

Report S1

Fortran

Kengo Nakajima
Information Technology Center
The University of Tokyo

Report S1 (1/2)

- Problem S1-1
 - Read local files <\$O-S1>/a1.0~a1.3, <\$O-S1>/a2.0~a2.3.
 - Develop codes which calculate norm $\|x\|$ of global vector for each case.
 - <\$O-S1>file.c, <\$T-S1>file2.c
- Problem S1-2
 - Read local files <\$O-S1>/a2.0~a2.3.
 - Develop a code which constructs “global vector” using MPI_Allgatherv.

Report S1 (2/2)

- Problem S1-3
 - Develop parallel program which calculates the following numerical integration using “trapezoidal rule” by MPI_Reduce, MPI_Bcast etc.
 - Measure computation time, and parallel performance

$$\int_0^1 \frac{4}{1+x^2} dx$$

Copying files on Reedbush-U

Copy

```
>$ cd /lustre/gt18/t18XXX/pFEM  
>$ cp /lustre/gt00/z30088/class_eps/F/s1r-f.tar .  
>$ tar xvf s1r-f.tar
```

Confirm directory

```
>$ ls  
mpi  
>$ cd mpi/S1-ref
```

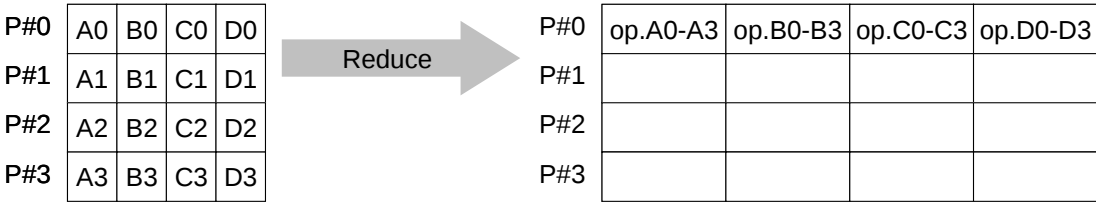
This directory is called as **<\$O-S1r>**.

<\$O-S1r> = <\$O-TOP>/mpi/S1-ref

S1-1: Reading Local Vector, Calc. Norm

- Problem S1-1
 - Read local files <\$O-S1>/a1.0~a1.3, <\$O-S1>/a2.0~a2.3.
 - Develop codes which calculate norm $\|x\|$ of global vector for each case.
- Use MPI_Allreduce (or MPI_Reduce)
- Advice
 - Checking each component of variables and arrays !

MPI_REDUCE



- Reduces values on all processes to a single value
 - Summation, Product, Max, Min etc.
- call **MPI_REDUCE**
(sendbuf, recvbuf, count, datatype, op, root, comm, ierr)
 - **sendbuf** choice I starting address of send buffer
 - **recvbuf** choice O starting address receive buffer
 - **count** I I number of elements in send/receive buffer
 - **datatype** I I data type of elements of send/recive buffer
 - FORTTRAN MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_CHARACTER etc.
 - C MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_CHAR etc
 - **op** I I reduce operation
 - MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND etc
 - Users can define operations by **MPI_OP_CREATE**
 - **root** I I rank of root process
 - **comm** I I communicator
 - **ierr** I O completion code

Send/Receive Buffer (Sending/Receiving)

- Arrays of “send (sending) buffer” and “receive (receiving) buffer” often appear in MPI.
- Addresses of “send (sending) buffer” and “receive (receiving) buffer” must be different.

“op” of MPI_Reduce/Allreduce

call MPI_REDUCE

(sendbuf, recvbuf, count, datatype, op, root, comm, ierr)

- **MPI_MAX, MPI_MIN** Max, Min
- **MPI_SUM, MPI_PROD** Summation, Product
- **MPI LAND** Logical AND

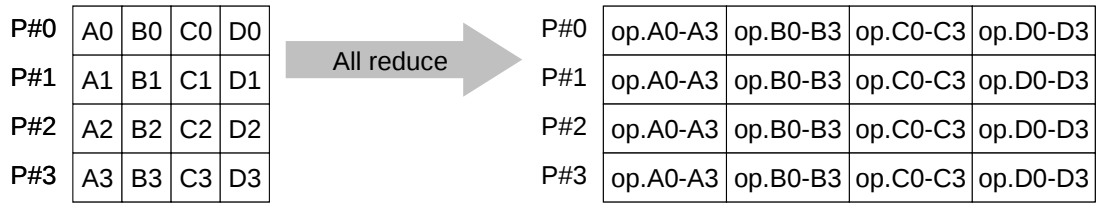
```
double x0, xsum;
```

```
MPI_Reduce  
(&x0, &xsum, 1, MPI_DOUBLE, MPI_SUM, 0, <comm>)
```

```
double x0[4];
```

```
MPI_Reduce  
(&x0[0], &x0[2], 2, MPI_DOUBLE_PRECISION, MPI_SUM, 0, <comm>)
```


MPI_ALLREDUCE



- MPI_Reduce + MPI_Bcast
- Summation (of dot products) and MAX/MIN values are likely to utilized in each process
- call MPI_ALLREDUCE
(sendbuf, recvbuf, count, datatype, op, comm, ierr)
 - sendbuf choice I starting address of send buffer
 - recvbuf choice 0 starting address receive buffer
 type is defined by "datatype"
 - count I I number of elements in send/recv buffer
 - datatype I I data type of elements in send/recv buffer
 - op I I reduce operation
 - comm I I commuinicator
 - ierr I 0 completion code

S1-1: Local Vector, Norm Calculation

Uniform Vectors (a1.*): s1-1-for_a1.f

```

implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, SOLVER_COMM, ierr
real(kind=8), dimension(8) :: VEC
character(len=80)          :: filename

call MPI_INIT          (ierr)
call MPI_COMM_SIZE     (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK     (MPI_COMM_WORLD, my_rank, ierr )

if (my_rank.eq.0) filename= 'a1.0'
if (my_rank.eq.1) filename= 'a1.1'
if (my_rank.eq.2) filename= 'a1.2'
if (my_rank.eq.3) filename= 'a1.3'

N=8

open (21, file= filename, status= 'unknown')
do i= 1, N
  read (21,*) VEC(i)
enddo

sum0= 0.d0
do i= 1, N
  sum0= sum0 + VEC(i)**2
enddo

call MPI_allREDUCE (sum0, sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM, MPI_COMM_WORLD, ierr)
sum= dsqrt(sum)

if (my_rank.eq.0) write (*,'(1pe16.6)') sum

call MPI_FINALIZE (ierr)
stop
end

```

write(filename, '(a,i1.1)') 'a1.', my_rank

**call MPI_Allreduce
(sendbuf, recvbuf, count, datatype, op, comm, ierr)**

S1-1: Local Vector, Norm Calculation

Uniform Vectors (a1.*): s1-1-for_a2.f

```

implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, SOLVER_COMM, ierr
real(kind=8), dimension(:), allocatable :: VEC, VEC2
character(len=80) :: filename

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

if (my_rank.eq.0) filename= 'a2.0'
if (my_rank.eq.1) filename= 'a2.1'
if (my_rank.eq.2) filename= 'a2.2'
if (my_rank.eq.3) filename= 'a2.3'

open (21, file= filename, status= 'unknown')
  read (21,*) N
  allocate (VEC(N))
  do i= 1, N
    read (21,*) VEC(i)
  enddo

sum0= 0.d0
do i= 1, N
  sum0= sum0 + VEC(i)**2
enddo

call MPI_Allreduce
  (sendbuf,recvbuf,count,datatype,op, comm,ierr)

call MPI_allREDUCE (sum0, sum, 1, MPI_DOUBLE_PRECISION, MPI_SUM, MPI_COMM_WORLD, ierr)
sum= dsqrt (sum)

if (my_rank.eq.0) write (*,'(1pe16.6)') sum

call MPI_FINALIZE (ierr)
stop
end

```

S1-1: Running the Codes

FORTAN

```
$ cd /lustre/gt18/t18XXX/pFEM/mpi/S1-ref  
$ mpiifort -O3 s1-1-for_a1.f  
$ mpiifort -O3 s1-1-for_a2.f  
  
(modify "go4.sh")  
$ qsub go4.sh
```

C

```
$ cd /lustre/gt18/t18XXX/pFEM/mpi/S1-ref  
$ mpicc -O3 s1-1-for_a1.c  
$ mpicc -O3 s1-1-for_a2.c  
  
(modify "go4.sh")  
$ qsub go4.sh
```

S1-1: Local Vector, Calc. Norm Results

Results using one core

```
a1.* 1.62088247569032590000E+03  
a2.* 1.22218492872396360000E+03
```

```
$> ifort -O3 dot-a1.f  
$> qsub go1.sh
```

```
$> icc -O3 dot-a2.f  
$> qsub go1.sh
```

Results

```
a1.* 1.62088247569032590000E+03  
a2.* 1.22218492872396360000E+03
```

go1.sh

```
#!/bin/sh  
#PBS -q u-lecture8  
#PBS -N test  
#PBS -l select=1:mpiprocs=1  
#PBS -Wgroup_list=gt18  
#PBS -l walltime=00:05:00  
#PBS -e err  
#PBS -o test.lst  
  
cd $PBS_O_WORKDIR  
. /etc/profile.d/modules.sh  
  
mpirun ./impimap.sh ./a.out
```

S1-1: Local Vector, Calc. Norm

If SENDBUF=RECVBUF, what happens ?

True

```
call MPI_allREDUCE(sum0, sum, 1, MPI_DOUBLE_PRECISION,  
                  MPI_SUM, MPI_COMM_WORLD, ierr)
```

False

```
call MPI_allREDUCE(sum0, sum0, 1, MPI_DOUBLE_PRECISION,  
                  MPI_SUM, MPI_COMM_WORLD, ierr)
```

S1-1: Local Vector, Calc. Norm

If SENDBUF=RECVBUF, what happens ?

