IF(MYID.EQ.0) THEN
    OPEN(7,FILE='input.dat',STATUS='OLD',FORM='UNFORMATTED')
    READ (7) B
    READ (7) C
    READ (7) D
 ENDIF
 IROOT=0
 ALL MPI_BCAST ( B, NTOTAL, MPI_REAL8, IROOT,
 1                          MPI_COMM_WORLD, IERR)
 CALL MPI_BCAST( C, NTOTAL, MPI_REAL8, IROOT,
 1                           MPI_COMM_WORLD, IERR)
 CALL MPI_BCAST( D, NTOTAL, MPI_REAL8, IROOT,
 1                           MPI_COMM_WORLD, IERR)
```

# 需要邊界資料交換的平行程式

- MPI_SENDRECV、 MPI_BCAST
- 需要邊界資料交換的循序程式T3SEQ
- 資料不切割的平行程式T3CP
- **交換一個陣列元素的資料切割平行程式 T3DCP_1**
- 交換兩個陣列元素的資料切割平行程式 T3DCP_2

# 資料切割的邊界資料交換示意圖



# 資料切割的平行程式T3DCP_1(1)

```
      PROGRAM T3DCP_1
C
CC    Exchange 1 element  both on right and left hand side
C
      PARAMETER (NTOTAL=200, NP=4, N=NTOTAL/NP)
      INCLUDE  'mpif.h'
      REAL*8    A(0:N+1), B(0:N+1), C(0:N+1), D(0:N+1), T(NTOTAL),
     1          AMAX, GMAX
      INTEGER  NPROC, MYID, ISTAT(MPI_STATUS_SIZE), ISTART, IEND,
     1          L_NBR, R_NBR

      CALL MPI_INIT (IERR)
      CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NPROC, IERR)
      CALL MPI_COMM_RANK (MPI_COMM_WORLD, MYID, IERR)
```

# 資料切割的平行程式T3DCP_1(2)

```
      ISTART=1
      IEND=N
      ISTARTM1=ISTART-1
      IENDP1=IEND+1
      ISTART2=1
      IF(MYID.EQ.0) ISTART2=2
      IEND1=N
      IF(MYID.EQ.NPROC-1) IEND1=N-1
       L_NBR=MYID-1
       R_NBR=MYID+1
       IF(MYID.EQ.0)              L_NBR=MPI_PROC_NULL
       IF(MYID.EQ.NPROC-1) R_NBR=MPI_PROC_NULL
C
```

# 資料切割的平行程式T3DCP_1(3)

```
  IF(MYID.EQ.0) THEN
       OPEN(7,FILE='input.dat',STATUS='OLD',FORM='UNFORMATTED')
       READ (7) T            ! read array B
  ENDIF
  IROOT=0
  CALL MPI_SCATTER (T, N, MPI_REAL8,  B(1), N, MPI_REAL8,
1                              IROOT, MPI_COMM_WORLD, IERR)
  IF(MYID.EQ.0) THEN
       READ (7) T            ! read array C
  ENDIF
  CALL MPI_SCATTER (T, N, MPI_REAL8,  C(1), N, MPI_REAL8,
1                              IROOT, MPI_COMM_WORLD, IERR)
  IF(MYID.EQ.0) THEN
       READ (7) T            ! read array D
  ENDIF
   CALL MPI_SCATTER (T, N, MPI_REAL8,  D(1), N, MPI_REAL8,
   1                            IROOT, MPI_COMM_WORLD, IERR)
```

# 資料切割的平行程式T3DCP_1(4)

```
      ITAG=110
      CALL MPI_SENDRECV (B(IEND),         1, MPI_REAL8,  R_NBR, ITAG,
     1                   B(ISTARTM1), 1, MPI_REAL8,  L_NBR, ITAG,
     2                   MPI_COMM_WORLD, ISTAT, IERR)
       ITAG=120
      CALL MPI_SENDRECV (B(ISTART), 1, MPI_REAL8,  L_NBR, ITAG,
     1                   B(IENDP1), 1, MPI_REAL8,  R_NBR, ITAG,
     2                   MPI_COMM_WORLD, ISTAT, IERR)
C
CC    Compute and output result
C
      AMAX=-1.0D12
CC  DO I=2,NTOTAL-1
      DO I=ISTART2, IEND1
          A(I)=C(I)*D(I)+(B(I-1)+2.0*B(I)+B(I+1))*0.25
          AMAX=MAX(AMAX,A(I))
      ENDDO
```

# 資料切割的平行程式T3DCP_1(5)

```
      CALL MPI_GATHER (A(1), N, MPI_REAL8,  T, N, MPI_REAL8,
     1                                 0, MPI_COMM_WORLD, IERR)
      CALL MPI_REDUCE (AMAX, GMAX, 1, MPI_REAL8, MPI_MAX,
     1                                  0, MPI_COMM_WORLD, IERR)
      IF(MYID.EQ.0) THEN
          WRITE(*,101) (T(I),I=1,NTOTAL,5)
          WRITE(*,102)  GMAX
      ENDIF
101   FORMAT(10F8.3)
102   FORMAT('MAXIMUM VALUE OF ARRAY A is', E15.5)
      CALL MPI_FINALIZE (IERR)
      STOP
      END
```

# 資料切割平行程式T3DCP_1的測試結果

```
ATTENTION: 0031-408  4 nodes allocated by LoadLeveler, continuing...
     .000   3.063   2.563   2.383   2.290   2.234   2.196   2.168   2.148   2.131
    2.118   2.108   2.099   2.091   2.085   2.079   2.074   2.070   2.066   2.063
    2.060   2.057   2.054   2.052   2.050   2.048   2.046   2.044   2.043   2.041
    2.040   2.039   2.037   2.036   2.035   2.034   2.033   2.032   2.031   2.031
 MAXIMUM VALUE OF ARRAY A is     .57500E+01
```

# 需要邊界資料交換的平行程式

- MPI_SENDRECV、 MPI_BCAST
- 需要邊界資料交換的循序程式T3SEQ
- 資料不切割的平行程式T3CP
- 交換一個陣列元素的資料切割平行程式 T3DCP_1
- **交換兩個陣列元素的資料切割平行程式 T3DCP_2**

# 交換兩個邊界資料的平行程式(1)

- 如果3.4節裏程式T3DCP_1的DO loop需要用到兩個鄰近陣列元素時，如下列所示：

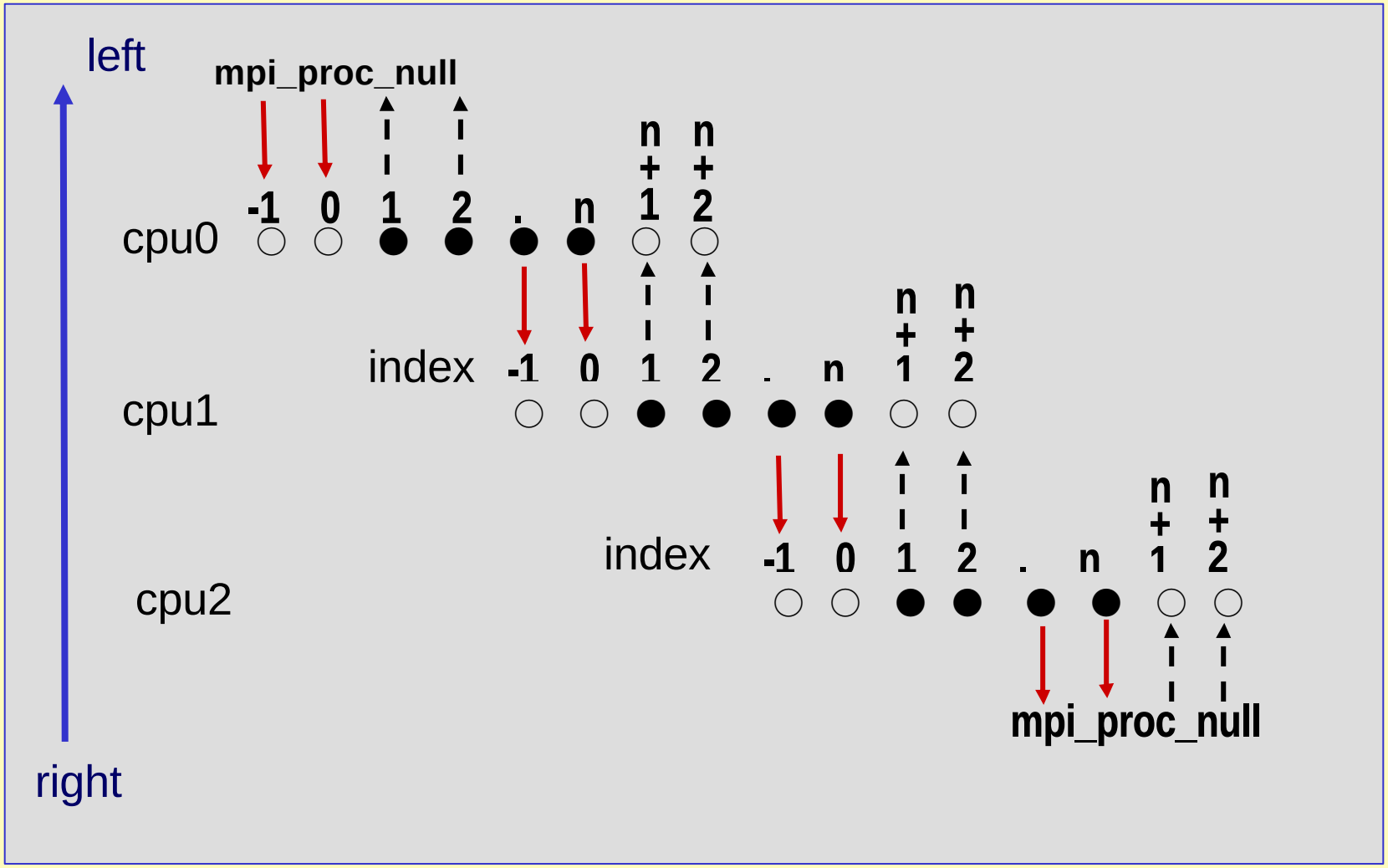
```
DO I=3,NTOTAL-2
  A(I)=C(I)*D(I)+(  B(I-2)+2.0*B(I-1)+2.0*B(I)+2.0*B(I+1)+B(I+2))*0.125
ENDDO
```

- 其平行化方法與程式T3DCP_1相同。只要把切割後陣列的DIMENSION前後各加兩個陣列元素，把DIMENSION改為 (-1:N+2)，其轄區內資料仍然是存放在index 1到N。

# 交換兩個邊界資料的平行程式(2)

```
 REAL*8   A(-1:N+2), B(-1:N+2), C(-1:N+2),
&         D(-1:N+2), T(NTOTAL), AMAX, GMAX

  ISTART=1
  IEND=N
  ISTART3=1
  IF(MYID.EQ.0) ISTART3=3
  IEND2=IEND
  IF(MYID.EQ.NPROC-1) IEND2=IEND-2
```

# 交換兩個邊界資料的示意圖



# 交換兩個邊界資料的 SENDRECV(1)

```
IENDM1=IEND-1
ISTARTM2=ISTART-2
```

- 邊界資料的交換量也要由一個改為兩個。每一個CPU要把轄區內最後兩個陣列元素送給右邊的CPU，這兩個陣列元素的起點是iend-1，同時要接受來自左邊CPU的兩個陣列元素，從istart-2放起。所以：

```
 ITAG=110
 CALL MPI_SENDRECV (
1        B(IENDM1),    2, MPI_REAL8,  R_NBR, ITAG,
2        B(ISTARTM2), 2, MPI_REAL8,  L_NBR, ITAG,
3        MPI_COMM_WORLD, ISTAT, IERR)
```

# 交換兩個邊界資料的 SENDRECV(2)

每一個CPU也要把轄區內最前面兩個陣列元素送給左邊的CPU，這兩個陣列元素的起點是istart，同時要接受來自右邊CPU的兩個陣列元素，從iend+1放起。所以：

```
 IENDP1=IEND+1
 ITAG=120
 CALL MPI_SENDRECV (
1        B(ISTART), 2, MPI_REAL8,  L_NBR,  ITAG,
2        B(IENDP1), 2, MPI_REAL8,  R_NBR,  ITAG,
3        MPI_COMM_WORLD, ISTAT, IERR)
```

# 交換兩個邊界資料的 T3DCP_2(1)

```
      PROGRAM T3DCP_2
C
      PARAMETER (NTOTAL=200,NP=4,N=NTOTAL/NP)
      INCLUDE  'mpif.h'
      REAL*8   A(-1:N+2), B(-1:N+2), C(-1:N+2), D(-1:N+2), T(NTOTAL),
     &              AMAX, GMAX
      INTEGER  NPROC, MYID, ISTAT(MPI_STATUS_SIZE), ISTART, IEND,
     1              L_NBR, R_NBR

      CALL MPI_INIT (IERR)
      CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NPROC, IERR)
      CALL MPI_COMM_RANK (MPI_COMM_WORLD, MYID, IERR)
```

# 交換兩個邊界資料的 T3DCP_2(2)

```
      ISTART=1
      ISTARTM2=ISTART-2
      ISTART3=1
      IF(MYID.EQ.0) ISTART3=3
      IEND=N
      IEND2=IEND
      IF(MYID.EQ.NPROC-1) IEND2=IEND-2
      IENDM1=IEND-1
      IENDP1=IEND+1
       L_NBR=MYID-1
      R_NBR=MYID+1
      IF(MYID.EQ.0)             L_NBR=MPI_PROC_NULL
      IF(MYID.EQ.NPROC-1) R_NBR=MPI_PROC_NULL
C
```

# 交換兩個邊界資料的 T3DCP_2(3)

```
 IF(MYID.EQ.0) THEN
     OPEN(7,FILE='input.dat',STATUS='OLD',FORM='UNFORMATTED')
     READ (7) T            ! read array B
 ENDIF
 IROOT=0
 CALL MPI_SCATTER (T, N, MPI_REAL8,  B(1), N, MPI_REAL8,
1                            IROOT, MPI_COMM_WORLD, IERR)
 IF(MYID.EQ.0) THEN
     READ (7) T            ! read array C
 ENDIF
 CALL MPI_SCATTER (T, N, MPI_REAL8,  C(1), N, MPI_REAL8,
1                            IROOT, MPI_COMM_WORLD, IERR)
  IF(MYID.EQ.0) THEN
       READ (7) T            ! read array D
  ENDIF
  CALL MPI_SCATTER (T, N, MPI_REAL8,  D(1), N, MPI_REAL8,
 1                            IROOT, MPI_COMM_WORLD, IERR)
```

# 交換兩個邊界資料的 T3DCP_2(4)

```
       ITAG=110
       CALL MPI_SENDRECV (B(IENDM1),     2, MPI_REAL8,  R_NBR, ITAG,
      1                   B(ISTARTM2), 2, MPI_REAL8,  L_NBR, ITAG,
      2                   MPI_COMM_WORLD, ISTAT, IERR)
       ITAG=120
       CALL MPI_SENDRECV (B(ISTART), 2, MPI_REAL8,  L_NBR, ITAG,
      1                   B(IENDP1), 2, MPI_REAL8,  R_NBR, ITAG,
      2                   MPI_COMM_WORLD, ISTAT, IERR)
C
       AMAX=-1.0D12
CC     DO I=3,NTOTAL-2
       DO I=ISTART3,IEND2
          A(I)=C(I)*D(I)+( B(I-2)+2.0*B(I-1)+2.0*B(I)
     &                  +2.0*B(I+1)+B(I+2) )*0.125
          AMAX=MAX(AMAX,A(I))
       ENDDO
```

# 交換兩個邊界資料的 T3DCP_2(5)

