演習２：ＭＰＩ初歩

谷口 隆晴

神戸大学 システム情報学研究科 計算科学専攻

2012 年 8 月 7 日

演習２：ＭＰＩ初歩

内容：MPI を使ってみる！

- 演習１：Hello World の並列化 I (通信無し)
- 演習２：Hello World の並列化 II (1 対 1 通信)
- 演習３：内積の計算 I (時間計測)
- 演習４：内積の計算 II (集団通信)
- 演習５：配列のスワップ (ノンブロッキング通信，**sendrecv**)
- 演習６：π の数値計算 (まとめ)

MPI の基本的な命令を復習しながら進めていきます．解説は基本的に fortran で行います．C 言語用の資料は後半にあります．

演習用のプログラムファイルを
wget http://www.na.scitec.kobe-u.ac.jp/~yaguchi/riken2012/enshu2.zip
でダウンロード，unzip enshu2.zip で解凍して利用して下さい．

FORTRAN 編

【復習】MPI プログラムの基本構成

```
program main
use mpi                                                    （mpif.h のインクルード）
implicit none
integer :: nprocs, myrank, ierr
call mpi_init(ierr)                                        （MPI の初期化）
call mpi_comm_size(MPI_COMM_WORLD,nprocs,ierr)             （全プロセス数を取得）
call mpi_comm_rank(MPI_COMM_WORLD,myrank,ierr)             （自分の番号を取得）

（この部分を並列実行）

call mpi_finalize(ierr)                                    （MPI の終了処理）
end program main
```

（この部分を並列実行）の部分は同じプログラムコード．myrank の値によって，うまく仕事が割り振られるように書く．

（このファイルは fortran/template.f90 にあります）

【復習】各命令の意味

- mpi_init(ierr)
  - MPI の初期化を行う．プログラムの最初に必ず入れる．
- mpi_comm_size(MPI_COMM_WORLD,nprocs,ierr)
  - 全プロセス数を取得し，nprocs に保存する．
    - MPI_COMM_WORLD はコミュニケータの一種．
    - コミュニケータは作業グループを指定．
    - MPI_COMM_WORLD は全員からなる作業グループ．
    - これを別のものにすると，その作業グループのプロセス数を取得．
- mpi_comm_rank(MPI_COMM_WORLD,myrank,ierr)
  - 自分のプロセス番号を取得し，myrank に保存する．
  - MPI_COMM_WORLD 以外を指定すると，その作業グループでの番号を取得．
- mpi_finalize(ierr)
  - MPI の終了処理を行う．プログラムの最後に必ず入れる．

基本的に前のスライドのプログラムと同じ順番で呼べば良い．

【演習 1 】Hello World!

課題 1

“hello world from （自分の番号）” を表示する並列プログラムを作成せよ．
つまり

- hello world のプログラムを並列化し，
- ついでに自分の番号（myrank）も表示するようにせよ．

プログラムを作成したらコンパイルし，４プロセスで実行してみよ．

４プロセスで実行した場合の実行例

```
hello  world  from      0
hello  world  from      2
hello  world  from      3
hello  world  from      1
```

- コンパイル方法

  mpifrtpx hello.f90
  （プログラムファイル名）

- 実行方法（後述）

  pjsub job.sh
  （スクリプトファイル名）

【重要】./a.out で実行しないこと！

大型計算機上でのプログラム実行：キューイングシステム

スパコンは共有財産！ ➡ ジョブの管理が必要

キューイングシステム

- 負荷状況・リソース使用量を監視し，ユーザが投入したジョブを適切な計算ノードに割り当て，実行するソフトウェア.
- 今回は富士通 Technical Computing Suite を使用.

プログラム実行の流れ

1. ジョブスクリプトを作成
2. ジョブを投入
3. （ジョブの状態を確認）
4. 結果を確認

```
./a.out
```

で実行するのでは ない.

