第1回 MPI ゼミ

ゼミ担当者　：　中尾　昌広，永松　秀人
指導院生　　：　伏見　俊彦，小椋　信弥
開催日　　：　2002 年 10 月 31 日

ゼミ内容: 本ゼミでは，MPI の基本的な関数の使い方を説明する．

1　メッセージパッシングの形態

メッセージパッシング方式とは，プロセス間でメッセージを交信しながら並列処理を実現する方法である．並列処理では，複数のプロセスが通信を行いながら同時に処理を進めていく．そこで問題となるのがプロセス間の通信であるが，メッセージパッシング方式ではプロセス間での通信をお互いのデータの送受信にて行う．
メッセージパッシング方式には 1 対 1 通信とグループ通信の 2 種類が存在する．また，それぞれについてブロッキング通信とノンブロッキング通信という方式が存在する．

1.1　1 対 1 通信とグループ通信

- **1 対 1 通信**
  メッセージパッシングにおけるもっとも基本的な通信機能であり，一つのプロセスが送信元，相手のもう一つのプロセスが受信元になって行われる．Fig. 1 で示すと，プロセス A がプロセス B にメッセージを送っていることになる．この場合，ほかのプロセス C〜F にはメッセージは送られない．

- **グループ通信**
  プロセスがグループ内での集団通信動作．例えば，一台のマシンが他のマシン全体に対してデータを送信することである．Fig. 2 の例では，プロセス A がグループ中のそのほかのプロセス，プロセス B〜F すべてにメッセージを送ることになる．

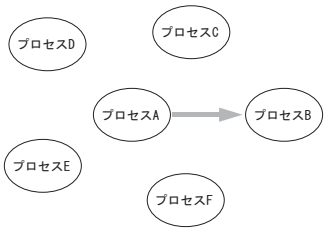

Fig. 1　1 対 1 通信の形態

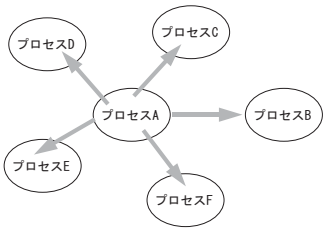

Fig. 2　グループ通信の形態

1.2　ブロッキング通信とノンブロッキング通信

- **ブロッキング通信**
  ブロッキング通信とは，操作が完了するまで手続きから戻る事がない場合のことを意味する．この場合，各作業はその手続きが終了するまで待たされることになり効率が悪くなる場合がある．つまり，処理の流れをいったんブロックするわけである．具体的には，Fig. 3 に示すように送信から受信するまでの処理は待機状態になるため非常に無駄が多くなってしまう．ただし，各操作の完結が保証されているためノンブロッキング通信に比べ処理が簡単になる．

- **ノンブロッキング通信**
  ノンブロッキング通信とは，操作が完了する前に手続きから戻ることがあり得る場合のことを意味する．ノンブロッキング通信を用いることにより効率のよいプログラムを作成することができる．Fig. 4 に示すように，送信

から受信するまでの処理は引き続き継続することになるため，ブロッキング通信に比べ通信待ちの時間が少なくなり処理時間の軽減を計ることができる．特に，非同期通信などを行う場合にはノンブロッキング通信は必要不可欠である．しかし，操作の完了が保証されていないため，ノンブロッキング通信を完了するための手続きを呼び出す必要がある．

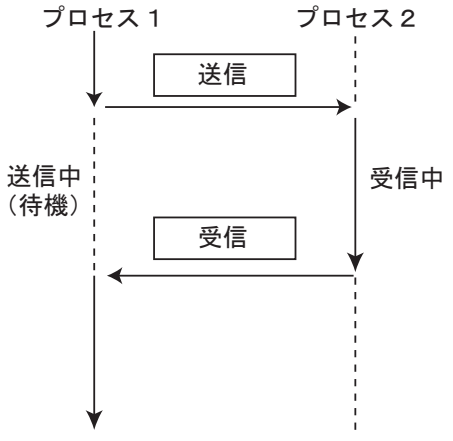

Fig. 3　ブロッキング通信

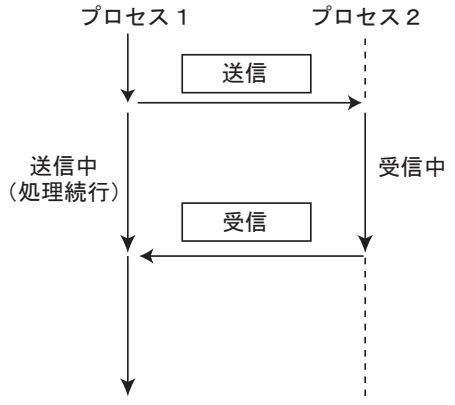

Fig. 4　ノンブロッキング通信

2　MPI Programming

2.1　制御関数

MPI には必ず制御関数を記述する必要がある．制御関数には，MPI を用いる際の初期化のための関数や，MPI ライブラリ利用の終了を宣言するための関数などがある．最低限の制御関数を Table 1 に示す．

