メッセージパッシング プログラミング

天野

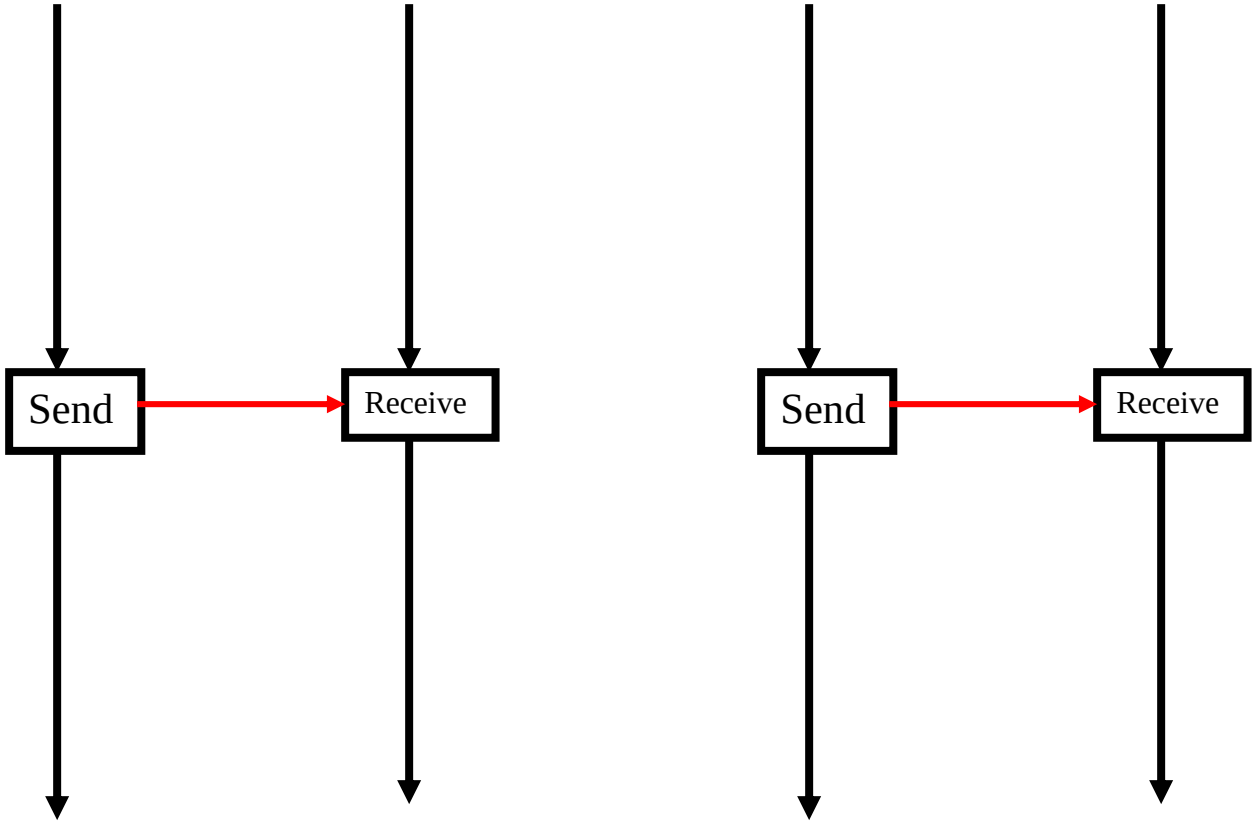
共有メモリ　対　メッセージパッシング

- 共有メモリモデル
  - 共有変数を用いた単純な記述
  - 自動並列化コンパイラ
  - 簡単なディレクティブによる並列化：OpenMP
- メッセージパッシング
  - 形式検証が可能 (ブロッキング)
  - 副作用がない(共有変数は副作用そのもの)
  - コストが小さい

