計算科学演習Ⅰ
MPIを用いた並列計算（I）

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2015度担当の横川先生の資料を参考にさせて頂いています

講義概要

- 分散メモリ型計算機上のプログラミング
- メッセージ・パシング・インターフェイス
  （Message Passing Interface，MPI）
- MPIプログラム（M-1）：Hello, world!
- MPIプログラム（M-2）：1対1通信関数
- MPIプログラム（M-3）：集団通信関数

コンパイル・実行方法

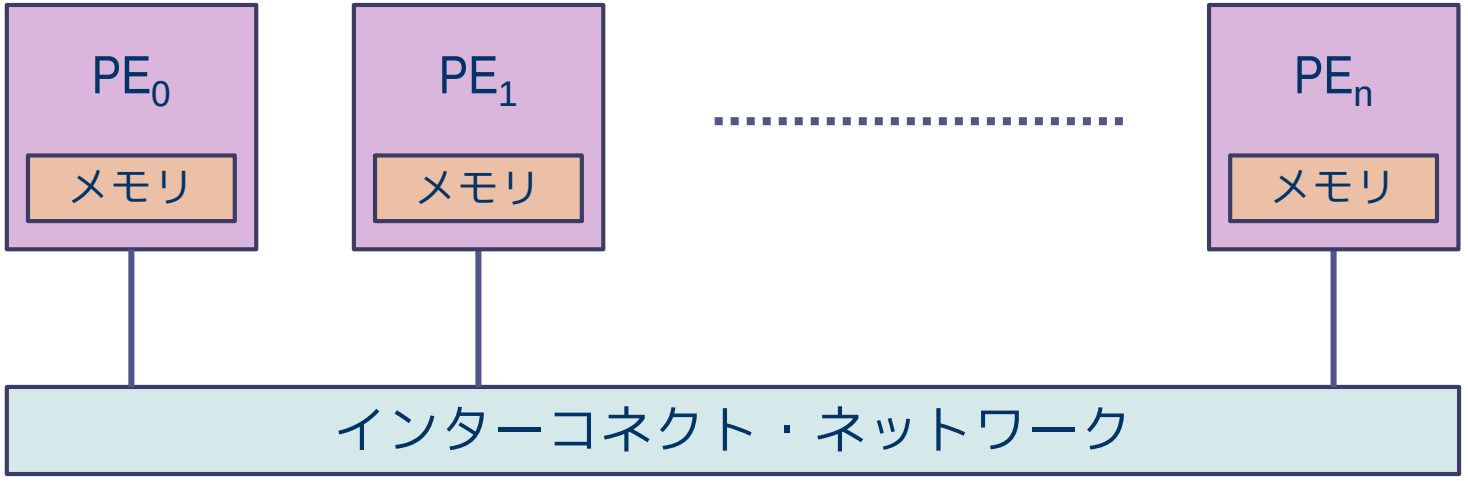
基本的な考え方
（ＳＰＭＤ）に慣れる

分散メモリ型並列計算機

- 複数のプロセッサがネットワークで接続
- それぞれのプロセッサ（PE）が，メモリを保有.
  → 各PEが自分のメモリ領域のみアクセス可能.

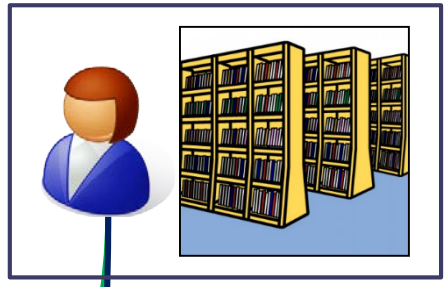
特徴

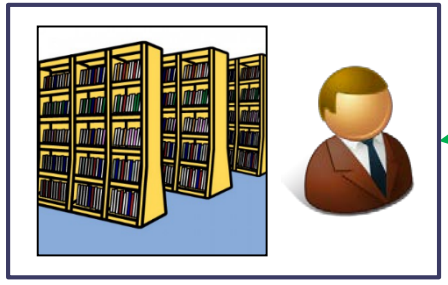
数千から数万PE規模の並列システムが可能
PEの間のデータ分散を意識したプログラミングが必要.
プログラミング技術：メッセージ・パシング・インターフェイス（MPI）