Table 1　最低限の制御関数

MPI_Init()	MPI ライブラリを利用するための初期化
MPI_Comm_size()	コミュニケータ内のプロセス数を取得
MPI_Comm_rank()	コミュニケータ内の各プロセスが自分の rank を取得
MPI_Finalize()	MPI ライブラリの利用の終了処理

- **コミュニケータ**
  お互いに通信を行うプロセスの集合である．ほとんどの MPI ルーチンは引数としてコミュニケータをとる．変数 MPI_COMM_WORLD は，あるアプリケーションをいっしょに実行している全プロセスからなるグループを表しており，これは最初から用意されている．また新しいコミュニケータを作成することも可能である．

- **rank**
  コミュニケータ内のすべてのプロセスは，プロセスが初期化されたときにシステムによって示された ID を持っている．これは，0 から始まる連続した整数が割り当てられる．プログラマはこれを用いて，処理の分岐，あるいはメッセージの送信元や受信先を指定することができる．

3　MPI の基本的な通信関数

MPI にはさまざまな通信パターンに対応する関数が 127 個用意されている．それらは 1 対 1 通信の組み合わせでも実現できるが，専用の関数を使えば，プログラムの意図がわかりやすくなり，また通信効率もよくなる．ここでは，いくつかの通信パターンを示す．

3.1　1 対 1 通信

2 節でも述べたように，1 対 1 の通信にはブロッキング通信とノンブロッキング通信の 2 通りの通信方法が存在する．また，ほとんどの MPI 関数のプロトタイプでは，成功した場合の戻り値は MPI_SUCCESS になる．失敗した場

合の戻り値は実装によって異なる.

3.1.1 MPI_Send

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

基本的なブロッキング送信の操作を行う．送信バッファのデータを特定の受信先に送信する.

void *buf	:送信バッファの開始アドレス
int count	:データの要素数
MPI_Datatype datatype	:データタイプ
int dest	:受信先
MPI_Comm comm	:コミュニケータ

3.1.2 MPI_Recv

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

要求されたデータを受信バッファから取り出す．また，それが可能になるまで待つ.

void *buf	:受信バッファの開始アドレス
int count	:データの要素数
MPI_Datatype datatype	:データタイプ
int source	:送信元
int tag	:メッセージ・タグ
MPI_Comm comm	:コミュニケータ
MPI_Status *status	:ステータス

3.1.3 MPI_Isend

int MPI_Isend(void *sendbuf,int sendcount,MPI_Datatype sendtype,int dest,MPI_Comm comm,MPI_Request *request)

基本的なノンブロッキング送信の操作を行う.

void *sendbuf	:送信バッファの開始アドレス
int sendcount	:データの要素数
MPI_Datatype sendtype	:データタイプ
int dest	:受信先
MPI_Comm comm	:コミュニケータ
MPI_Request *request	:通信要求（ハンドル）を返す

3.1.4 MPI_Irecv

int MPI_Irecv(void *recvbuf,int recvcount,MPI_Datatype recvtype,int source,int recvtag,MPI_Comm comm,MPI_Request *request)

基本的なノンブロッキング受信の操作を行う.

void *recvbuf	:受信バッファの開始アドレス
int recvcount	:データの要素数
MPI_Datatype recvtype	:データタイプ
int source	:送信元
int recvtag	:メッセージ・タグ
MPI_Comm comm	:コミュニケータ
MPI_Request *request	:通信要求（）ハンドルを返す

```
int MPI_Wait(MPI_Request *request,MPI_Status *status)
```

