第7章　MPIの集団通信

MPI で最も多用される関数は前節で扱った 1 対 1 通信関数と，ここで紹介する集団通信関数群である。この概念を理解しておくと，多数のノード間で同じデータを一度に送受信する処理が楽に書けるようになるので，是非ともモノにして頂きたい。

7.1　Bcast(ブロードキャスト)

全てのプロセスに同じ変数が宣言され，allocate されているとする。この時，ある一つのプロセスが保持しているデータを全プロセスに送信し共通化する処理が Bcast である。

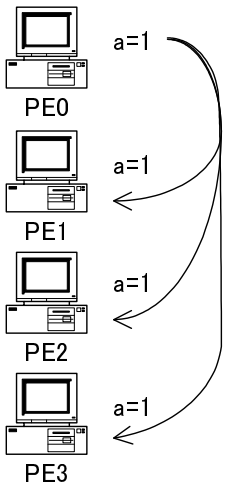

図 7.1: Bcast

```
1 : #include <stdio.h>
2 : #include <stdlib.h>
3 : #include <math.h>
4 :
```

```
 5 :#include "mpi.h"
 6 :
 7 :main(int argc, char *argv[])
 8 :{
 9 :    int num_procs, myrank;
10 :    double a, b;
11 :    int tag = 0;
12 :    MPI_Status status;
13 :
14 :    MPI_Init(&argc, &argv);
15 :
16 :    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
17 :    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
18 :
19 :    a = 0;
20 :    b = 0;
21 :    if(myrank == 0)
22 :        a = 1.0;
23 :
24 :    MPI_Bcast((void *)&a, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
25 :
26 :    printf("Process %d: a = %e, b = %e\n", myrank, a, b);
27 :
28 :    MPI_Finalize();
29 :
30 :    return EXIT_SUCCESS;
31 :}
32 :
```

このプログラムでは最初，PE0 の変数 a にのみ 1.0 が代入されている。これが MPI_Bcast 関数によって，全てのプロセスの変数 a に 1.0 が送信され，共通化されることになる。実行結果は以下の通りである。

```
% mpirun -np 4 ./mpi-bcast
Process 0: a = 1.000000e+00, b = 0.000000e+00
Process 1: a = 1.000000e+00, b = 0.000000e+00
Process 2: a = 1.000000e+00, b = 0.000000e+00
Process 3: a = 1.000000e+00, b = 0.000000e+00
%
```

多倍長化したものも同様である。

```
 1 :#include <stdio.h>
 2 :#include <stdlib.h>
 3 :#include <math.h>
```

```
 4 :
 5 : #include "mpi.h"
 6 :
 7 : #define USE_GMP
 8 : #define USE_MPFR
 9 : #include "mpi_bnc.h"
10 :
11 : int main(int argc, char *argv[])
12 : {
13 :     int num_procs, myrank;
14 :     mpf_t a, b;
15 :     void *buf;
16 :     int tag = 0;
17 :     MPI_Status status;
18 :
19 :     MPI_Init(&argc, &argv);
20 :
21 :     _mpi_set_bnc_default_prec_decimal(50, MPI_COMM_WORLD);
22 :     commit_mpf(&(MPI_MPF), ceil(50/log10(2.0)), MPI_COMM_WORLD);
23 :
24 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
25 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
26 :
27 :     mpf_init_set_ui(a, 0);
28 :     mpf_init_set_ui(b, 0);
29 :     if(myrank == 0)
30 :         mpf_set_ui(a, 1);
31 :
32 :     buf = allocbuf_mpf(mpf_get_prec(a), 1);
33 :     pack_mpf(a, 1, buf);
34 :     MPI_Bcast(buf, 1, MPI_MPF, tag, MPI_COMM_WORLD);
35 :     unpack_mpf(buf, a, 1);
36 :
37 :     printf("Process %d: a = ", myrank);
38 :     mpf_out_str(stdout, 10, 0, a);
39 :     printf(", b = ");
40 :     mpf_out_str(stdout, 10, 0, b);
41 :     printf("\n");
42 :
43 :     free(buf);
44 :     mpf_clear(a);
45 :     mpf_clear(b);
46 :     free_mpf(&(MPI_MPF));
47 :
48 :     MPI_Finalize();
49 :
```

```
 50 :     return EXIT_SUCCESS;
 51 : }
 52 :

% mpirun -np 4 ./mpi-bcast-gmp
------------------------------------------------------------------------
BNC Default Precision    : 167 bits(50.3 decimal digits)
BNC Default Rounding Mode: Round to Nearest
------------------------------------------------------------------------
Process 0: a = 1.000000000000000000000000000000000000000000000000, b = 0
Process 1: a = 1.000000000000000000000000000000000000000000000000, b = 0
Process 2: a = 1.000000000000000000000000000000000000000000000000, b = 0
Process 3: a = 1.000000000000000000000000000000000000000000000000, b = 0
%
```

