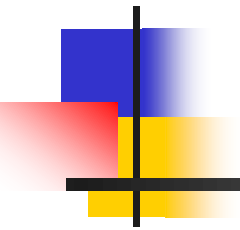
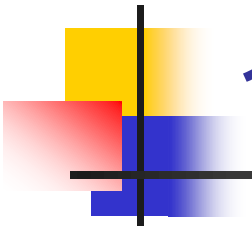


計算科学演習I　第8回講義
「MPIを用いた並列計算(I)」

2013年6月6日

システム情報学研究科 計算科学専攻
山本有作

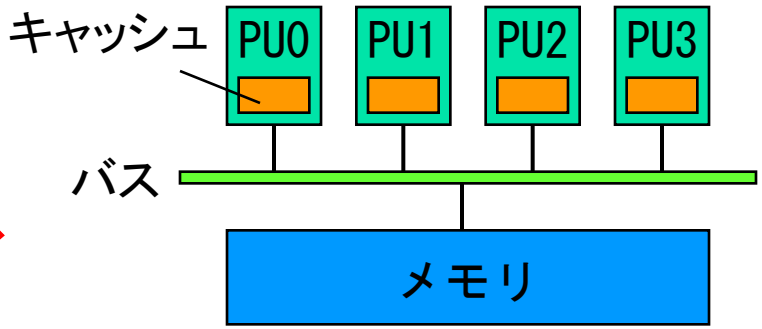


今回の講義の概要

1. MPI とは
2. 簡単な MPI プログラムの例（1）
3. 簡単な MPI プログラムの例（2）：1対1通信
4. 簡単な MPI プログラムの例（3）：集団通信

共有メモリ型並列計算機(復習)

- 共有メモリ型並列計算機
  - 複数のプロセッサ(PU)がバスを通してメモリを共有
  - **どのPUも同じメモリ領域にアクセスできる**

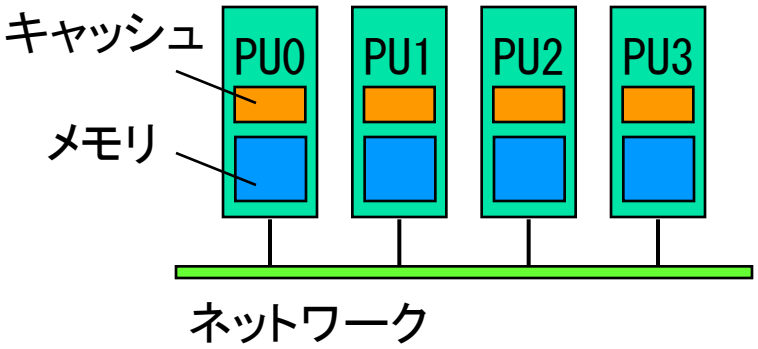


- 特徴
  - メモリ空間が単一のためプログラミングが容易
  - PUの数が多すぎると，アクセス競合により性能が低下
    → 2～16台程度の並列が多い
- プログラミング言語
  - **OpenMP** (FORTRAN/C/C++ ＋ 指示文)を使用
    ※メモリ領域を分割し，MPI を利用することも可能

前回，前々回

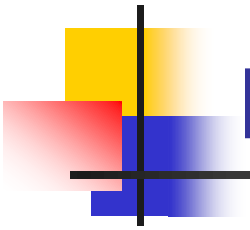
分散メモリ型並列計算機(復習)

- アーキテクチャ
  - 各々がメモリを持つ複数のPUを ネットワークで接続
  - **各PUはそれぞれ自分の持つメモリのみにアクセス可能**



- 特徴
  - 数千～数万PU規模の並列が可能
  - PU間へのデータ分散を意識したプログラミングが必要
- プログラミング言語
  - FORTRAN/C/C++ ＋ **MPI** を使用

