1 MPI_Send,MPI_Recv

1.1 プログラム例

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

int main(int argc,char *argv[])
{
    int myid,procs,src,dest,tag;=1000,count;
    char inmsg[10],outmsg[]="hello";
    MPI_Status stat;

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);

    count=sizeof(outmsg)/sizeof(char);  //outmsg の要素数

    if(myid==0){
        src=1;   //送信元に 1 を指定
        dest=1;  //受信先に 1 を指定
        MPI_Send(&outmsg,count,MPI_CHAR,dest,tag,MPI_COMM_WORLD);
        MPI_Recv(&inmsg,count,MPI_CHAR,src,tag,MPI_COMM_WORLD,&stat);
        printf("%s from rank %d\n",&inmsg,src);
    }
    else{
        src=0;   //送信元に 0 を指定
        dest=0;  //受信先に 0 を指定
        MPI_Recv(&inmsg,count,MPI_CHAR,src,tag,MPI_COMM_WORLD,&stat);
        MPI_Send(&outmsg,count,MPI_CHAR,dest,tag,MPI_COMM_WORLD);
        printf("%s from rank %d\n",&inmsg,src);
    }

    MPI_Finalize();
    return 0;
}
```

1.2 結果

```
hello from rank 1
hello from rank 0
```

2 MPI_Isend,MPI_Irecv

2.1 プログラム例

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

int main(int argc,char *argv[])
{
    int myid,procs,src,dest,tag=1000,count;
    int data[100];
    MPI_Status stat;
    MPI_Request request;

    MPI_Init(&argc,&argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    //略（何らかのプラグラム操作）

    if(myid==0){
        src=1;
        dest=1;
        count=100;
        MPI_Irecv(&data,count,MPI_INT,tag,MPI_COMM_WORLD,&request);
        //略（何らかのプログラム操作）
        MPI_Wait(&request,&stat);
    }
    else{
        src=0;
        dest=0;
        count=100;
        MPI_Isend(&data,count,MPI_INT,src,tag,MPI_COMM_WORLD,&request);
        //略（何らかのプログラム操作）
        MPI_Wait(&request,&stat);
    }
    //略（何らかのプログラム操作）
    MPI_Finalize();
    return 0
}
```

3 MPI_Satter

3.1 プログラム例

```
1   #include "mpi.h"
2   #include <stdio.h>
3
4   int main(int argc, char *argv[])
5   {
6     int myid;
7     int send[6]={11,12,21,22,31,32},recv[2]={0,0};
8
9     MPI_Init(&argc,&argv); // MPI の実行環境の初期化
10    MPI_Comm_rank(MPI_COMM_WORLD,&myid);// コミュニケータの各プロセスが自分のランク (myid) を取得
11
12    printf("Rank:%d Value:%d %d \n",myid,recv[0],recv[1]);
13
14    MPI_Scatter(send,2,MPI_INT,recv,2,MPI_INT,0,MPI_COMM_WORLD);// データを全てのノードに送る
15
16    printf("Rank:%d Value:%d %d \n",myid,recv[0],recv[1]);
17
18    MPI_Finalize();
19    return 0;
20  }
```

3.2 結果

```
Rank:0 Value:0 0
Rank:1 Value:0 0
Rank:2 Value:0 0
Rank:0 Value:11 12
Rank:1 Value:21 22
Rank:2 Value:31 32
```

3.3 解説

7 行目で送信用の配列と受信用の配列を宣言し，それぞれ初期値を代入する．12 行目で送信前の受信バッファの値を出力．14 行目で関数 MPI_Scatter により，rank0 のノードが全てのノードに送信バッファの先頭アドレスから 2 つずつ値を送る．16 行目で受信後のそれぞれの値を出力．