スクリプトファイルの例

```
#!/bin/sh

#PJM -L "rscgrp=school"                （キューの指定，今回は school のみ）
#PJM -L "node=8"                       （ノード数=最大プロセス数の指定）
#PJM -L "elapse=3:00"                  （最大で３分）
#PJM -j                                （標準出力と標準エラー出力をまとめる）

mpiexec -n 4 ./a.out                   （プロセス数を指定して実行）
```

（このファイルは fortran/job.sh にあります）

- プロセス数を変える場合は赤字部分の数字を変える.
- 以下の課題でも，適宜，修正して使いまわして下さい.

ジョブの投入・状態確認・削除

- ジョブの投入

```
pjsub （ジョブスクリプト名）
```

- ジョブの状態確認

```
pjstat
```

- ジョブのキャンセル

```
pjdel （ジョブ番号）
```

【実行方法と結果の確認】

- Hello World のジョブを投入

```
pjsub job.sh
```

[INFO] PJM 0000 pjsub Job 1057 submitted. などと表示.

➡ **1057** の部分がジョブ番号.

- job.sh.oXXXX（XXXX はジョブ番号）などというファイルが作成され，その中に”hello world” （またはエラー）が出力されます.

【復習】 1 対 1 通信，送信側

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

- buff: 送信するデータの先頭アドレス．
- count: 整数型． 送信するデータの数．
- datatype: 送信するデータの型． MPI_CHARACTER, MPI_INTEGER, MPI_DOUBLE_PRECISION など．
- dest: 整数型． 送信相手の番号．
- tag: 整数型． 送るデータを区別するための “整理番号”．
  （≃ 宅急便の取扱い番号．同一の相手と複数送受信するときの識別用．
  通常は 0 で良い）
- comm: コミュニケータ． 特に作業グループを指定する必要がなければ MPI_COMM_WORLD．
- ierr: 整数型． エラーコード．

【復習】 1 対 1 通信，受信側

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

- buff: 受信したデータ格納先の先頭アドレス．
- count: 整数型． 受信するデータの数．
- datatype: 受信するデータの型． MPI_CHARACTER, MPI_INTEGER, MPI_DOUBLE_PRECISION など．
- source: 整数型． 送り主の番号．
- tag: 整数型． 送信時につけた “整理番号”．
- comm: コミュニケータ． 特に作業グループを指定する必要がなければ MPI_COMM_WORLD．
- status: 整数型の配列． サイズは MPI_STATUS_SIZE．

  （送信側には含まれなかった引数なので忘れないように．）
- ierr: 整数型． エラーコード．

【演習２】Hello World (Part II)

課題２

プロセス 0 から受け取ったメッセージに自分の番号を追加して表示するプログラムを作成せよ．例えば

- プロセス 0 は “hello world from”という文字列（16 文字）を他のプロセスに送る．
- 他のプロセスは，この文字列を受け取り，自分の番号（myrank）を加えて表示する．

プログラムを作成したらコンパイルし，４プロセスで実行してみよ．

４プロセスで実行した場合の実行例

```
hello world from 2
hello world from 3
hello world from 1
```

簡単な時間計測方法

mpi_wtime()

- 過去のある時刻からの経過時間を秒単位の実数値（double precision）で返す関数.
- 計測したい部分をこの関数で挟めば時間計測ができる.
- ただし，全員がその地点にたどり着いていることを保証するため，直前に mpi_barrier をとる.

```
call mpi_barrier(MPI_COMM_WORLD, ierr)
time0 = mpi_wtime()
（計測する部分）
call mpi_barrier(MPI_COMM_WORLD, ierr)
time1 = mpi_wtime()
（time1-time0 を誰か（例えばプロセス 0）が出力）
```

mpi_barrier(comm,ierr)

- comm で指定した作業グループのメンバーは，そのグループ全員がその地点にたどり着くまで待つ.

