

課題S1解説

2007年6月6日

中島研吾

並列計算プログラミング(616-2057)・先端計算機演習I(616-4009)

課題S1 (1/2)

- 内容

- 「<\$S1>/a1.0~a1.3」, 「<\$S1>/a2.0~a2.3」から局所ベクトル情報を読み込み, 全体ベクトルのノルム($\|x\|$)を求めるプログラムを作成する(S1-1).
 - <\$S1>file.f, <\$S1>file2.fをそれぞれ参考にする。
- 「<\$S1>/a2.0~a2.3」から局所ベクトル情報を読み込み, 「全体ベクトル」情報を各プロセッサに生成するプログラムを作成する。MPI_ALLGATHERVを使用する(S1-2)。

課題S1 (2/2)

- 内容(続き)

- 下記の数値積分の結果を台形公式によって求めるプログラムを作成する。MPI_REDUCE, MPI_BCAST等を使用して並列化を実施し、プロセッサ数を変化させた場合の計算時間を測定する([S1-3](#))。

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

ファイルコピー

```
>$ cd <$07S> 各自作成したディレクトリ
```

FORTRAN

```
>$ cp /home/nakajima/class/2007summer/F/s1r-f.tar .  
>$ tar xvf s1r-f.tar
```

C

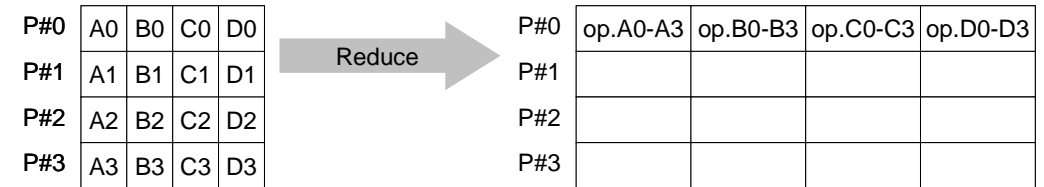
```
>$ cp /home/nakajima/class/2007summer/C/s1r-c.tar .  
>$ tar xvf s1r-c.tar
```

直下に「/s1-ref」というディレクトリができている。
<\$07S>/s1-refを<\$S1r>と呼ぶ。

S1-1: 局所ベクトル読み込み, ノルム計算

- 「 $\langle \$S1 \rangle / a1.0 \sim a1.3$ 」, 「 $\langle \$S1 \rangle / a2.0 \sim a2.3$ 」から局所ベクトル情報を読み込み, 全体ベクトルのノルム ($\|x\|$) を求めるプログラムを作成する (S1-1)。
- MPI_ALLREDUCE (またはMPI_REDUCE) の利用
- ワンポイントアドバイス
 - 変数の中身を逐一確認しよう!

MPI_REDUCE



- コミュニケーター「comm」内の、各プロセスの送信バッファ「sendbuf」について、演算「op」を実施し、その結果を1つの受信プロセス「root」の受信バッファ「recvbuf」に格納する。
 - 総和, 積, 最大, 最小 他

- `call MPI_REDUCE`

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

- `sendbuf` 任意 I 送信バッファの先頭アドレス,
- `recvbuf` 任意 O 受信バッファの先頭アドレス,
タイプは「datatype」により決定
- `count` 整数 I メッセージのサイズ
- `datatype` 整数 I メッセージのデータタイプ
FORTRAN MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_CHARACTER etc.
C MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_CHAR etc
- `op` 整数 I 計算の種類
MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND etc
ユーザーによる定義も可能: MPI_OP_CREATE
- `root` 整数 I 受信元プロセスのID(ランク)
- `comm` 整数 I コミュニケータを指定する
- `ierr` 整数 O 完了コード

送信バッファと受信バッファ

- MPIでは「送信バッファ」, 「受信バッファ」という変数がしばしば登場する。
- 送信バッファと受信バッファは必ずしも異なった名称の配列である必要はないが, 必ずアドレスが異なっていなければならない。

MPI_Reduce/Allreduceの“op”

```
call MPI_REDUCE
```

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

- MPI_MAX, MPI_MIN 最大値, 最小値
- MPI_SUM, MPI_PROD 総和, 積
- MPI_LAND 論理AND

```
real(kind=8):: X0, X1
```

```
call MPI_REDUCE
```

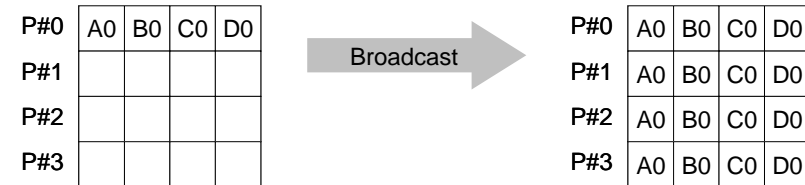
```
(X0, X1, 1, MPI_DOUBLE_PRECISION, MPI_MAX, 0, <comm>, ierr)
```

```
real(kind=8):: X0(4), XMAX(4)
```

```
call MPI_REDUCE
```

```
(X0, XMAX, 4, MPI_DOUBLE_PRECISION, MPI_MAX, 0, <comm>, ierr)
```


MPI_BCAST



- コミュニケータ「comm」内の一つの送信元プロセス「root」のバッファ「buffer」から、その他全てのプロセスのバッファ「buffer」にメッセージを送信。
- **call MPI_BCAST (buffer, count, datatype, root, comm, ierr)**
 - **buffer** 任意 I/O バッファの先頭アドレス,
タイプは「datatype」により決定
 - **count** 整数 I メッセージのサイズ
 - **datatype** 整数 I メッセージのデータタイプ
 FORTRAN MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISION, MPI_CHARACTER etc.
 C MPI_INT, MPI_FLOAT, MPI_DOUBLE, MPI_CHAR etc.
 - **root** 整数 I 送信元プロセスのID(ランク)
 - **comm** 整数 I コミュニケータを指定する
 - **ierr** 整数 O 完了コード

S1-1: 局所ベクトル読み込み, ノルム計算

均一長さベクトルの場合 (a1.*): s1-1-for_a1.f/c

```
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 )
call MPI_COMM_DUP     (MPI_COMM_WORLD, SOLVER_COMM, 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
(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
```

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

中身を書き出して見よう

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

S1-1: 局所ベクトル読み込み, ノルム計算

不均一長さベクトルの場合 (a2.*): s1-1-for_a2.f/c

```

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 )
call MPI_COMM_DUP  (MPI_COMM_WORLD, SOLVER_COMM, 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

```

中身を書き出して見よう

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

実行(課題S1-1)

FORTRAN

```
$ cd <$S1r>  
$ mpif90 -O s1-1-for_a1.f  
$ mpirun -np 4 a.out
```

```
$ mpif90 -O s1-1-for_a2.f  
$ mpirun -np 4 a.out
```

C

```
$ cd <$S1r>  
$ mpicc -O3 s1-1-for_a1.c  
$ mpirun -np 4 a.out
```

```
$ mpicc -O3 s1-1-for_a2.c  
$ mpirun -np 4 a.out
```

S1-1: 局所ベクトル読み込み, ノルム計算 計算結果

予め求めておいた答え

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

```
$> ./chk-a1
```

```
$> ./chk-a2
```

```
"a1x.all", "a2x.all"に全体データが入っています  
"dot-a1.f", "dot-a2.f"にソースコード
```

計算結果

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

S1-1: 局所ベクトル読み込み, ノルム計算

SENDBUFとRECVBUFを同じにしたら...

正

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

誤

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

以下のランタイムエラーが出る:

```
0 - MPI_ALLREDUCE : Invalid buffer pointer: Arguments must specify  
different buffers (no aliasing)
```

```
[0] Aborting program !
```

```
[0] Aborting program!
```

```
p0_1654: p4_error: : 8641
```

```
q/usr/local/mpich-1.2.5/bin/mpirun: line 1: 1654 Broken pipe  
/home/nakajima/class/wk/2004summer/C2/a.out -p4pg /home/na
```

S1-1: 局所ベクトル読み込み, ノルム計算

SENDBUFとRECVBUFを同じにしたら...