True

```
call MPI_allREDUCE(sum0, sum, 1, MPI_DOUBLE_PRECISION,  
                  MPI_SUM, MPI_COMM_WORLD, ierr)
```

False

```
call MPI_allREDUCE(sum0, sum0, 1, MPI_DOUBLE_PRECISION,  
                  MPI_SUM, MPI_COMM_WORLD, ierr)
```

True

```
call MPI_allREDUCE(sumK(1), sumK(2), 1, MPI_DOUBLE_PRECISION,  
                  MPI_SUM, MPI_COMM_WORLD, ierr)
```

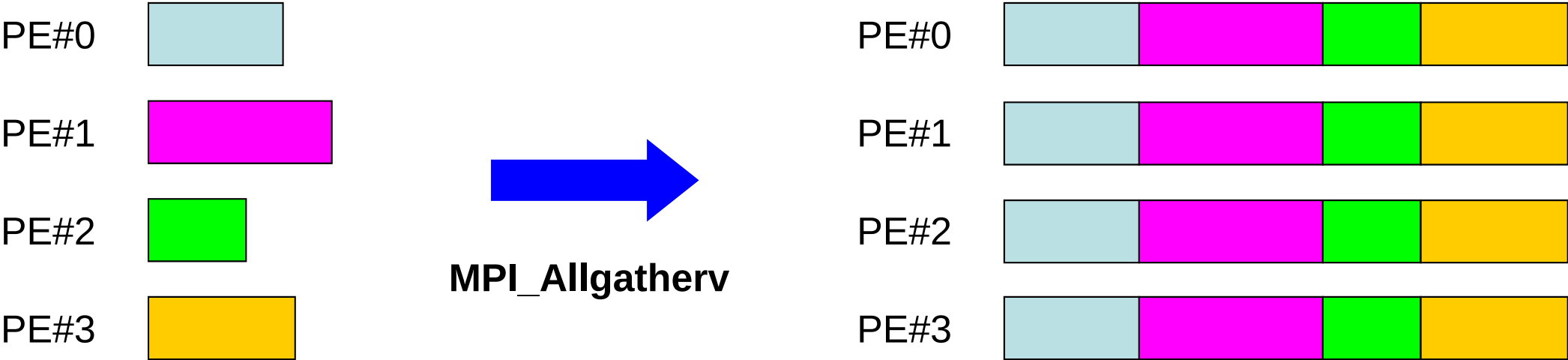
SENDBUF .ne. RECVBUF

S1-2: Local -> Global Vector

- Problem S1-2
 - Read local files <\$O-S1>/a2.0~a2.3.
 - Develop a code which constructs “global vector” using MPI_Allgatherv.

S1-2: Local -> Global Vector

MPI_Allgatherv (1/5)

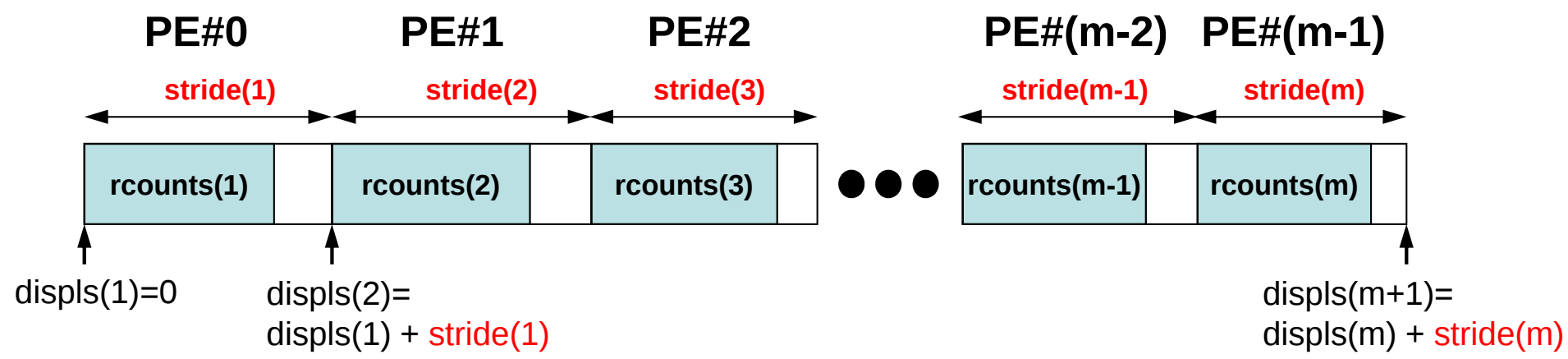


MPI_ALLGATHERV

- Variable count version of MPI_Allgather
 - creates “global data” from “local data”
- call **MPI_ALLGATHERV (sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, comm, ierr)**
 - **sendbuf** choice I starting address of sending buffer
 - **scount** I I number of elements sent to each process
 - **sendtype** I I data type of elements of sending buffer
 - **recvbuf** choice O starting address of receiving buffer
 - **rcounts** I I integer array (of length group size) containing the number of elements that are to be received from each process (array: size= PETOT)
 - **displs** I I integer array (of length group size). Entry *i* specifies the displacement (relative to recvbuf) at which to place the incoming data from process *i* (array: size= PETOT+1)
 - **recvtype** I I data type of elements of receiving buffer
 - **comm** I I communicator
 - **ierr** I O completion code

MPI_ALLGATHERV (cont.)

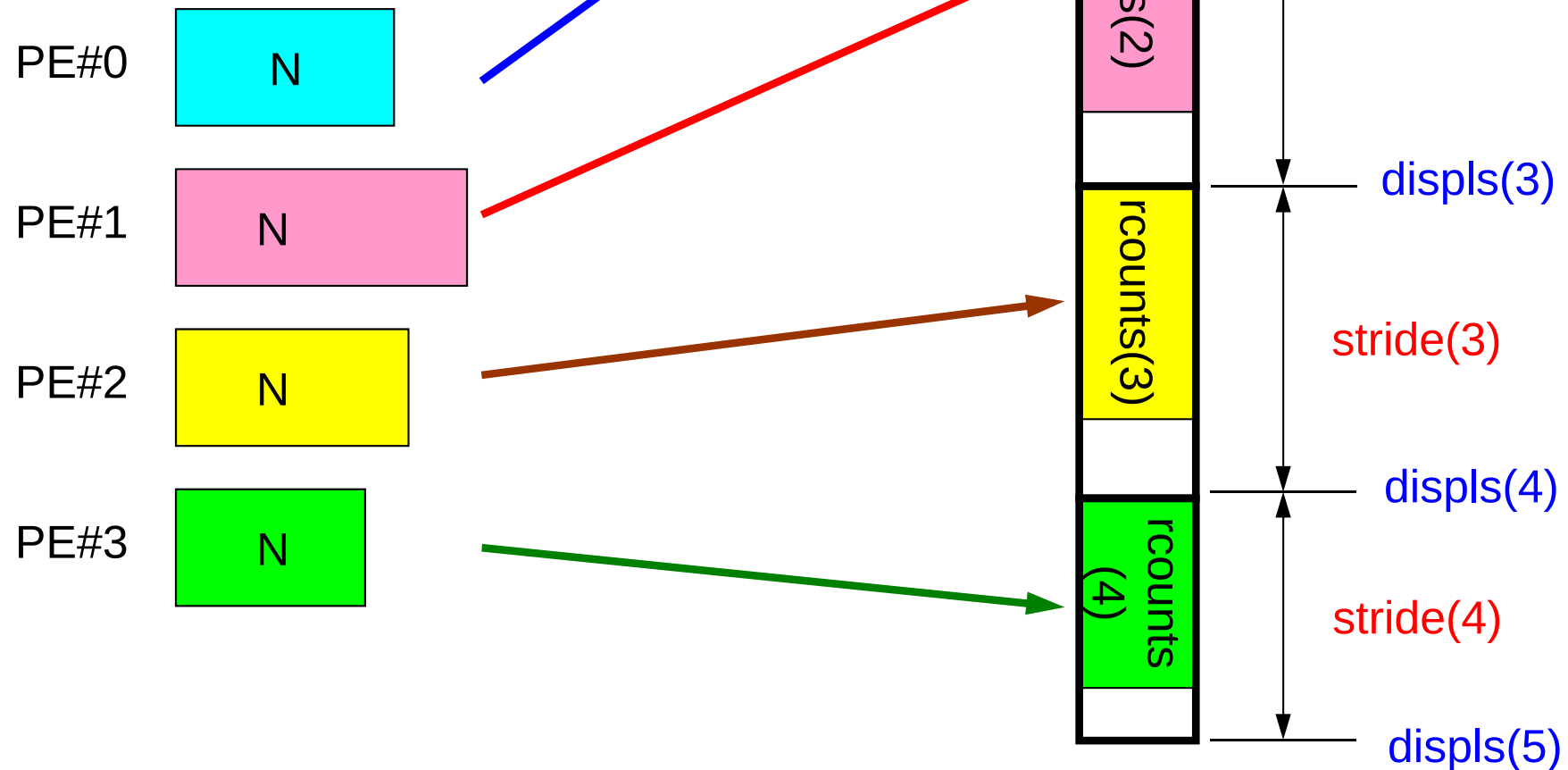
- call `MPI_ALLGATHERV (sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, comm, ierr)`
 - `rcounts` I I integer array (of length group size) containing the number of elements that are to be received from each process (array: size= PETOT)
 - `displs` I I integer array (of length group size). Entry *i* specifies the displacement (relative to `recvbuf`) at which to place the incoming data from process *i* (array: size= PETOT+1)
 - These two arrays are related to size of final “global data”, therefore each process requires information of these arrays (`rcounts`, `displs`)
 - Each process must have same values for all components of both vectors
 - Usually, `stride(i)=rcounts(i)`