今回～次々回



MPIとは

- Message Passing Interface

  ※余談ですが，OpenMP：Open Multi-Processing

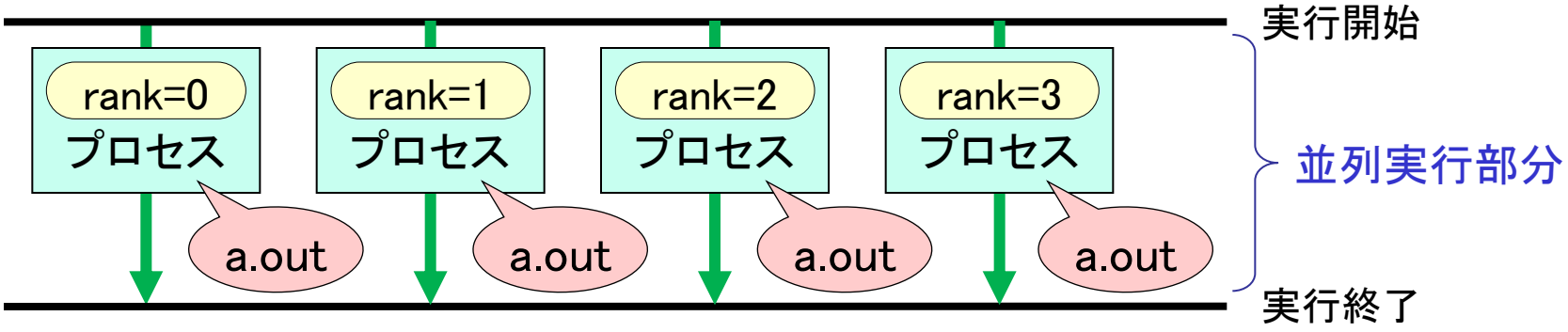
- 分散メモリ型並列計算機上での並列プログラミングのための
  プロセッサ間通信の標準規格（ライブラリを指すこともある）
- 1992年頃より米国の計算機メーカー・大学を中心に標準化
- MPI の規格と歴史
  - 1994　MPI-1
  - 1997　MPI-2
  - 2012　MPI-3

    http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

    ※852ページ（2.7MB）

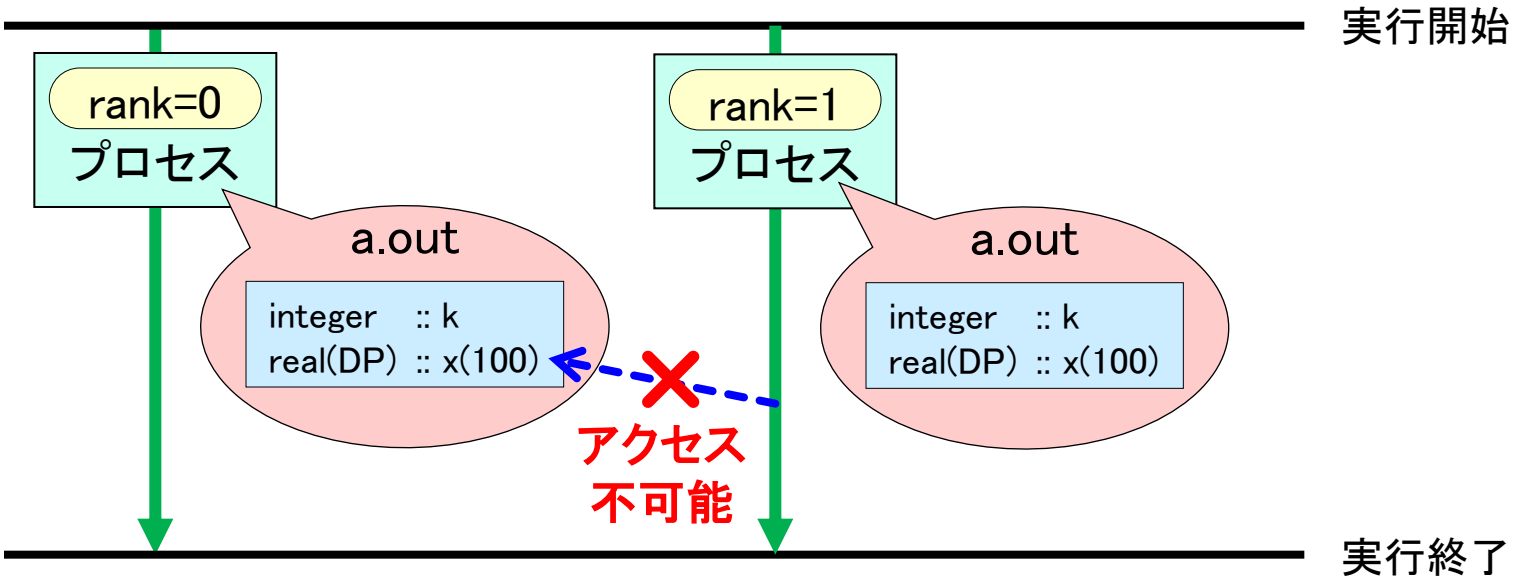
MPIの実行モデル(1/3)