正

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

誤

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

正

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

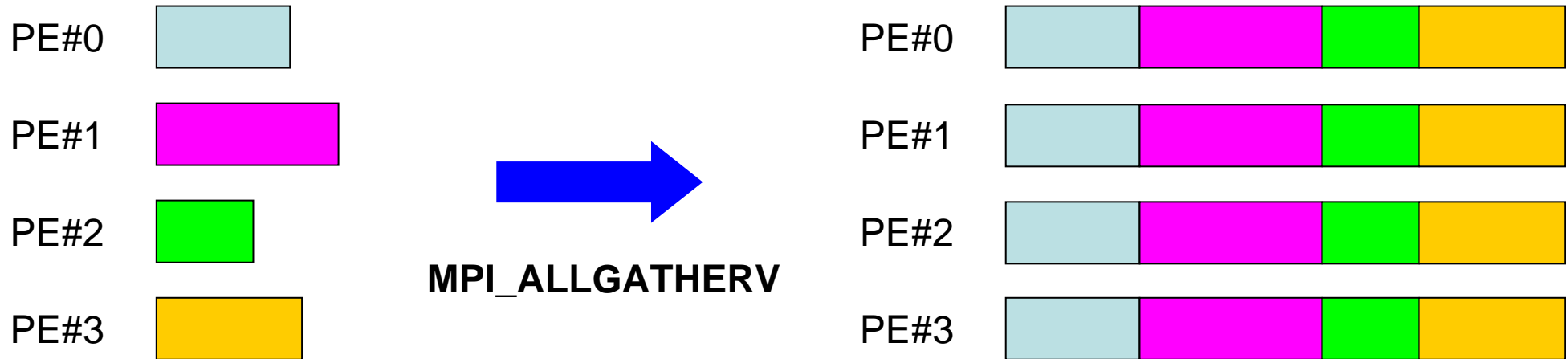
これバッファが重なっていないのでOK

S1-2: 局所ベクトルから全体ベクトル生成

- 「$S1$/a2.0~a2.3」から局所ベクトル情報を読み込み、「全体ベクトル」情報を各プロセッサに生成するプログラムを作成する。MPI_ALLGATHERVを使用する。

S1-2: 局所ベクトルから全体ベクトル生成

ALLGATHERVを使う場合 (1/5)



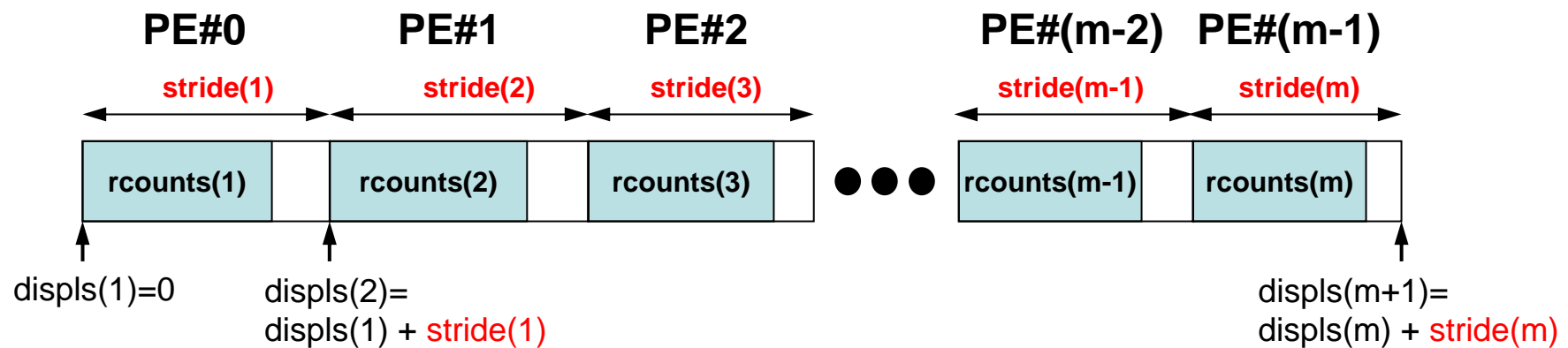
MPI_ALLGATHERV

- MPI_ALLGATHER の可変長さベクトル版
 - 「局所データ」から「全体データ」を生成する

- `call MPI_ALLGATHERV (sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, comm, ierr)`
 - `sendbuf` 任意 I 送信バッファの先頭アドレス,
 - `scount` 整数 I 送信メッセージのサイズ
 - `sendtype` 整数 I 送信メッセージのデータタイプ
 - `recvbuf` 任意 O 受信バッファの先頭アドレス,
 - **`rcounts` 整数 I 受信メッセージのサイズ(配列:サイズ=PETOT)**
 - **`displs` 整数 I 受信メッセージのインデックス(配列:サイズ=PETOT+1)**
 - `recvtype` 整数 I 受信メッセージのデータタイプ
 - `comm` 整数 I コミュニケータを指定する
 - `ierr` 整数 O 完了コード

MPI_ALLGATHERV (続き)

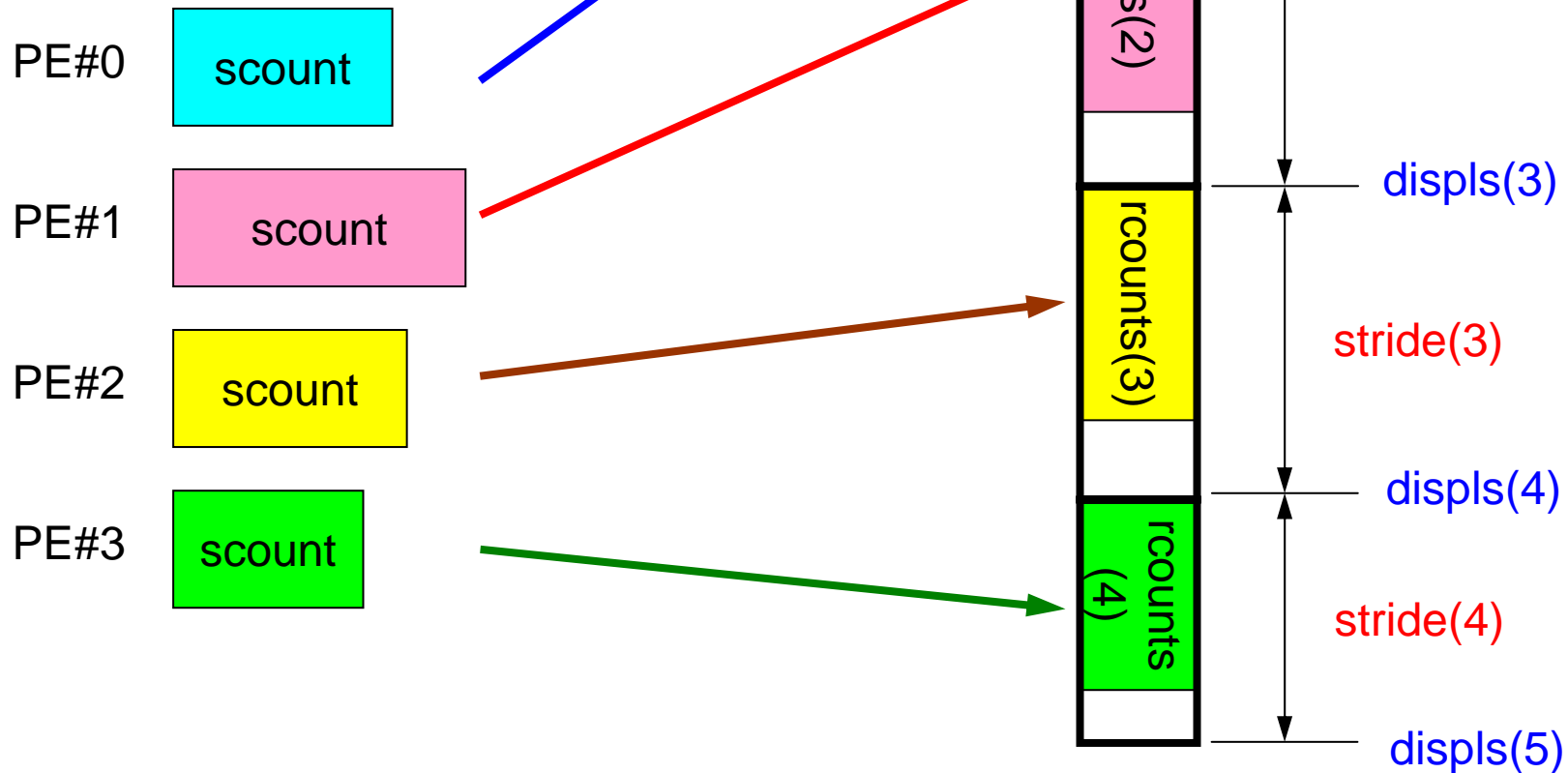
- call `MPI_ALLGATHERV (sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, comm, ierr)`
 - `rcounts` 整数 I 受信メッセージのサイズ (配列: サイズ = PETOT)
 - `displs` 整数 I 受信メッセージのインデックス (配列: サイズ = PETOT+1)
 - この2つの配列は、最終的に生成される「全体データ」のサイズに関する配列であるため、各プロセスで配列の全ての値が必要になる:
 - もちろん各プロセスで共通の値を持つ必要がある。
 - 通常は $\text{stride}(i) = \text{rcounts}(i)$