$$\text{size(recvbuf)} = \text{displs(PETOT+1)} = \text{sum(stride)}$$

What MPI_Allgatherv is doing

Generating global data from local data

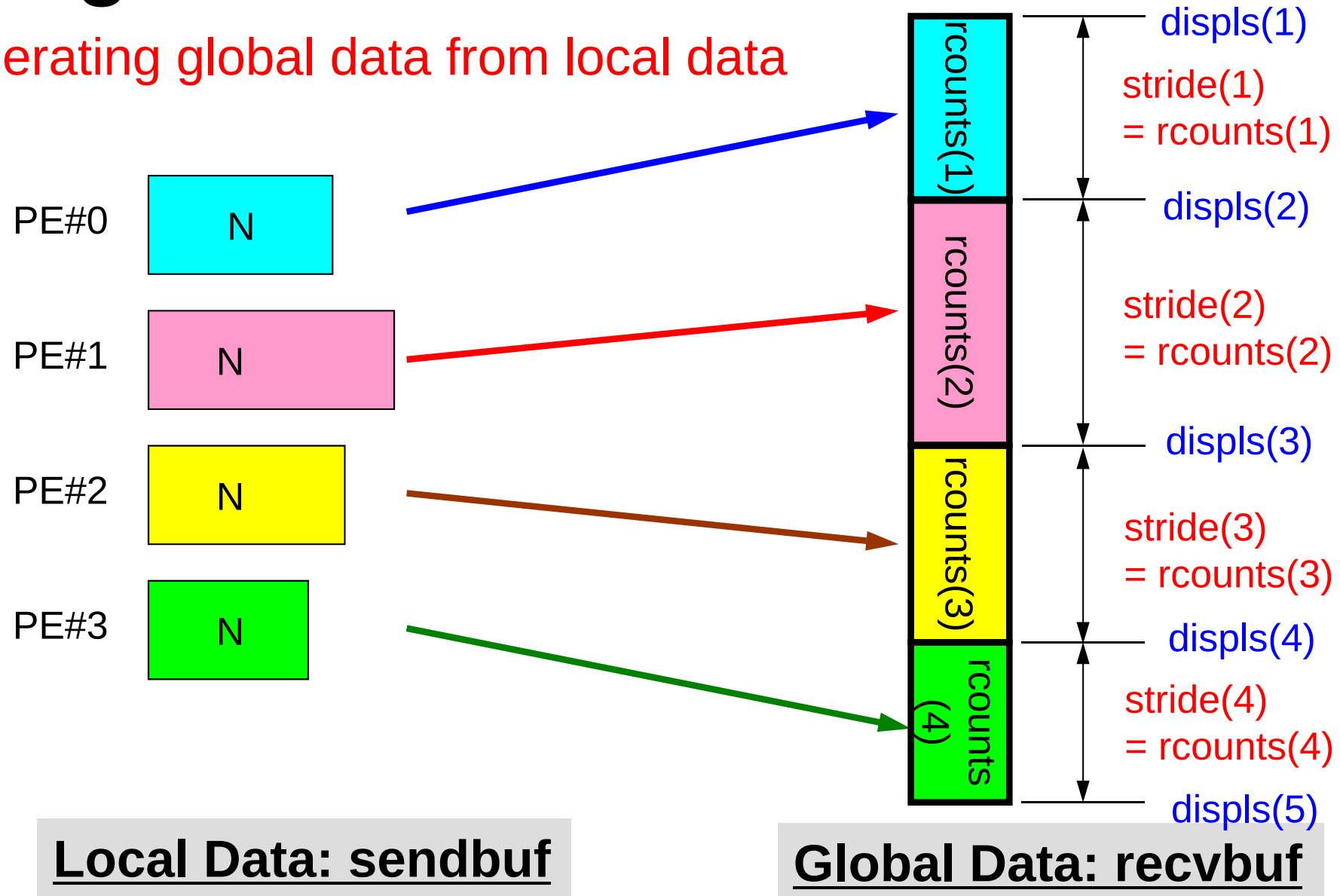


Local Data: sendbuf

Global Data: recvbuf

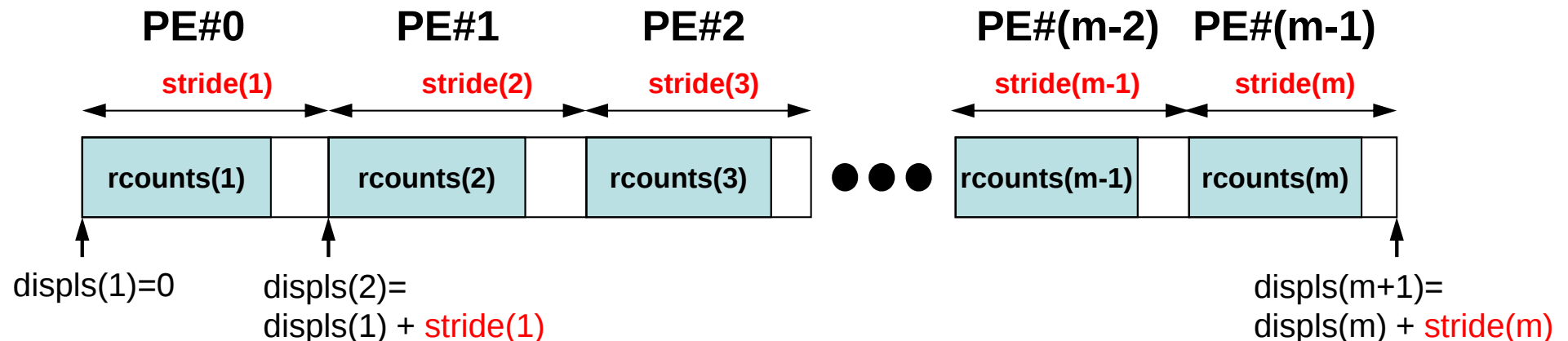
What MPI_Allgatherv is doing

Generating global data from local data



MPI_Allgatherv in detail (1/2)

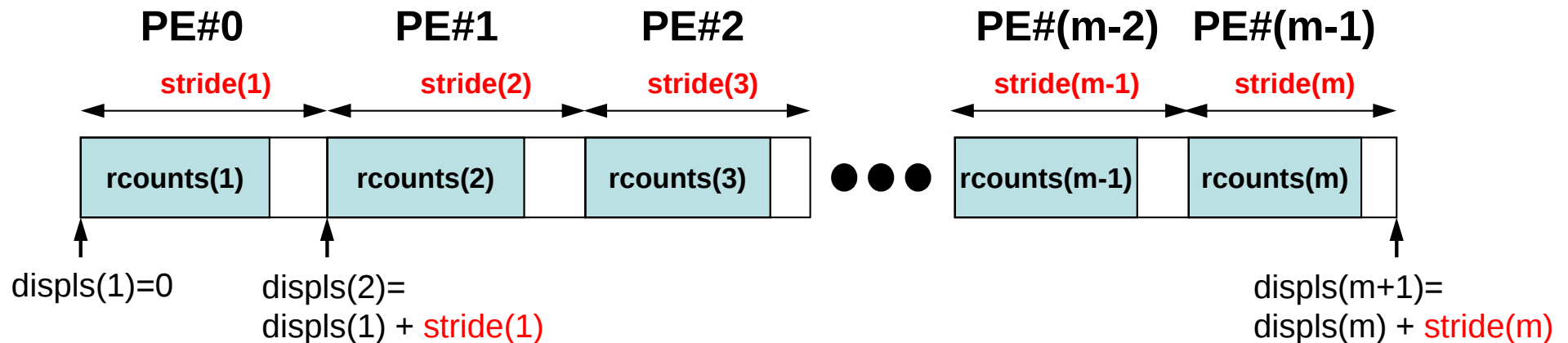
- call `MPI_ALLGATHERV (sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, comm, ierr)`
- **rcounts**
 - Size of message from each PE: Size of Local Data (Length of Local Vector)
- **displs**
 - Address/index of each local data in the vector of global data
 - **displs (PETOT+1) = Size of Entire Global Data (Global Vector)**



$$\text{size(recvbuf)} = \text{displs}(\text{PETOT}+1) = \text{sum}(\text{stride})$$

MPI_Allgatherv in detail (2/2)

- Each process needs information of **rcounts** & **displs**
 - "**rcounts**" can be created by gathering local vector length "**N**" from each process.
 - On each process, "**displs**" can be generated from "**rcounts**" on each process.
 - $\text{stride}[i] = \text{rcounts}[i]$
 - Size of "**recvbuf**" is calculated by summation of "**rcounts**".



$$\text{size(recvbuf)} = \text{displs}(\text{PETOT}+1) = \text{sum}(\text{stride})$$

Preparation for MPI_Allgatherv <\$O-S1>/agv.f

- “Generating global vector from “a2.0”~”a2.3”.
- Length of the each vector is 8, 5, 7, and 3, respectively. Therefore, size of final global vector is 23 (= 8+5+7+3).