MPI_Wait は，request によって識別された操作が完了した時点で手続きが終了する．このルーチンによりノンブロッキング通信は完了する．具体的には，ノンブロッキングでの送信，受信呼び出しによって作成された request が解除され，MPI_REQUEST_NULL が設定される．また，完了した操作に関する情報は status 内に設定される．
ここに，MPI プログラミングの基本的な枠組みを示す．

```
#include"mpi.h"  // ヘッダファイルの読み込み

int main(int argc,char *argv[])
{
    int numprocs,myid;

    MPI_Init(&argc,&argv);  // MPI ライブラリを利用するための準備（初期化）
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    // コミュニケータ内のプロセスの数を取得.
    この関数により numprocs にはプロセス数が入力される
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    // コミュニケータ内の各プロセスが自分の rank を取得.
    この関数により myid には自分の rank 番号が入力される

    /* 並列処理の記述

    MPI_Finalize();  // MPI ライブラリの利用の終了処理
    return 0;
}
```

3.2 グループ通信

本節ではグループ通信に良く使われる関数，MPI_Scatter，MPI_Gather，MPI_Barrier，MPI_Bcast，MPI_Reduce について説明する．

3.2.1 MPI_Scatter

まず MPI_Scatter について紹介する．機能は以下の通りである．Fig. 5 に通信のイメージ図を示す．

- コミュニケータ (comm) 内の，1 つの送信元プロセス (root) の送信バッファ (sendbuf) から，全プロセスの受信バッファ (recvbuf) にメッセージを送信する．
- 各宛先プロセスへのメッセージの長さは一定で，送信バッファ (sendbuf) の先頭から宛先プロセスのランクが小さい順に送信される．

MPI_Scatter(sendbuf,sendcount,sendtype,recvbuf,recvcount,recvtype,root,comm,ierror)

sendbuf	送信バッファの先頭アドレスを指定する.
sendcount	整数. １つのプロセス に送信する送信メッセージの要素数を指定する.
sendtype	整数. 送信メッセージのデータタイプを指定する. root プロセスのみで意味を持つ.
recvbuf	受信バッファの先頭アドレスを指定する.
	送信バッファと受信バッファの実際に使用する部分は, メモリ上で重なってはいけない.
recvcount	整数. 受信メッセージの要素数を指定する.
recvtype	整数. 受信メッセージのデータタイプを指定する.
root	整数. 送信元プロセスの comm 内でのランクを指定する.
comm	整数. 送受信に参加する全てのプロセスを含むグループのコミュニケータを指定する.
	全プロセスが同じ値を指定する必要がある.
ierror	整数. 完了コードが戻る.

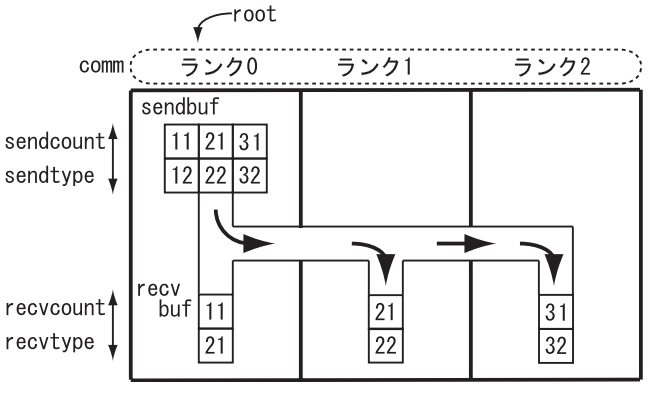

Fig. 5　MPI_Scatter によるデータの移動

3.2.2　MPI_Gather

MPI_Gather の機能は以下の通りである. Fig. 6 に通信のイメージ図を示す.

- コミュニケータ (comm) 内の, 全プロセスの送信バッファ (sendbuf) から, 1 つの宛先プロセス (root) の受信バッファ (recvbuf) にメッセージを送信する.
- 各送信元プロセスからの受信メッセージの長さは一定で, 受信バッファ (recvbuf) の先頭から送信元プロセスのランクが小さい順に入る.
- 本サブルーチンは MPI_Scatter と逆の通信になる.

MPI_Gather(sendbuf,sendcount,sendtype,recvbuf,recvcount,recvtype,root,comm,ierror)

sendbuf	送信バッファの先頭アドレスを指定する.
sendcount	整数. 送信メッセージの要素数を指定する.
sendtype	整数. 送信メッセージのデータタイプを指定する.
recvbuf	受信バッファの先頭アドレスを指定する. root プロセスのみで意味を持つ.
	送信バッファと受信バッファの実際に使用する部分は, メモリ上で重なってはいけない.
recvcount	整数. １つのプロセスから に受信メッセージの要素数を指定する. root プロセスのみで意味を持つ.
recvtype	整数. 受信メッセージのデータタイプを指定する. root プロセスのみで意味を持つ.
root	整数. 宛先プロセスの comm 内でのランクを指定する. comm 内の全プロセスが同じ値を指定する必要がある.
comm	整数. 送受信に参加する全てのプロセスを含むグループのコミュニケータを指定する.
	全プロセスが同じ値を指定する必要がある.
ierror	整数. 完了コードが戻る.