MPI_ALLGATHERV

でやっていること

局所データから全体データを生成する



S1-2

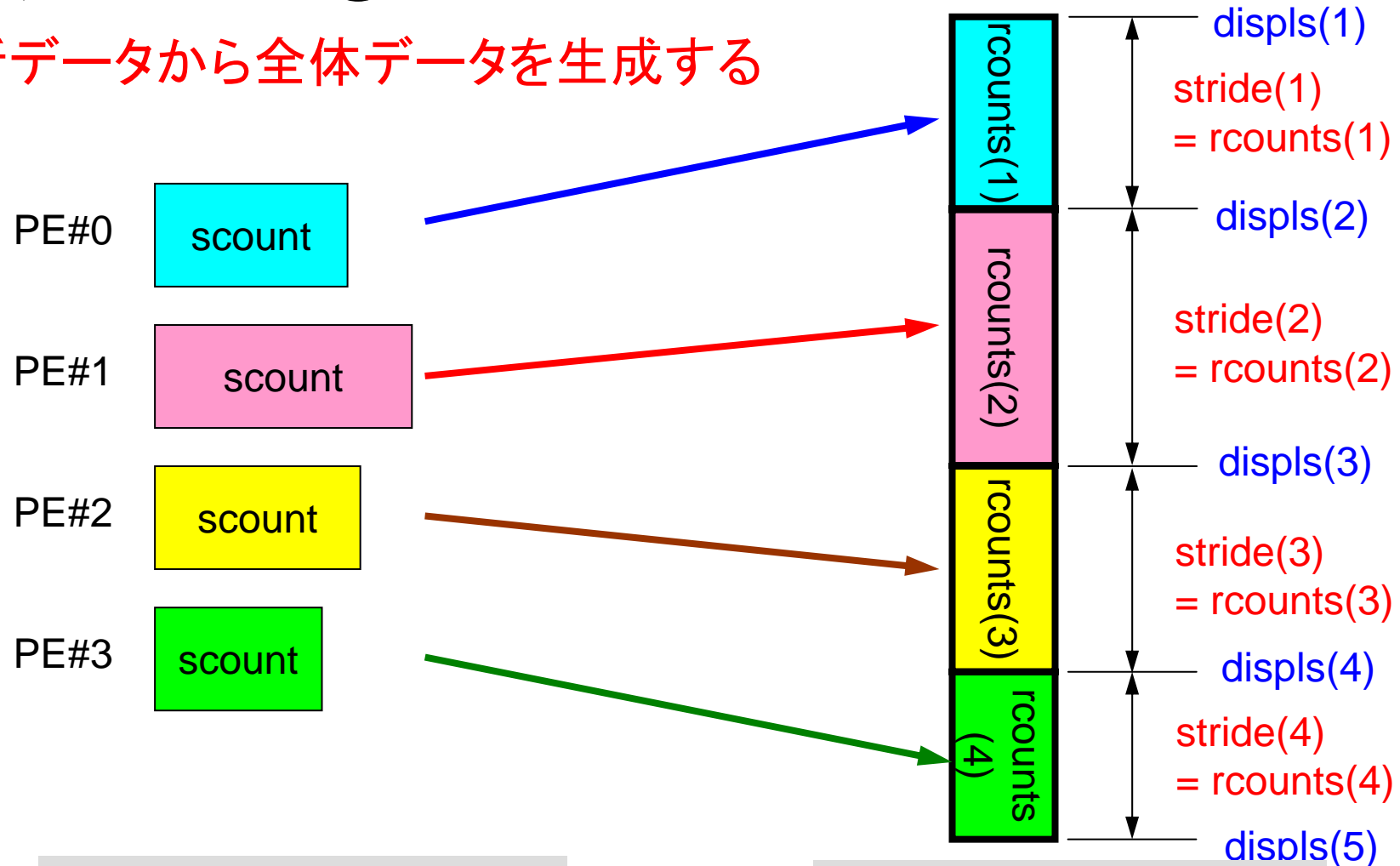
局所データ: sendbuf

全体データ: recvbuf

MPI_ALLGATHERV

でやっていること

局所データから全体データを生成する



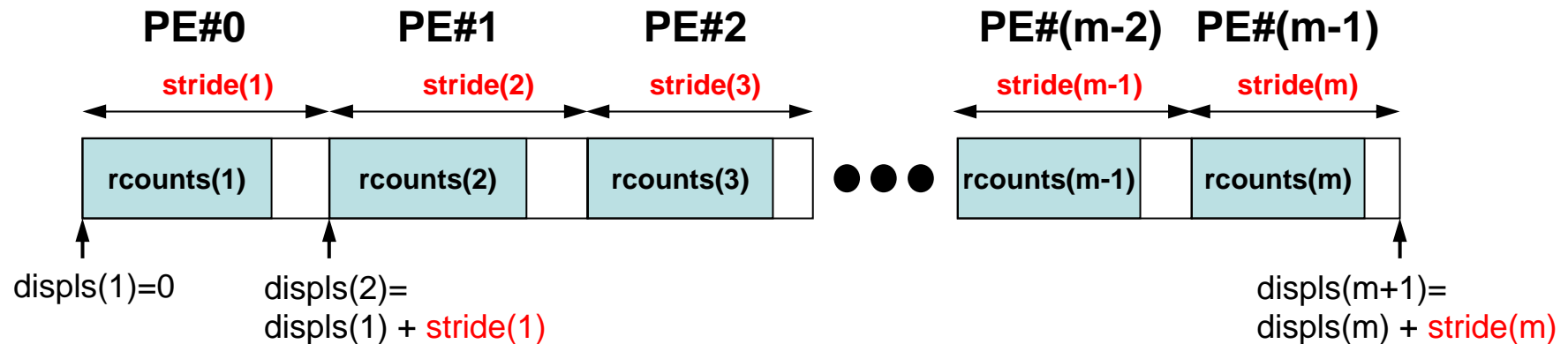
S1-2

局所データ: sendbuf

全体データ: recvbuf

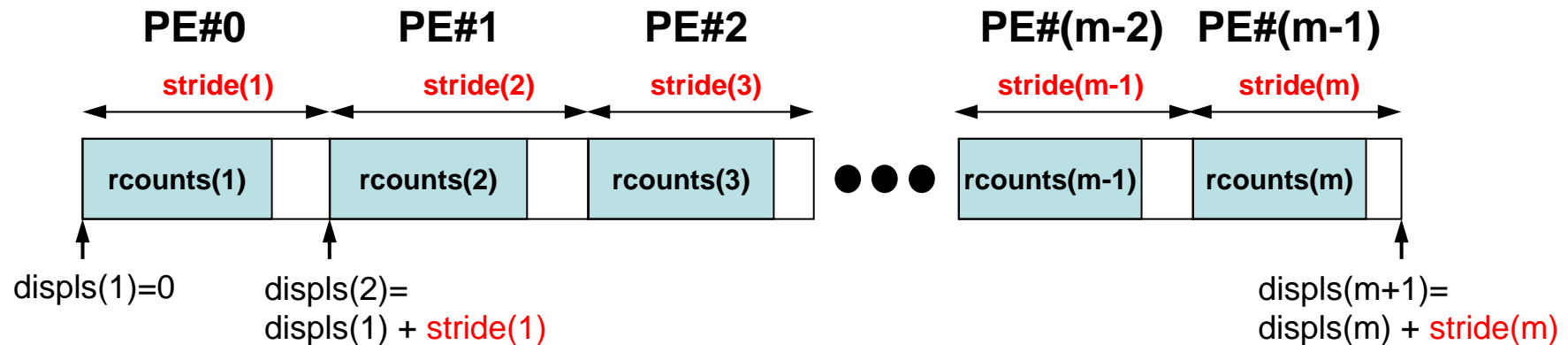
MPI_ALLGATHERV詳細(1/2)

- call `MPI_ALLGATHERV (sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, comm, ierr)`
 - `rcounts` 整数 I 受信メッセージのサイズ(配列:サイズ=`PETOT`)
 - `displs` 整数 I 受信メッセージのインデックス(配列:サイズ=`PETOT+1`)
- `rcounts`
 - 各PEにおけるメッセージサイズ:局所データのサイズ
- `displs`
 - 各局所データの全体データにおけるインデックス
 - `displs(PETOT+1)`が全体データのサイズ



MPI_ALLGATHERV詳細(2/2)

- `rcounts`と`displs`は各プロセスで共通の値が必要
 - 各プロセスのベクトルの大きさ N を`allgather`して, `rcounts`に相当するベクトルを作る。
 - `rcounts`から各プロセスにおいて`displs`を作る(同じものができる)。
 - $\text{stride}(i) = \text{rcounts}(i)$ とする
 - `rcounts`の和にしたがって`recvbuf`の記憶領域を確保する。



MPI_ALLGATHERV使用準備

例題: <\$S1>/agv.f, <\$S1>/agv.c

- “a2.0”~”a2.3”から, 全体ベクトルを生成する。
- 各ファイルのベクトルのサイズが, 8,5,7,3であるから, 長さ23(=8+5+7+3)のベクトルができることになる。

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

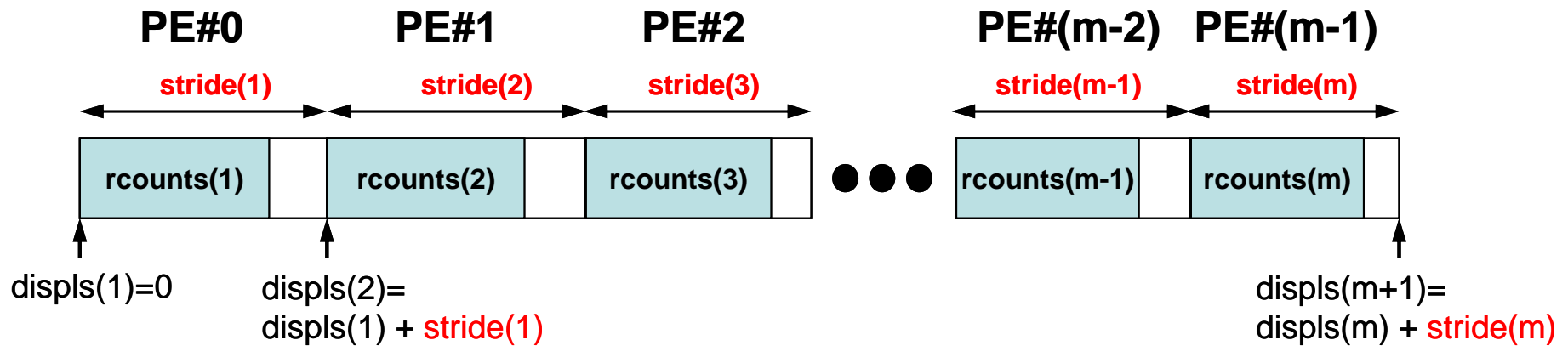
PE#2

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: 局所⇒全体ベクトル生成: 手順



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

- 局所ベクトル情報を読み込む
- 「rcounts」, 「displs」を作成する
- 「recvbuf」を準備する
- ALLGATHERV

S1-2: 局所⇒全体ベクトル生成 (1/2)

s1-2.f/c

```

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 )
call MPI_COMM_DUP  (MPI_COMM_WORLD, SOLVER_COMM, 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