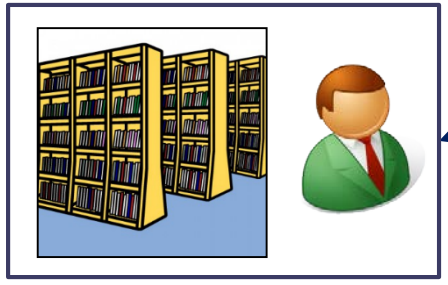


MPIプログラミングのイメージ

- 離れた場所にいる複数の人に仕事を依頼.
- それぞれの人が，本棚に一連のノートを所持.
  - それぞれの人には，名前が付いている（人を区別できる）.
  - ノートには同じ名前が付けられているが，中身は違っている.
- 本棚のノートに対し，
  それぞれが，読んだり書いたり…
  - 大体は同じ作業だが，最初のノートの中身が違うので中身はそれぞれ違う.
  - ある人に，他の人とは違う作業をさせたい場合には，名前で作業を指示.
- 時々，他の人のノートを見たい.
  - 相手にノートの中身を送ってあげる.
  - 送られた人は，それを違う名前のノートに中身を書き写す.



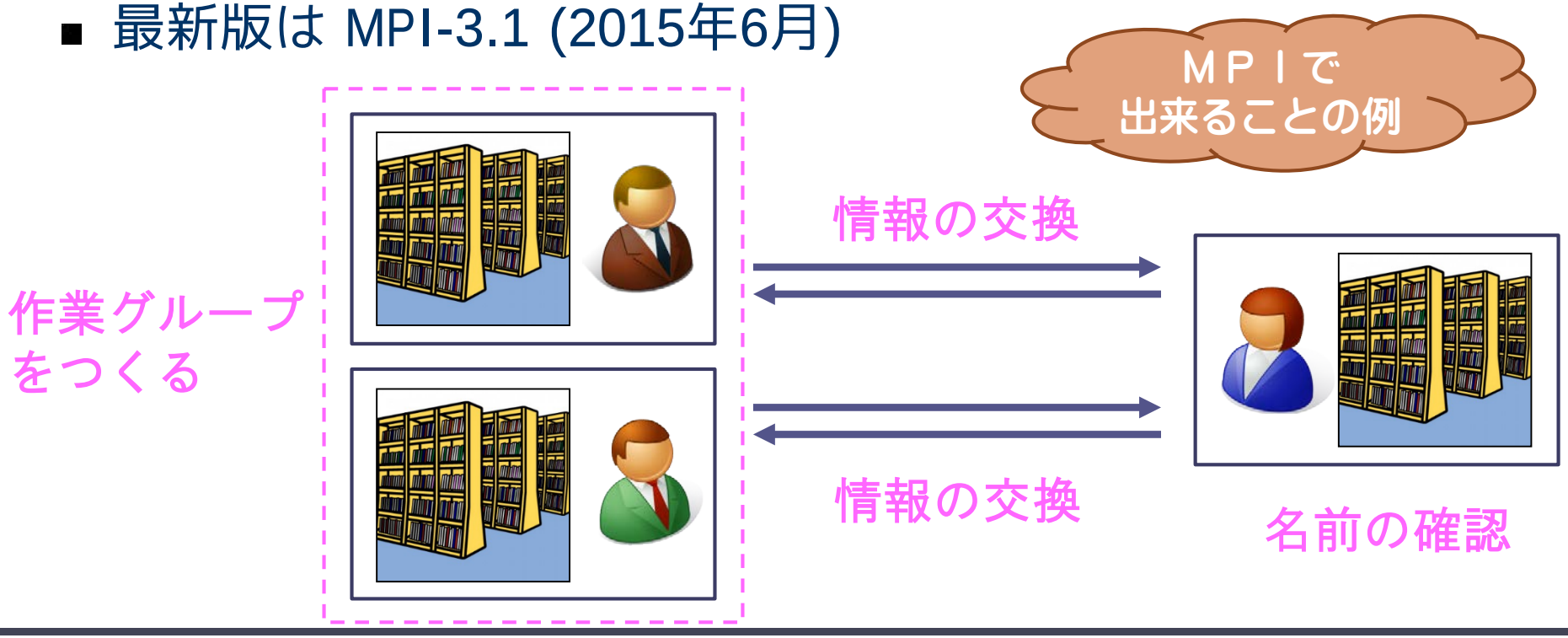




【必要な情報】
- ノートの名前