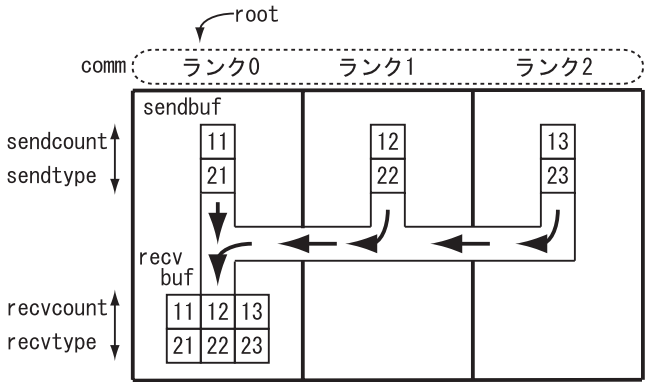

Fig. 6　MPI_Gather によるデータの移動

3.2.3　MPI_Barrier

コミュニケータ (comm) 内の全プロセスで (何らかの理由で) 同期を取りたい場合，MPI_Barrier を使用する.『処理 1』→『MPI_Barrier』→『処理 2』の順に処理を行うプログラムを 3 プロセスで並列に実行する場合を想定する．このとき『処理 1』の部分の進行が，Fig. 7 のようにプロセスによって異なっているとする．まずランク 1 のプロセスが

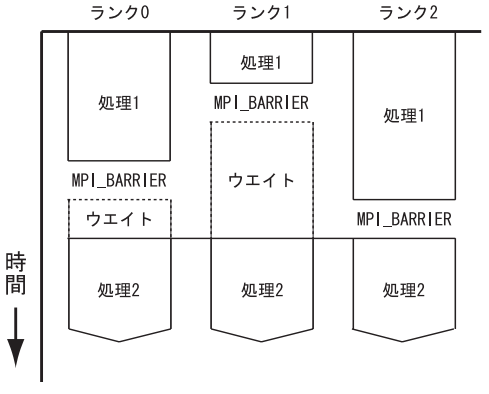

Fig. 7　MPI_Barrier による同期

『MPI_Barrier』に到達しますが，他のプロセスがまだ到着していないため，ランク 1 はウエイトする．次にランク 0 が『MPI_Barrier』に到着し，同様にウエイトする．そして最後にランク 2 が『MPI_Barrier』に到達すると，各プロセスはいっせいに実行を再開する．

```
MPI_Barrier(comm,ierror)
```

comm	整数．同期をとりたい全てのプロセスを含むグループのコミュニケータを指定する． 全プロセスが同じ値を指定する必要がある．
ierror	整数．完了コードが戻る．

3.2.4　MPI_Bcast

MPI_Bcast の機能は以下の通りである．Fig. 8 に通信のイメージ図を示す.

- コミュニケータ (comm) 内の，1 つの送信元プロセス (root) の送信バッファ (buffer) から，その他全てのプロセスの受信バッファ (buffer) にメッセージを送信する．

```
MPI_Bcast(buffer,count,detetype,root,comm,ierror)
```

buffer	送信元プロセスでは，送信バッファの先頭アドレスを指定する． 宛先プロセスでは，受信バッファの先頭アドレスを指定する．
count	整数．メッセージの要素数を指定する．
detetype	整数．メッセージのデータタイプを指定する．
root	整数．送信元プロセスの comm 内でのランクを指定する． 整数．comm 内の全プロセスが同じ値を指定する必要があります．
comm	整数．送受信に参加する全てのプロセスを含むグループのコミュニケータを指定する． 全プロセスが同じ値を指定する必要がある．
ierror	整数．完了コードが戻る．