```

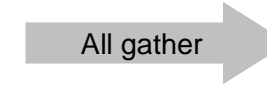
中身を書き出して見よう

各PEにおけるベクトル長さの情報が「COUNT」に入る(「rcounts」)

中身を書き出して見よう

MPI_ALLGATHER

P#0	A0			
P#1	B0			
P#2	C0			
P#3	D0			



P#0	A0	B0	C0	D0
P#1	A0	B0	C0	D0
P#2	A0	B0	C0	D0
P#3	A0	B0	C0	D0

- MPI_GATHER+MPI_BCAST
- call MPI_ALLGATHER (sendbuf, scount, sendtype, recvbuf, rcount, recvtype, comm, ierr)
 - sendbuf 任意 I 送信バッファの先頭アドレス,
 - scount 整数 I 送信メッセージのサイズ
 - sendtype 整数 I 送信メッセージのデータタイプ
 - recvbuf 任意 O 受信バッファの先頭アドレス,
 - rcount 整数 I 受信メッセージのサイズ
 - recvtype 整数 I 受信メッセージのデータタイプ
 - comm 整数 I コミュニケータを指定する
 - ierr 整数 O 完了コード

S1-2: 局所⇒全体ベクトル生成 (2/2)

s1-2.f/c

```

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)

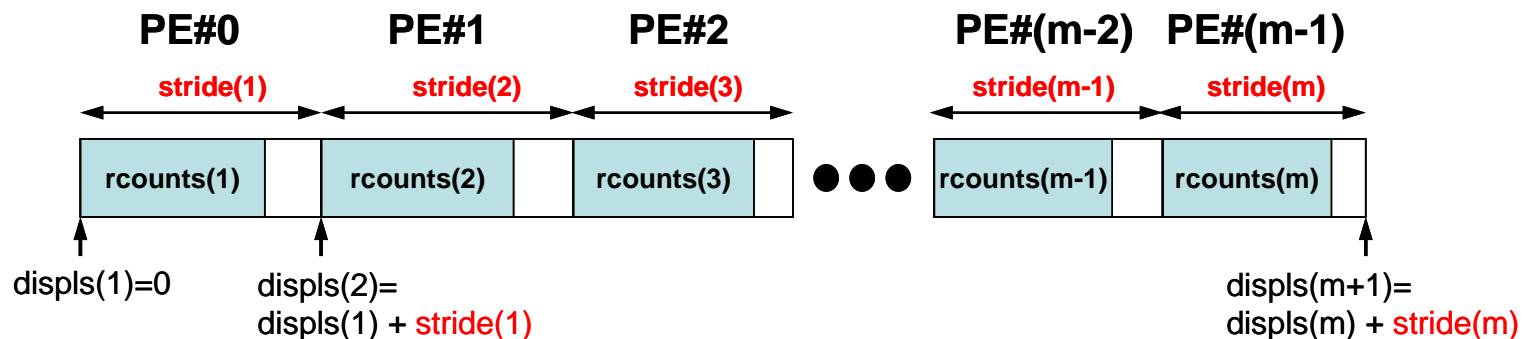
do i= 1, COUNTindex(PETOT+1)
  write (*,'(2i8,f10.0)') my_rank, i, VECg(i)
enddo

call MPI_FINALIZE (ierr)

stop
end

```

「displs」に相当するものを生成。



S1-2

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

S1-2: 局所⇒全体ベクトル生成 (2/2)

s1-2.f/c

```

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

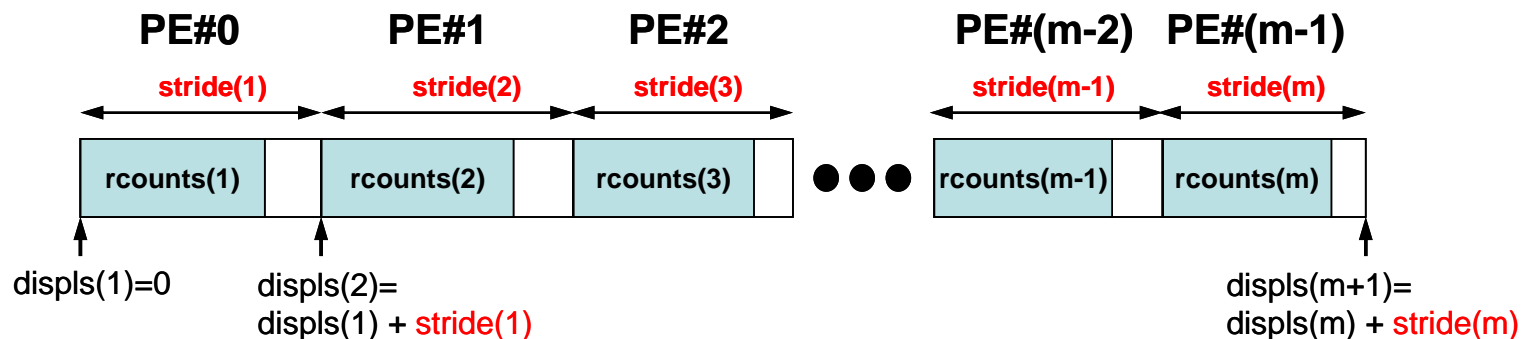
call MPI_FINALIZE (ierr)

stop
end

```

「recvbuf」のサイズ

&
&
&



S1-2

size(recvbuf) = displs(PETOT+1) = sum(stride)

S1-2: 局所⇒全体ベクトル生成 (2/2)

s1-2.f/c

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

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

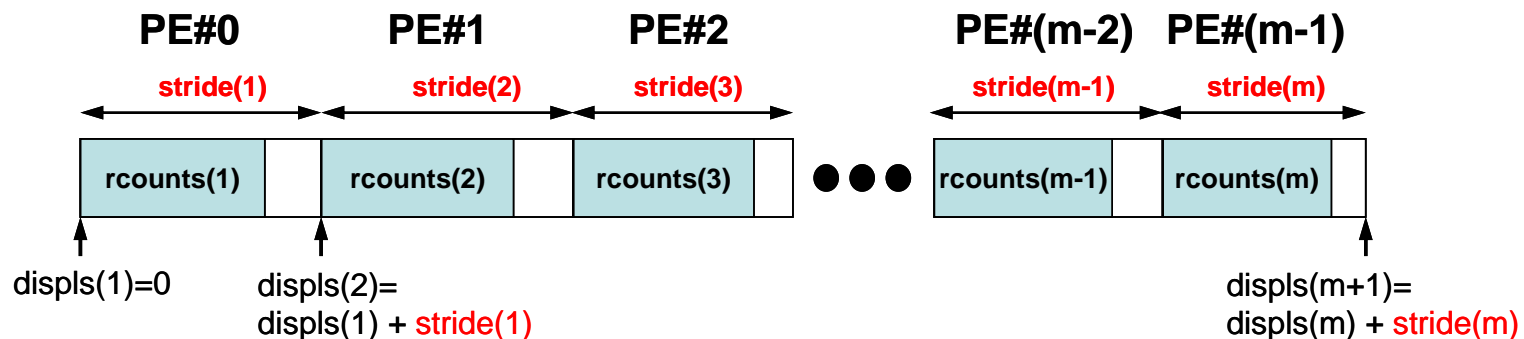
```
call MPI_allGATHERv                                &
&  ( VEC , N, MPI_DOUBLE_PRECISION,                &
&    VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION, &
&    MPI_COMM_WORLD, ierr)                        &
```

```
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)
```



実行(課題S1-2)

FORTRAN

```
$ mpif90 -O3 s1-2.f  
$ mpirun -np 4 a.out
```

C

```
$ mpicc -O3 s1-2.c  
$ mpirun -np 4 a.out
```

S1-2: 結果

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.