- 何冊
- 誰に
- 荷物のタグなど

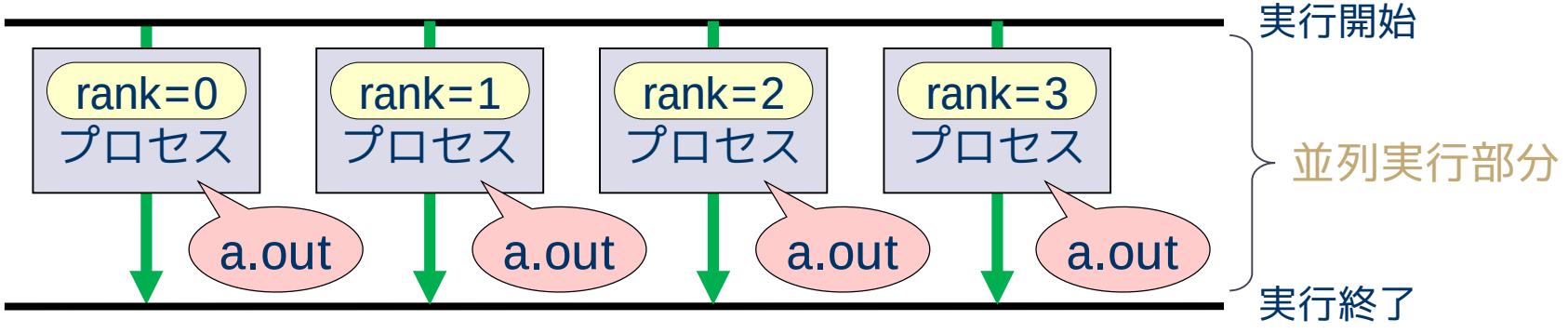
Message Passing Interface (MPI)

- 複数の独立したプロセス間で，並列処理を行うための
  プロセス間メッセージ通信の標準規格
- 1992年頃より米国の計算機メーカ，大学などを中心に標準化
- 最新版は MPI-3.1 (2015年6月)



MPIの実行モデル：SPMD（Single Program, Multiple Data)

- 複数のプロセス（=離れた所にいる人）により並列実行.
- 実行開始から終了まで
  全プロセスが同じプログラムを実行
  各MPIプロセスは固有の番号（ランク番号）を持つ
  - P個のプロセスで実行する場合,
    プロセス番号は0から(P-1) までの整数
  - この番号によって仕事を変える　← プログラマの仕事