【演習３】単純な内積の計算

課題３

次のスライドのプログラムは２つのベクトルの内積を計算するプログラムである．並列化されている（？）が，このままでは，全員，同じ計算をしているので意味がない．そこで，

- 各プロセスで部分和を計算して
- それを mpi_send でプロセス 0 に送り，
- プロセス 0 で総和を求め，出力する

ように修正せよ．
また，mpi_wtime を用いて，赤字部分に相当する総和計算部分の実行時間を計測できるように修正し，1, 2, 4 プロセスで実行した場合の実行時間を計測せよ．

- パラメータ n はプロセス数で割りきれると仮定して良い．
- このとき，n 個のデータを p 分割したときの第 i 区間は $[(i*n)/p+1, ((i+1)*n)/p]$.

演習３のプログラム

```
program main
use mpi
implicit none
integer :: i, nprocs, myrank, ierr
integer, parameter :: n=10000
double precision :: v(n), w(n)
double precision :: res
call mpi_init(ierr)
call mpi_comm_size(MPI_COMM_WORLD,nprocs,ierr)
call mpi_comm_rank(MPI_COMM_WORLD,myrank,ierr)

do i=1,n
v(i) = dsin(i*0.1d0)
w(i) = dcos(i*0.1d0)
end do

res = 0.0d0
do i=1,n
res = res + v(i)*w(i)
end do

print *, res
call mpi_finalize(ierr)
end program main
```

（このファイルは fortran/innerproduct.f90 にあります）

【復習】集団通信，reduction

演習３での総和計算は専用の命令が存在.

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

- sendbuff: 送信するデータの先頭アドレス.
- recvbuff: 受信するデータの先頭アドレス.
- count: 整数型．データの個数.
- datatype: 送受信するデータの型．MPI_DOUBLE_PRECISION など.
- op: 最後に適用する演算の種類．MPI_SUM, MPI_PROD, MPI_MAX, MPI_MIN など.
- root: 演算結果の送り先.
- comm: コミュニケータ.
- ierr: エラーコード.

【復習】集団通信，**allreduce**

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

- mpi_reduce と同様だが，演算結果は全員に送られる．
- sendbuff: 送信するデータの先頭アドレス．
- recvbuff: 受信するデータの先頭アドレス．
- count: 整数型．データの個数．
- datatype: 送受信するデータの型．MPI_DOUBLE_PRECISION など．
- op: 最後に適用する演算の種類．MPI_SUM, MPI_PROD, MPI_MAX, MPI_MIN など．
- ~~root: 演算結果の送り先．~~ (これは不要)
- comm: コミュニケータ．
- ierr: エラーコード．

【復習】集団通信，broadcast

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

- root が持っている buff のデータを全員に配布し，それぞれの buff に格納.
- buff: root にとっては送信するデータの先頭アドレス．他にとっては受信したデータ格納先の先頭アドレス
- count: 整数型．データの個数.
- datatype: 送受信するデータの型．MPI_DOUBLE_PRECISION など.
- root: 誰のデータを配布するかを指定.
- comm: コミュニケータ.
- ierr: エラーコード.

【演習４】集団通信の利用

課題４

課題３のプログラムについて

- mpi_reduce を使って書き換えよ．ただし，計算結果の総和はプロセス 0 に送ることにする．
- mpi_bcast を用いてプロセス 0 から，求まった総和を全員に配布せよ．
- 各プロセスで総和を出力し，正しく計算できていることを確認せよ．

また，同様のプログラムを mpi_allreduce を用いて書いてみよ．

【復習】ノンブロッキング通信，送信側

send/recv は通信が終わるまで待っている ⇨ デッドロックの可能性.

mpi_isend(buff,count,datatype,dest,tag,comm,request,ierr)

- mpi_send と同様だが，通信が終わるまで待たずに次の作業に移る.
- ただし，通信終了確認のため mpi_wait/mpi_waitall を呼ぶ必要がある.
- これを呼ぶまでは送信したデータを変更してはいけない.

- buff: 送信するデータの先頭アドレス.
- count: 整数型. 送信するデータの数.
- datatype: 送信するデータの型. MPI_CHARACTER, MPI_INTEGER, MPI_DOUBLE_PRECISION など.
- dest: 整数型. 送信相手の番号.
- tag: 整数型. 送るデータを区別するための “整理番号”.
- comm: コミュニケータ. 特に作業グループを指定する必要がなければ MPI_COMM_WORLD.
- request: 整数型. 通信識別子. 送受信終了確認用.
- ierr: 整数型. エラーコード.

