演習２：MPI初歩

ー 並列に計算する ー

2013年8月6日

神戸大学大学院システム情報学研究科 計算科学専攻
横川三津夫

MPI（メッセージ・パッシング・インターフェース）を使おう！

[演習２の内容]

- はじめの一歩
  - 課題１：“Hello, world”を並列に出力する．
  - 課題2：プロセス0からのメッセージを受け取る（1対1通信）．
- 部分に分けて計算しよう
  - 課題3：2個のベクトルの内積を求める．
  - 課題4：結果を配布する．
  - 課題5：並列数を変えてみよう．
- π（=3.141592…）を計算しよう．
  - 課題6： $\frac{\pi}{4} = \int_0^1 \frac{1}{1+x^2} dx$

MPIの基本的な関数（サブルーチン）を復習しながら進めます．解説では，Fortran言語によるプログラムを用います．C言語版の資料は後半にあります．

※ 演習用のプログラムファイルは，以下のディレクトリに置いてありますので，利用して下さい．

```
Fortran言語版 /tmp/school/ex2/fortran/
C言語版       /tmp/school/ex2/c/
```

Fortran言語編

【復習】 MPIプログラムの基本構成

```
program main
use mpi                                                    MPIを使うおまじない
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
```

- それぞれのプロセスが何の計算をするかは，myrankの値で場合分けし，うまく仕事が割り振られるようにする.

（このファイルは，/tmp/school/ex2/fortran/skelton.f90 にあります）

【復習】 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の全プロセス数を取得し，2番目の引数 nprocs（整数型）に取得する．
  - MPI_COMM_WORLDはコミュニケータと呼ばれ，最初に割り当てられるすべてのプロセスの集合
- `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）に取得する．
- `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．

【課題1】 “Hello, world”を並列に出力する.

プログラム hello.f90を利用し，“Hello, world from（プロセス番号）”を表示する並列プログラムを作成しなさい.

① hello.f90 に，2ページ「MPIプログラムの基本構成」を参考にし，並列化する.

② print文に，自分のプロセス番号（myrank）を出力するよう修正する.

③ プログラムを作成したら，コンパイルし，4プロセスで実行せよ.

【実行結果の例】
※ 必ずしも，プロセスの順番に出力されるとは限らない.

```
Hello, world from      1
Hello, world from      0
Hello, world from      3
Hello, world from      2
```

（このファイルは，/tmp/school/ex2/fortran/hello.f90 にあります）

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（修正）

② プログラムのコンパイル：
```
# mpifrtpx  xxx.f90
```
← 入力コマンドです.
- a.out が作られる.

③ジョブスクリプトを作成：　エディタを用いて，job.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.

ジョブスクリプトファイル例（job.sh）

```
#!/bin/bash                          ← おまじない１

#PJM -L “rscgrp=school”              ← おまじない２： 実行キューの指定
#PJM -L “node=8”                     ← 利用する計算ノード数（24まで可能）
#PJM -L “elapse=00:03:00”            ← 計算時間の予測値（実際の計算時間よりも少し大きく）

mpiexec -n 4 ./a.out                 ← MPIプログラムの実行.
                                        - この例では，MPIプロセス数が4であることに注意
```

※ 他の課題においても，このジョブスクリプトを適宜，修正のこと.

（このファイルは，/tmp/school/ex2/fortran/job.sh にあります）

【復習】 1対1通信 – 送信関数（送り出し側）

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

- 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_DOUBLE_PRECISION, MPI_CHARACTER など
- dest:　　送信先のプロセス番号
- tag:　　メッセージ識別番号．送るデータを区別するための番号
- comm:　　コミュニケータ（例えば，MPI_COMM_WORLD）
- ierr:　　戻りコード（整数型）

【復習】 1対1通信 – 受信関数（受け取り側）

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

- 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_DOUBLE_PRECISION, MPI_CHARACTER など
- source: 送信してくる相手のプロセス番号
- tag: メッセージ識別番号．送られて来たデータを区別するための番号
- 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）

【課題2】 プロセス0からのメッセージを受け取る（1対1通信）

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

- プロセス0の処理と他のプロセスの処理は，myrankの値による場合分けを行う.

【プロセス番号 0の処理】
① 送るメッセージを作る．メッセージの長さはきめておく.
　- 例えば，”Hello, world from”（17文字）
② mpi_send関数で他のプロセスにメッセージを送る.

【他のプロセスの処理】
① プロセス番号0から送られてくるメッセージを，mpi_recv関数で受け取る.
② 送られたメッセージに自分のプロセス番号を追加して出力する.

【実行結果の例】

```
Hello, world from      1
Hello, world from      3
Hello, world from      2
```

【課題３】 2個のベクトルの内積を求める.