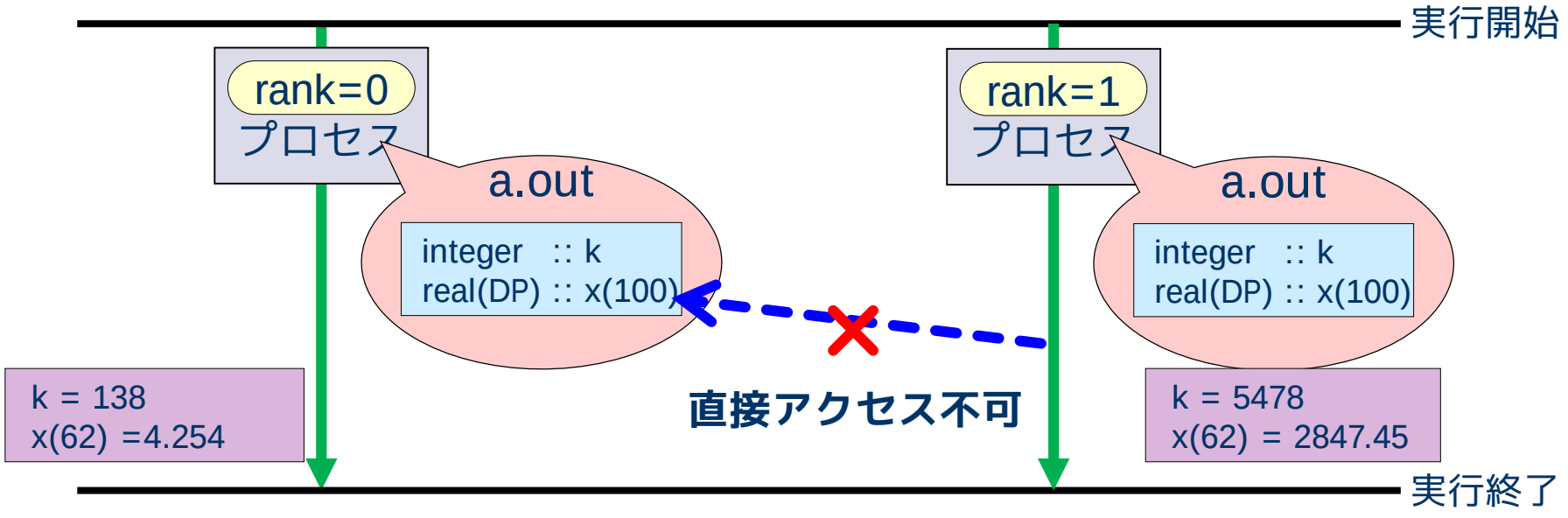


MPIでのメモリ空間とプロセス間通信

メモリ空間：プロセスごとに**独立して保持**

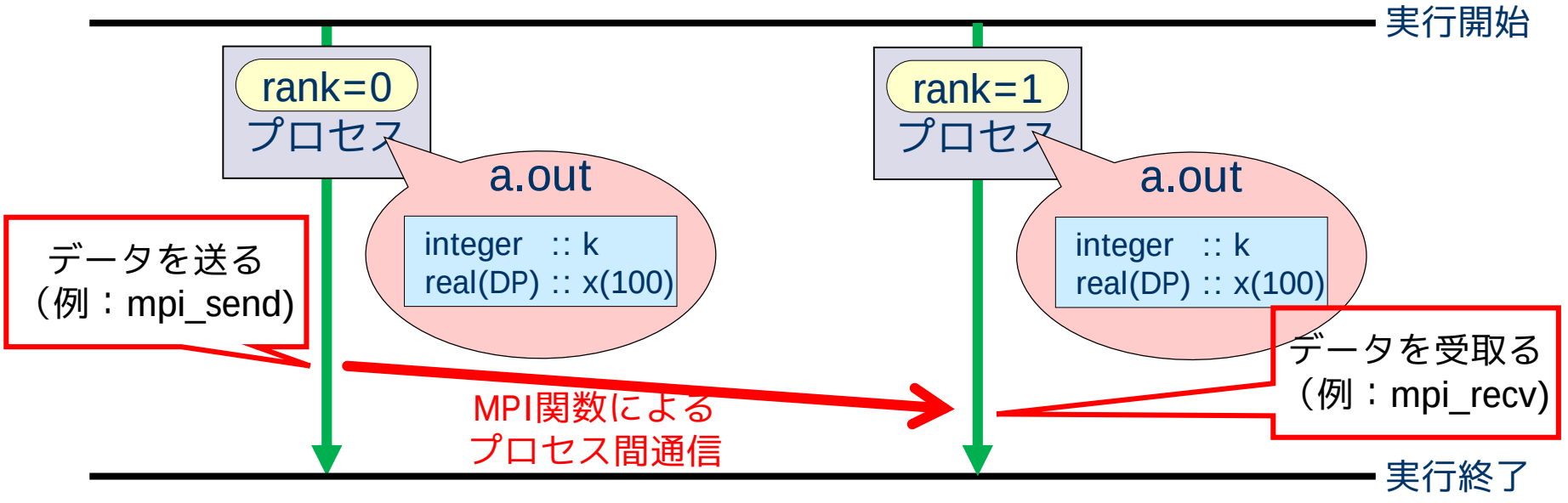
- プログラム中で定義された変数や配列は，
  同じ名前で各プロセスのメモリ上に独立に割り当てられる．

- 同じ変数や配列に対して，プロセスごとに違う値を与えることが可能．
  他のプロセスの持つ変数や配列には，直接にアクセスできない．
  → プロセス間通信でデータを送ってもらう．



メッセージパッシング方式

- メッセージパッシング方式：
  送り手から受けてへメッセージ（データ）を伝達.
  （≃荷物の配送，手紙）
- ＭＰＩ ≃ この方式によるプロセス間通信関数の集まり.