4 MPI_Gather

4.1 プログラム例

```
1   #include "mpi.h"
2   #include <stdio.h>
3
4   int main(int argc, char *argv[])
5   {
6     int myid;
7     int send[6]={11,12,21,22,31,32},recv[2]={0,0}
8     int uni[6]={0,0,0,0,0,0};
9
10    MPI_Init(&argc,&argv); // MPI の実行環境の初期化
11    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ランク (myid) を取得
12
13    MPI_Scatter(send,2,MPI_INT,recv,2,MPI_INT,0,MPI_COMM_WORLD);
14
15    printf("Rank:%d Value:%d %d \n",myid,recv[0],recv[1]);
16
17    MPI_Gather(recv,2,MPI_INT,uni,2,MPI_INT,0,MPI_COMM_WORLD);// データを集める.
18
19    if(myid==0)
20      printf("Values : %d  %d  %d  %d  %d  %d\n",uni[0],uni[1],uni[2],uni[3],uni[4],uni[5]);
21
22    MPI_Finalize();
23    return 0;
24  }
```

4.2 結果

```
Rank:0 Value:11 12
Rank:1 Value:21 22
Rank:2 Value:31 32
Values : 11  12  21  22  31  32
```

4.3 解説

7 行目で送信用の配列 (send[6]) と受信用の配列 (recv[2]) を宣言．そして 8 行目で結合用の配列 (uni[6]) を宣言．13 行目で関数 MPI_Scatter により送信用の配列を各ノードの受信配列に送信する．15 行目で各ノードの受信バッファを出力．17 行目で関数 MPI_Gather により rank0 の結合用の配列 (uni[6]) に各受信用の配列 (recv[2]) を送る．20 行目に結合した配列を出力.

5 MPI_Barrier

MPI_Barrier は並列ジョブの経過測定時間の誤差を少なくするために用いられる．その例を以下に示す．

5.1 プログラム例

```
1  #include <stdio.h>
2  #include"mpi.h"
3
4  //通信を行う関数の定義
5  void clock(int myid,int buf,int tag,MPI_Status stat){
6    if(myid==0)
7      MPI_Send(&buf,1,MPI_INT,1,tag,MPI_COMM_WORLD);
8    else
9      MPI_Recv(&buf,1,MPI_INT,0,tag,MPI_COMM_WORLD,&stat);
10 }
11
12 int main(int argc, char *argv[]){
13   int myid,i,a,buf=0,tag;
14   double start,end;
15   MPI_Status stat;
16
17   MPI_Init(&argc,&argv);// MPI の実行環境の初期化
18   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ランク (myid) を取得
19
20   if(myid==0)   buf=1;
21   printf("BEFOUR---rank:%d buf:%d\n",myid,buf)//通信前の buf の値;
22
23 //同期をとらない計算の場合
24   if(myid==0) for(i=1;i<1000000;i++) a=i; //時間差を見るための計算 (処理 1)
25   start=MPI_Wtime();//計測開始
26   clock(myid,buf,tag,stat);//通信 (処理 2)
27   end=MPI_Wtime();//計測終了
28   printf("rank:%d NB-time=%fsec\n",myid,end-start); //計算時間の出力
29
30 //同期をとる計算の場合
31   if(myid==0) for(i=1;i<1000000;i++) a=i; //時間差を見るための計算 (処理 1)
32   MPI_Barrier(MPI_COMM_WORLD); //同期をとる
33   start=MPI_Wtime();//計測開始
34   clock(myid,buf,tag,stat);//通信 (処理 2)
35   end=MPI_Wtime();//計測終了
36   printf("rank:%d  B-time=%fsec\n",myid,end-start); //計算時間の出力
37
38   printf("AFTER ---rank:%d buf:%d\n",myid,buf);//通信後の buf の値
39   MPI_Finalize();
40   return 0;
41 }
```

5.2 結果

```
BEFOUR---rank:0 buf:1
BEFOUR---rank:1 buf:0
rank:0 NB-time=0.000016sec
rank:1 NB-time=0.001147sec
rank:0  B-time=0.000002sec
rank:1  B-time=0.000014sec
AFTER ---rank:0 buf:1
AFTER ---rank:1 buf:1
```

