計算科学演習A2
MPIを用いた並列計算（II）

神戸大学大学院システム情報学研究科
横川 三津夫
yokokawa@port.kobe-u.ac.jp

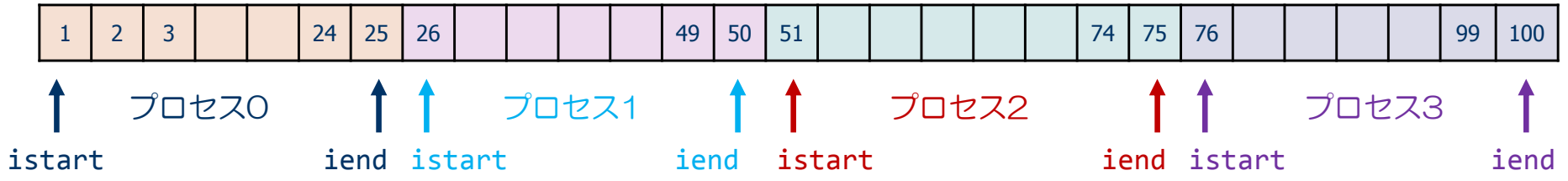
今週の講義の概要

1. 前回課題（M1-3）の解説
2. 計算時間の計測
3. 集団通信（mpi_allreduce関数）

演習M1-3 （提出課題１） 【再掲】

- 1 から 100 までの整数の和を求めるプログラムを，4並列で実行できるように修正し，4プロセスで実行せよ.
  - mpi_send，mpi_recv関数を使うこと.

- プログラム改良のポイント
  - 各プロセスの部分和を計算する範囲を，myrank をうまく使って求める.



  - myrank ≠ 0 以外のプロセスから，プロセス0（myrank=0）に部分和を送信する.
  - プロセス0（myrank=0）は，他の3つのプロセスから送られた部分和を受信し，全体の和を計算する.
  - 計算結果が正しい値（5050）であることを確認せよ.

演習M1-3 プログラム例

```
program sum100_by_mpi
use mpi
implicit none
integer :: i, istart, iend, isum_local, isum_tmp, isum, src
integer :: nprocs, myrank, ierr
integer :: istat(MPI_STATUS_SIZE)
call mpi_init( ierr )
call mpi_comm_size( MPI_COMM_WORLD, nprocs, ierr )
call mpi_comm_rank( MPI_COMM_WORLD, myrank, ierr )
istart = myrank*25 + 1
iend   = (myrank+1)*25
isum_local = 0
do i = istart, iend
  isum_local = isum_local + i
enddo
if( myrank /= 0 ) then
  call mpi_send( isum_local, 1, MPI_INTEGER, 0, 100, MPI_COMM_WORLD, ierr )
else
  isum = isum_local
  do src = 1, 3
    call mpi_recv( isum_tmp, 1, MPI_INTEGER, src, 100, MPI_COMM_WORLD, istat, ierr )
    isum = isum + isum_tmp
  end do
end if
if( myrank == 0 )  print *, 'sum =', isum
call mpi_finalize( ierr )
end program sum100_by_mpi
```

解答例

```
if( myrank == 1 ) then
  call mpi_send( isum_local, 1, MPI_INTEGER, 0, 100, MPI_COMM_WORLD, ierr )
end if

if( myrank == 2 ) then
  call mpi_send( isum_local, 1, MPI_INTEGER, 0, 200, MPI_COMM_WORLD, ierr )
end if

if( myrank == 3 ) then
  call mpi_send( isum_local, 1, MPI_INTEGER, 0, 300, MPI_COMM_WORLD, ierr )
end if


if( myrank == 0 ) then
  call mpi_recv( isum_tmp1, 1, MPI_INTEGER, 1, 100, MPI_COMM_WORLD, istat, ierr )
  call mpi_recv( isum_tmp2, 1, MPI_INTEGER, 2, 200, MPI_COMM_WORLD, istat, ierr )
  call mpi_recv( isum_tmp3, 1, MPI_INTEGER, 3, 300, MPI_COMM_WORLD, istat, ierr )
end if
```

実行時間の計測

- 並列計算の目的は．．．
  - 計算時間を短縮すること．
  - 一つの計算ノードのメモリ容量では足りない大規模問題を複数の計算ノードを利用して解くこと．

- 同じ結果が得られるが，アルゴリズムや書き方が異なったプログラムのうち，どれが一番良いか？
  - 一つの基準として，（正しい結果が得られるならば）計算時間の短いものが良い．

- 計算時間を計って比較する．→ ***計算時間を計測する方法***

計算時間を計測する方法

```
real(DP) :: time0, time2                          計測のための変数を倍精度実数型で宣言する.
      •
      •
call mpi_barrier( MPI_COMM_WORLD, ierr )          MPI_barrier関数で，計測開始の足並みを揃える.
time0 = mpi_wtime()                               mpi_wtime関数で開始時刻をtime0に設定

 (計測する部分)

call mpi_barrier( MPI_COMM_WORLD, ierr )          全プロセスで終了の足並みを揃える.
time1 = mpi_wtime()                               mpi_wtime関数で終了時刻をtime1に設定

 (time1-time0 を出力する)                          time1-time0が計測した部分の計算時間となる.
```