MPIプログラムのスケルトン

```
program main
use mpi                                                    MPIモジュールの取り込み（おまじない1）
implicit none
integer :: nprocs, myrank, ierr                            MPIで使う変数の宣言

call mpi_init( ierr )                                      MPIの初期化（おまじない2）
call mpi_comm_size( MPI_COMM_WORLD, nprocs, ierr )         MPIで使うプロセス数を nprocs に取得
call mpi_comm_rank( MPI_COMM_WORLD, myrank, ierr )         自分のプロセス番号を myrank に取得


  (この部分に並列実行するプログラムを書く)



call mpi_finalize( ierr )                                  MPIの終了処理（おまじない3）

end program main
```

各プロセスに何をさせるかを myrank の値で場合分けし，うまく仕事を割り振る．

MPIプログラムの基本構成（説明）

- `call mpi_init( ierr )`
  MPIの初期化を行う．MPIプログラムの最初に必ず書く．

- `call mpi_comm_size( MPI_COMM_WORLD, nprocs, ierr )`
  MPIの全プロセス数を，2番目の引数 nprocs（整数型）に取得．
  MPI_COMM_WORLDはコミュニケータと呼ばれるものの一つで，
  「最初に割り当てられるすべてのプロセスの集合」を表す．

- `call mpi_comm_rank( MPI_COMM_WORLD, myrank, ierr )`
  自分のプロセス番号（0からnprocs-1のどれか）を，
  2番目の引数 myrank（整数型）に取得．

- `call mpi_finalize( ierr )`
  MPIの終了処理をする．MPIプログラムの最後に必ず書く．

π-コンピュータでのプログラムの実行手順

① プログラム作成：　エディタを用いてプログラム（xxx.f90）を作成

↓

② プログラムのコンパイル：　mpifrtpx xxx.f90　← これまでと違う．注意！

↓

③ ジョブスクリプトを作成：エディタを用いてスクリプト(xxx.sh)を作成

↓

④ ジョブを投入：　pjsub（ジョブスクリプトファイル名）

[INFO] PJM xxxx pjsub JOB nnnnn submitted.

【注意】 nnnnnがジョブ番号（ジョブに与えられたシステム内で唯一の番号）

↓

⑤ ジョブ状態の確認：　pjstat

↓

⑥ 結果の確認：　job.sh.onnnnn を確認する．

演習M1-1：Hello, world!

次のスライドのプログラム（M-1）をコンパイルし，
2 及び 4 プロセスで実行して結果を確認せよ．

```
$  mkdir mpi1
$  cd mpi1

$  mkdir M-1
$  cd M-1

$  cp /tmp/mpi1/hello_mpi.f90 ./

$  mpifrtpx hello_mpi.f90

$  cp /tmp/mpi1/go.sh ./


$  pjsub go.sh
[INFO] PJM 0000 pjsub Job nnnnn submitted.

$  cat hello.onnnnn
```

今日の演習用のディレクトリを作成する．

演習M-1用のディレクトリを作成する．

hello_mpi.f90 をコピー．

ソースプログラムをコンパイル．

ジョブスクリプト go.sh をコピー．
（プロセス数の指定など，必要な部分を修正）

ジョブを投入し，実行結果を確認．

※ helloは，go.sh内で指定した jobname

MPIプログラム（M-1）： Hello, world!

```
program hello_by_mpi

use mpi
implicit none

integer :: nprocs, myrank, ierr

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

print *, ‘Hello, world!  My rank number and nprocs are’, myrank, ‘,’, nprocs

call mpi_finalize( ierr )

end program hello_by_mpi
```

このプログラムは　/tmp/mpi1/hello_mpi.f90　に置いてあります．

ジョブスクリプト例（2プロセスで実行する場合）

`#!/bin/bash`	シェル名
`#PJM -N “jobname”`	ジョブ名（任意の名前）を指定
`#PJM -L “rscgrp=small”`	投入先のキュー
`#PJM -L “node=2”`	使用ノード数
`#PJM -L “elapse=00:02:00”`	最大実行時間
`#PJM -j`	stderrをstdoutにマージ
`mpiexec -n 2 ./a.out`	並列数を指定してMPIプログラムを実行

※ 4プロセスで実行する場合は 2 を 4 に変更.