次のプログラムは，2個のベクトルの内積を計算するプログラムである．並列化せよ．
また，並列化したプログラムにおいて，赤で示した部分に相当する部分の計算時間を計測せよ．

```
program InnerProduct
implicit none
integer :: i
integer, parameter :: n=10000
real(kind=8) :: v(n), w(n)
real(kind=8) :: ipr

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

ipr = 0.0d0
do i = 1, n
  ipr = ipr + v(i)*w(i)
enddo

print *, ipr

end program InnerProduct
```

ベクトルの長さ（定数）
２個のベクトルを格納する配列の宣言

２個のベクトルを設定
倍精度のsin関数
倍精度のcos関数

２個のベクトルの内積を計算

計算結果の出力

（このファイルは，/tmp/school/ex2/fortran/ipr.f90 にあります）

【課題３のヒント】

① 各プロセスで部分和を計算する．
n個のデータをp個のプロセスで分割した時の第iプロセス（iは0からp-1）の部分和の区間は，[(i*n)/p+1，((i+1)*n)/p]になる．ただし，この演習では，nはpで割り切れるものとする．第iプロセスの番号は，myrankと同じになることに注意．

例えば，n=10000 のベクトルを４個のプロセスで計算する場合
- プロセス０：　1- 2500の部分和を計算
- プロセス１：2501- 5000の部分和を計算
- プロセス２：5001- 7500の部分和を計算
- プロセス３：7501-10000の部分和を計算

② プロセス１，２，３は，mpi_send関数で，部分和をプロセス０に送る．

③ プロセス０は，他のプロセスから送られた部分和を mpi_recv関数で受け取り，自分の部分和と足し合わせ，最終的な総和を計算し，出力する．

プロセス０の処理と他のプロセスの処理は，myrankの値による場合分けを行う．

【課題３のヒント】 計算時間の計測をする方法

```
real(kind=8) :: time0, time2                    計測のための変数を倍精度実数で宣言する.
      •
      •
call mpi_barrier( MPI_COMM_WORLD, ierr )      MPI_barrier関数で，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 )
```

- 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()
```

【復習】 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rootが集め，演算（op）を適用する．

- 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_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
- 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）

【復習】 集団通信 － broadcast

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

※ rootが持つbuffの値を，commで指定された他のプロセスの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_DOUBLE_PRECISION, MPI_CHARACTER 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）

【復習】 集団通信 － reduction

```
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_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）

【課題４】 結果を配布する

【課題4-1】

課題３で作成した並列プログラムについて，

① mpi_reduceを使って書き換えなさい．ただし，計算結果の総和は，プロセス０で求めることにすること．

② mpi_bcastを用いてプロセス０で求めた総和を，すべてのプロセスに配布しなさい．

③ 各プロセスで総和を出力し，正しく計算が出来ていることを確認しなさい．

【課題4-2】

課題4-1と同様のことを，mpi_allreduceを使って書き換え，結果が正しいことを確認しなさい．

【課題５】 並列数を変えてみよう.

並列計算の目的は，計算時間を短縮することである.

課題4-1で作成したプログラムを用い，並列数を変えて計算時間を計測しなさい．また，以下の表を完成させ，どのくらい計算時間が短縮されるか確かめなさい.

並列数 n	計算時間 T(n)（秒）	速度向上率 T(n)/T(1)（n=1の計算時間T(1)との比）
1		1.0
2		
4		
8		
16		

【課題６】 π（=3.141592…）を計算しよう.

次のπを求めるプログラムを並列化しなさい．また，並列数を変えて時間を計測し，課題５で作成した表と同様の表を作成しなさい.

```
program pi
implicit none
integer, parameter :: n=1000000
integer :: i
real(kind=8) :: 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*x)*dx
enddo

print *, p

end program pi
```

$$\pi = \sum \frac{4}{1+x^2}\Delta x$$

総和部分を部分和に分けて，最終的に総和を求めるプログラムにする.

（このファイルは，/tmp/school/ex2/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を使うおまじない

int main( int argc, char **argv )
{
  int  nprocs, myrank;                                    MPIで使う変数の宣言

  MPI_init( &argc, &argv );                               MPIの初期化（おまじない２）
  MPI_Comm_size( MPI_COMM_WORLD, &nprocs );               MPIで使うプロセス数を nprocs に取得
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );               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）