a2.0~a2.3

PE#0

8
101.0
103.0
105.0
106.0
109.0
111.0
121.0
151.0

PE#1

5
201.0
203.0
205.0
206.0
209.0

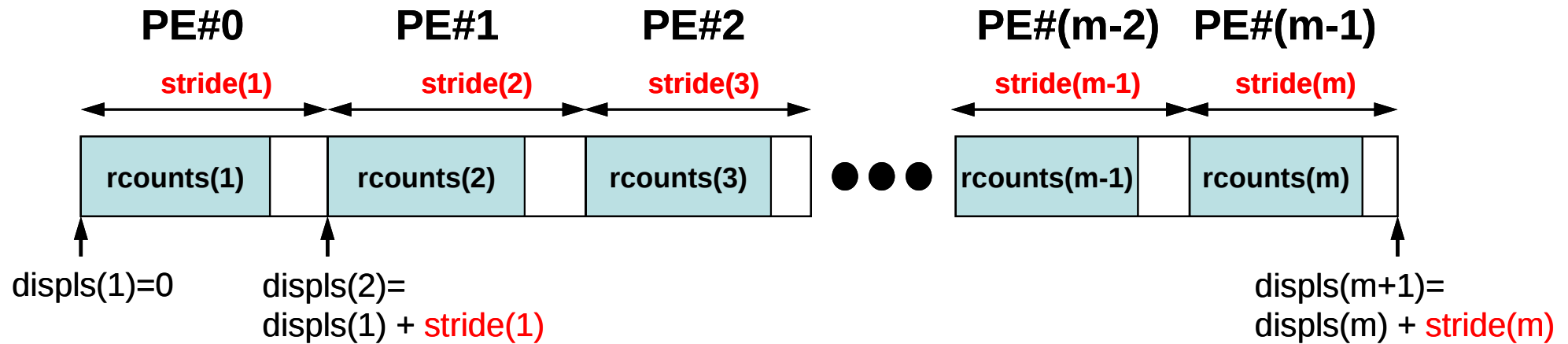
PE#2

7
301.0
303.0
305.0
306.0
311.0
321.0
351.0

PE#3

3
401.0
403.0
405.0

S1-2: Local -> Global Vector



$$\text{size(recvbuf)} = \text{displs}(\text{PETOT}+1) = \text{sum}(\text{stride})$$

- Read local vectors
- Create “rcounts” and “displs”
- Prepare “recvbuf”
- Do “Allgatherv”

S1-2: Local -> Global Vector (1/2)

s1-2.f

```

implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, SOLVER_COMM, ierr
real(kind=8), dimension(:), allocatable :: VEC, VEC2, VECg
integer (kind=4), dimension(:), allocatable :: COUNT, COUNTindex
character(len=80) :: filename

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

if (my_rank.eq.0) filename= 'a2.0'
if (my_rank.eq.1) filename= 'a2.1'
if (my_rank.eq.2) filename= 'a2.2'
if (my_rank.eq.3) filename= 'a2.3'

open (21, file= filename, status= 'unknown')
  read (21,*) N
  allocate (VEC(N))
  do i= 1, N
    read (21,*) VEC(i)
  enddo

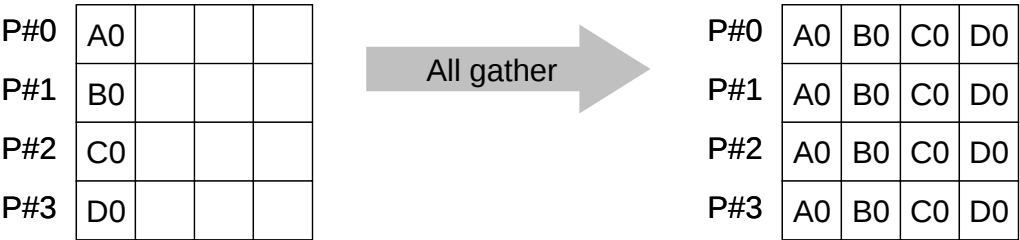
allocate (COUNT(PETOT), COUNTindex(PETOT+1))
call MPI_allGATHER ( N      , 1, MPI_INTEGER,
&                  COUNT, 1, MPI_INTEGER,
&                  MPI_COMM_WORLD, ierr)
COUNTindex(1)= 0

do ip= 1, PETOT
  COUNTindex(ip+1)= COUNTindex(ip) + COUNT(ip)
enddo

```

“COUNT (rcounts)”
vector length at each PE

MPI_ALLGATHER



- MPI_GATHER+MPI_BCAST
 - Gathers data from all tasks and distribute the combined data to all tasks
- call MPI_ALLGATHER (sendbuf, scount, sendtype, recvbuf, rcount, recvtype, comm, ierr)
 - sendbuf choice I starting address of sending buffer
 - scount I I number of elements sent to each process
 - sendtype I I data type of elements of sending buffer
 - recvbuf choice O starting address of receiving buffer
 - rcount I I number of elements received from each process
 - recvtype I I data type of elements of receiving buffer
 - comm I I communicator
 - ierr I O completion code

S1-2: Local -> Global Vector (2/2)

s1-2.f

```

do ip= 1, PETOT
  COUNTindex(ip+1)= COUNTindex(ip) + COUNT(ip)
enddo

allocate (VECg(COUNTindex(PETOT+1)))
VECg= 0.d0

call MPI_allGATHERv
&      ( VEC , N, MPI_DOUBLE_PRECISION,
&      VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION,
&      MPI_COMM_WORLD, ierr)

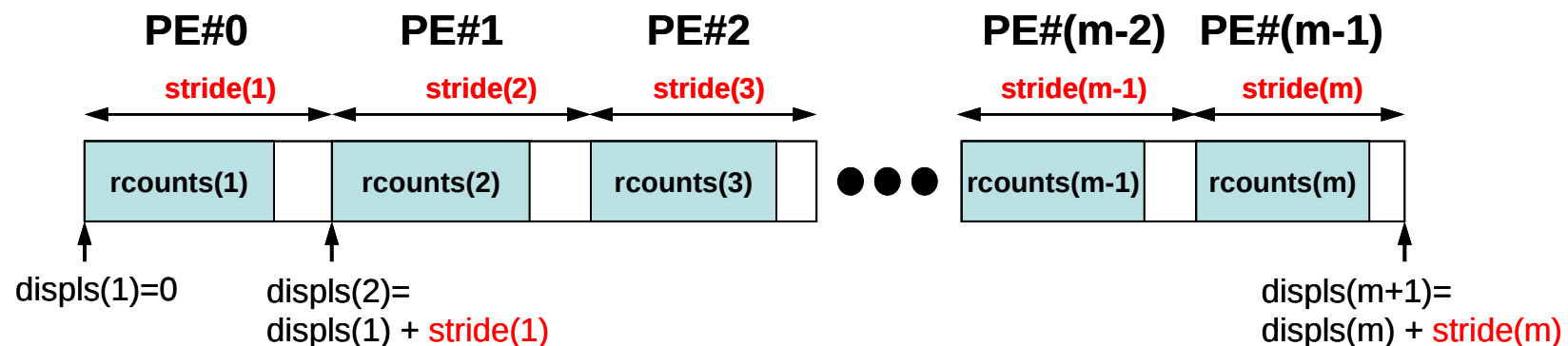
do i= 1, COUNTindex(PETOT+1)
  write (*,'(2i8,f10.0)') my_rank, i, VECg(i)
enddo

call MPI_FINALIZE (ierr)

stop
end

```

Creating “COUNTindex (displs)”



S1-2: Local -> Global Vector (2/2)

s1-2.f

```

do ip= 1, PETOT
  COUNTindex(ip+1)= COUNTindex(ip) + COUNT(ip)
enddo

allocate (VECg(COUNTindex(PETOT+1)))
VECg= 0.d0

call MPI_allGATHERv
&      ( VEC , N, MPI_DOUBLE_PRECISION,
&      VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION,
&      MPI_COMM_WORLD, ierr)

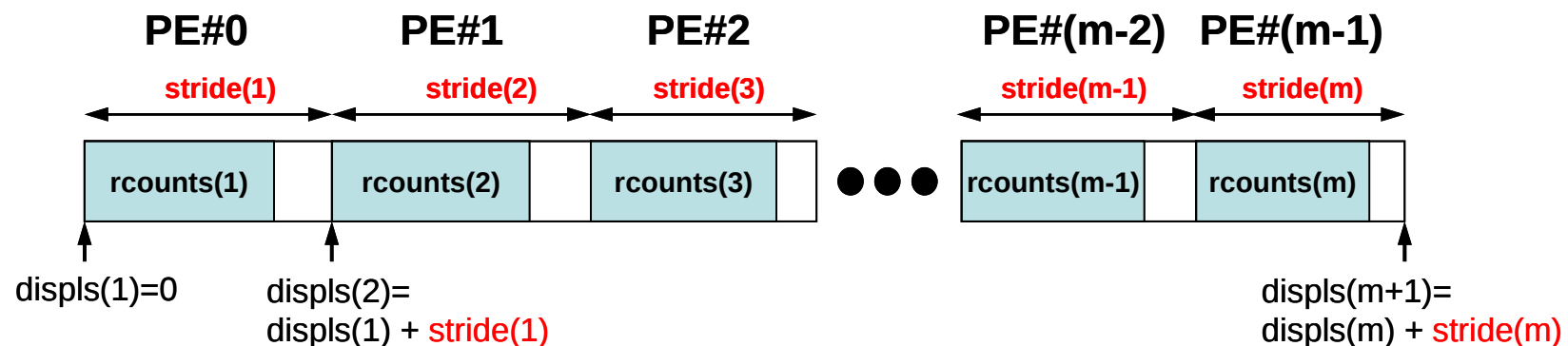
do i= 1, COUNTindex(PETOT+1)
  write (*,'(2i8,f10.0)') my_rank, i, VECg(i)
enddo

call MPI_FINALIZE (ierr)

stop
end

```

“recvbuf”



S1-2: Local -> Global Vector (2/2)

s1-2.f

```
do ip= 1, PETOT
  COUNTindex(ip+1)= COUNTindex(ip) + COUNT(ip)
enddo

allocate (VECg(COUNTindex(PETOT+1)))
VECg= 0.d0
```

```
call MPI_allGATHERv
&    ( VEC , N, MPI_DOUBLE_PRECISION,
&    VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION,
&    MPI_COMM_WORLD, ierr)
```