このスクリプトは　/tmp/mpi1/go.sh　に置いてあります.

プログラム M-1の実行結果の確認

- ２プロセスでの実行結果例

```
Hello, world!  My rank number and nprocs are 0 , 2
Hello, world!  My rank number and nprocs are 1 , 2
```

- ４プロセスでの実行結果例

```
Hello, world!  My rank number and nprocs are 2 , 4
Hello, world!  My rank number and nprocs are 0 , 4
Hello, world!  My rank number and nprocs are 3 , 4
Hello, world!  My rank number and nprocs are 1 , 4
```

（注意）出力はランク順に並ぶとは限らない．
また，実行するごとに出力の順番が異なることが多い．

ポイント

- 各プロセスが同じプログラムを実行している．
- 各プロセスが持っているランク番号(myrankの値）が異なっている．

→ これを利用して，各プロセスに異なる作業を指示．

プログラムM-1（hello_mpi.f90） の説明

`program hello_by_mpi`	
	（おまじない） mpi用のモジュールをインクルード
`use mpi` `implicit none`	
`integer :: nprocs, myrank, ierr`	
	（おまじない） MPIの初期化 MPIで使うプロセス数を nprocs に取得 自分のプロセス番号を myrank に取得

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

`print *, 'Hello, world!  My rank number and nprocs are ', myrank, ',',  nprocs`	
	各プロセスで myrank と nprocs を出力する. ※ myrank はプロセスごとに異なる ※ nprocs はすべてのプロセスで同じ
	（おまじない） MPIの終了処理
`call mpi_finalize( ierr )`	
`end program hello_by_mpi`	

MPIプログラム（M-2）：１対１通信関数

【問題】
1から100までの整数の和を2並列で求めなさい.

プログラムの方針

- プロセス0：　1から50までの和を求める.
- プロセス1：　51から100までの和を求める.

- プロセス1の結果をプロセス0に転送.

- プロセス0で，自分の結果と転送された結果を足して出力.

MPIプログラム M-2

```
program sum100_by_mpi
use mpi
implicit none
integer :: i, istart, iend, isum_local, isum_tmp
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*50 + 1
iend   = (myrank+1)*50
isum_local = 0
do i = istart, iend
  isum_local = isum_local + i
enddo
if( myrank == 1 ) then
  call mpi_send( isum_local, 1, MPI_INTEGER, 0, 100, MPI_COMM_WORLD, ierr )
else
  call mpi_recv( isum_tmp, 1, MPI_INTEGER, 1, 100, MPI_COMM_WORLD, istat, ierr )
end if
if( myrank == 0 )  print *, 'sum =', isum_local+isum_tmp
call mpi_finalize( ierr )
end program sum100_by_mpi
```

MPIプログラム M-2の説明

- 青：MPIプログラムのおまじない（既出）
- 緑：プロセス番号（ランク）に応じた処理
- 赤：MPI関数によるプロセス間通信

```
program sum100_by_mpi
use mpi
implicit none
integer :: i, istart, iend, isum_local, isum_tmp
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*50 + 1
iend   = (myrank+1)*50
isum_local = 0
do i = istart, iend
  isum_local = isum_local + i
enddo
if( myrank == 1 ) then
  call mpi_send( isum_local, 1, MPI_INTEGER, 0, 100, MPI_COMM_WORLD, ierr )
else
  call mpi_recv( isum_tmp, 1, MPI_INTEGER, 1, 100, MPI_COMM_WORLD, istat, ierr )
end if
if( myrank == 0 )  print *, 'sum =', isum_local+isum_
call mpi_finalize( ierr )
end program sum100_by_mpi
```

ランクの値から自分の計算範囲を求める

各プロセスが部分和を計算

プロセス1はプロセス0に自分の部分和を送信

プロセス0はプロセス1から部分和を受信（変数名が違うことに注意）

プロセス0が，総和を出力

1対1通信 – 送信関数 mpi_send（送り出し側）

```
mpi_send( buff, count, datatype, dest, tag, comm, ierr )
```