```
      IDEST=0
     CALL MPI_GATHER (A(1), N, MPI_REAL8,  T, N, MPI_REAL8,
    1                              IDEST, MPI_COMM_WORLD, IERR)
     CALL MPI_ALLREDUCE (AMAX, GMAX, 1, MPI_REAL8, MPI_MAX,
    1                                  MPI_COMM_WORLD, IERR)
     AMAX=GMAX
      IF(MYID.EQ.0) THEN
            WRITE(*,101) (T(I),I=1,NTOTAL,5)
            WRITE(*,102) AMAX
       ENDIF
101   FORMAT(10F8.3)
102   FORMAT('MAXIMUM VALUE OF ARRAY A is ', E15.5)
       CALL MPI_FINALIZE (IERR)
       STOP
       END
```

# 交換兩個邊界資料的T3DCP_2的測試結果

ATTENTION: 0031-408  4 nodes allocated by LoadLeveler, continuing...

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| .000 | 3.078 | 2.565 | 2.384 | 2.291 | 2.234 | 2.196 | 2.168 | 2.148 | 2.131 |
| 2.118 | 2.108 | 2.099 | 2.091 | 2.085 | 2.079 | 2.074 | 2.070 | 2.066 | 2.063 |
| 2.060 | 2.057 | 2.054 | 2.052 | 2.050 | 2.048 | 2.046 | 2.044 | 2.043 | 2.041 |
| 2.040 | 2.039 | 2.037 | 2.036 | 2.035 | 2.034 | 2.033 | 2.032 | 2.031 | 2.031 |

MAXIMUM VALUE OF ARRAY A is  .44847E+01

# 課 程 大 綱

1. 前言
2. 無邊界資料交換的平行程式
3. 需要邊界資料交換的平行程式
4. **格點數不能整除的平行程式**
5. 多維陣列的平行程式
6. MPI平行程式的效率提昇

# 格點數不能整除的平行程式

- 循序程式T4SEQ裏陣列的dimension是161，只能夠被7和23整除
- 集體通訊指令 MPI_SCATTERV 和 MPI_GATHERV
- 資料集結與分解指令 MPI_PACK 和 MPI_UNPACK，以及同步指令 MPI_BARRIER 和取得時鐘時刻指令 MPI_WTIME
- 使用這些指令來把循序程式 T4SEQ 改寫成平行程式 T4DCP

# 格點數不能整除的循序程式T4SEQ(1)

```
      PROGRAM    T4SEQ
      PARAMETER (NTOTAL=161)
      REAL*8    A(NTOTAL), B(NTOTAL), C(NTOTAL),
     1           D(NTOTAL), P, Q, R
      DATA      P, Q, R/1.45, 2.62, 0.5/

      DO I=1,NTOTAL
         B(I)=3.D0/DFLOAT(I)+1.D0
         C(I)=2.D0/DFLOAT(I)+1.D0
         D(I)=1.D0/DFLOAT(I)+1.D0
      ENDDO
```

# 格點數不能整除的循序程式T4SEQ(2)

```
OPEN(1,FILE='input.dat',FORM='UNFORMATTED')
 WRITE(1) B
 WRITE(1) C
 WRITE(1) D
 WRITE(1) P,Q,R
 CLOSE(1)
 OPEN(1,FILE='input.dat',STATUS='OLD',FORM='UNFORMATTED')
 READ (1) B
 READ (1) C
 READ (1) D
 READ (1) P,Q,R
```

# 格點數不能整除的循序程式T4SEQ(3)

```
C
      DO I=2,NTOTAL-1
         A(I)=C(I)*D(I)*P+(B(I-1)+2.0*B(I)+B(I+1))*Q+R
      ENDDO
      WRITE(*,101) (A(I),I=1,NTOTAL,5)
101   FORMAT(10F8.3)
      STOP
      END
```

# 格點數不能整除循序程式T4SEQ的測試結果

```
   .000  18.550  15.720  14.682  14.143  13.812  13.588  13.427  13.305  13.210
13.133  13.070  13.018  12.973  12.935  12.901  12.872  12.847  12.824  12.803
12.785  12.768  12.753  12.739  12.726  12.714  12.703  12.693  12.684  12.675
12.667  12.660   .000
```

# 格點數不能整除的平行程式

- 循序程式T4SEQ裏陣列的dimension是161，只能夠被7和23整除
- 集體通訊指令 MPI_SCATTERV 和 MPI_GATHERV
- 資料集結與分解指令 MPI_PACK 和 MPI_UNPACK，以及同步指令 MPI_BARRIER 和取得時鐘時刻指令 MPI_WTIME
- 使用這些指令來把循序程式 T4SEQ 改寫成平行程式 T4DCP

# 集體通訊指令MPI_SCATTERV、 MPI_GATHERV(1)

```
 IROOT=0
 CALL MPI_SCATTERV( T, GCOUNT, GDISP, MPI_REAL8,
1                           C(1), MYCOUNT, MPI_REAL8,
2                           IROOT, MPI_COMM_WORLD, IERR)

 CALL MPI_GATHERV (A(1), MYKOUNT, MPI_REAL8,
1                          T, GCOUNT, GDISP, MPI_REAL8,
2                          IROOT, MPI_COMM_WORLD, IERR)
```

# 集體通訊指令MPI_SCATTERV、 MPI_GATHERV(2)

- GCOUNT　整數陣列，存放要送給各個CPU的資料數量
- GDISP　整數陣列，存放要送給各個CPU資料起點在T陣列上的相對位置

```
  CALL STARTEND (NPROC, 1, NTOTAL, GSTART,
1                          GEND, GCOUNT)

  DO I=0,NPROC-1
     GDISP(I)=GSTART(I)-1
  ENDDO
```

# 格點數不能整除的平行程式

- 循序程式T4SEQ裏陣列的dimension是161，只能夠被7和23整除
- 集體通訊指令 MPI_SCATTERV 和 MPI_GATHERV
- 資料集結與分解指令 MPI_PACK 和 MPI_UNPACK，以及同步指令 MPI_BARRIER 和取得時鐘時刻指令 MPI_WTIME
- 使用這些指令來把循序程式 T4SEQ 改寫成平行程式 T4DCP

# 資料集結與分解指令MPI_PACK、 MPI_UNPACK(1)

```
INTEGER       BUFSIZE
PARAMETER (BUFSIZE=24)
CHARACTER BUF1(BUFSIZE)

IF (MYID.EQ.0) THEN
   READ(7) P,Q,R
   IPOS = 0
   CALL MPI_PACK (P,1, MPI_REAL8, BUF1, BUFSIZE, IPOS, COMM, IERR)
   CALL MPI_PACK (Q,1,MPI_REAL8, BUF1, BUFSIZE, IPOS, COMM, IERR)
   CALL MPI_PACK (R,1,MPI_REAL8, BUF1, BUFSIZE, IPOS, COMM, IERR)
ENDIF
```

# 資料集結與分解指令MPI_PACK、 MPI_UNPACK(2)

```
CALL MPI_BCAST (BUF1, BUFSIZE, MPI_CHARACTER, 0, COMM, IERR)
IF (MYID.NE.0) THEN
   IPOS=0
   CALL MPI_UNPACK (BUF1, BUFSIZE, IPOS, P, 1, MPI_REAL8, COMM, IERR)
   CALL MPI_UNPACK (BUF1, BUFSIZE, IPOS, Q, 1, MPI_REAL8, COMM, IERR)
   CALL MPI_UNPACK (BUF1, BUFSIZE, IPOS, R, 1, MPI_REAL8, COMM, IERR)
ENDIF
```

# 人工 PACK、UNPACK

```
REAL*8   BUFF(3), P, Q, R

IF (MYID.EQ.0) THEN
   READ(7) P,Q,R
    BUFF(1)=P
    BUFF(2)=Q
    BUFF(3)=R
ENDIF
IROOT=0
CALL MPI_BCAST (BUFF, 3, MPI_REAL8, IROOT, COMM, IERR)
 IF (MYID.GT.0) THEN
     P=BUFF(1)
     Q=BUFF(2)
     R=BUFF(3)
 ENDIF
```

# MPI_BARRIER, MPI_WTIME(1)

- MPI_BARRIER是屬於 集體通訊 ’類副程式，提供屬於同一個communicator的所有CPU 達到 同步 ’(synchronized) 的狀況。
- 如果想要知道平行程式執行時的時鐘時刻(wall clock time)，可利用 MPI 涵數 MPI_WTIME，其計量單位是 秒鐘 ’。
- 如果想要知道參與平行計算各個CPU花費多少時間來完成平行程式的執行工作，可以在叫用 MPI_INIT 之後取得一個 MPI_WTIME，然後在叫用MPI_FINALIZE 之前再取得一個 MPI_WTIME，後者減掉前者，就得到兩者之間所經歷的時間，例如：

# MPI_BARRIER, MPI_WTIME(2)

```
REAL*8    TIME1, TIME2
CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NPROC, IERR)
CALL MPI_COMM_RANK (MPI_COMM_WORLD, MYID, IERR)
CALL MPI_BARRIER  (MPI_COMM_WORLD, IERR)
TIME1=MPI_WTIME()
 . . .
TIME2=MPI_WTIME() – TIME1
PRINT *,' MYID, CLOCK TIME=', MYID, TIME2
CALL MPI_FINALIZE(IERR)
STOP
END
```

# 格點數不能整除的平行程式

- 循序程式T4SEQ裏陣列的dimension是161，只能夠被7和23整除
- 集體通訊指令 MPI_SCATTERV 和 MPI_GATHERV
- 資料集結與分解指令 MPI_PACK 和 MPI_UNPACK，以及同步指令 MPI_BARRIER 和取得時鐘時刻指令 MPI_WTIME
- **使用這些指令來把循序程式 T4SEQ 改寫成平行程式 T4DCP**

# 格點數不能整除的程式T4DCP(1)

```
      PROGRAM T4DCP
      PARAMETER (NTOTAL=161, NP=4, N=NTOTAL/NP+1)
      INCLUDE  'mpif.h'
      REAL*8   A(0:N+1), B(0:N+1), C(0:N+1), D(0:N+1), T(NTOTAL)

      INTEGER  NPROC, MYID, ISTAT(MPI_STATUS_SIZE), ISTART, IEND,
     1              L_NBR, R_NBR, GCOUNT(0:NP), GDISP(0:NP), GEND(0:NP)
      CALL MPI_INIT (IERR)
      CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NPROC, IERR)
      CALL MPI_COMM_RANK (MPI_COMM_WORLD, MYID, IERR)
      CALL MPI_BARRIER  (MPI_COMM_WORLD, IERR)
```

# 格點數不能整除的程式T4DCP(2)

```
TIME1=MPI_WTIME()
 CALL STARTEND (NPROC,1,NTOTAL,GSTART,GEND, GCOUNT)
 DO I=0,NPROC-1
     GDISP(I)=GSTART(I)-1
 ENDDO
 MYKOUNT=GCOUNT(MYID)
 PRINT *,' NPROC,MYID,ISTART,IEND,ARRAY LENGTH=', NPROC,MYID,
&       GSTART(MYID), GEND(MYID), MYCOUNT
 ISTART=1
 IEND= MYKOUNT
```

# 格點數不能整除的程式T4DCP(3)

```
      ISTART2=ISTART
       IF(MYID.EQ.0) ISTART2=2
       IEND1=IEND
       IF(MYID.EQ.NPROC-1) IEND1=IEND-1
       ISTARTM1=ISTART-1
       IENDP1=IEND+1
       L_NBR=MYID-1
       R_NBR=MYID+1
       IF(MYID.EQ.0)              L_NBR=MPI_PROC_NULL
       IF(MYID.EQ.NPROC-1) R_NBR=MPI_PROC_NULL
C
```

# 格點數不能整除的程式T4DCP(4)

```
IF(MYID.EQ.0) THEN
      OPEN(7,FILE='input.dat',STATUS='OLD',FORM='UNFORMATTED')
      READ (7) T            ! read array B
  ENDIF
  IROOT=0
  CALL MPI_SCATTERV (T, GCOUNT, GDISP, MPI_REAL8, B(1),
 1          MYKOUNT, MPI_REAL8, IROOT, MPI_COMM_WORLD, IERR)
  IF(MYID.EQ.0) THEN
      READ (7) T            ! read array C
  ENDIF
```

# 格點數不能整除的程式T4DCP(5)

```
 CALL MPI_SCATTERV(T, GCOUNT, GDISP, MPI_REAL8, C(1), MYKOUNT,
1                          MPI_REAL8, IROOT, MPI_COMM_WORLD, IERR)
 IF(MYID.EQ.0) THEN
      READ (7) T            ! read array D
 ENDIF
 CALL MPI_SCATTERV (T, GCOUNT, GDISP, MPI_REAL8, D(1),
1            MYKOUNT, MPI_REAL8, IROOT, MPI_COMM_WORLD, IERR)
```

# 格點數不能整除的程式T4DCP(6)

```
 ITAG=110
 CALL MPI_SENDRECV (B(IEND),        1, MPI_REAL8,  R_NBR, ITAG,
1                                B(ISTARTM1), 1, MPI_REAL8,  L_NBR, ITAG,
2                                 MPI_COMM_WORLD, ISTAT, IERR)
 ITAG=120
 CALL MPI_SENDRECV (B(ISTART), 1, MPI_REAL8,  L_NBR, ITAG,
1                                B(IENDP1), 1, MPI_REAL8,  R_NBR, ITAG,
2                                MPI_COMM_WORLD, ISTAT, IERR)
```

# 格點數不能整除的程式T4DCP(7)

```
CC  DO I=2,NTOTAL-1
      DO I=ISTART2,IEND1
         A(I)=C(I)*D(I)+(B(I-1)+2.0*B(I)+B(I+1))*0.25
      ENDDO
      CALL MPI_GATHERV (A(1), MYKOUNT, MPI_REAL8,
     1                                T, GCOUNT, GDISP, MPI_REAL8,
     2                                IROOT, MPI_COMM_WORLD, IERR)
      IF(MYID.EQ.0) THEN
        WRITE(*,101) (T(I),I=1,NTOTAL,5)
      ENDIF
101 FORMAT(10F8.3)
      TIME2=MPI_WTIME()-TIME1
      PRINT *,' MYID, CLOCK TIME=', MYID,TIME2
      CALL MPI_FINALIZE(IERR)
      STOP
      END
```

# 格點數不能整除程式T4DCP的測試結果