S1-2: 局所ベクトルから全体ベクトル生成

こうやってもできる(<\$S1r>/s1-2b.f)

```

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

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

if (my_rank.eq.0) then
  call MPI_allGATHERv
  & ( VEC, 8, MPI_DOUBLE_PRECISION,
  & VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION,
  & MPI_COMM_WORLD, ierr)
else if
  & (my_rank.eq.1) then
  call MPI_allGATHERv
  & ( VEC, 5, MPI_DOUBLE_PRECISION,
  & VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION,
  & MPI_COMM_WORLD, ierr)
else if
  & (my_rank.eq.2) then
  call MPI_allGATHERv
  & ( VEC, 7, MPI_DOUBLE_PRECISION,
  & VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION,
  & MPI_COMM_WORLD, ierr)
else
  call MPI_allGATHERv
  & ( VEC, 3, MPI_DOUBLE_PRECISION,
  & VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION,
  & MPI_COMM_WORLD, ierr)
endif

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

```

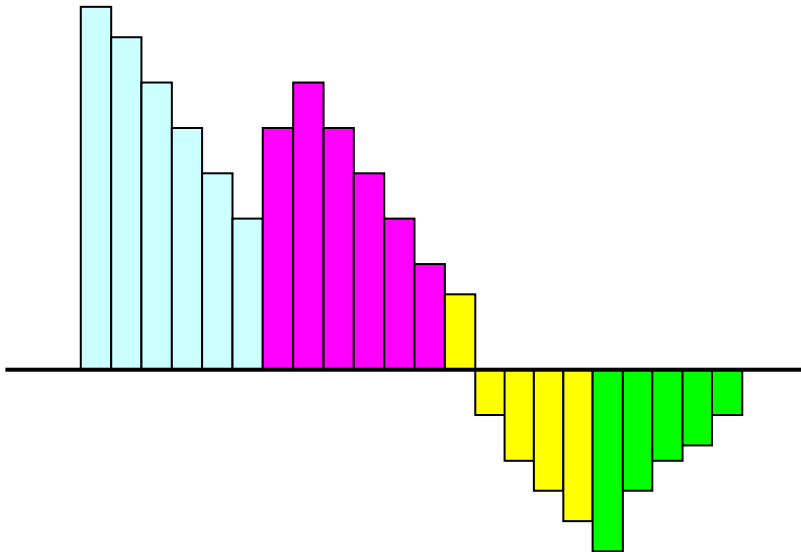
S1-3: 台形則による積分

- 下記の数値積分の結果を台形公式によって求めるプログラムを作成する。MPI_REDUCE, MPI_BCASTを使用して並列化を実施し, プロセッサ数を変化させた場合の計算時間を測定する。

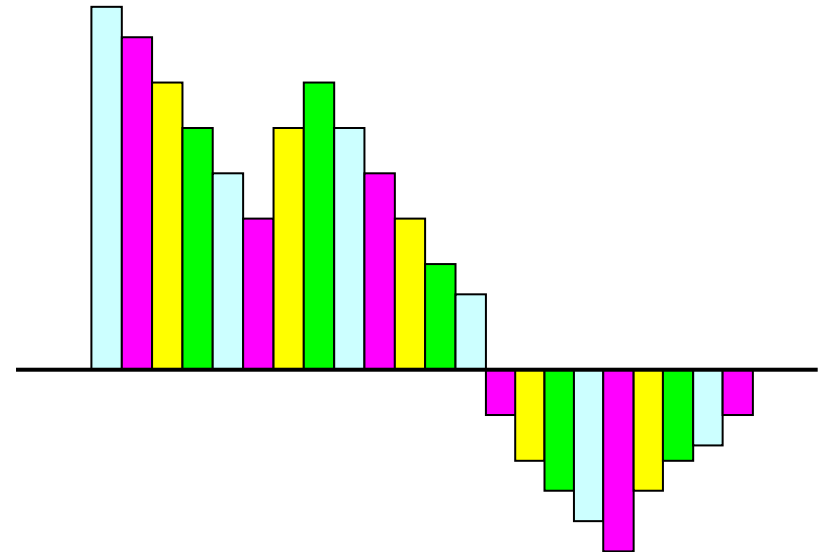
$$\int_0^1 \frac{4}{1+x^2} dx$$

S1-3: 台形則による積分 プロセッサへの配分の手法

タイプA



タイプB



$\frac{1}{2} \Delta x \left(f_1 + f_{N+1} + \sum_{i=2}^N 2f_i \right)$ を使うとすると必然的に「タイプA」となるが...

S1-3: 台形則による計算

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

```
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

if (my_rank+1.le.nr) then
  nnn= nnn + 1
endif
```

“input.dat”で分割数Nを指定
中身を書き出して見よう:N

S1-3: 台形則による計算

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

```
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
```

各PEにおける小領域数が「nnn」
(またはnnn+1)

各PEで INDEX(my_rank+1)
の中身を書き出して見よう:

S1-3: 台形則による計算

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

```

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()

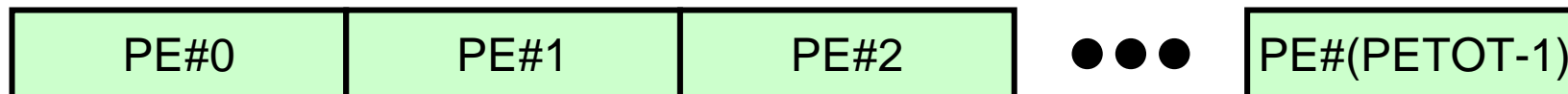
if (my_rank.eq.0) write (*,*) SUM, 4.d0*datan(1.d0), Etime-Stime

call MPI_FINALIZE (ierr)

stop
end

```

中身を書き出して見よう: index



INDEX(0)+1

INDEX(1)+1

INDEX(2)+1

INDEX(3)+1

INDEX(PETOT-1)+1

INDEX(PETOT)
=N

S1-3: 台形則による計算

TYPE-B : s1-3b.f/c

```

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)

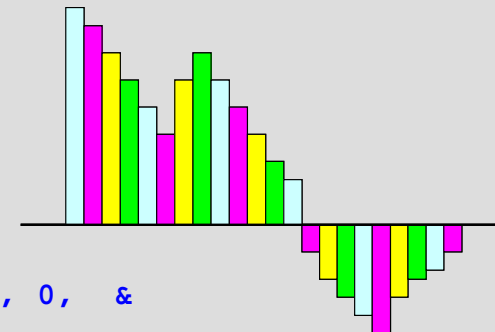
Stime= MPI_WTIME()
SUM0= 0.d0
do i= my_rank+1, N, PETOT
  X0= dfloat(i-1) * dx
  X1= dfloat(i  ) * dx
  F0= 4.d0/(1.d0+X0*X0)
  F1= 4.d0/(1.d0+X1*X1)
  SUM0= SUM0 + 0.50d0 * ( F0 + F1 ) * dx
enddo

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

```

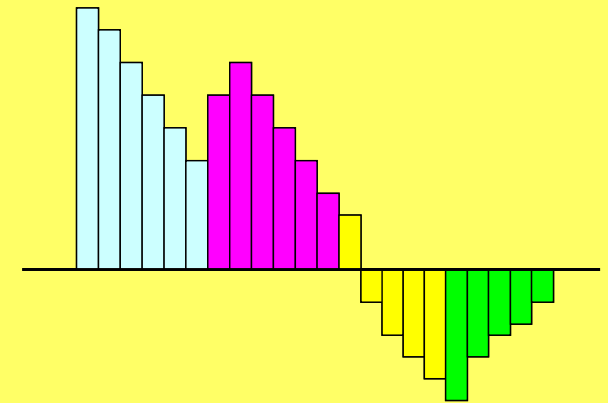


コンパイル(課題S1-3)

FORTRAN

```
$ mpif90 -O3 s1-3a.f  
$ mpif90 -O3 s1-3b.f
```

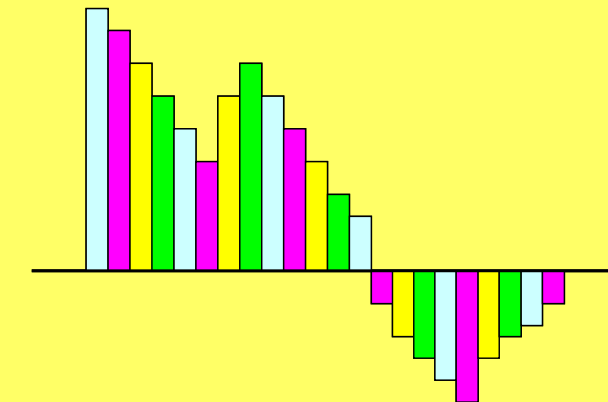
タイプA



C

```
$ mpicc -O3 s1-3a.c  
$ mpicc -O3 s1-3b.c
```

タイプB



実行(課題S1-3)

```
$ cat go.sh

#!/bin/sh
#PBS -N test                ケース名
#PBS -o test.lst           標準出力ファイル名(ケースごとに変わると良い)
#PBS -l nodes=2:ppn=2      使用プロセッサ数(この場合は2×2=4)
#PBS -e err                エラー出力ファイル名
cd $PBS_O_WORKDIR          必須
NPROCS=`wc -l < $PBS_NODEFILE` 必須
mpirun -v -machinefile $PBS_NODEFILE -np $NPROCS a.out
                           最後に実行ファイル名(この場合は"a.out")

$ qsub go.sh
$ qstat

$ cat test.lst
```