【復習】ノンブロッキング通信，受信側

mpi_irecv(buff,count,datatype,source,tag,comm,request,ierr)

- mpi_isend と同様に，通信が終わるまで待たずに次の作業に移る．
- 通信終了確認のため mpi_wait/mpi_waitall を呼ぶ必要がある．
- これを呼ぶまでは受信したデータを使用してはいけない．

- buff: 受信したデータ格納先の先頭アドレス．
- count: 整数型．受信するデータの数．
- datatype: 受信するデータの型．MPI_CHARACTER, MPI_INTEGER, MPI_DOUBLE_PRECISION など．
- source: 整数型．送り主の番号．
- tag: 整数型．送信時につけた“整理番号”．
- comm: コミュニケータ．特に作業グループを指定する必要がなければ MPI_COMM_WORLD．
- request: 整数型．通信識別子．送受信終了確認用．
- ierr: 整数型．エラーコード．

【復習】送受信確認

mpi_wait(request, status, ierr)

- isend/irecv の送受信を確認.
- request: 整数型．通信リクエスト．送受信時に設定したもの.
- status: 整数型の配列．isend しか行っていないプロセスも設定する.
- ierr: 整数型．エラーコード.

mpi_waitall(count, request, status, ierr)

- 複数の isend/irecv の送受信を確認.
- count: 同期する通信の数.
- request: 整数型の配列．通信識別子．送受信時に設定したもの．サイズは同期が必要な通信数.
- status: （整数型の配列）の配列．isend しか行っていないプロセスも設定する．サイズは (MPI_STATUS_SIZE, 同期が必要な通信数).
- ierr: 整数型．エラーコード.

【復習】送信と受信を一度に行うこともできる

mpi_sendrecv(sendbuff, sendcount, sendtype, dest, sendtag, recvbuff, recvcount, recvtype, source, recvtag, comm, status, ierr)

- sendbuff: 送信するデータの先頭アドレス.
- sendcount: 整数型. 送信するデータの数.
- sendtype: 送信するデータの型. MPI_DOUBLE_PRECISION など.
- dest: 整数型. 送信先. （source と同じである必要はない.）
- sendtag: 整数型. 送信するデータの “整理番号”.
- recvbuff: 受信したデータ格納先の先頭アドレス.
- recvcount: 整数型. 受信するデータの数.
- recvtype: 受信するデータの型. MPI_DOUBLE_PRECISION など.
- source: 整数型. 送り主の番号. （dest と同じである必要はない.）
- recvtag: 整数型. 受信するデータの “整理番号”.
- comm: コミュニケータ.
- status: 整数型の配列.
- ierr: 整数型. エラーコード.

【演習５】データのスワップ

課題５

次のページのプログラムについて，２プロセスで実行することにし，赤文字部分に適切な命令を入れることで，両プロセスがもっている配列 a の値を入れ替えたい．

- 赤文字部分に isend，irecv, wait, waitall などを挿入することで，配列 a の値を入れ替えよ．
- 同様に sendrecv を用いて配列 a の値を入れ替えよ．

（hint: プロセスが 0，1 だけのとき通信相手は 1-myrank）

演習５のプログラム

```
program main
use mpi
implicit none
integer :: i, nprocs, myrank, ierr
integer, parameter :: n=10
integer :: a(n)
call mpi_init(ierr)
call mpi_comm_size(MPI_COMM_WORLD,nprocs,ierr)
call mpi_comm_rank(MPI_COMM_WORLD,myrank,ierr)

do i=1,n
a(i) = i + myrank*10
end do
```

（この部分に命令を挿入）

```
do i=1,n
print *, myrank, i, a(i)
end do

call mpi_finalize(ierr)
end program main
```