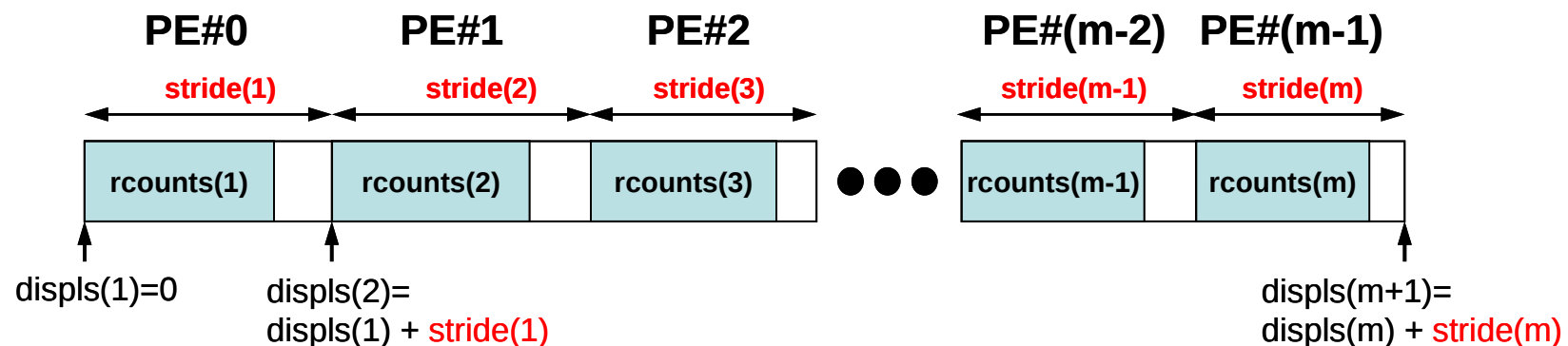
&
&
&

```
do i= 1, COUNTindex(PETOT+1)
  write (*,'(2i8,f10.0)') my_rank, i, VECg(i)
enddo
```

```
call MPI_FINALIZE (ierr)
```

```
stop
end
```

call MPI_ALLGATHERV
(sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, comm, ierr)



S1-2: Running the Codes

```
$ mpiifort -O3 s1-2.f
```

```
(modify "go4.sh")
```

```
$ qsub go4.sh
```

S1-2: Results

my_rank	ID	VAL
0	1	101.
0	2	103.
0	3	105.
0	4	106.
0	5	109.
0	6	111.
0	7	121.
0	8	151.
0	9	201.
0	10	203.
0	11	205.
0	12	206.
0	13	209.
0	14	301.
0	15	303.
0	16	305.
0	17	306.
0	18	311.
0	19	321.
0	20	351.
0	21	401.
0	22	403.
0	23	405.

my_rank	ID	VAL
1	1	101.
1	2	103.
1	3	105.
1	4	106.
1	5	109.
1	6	111.
1	7	121.
1	8	151.
1	9	201.
1	10	203.
1	11	205.
1	12	206.
1	13	209.
1	14	301.
1	15	303.
1	16	305.
1	17	306.
1	18	311.
1	19	321.
1	20	351.
1	21	401.
1	22	403.
1	23	405.

my_rank	ID	VAL
2	1	101.
2	2	103.
2	3	105.
2	4	106.
2	5	109.
2	6	111.
2	7	121.
2	8	151.
2	9	201.
2	10	203.
2	11	205.
2	12	206.
2	13	209.
2	14	301.
2	15	303.
2	16	305.
2	17	306.
2	18	311.
2	19	321.
2	20	351.
2	21	401.
2	22	403.
2	23	405.

my_rank	ID	VAL
3	1	101.
3	2	103.
3	3	105.
3	4	106.
3	5	109.
3	6	111.
3	7	121.
3	8	151.
3	9	201.
3	10	203.
3	11	205.
3	12	206.
3	13	209.
3	14	301.
3	15	303.
3	16	305.
3	17	306.
3	18	311.
3	19	321.
3	20	351.
3	21	401.
3	22	403.
3	23	405.

S1-3: Integration by Trapezoidal Rule

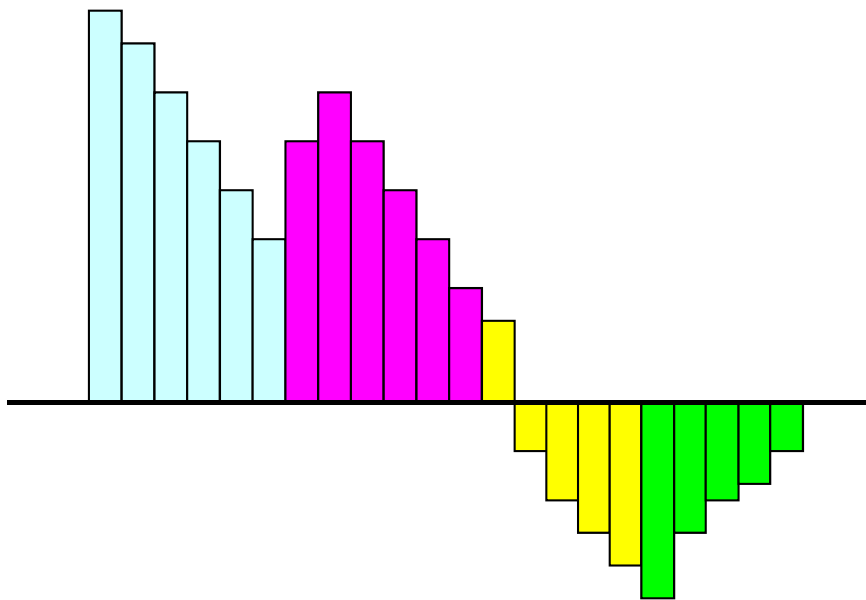
- Problem S1-3
 - Develop parallel program which calculates the following numerical integration using “trapezoidal rule” by MPI_Reduce, MPI_Bcast etc.
 - Measure computation time, and parallel performance

$$\int_0^1 \frac{4}{1+x^2} dx$$

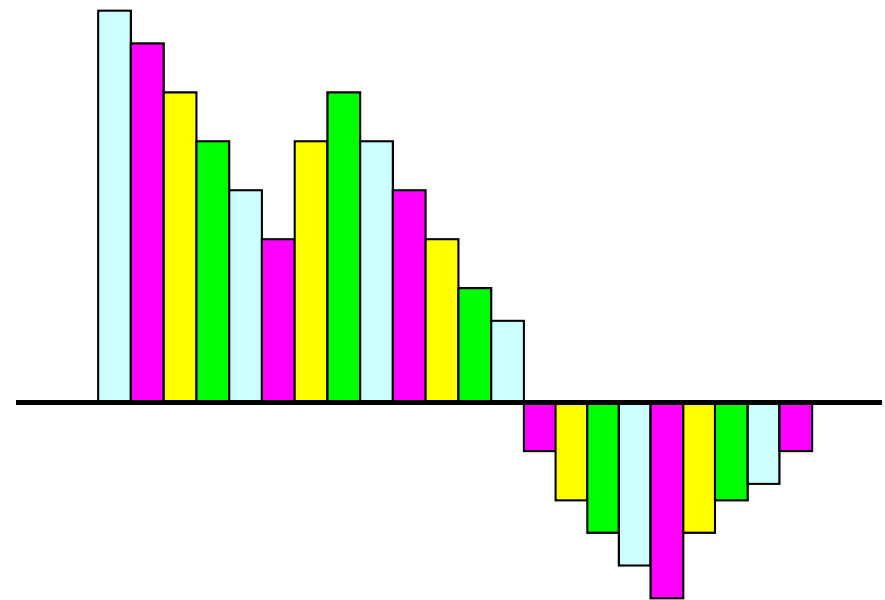
S1-3: Integration by Trapezoidal Rule

Two Types of Load Distribution

Type-A



Type-B



$$\frac{1}{2} \Delta x \left(f_1 + f_{N+1} + \sum_{i=2}^N 2 f_i \right) \text{ corresponds to "Type-A".}$$

S1-3: Integration by Trapezoidal Rule

TYPE-A(1/2) : s1-3a.f

```

implicit REAL*8 (A-H,O-Z)
include 'mpif.h'

integer :: PETOT, my_rank, ierr, N
integer, dimension(:), allocatable :: INDEX
real (kind=8) :: dx

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

allocate (INDEX(0:PETOT))
INDEX= 0

if (my_rank.eq.0) then
  open (11, file='input.dat', status='unknown')
  read (11,*)  N
  close (11)
endif

call MPI_BCAST (N, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
dx= 1.d0 / dfloat(N)

nnn= N / PETOT
nr = N - PETOT * nnn

do ip= 1, PETOT
  if (ip.le.nr) then
    INDEX(ip)= nnn + 1
  else
    INDEX(ip)= nnn
  endif
enddo

```

“N (number of segments) “ is specified in “input.dat”

S1-3: Integration by Trapezoidal Rule

TYPE-A (2/2) : s1-3a.f

```

do ip= 1, PETOT
  INDEX(ip)= INDEX(ip-1) + INDEX(ip)
enddo

Stime= MPI_WTIME()
SUM0= 0.d0
do i= INDEX(my_rank)+1, INDEX(my_rank+1)
  X0= dfloat(i-1) * dx
  X1= dfloat(i) * dx
  F0= 4.d0/(1.d0+X0*X0)
  F1= 4.d0/(1.d0+X1*X1)
  SUM0= SUM0 + 0.50d0 * ( F0 + F1 ) * dx
enddo

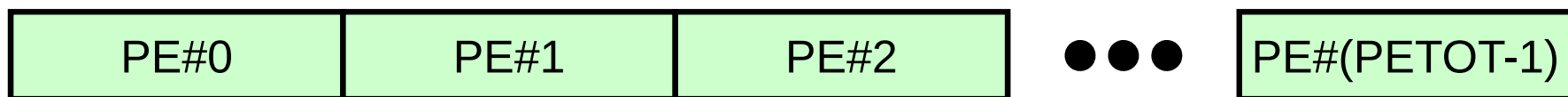
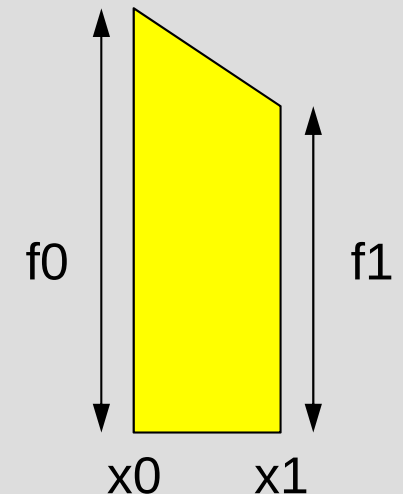
call MPI_REDUCE (SUM0, SUM, 1, MPI_DOUBLE_PRECISION, MPI_SUM, 0, &
& MPI_COMM_WORLD, ierr)
Etime= MPI_WTIME()

if (my_rank.eq.0) write (*,*) SUM, 4.d0*datan(1.d0), Etime-Stime

call MPI_FINALIZE (ierr)

stop
end