```
NPROC,MYID,ISTART,IEND,ARRAY LENGTH= 4  0   1   41  41
NPROC,MYID,ISTART,IEND ARRAY LENGTH = 4  2  82  121  40
NPROC,MYID,ISTART,IEND ARRAY LENGTH = 4  1  42   81  40
NPROC,MYID,ISTART,IEND ARRAY LENGTH = 4  3  122  161  40
MYID, CLOCK TIME= 2  0.8412782848E-01
MYID, CLOCK TIME= 1  0.8406087756E-01
MYID, CLOCK TIME= 3  0.8399387449E-01
0.0000  18.550  15.720  14.682  14.143  13.812  13.588  13.427  13.305  13.210
13.133  13.070  13.018  12.973  12.935  12.901  12.872  12.847  12.824  12.803
12.785  12.768  12.753  12.739  12.726  12.714  12.703  12.693  12.684  12.675
12.667  12.660    .000
MYID, CLOCK TIME= 0  0.1014785469
```

- 從各個CPU列印出來的時間可看出CPU1、CPU2、CPU3所耗用的時間相差無幾，而CPU0耗用較多。這是合理的，因為它比其他CPU多作一些工作，像輸入資料的讀入與輸出資料的寫出等。

# 課 程 大 綱

1. 前言
2. 無邊界資料交換的平行程式
3. 需要邊界資料交換的平行程式
4. 格點數不能整除的平行程式
5. **多維陣列的平行程式**
6. MPI平行程式的效率提昇

# 多維陣列的平行程式

- 循序程式T5SEQ，該程式是由一個副程式改寫而成，陣列的index順序也由原來的 (I,J,K) 改寫為 (K,J,I)
- 資料不切割的平行程式T5CP
- 資料切割的平行程式 T5DCP
- 與二維切割有關的MPI指令
- 末二維切割的平行程式T5_2D

# 多維陣列的平行程式T5SEQ(1)

```
      PARAMETER  (KK=20,NN=120,MM=160,KM=3,MM1=MM-1,NN1=NN-1)
      COMMON  U1(KK,NN,MM),V1(KK,NN,MM),PS1(NN,MM),
     1        F1(KM,NN,MM),F2(KM,NN,MM),
     2        HXU(NN,MM),HXV(NN,MM),HMMX(NN,MM),HMMY(NN,MM),
     3        VECINV(KK,KK),AM7(KK),SUMF1,SUMF2

      PROGRAM  T5SEQ
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE  'varseq '
      CALL SYSTEM_CLOCK(IC,IR,IM)
      CLOCK=DBLE(IC)/DBLE(IR)
      CALL DATAGEN
      CALL COMPUTATION
      CALL OUTPUT
      CALL SYSTEM_CLOCK(IC,IR,IM)
      CLOCK=DBLE(IC)/DBLE(IR)-CLOCK
      PRINT *,'CLOCK TIME=', CLOCK
      END
```

檔案 varseq 之內容

# 多維陣列的平行程式T5SEQ(2)

```
      SUBROUTINE DATAGEN
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE  'varseq '
      DO 10 I=1,MM1
      DO 10 J=1,NN
      DO 10 K=1,KK
          U1(K,J,I)=1.D0/DFLOAT(I)+1.D0/DFLOAT(J)+1.D0/DFLOAT(K)
10    CONTINUE
      DO 20 I=1,MM
      DO 20 J=1,NN1
      DO 20 K=1,KK
          V1(K,J,I)=2.D0/DFLOAT(I)+1.D0/DFLOAT(J)+1.D0/DFLOAT(K)
20    CONTINUE
```

# 多維陣列的平行程式T5SEQ(3)

```
      DO 30 I=1,MM
      DO 30 J=1,NN
          PS1(J,I)=1.D0/DFLOAT(I)+1.D0/DFLOAT(J)
          HXU(J,I)=2.D0/DFLOAT(I)+1.D0/DFLOAT(J)
          HXV(J,I)=1.D0/DFLOAT(I)+2.D0/DFLOAT(J)
          HMMX(J,I)=2.D0/DFLOAT(I)+1.D0/DFLOAT(J)
          HMMY(J,I)=1.D0/DFLOAT(I)+2.D0/DFLOAT(J)
30    CONTINUE
      DO 40 K=1,KK
      AM7(K)=1.D0/DFLOAT(K)
      DO 40 KA=1,KK
          VECINV(KA,K)=1.D0/DFLOAT(KA)+1.D0/DFLOAT(K)
40    CONTINUE
      END
```

# 多維陣列的平行程式T5SEQ(4)

```
      SUBROUTINE COMPUTATION
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE ‘varseq ’
      DIMENSION   D7(NN,MM),  D8(NN,MM),  D00(KK,NN,MM)
      DO 210 I=1,MM
      DO 210 J=1,NN
      DO 210 K=1,KM
           F1(K,J,I)=0.0D0
           F2(K,J,I)=0.0D0
210   CONTINUE
      DO 220 I=1,MM1
      DO 220 J=2,NN1
           D7(J,I)=(PS1(J,I+1)+PS1(J,I))*0.5D0*HXU(J,I)
220   CONTINUE
      DO 230 I=2,MM1
      DO 230 J=1,NN1
           D8(J,I)=(PS1(J+1,I)+PS1(J,I))*0.5D0*HXV(J,I)
230   CONTINUE
```

# 多維陣列的平行程式T5SEQ(5)

```
      DO 240 I=2,MM1
      DO 240 J=2,NN1
      DO 240 K=1,KK
         D00(K,J,I)=(D7(J,I)*U1(K,J,I)-D7(J,I-1)*U1(K,J,I-1))*HMMX(J,I)
     &                +(D8(J,I)*V1(K,J,I)-D8(J-1,I)*V1(K,J-1,I))*HMMY(J,I)
240   CONTINUE
      DO 260 I=2,MM1
      DO 260 KA=1,KK
      DO 260 J=2,NN1
      DO 260 K=1,KM
         F1(K,J,I)=F1(K,J,I)-VECINV(K,KA)*D00(KA,J,I)
 260 CONTINUE
```

# 多維陣列的平行程式T5SEQ(6)

```
      SUMF1=0.D0
      SUMF2=0.D0
      DO 270 I=2,MM1
      DO 270 J=2,NN1
      DO 270 K=1,KM
           F2(K,J,I)=-AM7(K)*PS1(J,I)
           SUMF1=SUMF1+F1(K,J,I)
           SUMF2=SUMF2+F2(K,J,I)
270   CONTINUE
      RETURN
      END
```

# 多維陣列的平行程式T5SEQ(7)

```
      SUBROUTINE OUTPUT
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE  ‘varseq ’
      PRINT *,' SUMF1,SUMF2=', SUMF1,SUMF2
      PRINT *,' F2(2,2,I),I=1,160,5'
      PRINT 301,(F2(2,2,I),I=1,160,5)
301   FORMAT(8E10.3)
      RETURN
      END
```

# 多維陣列的平行程式的測試結果

SUMF1,SUMF2= 26172.4605364288109  -2268.89180334316325

F2(2,2,I),I=1,160,5

.000E+00 -.333E+00 -.295E+00 -.281E+00 -.274E+00 -.269E+00 -.266E+00 -.264E+00

-.262E+00 -.261E+00 -.260E+00 -.259E+00 -.258E+00 -.258E+00 -.257E+00 -.257E+00

-.256E+00 -.256E+00 -.255E+00 -.255E+00 -.255E+00 -.255E+00 -.255E+00 -.254E+00

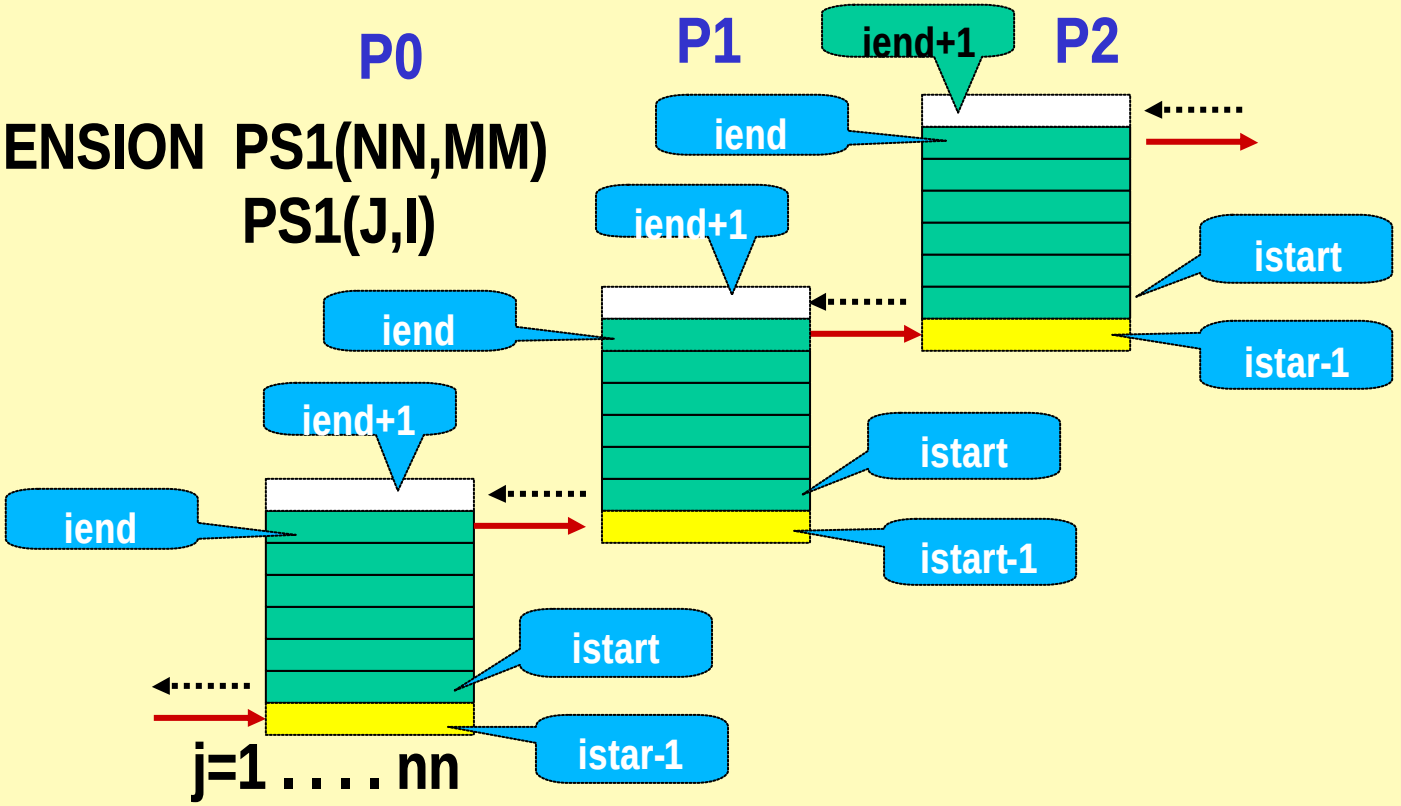
-.254E+00 -.254E+00 -.254E+00 -.254E+00 -.254E+00 -.253E+00 -.253E+00 -.253E+00

CLOCK TIME= 0.799999999981082510E-01

# 多維陣列的平行程式

- 循序程式T5SEQ，該程式是由一個副程式改寫而成，陣列的index順序也由原來的 (I,J,K) 改寫為 (K,J,I)
- **資料不切割的平行程式T5CP**
- 資料切割的平行程式 T5DCP
- 與二維切割有關的MPI指令
- 末二維切割的平行程式T5_2D

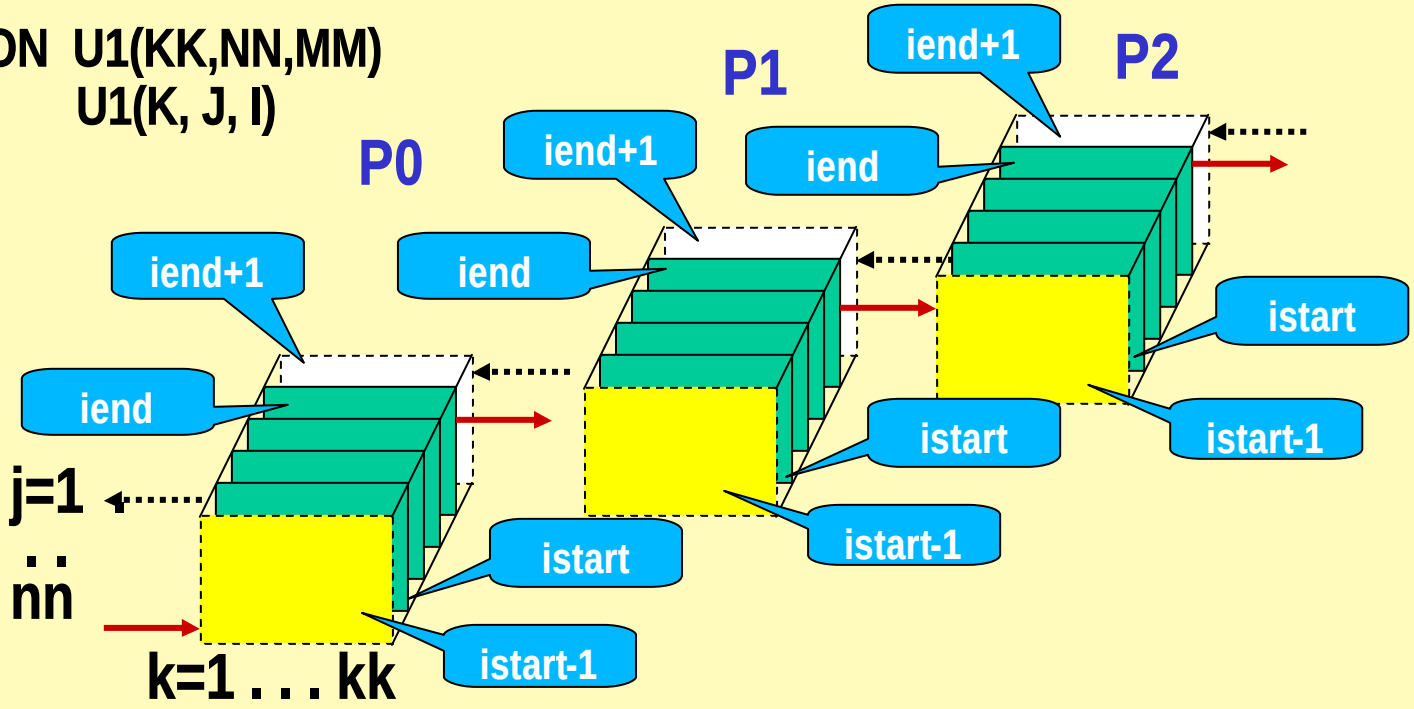
# 二維陣列末維切割示意圖




```
ITAG=20
CALL MPI_SENDRECV (PS1(1,ISTART), NN, MPI_REAL8, L_NBR, ITAG,
1                  PS1(1,IENDP1), NN, MPI_REAL8, R_NBR, ITAG,
 2                 MPI_COMM_WORLD, ISTAT, IERR)
```

# 三維陣列末維切割示意圖

DIMENSION U1(KK,NN,MM)
U1(K, J, I)




```
NNKK=NN*KK
CALL MPI_SENDRECV (U1(1,1,IEND), NNKK, MPI_REAL8, R_NBR, ITAG,
 1                  U1(1,1,ISTARTM1), NNKK, MPI_REAL8, L_NBR, ITAG,
 2                  MPI_COMM_WORLD, ISTAT, IERR)
```

# 資料不切割的平行程式T5CP(1)

檔案 varcp 之內容

```
      PARAMETER  (KK=20,NN=120,MM=160,KM=3,MM1=MM-1,NN1=NN-1)