（このファイルは fortran/swap.f90 にあります）

【演習６】πの数値計算

課題６

次のスライドのプログラムを並列化，つまり，

- MPI_Init など，必要な記述を加え，
- 和の取り方を適切に設定し，
- 時間測定のための記述を適切に挿入

して，１，２，４，８プロセスを用いた場合について計算時間を比較せよ．

演習６のプログラム

```
program pi
implicit none
integer, parameter :: n = 1000000
integer :: i
double precision :: x, dx, p

dx = 1.0d0/dble(n)

p = 0.0d0
do i = 1,n
x = dble(i) * dx
p = p + 4.0d0/(1.0d0 + x**2)*dx
end do

print *, p

end program
```

（このファイルは fortran/pi.f90 にあります）

C 言語編

【復習】MPI プログラムの基本構成

```
#include "mpi.h"                                    (mpi.h のインクルード)

int main(int argc, char **argv)
{
int nprocs, myrank;
MPI_Init(&argc,&argv);                              (MPI の初期化)
MPI_Comm_size(MPI_COMM_WORLD,&nprocs);              (全プロセス数を取得)
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);              (自分の番号を取得)

 (この部分を並列実行)

MPI_Finalize();                                     (MPI の終了処理)
return 0;
}
```

(この部分を並列実行)の部分は同じプログラムコード．myrank の値によって，うまく仕事が割り振られるように書く．

(このファイルは c/template.c にあります)

【復習】各命令の意味

- MPI_Init(&argc, &argv)
  - MPI の初期化を行う．プログラムの最初に必ず入れる．
- MPI_Comm_size(MPI_COMM_WORLD, &nprocs)
  - 全プロセス数を取得し，nprocs に保存する．
    - MPI_COMM_WORLD はコミュニケータの一種．
    - コミュニケータは作業グループを指定．
    - MPI_COMM_WORLD は全員からなる作業グループ．
    - これを別のものにすると，その作業グループのプロセス数を取得．
- MPI_Comm_rank(MPI_COMM_WORLD, &myrank)
  - 自分のプロセス番号を取得し，myrank に保存する．
  - MPI_COMM_WORLD 以外を指定すると，その作業グループでの番号を取得．
- MPI_Finalize()
  - MPI の終了処理を行う．プログラムの最後に必ず入れる．

基本的に前のスライドのプログラムと同じ順番で呼べば良い．

【演習 1 】Hello World!

課題 1

“hello world from （自分の番号）” を表示する並列プログラムを作成せよ．
つまり

- hello world のプログラムを並列化し，
- ついでに自分の番号（myrank）も表示するようにせよ．

プログラムを作成したらコンパイルし，４プロセスで実行してみよ．

４プロセスで実行した場合の実行例

```
hello world from 0
hello world from 2
hello world from 3
hello world from 1
```

- コンパイル方法

  mpifccpx hello.c
  （プログラムファイル名）

- 実行方法（後述）

  pjsub job.sh
  （スクリプトファイル名）

【重要】./a.out で実行しないこと！

大型計算機上でのプログラム実行：キューイングシステム

スパコンは共有財産！ ➡ ジョブの管理が必要

キューイングシステム

- 負荷状況・リソース使用量を監視し，ユーザが投入したジョブを適切な計算ノードに割り当て，実行するソフトウェア.
- 今回は富士通 Technical Computing Suite を使用.

プログラム実行の流れ

1. ジョブスクリプトを作成
2. ジョブを投入
3. （ジョブの状態を確認）
4. 結果を確認

```
./a.out
```

で実行するのでは ない.

スクリプトファイルの例

```
#!/bin/sh

#PJM -L "rscgrp=school"                （キューの指定，今回は school のみ）
#PJM -L "node=8"                       （ノード数=最大プロセス数の指定）
#PJM -L "elapse=3:00"                  （最大で３分）
#PJM -j                                （標準出力と標準エラー出力をまとめる）

mpiexec –n 4 ./a.out                   （プロセス数を指定して実行）
```

（このファイルは fortran/job.sh にあります）

- プロセス数を変える場合は赤字部分の数字を変える.
- 以下の課題でも，適宜，修正して使いまわして下さい.