7.2　Gather(ギャザ)

その名の通り，全てのプロセスが保持している同変数のデータを，あるプロセスへ集める (gather) 処理である。

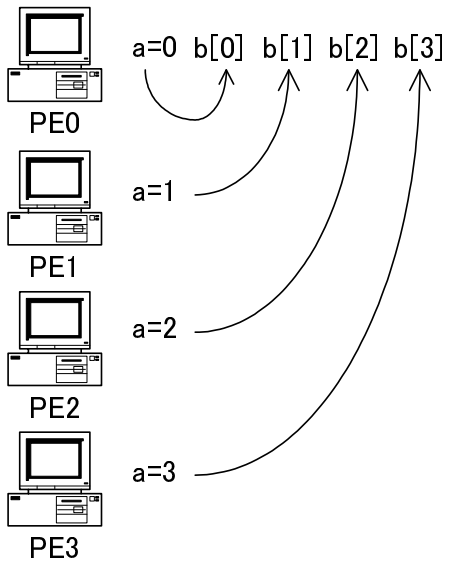

図 7.2: Gather

この例では，各プロセスの変数aに入っているデータを，PE0の配列b[0]〜b[3]へ集約している。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #include "mpi.h"
 6 :
 7 : int main(int argc, char *argv[])
 8 : {
 9 :     int num_procs, myrank;
10 :     double a, b[128];
11 :     int tag = 0, i;
12 :     MPI_Status status;
13 :
14 :     MPI_Init(&argc, &argv);
15 :
16 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
17 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
18 :
19 :     a = (double)myrank;
20 :
21 :     MPI_Gather((void *)&a, 1, MPI_DOUBLE, (void *)&b, 1, MPI_DOUB
   LE, 0, MPI_COMM_WORLD);
22 :
23 :     printf("Process %d: a = %e\n", myrank, a);
24 :     if(myrank == 0)
25 :         for(i = 0; i < num_procs; i++)
26 :             printf("b[%d] = %e\n", i, b[i]);
27 :
28 :     MPI_Finalize();
29 :
30 :     return EXIT_SUCCESS;
31 : }
32 :
```

```
% mpirun -np 4 ./mpi-gather
Process 0: a = 0.000000e+00
b[0] = 0.000000e+00
b[1] = 1.000000e+00
b[2] = 2.000000e+00
b[3] = 3.000000e+00
Process 1: a = 1.000000e+00
```

```
Process 3: a = 3.000000e+00
Process 2: a = 2.000000e+00
%
```

多倍長化したものは以下の通り。

```
 1 :#include <stdio.h>
 2 :#include <stdlib.h>
 3 :#include <math.h>
 4 :
 5 :#include "mpi.h"
 6 :
 7 :#define USE_GMP
 8 :#define USE_MPFR
 9 :#include "mpi_bnc.h"
10 :
11 :int main(int argc, char *argv[])
12 :{
13 :    int num_procs, myrank;
14 :    mpf_t a, b[128];
15 :    void *buf_a, *buf_b;
16 :    int tag = 0, i;
17 :    MPI_Status status;
18 :
19 :    MPI_Init(&argc, &argv);
20 :
21 :    _mpi_set_bnc_default_prec_decimal(50, MPI_COMM_WORLD);
22 :    commit_mpf(&(MPI_MPF), ceil(50/log10(2.0)), MPI_COMM_WORLD);
23 :
24 :    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
25 :    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
26 :
27 :    mpf_init_set_ui(a, (unsigned long)myrank);
28 :    buf_a = allocbuf_mpf(mpf_get_prec(a), 1);
29 :    buf_b = allocbuf_mpf(mpf_get_prec(a), num_procs);
30 :
31 :    pack_mpf(a, 1, buf_a);
32 :    MPI_Gather(buf_a, 1, MPI_MPF, buf_b, 1, MPI_MPF, 0, MPI_COMM_
   WORLD);
33 :
34 :    if(myrank == 0)
35 :    {
36 :        for(i = 0; i < num_procs; i++)
37 :            mpf_init(b[i]);
38 :        unpack_mpf(buf_b, b[0], num_procs);
39 :    }
```

```
 40 :
 41 :     printf("Process %d: a = ", myrank);
 42 :     mpf_out_str(stdout, 10, 0, a);
 43 :     printf("\n");
 44 :     if(myrank == 0)
 45 :     {
 46 :         for(i = 0; i < num_procs; i++)
 47 :         {
 48 :             printf("b[%d] = ", i);
 49 :             mpf_out_str(stdout, 10, 0, b[i]);
 50 :             printf("\n");
 51 :         }
 52 :     }
 53 :
 54 :     free(buf_a);
 55 :     free(buf_b);
 56 :
 57 :     mpf_clear(a);
 58 :     if(myrank == 0)
 59 :     {
 60 :         for(i = 0; i < num_procs; i++)
 61 :             mpf_clear(b[i]);
 62 :     }
 63 :
 64 :     free_mpf(&(MPI_MPF));
 65 :
 66 :     MPI_Finalize();
 67 :
 68 :     return EXIT_SUCCESS;
 69 : }
 70 :
```

このプログラムの実行結果は以下の通りである。

```
 mpirun -np 4 ./mpi-gather-gmp
--------------------------------------------------------------------------------
BNC Default Precision    : 167 bits(50.3 decimal digits)
BNC Default Rounding Mode: Round to Nearest
--------------------------------------------------------------------------------
Process 0: a = 0
b[0] = 0
b[1] = 1.00000000000000000000000000000000000000000000000000
b[2] = 2.00000000000000000000000000000000000000000000000000
b[3] = 3.00000000000000000000000000000000000000000000000000
```