S1-3: 台形則による計算

TYPE-B, PETOT=4, N=102の場合

```
do i= my_rank+1, N, PETOT  
  (OPERATIONS)  
enddo
```

my_rank=0

```
do i= 1, 101, 4  
  (op)  
enddo
```

i= 1, 5, 9, 13, -, 93, 97, 101

my_rank=1

```
do i= 2, 102, 4  
  (op)  
enddo
```

i= 2, 6, 10, 14, -, 94, 98, 102

my_rank=2

```
do i= 3, 99, 4  
  (op)  
enddo
```

i= 3, 7, 11, 15, -, 95, 99

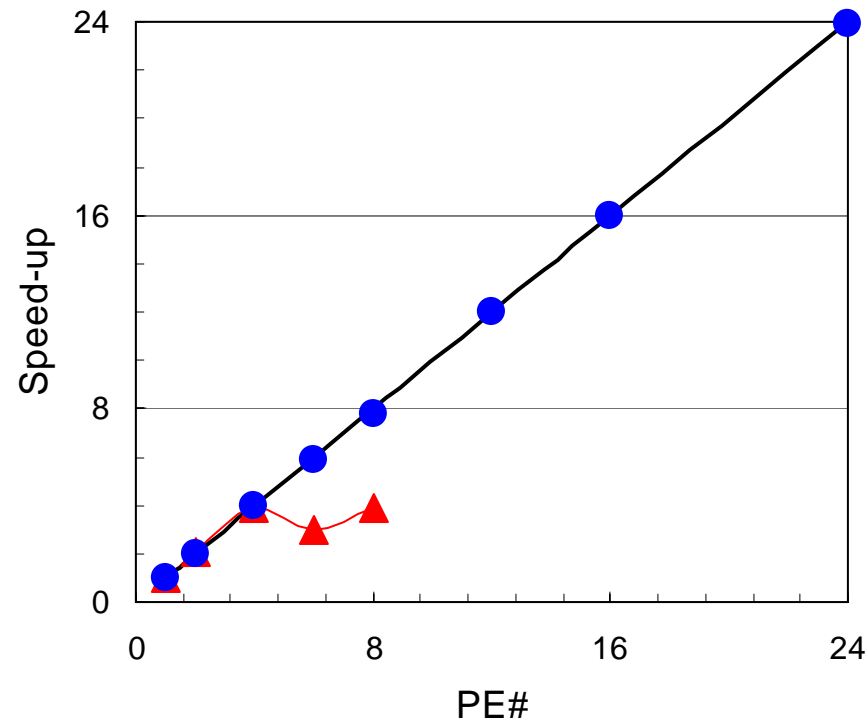
my_rank=3

```
do i= 4, 100, 4  
  (op)  
enddo
```

i= 4, 8, 12, 16, -, 96, 100

S1-3: cenjuにおける並列効果

- : $N=10^9$, ▲ : $N=10^7$
- : 理想値
- 1PEにおける計測結果(sec.)からそれぞれ算出
 - 2PE以上では各ノード2PE使用



理想値からのずれ

- MPI通信そのものに要する時間
 - データを送付している時間
 - ノード間においては通信バンド幅によって決まる
 - Gigabit Ethernetでは 1Gbit/sec.(理想値)
 - 通信時間は送受信バッファのサイズに比例
- MPIの立ち上がり時間
 - latency
 - 送受信バッファのサイズによらない
 - 呼び出し回数依存, プロセス数が増加すると増加する傾向
 - 通常, 数~数十 μ secのオーダー
- MPIの同期のための時間
 - プロセス数が増加すると増加する傾向

理想値からのずれ(続き)

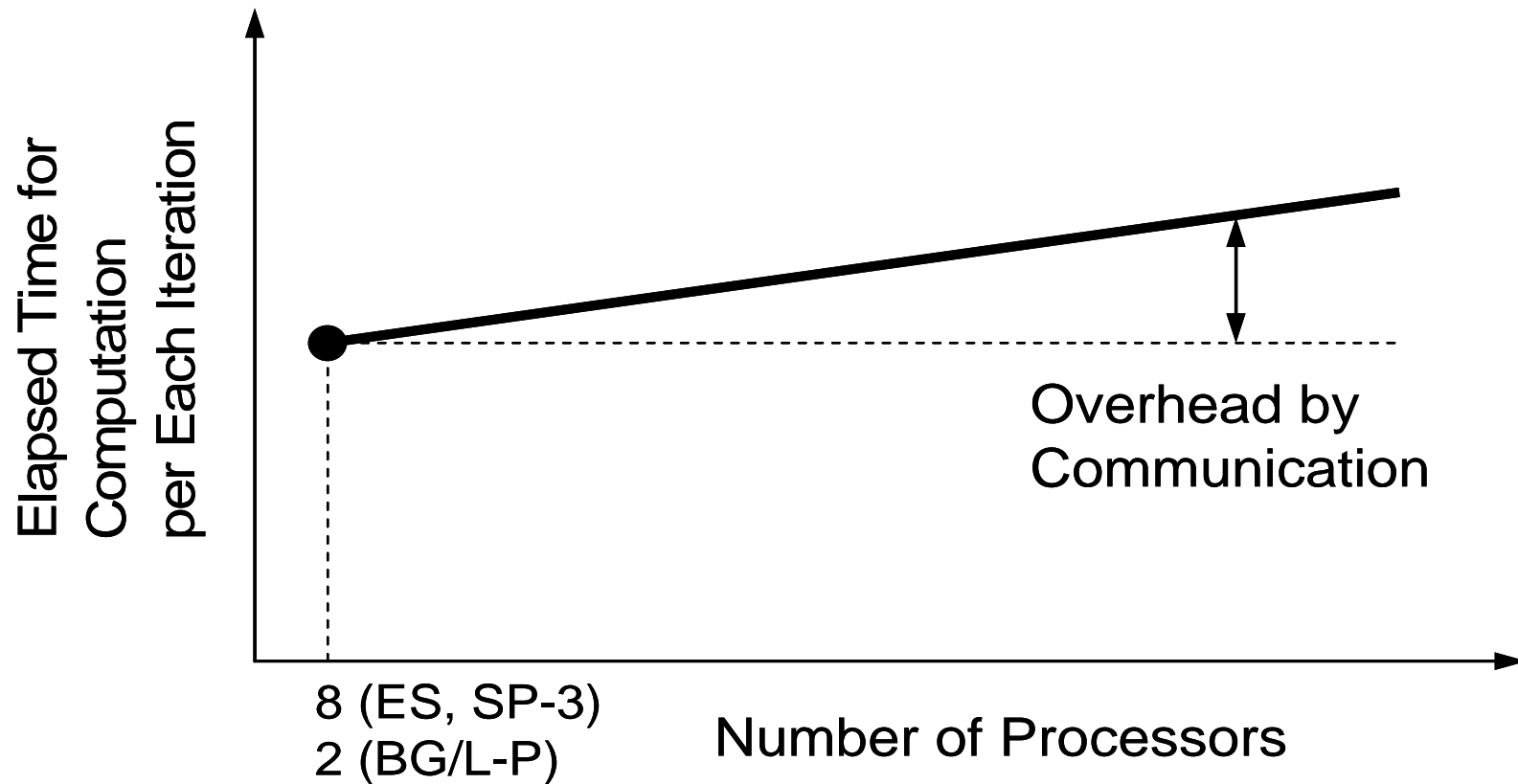
- 計算時間が小さい場合(S1-3ではNが小さい場合)はこれらの効果を見捨てるできない。
 - 特に, 送信メッセージ数が小さい場合は, 「Latency」が効く。

研究例

- Weak Scaling
 - プロセッサあたりの問題規模を固定
 - プロセッサ数を増加させる
 - 「計算性能」としては不変のはずであるが、普通はプロセッサ数を増やすと性能は悪化
- Nakajima, K. (2007), The Impact of Parallel Programming Models on the Linear Algebra Performance for Finite Element Simulations, Lecture Notes in Computer Science 4395, 334-348.
 - 並列有限要素法(特に latency の影響大)

Communication Overhead in Parallel FEM (Weak Scaling)