  MPI_Finalize() ;                                        MPIの終了処理（おまじない３）
  return 0 ;
}
```

- それぞれのプロセスが何の計算をするかは，myrankの値で場合分けし，うまく仕事が割り振られるようにする.

（このファイルは，/tmp/school/ex2/c/skelton.c にあります）

【復習】 MPIプログラムの基本構成（説明）

- `int MPI_Init(int *argc, char ***argv )`
  - MPIの初期化を行う．MPIプログラムの最初に必ず書く．
- `int MPI_Comm_size( MPI_Comm comm, int *nprocs )`
  - MPIの全プロセス数を取得し，2番目の引数 nprocs（整数型）に取得する．
  - MPI_COMM_WORLDはコミュニケータと呼ばれ，最初に割り当てられるすべてのプロセスの集合
- `int MPI_Comm_rank( MPI_Comm comm, int *myrank )`
  - 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）に取得する．
- `int MPI_Finalize(void)`
  - MPIの終了処理をする．MPIプログラムの最後に必ず書く．

【課題1】 “Hello, world”を並列に出力する.

プログラム hello.cを利用し，“Hello, world from（プロセス番号）”を表示する並列プログラムを作成しなさい.

① hello.c に，21ページ「MPIプログラムの基本構成」を参考にし，並列化する.

② printf文に，自分のプロセス番号（myrank）を出力するよう修正する.

③ プログラムを作成したら，コンパイルし，4プロセスで実行せよ.

【実行結果の例】
※ 必ずしも，プロセスの順番に出力されるとは限らない.

```
Hello, world from      2
Hello, world from      3
Hello, world from      0
Hello, world from      1
```

（このファイルは，/tmp/school/ex2/c/hello.c にあります）

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.c）を作成（修正）

② プログラムのコンパイル：
```
# mpifccpx xxx.c -lmpi (-lm)
```
← 入力コマンドです.
- a.out が作られる.

③ジョブスクリプトを作成： エディタを用いて，job.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.

ジョブスクリプトファイル例（job.sh）

```
#!/bin/bash                          ← おまじない１

#PJM -L “rscgrp=school”              ← おまじない２： 実行キューの指定
#PJM -L “node=8”                     ← 利用する計算ノード数（24まで可能）
#PJM -L “elapse=00:03:00”            ← 計算時間の予測値（実際の計算時間よりも少し大きく）

mpiexec -n 4 ./a.out                 ← MPIプログラムの実行.
                                        - この例では，MPIプロセス数が4であることに注意
```

※ 他の課題においても，このジョブスクリプトを適宜，修正のこと.

（このファイルは，/tmp/school/ex2/c/job.sh にあります）

【復習】 1対1通信 – 送信関数（送り出し側）

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

- buff:　　送信するデータの変数名（先頭アドレス）
- count:　　送信するデータの個数
- datatype:　送信するデータの型
  - MPI_CHAR, MPI_INT, MPI_DOUBLE など
- dest:　　送信先のMPIプロセス番号
- tag:　　メッセージ識別番号．送るデータを区別するための番号
- comm:　　コミュニケータ（例えば，MPI_COMM_WORLD）
- 関数の戻りコードは，エラーコード

【復習】 1対1通信 – 受信関数（受け取り側）

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

- buff: 送信するデータの変数名（先頭アドレス）
- count: 送信するデータの個数
- datatype: 送信するデータの型
  - MPI_CHAR, MPI_INT, MPI_DOUBLE など
- source: 送信先のMPIプロセス番号
- tag: メッセージ識別番号．送るデータを区別するための番号
- comm: コミュニケータ（例えば，MPI_COMM_WORLD）
- status: 状況オブジェクト．MPI_Sendにはないので注意．
- 関数の戻りコードは，エラーコード

【課題2】 プロセス0からのメッセージを受け取る（1対1通信）

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

- プロセス0の処理と他のプロセスの処理は，myrankの値による場合分けを行う.

【プロセス0の処理】
① 送るメッセージを作る．メッセージの長さはきめておく.
　– 例えば，”Hello, world from”（17文字）
② MPI_Send関数で他のプロセスにメッセージを送る.

【他のプロセスの処理】
① プロセス0から送られてくるメッセージを，MPI_Recv関数で受け取る.
② 送られたメッセージに自分のプロセス番号を追加して出力する.

【実行結果の例】

```
Hello, world from      1
Hello, world from      3
Hello, world from      2
```

【参考】 文字列の扱い

```
#include “string.h”

char  str[18];
strcpy( str, “Hello, world from” ) ;
```

【課題３】 2個のベクトルの内積を求める.

次のプログラムは，2個のベクトルの内積を計算するプログラムである．並列化せよ．
また，並列化したプログラムにおいて，赤で示した部分に相当する部分の計算時間を計測せよ．