```
Process 2: a = 2.000000000000000000000000000000000000000000000000000000
Process 1: a = 1.000000000000000000000000000000000000000000000000000000
Process 3: a = 3.000000000000000000000000000000000000000000000000000000
%
```

7.3　Scatter(スキャタ)

Gather とは逆に，特定のプロセスのデータを他のプロセスへばらまく (scatter) する機能である。Bcast と似ているが，異なる名前の変数間でのやりとりになる点が異なっている。

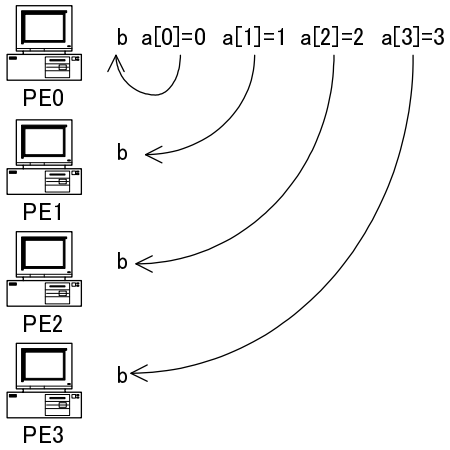

図 7.3: Scatter

この例では，PE0 の b[0]〜b[3] のデータが，各プロセスの変数 a へ，一個ずつばらまかれている。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #include "mpi.h"
 6 :
 7 : int main(int argc, char *argv[])
 8 : {
 9 :     int num_procs, myrank;
10 :     double a[128], b;
11 :     int tag = 0, i;
```

```
 12 :     MPI_Status status;
 13 :
 14 :     MPI_Init(&argc, &argv);
 15 :
 16 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
 17 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 18 :
 19 :     if(myrank == 0)
 20 :         for(i = 0; i < num_procs; i++)
 21 :             a[i] = (double)i;
 22 :
 23 :     MPI_Scatter((void *)&a, 1, MPI_DOUBLE, (void *)&b, 1, MPI_DOU
    BLE, 0, MPI_COMM_WORLD);
 24 :
 25 :     printf("Process %d: b = %e\n", myrank, b);
 26 :
 27 :     MPI_Finalize();
 28 :
 29 :     return EXIT_SUCCESS;
 30 : }
 31 :
```

```
% mpirun -np 4 ./mpi-scatter
Process 0: b = 0.000000e+00
Process 2: b = 2.000000e+00
Process 1: b = 1.000000e+00
Process 3: b = 3.000000e+00
%
```

多倍長化したものは以下の通り。

```
  1 : #include <stdio.h>
  2 : #include <stdlib.h>
  3 : #include <math.h>
  4 :
  5 : #include "mpi.h"
  6 :
  7 : #define USE_GMP
  8 : #define USE_MPFR
  9 : #include "mpi_bnc.h"
 10 :
 11 : int main(int argc, char *argv[])
 12 : {
 13 :     int num_procs, myrank;
 14 :     mpf_t a[128], b;
```

```
 15 :     void *buf_a, *buf_b;
 16 :     int tag = 0, i;
 17 :     MPI_Status status;
 18 :
 19 :     MPI_Init(&argc, &argv);
 20 :
 21 :     _mpi_set_bnc_default_prec_decimal(50, MPI_COMM_WORLD);
 22 :     commit_mpf(&(MPI_MPF), ceil(50/log10(2.0)), MPI_COMM_WORLD);
 23 :
 24 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
 25 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 26 :
 27 :     if(myrank == 0)
 28 :     {
 29 :         for(i = 0; i < num_procs; i++)
 30 :             mpf_init_set_ui(a[i], i);
 31 :         buf_a = allocbuf_mpf(mpf_get_prec(a[0]), num_procs);
 32 :         pack_mpf(a[0], num_procs, buf_a);
 33 :     }
 34 :     mpf_init(b);
 35 :     buf_b = allocbuf_mpf(mpf_get_prec(b), 1);
 36 :
 37 :     MPI_Scatter(buf_a, 1, MPI_MPF, buf_b, 1, MPI_MPF, 0, MPI_COMM
    _WORLD);
 38 :     unpack_mpf(buf_b, b, 1);
 39 :
 40 :     printf("Process %d: b =", myrank);
 41 :     mpf_out_str(stdout, 10, 0, b);
 42 :     printf("\n");
 43 :
 44 :     MPI_Finalize();
 45 :
 46 :     return EXIT_SUCCESS;
 47 :}
 48 :
```

実行結果は以下の通りである。

```
% mpirun -np 4 ./mpi-scatter-gmp
------------------------------------------------------------------------
BNC Default Precision    : 167 bits(50.3 decimal digits)
BNC Default Rounding Mode: Round to Nearest
------------------------------------------------------------------------
Process 0: b =0
Process 1: b =1.00000000000000000000000000000000000000000000000000
```

```
Process 2: b =2.000000000000000000000000000000000000000000000000
Process 3: b =3.000000000000000000000000000000000000000000000000
%
```

7.4 Reduce(レデュース)

Gather に似ているが，特定のプロセスに集約する際に演算を併せて行う点が異なる。

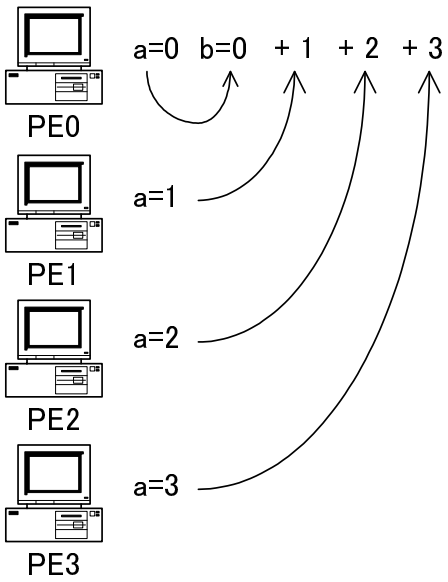

図 7.4: Reduce

例えばこの例では，全てのプロセスから変数 a のデータを集約して和を計算し，その結果を PE0 の変数 b へ代入している。もちろん和以外の計算を行うことも可能であるが，本書では和のみ扱うことにする。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #include "mpi.h"
 6 :
 7 : int main(int argc, char *argv[])
 8 : {
 9 :     int num_procs, myrank;
10 :     double a, b;
```