5.3 解説

5 から 10 行目までは処理 2 の為の通信を行う関数である．具体的には rank0 のノードが rank1 のノードにデータを 1 つ送っている．

23 行目から 27 行目は同期をとらない場合であり，rank0 のノードだけを働かすことにより，あとの処理を遅らせている．28 行目で同期をとらない場合の計測時間を出力している．ちなみに関数 MPI_Write は現在の時間を出力する関数である．

そして 31 行目から 35 行目は同期をとる場合であり，32 行目の関数 MPI_Barrier で同期をとっている．そして 36 行目で同期をとる場合の計測時間を出力している．

結果を見てわかるように同期をとる場合の方が計算時間の誤差が少なくなっている．

6 MPI_Bacst MPI_Reduce

6.1 プログラム例

```
1   #include "mpi.h"
2   #include <stdio.h>
3
4   void main(int argc, char *argv[]){
5     int myid,i;
6     int sendrev[2],recv[2];
7
8     MPI_Init(&argc,&argv);
9     MPI_Comm_rank(MPI_COMM_WORLD,&myid);
10
11    if(myid==0){sendrev[0]=1;sendrev[1]=2;}
12    else{sendrev[0]=0;sendrev[1]=0;}
13
14    printf("Rank:%d BEFORE-Value:%d %d \n",myid,sendrev[0],sendrev[1]);
15
16    MPI_Bcast(sendrev,2,MPI_INT,0,MPI_COMM_WORLD);
17
18    printf("Rank:%d  AFTER=Value:%d %d \n",myid,sendrev[0],sendrev[1]);
19
20    for(i=0;i<2;i++)  sendrev[i]+=3*myid;
21
22    printf("RANK:%d Value:%d %d\n",myid,sendrev[0],sendrev[1]);
23
24    MPI_Reduce(sendrev,recv,2,MPI_INT,MPI_SUM,0,MPI_COMM_WORLD);
25
26    if(myid==0) printf("%d %d\n",recv[0],recv[1]);
27
28    MPI_Reduce(sendrev,recv,2,MPI_INT,MPI_MAX,0,MPI_COMM_WORLD);
29
30    if(myid==0) printf(" %d  %d\n",recv[0],recv[1]);
31
32    MPI_Finalize();
33    return 0;
34  }
```

6.2 結果

```
Rank:0 BEFORE-Value:1 2
Rank:1 BEFORE-Value:0 0
Rank:2 BEFORE-Value:0 0
Rank:1 AFTER=Value:1 2
Rank:0 AFTER=Value:1 2
Rank:2 AFTER=Value:1 2
RANK:0 Value:1 2
RANK:1 Value:4 5
RANK:2 Value:7 8
SUM:12 15
MAX: 7 8
```

6.3 解説

11 行目で rank0 のノードに初期値 1 と 2 を配列 sendrev に代入．その他のノードには 0 を代入する．14 行目で送信前の配列 sendrev の値を出力 16 行目で関数 MPI_Bcast で rank0 の sendrev を全てのノードに送る．18 行目で送信後の sendrev の値を出力．20 行目で各ノード固有の計算を行い，22 行目で計算後の値を出力．24 行目で関数 MPI_Reduce で配列 sendrev の値を合計し，rank0 の配列 rev に代入する．26 行目でそれを出力する．28 行目で全てのノードの sendrev の内最大の数を rank0 の配列 rev に代入する．30 行目でそれを出力する．

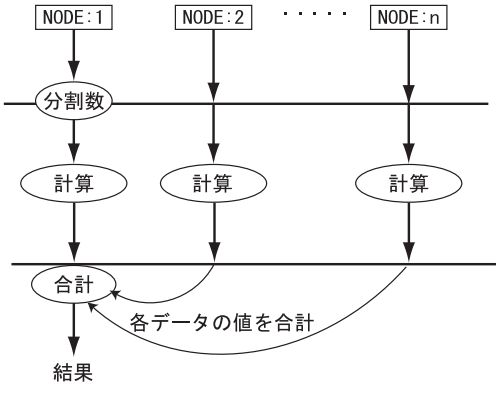

Fig. 1　MPI_Reduce の計算における通信

結果は各ノードのバッファの値の合計 (SUM) と最大値 (MAX) の値を出力させている．