      COMMON   U1(KK,NN,MM ),V1(KK,NN,MM ),PS1(NN,MM),
     1          F1(KM,NN,MM),F2(KM,NN,MM),
     2          HXU(NN,MM),HXV(NN,MM),HMMX(NN,MM),HMMY(NN,MM),
     3          VECINV(KK,KK),AM7(KK),SUMF1,SUMF2
C
      INTEGER             L_NBR,R_NBR,ISTAT(MPI_STATUS_SIZE),
     1                    GCOUNT, GSTART, GEND
      COMMON/MPIVAR/MYID,NPROC,L_NBR,R_NBR,
     1                    ISTART,ISTART2,ISTART3, IEND,IEND1,
     2                    ISTARTM1, IENDM1,IENDP1, MYDIM,
     3                    GSTART(0:50),GEND(0:50),GCOUNT(0:50)
```

# 資料不切割的平行程式T5CP(2)

```
PROGRAM  T5CP
IMPLICIT REAL*8 (A-H,O-Z)
INCLUDE 'mpif.h'
INCLUDE 'varcp'

CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NPROC,IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYID, IERR)
CALL MPI_BARRIER(MPI_COMM_WORLD,IERR)
CLOCK=MPI_WTIME()
CALL STARTEND(NPROC,1,MM,GSTART,GEND,GCOUNT)
ISTART=GSTART(MYID)
IEND=GEND(MYID)
```

# 資料不切割的平行程式T5CP(3)

```
      MYCOUNT=GCOUNT(MYID)
      PRINT 101, MYID,NPROC,ISTART,IEND
 101 FORMAT( ' MYID,NPROC,ISTART,IEND=', 4I6)
C-----------------------------------------------------------------
C       for DO I=x,MM1 (MM-1)
C-----------------------------------------------------------------
        IEND1=IEND
        IF( MYID.EQ.NPROC-1) IEND1=IEND-1
C-----------------------------------------------------------------
C       for DO I=2,x
C-----------------------------------------------------------------
        ISTART2=ISTART
        IF( MYID.EQ.0) ISTART2=2
C-----------------------------------------------------------------
C       for ghost point
C-----------------------------------------------------------------
        ISTARTM1=ISTART-1
        IENDP1=IEND+1
```

# 資料不切割的平行程式T5CP(4)

```
      L_NBR=MYID-1
      R_NBR=MYID+1
      IF(MYID.EQ.0)            L_NBR=MPI_PROC_NULL
      IF(MYID.EQ.NPROC-1) R_NBR=MPI_PROC_NULL
      CALL DATAGEN
      CALL COMPUTATION
      CALL OUTPUT
      CLOCK=MPI_WTIME() - CLOCK
      PRINT *,' MYID, CLOCK TIME=', MYID,CLOCK
      CALL MPI_FINALIZE(IERR)
      STOP
      END
```

# 資料不切割的平行程式T5CP(5)

```
SUBROUTINE STARTEND(NPROC,IS1,IS2,ISTART,IEND,ICOUNT)
 INTEGER NPROC,IS1,IS2,ISTART(0:31),IEND(0:31),ICOUNT(0:31)
 ILENGTH=IS2-IS1+1
 IBLOCK=ILENGTH/NPROC
 IR=ILENGTH-IBLOCK*NPROC
 DO I=0,NPROC-1
    IF(I.LT.IR) THEN
       ISTART(I)=IS1+I*(IBLOCK+1)
       IEND(I)=ISTART(I)+IBLOCK
    ELSE
       ISTART(I)=IS1+I*IBLOCK+IR
       IEND(I)=ISTART(I)+IBLOCK-1
    ENDIF
```

# 資料不切割的平行程式T5CP(6)

```
      IF(ILENGTH.LT.1) THEN
          ISTART(I)=1
          IEND(I)=0
      ENDIF
      ICOUNT(I)=IEND(I)-ISTART(I)+1
    ENDDO
    RETURN
    END
```

# 資料不切割的平行程式T5CP(7)

```
      SUBROUTINE   DATAGEN
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE  'mpif.h'
      INCLUDE  'varcp ‘
CCC DO 10 I=1,MM1
      DO 10 I=ISTART,IEND1
      DO 10 J=1,NN
      DO 10 K=1,KK
           U1(K,J,I)=1.D0/DFLOAT(I)+1.D0/DFLOAT(J)+1.D0/DFLOAT(K)
  10  CONTINUE
CCC DO 20 I=1,MM
      DO 20 I=ISTART,IEND
      DO 20 J=1,NN1
      DO 20 K=1,KK
              V1(K,J,I)=2.D0/DFLOAT(I)+1.D0/DFLOAT(J)+1.D0/DFLOAT(K)
  20  CONTINUE
```

# 資料不切割的平行程式T5CP(8)

```
CCC DO 30 I=1,MM
      DO 30 I=ISTART,IEND
      DO 30 J=1,NN
         PS1(J,I)=1.D0/DFLOAT(I)+1.D0/DFLOAT(J)
          HXU(J,I)=2.D0/DFLOAT(I)+1.D0/DFLOAT(J)
          HXV(J,I)=1.D0/DFLOAT(I)+2.D0/DFLOAT(J)
          HMMX(J,I)=2.D0/DFLOAT(I)+1.D0/DFLOAT(J)
          HMMY(J,I)=1.D0/DFLOAT(I)+2.D0/DFLOAT(J)
  30  CONTINUE
      DO 40 K=1,KK
      AM7(K)=1.D0/DFLOAT(K)
      DO 40 KA=1,KK
         VECINV(KA,K)=1.D0/DFLOAT(KA)+1.D0/DFLOAT(K)
  40  CONTINUE
      END
```

# 資料不切割的平行程式T5CP(9)

```
      SUBROUTINE COMPUTATION
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE 'mpif.h'
      INCLUDE 'varcp'
      DIMENSION   D7(NN,MM),D8(NN,MM),D00(KK,NN,MM)
      NNKK=NN*KK
      CALL MPI_SENDRECV(U1(1,1,IEND),       NNKK,MPI_REAL8,R_NBR,10,
     1                  U1(1,1,ISTARTM1),NNKK,MPI_REAL8,L_NBR,10,
     2                  MPI_COMM_WORLD,ISTAT,IERR)
      CALL MPI_SENDRECV(PS1(1,ISTART),NN,MPI_REAL8, L_NBR,20,
     1                  PS1(1,IENDP1),NN,MPI_REAL8, R_NBR,20,
     2                  MPI_COMM_WORLD,ISTAT,IERR)
```

# 資料不切割的平行程式T5CP(10)

```
CC    DO 210 I=1,MM
      DO 210 I=ISTART,IEND
      DO 210 J=1,NN
      DO 210 K=1,KM
           F1(K,J,I)=0.0D0
           F2(K,J,I)=0.0D0
 210 CONTINUE
```

# 資料不切割的平行程式T5CP(11)

```
CC    DO 220 I=1,MM1
      DO 220 I=ISTART,IEND1
      DO 220 J=2,NN1
          D7(J,I)=(PS1(J,I+1)+PS1(J,I))*0.5D0*HXU(J,I)
 220 CONTINUE
CC    DO 230 I=2,MM1
      DO 230 I=ISTART2,IEND1
      DO 230 J=1,NN1
          D8(J,I)=(PS1(J+1,I)+PS1(J,I))*0.5D0*HXV(J,I)
 230 CONTINUE
```

# 資料不切割的平行程式T5CP(12)

```
      ITAG=30
      CALL MPI_SENDRECV(D7(1,IEND),      NN,MPI_REAL8, R_NBR,ITAG,
     1                  D7(1,ISTARTM1),NN,MPI_REAL8, L_NBR,ITAG,
     2                  MPI_COMM_WORLD,ISTAT,IERR)

CC    DO 240 I=2,MM1
      DO 240 I=ISTART2,IEND1
      DO 240 J=2,NN1
      DO 240 K=1,KK
           D00(K,J,I)=(D7(J,I)*U1(K,J,I)-D7(J,I-1)*U1(K,J,I-1))*HMMX(J,I)
     1               +(D8(J,I)*V1(K,J,I)-D8(J-1,I)*V1(K,J-1,I))*HMMY(J,I)
 240 CONTINUE
```

# 資料不切割的平行程式T5CP(13)

```
CC   DO 260 I=2,MM1
     DO 260 I=ISTART2,IEND1
     DO 260 KA=1,KK
     DO 260 J=2,NN1
     DO 260 K=1,KM
          F1(K,J,I)=F1(K,J,I)-VECINV(K,KA)*D00(KA,J,I)
 260 CONTINUE
```

# 資料不切割的平行程式T5CP(14)

```
      SUMF1=0.D0
      SUMF2=0.D0
CC    DO 270 I=2,MM1
      DO 270 I=ISTART2,IEND1
      DO 270 J=2,NN1
      DO 270 K=1,KM
           F2(K,J,I)=-AM7(K)*PS1(J,I)
           SUMF1=SUMF1+F1(K,J,I)
           SUMF2=SUMF2+F2(K,J,I)
 270 CONTINUE
      RETURN
      END
```

# 資料不切割的平行程式T5CP(15)

```
      SUBROUTINE OUTPUT
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE 'mpif.h'
      INCLUDE 'varcp'

      IROOT=0
      CALL MPI_REDUCE(SUMF1,GSUMF1,1,MPI_REAL8,
     &          MPI_SUM,IROOT, MPI_COMM_WORLD,IERR)
      CALL MPI_REDUCE(SUMF2,GSUMF2,1,MPI_REAL8,
     &          MPI_SUM,IROOT, MPI_COMM_WORLD,IERR)
      SUMF1=GSUMF1
      SUMF2=GSUMF2
```

# 資料不切割的平行程式T5CP(16)

```
 ITAG=40
 IF(MYID.NE.0) THEN
    KNT=KM*NN*GCOUNT(MYID)
    CALL MPI_SEND (F2(1,1,ISTART), KNT, MPI_REAL8, 0,
1                          ITAG, MPI_COMM_WORLD, IERR)
  ELSE
     DO I=1,NPROC-1
        IST1=GSTART(I)
        KNT1=KM*NN*GCOUNT(I)
        CALL MPI_RECV (F2(1,1,IST1), KNT1, MPI_REAL8,
1                    I, ITAG, MPI_COMM_WORLD, ISTAT, IERR)
      ENDDO
  ENDIF
```

# 資料不切割的平行程式T5CP(17)

```
301  FORMAT(8F10.3)
       IF(MYID.EQ.0) THEN
          PRINT *,'SUMF1,SUMF2=', SUMF1,SUMF2
          PRINT *,' F2(2,2,I),I=1,160,5'
          PRINT 301,(F2(2,2,I),I=1,160,5)
       ENDIF
       RETURN
       END
```

# 平行程式T5CP的輸出結果

```
ATTENTION: 0031-408  4 tasks allocated by LoadLeveler, continuing...
 SUMF1,SUMF2= 26172.4605364287563 -2268.89180334309913
 F2(2,2,I),I=1,160,5
   .000    -.333    -.295    -.281    -.274    -.269    -.266    -.264
  -.262    -.261    -.260    -.259    -.258    -.258    -.257    -.257
  -.256    -.256    -.255    -.255    -.255    -.255    -.255    -.254
  -.254    -.254    -.254    -.254    -.254    -.253    -.253    -.253
MYID, CLOCK TIME= 1 0.42296275496482 8491E-01
MYID, CLOCK TIME= 2 0.44258190318942 0700E-01
MYID, CLOCK TIME= 3 0.46218616887927 0554E-01
MYID, CLOCK TIME= 0 0.57047871872782 7072E-01
```

- 執行循序程式T5SEQ需時0.07秒，在四個CPU上執行平行程式T5CP需時0.057秒，平行效率 (parallel speed up) 為0.07/0.057= 1.23倍。

# 多維陣列的平行程式

- 循序程式T5SEQ，該程式是由一個副程式改寫而成，陣列的index順序也由原來的 (I,J,K) 改寫為 (K,J,I)
- 資料不切割的平行程式T5CP
- **資料切割的平行程式 T5DCP**
- 與二維切割有關的MPI指令
- 末二維切割的平行程式T5_2D

# 末維資料切割的平行程式T5DCP(1)

- MM能被NP整除時切割後的陣列長度M為MM/NP，MM不能被NP整除時切割後的陣列長度M必須為MM/NP+1。使用PARAMETER陳述設定如下：

```
  PARAMETER  (KK=20, NN=120, MM=160,
1                    KM=3, MM1=MM-1, NN1=NN-1)
  PARAMETER  (NP=8, M=MM/NP)
```

# 末維資料切割的平行程式T5DCP(2)

- 則需要做資料交換的dimension設為 (0:M+1)，不需要做資料交換的dimension設為 (M)。

```
DIMENSION      TT(KM,NN,MM)
DIMENSION      U1(KK,NN,0:M+1), V1(KK,NN,M),
                      PS1(NN,0:M+1)
COMMON/BLK4/F1(KM,NN,M), F2(KM,NN,M),
1                        HXU(NN,M), HXV(NN,M),
2                        HMMX(NN,M), HMMY(NN,M)
COMMON/BLK5/VECINV(KK,KK), AM7(KK)
DIMENSION      D7(NN,0:M+1), D8(NN,M), D00(KK,NN,M)
```

# 三維陣列末維切割的 MPI_GATHERV(1)

- 陣列F1、F2在各個CPU上的資料長度GCOUNT和在陣列TT上的相對位置GDISP為：

```
CALL STARTEND (NPROC, 1, MM, GSTART, GEND,GCOUNT)
 DO I=0,NPROC-1
       GCOUNT(I)=KM*NN*GCOUNT(I)
       GDISP(I)=KM*NN*(GSTART(I)-1)
 ENDDO
 ISTARTG=GSTART(MYID)
 IENDG=GEND(MYID)
 MYCOUNT=IENDG-ISTARTG+1
```

# 三維陣列末維切割的 MPI_GATHERV(2)

- 三維陣列資料上的 MPI_GATHERV 寫法如下：

```
IROOT=0
KOUNT= GCOUNT(MYID)
CALL MPI_GATHERV (F1, KOUNT, MPI_REAL8,
1                 TT, GCOUNT, GDISP, MPI_REAL8,
2                 IROOT, MPI_COMM_WORLD, IERR)
```

# 多維陣列末維切割的平行程式(1)

**檔案 vardcp 之內容**

```
      PARAMETER  (KK=20,NN=120,MM=160,KM=3,MM1=MM-1,NN1=NN-1)
      PARAMETER  (NP=4,M=MM/NP)

      COMMON     U1(KK,NN,0:M+1),V1(KK,NN,M),PS1(NN,0:M+1),
     1           F1(KM,NN,M),F2(KM,NN,M),
     2           HXU(NN,M),HXV(NN,M), HMMX(NN,M),HMMY(NN,M),
     3           VECINV(KK,KK),AM7(KK), SUMF1,SUMF2
C-----------------------------------------------------
      INTEGER    L_NBR,R_NBR,ISTAT(MPI_STATUS_SIZE),
     1           GSTART,GEND,GCOUNT,GDISP
      COMMON/MPIVAR/MYID,NPROC,ISTAT,L_NBR,R_NBR,ISTARTG,IENDG,
     1                  ISTART,ISTART2,ISTART3, IEND,IEND1,
     2                  ISTARTM1, IENDM1,IENDP1, ICOUNT,
     3                  GSTART(0:31),GEND(0:31),GCOUNT(0:31),GDISP(0:31)
C-----------------------------------------------------
```

# 多維陣列末維切割的平行程式(2)