※ ランク番号destのプロセスに，変数buffの値を送信する．

- buff:　　送信するデータの変数名（先頭アドレス）
- count:　　送信するデータの数（整数型）
- datatype:　送信するデータの型
  - MPI_INTEGER, MPI_REAL, MPI_DOUBLE_PRECISIONなど
- dest:　　送信先プロセスのランク番号（整数型）
- tag:　　メッセージ識別番号．送るデータを区別（整数型）
- comm:　　コミュニケータ（例えば，MPI_COMM_WORLD）
- ierr:　　戻りコード（整数型）

1対1通信 – 受信関数 mpi_recv（受け取り側）

```
mpi_recv( buff, count, datatype, source, tag, comm, status, ierr )
```

※ ランク番号sourceのプロセスから送られたデータを，変数buffに格納する．

- buff:　　受信するデータのための変数名（先頭アドレス）
- count:　　受信するデータの数（整数型）
- datatype:　受信するデータの型
  - MPI_INTEGER，MPI_REAL，MPI_DOUBLE_PRECISIONなど
- source:　　送信してくる相手プロセスのランク番号（整数型）
- tag:　　メッセージ識別番号．受け取ったデータを区別（整数型）
- comm:　　コミュニケータ（例えば，MPI_COMM_WORLD）
- status:　　受信の状態を格納するサイズMPI_STATUS_SIZEの配列（整数型）
- ierr:　　戻りコード（整数型）

関数の引数に関する注意（共通）

- buff
  - 送信するデータは領域は，メモリ上で連続アドレスでなければならない．
    “先頭アドレスからｘｘバイトを送れ” という関数なので．
  - 他の通信関数でも同じ．
  - したがって，メモリ上で離れた変数を同時に送りたい場合は，別の変数に連続してパック（pack）させてから送る必要がある．

- datatype：予約語がある．
  - MPI_INTEGER（整数型），MPI_REAL（単精度実数型），MPI_DOUBLE_PRECISION または MPI_REAL8（倍精度実数型）などが使用できる．
  - バイト数を計算するために必要（c言語のsizeof()のようなもの）

- tag
  - 同じプロセスに対し，複数回メッセージを送るとき，受取ったプロセスが，どのメッセージかを区別するために使用．
  - mpi_send / mpi_recv で，対応するメッセージには同じ tag を指定．
  - 複数回のメッセージでも，送受信の順番などを区別できる場合は，同じtagでも良い．

演習M1-2　1から100までの和を2並列で求めるプログラムの実行

1 から 100 までの整数の和を2並列で求めるプログラムを
2プロセスで実行し，結果を確認せよ.

【手順】

① /tmp/mpi1/sum100_mpi.f90 を適切なディレクトリにコピー.
② コンパイルし，ジョブを実行.
③ 結果（sum100.o*nnnnn*）を確認する.

出力に正しい答え（sum = 5050）が出力されているか？

※ プロセス0だけが出力していることに注意.

演習M1-3 （提出課題１）

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．

※ 後で学ぶ mpi_reduce は使わないこと．

ポイント

- myrank をうまく使い，各プロセスの部分和を計算する範囲を求める．
- 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

MPIプログラム（M3）：集団通信関数

- １対１通信関数の煩雑な点
  - プロセス数が多くなると，１対１通信関数を用いたプログラムは複雑
  - 煩雑になるとバグが入りやすい.

- もっと簡単な方法はないのか？ → 集団通信関数
  - mpi_bcast
    あるプロセスから，すべてのプロセスに値を一斉に配る関数
  - mpi_reduce
    すべてのプロセスから，あるプロセスに値を集めて，何らかの演算（+，ｘ，max，min）をする関数

MPIプログラム M-3（集団通信関数を使う）

```
program sum_by_reduction
use mpi
implicit none
integer :: n, i, istart, iend, isum_local, isum
integer :: nprocs, myrank, ierr

call mpi_init( ierr )
call mpi_comm_size( MPI_COMM_WORLD, nprocs, ierr )
call mpi_comm_rank( MPI_COMM_WORLD, myrank, ierr )
if( myrank==0)  n=10000
call mpi_bcast( n, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr )
istart = (n/nprocs)*myrank + 1
iend   = (n/nprocs)*(myrank+1)
isum_local = 0
do i = istart, iend
  isum_local = isum_local + i
enddo
call mpi_reduce( isum_local, isum, 1, MPI_INTEGER, MPI_SUM, 0, &
                 MPI_COMM_WORLD, ierr )
if( myrank == 0 )  print *, 'sum (by reduction function) =', isum

call mpi_finalize( ierr )
end program sum_by_reduction
```