due to latency, (finite value of) bandwidth

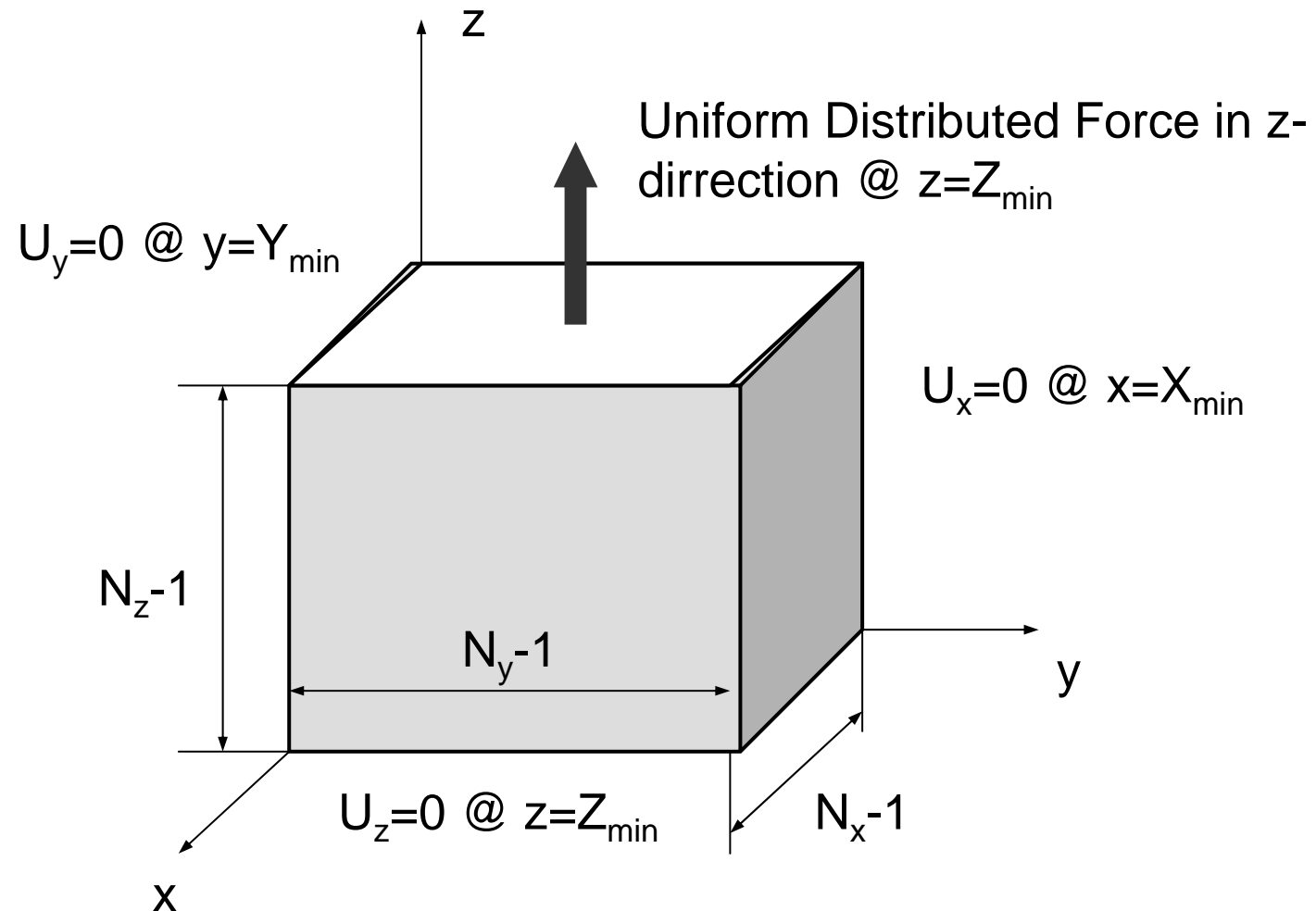


Platforms

	Earth Simulator	Hitachi SR8000 (U.Tokyo)	IBM-SP3 (LBNL)	IBM p5-595 (LBNL)	IBM BG/L-proto (Prototype)
PE#/node	8	8	16	8	2
Clock rate (MHz)	500	450	375	1,900	500
Peak Performance (GFLOPS/PE)	8.00	1.80	1.50	7.60	1.00 (w/singe FPU)
Memory Size (GB/node)	16	16	16~64	32	0.256
Peak Memory BW (GB/sec/node)	256	32	16	100	3.4
Network Topology	single stage crossbar	3D crossbar	Switch	Switch	3D Torus
Network BW (GB/sec/node)	12.3	1.6	1.0	32	1.32
MPI Latency (μsec)	5.6-7.7	6-20	16.3	3.0	6.0

**Only 8 of 16 PE's have been used for each node of IBM-SP3 and IBM p5-595.
Only Flat-MPI for IBM BG/L-proto.**

Simple 3D Cubic Model

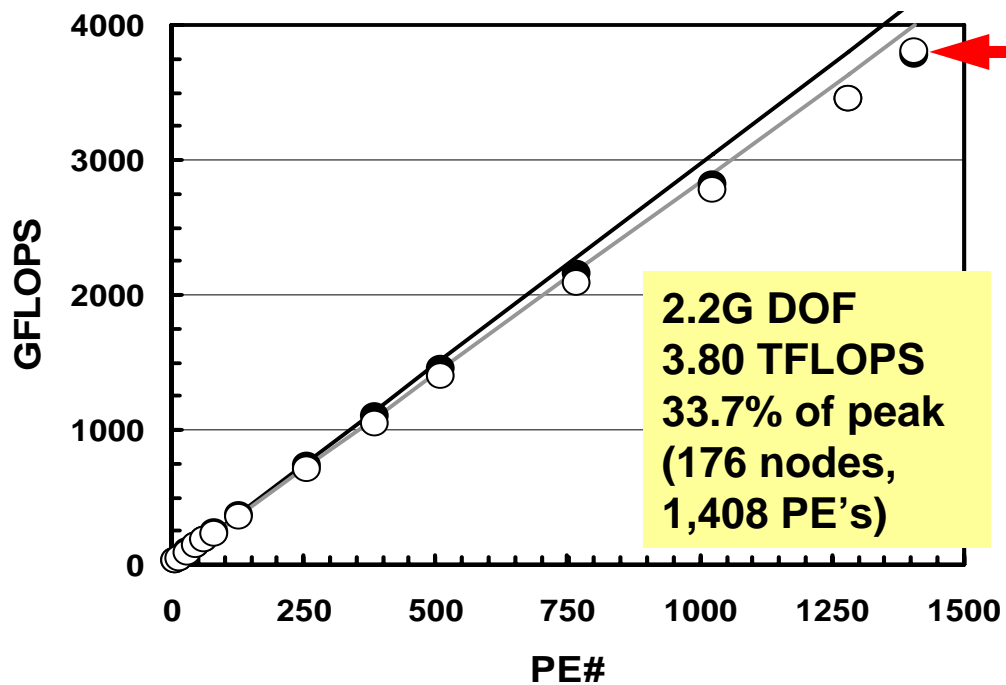


Weak Scaling: LARGE

- Flat-MPI DJDS
- Hybrid DJDS
- Flat-MPI(ideal)
- Hybrid (ideal)

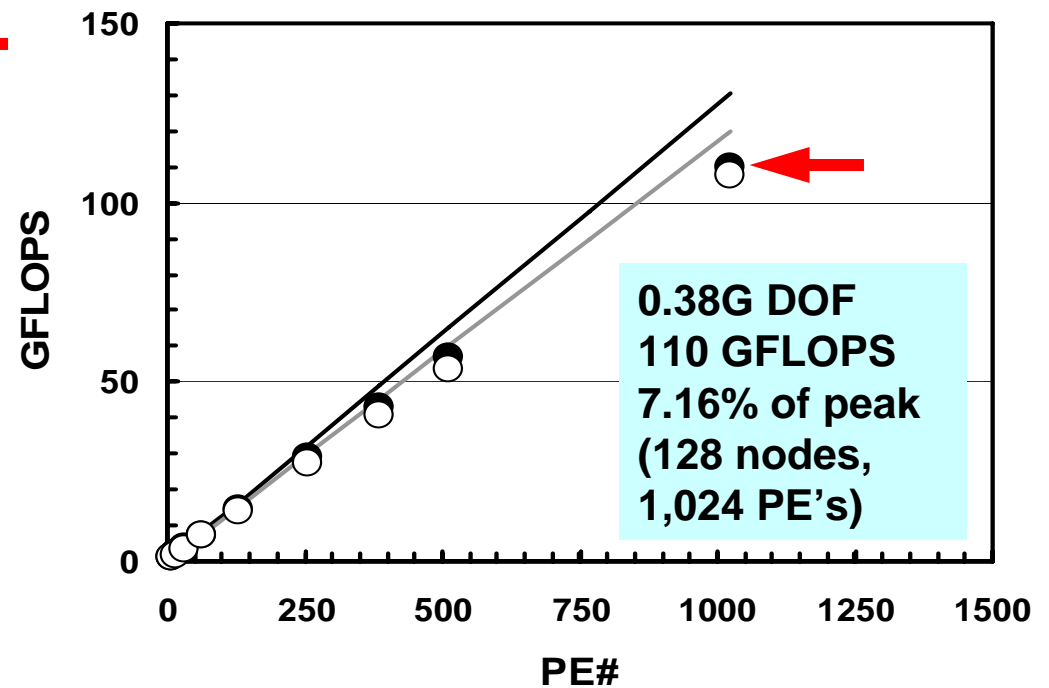
Earth Simulator

1,572,864 DOF/PE
(=3x128x64x64)



IBM SP-3 (Seaborg at LBNL)

375,000 DOF/PE
(=3x50³)

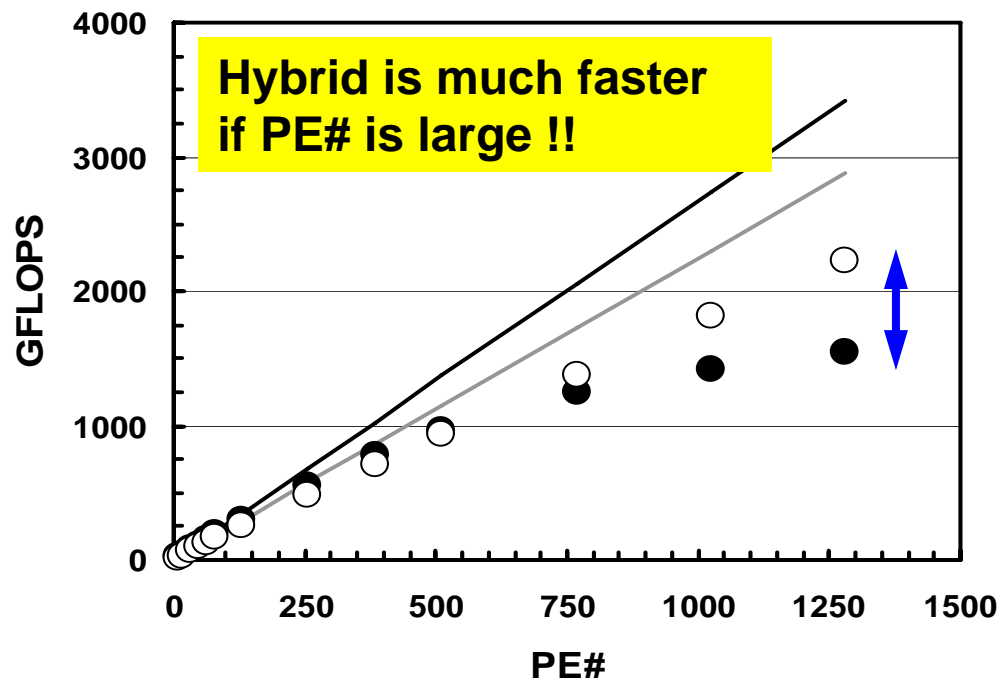


Weak Scaling: SMALL

- Flat-MPI DJDS
- Hybrid DJDS
- Flat-MPI(ideal)
- Hybrid (ideal)