```
PROGRAM  T5DCP
IMPLICIT REAL*8 (A-H,O-Z)
INCLUDE 'mpif.h'
INCLUDE 'vardcp'

CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NPROC,IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYID, IERR)
CALL MPI_BARRIER(MPI_COMM_WORLD,IERR)
CLOCK=MPI_WTIME()
LASTP=NPROC-1
CALL STARTEND(NPROC,1,MM,GSTART,GEND,GCOUNT)
```

# 多維陣列末維切割的平行程式(3)

```
      DO I=0,LASTP
         GCOUNT(I)=KM*NN*GCOUNT(I)
         GDISP(I) =KM*NN*(GSTART(I)-1)
      ENDDO
      ISTARTG=GSTART(MYID)
      IENDG=GEND(MYID)
      ICOUNT=IENDG-ISTARTG+1
      PRINT 101, MYID,NPROC,ISTARTG,IENDG
101 FORMAT( ' MYID,NPROC,ISTARTG,IENDG=', 4I6)
```

# 多維陣列末維切割的平行程式(4)

```
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C       for DO I=x,MM1 (MM-1)
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      IEND1=IEND
      IF( MYID.EQ.NPROC-1) IEND1=IEND1-1
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
C       for DO I=2,x
C - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      ISTART2=1
      IF( MYID.EQ.0) ISTART2=2
      ISTARTM1=ISTART-1
      IENDP1=IEND+1
      L_NBR=MYID-1
      R_NBR=MYID+1
```

# 多維陣列末維切割的平行程式(5)

```
IF(MYID.EQ.0)         L_NBR=MPI_PROC_NULL
IF(MYID.EQ.LASTP) R_NBR=MPI_PROC_NULL
CALL DATAGEN
CALL COMPUTATION
CALL OUTPUT
CLOCK=MPI_WTIME() - CLOCK
PRINT *,' MYID, CLOCK TIME=', MYID,CLOCK
CALL MPI_FINALIZE(IERR)
STOP
END
```

# 多維陣列末維切割的平行程式(6)

```
      SUBROUTINE   DATAGEN
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE 'mpif.h'
      INCLUDE 'vardcp'

CC    DO 10 I=1,MM1
      DO 10 I=1,IEND1
      II=I+ISTARTG-1
      DO 10 J=1,NN
      DO 10 K=1,KK
            U1(K,J,I)=1.D0/DFLOAT(II)+1.D0/DFLOAT(J)+1.D0/DFLOAT(K)
 10   CONTINUE
```

# 多維陣列末維切割的平行程式(7)

```
CC    DO 20 I=1,MM
      DO 20 I=1,IEND
      II=I+ISTARTG-1
      DO 20 J=1,NN1
      DO 20 K=1,KK
            V1(K,J,I)=2.D0/DFLOAT(II)+1.D0/DFLOAT(J)+1.D0/DFLOAT(K)
  20  CONTINUE
```

# 多維陣列末維切割的平行程式(8)

```
   SUBROUTINE COMPUTATION
   MPLICIT REAL*8 (A-H,O-Z)
   INCLUDE   'mpif.h'
   INCLUDE   'vardcp'
   DIMENSION D7(NN,0:M+1),D8(NN,M),D00(KK,NN,M)

  CALL MPI_BARRIER(MPI_COMM_WORLD,IERR)
  CLOCK=MPI_WTIME()
  NNKK=NN*KK
  CALL MPI_SENDRECV(
1         U1(1,1,IEND ),       NNKK,MPI_REAL8, R_NBR,10,
2         U1(1,1,ISTARTM1),NNKK,MPI_REAL8, L_NBR,10,
3         MPI_COMM_WORLD,ISTAT,IERR)
```

# 多維陣列末維切割的平行程式(9)

```
      CALL MPI_SENDRECV(
     1          PS1(1,ISTART),NN,MPI_REAL8, L_NBR,30,
     2          PS1(1,IENDP1),NN,MPI_REAL8, R_NBR,30,
     3          MPI_COMM_WORLD,ISTAT,IERR)
CC    DO 210 I=1,MM
      DO 210 I=ISTART,IEND
      DO 210 J=1,NN
      DO 210 K=1,KM
           F1(K,J,I)=0.0D0
           F2(K,J,I)=0.0D0
 210 CONTINUE
```

# 多維陣列末維切割的平行程式(10)

```
CC    DO 220 I=1,MM1
      DO 220 I=ISTART,IEND1
      DO 220 J=2,NN1
          D7(J,I)=(PS1(J,I+1)+PS1(J,I))*0.5D0*HXU(J,I)
 220 CONTINUE
CC    DO 230 I=2,MM1
      DO 230 I=ISTART2,IEND1
      DO 230 J=1,NN1
          D8(J,I)=(PS1(J+1,I)+PS1(J,I))*0.5D0*HXV(J,I)
 230 CONTINUE
```

# 多維陣列末維切割的平行程式(11)

```
      ITAG=50
      CALL MPI_SENDRECV(
     1          D7(1,IEND),       NN,MPI_REAL8,  R_NBR, ITAG,
     2          D7(1,ISTARTM1),NN,MPI_REAL8,  L_NBR, ITAG,
     3          MPI_COMM_WORLD,ISTAT,IERR)
CC    DO 240 I=2,MM1
      DO 240 I=ISTART2,IEND1
      DO 240 J=2,NN1
      DO 240 K=1,KK
          D00(K,J,I)=(D7(J,I)*U1(K,J,I)-D7(J,I-1)*U1(K,J,I-1))*HMMX(J,I)
     1               +(D8(J,I)*V1(K,J,I)-D8(J-1,I)*V1(K,J-1,I))*HMMY(J,I)
 240 CONTINUE
```

# 多維陣列末維切割的平行程式(12)

```
      DO 260 I=2,MM1
      DO 260 I=ISTART2,IEND1
      DO 260 KA=1,KK
      DO 260 J=2,NN1
      DO 260 K=1,KM
           F1(K,J,I)=F1(K,J,I)-VECINV(K,KA)*D00(KA,J,I)
260 CONTINUE
```

# 多維陣列末維切割的平行程式(13)

```
      SUMF1=0.D0
      SUMF2=0.D0
CC    DO 270 I=2,MM1
      DO 270 I=ISTART2,IEND1
      DO 270 J=2,NN1
      DO 270 K=1,KM
           F2(K,J,I)=-AM7(K)*PS1(J,I)
           SUMF1=SUMF1+F1(K,J,I)
           SUMF2=SUMF2+F2(K,J,I)
 270 CONTINUE
      RETURN
      END
```

# 多維陣列末維切割的平行程式(14)

```
      SUBROUTINE OUTPUT
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE 'mpif.h'
      INCLUDE 'vardcp'
      DIMENSION   TT(KM,NN,MM)
      CALL MPI_BARRIER(MPI_COMM_WORLD,IERR)
      IROOT=0
      KOUNT=GCOUNT(MYID)
      CALL MPI_GATHERV (F2, KOUNT, MPI_REAL8,
     1                  TT, GCOUNT, GDISP, MPI_REAL8,
     2                  IROOT, MPI_COMM_WORLD, IERR)
```

# 多維陣列末維切割的平行程式(15)

```
      CALL MPI_REDUCE(SUMF1, GSUMF1, 1, MPI_REAL8,
     &        MPI_SUM, IROOT, MPI_COMM_WORLD, IERR)
      CALL MPI_REDUCE(SUMF2, GSUMF2, 1, MPI_REAL8,
     &        MPI_SUM, IROOT, MPI_COMM_WORLD, IERR)
301   FORMAT(8F10.3)
      IF(MYID.EQ.0) THEN
        PRINT *,'SUMF1,SUMF2=', GSUMF1,GSUMF2
        PRINT *,' F2(2,2,I),I=1,160,5'
        PRINT 301,(TT(2,2,I),I=1,160,5)
      ENDIF
      RETURN
      END
```

# 平行程式T5DCP的測試結果(1)

```
ATTENTION: 0031-408  4 tasks allocated by LoadLeveler,
   continuing...
MYID, CLOCK TIME= 1 0.31331064179539680 5E-01
MYID, CLOCK TIME= 3 0.32239789143204689 0E-01
SUMF1,SUMF2= 26172.4605364287 -2268.89180334309
F2(2,2,I),I=1,160,5
MYID, CLOCK TIME= 2 0.32200107350945472 7E-01
  .000    -.333    -.295    -.281    -.274    -.269    -.266    -.264
 -.262    -.261    -.260    -.259    -.258    -.258    -.257    -.257
 -.256    -.256    -.255    -.255    -.255    -.255    -.255    -.254
 -.254    -.254    -.254    -.254    -.254    -.253    -.253    -.253
MYID, CLOCK TIME= 0 0.32503936439752578 7E-01
```

# 平行程式T5DCP的測試結果(2)

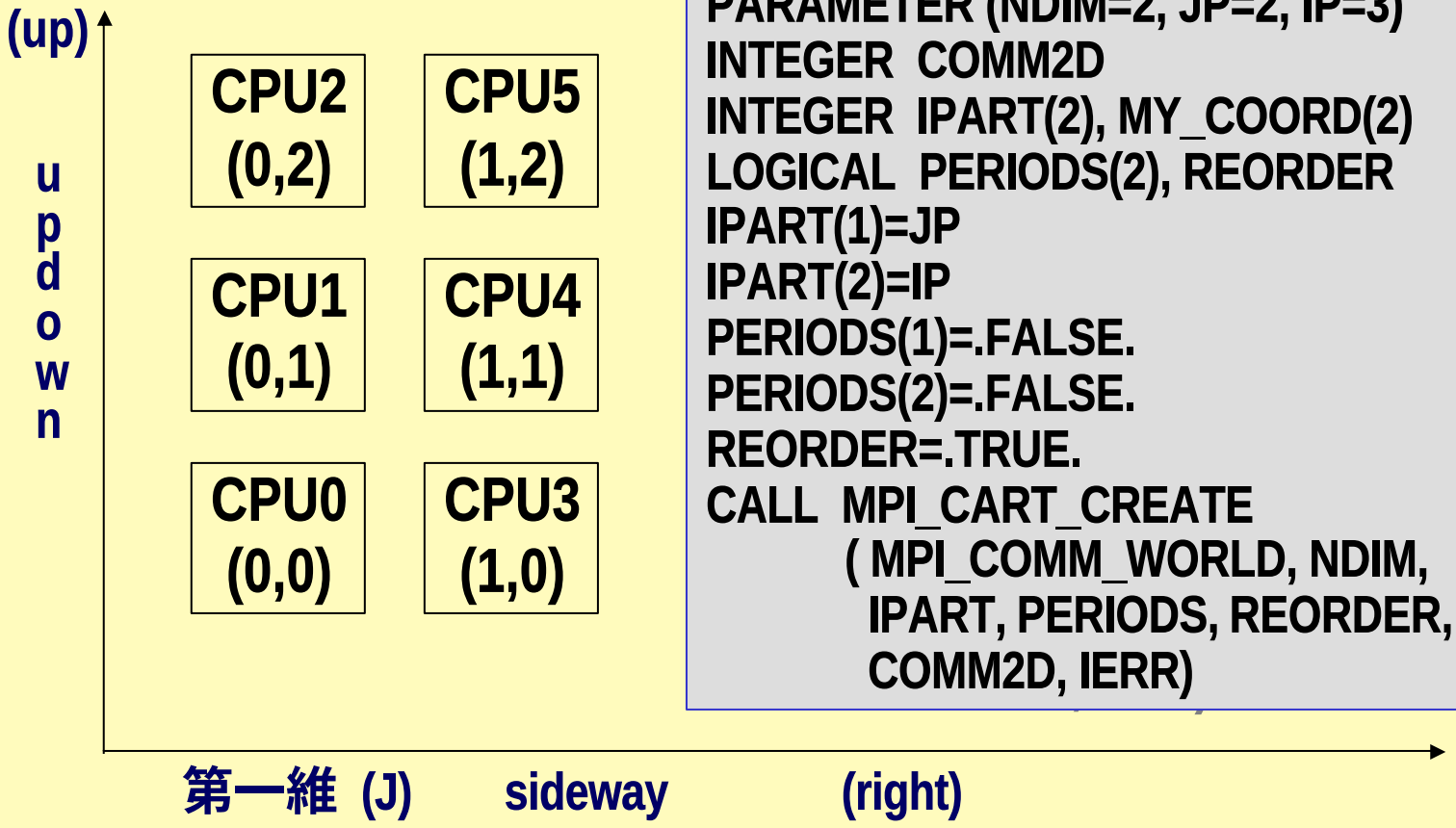
- 執行循序程式T5SEQ需時0.07秒，在四個CPU上執行平行程式T5DCP需時0.0325秒，平行效率 (parallel speed up) 為0.07/0.0325 = 2.15倍。
- 在四個CPU上平行計算時，採用陣列末維資料切割方法需時0.0325，而採用資料不切割方法是需時0.057秒，前者的平行效率較佳。

# 多維陣列的平行程式

- 循序程式T5SEQ，該程式是由一個副程式改寫而成，陣列的index順序也由原來的 (I,J,K) 改寫為 (K,J,I)
- 資料不切割的平行程式T5CP
- 資料切割的平行程式 T5DCP
- **與二維切割有關的MPI指令**
- 末二維切割的平行程式T5_2D

# 定義二維切割的 MPI 指令(1)

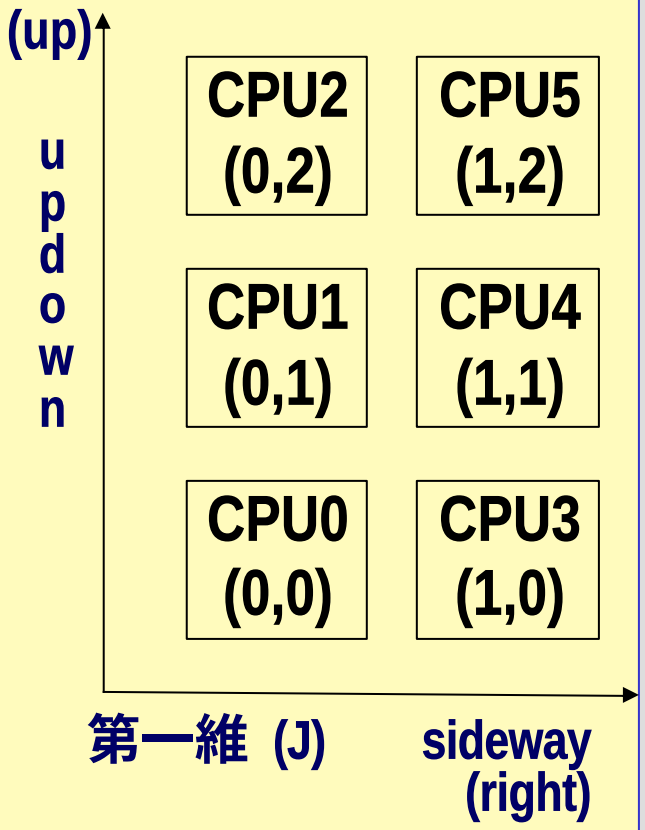
## 第二維 (I)　垂直座標圖示法則 (Cartesian Topology)