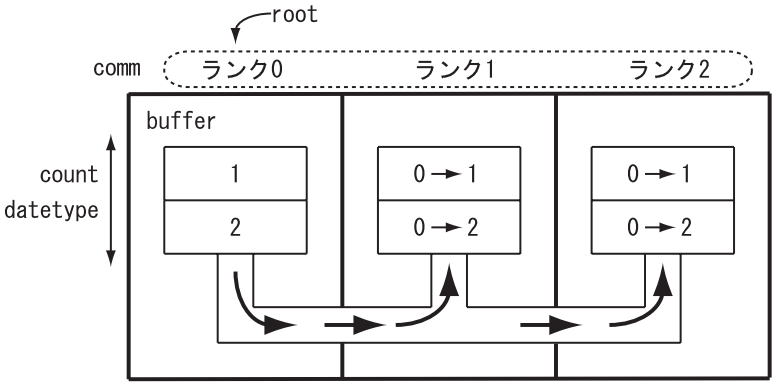

Fig. 8　MPI_Bcast によるデータの移動

3.2.5　MPI_Reduce

MPI_Bcast の機能は以下の通りである．Fig. 9 に通信のイメージ図を示す．

- コミュニケータ (comm) 内の，全プロセスの送信元プロセス (root) の送信バッファ (buffer) のメッセージが，通信しながら演算 (op) され，結果が 1 つの宛先プロセス (root) の受信バッファ (recvbuf) に入る．
- MPI であらかじめ用意されている演算 (op) と，可能なデータタイプ (detetype) の組み合わせを以下に示す．演算をユーザが自分で定義することもでる．

演算 (op)	データタイプ (detetype)
MPI_SUM (合計) MPI_PROD (積)	MPI_INTEGER MPI_REAL MPI_DOUBLE_PRECISION MPI_COMPLEX
MPI_MAX (最大) MPI_MIN (最小)	MPI_INTEGER MPI_REAL MPI_DOUBLE_PRECISION
MPI_MAXLOC (最大と位置) MPI_MINLOC (最小と位置)	MPI_2INTEGER MPI_2REAL MPI_2DOUBLE_PRECISION
MPI_LAND (論理 AND) MPI_LOR (論理 OR) MPI_LXOR (論理 XOR)	MPI_LOGICAL
MPI_BAND (ビット AND) MPI_BOR (ビット OR) MPI_BXOR (ビット XOR)	MPI_INTEGER MPI_BYTE

MPI_Reduce(sendbuf,recvbuf,count,detetype,op,root,comm,ierror)

sendbuf	送信バッファの先頭アドレスを指定する.
recvbuf	受信バッファの先頭アドレスを指定する. root プロセスのみ意味を持つ 送信バッファと受信バッファの実際に使用する部分は，メモリ上で重なってはいけない.
count	整数. 送 (受) 信のメッセージの要素数を指定する. comm 内の全プロセスが同じ値を指定する必要がある.
detetype	整数. 送 (受) 信のメッセージのデータタイプを指定する. comm 内の全プロセスが同じ値を指定する必要がある.
op	整数. 演算の種類を指定する. comm 内の全プロセスが同じ値を指定する必要がある.
root	整数. 宛先プロセスの comm 内でのランクを指定する. 整数. comm 内の全プロセスが同じ値を指定する必要がある.
comm	整数. 送受信に参加する全てのプロセスを含むグループのコミュニケータを指定する. 全プロセスが同じ値を指定する必要がある.
ierror	整数. 完了コードが戻る.

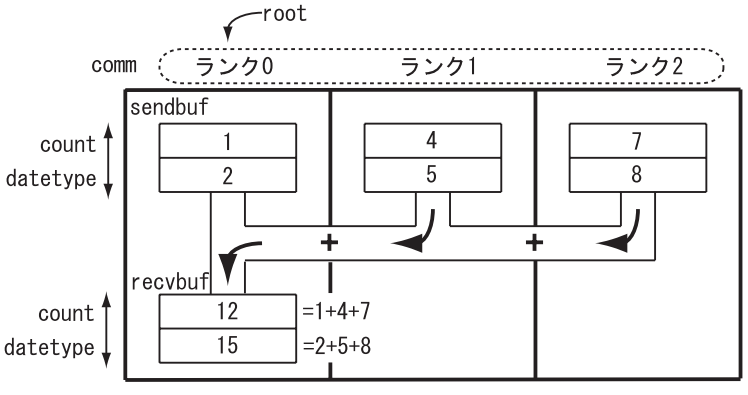

Fig. 9　MPI_Reduce によるデータの移動

参考文献

1) 渡邊真也：MPI による並列プログラミングの基礎，http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/.

2) 三木光範：並列処理入門，http://mikilab.doshisha.ac.jp/dia/smpp/cluster2000/.