```
 11 :     int tag = 0, i;
 12 :     MPI_Status status;
 13 :
 14 :     MPI_Init(&argc, &argv);
 15 :
 16 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
 17 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 18 :
 19 :     a = (double)myrank;
 20 :
 21 :     MPI_Reduce((void *)&a, (void *)&b, 1, MPI_DOUBLE, MPI_SUM, 0,
    MPI_COMM_WORLD);
 22 :
 23 :     printf("Process %d: a = %e\n", myrank, a);
 24 :     if(myrank == 0)
 25 :         printf("b = %e\n", b);
 26 :
 27 :     MPI_Finalize();
 28 :
 29 :     return EXIT_SUCCESS;
 30 :}
 31 :
```

```
% mpirun -np 4 ./mpi-reduce
Process 0: a = 0.000000e+00
b = 6.000000e+00
Process 1: a = 1.000000e+00
Process 2: a = 2.000000e+00
Process 3: a = 3.000000e+00
%
```

これを多倍長化したものは以下の通り。

```
  1 :#include <stdio.h>
  2 :#include <stdlib.h>
  3 :#include <math.h>
  4 :
  5 :#include "mpi.h"
  6 :
  7 :#define USE_GMP
  8 :#define USE_MPFR
  9 :#include "mpi_bnc.h"
 10 :
 11 :int main(int argc, char *argv[])
 12 :{
```

```
13 :     int num_procs, myrank;
14 :     mpf_t a, b;
15 :     void *buf_a, *buf_b;
16 :     int tag = 0, i;
17 :     MPI_Status status;
18 :
19 :     MPI_Init(&argc, &argv);
20 :
21 :     _mpi_set_bnc_default_prec_decimal(50, MPI_COMM_WORLD);
22 :     commit_mpf(&(MPI_MPF), ceil(50/log10(2.0)), MPI_COMM_WORLD);
23 :     create_mpf_op(&(MPI_MPF_SUM), _mpi_mpf_add, MPI_COMM_WORLD);
24 :
25 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
26 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
27 :
28 :     mpf_init_set_ui(a, myrank);
29 :     mpf_init(b);
30 :
31 :     buf_a = allocbuf_mpf(mpf_get_prec(a), 1);
32 :     buf_b = allocbuf_mpf(mpf_get_prec(b), 1);
33 :
34 :     pack_mpf(a, 1, buf_a);
35 :     MPI_Reduce(buf_a, buf_b, 1, MPI_MPF, MPI_MPF_SUM, 0, MPI_COMM
   _WORLD);
36 :
37 :     printf("Process %d: a = ", myrank);
38 :     mpf_out_str(stdout, 10, 0, a);
39 :     printf("\n");
40 :
41 :     if(myrank == 0)
42 :     {
43 :         unpack_mpf(buf_b, b, 1);
44 :         printf("b = ");
45 :         mpf_out_str(stdout, 10, 0, b);
46 :         printf("\n");
47 :     }
48 :
49 :     free(buf_a);
50 :     free(buf_b);
51 :
52 :     mpf_clear(a);
53 :     mpf_clear(b);
54 :
55 :     free_mpf_op(&(MPI_MPF_SUM));
56 :     free_mpf(&(MPI_MPF));
57 :
```

```
 58 :     MPI_Finalize();
 59 :
 60 :     return EXIT_SUCCESS;
 61 :}
 62 :
```

実行結果は次のようになる。

```
% mpirun -np 4 ./mpi-reduce-gmp
--------------------------------------------------------------------------------
BNC Default Precision    : 167 bits(50.3 decimal digits)
BNC Default Rounding Mode: Round to Nearest
--------------------------------------------------------------------------------
Process 0: a = 0
b = 6.000000000000000000000000000000000000000000000000
Process 1: a = 1.000000000000000000000000000000000000000000000000
Process 2: a = 2.000000000000000000000000000000000000000000000000
Process 3: a = 3.000000000000000000000000000000000000000000000000
%
```

7.5　Allgather(オールギャザ)

Gather の機能に Bcast を合わせたもので，全てのプロセスにある変数のデータを集約し，それを全部のプロセスで共通化するのである。

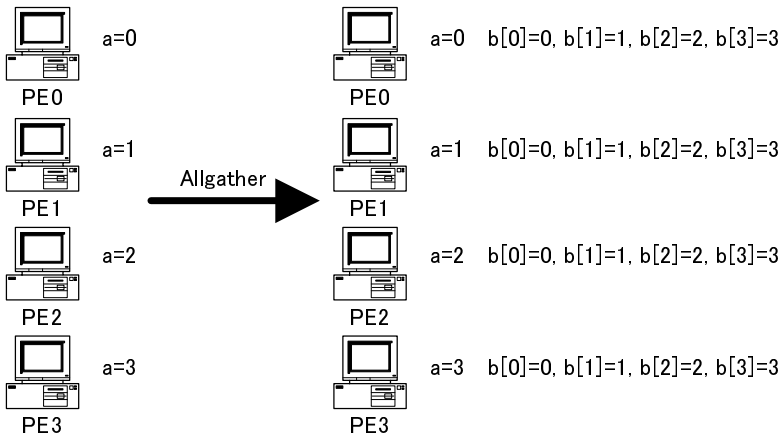

図 7.5: Allgather

この例では，全プロセスの変数 a のデータを変数 b[0]〜b[3] へ集約し，全プロセスでそれを共通化している。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #include "mpi.h"
 6 :
 7 : int main(int argc, char *argv[])
 8 : {
 9 :     int num_procs, myrank;
10 :     double a, b[128];
11 :     int tag = 0, i;
12 :     MPI_Status status;
13 :
14 :     MPI_Init(&argc, &argv);
15 :
16 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
17 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
18 :
19 :     a = (double)myrank;
20 :
21 :     MPI_Allgather((void *)&a, 1, MPI_DOUBLE, (void *)&b, 1, MPI_D
   OUBLE, MPI_COMM_WORLD);
22 :
23 :     printf("Process %d: a = %e\n", myrank, a);
24 :     for(i = 0; i < num_procs; i++)
25 :         printf("b[%d] = %e\n", i, b[i]);
26 :
27 :     MPI_Finalize();
28 :
29 :     return EXIT_SUCCESS;
30 : }
31 :
```

```
% mpirun -np 4 ./mpi-allgather
Process 0: a = 0.000000e+00
b[0] = 0.000000e+00
b[1] = 1.000000e+00
b[2] = 2.000000e+00
b[3] = 3.000000e+00
Process 1: a = 1.000000e+00
b[0] = 0.000000e+00
```

```
b[1] = 1.000000e+00
b[2] = 2.000000e+00
b[3] = 3.000000e+00
Process 2: a = 2.000000e+00
b[0] = 0.000000e+00
b[1] = 1.000000e+00
b[2] = 2.000000e+00
b[3] = 3.000000e+00
Process 3: a = 3.000000e+00
b[0] = 0.000000e+00
b[1] = 1.000000e+00
b[2] = 2.000000e+00
b[3] = 3.000000e+00
%
```