```
PARAMETER (NDIM=2, JP=2, IP=3)
INTEGER  COMM2D
INTEGER  IPART(2), MY_COORD(2)
LOGICAL  PERIODS(2), REORDER
IPART(1)=JP
IPART(2)=IP
PERIODS(1)=.FALSE.
PERIODS(2)=.FALSE.
REORDER=.TRUE.
CALL  MPI_CART_CREATE
      ( MPI_COMM_WORLD, NDIM,
        IPART, PERIODS, REORDER,
        COMM2D, IERR)
```

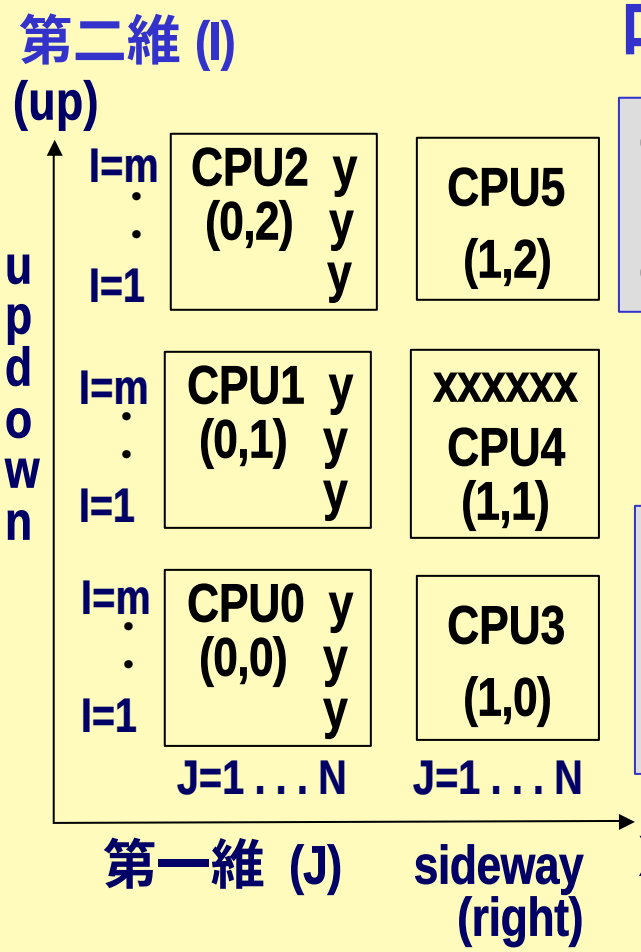
# 定義二維切割的 MPI 指令(2)

## 垂直座標圖示法則 (Cartesian Topology)




```
PARAMETER (NDIM=2, JP=2, IP=3)
INTEGER  COMM2D, SIDEWAY, UPDOWN
INTEGER  UP, RIGHT, MY_COORD(2)
INTEGER  L_NBR, R_NBR, B_NBR, T_NBR
CALL MPI_COMM_RANK(COMM2D, MYID, IERR)
CALL MPI_CART_COORDS (COMM2D,
                MYID, NDIM, MY_COORD, IERR)
SIDEWAY=0
UPDOWN=1
RIGHT=1
UP=1
CALL MPI_CART_SHIFT(COMM2D,
       SIDEWAY, RIGHT, L_NBR, R_NBR, IERR)
CALL MPI_CART_SHIFT(COMM2D,
        UPDOWN, UP, B_NBR, T_NBR, IERR)
```

# 定義固定間隔資料的MPI指令




## 叫用格式

```
CALL MPI_TYPE_VECTOR( COUNT,BLKLEN,
                      STRIDE, OLDTYPR, NEWTYPE,IERR)
CALL MPI_TYPE_COMMIT(NEWTYPE, IERR)
```

## yyyy串資料的設定

```
INTEGER  VECTOR_M
CALL MPI_TYPE_VECTOR( M, 1, N,
                      MPI_REAL8, VECTOR_M, IERR)
CALL MPI_TYPE_COMMIT( VECTOR_M, IERR)
```

## xxxxxx串資料為連續位址之資料

# 未二維切割的平行程式T5_2D(1)

```
      PARAMETER  (JP=2,IP=4,N=NN/JP,M=MM/IP,NP=IP*JP)
      PARAMETER  (LASTPJ=JP-1, LASTPI=IP-1)
      COMMON    U1(KK,0:N+1,0:M+1),V1(KK,0:N+1,0:M+1),PS1(0:N+1,0:M+1),
     1          F1(KM,0:N+1,0:M+1),F2(KM,0:N+1,0:M+1),
     2          HXU(0:N+1,0:M+1),HXV(0:N+1,0:M+1),
     3          HMMX(0:N+1,0:M+1),HMMY(0:N+1,0:M+1),
     4          VECINV(KK,KK),AM7(KK),SUMF1,SUMF2
C-----------------------------------------------------------------------
      INTEGER    ISTAT(MPI_STATUS_SIZE),COMM2D,
     &           L_NBR,R_NBR,T_NBR,B_NBR,MYID,MY_COORD(2)
      COMMON/MPIVAR/MYID,MYID_I,MYID_J,NPROC,COMM2D,MY_COORD,
     1           L_NBR,R_NBR,T_NBR,B_NBR,
     2           ISTART,ISTART2,IEND,IEND1,ISTARTM1,IENDP1,
     3           JSTART,JSTART2,JEND,JEND1,JSTARTM1,JENDP1,
     4           ISTARTG(0:31),IENDG(0:31),ICOUNTG(0:31),
     5           JSTARTG(0:31),JENDG(0:31),JCOUNTG(0:31)
C-----------------------------------------------------------------------
```

檔案 var2d 之內容

# 末二維切割的平行程式T5_2D(2)

```
      PROGRAM  T5_2D
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE 'mpif.h'
      INCLUDE 'var2d'

      CALL MPI_INIT(IERR)
      CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NPROC,IERR)
      CALL MPI_BARRIER(MPI_COMM_WORLD, IERR)
      CLOCK=MPI_WTIME()
      CALL NBR2D(COMM2D,MYID,MY_COORD,L_NBR,R_NBR,
     &                    B_NBR,T_NBR, JP,IP)
      MYID_J=MY_COORD(1)
      MYID_I=MY_COORD(2)
      CALL STARTEND(IP,1,MM,ISTARTG,IENDG,ICOUNTG)
      CALL STARTEND(JP,1,NN,JSTARTG,JENDG,JCOUNTG)
      ISTART=1
      IEND=M
      JSTART=1
      JEND=N
```

# 末二維切割的平行程式T5_2D(3)

```
C-----------------------------------------------------------------
C       for DO I=x,MM1 (MM-1)       for DO J=x,NN1 (NN-1)
C-----------------------------------------------------------------
      IEND1=IEND
      JEND1=JEND
      IF( MYID_I.EQ.IP-1) IEND1=IEND1-1
      IF( MYID_J.EQ.JP-1) JEND1=JEND1-1
C  - -  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
      ISTART2=ISTART
      JSTART2=JSTART
```

# 末二維切割的平行程式T5_2D(4)

```
IF( MY_COORD(2).EQ.0) ISTART2=2
IF( MY_COORD(1).EQ.0) JSTART2=2
ISTARTM1=ISTART-1
IENDP1=IEND+1
JSTARTM1=JSTART-1
JENDP1=JEND+1
CALL DATAGEN
CALL COMPUTATION
CALL OUTPUT
CLOCK=MPI_WTIME() – CLOCK
PRINT *,' MYID, CLOCK TIME=', MYID,CLOCK
CALL MPI_FINALIZE(IERR)
END
```

# 末二維切割的平行程式T5_2D(5)

```
      SUBROUTINE DATAGEN
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE 'mpif.h'
      INCLUDE 'var2d'
CC    DO 10 I=1,MM1
      DO 10 I=ISTART,IEND1
      II=I+ISTARTG(MYID_I)-1
CC    DO 10 J=1,NN
      DO 10 J=JSTART,JEND
      JJ=J+JSTARTG(MYID_J)-1
      DO 10 K=1,KK
            U1(K,J,I)=1.D0/DFLOAT(II)+1.D0/DFLOAT(JJ)+1.D0/DFLOAT(K)
 10   CONTINUE
```

# 末二維切割的平行程式T5_2D(6)

```
CC   DO 20 I=1,MM
      DO 20 I=1,IEND
       II=I+ISTARTG(MY_COORD(2))-1
CC   DO 20 J=1,NN1
      DO 20 J=1,JEND1
      JJ=J+JSTARTG(MY_COORD(1))-1
      DO 20 K=1,KK
          V1(K,J,I)=2.D0/DFLOAT(II)+1.D0/DFLOAT(JJ)+1.D0/DFLOAT(K)
 20   CONTINUE
```

# 末二維切割的平行程式T5_2D(7)

```
CC   DO 30 I=1,MM
      DO 30 I=1,IEND
      II=I+ISTARTG(MY_COORD(2))-1
CC  DO 30 J=1,NN
     DO 30 J=1,JEND
         JJ=J+JSTARTG(MY_COORD(1))-1
         PS1(J,I)=1.D0/DFLOAT(II)+1.D0/DFLOAT(JJ)
         HXU(J,I)=2.D0/DFLOAT(II)+1.D0/DFLOAT(JJ)
         HXV(J,I)=1.D0/DFLOAT(II)+2.D0/DFLOAT(JJ)
         HMMX(J,I)=2.D0/DFLOAT(II)+1.D0/DFLOAT(JJ)
         HMMY(J,I)=1.D0/DFLOAT(II)+2.D0/DFLOAT(JJ)
 30   CONTINUE
```

# 末二維切割的平行程式T5_2D(8)

```
DO 40 K=1,KK
      AM7(K)=1.D0/DFLOAT(K)
      DO 40 KA=1,KK
         VECINV(KA,K)=1.D0/DFLOAT(KA)+1.D0/DFLOAT(K)
 40  CONTINUE
     RETURN
     END
```

# 末二維切割的平行程式T5_2D(9)

```
SUBROUTINE COMPUTATION
IMPLICIT REAL*8 (A-H,O-Z)
INCLUDE 'mpif.h'
INCLUDE 'var2d'
DIMENSION   D7(0:N+1,0:M+1),D8(0:N+1,0:M+1),D00(KK,0:N+1,0:M+1)
N2=N+2
N2KK=N2*KK
CALL MPI_TYPE_VECTOR (M, KK, N2KK, MPI_REAL8, IVECT_3D, IERR)
CALL MPI_TYPE_COMMIT (IVECT_3D, IERR)
CALL MPI_TYPE_VECTOR (M, 1, N2, MPI_REAL8, IVECT_2D, IERR)
CALL MPI_TYPE_COMMIT (IVECT_2D, IERR)
CALL MPI_BARRIER (COMM2D, IERR)
```

# 末二維切割的平行程式T5_2D(10)

```
 CALL MPI_SENDRECV (U1(1,1,IEND ),       N2KK, MPI_REAL8,T_NBR,10,
1                                U1(1,1,ISTARTM1),N2KK, MPI_REAL8,B_NBR,10,
2                                COMM2D, ISTAT, IERR)
 CALL MPI_SENDRECV(V1(1,JEND,1 ),        1, IVECT_3D, R_NBR, 20,
1                                V1(1,JSTARTM1,1), 1, IVECT_3D, L_NBR, 20,
2                                 COMM2D, ISTAT, IERR)
 CALL MPI_SENDRECV (PS1(1,ISTART), N, MPI_REAL8, B_NBR, 30,
1                                 PS1(1,IENDP1), N, MPI_REAL8, T_NBR, 30,
                                  COMM2D, ISTAT, IERR)
```

# 末二維切割的平行程式T5_2D(11)

```
      ITAG=40
      CALL MPI_SENDRECV (PS1(JSTART,1), 1, IVECT_2D, L_NBR, ITAG,
     1                   PS1(JENDP1,1), 1, IVECT_2D, R_NBR, ITAG,
     2                   COMM2D, ISTAT, IERR)
CC    DO 210 I=1,MM
CC    DO 210 J=1,NN
      DO 210 I=ISTART,IEND
      DO 210 J=JSTART,JEND
      DO 210 K=1,KM
         F1(K,J,I)=0.0D0
         F2(K,J,I)=0.0D0
210    CONTINUE
CC    DO 220 I=1,MM1
CC    DO 220 J=2,NN1
      DO 220 I=ISTART,IEND1
      DO 220 J=JSTART2,JEND1
         D7(J,I)=(PS1(J,I+1)+PS1(J,I))*0.5D0*HXU(J,I)
220   CONTINUE
```

# 末二維切割的平行程式T5_2D(12)

```
CC   DO 230 I=2,MM1
CC   DO 230 J=1,NN1
       DO 230 I=ISTART2,IEND1
       DO 230 J=JSTART,JEND1
          D8(J,I)=(PS1(J+1,I)+PS1(J,I))*0.5D0*HXV(J,I)
230  CONTINUE

      CALL MPI_SENDRECV (D7(1,IEND),        N, MPI_REAL8, T_NBR, 50,
     1                   D7(1,ISTARTM1), N, MPI_REAL8, B_NBR, 50,
     2                   COMM2D, ISTAT,  IERR)
      CALL MPI_SENDRECV (D8(JEND,1),        1,  IVECT_2D, R_NBR, 60,
     1                   D8(JSTARTM1,1), 1, IVECT_2D, L_NBR, 60,
     2                   COMM2D, ISTAT,  IERR)
```

# 末二維切割的平行程式T5_2D(13)

```
CC  DO 240 I=2,MM1
CC  DO 240 J=2,NN1
      DO 240 I=ISTART2,IEND1
      DO 240 J=JSTART2,JEND1
      DO 240 K=1,KK
         D00(K,J,I)=(D7(J,I)*U1(K,J,I)-D7(J,I-1)*U1(K,J,I-1))*HMMX(J,I)
     1              +(D8(J,I)*V1(K,J,I)-D8(J-1,I)*V1(K,J-1,I))*HMMY(J,I)
240 CONTINUE
```

# 末二維切割的平行程式T5_2D(14)

```
CC  DO 260 I=2,MM1
      DO 260 I=ISTART2,IEND1
      DO 260 KA=1,KK
CC  DO 260 J=2,NN1
      DO 260 J=JSTART2,JEND1
      DO 260 K=1,KM
          F1(K,J,I)=F1(K,J,I)-VECINV(K,KA)*D00(KA,J,I)
260 CONTINUE
CC  DO 270 I=2,MM1
      DO 270 I=ISTART2,IEND1
CC  DO 270 J=2,NN1
      DO 270 J=JSTART2,JEND1
      DO 270 K=1,KM
         F2(K,J,I)=-AM7(K)*PS1(J,I)
         SUMF1=SUMF1+F1(K,J,I)
         SUMF2=SUMF2+F2(K,J,I)
270 CONTINUE
      RETURN
      END
```

# 末二維切割的平行程式T5_2D(15)

```
      SUBROUTINE OUTPUT
C-----------------------------------------------------------------
C     Output data for validation
C-----------------------------------------------------------------
      IMPLICIT REAL*8 (A-H,O-Z)
      INCLUDE 'mpif.h'
      INCLUDE 'var2d ‘
      DIMENSION   TT(KM,NN,MM)

     CALL MPI_BARRIER (COMM2D, IERR)
     IROOT=0
     CALL MPI_REDUCE (SUMF1, GSUMF1, 1, MPI_REAL8, MPI_SUM,
    1                          IROOT, COMM2D, IERR)
     CALL MPI_REDUCE (SUMF2, GSUMF2, 1, MPI_REAL8, MPI_SUM,
    1                          IROOT, COMM2D, IERR)
```

# 末二維切割的平行程式T5_2D(16)