- 実行モデル
  - SPMD : Single Program Multiple Data
  - 複数の**プロセス**により並列実行　※OpenMPでは複数スレッド
  - 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**
  - 各プロセスは**固有のプロセス番号(ランク)**を持つ
    - P 個のプロセスで実行するとき，ランクは 0 から P-1 までの整数
  - 演算対象データやIF文による分岐などは，プロセスごとに異なってよい



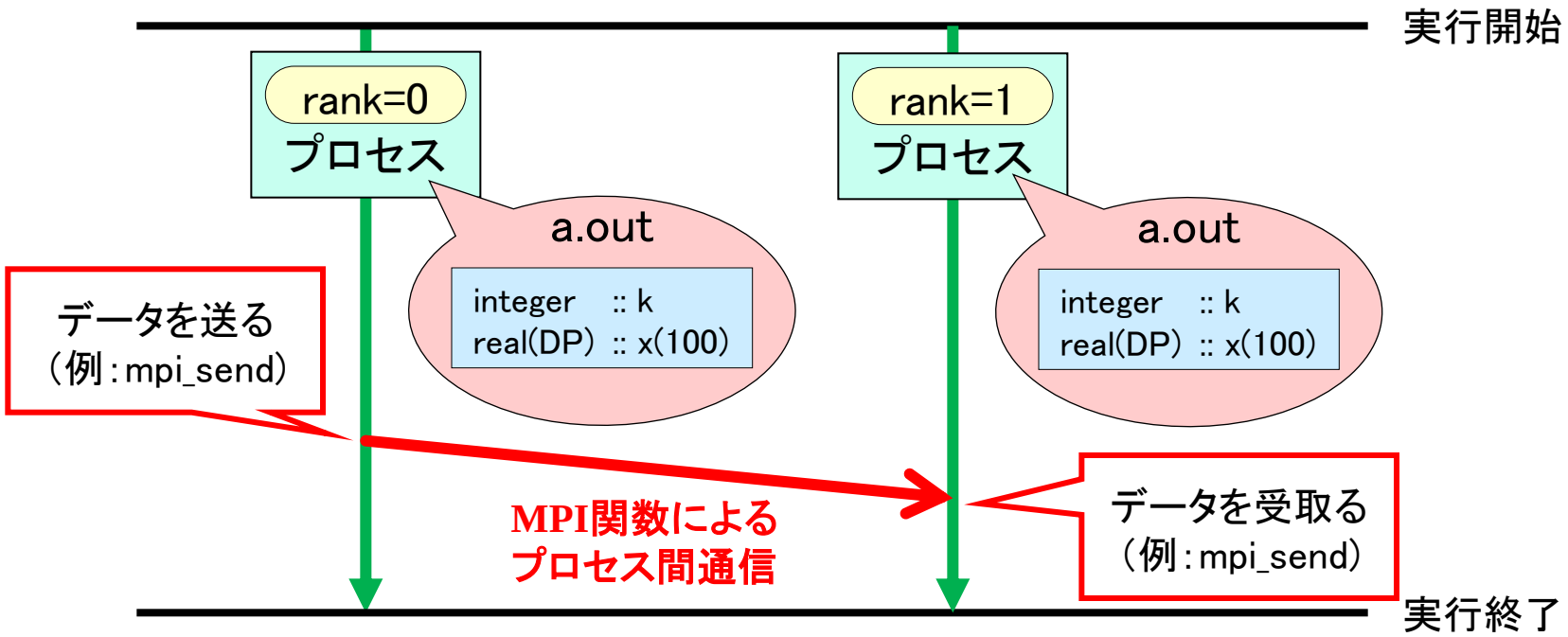
MPIの実行モデル(2/3)