多倍長化したものは以下の通り。

```
 1 :#include <stdio.h>
 2 :#include <stdlib.h>
 3 :#include <math.h>
 4 :
 5 :#include "mpi.h"
 6 :
 7 :#define USE_GMP
 8 :#define USE_MPFR
 9 :#include "mpi_bnc.h"
10 :
11 :int main(int argc, char *argv[])
12 :{
13 :    int num_procs, myrank;
14 :    mpf_t a, b[128];
15 :    void *buf_a, *buf_b;
16 :    int tag = 0, i;
17 :    MPI_Status status;
18 :
19 :    MPI_Init(&argc, &argv);
20 :
21 :    _mpi_set_bnc_default_prec_decimal(50, MPI_COMM_WORLD);
22 :    commit_mpf(&(MPI_MPF), ceil(50/log10(2.0)), MPI_COMM_WORLD);
23 :
24 :    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
25 :    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
26 :
```

```
 27 :     mpf_init_set_ui(a, myrank);
 28 :     for(i = 0; i < num_procs; i++)
 29 :         mpf_init(b[i]);
 30 :
 31 :     buf_a = allocbuf_mpf(mpf_get_prec(a), 1);
 32 :     buf_b = allocbuf_mpf(mpf_get_prec(b[0]), num_procs);
 33 :
 34 :     pack_mpf(a, 1, buf_a);
 35 :     MPI_Allgather(buf_a, 1, MPI_MPF, buf_b, 1, MPI_MPF, MPI_COMM_
    WORLD);
 36 :     unpack_mpf(buf_b, b[0], num_procs);
 37 :
 38 :     printf("Process %d: a = ", myrank);
 39 :     mpf_out_str(stdout, 10, 0, a);
 40 :     printf("\n");
 41 :     for(i = 0; i < num_procs; i++)
 42 :     {
 43 :         printf("b[%d] = ", i);
 44 :         mpf_out_str(stdout, 10, 0, b[i]);
 45 :         printf("\n");
 46 :     }
 47 :
 48 :     free(buf_a);
 49 :     free(buf_b);
 50 :
 51 :     mpf_clear(a);
 52 :     for(i = 0; i < num_procs; i++)
 53 :         mpf_clear(b[i]);
 54 :
 55 :     free_mpf(&(MPI_MPF));
 56 :
 57 :     MPI_Finalize();
 58 :
 59 :     return EXIT_SUCCESS;
 60 : }
 61 :
```

実行結果は次の通りである。

```
% mpirun -np 4 ./mpi-allgather-gmp
------------------------------------------------------------------------
BNC Default Precision    : 167 bits(50.3 decimal digits)
BNC Default Rounding Mode: Round to Nearest
------------------------------------------------------------------------
Process 0: a = 0
```

```
b[0] = 0
b[1] = 1.000000000000000000000000000000000000000000000000
b[2] = 2.000000000000000000000000000000000000000000000000
b[3] = 3.000000000000000000000000000000000000000000000000
Process 1: a = 1.0000000000000000000000000000000000000000000000000000
b[0] = 0
b[1] = 1.000000000000000000000000000000000000000000000000
b[2] = 2.000000000000000000000000000000000000000000000000
b[3] = 3.000000000000000000000000000000000000000000000000
Process 3: a = 3.0000000000000000000000000000000000000000000000000000
b[0] = 0
b[1] = 1.000000000000000000000000000000000000000000000000
b[2] = 2.000000000000000000000000000000000000000000000000
b[3] = 3.000000000000000000000000000000000000000000000000
Process 2: a = 2.0000000000000000000000000000000000000000000000000000
b[0] = 0
b[1] = 1.000000000000000000000000000000000000000000000000
b[2] = 2.000000000000000000000000000000000000000000000000
b[3] = 3.000000000000000000000000000000000000000000000000
%
```