```
 KNT=KM*(N+2)*(M+2)
 IF (MYID.NE.0) THEN
    CALL MPI_SEND (F2, KNT, MPI_REAL8, IROOT, 70, COMM2D, IERR)
 ELSE
    CALL COPY1(MYID, F2, TT,ISTARTG,JSTARTG)
    DO ISRC=1,NPROC-1
        CALL MPI_RECV (F2, KNT, MPI_REAL8, ISRC, 70,COMM2D,
1                              ISTATUS, IERR)
        CALL COPY1 (ISRC, F2, TT,ISTARTG,JSTARTG)
     ENDDO
  ENDIF
```

# 末二維切割的平行程式T5_2D(17)

```
321   FORMAT(8F10.3)
      IF(MYID.EQ.0) THEN
          PRINT *,'SUMF1,SUMF2=', GSUMF1,GSUMF2
          PRINT *,' F2(2,2,I),I=1, 160,5'
          PRINT 321,(TT(2,2,I),I=1, 160,5)
      ENDIF
      RETURN
      END
```

# 末二維切割的平行程式T5_2D(18)

```
      SUBROUTINE NBR2D(COMM2D,MYID,MY_COORD,L_NBR,
     1                           R_NBR,B_NBR, T_NBR, JP, IP)
      INCLUDE   'mpif.h'
      PARAMETER (NDIM=2)
      INTEGER   COMM2D, MYID, MY_COORD(2),
     1           L_NBR, R_NBR,  B_NBR, T_NBR, JP, IP
      INTEGER   IPART(2), SIDEWAYS, UPDOWN, RIGHT, UP
      LOGICAL   PERIODS(2), REORDER
      IPART(1)=JP
      IPART(2)=IP
      PERIODS(1)=.FALSE.
      PERIODS(2)=.FALSE.
      REORDER=.TRUE.
```

# 末二維切割的平行程式T5_2D(19)

```
      SIDEWAYS=0
       UPDOWN=1
       RIGHT=1
       UP=1
       CALL MPI_CART_CREATE ( MPI_COMM_WORLD, NDIM, IPART,
      1                       PERIODS, REORDER, COMM2D, IERR)
       CALL MPI_COMM_RANK ( COMM2D, MYID, IERR)
       CALL MPI_CART_COORDS( COMM2D, MYID, NDIM, MY_COORD,
      1                        IERR)
       CALL MPI_CART_SHIFT( COMM2D,SIDEWAYS,RIGHT,
      1                     L_NBR,R_NBR, IERR)
       CALL MPI_CART_SHIFT( COMM2D, UPDOWN, UP,
      1                     B_NBR, T_NBR, IERR)
       END
```

# 末二維切割的平行程式T5_2D(20)

```
SUBROUTINE COPY1(ISRC, FF, TT, ISTARTG, JSTARTG)
PARAMETER  (KK=20,NN=120,MM=160, KM=3,MM1=MM-1,NN1=NN-1)
PARAMETER  (JP=2, IP=4, N=NN/JP, M=MM/IP, NP=IP*JP)
REAL*8          FF(KM,0:N+1,0:M+1),  TT(KM,NN,MM)
INTEGER         ISTARTG(0:NP),  JSTARTG(0:NP)
IF(ISRC.LT.IP) THEN
    JJ=0
     II=ISRC
 ELSE
     JJ=ISRC/IP
      II=ISRC-JJ*IP
 ENDIF
```

# 末二維切割的平行程式T5_2D(21)

```
DO I=1,M
   IG=ISTARTG(II)+I-1
   DO J=1,N
      JG=JSTARTG(JJ)+J-1
      DO K=1,KM
         TT(K,JG,IG)=FF(K,J,I)
      ENDDO
   ENDDO
ENDDO
END
```

# 平行程式T5_2D的測試結果

```
ATTENTION: 0031-408  8 nodes allocated by LoadLeveler, continuing...
   MYID, CLOCK TIME= 4 0.6585404276847839 36E-01
   MYID, CLOCK TIME= 5 0.6655255518853664 40E-01
   MYID, CLOCK TIME= 6 0.6760265864431858 06E-01
   MYID, CLOCK TIME= 1 0.6095638312399387 36E-01
   MYID, CLOCK TIME= 2 0.6263478472828865 05E-01
   MYID, CLOCK TIME= 7 0.6847446784377098 08E-01
   MYID, CLOCK TIME= 3 0.6446106545627117 16E-01

 SUMF1,SUMF2= 26172.4605364287345  -2268.89180334310413
 F2(2,2,I),I=1,160,5
  .000E+00 -.333E+00 -.295E+00 -.281E+00 -.274E+00 -.269E+00 -.266E+00 -.264E+00
 -.262E+00 -.261E+00 -.260E+00 -.259E+00 -.258E+00 -.258E+00 -.257E+00 -.257E+00
 -.256E+00 -.256E+00 -.255E+00 -.255E+00 -.255E+00 -.255E+00 -.255E+00 -.254E+00
 -.254E+00 -.254E+00 -.254E+00 -.254E+00 -.254E+00 -.253E+00 -.253E+00 -.253E+00
  MY_CID, CLOCK TIME= 0  0.1250972000416368 25
  MYID, CLOCK TIME= 0 0.6888415105640888 21E-01
```

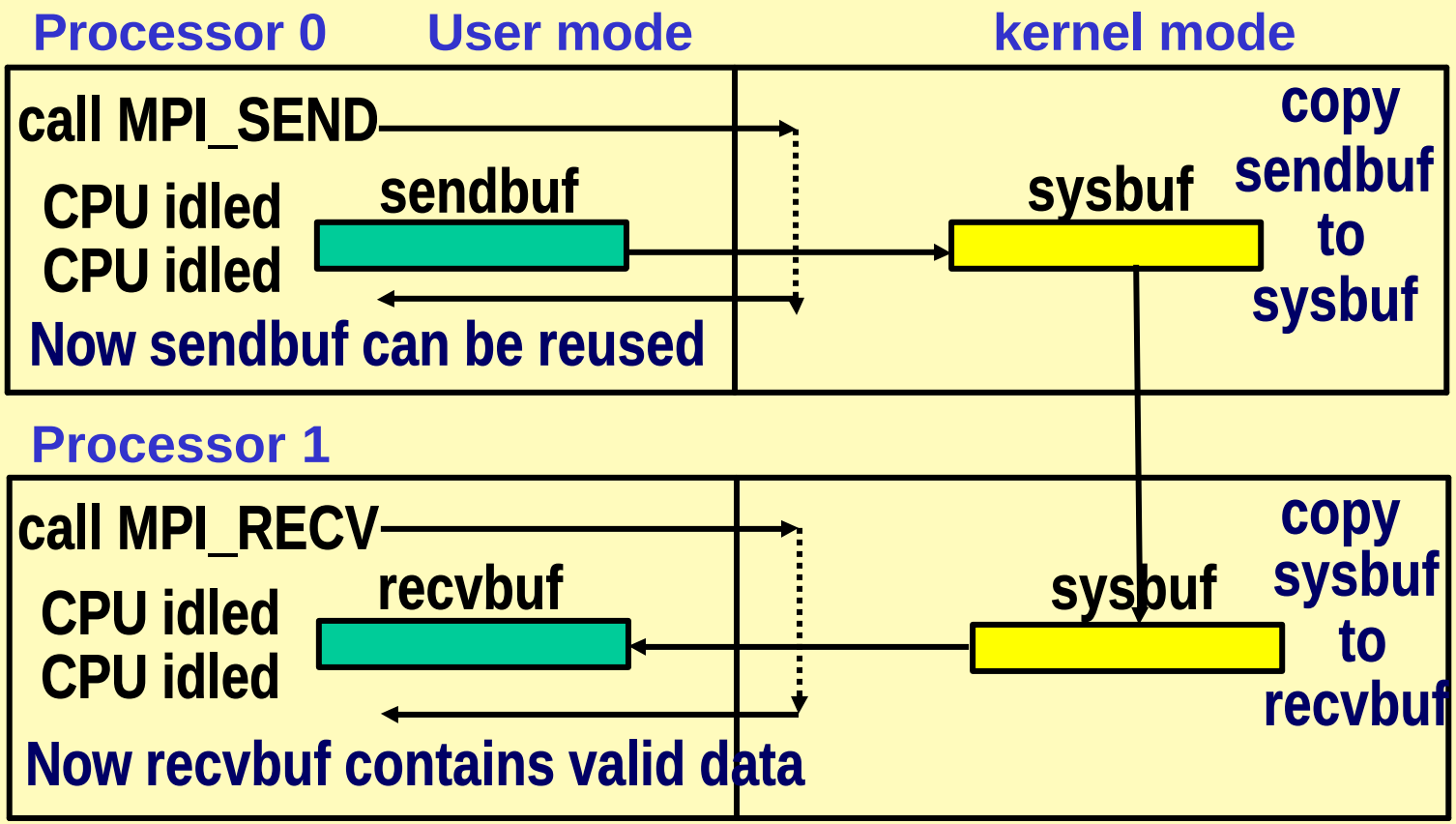
# 課 程 大 綱

1. 前言
2. 無邊界資料交換的平行程式
3. 需要邊界資料交換的平行程式
4. 格點數不能整除的平行程式
5. 多維陣列的平行程式
6. **MPI平行程式的效率提昇**

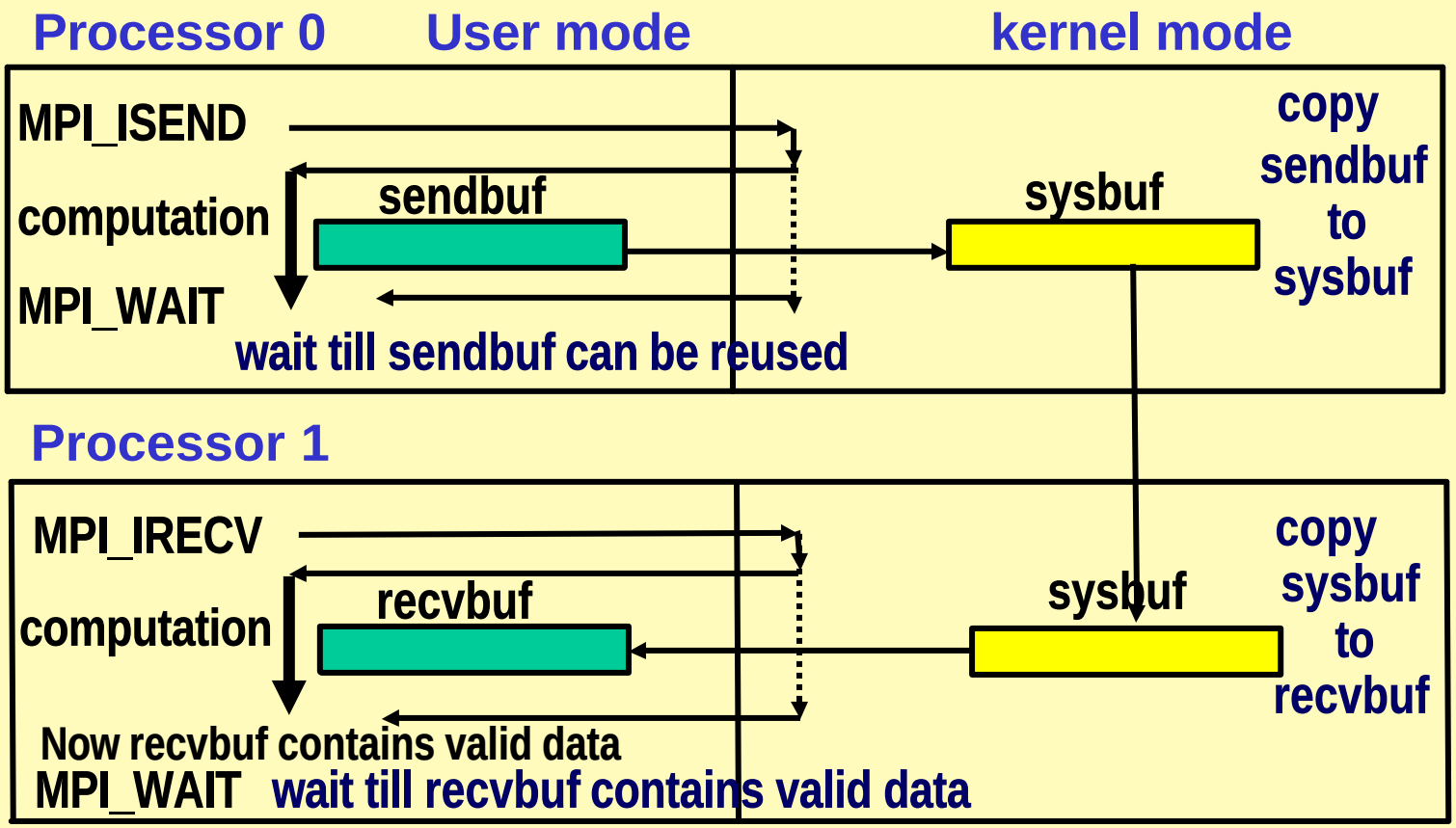
# 平行程式的效率提昇

- Nonblocking 資料傳送
- 傳送資料的合併 (含MPI_STRUCT)
- 以邊界資料計算取代邊界資料交換
- 事先切割輸入資料與事後收集切割過的輸出資料

# Blocking SEND/RECV示意圖



# Nonblocking SEND/RECV示意圖



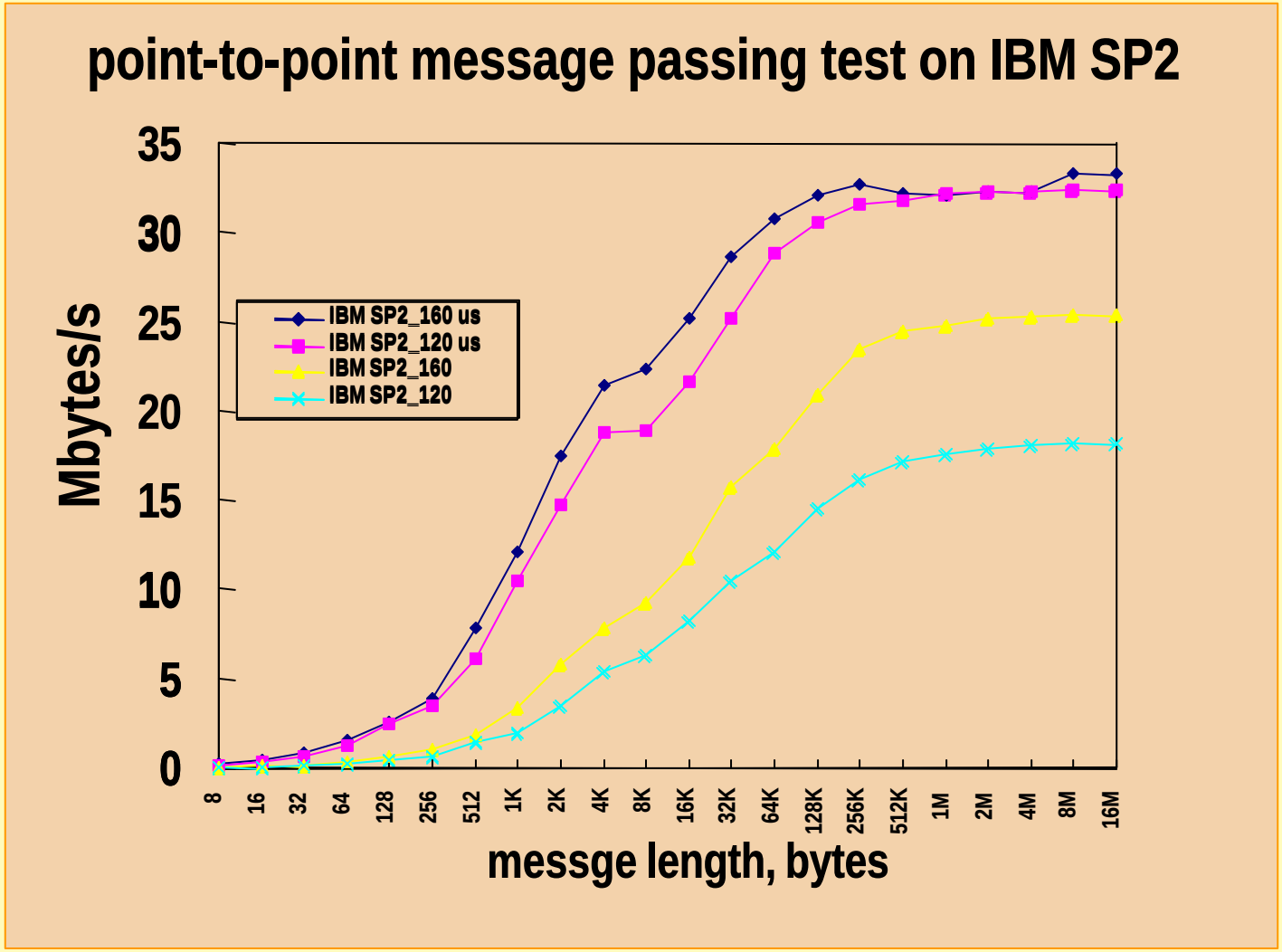
# Nonblocking SEND/RECV