`mpi_barrier(comm, ierr)` ： バリア同期関数

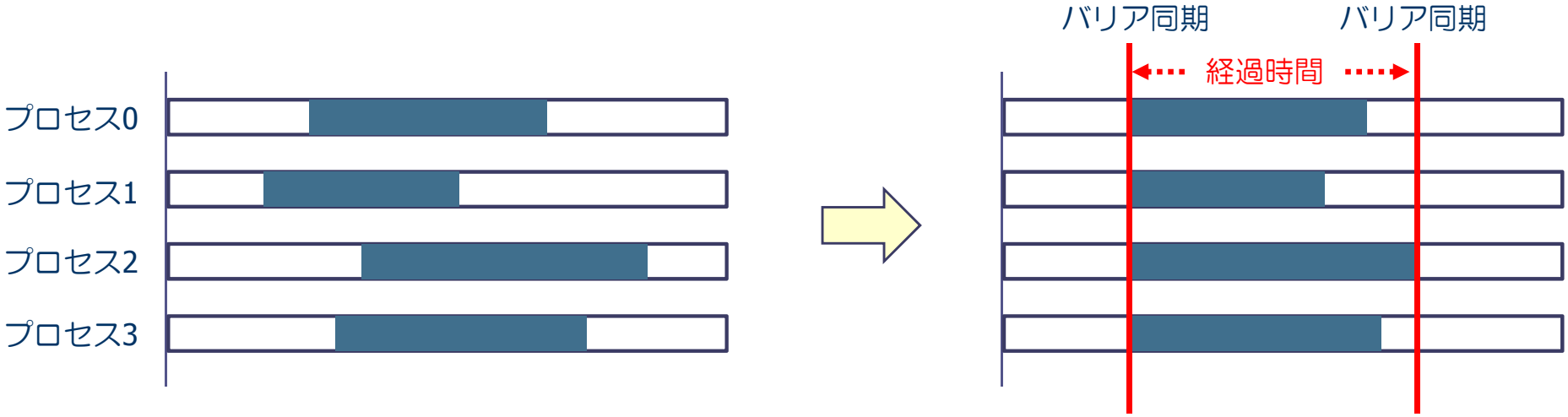
- comm: コミュニケータ（例えば，MPI_COMM_WORLD）
- ierr: 戻りコード（整数型）

var = `mpi_wtime()` ：倍精度実数を返す関数

時間計測のイメージ

- 各プロセスでの計算時間の測定関数
  - mpi_wtime()
    - ある時点を基準とした経過秒数を倍精度実数型で返す関数

- プログラムのある区間の計算時間の測定
  - プログラムの実行は各プロセスで独立なので，開始時間や終了時間が異なる.
  - ある部分の計算時間の計測では，バリア同期（MPI_barrier）により測定開始と測定終了の足並みを揃えて，計測する.



演習M2-1

- 演習M1-5で並列化したプログラムM-4（πの数値計算）の並列化プログラムについて，次の区間の計算時間をプロセス数を変えて計測せよ.
  - mpi_bcastの開始からmpi_reduceの終了までの時間
  - 計測した時間は，ランク0のプロセスに出力させる.
  - n=1,000,000として，1, 2, 4, 8プロセスで実行し，それぞれ結果が正しいことを確かめる．計算時間が短ければ，nを10倍にしてみる.
  - 計算時間を，gnuplotを使って図示せよ.
    - x軸をプロセス数, y軸を計算時間とする.
    - x軸，y軸とも対数スケールにする.　　`$ set logscale xy`

集団通信（mpi_allreduce関数）

- mpi_reduce関数は，ひとつのプロセスに，すべてのプロセスの同じ変数のデータを集め，（足し算や掛け算などの）リダクション演算を行った結果を得る関数

- すべてのプロセスで，この結果を共有したいときは，どうすれば良いか？

- プログラム方針
  ① mpi_reduce関数により，ひとつのプロセスで結果をもとめ，その結果を mpi_bcastで全プロセスに放送する.

  ② mpi_allreduce関数を使う.