7.6　Allreduce(オールレデュース)

Allreduce は Reduce した結果を全プロセスで共通化するものである。

この例では，全プロセスの変数 a のデータを集約して和を取り，その結果を全プロセスの変数 b に持たせている。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #include "mpi.h"
 6 :
 7 : int main(int argc, char *argv[])
 8 : {
 9 :     int num_procs, myrank;
10 :     double a, b;
11 :     int tag = 0, i;
12 :     MPI_Status status;
```

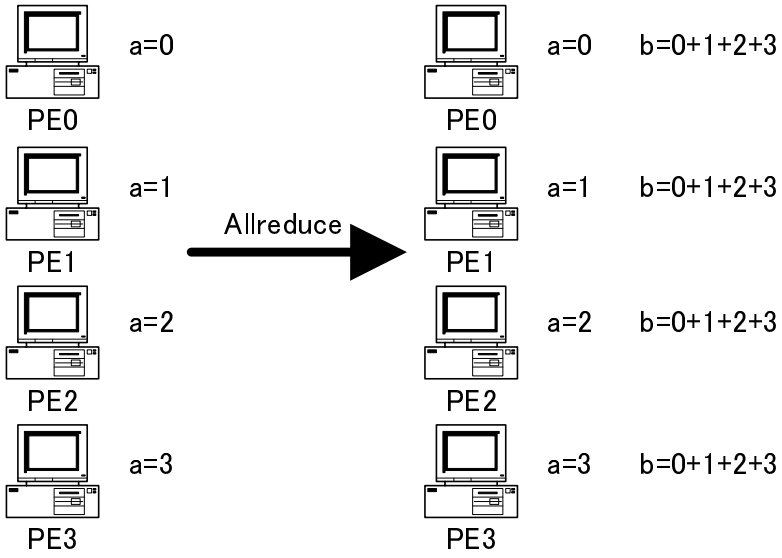

図 7.6: Allreduce

```
 13 :
 14 :     MPI_Init(&argc, &argv);
 15 :
 16 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
 17 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 18 :
 19 :     a = (double)myrank;
 20 :
 21 :     MPI_Allreduce((void *)&a, (void *)&b, 1, MPI_DOUBLE, MPI_SUM,
     MPI_COMM_WORLD);
 22 :
 23 :     printf("Process %d: a = %e, b = %e\n", myrank, a, b);
 24 :
 25 :     MPI_Finalize();
 26 :
 27 :     return EXIT_SUCCESS;
 28 : }
 29 :
```

```
% mpirun -np 4 ./mpi-allreduce
Process 0: a = 0.000000e+00, b = 6.000000e+00
Process 2: a = 2.000000e+00, b = 6.000000e+00
Process 3: a = 3.000000e+00, b = 6.000000e+00
Process 1: a = 1.000000e+00, b = 6.000000e+00
```

%

多倍長化したものは以下の通り。

```
 1 :#include <stdio.h>
 2 :#include <stdlib.h>
 3 :#include <math.h>
 4 :
 5 :#include "mpi.h"
 6 :
 7 :#define USE_GMP
 8 :#define USE_MPFR
 9 :#include "mpi_bnc.h"
10 :
11 :int main(int argc, char *argv[])
12 :{
13 :    int num_procs, myrank;
14 :    mpf_t a, b;
15 :    void *buf_a, *buf_b;
16 :    int tag = 0, i;
17 :    MPI_Status status;
18 :
19 :    MPI_Init(&argc, &argv);
20 :
21 :    _mpi_set_bnc_default_prec_decimal(50, MPI_COMM_WORLD);
22 :    commit_mpf(&(MPI_MPF), ceil(50/log10(2.0)), MPI_COMM_WORLD);
23 :    create_mpf_op(&(MPI_MPF_SUM), _mpi_mpf_add, MPI_COMM_WORLD);
24 :
25 :    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
26 :    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
27 :
28 :    mpf_init_set_ui(a, myrank);
29 :    mpf_init(b);
30 :
31 :    buf_a = allocbuf_mpf(mpf_get_prec(a), 1);
32 :    buf_b = allocbuf_mpf(mpf_get_prec(b), 1);
33 :
34 :    pack_mpf(a, 1, buf_a);
35 :    MPI_Allreduce(buf_a, buf_b, 1, MPI_MPF, MPI_MPF_SUM, MPI_COMM
   _WORLD);
36 :    unpack_mpf(buf_b, b, 1);
37 :
38 :    printf("Process %d: a = ", myrank);
39 :    mpf_out_str(stdout, 10, 0, a);
40 :    printf(", b = ");
41 :    mpf_out_str(stdout, 10, 0, b);
42 :    printf("\n");
```

```
 43 :
 44 :     free(buf_a);
 45 :     free(buf_b);
 46 :
 47 :     mpf_clear(a);
 48 :     mpf_clear(b);
 49 :
 50 :     free_mpf_op(&(MPI_MPF_SUM));
 51 :     free_mpf(&(MPI_MPF));
 52 :
 53 :     MPI_Finalize();
 54 :
 55 :     return EXIT_SUCCESS;
 56 : }
 57 :
```

実行結果は以下の通りである。

