計算科学演習I
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
谷口 隆晴
yaguchi@pearl.kobe-u.ac.jp

この資料は以前担当であった横川先生の資料を参考にさせて頂いています

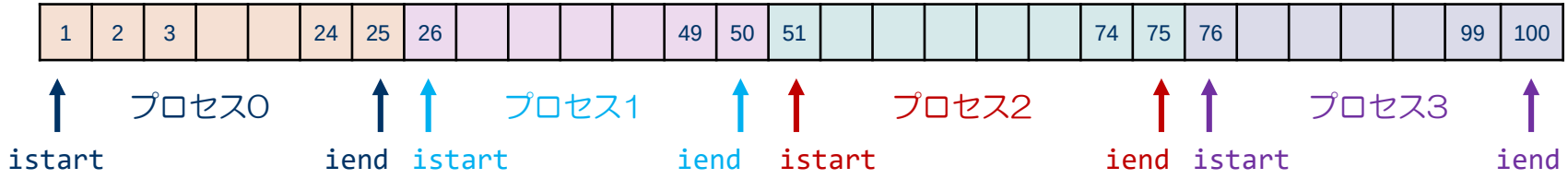
今週の講義の概要

1. 前回課題（M1-3，M1-5）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.

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*100/nprocs + 1
iend   = (myrank+1)*100/nprocs
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, nprocs-1
    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
```

演習M1-5（提出課題２）

プログラム M-3を参考にして，次のページのプログラム（M-4）を，並列計算できるようにMPIで並列化せよ．また，実際に８プロセスを用いて計算してみよ．

ただし，

- mpi_reduce を用いること．
- 結果はプロセス0が出力するようにせよ．

また，もしも，時間に余裕があり，前回，取り組んでいない場合には，以下の自由課題にも取り組んでみよ．

【自由課題】計算結果を真の値 3.141592653589... と比較せよ．
変数 n の値を10倍，1/10倍としてみると結果はどのようになるか．
なぜ，このような結果が得られるのかを考えてみよ．

プログラム例

```
program pi
use mpi
implicit none
integer, parameter :: SP = kind(1.0)
integer, parameter :: DP = selected_real_kind(2*precision(1.0_SP))
integer, parameter :: n = 1000000
integer :: i, istart, iend, nprocs, myrank, ierr
real(DP) :: x, dx, p, pg

call mpi_init(ierr)
call mpi_comm_size(MPI_COMM_WORLD,nprocs, ierr)
call mpi_comm_rank(MPI_COMM_WORLD,myrank, ierr)

istart = myrank*(n/nprocs)+1
iend = (myrank+1)*(n/nprocs)

dx = 1.0_DP/real(n,DP)
p = 0.0_DP
do i = istart, iend
  x = real(i, DP) * dx
  p = p + 4.0_DP/(1.0_DP + x ** 2)*dx
end do
call mpi_reduce(p, pg, 1, MPI_DOUBLE_PRECISION, MPI_SUM, 0, MPI_COMM_WORLD, ierr)

if(myrank == 0) then
  print *, pg
end if

call mpi_finalize(ierr)

end program pi
```

真の値との比較	3.14159265358979 3
n= 10000	3.14149265192312 7
n= 100000	3.14158265357312 2

実行時間の計測

- 並列計算の目的の一つは，計算時間の短縮．

  他の目的の例：大規模問題を解くための，メモリの確保．

- 目的を達成できるアルゴリズムの選択・プログラムの書き方は，たくさんあり得る．どれが良いか？

  ⇨（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．

- 計算時間を計って比較．

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.
```