MPIプログラム M-3の説明

- 青：MPIプログラムのおまじない（既出）
- 緑：プロセス番号（ランク）に応じた処理
- 赤：MPI関数によるプロセス間通信

```
program sum_by_reduction
use mpi
implicit none
integer :: n, i, istart, iend, isum_local, isum
integer :: nprocs, myrank, ierr

call mpi_init( ierr )
call mpi_comm_size( MPI_COMM_WORLD, nprocs, ierr )
call mpi_comm_rank( MPI_COMM_WORLD, myrank, ierr )
if( myrank==0)  n=10000
call mpi_bcast( n, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr )
istart = (n/nprocs)*myrank + 1
iend   = (n/nprocs)*(myrank+1)
isum_local = 0
do i = istart, iend
  isum_local = isum_local + i
enddo
call mpi_reduce( isum_local, isum, 1, MPI_INTEGER, MPI_SUM, 0, &
                 MPI_COMM_WORLD, ierr )
if( myrank == 0 )  print *, 'sum (by reduction function) =', isum

call mpi_finalize( ierr )
end program sum_by_reduction
```

プロセス0がnの値をセットする

nの値を放送

ランクの値から自分の計算範囲を求める

各プロセスが部分和を計算

部分和の総和を計算（プロセス0に集める）

プロセス0だけが結果を出力

集団通信 － broadcast

```
mpi_bcast( buff, count, datatype, root, comm, ierr )
```

※ ランク番号rootのプロセスが持つbuffの値を，commで指定された他のすべてのプロセスのbuffに配布する．

- buff: 送り主（root）が送信するデータの変数名（先頭アドレス）
  他のMPIプロセスは，同じ変数名でデータを受け取る．
- 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など
- root: 送り主のMPIプロセス番号
- comm: コミュニケータ（例えば，MPI_COMM_WORLD）
- ierr: 戻りコード（整数型）

集団通信 － reduction

```
mpi_reduce( sendbuff, recvbuff, count, datatype, op, root, comm, ierr )
```

※ commで指定されたすべてのプロセスからデータを，ランク番号 root のプロセスに集め，演算（op）を適用した結果をrecvbuffに設定する．

- 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
- root:　データを集めるMPIプロセス番号
- comm:　コミュニケータ（例えば，MPI_COMM_WORLD）
- ierr:　戻りコード（整数型）

リダクション演算とは

- リダクション演算
  - 加算，乗算，最大値のように，複数のデータを入力として1個の出力データを求める演算
- MPIで使えるリダクション演算
  - MPI_SUM（和），MPI_PROD（積），
  - MPI_MAX（最大値），MPI_MIN（最小値）
    ※他にも論理和などがある
- ベクトルに対するリダクション演算も可能
  - ベクトルの各要素に対してリダクション演算を行い，その結果を要素とするベクトルを生成
  - x_1，x_2，…，x_m をそれぞれ長さ n のベクトルとするとき，それらの和 $x = x_1 + x_2 + \ldots + x_m$ を求める計算など
  - 引数 count に，ベクトルの長さ n を入れればよい

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M-4（提出課題２： 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/mpi1/pi.f90　に置いてあります．

課題の提出方法と提出期限

- 演習M1-3，M1-5 の提出方法
  ① 課題ごとに修正したプログラムと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 下記のアドレスまでメールにより提出

【M1-3の場合】

```
nkf –Lu report-xx.txt | mail –s “1-3:アカウント名” yaguchi@pearl.kobe-u.ac.jp
```

【M1-5の場合】

```
nkf –Lu report-xx.txt | mail –s “1-5:アカウント名” yaguchi@pearl.kobe-u.ac.jp
```

※ 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21日（水） 午後5時．
  なるべくこの時間に終わらせましょう！