```
% mpirun -np 4 ./mpi-allreduce-gmp
------------------------------------------------------------------------------
BNC Default Precision    : 167 bits(50.3 decimal digits)
BNC Default Rounding Mode: Round to Nearest
------------------------------------------------------------------------------
Process 0: a = 0, b = 6.000000000000000000000000000000000000000000000000
Process 1: a = 1.000000000000000000000000000000000000000000000000, b = 6.0
00000000000000000000000000000000000000000000
Process 2: a = 2.000000000000000000000000000000000000000000000000, b = 6.0
00000000000000000000000000000000000000000000
Process 3: a = 3.000000000000000000000000000000000000000000000000, b = 6.0
00000000000000000000000000000000000000000000
%
```

7.7 Alltoall

最後は Alltoall 処理であるが，これは次の二つのソースファイル (mpi-alltoall.c, mpi-alltoall2.c) とその実行結果を見比べて何をしているか考えて頂きたい (→演習問題)。

```
  1 : #include <stdio.h>
  2 : #include <stdlib.h>
  3 : #include <math.h>
```

```
 4 :
 5 :#include "mpi.h"
 6 :
 7 :int main(int argc, char *argv[])
 8 :{
 9 :    int num_procs, myrank;
10 :    double a[128], b[128];
11 :    int tag = 0, i;
12 :    MPI_Status status;
13 :
14 :    MPI_Init(&argc, &argv);
15 :
16 :    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
17 :    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
18 :
19 :    for(i = 0; i < num_procs; i++)
20 :        a[i] = (double)i;
21 :
22 :    MPI_Alltoall((void *)&a, 1, MPI_DOUBLE, (void *)&b, 1, MPI_DO
   UBLE, MPI_COMM_WORLD);
23 :
24 :    printf("Process %d:\n", myrank);
25 :    for(i = 0; i < num_procs; i++)
26 :        printf("a[%d] = %e, b[%d] = %e\n", i, a[i], i, b[i]);
27 :
28 :    MPI_Finalize();
29 :
30 :    return EXIT_SUCCESS;
31 :}
32 :
```

この実行結果は以下の通りである。

```
% mpirun -np 4 ./mpi-alltoall
Process 0:
a[0] = 0.000000e+00, b[0] = 0.000000e+00
a[1] = 1.000000e+00, b[1] = 0.000000e+00
a[2] = 2.000000e+00, b[2] = 0.000000e+00
a[3] = 3.000000e+00, b[3] = 0.000000e+00
Process 1:
a[0] = 0.000000e+00, b[0] = 1.000000e+00
a[1] = 1.000000e+00, b[1] = 1.000000e+00
a[2] = 2.000000e+00, b[2] = 1.000000e+00
```

```
a[3] = 3.000000e+00, b[3] = 1.000000e+00
Process 2:
a[0] = 0.000000e+00, b[0] = 2.000000e+00
a[1] = 1.000000e+00, b[1] = 2.000000e+00
a[2] = 2.000000e+00, b[2] = 2.000000e+00
a[3] = 3.000000e+00, b[3] = 2.000000e+00
Process 3:
a[0] = 0.000000e+00, b[0] = 3.000000e+00
a[1] = 1.000000e+00, b[1] = 3.000000e+00
a[2] = 2.000000e+00, b[2] = 3.000000e+00
a[3] = 3.000000e+00, b[3] = 3.000000e+00
%
```

多倍長化したものは以下の通り。

```
 1 :#include <stdio.h>
 2 :#include <stdlib.h>
 3 :#include <math.h>
 4 :
 5 :#include "mpi.h"
 6 :
 7 :int main(int argc, char *argv[])
 8 :{
 9 :    int num_procs, myrank;
10 :    double a[128], b[128];
11 :    int tag = 0, i;
12 :    MPI_Status status;
13 :
14 :    MPI_Init(&argc, &argv);
15 :
16 :    MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
17 :    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
18 :
19 :    for(i = 0; i < num_procs; i++)
20 :        a[i] = (double)myrank;
21 :
22 :    MPI_Alltoall((void *)&a, 1, MPI_DOUBLE, (void *)&b, 1, MPI_DO
   UBLE, MPI_COMM_WORLD);
23 :
24 :    printf("Process %d:\n", myrank);
25 :    for(i = 0; i < num_procs; i++)
26 :        printf("a[%d] = %e, b[%d] = %e\n", i, a[i], i, b[i]);
27 :
```

```
 28 :     MPI_Finalize();
 29 :
 30 :     return EXIT_SUCCESS;
 31 : }
 32 :
```

この実行結果は以下の通りである。

```
% mpirun -np 4 ./mpi-alltoall2
Process 0:
a[0] = 0.000000e+00, b[0] = 0.000000e+00
a[1] = 0.000000e+00, b[1] = 1.000000e+00
a[2] = 0.000000e+00, b[2] = 2.000000e+00
a[3] = 0.000000e+00, b[3] = 3.000000e+00
Process 2:
a[0] = 2.000000e+00, b[0] = 0.000000e+00
a[1] = 2.000000e+00, b[1] = 1.000000e+00
a[2] = 2.000000e+00, b[2] = 2.000000e+00
a[3] = 2.000000e+00, b[3] = 3.000000e+00
Process 1:
a[0] = 1.000000e+00, b[0] = 0.000000e+00
a[1] = 1.000000e+00, b[1] = 1.000000e+00
a[2] = 1.000000e+00, b[2] = 2.000000e+00
a[3] = 1.000000e+00, b[3] = 3.000000e+00
Process 3:
a[0] = 3.000000e+00, b[0] = 0.000000e+00
a[1] = 3.000000e+00, b[1] = 1.000000e+00
a[2] = 3.000000e+00, b[2] = 2.000000e+00
a[3] = 3.000000e+00, b[3] = 3.000000e+00
%
```

同じく，Alltoall 通信を行っている別のプログラムを以下に示す。