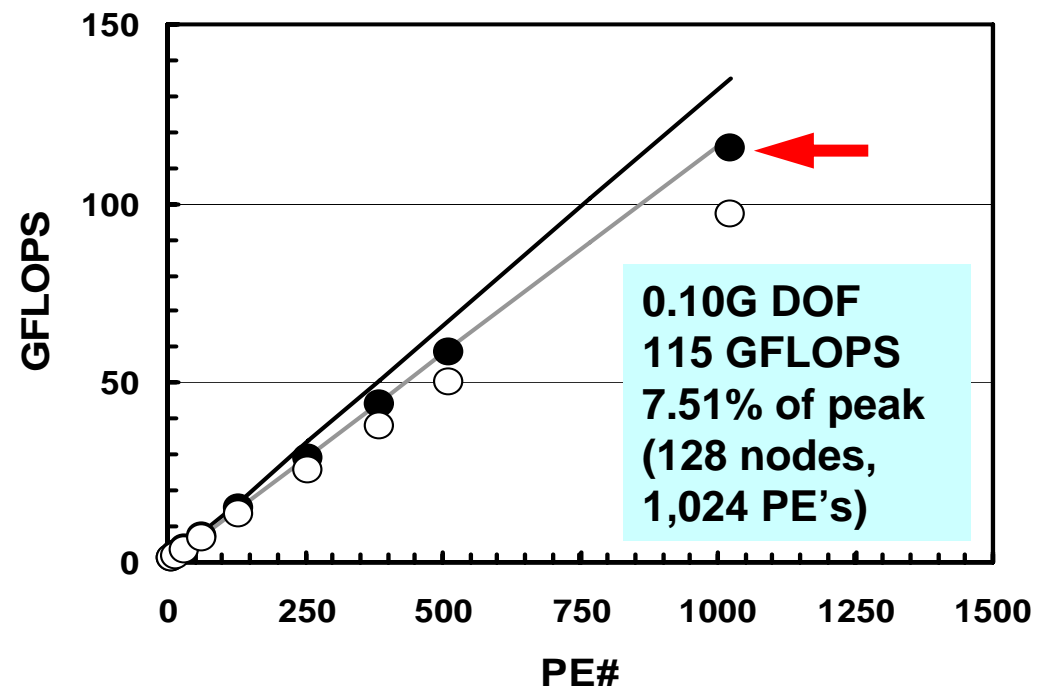
Earth Simulator

98,304 DOF/PE
(=3x32³)



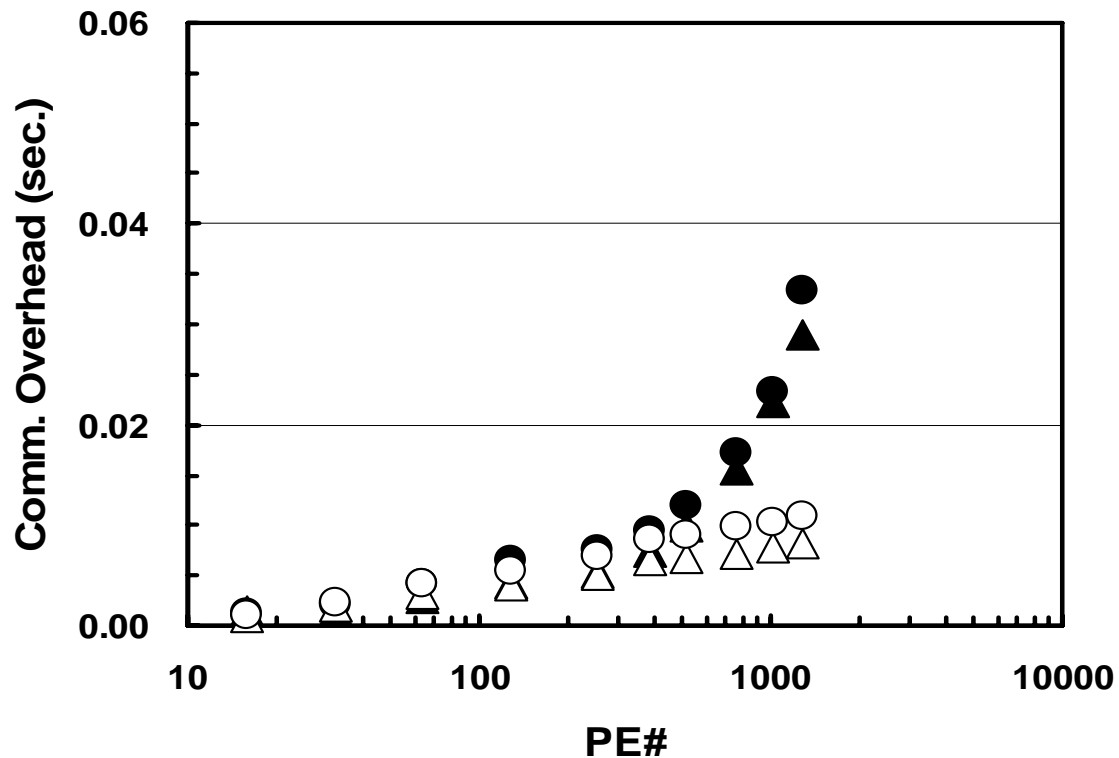
IBM SP-3 (Seaborg at LBNL)

98,304 DOF/PE
(=3x32³)



Communication Overhead

Weak Scaling: Earth Simulator

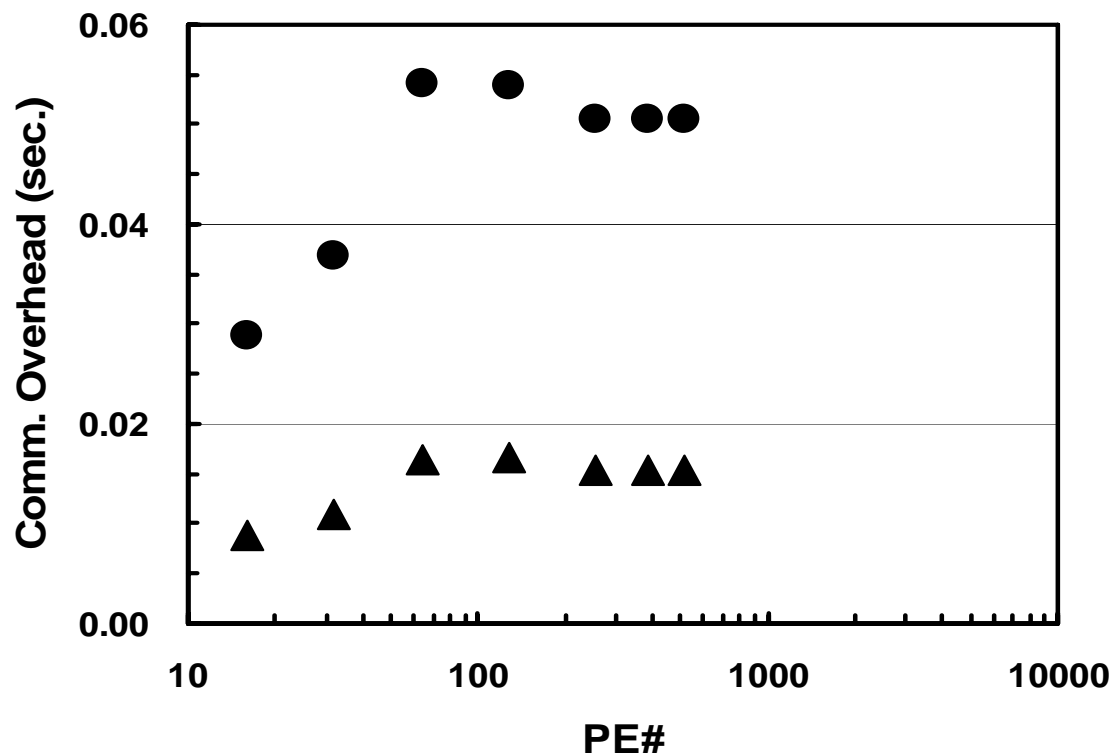


Effect of message size is small. Effect of latency is large.

Memory-copy is so fast.

Communication Overhead

Weak Scaling: IBM BG/L-PROTO



1 PE/node

**Effect of message size
is more significant.**

通信：メモリーコピー

実は以外にメモリの負担もかかる

```

do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= VAL(kk)
  enddo
enddo

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib )
  BUFlength_e= iE_e + 1 - iS_e
  iS_i= import_index(neib-1) + 1
  iE_i= import_index(neib )
  BUFlength_i= iE_i + 1 - iS_i

  call MPI_SENDRECV
&          (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
enddo

do neib= 1, NEIBPETOT
  do k= import_index(neib-1)+1, import_index(neib)
    kk= import_item(k)
    VAL(kk)= RECVbuf(k)
  enddo
enddo

```