KOBE HPC スプリングスクール 2021（中級）

MPI復習

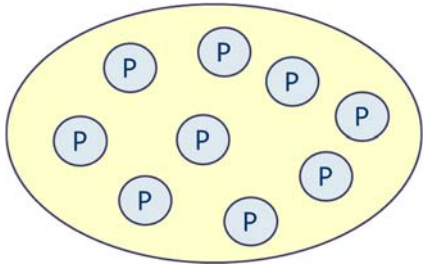
2021年3月8日

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

並列計計算機のアーキテクチャ

- 並列計算機
  - 複数のプロセッサ（ノイマン・アーキテクチャ）が，何らかの方法で接続されていて，協調して動作する計算機システム

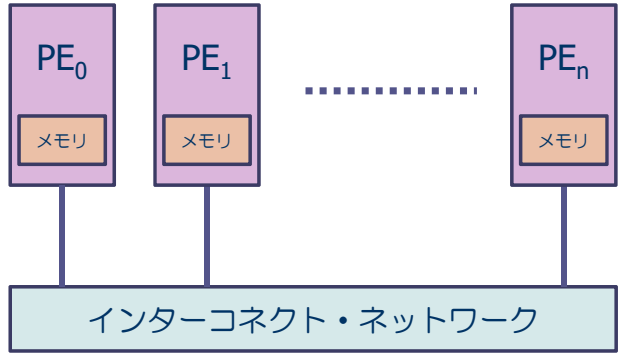
- 並列計算機のタイプ
  - SIMDとMIMD（Flynnの分類, 1966年）
    - 命令実行の流れとデータの流れに着目した分類方法
  - 共有メモリ型と分散メモリ型
    - メモリ空間に着目した分類方法
  - ホモジニアス型とヘテロジニアス型
    - ハードウェアの均質性に着目した分類方法



プロセッサ（P），プロセッサエレメント（PE），プロセッサユニット（PU），ノード（Node），コア（core）

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されており，それぞれのプロセッサ（PE）が，メモリを持っている.
  - 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.

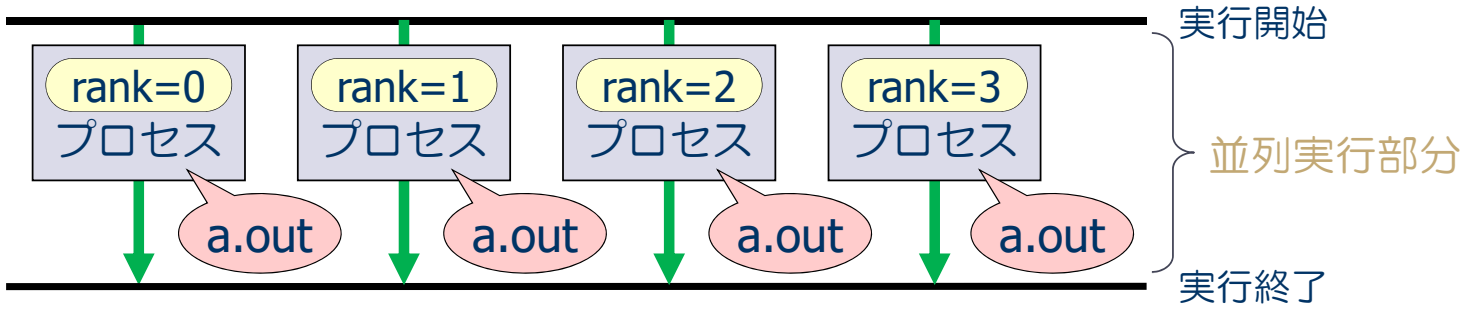
- プログラミング方法
  - メッセージ・パシング・インターフェイス（MPI）によるプログラミング

メッセージ・パシング・インターフェイス（MPI）

- Message Passing Interface（MPI）とは．．．
  - 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，大学などが中心になって標準化
  - MPI規格化の歴史
    - 1994　MPI-1
    - 1997　MPI-2（片方向通信など）
    - 2012　MPI-3
      - http://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf

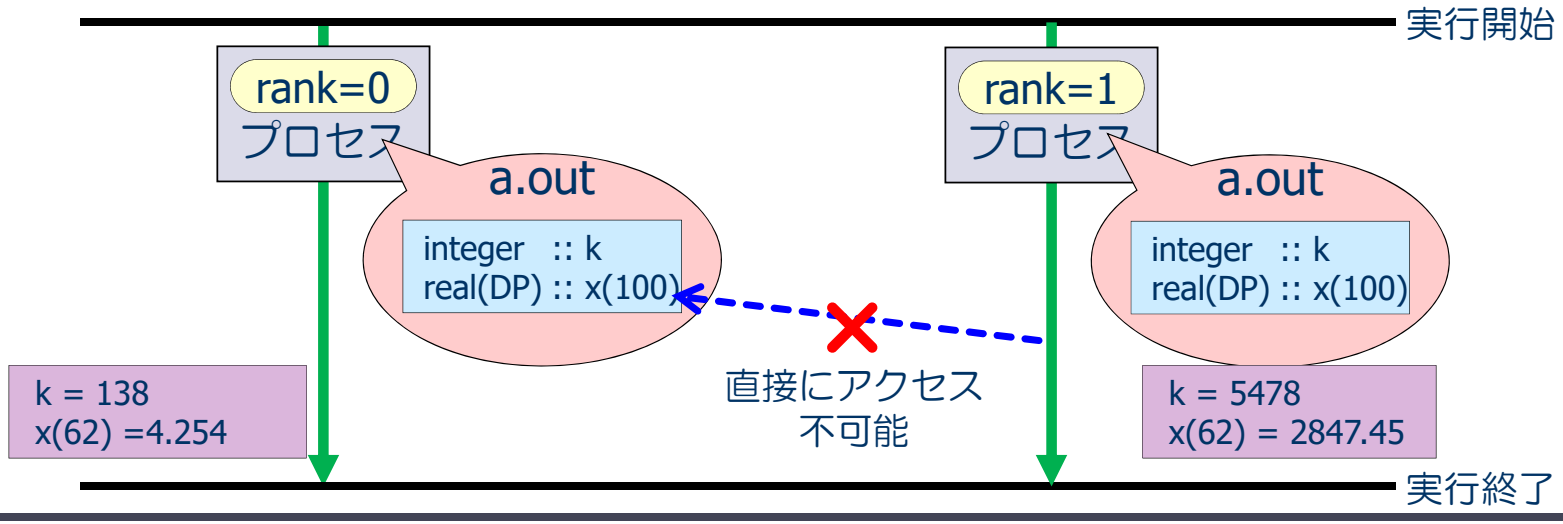
MPIの実行モデル：SPMD（Single Program, Multiple Data)

- 複数のプロセスにより並列実行
- 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MPIプロセスで実行する場合，ランク番号は 0 から (P-1)までの整数となる.
- 各プロセスで処理を変えたい時は，ランク番号を使った分岐により，各プロセスの処理を記述する.