ジョブの投入・状態確認・削除

- ジョブの投入

```
pjsub （ジョブスクリプト名）
```

- ジョブの状態確認

```
pjstat
```

- ジョブのキャンセル

```
pjdel （ジョブ番号）
```

【実行方法と結果の確認】

- Hello World のジョブを投入

```
pjsub job.sh
```

[INFO] PJM 0000 pjsub Job 1057 submitted. などと表示.

➡ **1057** の部分がジョブ番号.

- job.sh.oXXXX（XXXX はジョブ番号）などというファイルが作成され，その中に”hello world”（またはエラー）が出力されます.

【復習】 1 対 1 通信，送信側

int MPI_Send(void * buff, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

- buff: 送信するデータの先頭アドレス.
- count: 送信するデータの数.
- datatype: 送信するデータの型．MPI_CHAR, MPI_INT, MPI_DOUBLE など.
- dest: 送信相手の番号.
- tag: 送るデータを区別するための “整理番号”.
  （≃ 宅急便の取扱い番号．同一の相手と複数送受信するときの識別用.
  通常は 0 で良い）
- comm: コミュニケータ．特に作業グループを指定する必要がなければ MPI_COMM_WORLD.
- 返り値はエラーコード.

【復習】 1 対 1 通信，受信側

int MPI_Recv(void *buff, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

- buff: 受信したデータ格納先の先頭アドレス.
- count: 受信するデータの数.
- datatype: 受信するデータの型．MPI_CHAR, MPI_INT, MPI_DOUBLE など.
- source: 送り主の番号.
- tag: 送信時につけた “整理番号”.
- comm: コミュニケータ．特に作業グループを指定する必要がなければ MPI_COMM_WORLD.
- status: 状況オブジェクト．中身を使うことはあまりない.

  （送信側には含まれなかった引数なので忘れないように.）
- 返り値はエラーコード.

【演習２】Hello World (Part II)

課題２

プロセス 0 から受け取ったメッセージに自分の番号を追加して表示するプログラムを作成せよ．例えば

- プロセス 0 は “hello world from”という文字列（16 文字）を他のプロセスに送る．
- 他のプロセスは，この文字列を受け取り，自分の番号（myrank）を加えて表示する．

プログラムを作成したらコンパイルし，４プロセスで実行してみよ．

４プロセスで実行した場合の実行例

```
hello world from 2
hello world from 3
hello world from 1
```

【参考】文字列の取扱い

```
#include ”string.h”

char str[17];
strcpy(str, ”hello␣world␣from”);
strcpy(str, ””);
```

簡単な時間計測方法

double MPI_Wtime()

- 過去のある時刻からの経過時間を秒単位の実数値（double）で返す関数.
- 計測したい部分をこの関数で挟めば時間計測ができる.
- ただし，全員がその地点にたどり着いていることを保証するため，直前に MPI_Barrier をとる.

```
MPI_Barrier(MPI_COMM_WORLD)
time0 = MPI_Wtime()
（計測する部分）
MPI_Barrier(MPI_COMM_WORLD)
time1 = MPI_Wtime()
（time1-time0 を誰か（例えばプロセス 0）が出力）
```

int MPI_Barrier(comm)

- comm で指定した作業グループのメンバーは，そのグループ全員がその地点にたどり着くまで待つ.

【演習３】単純な内積の計算

課題３

次のスライドのプログラムは２つのベクトルの内積を計算するプログラムである．並列化されている（？）が，このままでは，全員，同じ計算をしているので意味がない．そこで，

- 各プロセスで部分和を計算して
- それを MPI_Send でプロセス 0 に送り，
- プロセス 0 で総和を求め，出力する

ように修正せよ．
また，MPI_Wtime を用いて，赤字部分に相当する総和計算部分の実行時間を計測できるように修正し，1, 2, 4 プロセスで実行した場合の実行時間を計測せよ．

- パラメータ n はプロセス数で割りきれると仮定して良い．
- このとき，n 個のデータを p 分割したときの第 i 区間は $[(i * n)/p, ((i+1) * n)/p - 1]$.