- メモリ空間
  - プロセスごとに**独立したメモリ空間**を保持
    - プログラム中で定義された変数・配列は，同じ名前で独立に各プロセスのメモリ上に割り当てられる
    - 同じ変数・配列に対して，プロセスごとに違う値を与えることは可能
    - 他のプロセスの持つ変数・配列にはアクセスできない



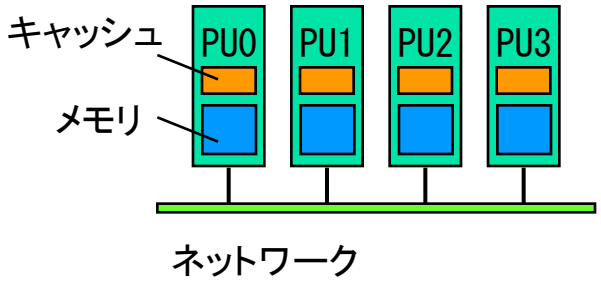
MPIの実行モデル(3/3)

- プロセス間通信
  - 他のプロセスの持つ変数・配列のデータにアクセスできない
    ⇒プロセス間通信により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(データ)の送り手と受け手
  - この方式によるプロセス間通信関数の集合 ≒ MPI

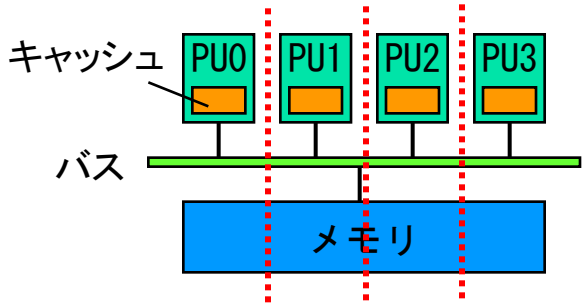


MPIの実行環境

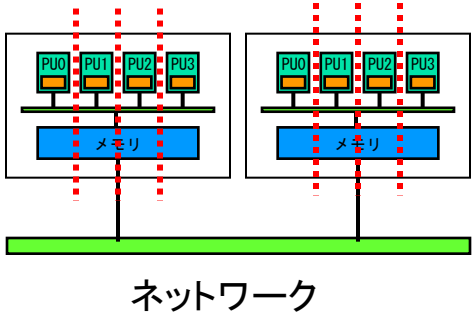
- 分散メモリ型並列計算機
  - MPI がもともと想定している環境
  - 各プロセスが1個のPUに割り当てられる



- 共有メモリ型並列計算機
  - メモリを分割して各PUに割り当て，仮想的な分散メモリ環境を作る
  - 既存のMPIプログラムを共有メモリ型並列計算機で実行するときに便利

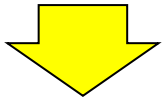


- SMP クラスタ
  - 各ノードのメモリを分割して各PUに割り当て，仮想的な分散メモリ環境を作る
  - 本講義でもこの環境を用いる

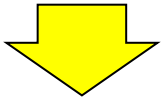


MPI プログラムの実行までの一例

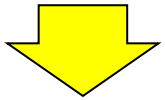
- （元となる）逐次版のプログラム
  - 通常の FORTRAN または C/C++ で書かれたプログラム



- **MPIプログラム**
  - 通常のFORTRAN または C/C++ で書かれたプログラム
  - 分散環境を意識した修正（データ分散や処理の分岐など）
  - MPI関数の追加（プロセス間通信など）