```



INDEX(0)+1

INDEX(1)+1

INDEX(2)+1

INDEX(3)+1

INDEX(PETOT-1)+1

INDEX(PETOT)
=N

S1-3: Integration by Trapezoidal Rule

TYPE-B : s1-3b.f

```

implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, ierr, N
real (kind=8) :: dx

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

if (my_rank.eq.0) then
  open (11, file='input.dat', status='unknown')
  read (11,*)  N
  close (11)
endif

call MPI_BCAST (N, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
dx= 1.d0 / dfloat(N)

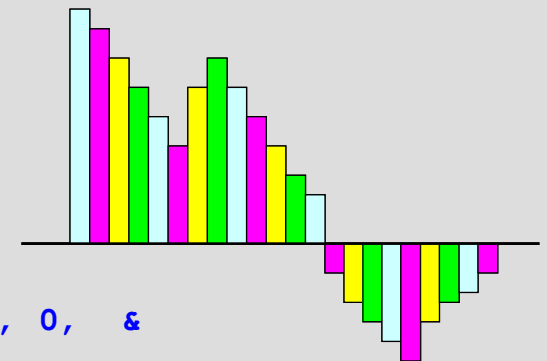
Stime= MPI_WTIME()
SUM0= 0.d0
do i= my_rank+1, N, PETOT
  X0= dfloat(i-1) * dx
  X1= dfloat(i)   * dx
  F0= 4.d0/(1.d0+X0*X0)
  F1= 4.d0/(1.d0+X1*X1)
  SUM0= SUM0 + 0.50d0 * ( F0 + F1 ) * dx
enddo

call MPI_REDUCE (SUM0, SUM, 1, MPI_DOUBLE_PRECISION, MPI_SUM, 0, &
& MPI_COMM_WORLD, ierr)
Etime= MPI_WTIME()

if (my_rank.eq.0) write (*,*) SUM, 4.d0*datan(1.d0), Etime-Stime

call MPI_FINALIZE (ierr)
stop
end

```



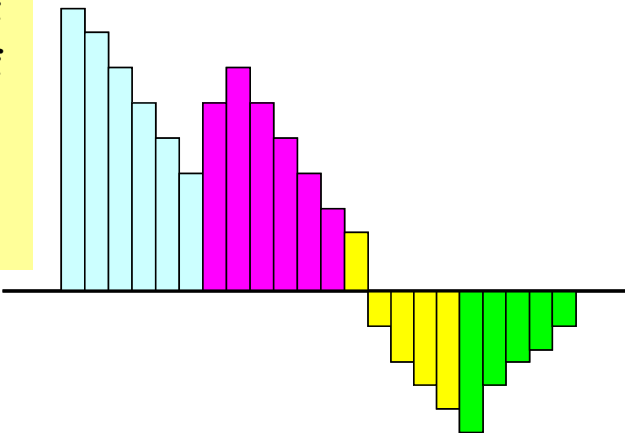
S1-3: Running the Codes

FORTRAN

```
$ mpiifort -O3 -xCORE-AVX2 -align array32byte s1-3a.f
$ mpiifort -O3 -xCORE-AVX2 -align array32byte s1-3b.f

(modify "go.sh")
$ qsub go.sh
```

Type-A

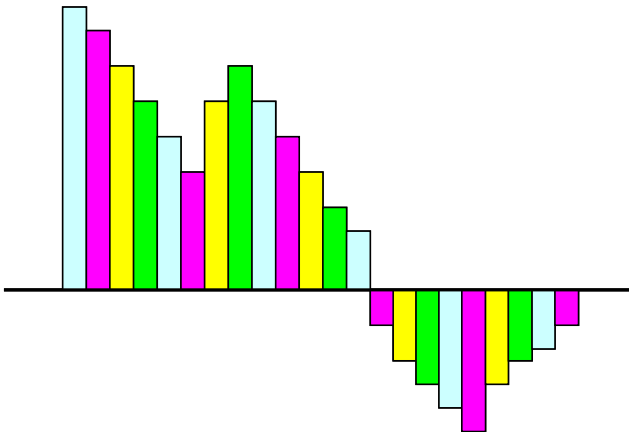


C

```
$ mpicc -O3 -xCORE-AVX2 -align s1-3a.c
$ mpicc -O3 -xCORE-AVX2 -align s1-3b.c

(modify "go.sh")
$ qsub go.sh
```

Type-B

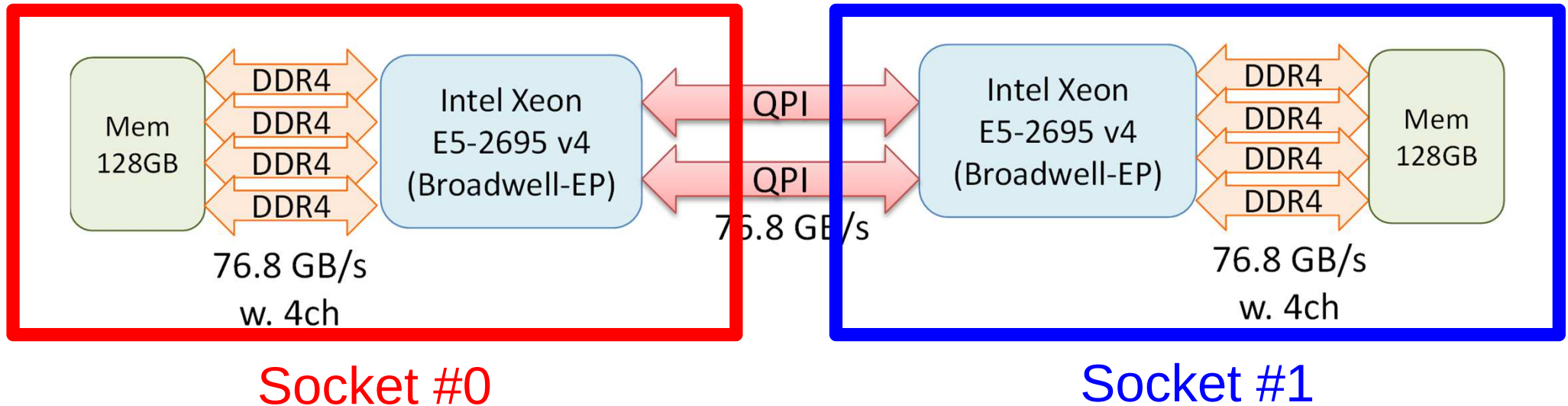


go.sh

<code>#!/bin/sh</code>	
<code>#PBS -q u-lecture8</code>	Name of "QUEUE"
<code>#PBS -N test</code>	Job Name
<code>#PBS -l select=8:mpiprocs=32</code>	node#, proc#/node
<code>#PBS -Wgroup_list=gt18</code>	Group Name (Wallet)
<code>#PBS -l walltime=00:05:00</code>	Computation Time
<code>#PBS -e err</code>	Standard Error
<code>#PBS -o test.lst</code>	Standard Output
<code>cd \$PBS_O_WORKDIR</code>	go to current dir
<code>. /etc/profile.d/modules.sh</code>	(ESSENTIAL)
<code>export I_MPI_PIN_DOMAIN=socket</code>	Execution on each socket
<code>export I_MPI_PERHOST=32</code>	=mpiprocs
<code>mpirun ./impimap.sh ./a.out</code>	Exec's

<code>#PBS -l select=1:mpiprocs=4</code>	1-node, 4-proc's
<code>#PBS -l select=1:mpiprocs=16</code>	1-node, 16-proc's
<code>#PBS -l select=1:mpiprocs=36</code>	1-node, 36-proc's
<code>#PBS -l select=2:mpiprocs=32</code>	2-nodes, 32x2=64-proc's
<code>#PBS -l select=8:mpiprocs=36</code>	8-nodes, 36x8=288-proc's

