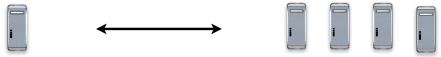
並列処理（並列コンピューティング）

- 同じ仕事量を複数人でこなせば、時間短縮
  - 封筒の袋詰め、仕分け、など
- 複数のCPUを用いて同じような並列処理が出来ないか？

CPU1台で処理　　CPU4台で処理



効率的な処理が可能？

並列コンピュータ

- 複数プロセッサが主記憶を共有する、共有メモリ型
- 各プロセッサが独自のメモリを持つ、分散メモリ型

本実験では、分散メモリ型の並列処理を実現する MPI を学ぶ

MPI = Message Passing Interface

プロセッサ間のデータ通信に関する標準規格

並列処理の実装はプログラマの責任

MPI

- 分散メモリ型の並列処理では、プロセッサ間の通信が欠かせない
- プロセッサ間のデータ通信をライブラリとして提供
- 様々な実装があるが本実験では OpenMPI を用いる

ウェブ上には様々なMPIライブラリの解説がある

様々な並列処理

具体例：数列の和の計算　$\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$

プロセッサ2台を用いて数列和を並列実行する場合

CPU 0　　CPU 1

```
sum = 0;
for(i=1; i<=N; i+=2)
  sum += i;
```

```
sum = 0;
for(i=2; i<=N; i+=2)
  sum += i;
```

CPU0 と CPU1 の計算結果 sum の和が求める解
（プロセッサ間の通信が必要）

MPIの基礎

- MPIでは、各プロセッサは固有ID（rank）を持つ
- rankに従う各プロセッサの動作をプログラマが記述
- 必要であれば、各プロセッサ間の通信をプログラマが記述

rank=0　rank=1　rank=2　rank=3

実行に参加するプロセッサの数 size = 4

MPIひな型

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

int main( int argc, char *argv[] )
{
    int rank, size, namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Get_processor_name(processor_name, &namelen);

    printf( "Hello world from process %d of %d on %s\n", rank, size,  processor_name );

    MPI_Finalize();

    return 0;
}
```


例1

```c
int main( int argc, char *argv[] )
{
    int rank, size, namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init( &argc, &argv );
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Get_processor_name(processor_name, &namelen);

    printf( "Hello world from process %d of %d on %s\n", rank, size, processor_name );

    if( rank == 0 )
      printf("I am rank %d. Do you see me?\n", rank);
    else
      printf("I am rank %d. Hello!\n", rank);

    MPI_Finalize();

    return 0;
}
```


プロセッサ間通信

プロセッサ間の1対1通信を行う関数 MPI_Send と MPI_Recv

データ送信

```
MPI_Send(
  void *buf,                 送信するデータのアドレス
  int count,                 送信するデータの数
  MPI_Datatype datatype,     送信するデータの型 MPI_INT, MPI_DOUBLE
  int dest,                  送信先プロセッサ rank
  int tag,                   タグ
  MPI_Comm comm              送受信グループ MPI_COMM_WORLD
)
```

プロセッサ間通信

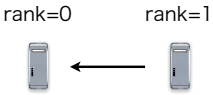
プロセッサ間の1対1通信を行う関数 MPI_Send と MPI_Recv

データ受信

```
MPI_Recv(
  void *buf,                受信データが格納されるアドレス
  int count,                受信するデータの数
  MPI_Datatype datatype,    受信するデータの型 MPI_INT, MPI_DOUBLE
  int source,               送信先元プロセッサ rank
  int tag,                  タグ
  MPI_Comm comm,            送受信グループ MPI_COMM_WORLD
  MPI_Status *status        受信結果の情報を格納するアドレス
)
```


プロセッサ間通信

rank1 から rank0 へ
実数値 double x を送る



rank1 での処理

```
dest = 0;
tag = 777;
MPI_Send(&x, 1, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
```

rank0 での処理

```
source = 1;
tag = 777;
MPI_Recv(&x, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD,
&mpi_stat);
```


1対1通信の例

```
MPI_Status mpi_stat;
int source, dest, tag;
double x;

.....

if( rank == 0 ){

   printf("Rank %d is going to receive a data\n", rank);
   source = 1;
   tag = 777;
   MPI_Recv( &x, 1, MPI_DOUBLE, source, tag, MPI_COMM_WORLD, &mpi_stat);
   printf("Rank %d has received a data %f\n", rank, x);

} else {

   x = 123.0;
   printf("Rank %d is going to send a data %f\n", rank, x);
   dest = 0;
   tag = 777;
   MPI_Send( &x, 1, MPI_DOUBLE, dest, tag, MPI_COMM_WORLD);
   printf("Rank %d has sent a data\n", rank);

}
```


数列和の並列計算

プロセッサ2台を用いて数列和を並列実行 $\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$

rank0 の処理

```
int sum;

sum = 0;
for(i=1; i<=N; i+=2)
  sum += i;
```

rank1のsumを受信

rank1 の処理

```
int sum;

sum = 0;
for(i=2; i<=N; i+=2)
  sum += i;
```

rank0へsumを送信

受信したrank1のsumと自分のsumの和を表示

課題

- 1からNまでの数列和を2台のプロセッサを用いて並列計算して結果を表示するプログラムを作れ
- 下記積分をリーマン和として計算し、円周率の近似値を求めるプログラムを、1）1台のCPUを用いる場合、2）2台のCPUを用いる場合、の二通り作成せよ。

$$\int_0^1 \frac{4}{1+x^2} dx = \pi$$

奈良女子大学理学部情報科学科
H22 計算機実験2

OpenMPI実行のための環境

- OpenMPIコマンド群へのパスの設定
- 他のCPUにパスワードの入力なしにログインできること
（他のCPUにジョブを展開するために必要）

上記の設定方法については別途説明する

並列処理プログラムのコンパイルと実行は以下の通り

```
% mpicc mpi-1.c
% mpirun -np 2 ./a.out
% mpirun -np 2 --host gpx10, gpx11 ./a.out
```