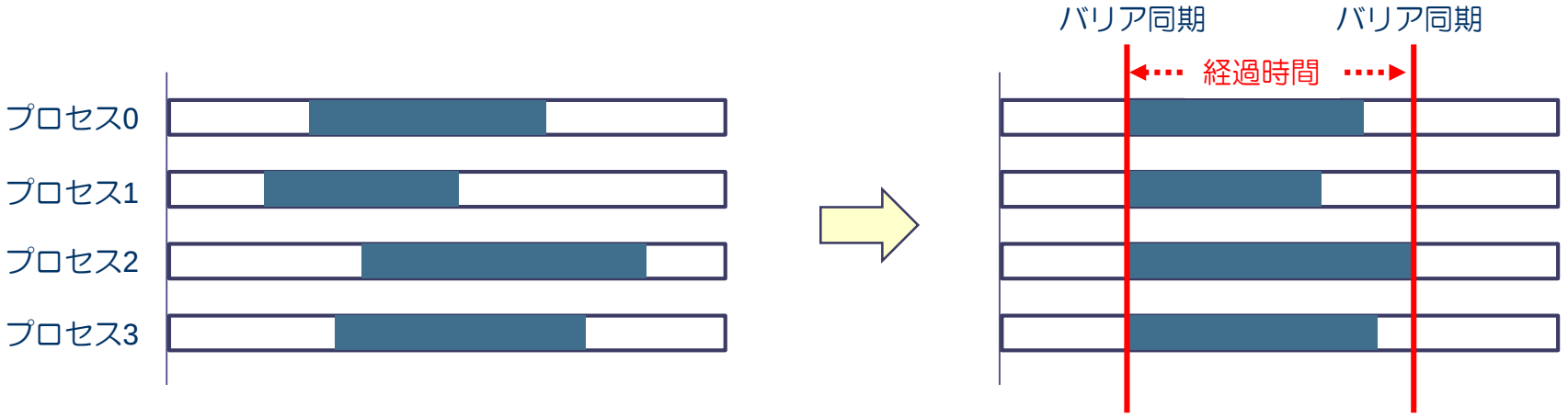
mpi_barrier(comm, ierr) ： バリア同期関数．全プロセスがここに到達するまで待機.

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

mpi_wtime() ：ある時点からの経過秒数を倍精度実数を返す関数.

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.
  → 開始時間や終了時間が，プロセスによって異なる.
- プログラムの一部の計算時間計測は，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

前回の提出課題２で作成したプログラムM-4について，次のページの赤い部分の計算時間をプロセス数を変えて計測せよ．
（次のページのプログラムは並列化前であることに注意）

- 計測した時間は，ランク0のプロセスに出力させる．
- 1, 2, 4, 8プロセスで実行．
  それぞれ結果が正しいことも確かめよ．
- 計算時間を gnuplot を使って図示せよ．結果は期待通りか？
  - x軸：プロセス数．y軸：計算時間．
  - x軸，y軸とも対数（`set logscale xy`）．

    （なぜ，両軸ともに対数にするとよいのか？）

プログラムM-4（πの数値計算）

```
program pi
implicit none
integer, parameter :: SP = kind(1.0)
integer, parameter :: DP = selected_real_kind(2*precision(1.0_SP))
integer, parameter :: n = 1000000
integer :: i
real(DP) :: x, dx, p

dx = 1.0_DP/real(n,DP)

p = 0.0_DP
do i = 1, n
  x = real(i, DP) * dx
  p = p + 4.0_DP/(1.0_DP + x ** 2)*dx
end do

print *, p

end program pi
```

このプログラムは　/tmp/mpi2/pi.f90　に置いてあります.
実行用のスクリプトの例を /tmp/mpi2/go.sh に置いてあります.

集団通信（mpi_allreduce関数）

- mpi_reduce 関数は，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，ひとつのプロセスに集める関数

  → すべてのプロセスで，この結果を共有するには？

- 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 ー mpi_allreduce()

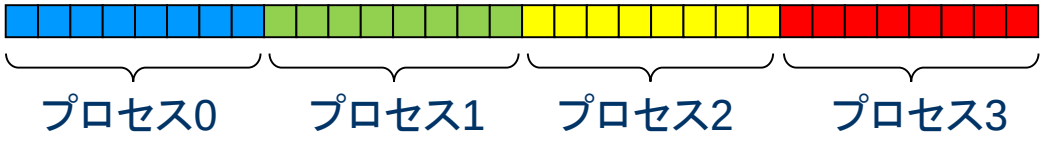
```
mpi_allreduce( sendbuff, recvbuff, count, datatype, op, comm, ierr )
```

※ 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_REAL, MPI_DOUBLE_PRECISION, 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）