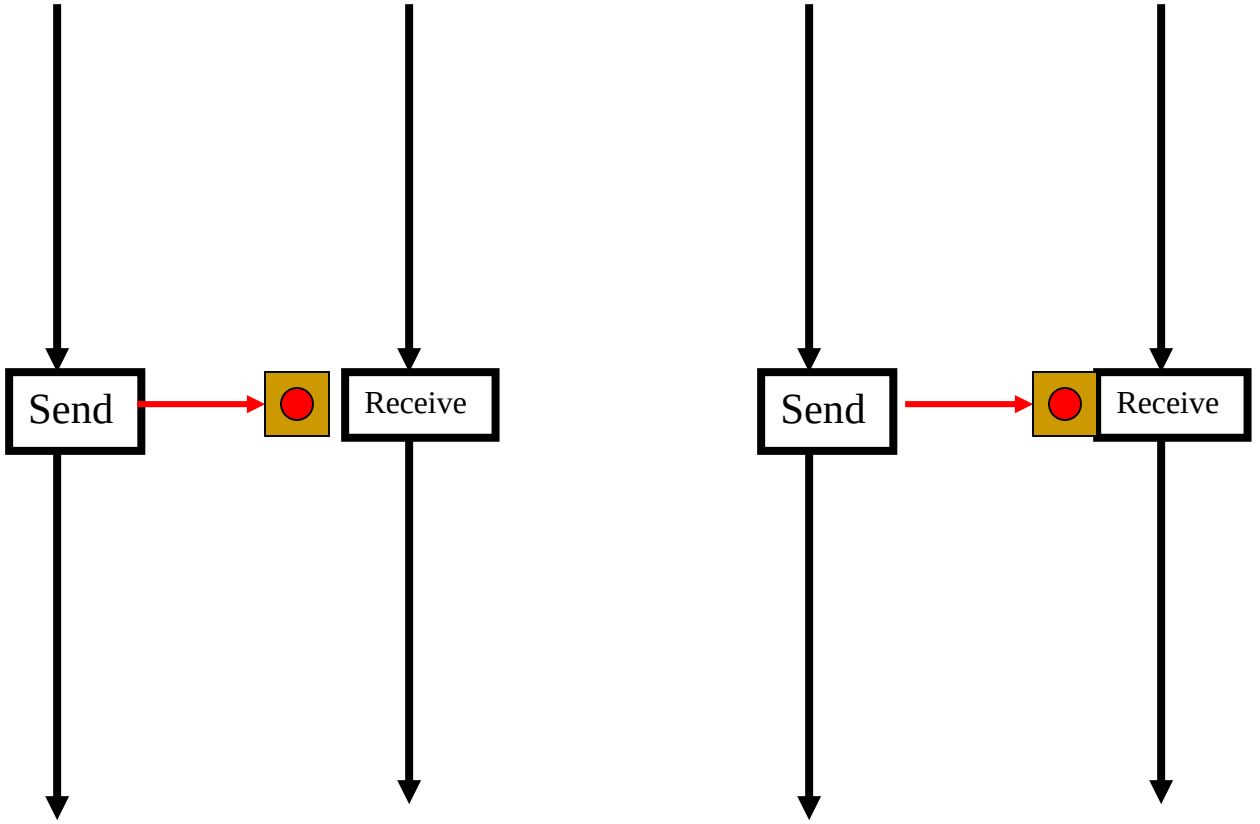
メッセージパッシングモデル

- 共有変数は使わない
- 共有メモリがないマシンでも実装可能
- クラスタ、大規模マシンで利用可能
- ここではMPIを紹介する

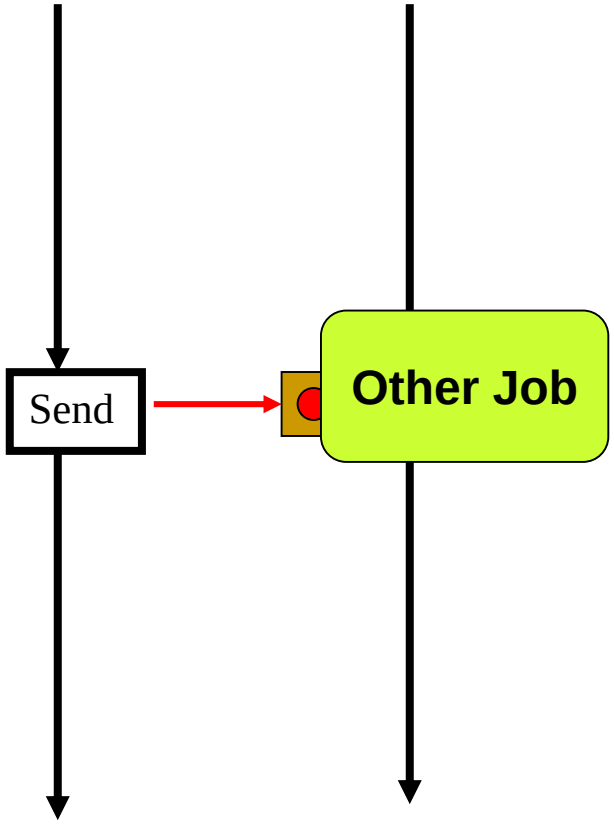
ブロッキング通信
（Blocking: ランデブ）



バッファ付き通信



ノンブロッキングのメッセージ通信



PVM (Parallel Virtual Machine)

- 送り側にはバッファが一つ存在
- 受信側にはブロッキング・ノンブロッキングの両方が可能。
- バリア同期を利用

MPI
(Message Passing Interface)

- 1対1の通信ではPVMのスーパーセット
- グループ通信
- 多様な通信をサポート
- Communication tagでエラーチェック

MPIのプログラミングモデル

- 基本的には、SPMD (Single Program Multiple Data Streams)
  - 同じプログラムが複数プロセスで実行
  - プロセス番号を識別すれば個別のプログラムが走る。
- MPIを用いたプログラム
  - 指定した数のプロセスが生成される
  - NORAマシンまたはPCクラスタの各ノードに分散される

交信の種類

- 1対1転送
  - 送信側と受信側が対応する関数を実行
  - きちんと対応していないとダメ
- 集合的な通信
  - 複数のプロセス間での通信
  - 共通の関数を実行
  - 1対1転送で置き換え可能だが、効率がやや向上する場合がある

基本的なMPI関数

この6つが使えれば多くのプログラムが書ける！

- **`MPI_Init()`** ... MPI Initialization
- **`MPI_Comm_rank()`** ... Get the process #
- **`MPI_Comm_size()`** ... Get the total process #
- **`MPI_Send()`** ... Message send