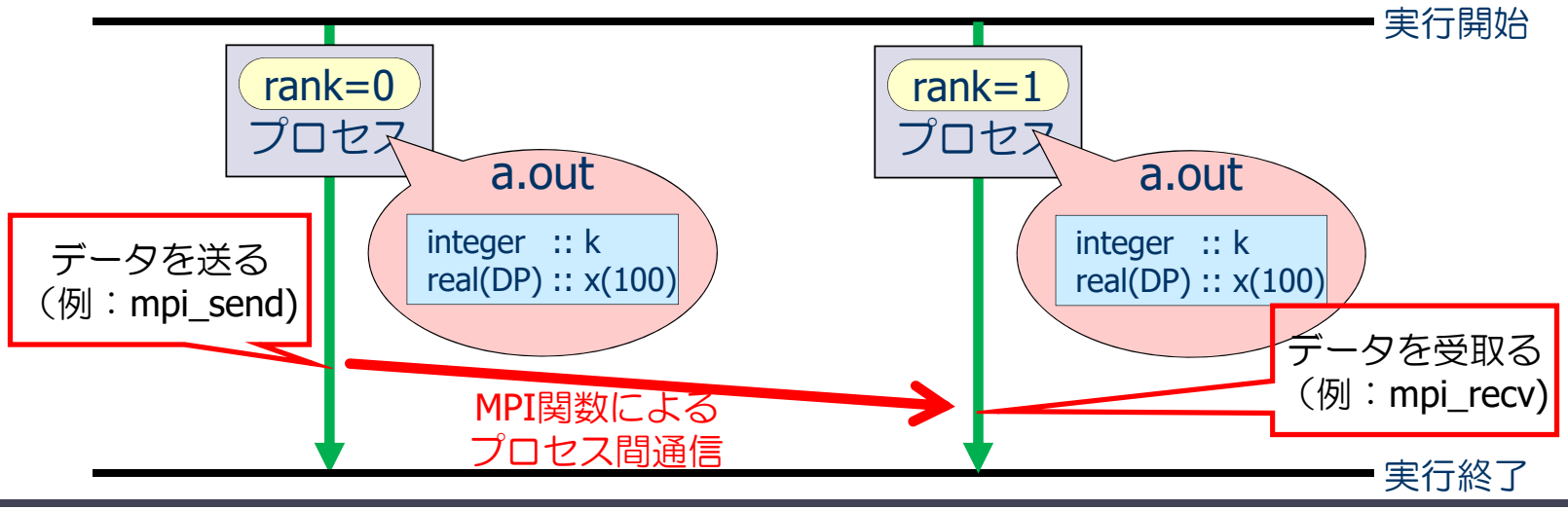
MPIの実行モデル（続き）

- 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．
    - 同じ名前の変数は，通常プロセスごとに違う値を持っている．
    - 他のプロセスの持つ変数には，直接アクセスできない．



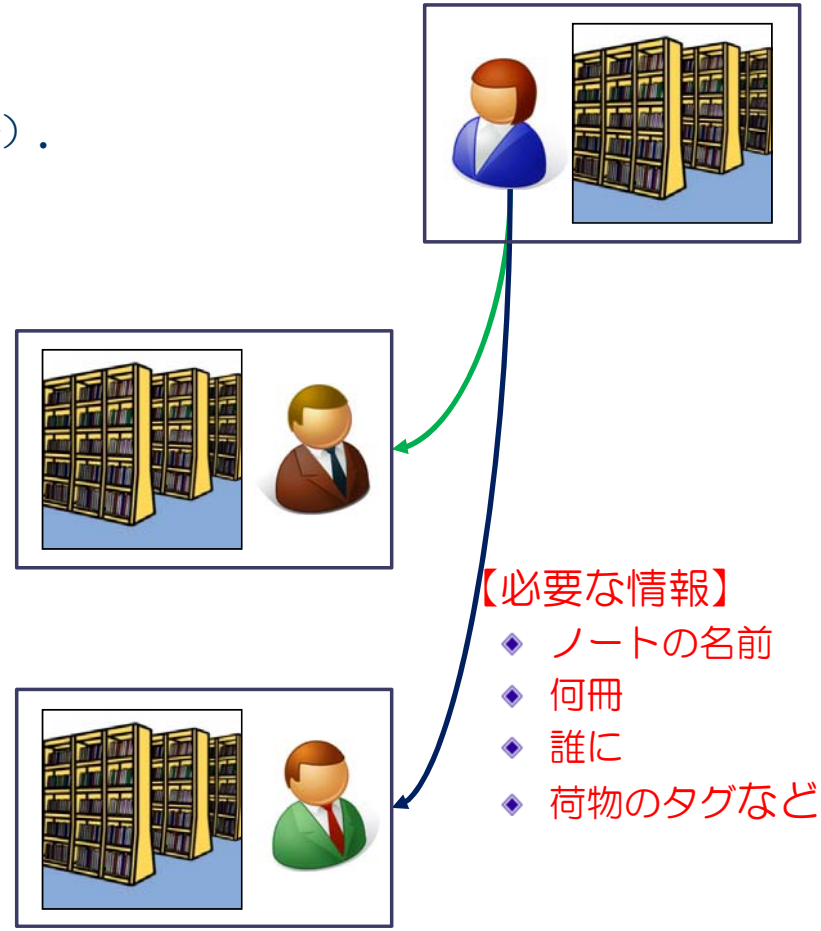
MPIの実行モデル（続き）

- プロセス間通信
  - 他のプロセスの持つ変数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.
  - メッセージパシング方式：メッセージ（データ）の送り手と受け手がある.
  - メッセージパシング方式によるプロセス間通信の集合 ≒ MPI