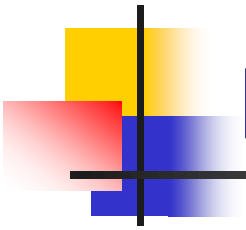
- MPIプログラム用のコンパイラでコンパイル



- MPIプログラム用の実行コマンドで実行
  - プログラムの実行に用いるプロセス数を指定

MPI プログラムの構成要素

- プログラムのベース
  - 通常の FORTRAN または C/C++ で書かれたプログラム
  - 分散メモリを意識して書かれている必要あり
    - 処理の範囲，保持するデータの範囲，処理の分岐，…
- MPI関数
  - 環境設定・環境取得のための関数
    - 初期化，終了処理，総プロセス数の取得，ランクの取得，…
  - 通信関数
    - 1対1通信，集団通信
  - 基本データ型・定義済み演算
    - 通信時のデータ型の指定，リダクション演算（後述）の指定，…
  - インクルードファイル
    - プログラムの最初で use mpi と記述



MPI の特徴

- 移植性
  - MPI でプログラムを書くことで，様々な分散メモリ型並列計算機での実行が可能
  - 共有メモリ型並列計算機でも実行可能
- スケーラビリティー
  - 適切なアルゴリズムに基づいて並列プログラムを書けば，数千～数万個のプロセッサを持つ並列計算機の能力を引き出すことが可能
- 多様な集団通信
  - 総和，ブロードキャストなど多様な通信関数が予め用意されている
- 性能解析ツール
  - 負荷分散，通信時間等の詳細な情報を取得するツールが使用可能

現在利用可能な MPI の実装

- MPICH
  - 米国のアルゴンヌ国立研究所が提供しているフリーな MPI
  - 現在では MPICH2 が主流
  - http://www.mcs.anl.gov/research/projects/mpich2/
- OpenMPI
  - MPI に関する様々な先行プロジェクトの成果を基に開発された MPI
  - http://www.open-mpi.org/
- 各計算機メーカーの提供する MPI
  - 並列計算機メーカー各社が，自社の計算機向けに，それぞれ MPI の実装を提供している

    （例）Π コンピュータでは富士通製のMPIライブラリを使用

MPI プログラムの例 (1)

- 並列版 “Hello World”

```
program hello
  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 is', myrank
  call mpi_finalize(ierr)
end program hello
```

※青字がMPI関数など（詳細は後で説明）

MPIプログラムの実行方法(1/3)

- プログラムのコンパイル
  - 「**mpifrtpx**」コマンドを使用　※Fortranの場合
    - mpifrtpx hello.f90　　⇒　実行ファイルa.outができる
    - mpifrtpx –o hello hello.f90　　⇒　実行ファイルhelloができる

- プログラムの実行
  - キューイングシステムにより実行　※OpenMPの演習の時と同じ
    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. 実行結果を確認

注意

今回の演習の内容は神戸大のΠコンピュータを想定したものです。
使用する計算機システムによってコンパイルコマンド等は異なります。

MPIプログラムの実行方法(2/3)

- ジョブスクリプトの作成(4プロセスで実行する例)

```
#!/bin/bash                                  シェル名
#PJM -N "jobname"                            ジョブ名
#PJM -L "rscgrp=small"                       投入先のキュー
#PJM -L "node=4"                             使用ノード数
#PJM -L "elapse=2:00"                        最大実行時間
#PJM -j

mpiexec -n 4 ./a.out      並列数を指定してMPIプログラムを実行
```

重要

今回の演習では1ノードに1プロセスとして実行する。

⇒「使用ノード数＝使用プロセス数」としてスクリプトを作成する。

(注意)最大実行時間は必要以上に長く設定しない！

MPIプログラムの実行方法(3/3)

- ジョブの投入
  - pjsub （ジョブファイル名）
- ジョブの状態確認
  - pjstat

JOB_ID	JOB_NAME	MD	ST	USER	START_DATE	ELAPSE_LIM	NODE_REQUIRED
17583	job_1	NM	RUN	user	(06/06 12:00)	0000:02:00	4
17584	job_2	NM	QUE	user	(06/06 13:00)	0000:10:00	12

- ジョブのキャンセル
  - pjdel （ジョブ番号）
- 実行結果の確認
  - うまく実行できれば「ジョブ名.o?」（?はジョブ番号）というファイルが作成され，その中に実行結果が出力される。

演習1-1

- 並列版 “Hello World”を**2，4プロセス**で実行し，結果を確認せよ！

（1）今日の演習用のディレクトリ（例：mpi_20130606）を作成

```
mkdir mpi_20130606 ⇒ cd mpi_20130606
```