- **`MPI_Recv()`** ... Message receive
- **`MPI_Finalize()`** ... MPI termination

その他のMPI 関数

- 同期、計測用
  - **MPI_Barrier()** … バリア同期
  - **MPI_Wtime()** … 時刻を持ってくる
- ノンブロッキング転送
  - 転送要求とチェックにより構成される。
  - 待ち時間中に別の関数を実行可能

例題

```
1: #include <stdio.h>
2: #include <mpi.h>
3:
4: #define MSIZE 64
5:
6: int main(int argc, char **argv)
7: {
8:      char msg[MSIZE];
9:      int pid, nprocs, i;
10:    MPI_Status status;
11:
12:    MPI_Init(&argc, &argv);
13:    MPI_Comm_rank(MPI_COMM_WORLD, &pid);
14:     MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
15:
16: if (pid == 0) {
17:    for (i = 1; i < nprocs; i++) {
18:      MPI_Recv(msg, MSIZE, MPI_CHAR, i, 0, MPI_COMM_WORLD, &status);
19:      fputs(msg, stdout);
20:      }
21:    }
22:    else {
23:      sprintf(msg, "Hello, world! (from process #%d)¥n", pid);
24:      MPI_Send(msg, MSIZE, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
25:    }
26:
27:    MPI_Finalize();
28:
29:    return 0;
30: }
```