SPMDのイメージ（例：ノートの貸し借り）

- それぞれの人が，本棚に一連のノートを持っている．
  - それぞれの人には，名前が付いている（人を区別できる：ランク番号）．
  - 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，それぞれの人が，読んだり書いたりできる．
  - 大体は同じ作業をするが，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，その人の名前で作業と指示してあげる．
- 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

MPIプログラムのスケルトン（C）

```
#include <stdio.h>
#include <mpi.h>

int main(int argc, char **argv)
{
  int  nprocs, myrank;

  MPI_Init( &argc, &argv );
  MPI_Comm_size( MPI_COMM_WORLD, &nprocs );
  MPI_Comm_rank( MPI_COMM_WORLD, &myrank );


(この部分に並列実行するプログラムを書く）

  MPI_Finalize() ;
  return 0 ;
}
```

MPIモジュールの取り込み（おまじない1）

MPIで使う変数の宣言

MPIの初期化（おまじない2）
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

MPIの終了処理（おまじない3）

- それぞれのプロセスで異なった処理をする場合は，ランク番号（myrank）の値で場合分けし，うまく仕事が振り分けられるようにする.

MPIプログラムの基本構成（説明：C）

- `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 *size )`
  - 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 *rank )`
  - 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．

MPIプログラムのスケルトン（Fortran）

```
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
```

- それぞれのプロセスで異なった処理をする場合は，ランク番号（myrank）の値で場合分けし，うまく仕事が振り分けられるようにする.

MPIプログラムの基本構成（説明：Fortran）

- `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．

【復習】簡単なMPIプログラム：Hello, world! 【C】

```c
#include <stdio.h>
#include <mpi.h>

int main( int argc, char **argv )
{
  int myrank, nprocs ;

  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myrank);

  printf (“Hello, World! My rank number and nprocs are %d and %d.¥n", myrank, nprocs );

  MPI_Finalize();

  return 0;
}
```

【復習】簡単なMPIプログラム：Hello, world! 【Fortran】

```
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
```

【OBCXでの実行】Hello, world! を並列に出力する

- MPI版 “Hello, world!” を 2，4，及び8プロセスで実行し，結果を確認せよ！ 以下は実行例.


```
$  pwd                                          現在のディレクトリを確認．t65nnnは，それぞれのアカウント
/home/t65nnn

$  cd /work/gt65/t65nnn                         workディレクトリに移動
$  pwd                                          移動を確認
/work/gt65/t65nnn

$  mkdir␣MPI                                    演習用のディレクトリを作成する．※ ␣はスペースを表わす．
$  cd MPI
$  cp␣/home/t65025/MPI/hello_mpi.c␣.           ソースプログラム hello_mpi.c をカレントディレクトリにコピーする．
                                                ※ 中身を見て確認すること．
$  mpiicc␣-o␣hello␣hello_mpi.c                 ソースプログラムをコンパイル

$  cp␣/home/t65025/MPI/go.sh␣./                ジョブスクリプト go.sh をコピーする．
                                                （プロセス数の指定など，必要な部分を変更する）

$  pjsub␣go.sh                                  ジョブを投入し，実行結果の出力ファイル（hello.onnnnnn）を確認する．
[INFO] PJM 0000 pjsub Job nnnnnn submitted.      ※ nnnnnnはジョブ番号

$  cat␣hello.onnnnnn                            ※ helloは，go.sh内で指定した jobname のこと
```

実行結果の確認

- ２プロセスでの実行結果

```
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

```
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ず，また，実行ごとに出力の順番が異なることがある.

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.