```
  1 : #include <stdio.h>
  2 : #include <stdlib.h>
  3 : #include <math.h>
  4 :
  5 : #include "mpi.h"
  6 :
  7 : #define USE_GMP
```

```
 8 : #define USE_MPFR
 9 : #include "mpi_bnc.h"
10 :
11 : int main(int argc, char *argv[])
12 : {
13 :     int num_procs, myrank;
14 :     mpf_t a[128], b[128];
15 :     void *buf_a, *buf_b;
16 :     int tag = 0, i;
17 :     MPI_Status status;
18 :
19 :     MPI_Init(&argc, &argv);
20 :
21 :     _mpi_set_bnc_default_prec_decimal(50, MPI_COMM_WORLD);
22 :     commit_mpf(&(MPI_MPF), ceil(50/log10(2.0)), MPI_COMM_WORLD);
23 :
24 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
25 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
26 :
27 :     for(i = 0; i < num_procs; i++)
28 :     {
29 :         mpf_init_set_ui(a[i], i);
30 :         mpf_init(b[i]);
31 :     }
32 :
33 :     buf_a = allocbuf_mpf(mpf_get_prec(a[0]), num_procs);
34 :     buf_b = allocbuf_mpf(mpf_get_prec(b[0]), num_procs);
35 :
36 :     pack_mpf(a[0], num_procs, buf_a);
37 :     MPI_Alltoall(buf_a, 1, MPI_MPF, buf_b, 1, MPI_MPF, MPI_COMM_W
   ORLD);
38 :     unpack_mpf(buf_b, b[0], num_procs);
39 :
40 :     printf("Process %d:\n", myrank);
41 :     for(i = 0; i < num_procs; i++)
42 :     {
43 :         printf("a[%d] = ", i);
44 :         mpf_out_str(stdout, 10, 0, a[i]);
45 :         printf(", b[%d] = ", i);
46 :         mpf_out_str(stdout, 10, 0, b[i]);
47 :         printf("\n");
48 :     }
49 :
50 :     free(buf_a);
51 :     free(buf_b);
52 :
```

```
53 :     for(i = 0; i < num_procs; i++)
54 :     {
55 :         mpf_clear(a[i]);
56 :         mpf_clear(b[i]);
57 :     }
58 :
59 :     free_mpf(&(MPI_MPF));
60 :
61 :     MPI_Finalize();
62 :
63 :     return EXIT_SUCCESS;
64 :}
65 :
```

多倍長化したものは以下の通り。

```
 1 :#include <stdio.h>
 2 :#include <stdlib.h>
 3 :#include <math.h>
 4 :
 5 :#include "mpi.h"
 6 :
 7 :#define USE_GMP
 8 :#define USE_MPFR
 9 :#include "mpi_bnc.h"
10 :
11 :int main(int argc, char *argv[])
12 :{
13 :     int num_procs, myrank;
14 :     mpf_t a[128], b[128];
15 :     void *buf_a, *buf_b;
16 :     int tag = 0, i;
17 :     MPI_Status status;
18 :
19 :     MPI_Init(&argc, &argv);
20 :
21 :     _mpi_set_bnc_default_prec_decimal(50, MPI_COMM_WORLD);
22 :     commit_mpf(&(MPI_MPF), ceil(50/log10(2.0)), MPI_COMM_WORLD);
23 :
24 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
25 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
26 :
27 :     for(i = 0; i < num_procs; i++)
28 :     {
29 :         mpf_init_set_ui(a[i], myrank);
30 :         mpf_init(b[i]);
```

```
31 :     }
32 :
33 :     buf_a = allocbuf_mpf(mpf_get_prec(a[0]), num_procs);
34 :     buf_b = allocbuf_mpf(mpf_get_prec(b[0]), num_procs);
35 :
36 :     pack_mpf(a[0], num_procs, buf_a);
37 :     MPI_Alltoall(buf_a, 1, MPI_MPF, buf_b, 1, MPI_MPF, MPI_COMM_W
   ORLD);
38 :     unpack_mpf(buf_b, b[0], num_procs);
39 :
40 :     printf("Process %d:\n", myrank);
41 :     for(i = 0; i < num_procs; i++)
42 :     {
43 :         printf("a[%d] = ", i);
44 :         mpf_out_str(stdout, 10, 0, a[i]);
45 :         printf(", b[%d] = ", i);
46 :         mpf_out_str(stdout, 10, 0, b[i]);
47 :         printf("\n");
48 :     }
49 :
50 :     free(buf_a);
51 :     free(buf_b);
52 :
53 :     for(i = 0; i < num_procs; i++)
54 :     {
55 :         mpf_clear(a[i]);
56 :         mpf_clear(b[i]);
57 :     }
58 :
59 :     free_mpf(&(MPI_MPF));
60 :
61 :     MPI_Finalize();
62 :
63 :     return EXIT_SUCCESS;
64 : }
65 :
```

演習問題

1. mpi-alltoall.c, mpi-alltoall-gmp.c の結果を元に Alltoall の処理を解説せよ。

2. mpi-alltoall2.c, mpi-alltoall-gmp2.c の実行結果を予測し，実際の結果と比較照合せよ。

3. Allgather, Allreduce の処理はそれぞれ Gather, Reduce に Bcast を組み合わせることで実現できる。その処理をプログラム化し，Allgather, Allreduce との実行時間を比較検討せよ。[ヒント: MPI プログラムの実行時間計測方法は，第 7 章を参考にせよ。]

4. それぞれの集団通信プログラムの実行時間を計り，比較検討せよ。