```
#include “stdio.h”
#inlucde “math.h”

int main( int argc, char **argv )
{
  int i;
  const int n=10000;                          ベクトルの長さ（定数）
  double v[n], w[n];                          ２個のベクトルを格納する配列の宣言
  double ipr;

  for( i=0; i<n; i++ ){                       ２個のベクトルを設定
    v[i] = sin( 0.1*(i+1) );                    倍精度のsin関数
    w[i] = cos( 0.1*(i+1) );                    倍精度のcos関数
  }

  for( ipr=0.0, i=0; i<n; i++ ){
    ipr += v[i]*v[i]                          ２個のベクトルの内積を計算
  }

  printf( “Inner product = %20.15lf¥n”, ipr);
  return 0;                                   計算結果の出力

}
```

（このファイルは，/tmp/school/ex2/c/ipr.c にあります）

【課題３のヒント】

① 各プロセスで部分和を計算する．
n個のデータをp個のプロセスで分割した時の第iプロセス（iは0からp-1）の部分和の区間は，[(i*n)/p+1，((i+1)*n)/p]になる．ただし，この演習では，nはpで割り切れるものとする．第iプロセスは，myrankと同じ値であることに注意．

例えば，n=10000 のベクトルを４個のプロセスで計算する場合
- プロセス０：　　1- 2500の部分和を計算
- プロセス１：　2501- 5000の部分和を計算
- プロセス２：　5001- 7500の部分和を計算
- プロセス３：　7501-10000の部分和を計算

② プロセス１，２，３は，MPI_Send関数で，部分和をプロセス０に送る．

③ プロセス０は，他のプロセスから送られた部分和をMPI_Recv関数で受け取り，自分の部分和と足し合わせ，最終的な総和を計算し，出力する．

プロセス０の処理と他のプロセスの処理は，myrankの値による場合分けを行う．

【課題３のヒント】 計算時間の計測をする方法

```
double time0, time2 ;                          計測のための変数を倍精度実数で宣言する.
      •
      •
MPI_Barrier( MPI_COMM_WORLD );                 MPI_Barrier関数で，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)

MPI_Barrier( MPI_COMM_WORLD );                 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.
```

`int MPI_Barrier(MPI_Comm comm)` : コミュニケータ間で同期を取る.

- comm: コミュニケータ（例えば，MPI_COMM_WORLD）

`double MPI_Wtime(void)` : timeの現行値（秒数）を倍精度浮動小数点で返す

【復習】 集団通信 － reduction

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

※ commで指定されたすべてのプロセスからデータをrootが集め，演算（op）を適用する．

- 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
- datatype:　送信するデータの型
  - MPI_INT, MPI_DOUBLE, MPI_CHAR 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）
- 関数の戻りコードは，エラーコードを表す．

【復習】 集団通信 － broadcast

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

※ rootが持つbuffの値を，commで指定された他のプロセスの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
- datatype: 送信するデータの型
  - MPI_INT, MPI_DOUBLE, MPI_CHAR 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）
- 関数の戻りコードは，エラーコードを表す．

【復習】 集団通信 － reduction

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

※ 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
- datatype:　送信するデータの型
  - MPI_INT, MPI_DOUBLE, MPI_CHAR 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）
- 関数の戻りコードは，エラーコードを表す．

【課題４】 結果を配布する

【課題4-1】

課題３で作成した並列プログラムについて，

① MPI_Reduceを使って書き換えなさい．ただし，計算結果の総和は，プロセス０で求めることにすること．

② MPI_Bcastを用いてプロセス０で求めた総和を，すべてのプロセスに配布しなさい．

③ 各プロセスで総和を出力し，正しく計算が出来ていることを確認しなさい．

【課題4-2】

課題4-1と同様のことを，MPI_Allreduceを使って書き換え，結果が正しいことを確認しなさい．

【課題５】 並列数を変えてみよう.

並列計算の目的は，計算時間を短縮することである.

課題4-1で作成したプログラムを用い，並列数を変えて計算時間を計測しなさい．また，以下の表を完成させ，どのくらい計算時間が短縮されるか確かめなさい.

並列数 n	計算時間 T(n)（秒）	速度向上率 T(n)/T(1)（n=1の計算時間T(1)との比）
1		1.0
2		
4		
8		
16		

【課題６】 π（=3.141592…）を計算しよう.

次のπを求めるプログラムを並列化しなさい．また，並列数を変えて時間を計測し，課題５で作成した表と同様の表を作成しなさい.

```
#include “stdio.h”

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

   dx = 1.0/n;

   for( pi=0.0, i=1; i<=n; i++ ){
      x = i*dx;
      pi += 4.0*dx/(1.0+x*x);
   }

   printf(“pi = %20.14lf^n”, pi );
   return 0;
}
```

$$\pi = \sum \frac{4}{1+x^2}\Delta x$$

- 総和部分を部分和に分けて，最終的に総和を求めるプログラムにする.

（このファイルは，/tmp/school/ex2/c/pi.c にあります）