（2）演習1-1用のディレクトリ（例：enshu_1_1）を作成

```
mkdir enshu_1_1 ⇒ cd enshu_1_1
```

（3）ソースコードをコピーし，中身を確認 ※編集の必要はなし

```
cp /tmp/mpi_20130606/1_1/hello.f90 ./ ⇒ emacs等で確認
```

（4）ソースコードをコンパイル

```
mpifrtpx hello.f90
```

（5）ジョブスクリプトをコピーし，必要な部分（プロセス数）を編集

```
cp /tmp/mpi_20130606/1_1/go.sh ./ ⇒ emacs等で編集
```

（6）ジョブを投入し，実行結果を確認

```
pjsub go.sh ⇒ pjstat ⇒ xxxx.o?の中身を確認
```

演習1-1の実行結果

- 2プロセスでの実行結果

```
Hello, World. My rank is 0
Hello, World. My rank is 1
```

- 4プロセスでの実行結果

```
Hello, World. My rank is 0
Hello, World. My rank is 2
Hello, World. My rank is 3
Hello, World. My rank is 1
```

（注意）両者ともランク順に並ぶとは限らず，実行ごとに順番が異なることもある

ポイント

- 各プロセスが同じ処理（Hello World）を実行している。
- 各プロセスが持っているランク(myrankの値）が異なっている。

MPI プログラム (1)の解説(1/2)

- 並列版 “Hello World”(再掲)

```
program hello
  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 is', myrank
  call mpi_finalize(ierr)
end program hello
```

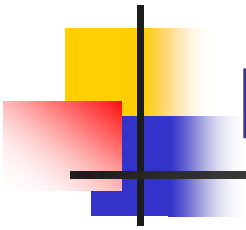
mpif.hのインクルード

初期化

総プロセス数を取得

自分のプロセス番号を取得

終了処理



MPI プログラム (1)の解説(2/2)

- **`mpi_init(ierr)`**
  - プログラムの最初にコールしてMPIの初期化処理を行う
- **`mpi_finalize(ierr)`**
  - プログラムの最後にコールしてMPIの終了処理を行う
- **`mpi_comm_world(comm, nprocs, ierr)`**
  - comm内の全プロセスがnprocsに入る
  - commはプロセスグループを指定する変数(コミュニケータと呼ぶ)
    (MPI_COMM_WORLDとすると全てのプロセスを含むグループを指す)
- **`mpi_comm_rank(comm, myrank, ierr)`**
  - comm内での自分のプロセス番号(ランク)がmyrankに入る

※ ierr はintegerの変数でエラーコードが入る

典型的なMPIプログラムの構成

```
program mpi_sample
   use mpi
   implicit none
   変数宣言
   call mpi_init(ierr)
   call mpi_comm_size(MPI_COMM_WORLD,nprocs,ierr)
   call mpi_comm_rank(MPI_COMM_WORLD,myrank,ierr)
```

- プロセスごとの処理（myrankに応じて分岐など）
- **プロセス間通信（MPI関数）** ※この後で説明

の繰り返し

```
   call mpi_finalize(ierr)
end program mpi_sample
```

MPI プログラムの例 (2):1対1通信

- 1から100までの整数の和を2並列で求めるプログラム

```
program sum100
  use mpi
  implicit none
  integer :: i,istart,iend,isum,isum1
  integer :: nprocs,myrank,ierr
  integer, dimension(MPI_STATUS_SIZE) :: istat
  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=0
  do i=istart, iend
     isum=isum+i
  end do
  if (myrank==1) then
     call mpi_send(isum,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,ierr)
  else
     call mpi_recv(isum1,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,istat,ierr)
  end if
  if (myrank==0) print *, 'sum =', isum+isum1
  call mpi_finalize(ierr)
end program sum100
```

MPI プログラム(2)の解説(1/4)