初期化と終結

```
int MPI_Init(
    int *argc, /* pointer to argc */
    char ***argv /* pointer to argv */ );
```

argc と argvはコマンドラインからの引数.

```
int MPI_Finalize();
```

例

```
MPI_Init (&argc, &argv);
…
MPI_Finalize();
```

コミュニケータ制御用の関数
コミュニケータは通信用の空間

MPI_COMM_WORLDは、全プロセス用のコミュニケーター＞今回はこれを使う

```
int MPI_Comm_rank(
    MPI_Comm comm, /* communicator */
    int *rank /* process ID (output) */ );
```
プロセスID（ランク）を返す

```
int MPI_Comm_size(
    MPI_Comm comm, /* communicator */
    int *size /* number of process (output) */ );
```
全プロセス数を返す

例：
```
int pid, nproc;
MPI_Comm_rank(MPI_COMM_WORLD, &pid);
MPI_Comm_rank(MPI_COMM_WORLD,&nproc);
```
自分のプロセスID
全プロセス数

MPI_Send

1対1のメッセージ送信

```
int MPI_Send(
   void *buf, /* send buffer */
   int count, /* # of elements to send */
   MPI_Datatype datatype, /* datatype of elements */
   int dest, /* destination (receiver) process ID */
   int tag, /* tag */
   MPI_Comm comm /* communicator */ );
```

```
MPI_Send(msg, MSIZE, MPI_CHAR, 0,0, MPI_COMM_WORLD);
```

メッセージ用文字列配列msgの中の文字をMSIZE分プロセス0（タグも0）で送る

タグが一致したMPI_Recvでのみ受け取ることが可能

MPI_Recv

1対1のメッセージ受信

```
int MPI_Recv(
        void        *buf,                /* receiver buffer */
        int          count,              /* # of elements to receive */
        MPI_Datatype datatype,           /* datatype of elements */
        int          source,             /* source (sender) process ID */
        int          tag,                /* tag */
        MPI_Comm comm,                   /* communicator */
        MPI_Status                       /* status (output) */ );
```

```
char msg[MSIZE]
MPI_Status status;

MPI_Recv(msg, MSIZE, MPI_CHAR, 1, 0, MPI_COMM_WORLD, &status);
fputs(msg, stdout);
```

プロセス1からのタグ0で送って来たサイズMSIZEの文字列を受信し、msgに入れる。

- statusは受信したメッセージの状態を示す。

MPI_Bcast

全プロセスに対してメッセージを転送

```
int MPI_Bcast(
   void *buf, /* send buffer */
   int count, /* # of elements to send */
   MPI_Datatype datatype, /* datatype of elements */
   int root, /* Root processor number */
   MPI_Comm comm /* communicator */ );

if (pid ==0)
  a=1.0;

MPI_Bcast(&a,1,MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

pid 0が他の全てに対してa=1.0を転送する。

メッセージのデータタイプ

- 通常のデータサイズに対応するMPIのデータサイズを指定
  - MPI_CHAR char
  - MPI_INT int
  - MPI_FLOAT float
  - MPI_DOUBLE double … etc.

コンパイルと実行

Webからmpiex.tarをダウンロード

tar xvf mpiex.tar

cd mpiex

% mpicc –o hello hello.c

% mpirun –np 4 ./hello

Hello, world! (from process #1)

Hello, world! (from process #2)

Hello, world! (from process #3)

演習問題

- 配列x[4096]がある.
- この2乗和を取る計算をMPIライブラリで並列化せよ。

```
sum = 0.0;
for (i=0; i<N; i++)
      for(j=0; j<N; j++)
     sum += (x[i]-x[j])*(x[i]-x[j]);
```

解説

- reduct.cを元にする
- xを全プロセスに転送
- 各プロセスでは部分和を計算

```
sum=0.0;
for (i=N/nproc*pid; i<N/nproc*(pid+1); i++)
        for(j=0; j<N; j++)
            sum += (x[i]-x[j])*(x[i]-x[j]);
```

  - 部分和を0に転送、0で総和を取る
- 1－4プロセスで実行してみて、実行時間を測定せよ
- あんまり早くならないと思う。
- 結果が微妙に違うかも（加算の順番が狂うので。。。）