演習３のプログラム

```
#include ”mpi.h”
#include ”stdio.h”
#include ”math.h”

int main(int argc, char **argv)
{
int i,nprocs, myrank;
const int n=10000;
double v[n], w[n];
double res;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

for(i=0;i¡n;i++){
v[i] = sin(i*0.1);
w[i] = cos(i*0.1);
}

res = 0.0;
for(i=0;i¡n;i++){
res = res + v[i]*w[i];
}

printf(”%f␣\n”, res);
MPI_Finalize();
return 0;
}
```

（このファイルは **c/innerproduct.c** にあります）

【復習】集団通信，reduction

演習３での総和計算は専用の命令が存在.

```
int MPI_Reduce(void *sendbuff, void *recvbuff, int count, MPI_Datatype
datatype, MPI_Op op, int root, MPI_Comm comm)
```

- sendbuff: 送信するデータの先頭アドレス.
- recvbuff: 受信するデータの先頭アドレス.
- count: データの個数.
- datatype: 送受信するデータの型．MPI_DOUBLE など.
- op: 最後に適用する演算の種類．MPI_SUM, MPI_PROD, MPI_MAX, MPI_MIN など.
- root: 演算結果の送り先.
- comm: コミュニケータ.
- 返り値はエラーコード.

【復習】集団通信，allreduce

int MPI_Allreduce (void *sendbuff, void *recvbuff, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

- mpi_reduce と同様だが，演算結果は全員に送られる．
- sendbuff: 送信するデータの先頭アドレス．
- recvbuff: 受信するデータの先頭アドレス．
- count: データの個数．
- datatype: 送受信するデータの型．MPI_DOUBLE など．
- op: 最後に適用する演算の種類．MPI_SUM, MPI_PROD, MPI_MAX, MPI_MIN など．
- ~~root: 演算結果の送り先．~~ (これは不要)
- comm: コミュニケータ．
- 返り値はエラーコード．

【復習】集団通信，broadcast

int MPI_Bcast(void *buff, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

- root が持っている buff のデータを全員に配布し，それぞれの buff に格納.
- buff: root にとっては送信するデータの先頭アドレス．他にとっては受信したデータ格納先の先頭アドレス
- count: 整数型．データの個数.
- datatype: 送受信するデータの型．MPI_DOUBLE など.
- root: 誰のデータを配布するかを指定.
- comm: コミュニケータ.
- 返り値はエラーコード.

【演習４】集団通信の利用

課題４

課題３のプログラムについて

- MPI_Reduce を使って書き換えよ．ただし，計算結果の総和はプロセス 0 に送ることにする．
- MPI_Bcast を用いてプロセス 0 から，求まった総和を全員に配布せよ．
- 各プロセスで総和を出力し，正しく計算できていることを確認せよ．

また，同様のプログラムを MPI_Allreduce を用いて書いてみよ．

【復習】ノンブロッキング通信，送信側

Send/Recv は通信が終わるまで待っている ⇨ デッドロックの可能性.

```
int MPI_Isend(void *buff, int count, MPI_Datatype datatype, int dest, int tag,
              MPI_Comm comm, MPI_Request *request)
```

- MPI_Send と同様だが，通信が終わるまで待たずに次の作業に移る.
- 通信終了確認のため MPI_Wait/MPI_Waitall を呼ぶ必要がある.
- これを呼ぶまでは送信したデータを変更してはいけない.

- buff: 送信するデータの先頭アドレス.
- count: 送信するデータの数.
- datatype: 送信するデータの型．MPI_DOUBLE など.
- dest: 送信相手の番号.
- tag: 送るデータを区別するための “整理番号”.
- comm: コミュニケータ．特に作業グループを指定する必要がなければ MPI_COMM_WORLD.
- request: 通信識別子．送受信終了確認用.
- 返り値はエラーコード.