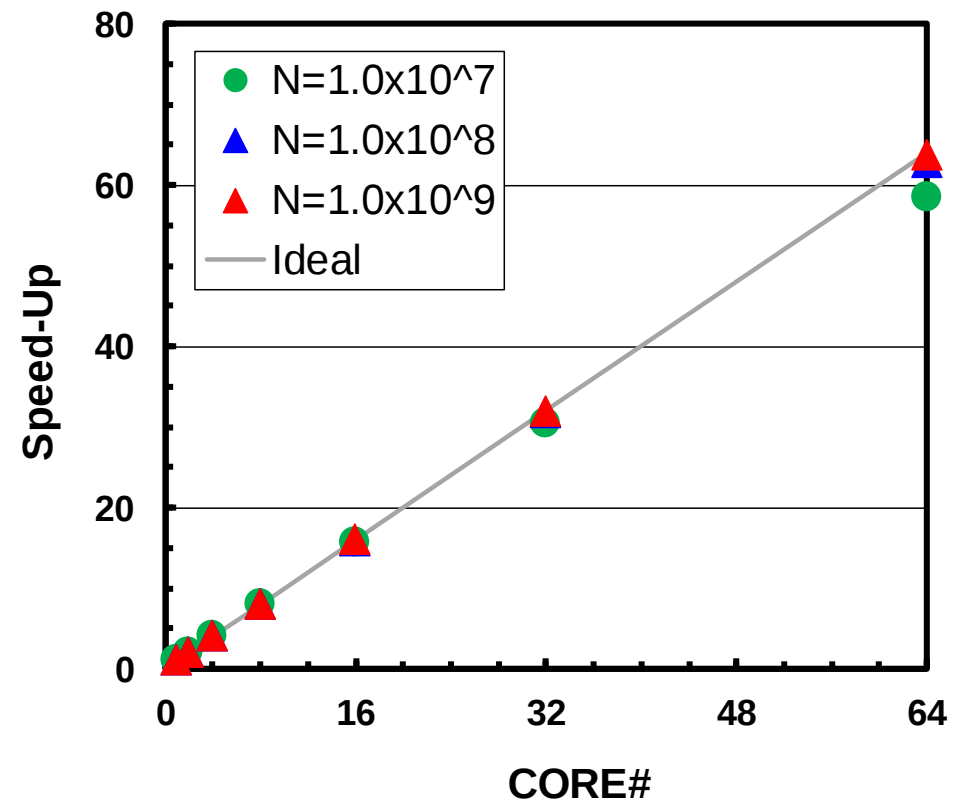
`export I_MPI_PIN_DOMAIN=socket`



- Each Node of Reedbush-U
 - 2 Sockets (CPU's) of Intel Broadwell-EP
 - Each socket has 18 cores
- Each core of a socket can access to the memory on the other socket : NUMA (Non-Uniform Memory Access)
 - `I_MPI_PIN_DOMAIN=socket`, `impimap.sh`: local memory to be used

S1-3: Performance on RB-U (1/4)

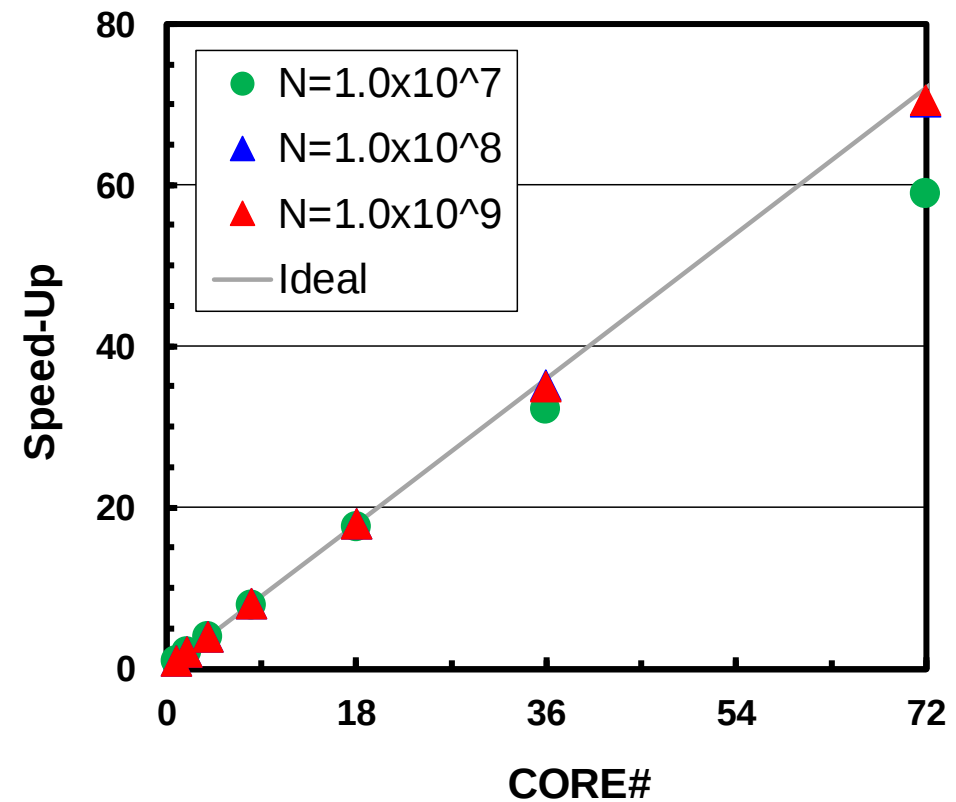
- ◆ : $N=10^7$, ● : 10^8 , ▲ : 10^9 , — : Ideal
- Based on results (sec.) using a single core
- Best of Type-A and Type-B
- Strong Scaling
 - Entire problem size fixed
 - $1/N$ comp. time using N -x cores
- Weak Scaling
 - Problem size/core is fixed
 - Comp. time is kept constant for N -x scale problems using N -x cores



32 cores/node, 16 cores/socket
up to 2 nodes (64 cores)

S1-3: Performance on RB-U (2/4)

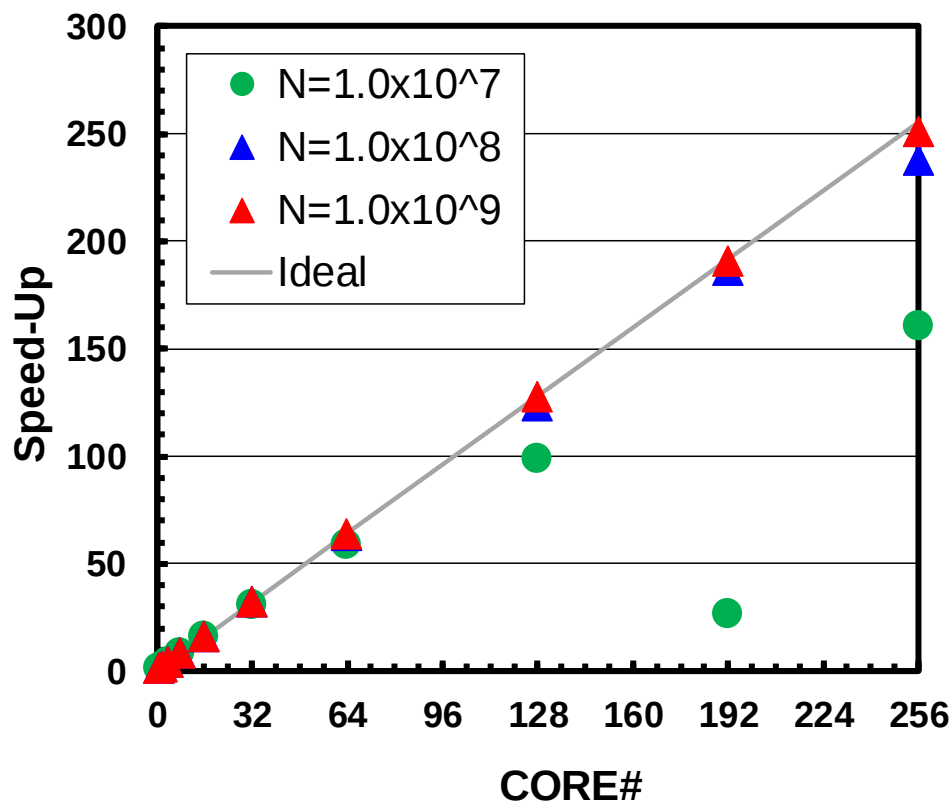
- ◆ : $N=10^7$, ● : 10^8 , ▲ : 10^9 , — : Ideal
- Based on results (sec.) using a single core
- Best of Type-A and Type-B
- Strong Scaling
 - Entire problem size fixed
 - $1/N$ comp. time using $N \times x$ cores
- Weak Scaling
 - Problem size/core is fixed
 - Comp. time is kept constant for $N \times x$ scale problems using $N \times x$ cores



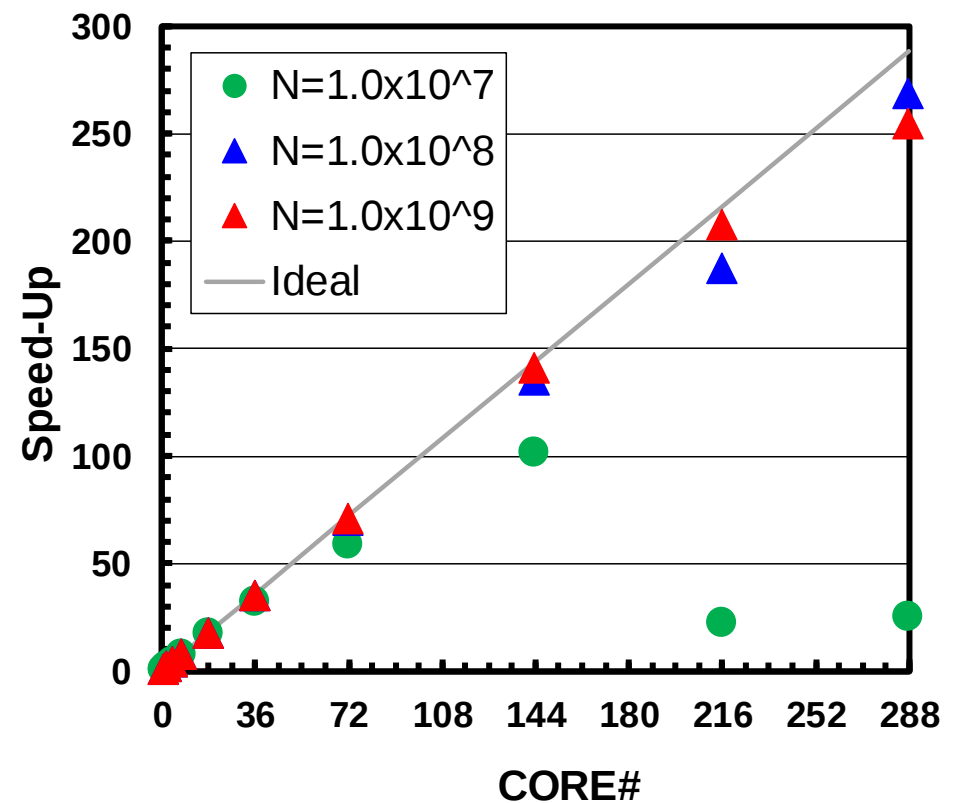
**36 cores/node, 18 cores/socket
up to 2 nodes (72 cores)**