- 1から100までの整数の和

- 青:MPIプログラムのお約束
- 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
  use mpi
  implicit none
  integer :: i,istart,iend,isum,isum1
  integer :: nprocs,myrank,ierr
  integer, dimension(MPI_STATUS_SIZE) :: istat
  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=0
  do i=istart, iend
     isum=isum+i
  end do
  if (myrank==1) then
     call mpi_send(isum,1,MPI_INTEGER,0,100,MPI_COMM_WORLD,ierr)
  else
     call mpi_recv(isum1,1,MPI_INTEGER,1,100,MPI_COMM_WORLD,istat,ierr)
  end if
  if (myrank==0) print *, 'sum =', isum+isum1
  call mpi_finalize(ierr)
end program sum100
```

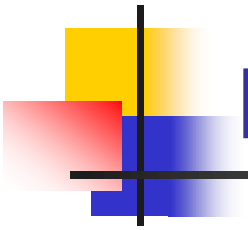
ランクの値から自分の計算範囲を求める

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

プロセス0は結果を出力



MPI プログラム(2)の解説（2/4）

- mpi_send
  - 他のプロセスにデータを送信
  - ブロッキング通信
    - データが送信バッファから送り出された後にリターンする
- mpi_recv
  - 他のプロセスからデータを受信
  - ブロッキング通信
    - データが受信バッファに完全に格納された後にリターンする

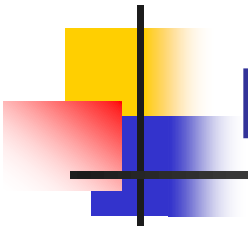
MPI プログラム(2)の解説(3/4)

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

```
buff      : 送信バッファの先頭アドレス
count     : 送信するデータの要素数
datatype  : 送信するデータの型
dest      : 送信相手のプロセスのランク
tag       : メッセージID
comm      : コミュニケータ
ierr      : エラーコード(出力)
```

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

```
buff      : 受信バッファの先頭アドレス
count     : 受信するデータの要素数
datatype  : 受信するデータの型
source    : 受信相手のプロセスのランク
tag       : メッセージID
comm      : コミュニケータ
status    : 状況オブジェクトの配列を指定　※サイズはMPI_STATUS_SIZE
ierr      : エラーコード(出力)
```



MPI プログラム(2)の解説(4/4)

- 引数に関する諸注意
  - buff
    - 送信する領域は，メモリ上で連続アドレスでなければならない
    - 他の通信関数でも同じ
  - datatype
    - MPI_INTEGER，MPI_REAL，MPI_DOUBLE_PRECISION などが使用できる
  - tag
    - 同じ相手プロセスに対して複数の mpi_send を行うとき，それらを区別するために使用する
    - 対応する mpi_recv では同じ tag を使わなければならない
    - 送受信の順番などによって複数の通信を区別できる場合は，tag の値は同じでもよい

演習1-2

- 1から100までの整数の和を2並列で求めるプログラム（sum100.f90）を2プロセスで実行し，結果を確認せよ
- 注意事項など
  - プログラムの所在 ： /tmp/mpi_20130606/1_2/sum100.f90
  - ジョブスクリプト　 ： 演習1-1のものを適切に修正して使用
- 結果の確認
  - （プロセス0が）正しい答え（1から100までの和）を出力することを確認

演習1-3（提出課題その1）

- 1から100までの整数の和を2並列で求めるプログラム（sum100.f90）を4プロセス用に書き換えて，4プロセスで実行せよ
- ポイント
  - 各プロセスが部分和を計算する範囲は・・・？
  - プロセス0は部分和を三回受信（プロセス1, 2, 3からそれぞれ）
  - 集団通信関数（後述）を使わないで実現しましょう

※課題の提出方法については最後にまとめて説明します。

MPIプログラムの例 (3):集団通信

- 1からnまでの整数の和を並列で求めるプログラム

```
program sumn
  use mpi
  implicit none
  integer :: n,i,istart,iend,isum,isum1
  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*myrank/nprocs+1
  iend=n*(myrank+1)/nprocs
  isum=0
  do i=istart, iend
     isum=isum+i
  end do
  call mpi_reduce(isum,isum1,1,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD,ierr)
  if (myrank==0) print *, 'sum =', isum1
  call mpi_finalize(ierr)
end program sumn
```

MPIプログラム(3)の解説(1/3)

- 1からnまでの整数の和を

- 青:MPIプログラムのお約束
- 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 sumn
  use mpi
  implicit none
  integer :: n,i,istart,iend,isum,isum1
  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*myrank/nprocs+1
  iend=n*(myrank+1)/nprocs
  isum=0
  do i=istart, iend
     isum=isum+i
  end do
  call mpi_reduce(isum,isum1,1,MPI_INTEGER,MPI_SUM,0,MPI_COMM_WORLD,ierr)
  if (myrank==0) print *, 'sum =', isum1
  call mpi_finalize(ierr)
end program sumn
```