集団通信 ― mpi_allreduce()

```
mpi_allreduce( sendbuff, recvbuff, count, datatype, op, comm, ierr )
```

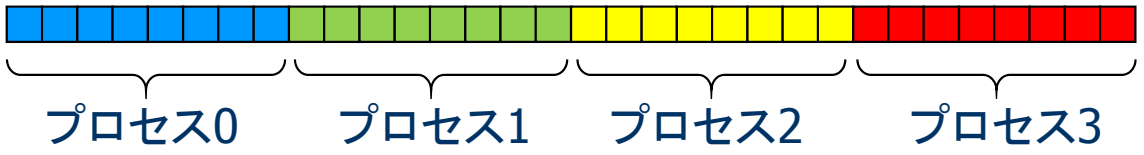
※ mpi_reduceとmpi_bcastを同時に行える関数．すべてのプロセスで同じ結果（総和など）が得られる．

- sendbuff:　送信するデータの変数名（先頭アドレス）
- recvbuff:　受信するデータの変数名（先頭アドレス）
- count:　　データの個数（整数型）
- datatype:　送信するデータの型
  - MPI_INTEGER，MPI_REAL8，MPI_CHARACTER など
- op:　　集まってきたデータに適用する演算の種類
  - MPI_SUM（総和），MPI_PROD（掛け算），MPI_MAX（最大値）など
- comm:　　コミュニケータ（例えば，MPI_COMM_WORLD）
- ierr:　　戻りコード（整数型）

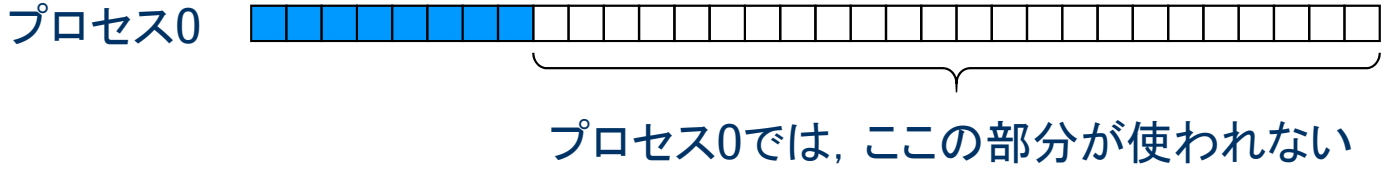
演習M2-2：ベクトルの正規化

- n次元ベクトル $\boldsymbol{x}$ の第 i 要素を i とする（$x(i) = i$）.
- このとき，$\boldsymbol{x}$ を正規化したベクトル $\boldsymbol{x}/\|\boldsymbol{x}\|_2$ を求めるプログラムを作成せよ.
  - $\|\boldsymbol{x}\|_2$ は $\boldsymbol{x}$ の各要素の2乗和の平方根である.
  - ベクトルは，ブロック分割で各プロセスに配置する.

- 各プロセスの担当する要素（nprocs はMPIプロセス数）
  - `istart = (n/nprocs)*myrank + 1`
  - `iend   = (n/nprocs)*(myrank+1)`



- ベクトルの格納方法
  - 各プロセスは長さ n の配列を持ち，そのうち自分の担当部分のみを使う



演習M2-2：ベクトルの正規化（方針）

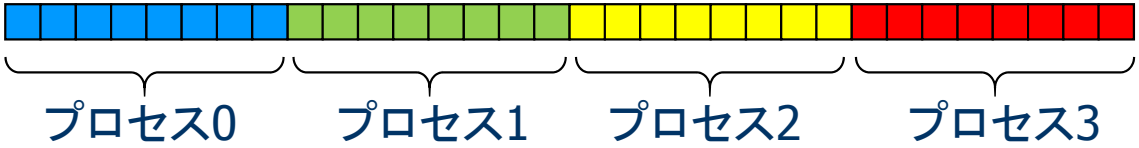
- プログラム dsumn をベースに修正する．配列x(....)を使う．
  - /tmp/mpi2/dsumn.f90
- まず，各プロセスが自分の担当分の要素について，2乗和を計算する．
- 全プロセスの総和を mpi_allredcue関数で求める．

- 各プロセスは mpi_allreduce の結果を用いて，自分の担当する要素について正規化を行う．

- n=1000 としてプロセス数を1，2，4，8と変えて計算せよ．
- 正規化されたベクトルの要素は $\tilde{x}(i) = i/\sqrt{n(n+1)(2n+1)/6}$ であるので，これと比較をして，計算が正しくできていることを確かめよ．

データの分割方法

1. ブロック分割
   - 全体のデータ（配列など）を，連続したアドレスの小部分に分割して，各プロセスに割り当てる方法



2. サイクリック分割
   - 全体のデータを，要素1個ずつ，サイクリックに各プロセスに割り当てる方法

3. ブロックサイクリック分割
   - 全体のデータを，複数の要素の塊にして，サイクリックに各プロセスに割り当てる方法

- 通信量，負荷分散などを考慮し，最適な分割を決める必要がある．

課題の提出方法と提出期限

- 演習M2-2 の提出方法
  ① 課題ごとに修正したプログラム，実行結果を一つのファイルにまとめる.

```
$ cat program.f90 > report-xx.txt
$ cat xxxxx.onnnnn >> report-xx.txt
```

  ② 以下の方法で，メールにより提出

```
$ nkf -Lu report-xx.txt | mail -s “2-2:アカウント” yokokawa@port.kobe-u.ac.jp
```

  Note) アカウントは自分のログインID
  番号（2-2）は，演習課題番号

  ※ プログラムがうまく動かない場合でも，途中結果を提出せよ.

- 期限：6月27日（火） 午後5時