【復習】ノンブロッキング通信，受信側

int MPI_Irecv(void *buff, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

- MPI_Isend と同様に，通信が終わるまで待たずに次の作業に移る．
- 通信終了確認のため MPI_Wait/MPI_Waitall を呼ぶ必要がある．
- これを呼ぶまでは受信したデータを使用してはいけない．

- buff: 受信したデータ格納先の先頭アドレス．
- count: 受信するデータの数．
- datatype: 受信するデータの型．MPI_DOUBLE など．
- source: 送り主の番号．
- tag: 送信時につけた “整理番号”．
- comm: コミュニケータ．特に作業グループを指定する必要がなければ MPI_COMM_WORLD．
- request: 通信識別子．送受信終了確認用．
- 返り値はエラーコード．

【復習】送受信確認

int MPI_Wait(MPI_Request *request, MPI_Status *status)

- isend/irecv の送受信を確認.
- request: 通信リクエスト．送受信時に設定したもの.
- status: isend しか行っていないプロセスも設定する.
- 返り値はエラーコード.

int MPI_Waitall(int count, MPI_Request request[], MPI_Status status[])

- 複数の isend/irecv の送受信を確認.
- count: 同期する通信の数.
- request: 通信識別子の配列．送受信時に設定したもの．サイズは同期が必要な通信数.
- status: 状況オブジェクトの配列．isend しか行っていないプロセスも設定する．サイズは同期が必要な通信数.
- 返り値はエラーコード.

【復習】送信と受信を一度に行うこともできる

int MPI_Sendrecv(void *sendbuff, int sendcount, MPI_Datatype sendtype, int dest, int sendtag, void *recvbuff, int recvcount, MPI_Datatype recvtype, int source, int recvtag, MPI_Comm comm, MPI_Status *status)

- sendbuff: 送信するデータの先頭アドレス.
- sendcount: 送信するデータの数.
- sendtype: 送信するデータの型. MPI_DOUBLE など.
- dest: 送信先. （source と同じである必要はない.）
- sendtag: 送信するデータの “整理番号”.
- recvbuff: 受信したデータ格納先の先頭アドレス.
- recvcount: 受信するデータの数.
- recvtype: 受信するデータの型. MPI_DOUBLE など.
- source: 送り主の番号. （dest と同じである必要はない.）
- recvtag: 受信するデータの “整理番号”.
- comm: コミュニケータ.
- status: 状況オブジェクト.
- 返り値はエラーコード.

【演習５】データのスワップ

課題５

次のページのプログラムについて，２プロセスで実行することにし，赤文字部分に適切な命令を入れることで，両プロセスがもっている配列 a の値を入れ替えたい．

- 赤文字部分に Isend，Irecv, Wait, Waitall などを挿入することで，配列 a の値を入れ替えよ．
- 同様に Sendrecv を用いて配列 a の値を入れ替えよ．

（hint: プロセスが 0，1 だけのとき通信相手は 1-myrank）

演習５のプログラム

```
#include ”mpi.h”
#include ”stdio.h”
#include ”math.h”

int main(int argc, char **argv)
{
int i,nprocs, myrank;
const int n=10;
int a[n];
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

for(i=0;i<n;i++){
a[i] = i + myrank*10;
}

（この部分に命令を挿入）

for(i=0;i<n;i++){
printf(”%d, %d, %d \n”, myrank, i, a[i]);
}

MPI_Finalize();
return 0;
}
```

（このファイルは **c/swap.c** にあります）

【演習６】πの数値計算

課題６

次のスライドのプログラムを並列化，つまり，

- MPI_Init など，必要な記述を加え，
- 和の取り方を適切に設定し，
- 時間測定のための記述を適切に挿入

して，１，２，４，８プロセスを用いた場合について計算時間を比較せよ．

演習６のプログラム

```
#include ”stdio.h”

int main(int argc, char **argv)
{
const int n=1000000;
int i;
double x, dx, p;

dx = 1.0/(double)n;

p = 0.0;
for(i=0;i<n;i++){
x = (double)i * dx;
p = p + 4.0 / (1 + x*x)*dx;
}

printf(”%16.14f\n”, p);

return 0;
}
```

（このファイルは c/pi.c にあります）