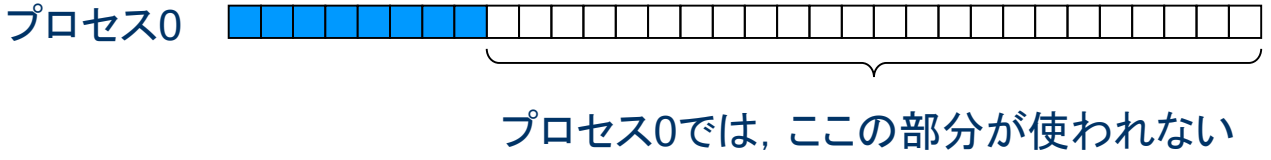
演習M2-2：ベクトルの正規化

- 第 i 要素が i である n 次元ベクトル $\boldsymbol{x}$ $(\boldsymbol{x}_i = i)$ について，$\boldsymbol{x}$ を正規化したベクトル $\boldsymbol{x}/\|\boldsymbol{x}\|_2$ を求めたい．
  - $\|\boldsymbol{x}\|_2$ は $\boldsymbol{x}$ の各要素の2乗和の平方根： $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{n} \boldsymbol{x}_i^2}$.
  - ベクトルは，ブロック分割で各プロセスに配置せよ．
- 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



※　余裕があれば，より無駄のない格納方法を考えてみよ．

演習M2-2：ベクトルの正規化（提出課題）

次のページのプログラムを，次の方針で並列化せよ．

方針

- 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．
- 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を正規化．

また，正規化するとベクトルの要素は

$$\tilde{x}(i) = i/\sqrt{n(n+1)(2n+1)/6}$$

となる．誤差（正解との差）を計算してプロセス0で出力せよ．

n=1000 として，プロセス数を1，2，4，8と変えて実行し，計算結果が正しいことを確かめよ．

※ 誤差，つまり「計算結果と正解の差」をどのように求めるべきかは自分で考えてみよ（hint: ベクトルとベクトルの差になることに注意）．

プログラムM-5

```
program dnorm
implicit none
integer, parameter :: n=1000
integer :: i
integer, parameter :: SP = kind(1.0)
integer, parameter :: DP = selected_real_kind(2*precision(1.0_SP))
real(DP) :: x(n), dsum, inorm

do i = 1, n
  x(i) =  dble(i)            ← 自分が担当する分だけを初期化
end do

dsum = 0.0_DP
do i = 1, n
  dsum = dsum + x(i)**2      ← mpi_allreduce をうまく使う.
end do
inorm = 1.0_DP/sqrt(dsum)

do i = 1, n
  x(i) = x(i)*inorm          ← 自分が担当する分を正規化.
end do
```

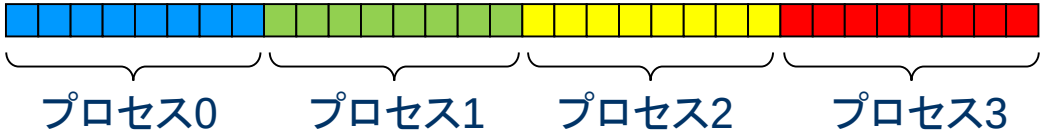
（ランク0で「正しい結果との差」を出力）
※「正しい結果との差」として何を出力すべきかは，自分で考えること.
　特に「各プロセスにおける正しい結果との差」をランク0に集めないといけないことに注意.

```
end program
```

このプログラムは /tmp/mpi2/dnorm.f90 においてあります.

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 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.txt
$ cat xxxxx.onnnnn >> report.txt
```

  ② 以下の方法で，メールにより提出

```
$ nkf -Lu report.txt | mail -s "2:アカウント" yaguchi@pearl.kobe-u.ac.jp
```

アカウントは自分のログインID

  ※ 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（水） 午後5時

参考：ターミナル上の操作を楽にする Tips

- tmux / screen コマンド

  ターミナル上にバーチャルターミナルを作るコマンド.
  使い方の詳細は各自で調べてみてください.

  - 「タブ機能」みたいなことができる.
    → プログラムを書く emacs のタブと,
    コンパイル用のターミナルのタブを作ると作業が楽.
  - バーチャルターミナルは，サーバー上に保管できる.
    → ログアウトする際，作業環境を保存できる.

- bash のヒストリー検索

  bash は ctrl-r で過去に入力したコマンドを検索し，実行可能.

  例）mpifrtpx program.f90 と一度入力したことがあれば
  mpi くらいまで打ち込めば ctrl + r で補完できる.