プロセス0がnを読み込む

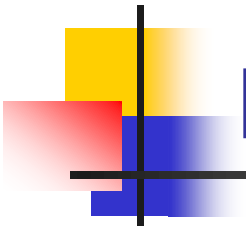
nの値を放送

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は結果を出力



MPIプログラム(3)の解説（2/3）

- mpi_bcast
  - 1個のプロセス（root）から他の全てのプロセスにデータを送信
  - ブロッキング通信
    - root はデータが送信バッファから送り出された後にリターン
    - 他のプロセスはデータが受信バッファに完全に格納された後にリターン
- mpi_reduce
  - 全プロセスの持つデータに対してリダクション演算を行い，結果をrootに送る
  - ブロッキング通信

MPIプログラム(3)の解説(3/3)

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

buff	:	(送信プロセス)送信バッファの先頭アドレス (受信プロセス)受信バッファの先頭アドレス
count	:	データの要素数
datatype	:	データの型
root	:	送信元のプロセスのランク
comm	:	コミュニケータ
ierr	:	エラーコード(出力)

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

sendbuff	:	送信バッファの先頭アドレス
recvbuff	:	受信バッファの先頭アドレス(rootでのみ使用)
count	:	送信するデータの要素数
datatype	:	送信するデータの型
op	:	リダクション演算の種類
root	:	リダクション演算の結果が送られるプロセスのランク
comm	:	コミュニケータ
ierr	:	エラーコード(出力)

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1-4

- 1からnまでの整数の和を並列で求めるプログラム（sumn.f90）を4，8プロセスで実行し，結果を確認せよ
- 注意事項など
  - プログラムの所在 ： /tmp/mpi_20130606/1_4/sumn.f90
  - ジョブスクリプト ： 演習1-1のものを適切に修正して使用
  - nについて ： スライドと同じで，n=10000で実行
- 結果の確認
  - （プロセス0が）正しい答え（1からnまでの和）を出力することを確認

演習1-5(提出課題その2)

- 1からnまでの整数の和を並列で求めるプログラム(sumn.f90)を以下のように書き換え，8プロセスで実行せよ
  - 変数 isum, isum1を倍精度実数 sum, sum1に変更
  - mpi_reduceでの計算も倍精度で行うように変更
- ポイント
  - 倍精度実数型の定義(臼井先生の講義を参照)
  - mpi_reduce内のdatatypeをMPI_DOUBLE_PRECISIONに変更

※課題の提出方法については次のスライドで説明

課題の提出方法と提出期限

- 課題
  - 演習1-3および演習1-5
- 提出方法
  - 課題ごとにプログラムと実行結果を1つのファイルにまとめる
    - まとめ方の例：
      1. cat program.f90 > report_1_x.txt
      2. cat result.o? >> report_1_x.txt
      3. report_1_x.txtの中身を確認
  - 以下の方法でメールにより提出（xは演習1-3なら3，1-5なら5）

    nkf –Lu report_1_x.txt | mail –s mpi_1_x comprampi@gmail.com

    ※改行コードの関係でnkfコマンドを使用しています。
- 期限：６月１２日（水） 午後５時

参考文献

- P.パチェコ（秋葉 博訳）：“MPI並列プログラミング”，培風館，2001.
- 青山幸也：“並列プログラミング入門　MPI版”，http://accc.riken.jp/HPC/training.html
- 片桐孝洋：“スパコンプログラミング入門 -並列処理とMPIの学習-”，東大出版会，2013.

質問は以下のアドレスにお願いします。

- 山本：yamamoto@cs.kobe-u.ac.jp
- 深谷：fukaya@people.kobe-u.ac.jp