S1-3: Performance on RB-U (3/4)

- ◆ : $N=10^7$, ● : 10^8 , ▲ : 10^9 , — : Ideal
- Based on results (sec.) using a single core
- Best of Type-A and Type-B



**32 cores/node, 16 cores/socket
up to 8 nodes (256 cores)**



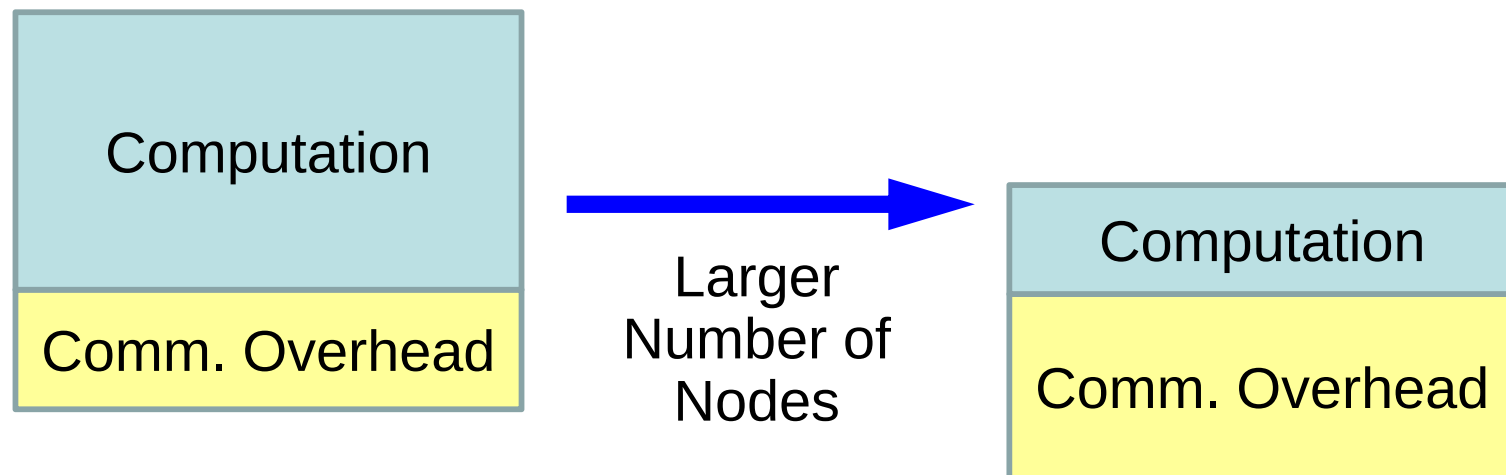
**36 cores/node, 18 cores/socket
up to 8 nodes (288 cores)**

Performance is lower than ideal one

- Time for MPI communication
 - Time for sending data
 - Communication bandwidth between nodes
 - Time is proportional to size of sending/receiving buffers
- Time for starting MPI
 - latency
 - does not depend on size of buffers
 - depends on number of calling, increases according to process #
 - $O(10^0)$ - $O(10^1)$ μsec .
- Synchronization of MPI
 - Increases according to number of processes

Performance is lower than ideal one (cont.)

- If computation time is relatively small (N is small in S1-3), these effects are not negligible.
 - If the size of messages is small, effect of “latency” is significant.
 - Granularity (粒度): Problem Size/PE



Shell Scripts

```
#!/bin/sh
#PBS -q u-lecture8
#PBS -N test
#PBS -l select=8:mpiprocs=32
#PBS -Wgroup_list=gt18
#PBS -l walltime=00:05:00
#PBS -e err
#PBS -o test.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

export I_MPI_PIN_DOMAIN=socket
export I_MPI_PERHOST=32

mpirun ./impimap.sh ./a.out
```

go.sh:

```
#!/bin/sh
#PBS -q u-lecture8
#PBS -N test
#PBS -l select=8:mpiprocs=32
#PBS -Wgroup_list=gt18
#PBS -l walltime=00:05:00
#PBS -e err
#PBS -o test.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

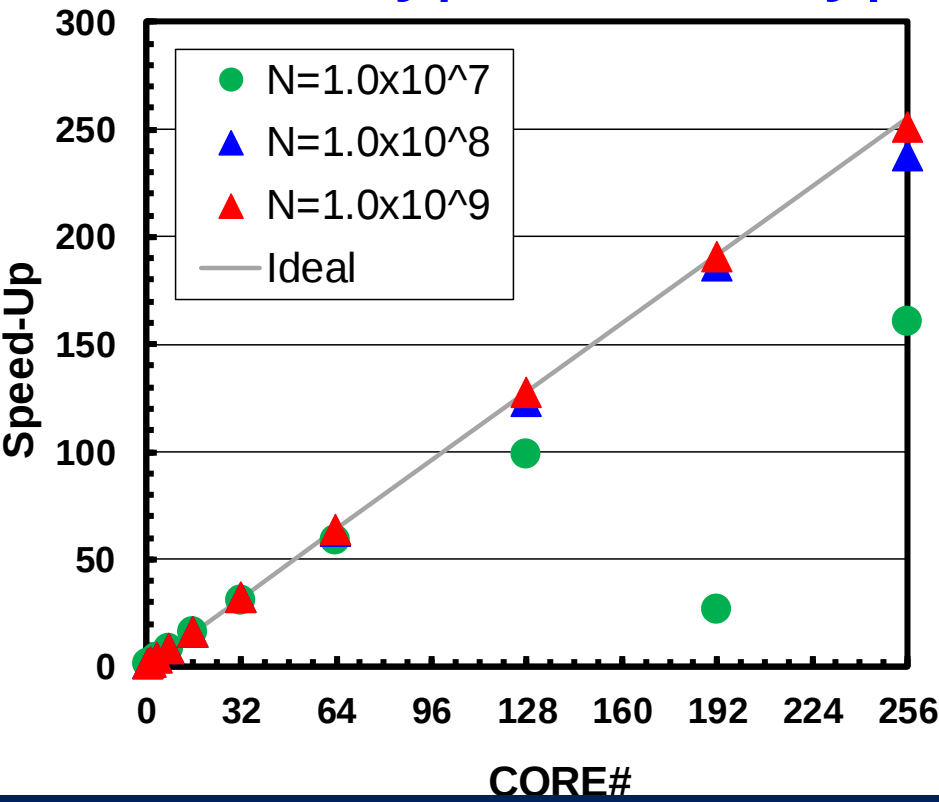
export I_MPI_PIN_PROCESSOR_LIST=0-15,18-33

mpirun ./impimap.sh ./a.out
```

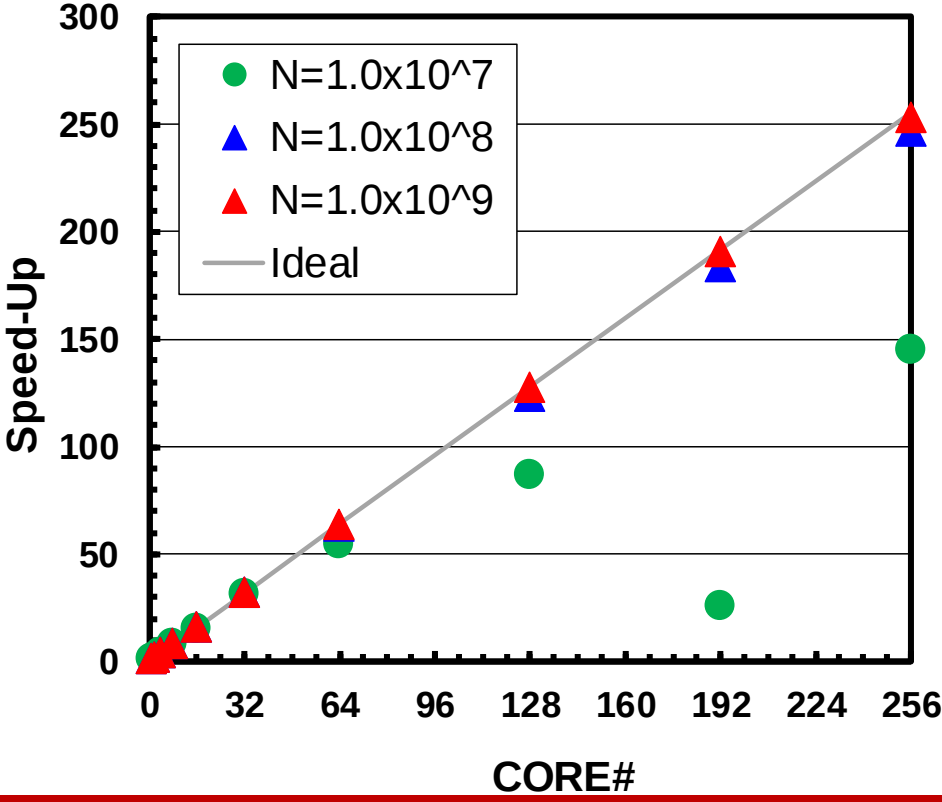
a32.sh: Performance is mostly same, but this one is more stable (small fluctuation)

S1-3: Performance on RB-U (4/4)

- ◆ : $N=10^7$, ● : 10^8 , ▲ : 10^9 , — : Ideal
- Based on results (sec.) using a single core
- Best of Type-A and Type-B



```
export I_MPI_PIN_DOMAIN=socket
export I_MPI_PERHOST=32
```



```
export I_MPI_PIN_PROCESSOR_LIST=
0-15,18-33
```