```
CALL MPI_ISEND(DATA1, ICOUNT, MPI_REAL, IDEST, ITAG,
                      MPI_COMM_WORLD, IREQUEST1, IERR)
CALL MPI_IRECV(DATA2, ICOUNT, MPI_REAL, ISRC, ITAG,
                      MPI_COMM_WORLD, IREQUEST2, IERR)
. . . . . .
. . . . . .(any computation not using DATA2)
. . . . . .
CALL MPI_WAIT (IREQUEST1, ISTATUS, IERR)
CALL MPI_WAIT (IREQUEST2, ISTATUS, IERR)
```

# 平行程式的效率提昇

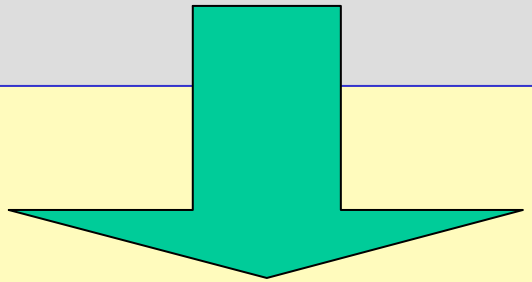
- Nonblocking 資料傳送
- 傳送資料的合併 (含MPI_STRUCT)
- 以邊界資料計算取代邊界資料交換
- 事先切割輸入資料與事後收集切割過的輸出資料

# 資料傳送的合併(1)

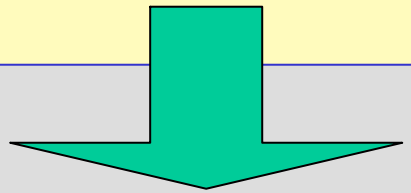

point-to-point message passing test on IBM SP2
Mbytes/s
35
30
25
20
15
10
5
0
IBM SP2_160 us
IBM SP2_120 us
IBM SP2_160
IBM SP2_120
8
16
32
64
128
256
512
1K
2K
4K
8K
16K
32K
64K
128K
256K
512K
1M
2M
4M
8M
16M
messge length, bytes

# 資料傳送的合併(2)

```
 CALL MPI_SENDRECV (
1          PS1(1,IEND),         N, MPI_REAL8, R_NBR,  110,
2          PS1(1,ISTARTM1), N, MPI_REAL8, L_NBR,  110,
3          MPI_COMM_WORLD, ISTAT, IERR)

 CALL MPI_SENDRECV (

 1         PS2(1,IEND),         N, MPI_REAL8,  R_NBR,  120,

 2         PS2(1,ISTARTM1), N, MPI_REAL8,  L_NBR,  120,

 3         MPI_COMM_WORLD, ISTAT, IERR)
```



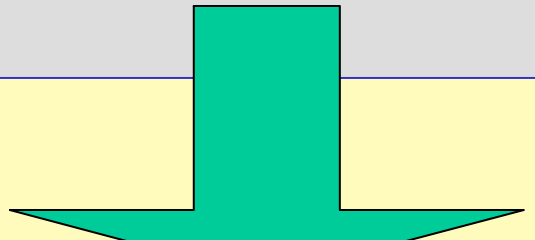
# 資料傳送的合併(3)



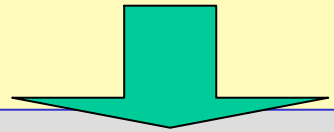
```
 REAL*8  BUF1(N,2), BUF2(N,2)
 DO J=1,N
    BUF1(J,1)=PS1(J,IEND)
    BUF1(J,2)=PS2(J,IEND)
 ENDDO
 KOUNT=N*2
 CALL MPI_SENDRECV (BUF1, KOUNT, MPI_REAL8,  R_NBR,  110,
1                           BUF2, KOUNT, MPI_REAL8,  L_NBR,  110,
2                           MPI_COMM_WORLD, STAT, IERR)
 IF (MYID.NE.0) THEN
   DO J=1,N
      PS1(J,ISTARTM1)=BUF2(J,1)
      PS2(J,ISTARTM1)=BUF2(J,2)
   ENDDO
ENDIF
```

# 資料傳送的合併(4)

```
 CALL MPI_SENDRECV (
1            PS1(1,IEND),         N, MPI_REAL8,  R_NBR,  110,
2            PS1(1,ISTARTM1), N, MPI_REAL8,  L_NBR,  110,
3            MPI_COMM_WORLD, ISTAT, IERR)
 CALL MPI_SENDRECV (
1           PS2(1,IEND),         N, MPI_REAL8,  R_NBR,  120,
2           PS2(1,ISTARTM1), N, MPI_REAL8,  L_NBR,  120,
3           MPI_COMM_WORLD, ISTAT, IERR)
```



# 資料傳送的合併(5)



```
INTEGER  IBLOCK(2), IDISP(2), ITYPE(2), ISTAT(MPI_STATUS_SIZE)
IBLOCK(1)=N
IBLOCK(2)=N
ITYPE(1)=MPI_REAL8
ITYPE(2)=MPI_REAL8
CALL  MPI_ADDRESS( PS1(1,ISTARTM1), IDISP(1), IERR)
CALL  MPI_ADDRESS( PS2(1,ISTARTM1), IDISP(2), IERR)
DO I=2,1,-1
    IDISP(I)=IDISP(I) – IDISP(1)
ENDDO
CALL MPI_TYPE_STRUCT (2, IBLOCK, IDISP, ITYPE, IPACK2, IERR)
CALL MPI_TYPE_COMMIT (IPACK2, IERR)
CALL MPI_SENDRECV (PS1(1,IEND),        1, IPACK2,  R_NBR,  110,
1                              PS1(1,ISTARTM1), 1, IPACK2,  L_NBR,  110,
2                              MPI_COMM_WORLD, ISTAT, IERR)
```

# 平行程式的效率提昇

- Nonblocking 資料傳送
- 傳送資料的合併 (含MPI_STRUCT)
- **以邊界資料計算取代邊界資料交換**
- 事先切割輸入資料與事後收集切割過的輸出資料

# 邊界資料計算取代交換(1)

- 在loop-30之前已經備妥PS2及U2 的I-1、 I+1邊界資料時，在loop-30裏由PS2導出的陣列D1在loop-40裏要用到I-1的邊界資料，因此必需在loop-40 之前交換D1的I-1邊界資料。

```
CC  DO 30 I=1,M1
      DO 30 I=ISTART,IEND1
      DO 30 J=1,N1
         D1(J,I)=(PS2(J,I+1)+PS2(J,I))*HXU(J,I)*0.5D0
         D2(J,I)=(PS2(J+1,I)+PS2(J,I))*HXV(J,I)*0.5D0
 30  CONTINUE
```

# 邊界資料計算取代交換(2)

```
      CALL MPI_SENDRECV(
     1       D1(1,IEND),         N, MPI_REAL8,  R_NBR, ITAG,
     2       D1(1,ISTARTM1), N, MPI_REAL8,  L_NBR, ITAG,
     3       MPI_COMM_WORLD, ISTAT, IERR)

CC   DO 40 I=2,M1
      DO 40 I=ISTART2,IEND1
      DO 40 K=1,KK
      DO 40 J=2,N1
         DUMMY1(J,K,I)=(D1(J,I)*U2(J,K,I)-D1(J,I-1)*U2(J,K,I-1))
    1   *HMMX(J,I)+(D2(J,I)*V2(J,K,I)-D2(J-1,I)*V2(J-1,K,I))*HMMY(J,I)
 40   CONTINUE
```

# 邊界資料計算取代交換(3)

- 既然在loop-30之前已經備妥PS2及U2的I-1、I+1邊界資料，就可以在loop-30裏多算I-1 的邊界資料，用來取代loop-30之後陣列D1的邊界資料交換。如下

```
CC  DO 30 I=1, M1
CC  DO 30 I=ISTART,IEND1
      DO 30 I=ISTARTM1,IEND1
      DO 30 J=1,N1
         D1(J,I)=(PS2(J,I+1)+PS2(J,I))*HXU(J,I)*0.5D0
         D2(J,I)=(PS2(J+1,I)+PS2(J,I))*HXV(J,I)*0.5D0
 30  CONTINUE
```

# 邊界資料計算取代交換(4)

```
CC  DO 40 I=2,M1
      DO 40 I=ISTART2,IEND1
      DO 40 K=1,KK
      DO 40 J=2,N1
         DUMMY1(J,K,I)=(D1(J,I)*U2(J,K,I)-D1(J,I-1)*U2(J,K,I-1))
     1                      *HMMX(J,I)+(D2(J,I)*V2(J,K,I)
     2                      -D2(J-1,I)*V2(J1,K,I))*HMMY(J,I)
 40  CONTINUE
```

# 平行程式的效率提昇

- Nonblocking 資料傳送
- 傳送資料的合併 (含MPI_STRUCT)
- 以邊界資料計算取代邊界資料交換
- **事先切割輸入資料與事後收集切割過的輸出資料**

# 事先切割輸入資料(1)

- 下列程式片段是把一維陣列B、C、D讀進來之後，切割為NP段，分別寫到檔名為input.11、input.12、input.13、 . . .等磁檔上

```
      CHARACTER  FNAME*8
       REAL*8       A(MM), B(MM), C(MM), D(MM)
       INTEGER    GSTART(0:7),GEND(0:7),GCNT(0:7)
      NP=4
      OPEN(7, FILE='input.dat', FORM='UNFORMATTED')
      READ (7) B, C, D
      CALL STARTEND(NP,1,MM,GSTART,GEND,GCNT)
101 FORMAT('input.',I2)
```

# 事先切割輸入資料(2)

```
DO II=0,NP-1
    IU=11+II
    WRITE(FNAME,101) IU
    OPEN(IU,FILE=FNAME,FORM='UNFORMATTED')
    ISTART=GSTART(II)
    IEND=GEND(II)
    WRITE(IU) (B(I), I=ISTART, IEND)
    WRITE(IU) (C(I), I=ISTART, IEND)
    WRITE(IU) (D(I), I=ISTART, IEND)
    CLOSE(IU)
ENDDO
```

# 事先切割輸入資料(3)

- 平行程式依相同的檔案命名方式從公用磁檔讀入該CPU所需要的資料如下:

```
      PARAMETER (MM=200, NP=4, M=MM/NP)
      INCLUDE        'mpif.h'
      REAL*8          A(0:M+1), B(0:M+1), C(0:M+1), D(0:M+1)
      INTEGER         NPROC, MYID, ISTART, IEND
     CHARACTER  FNAME*8
     CALL MPI_INIT (IERR)
     CALL MPI_COMM_SIZE  (MPI_COMM_WORLD,
    1                                NPROC, IERR)
     CALL MPI_COMM_RANK (MPI_COMM_WORLD,
    1                                 MYID, IERR)
```

# 事先切割輸入資料(4)

```
      ISTART=1
      IEND=M
      ....................
101 FORMAT ('input.',I2)
      IU=MYID+11
      WRITE(FNAME,101) IU
      OPEN(IU,FILE=FNAME,STATUS='OLD',
      &                  FORM='UNFORMATTED')
      READ (IU)  (B(I), I=ISTART, IEND)
      READ (IU)  (C(I), I=ISTART, IEND)
      READ (IU)  (D(I), I=ISTART, IEND)
```

# 事後收集切割過的輸出資料(1)

- 平行程式每個CPU寫出該CPU轄區內的資料到各別的檔案裏，然後寫一個小程式來收集這些檔案裏的資料，最後再寫成一個完整的檔案。例如在平行程式裏把計算出來的A陣列片段寫到output.xx檔裏如下:

```
PARAMETER (MM=200, NP=4, M=MM/NP)
INCLUDE       'mpif.h'
REAL*8        A(0:M+1), B(0:M+1), C(0:M+1), D(0:M+1)
INTEGER       NPROC, MYID, ISTART, IEND
CHARACTER  FNAME*12
```

# 事後收集切割過的輸出資料(2)

```
      CALL MPI_INIT (IERR)
      CALL MPI_COMM_SIZE (MPI_COMM_WORLD, NPROC, IERR)
      CALL MPI_COMM_RANK (MPI_COMM_WORLD, MYID, IERR)
      ISTART=1
      IEND=M
      . . . . . .
101   FORMAT ('output.',I2)
      IU=MYID+11
      WRITE (FNAME,101)  IU
      OPEN (IU, FILE=FNAME, STATUS='NEW',
       &                    FORM='UNFORMATTED')
      WRITE (IU) (A(I), I=ISTART, IEND)
```

# 事後收集切割過的輸出資料(3)

- 下面的程式片段是從output.11、output.12、output.13、 . . .等NP個磁檔讀入陣列的部分資料，組合成一個完整的陣列資料，再寫出去。

```
      CHARACTER  FNAME*12
      REAL*8          A(MM), B(MM), C(MM), D(MM)
      NP=4
101   FORMAT('output.',I2)
```

# 事後收集切割過的輸出資料(4)

```
      DO I=0,NP-1
         IU=11+I
         WRITE (FNAME,101) IU
         OPEN (IU, FILE=FNAME, STATUS='OLD',
         &                    FORM='UNFORMATTED')
         CALL STARTEND (I, NP, 1, MM, ISTART, IEND)
         READ (IU)  (A(I), I=ISTART, IEND)
         CLOSE (IU)
       ENDDO
       OPEN (7,FILE='output', STATUS='new',
        &                          FORM='unformatted')
       WRITE (7)  A
       CLOSE (7)
```