第9章　行列の積への応用

本章では正方行列同士の積を計算するプログラムを作成する。といっても，実際にはMPIBNCpackにはその機能があり，関数を呼び出すだけで計算可能である。ここではそこで使われているアルゴリズムを解説し，併せてそれが今まで習得したスキルを動員することで実現可能である，ということを理解して貰うことを目的とする。

9.1　BNCpackによる行列積

まず最初に，1CPUにて正方行列同士の積を計算するプログラムを示す。行列は先ほどと同様

$$X = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix}, B = \begin{bmatrix} 25 & 24 & 23 & 22 & 21 \\ 20 & 19 & 18 & 17 & 16 \\ 15 & 14 & 13 & 12 & 11 \\ 10 & 9 & 8 & 7 & 6 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix}$$

を用い，積 $C = XB$ を計算する。但し，行列 X がプログラム中の変数dmat1に，行列 B が変数dmat2に，積 C が変数dmat_ansにあたるので注意すること。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #include "bnc.h"
 6 :
 7 : #define DIM 5
 8 :
 9 : int main(int argc, char *argv[])
10 : {
11 :     DMatrix dmat_ans, dmat1, dmat2;
12 :     long int i, j;
13 :     double start_wtime, end_wtime;
```

```
14 :
15 :     dmat_ans = init_dmatrix(DIM, DIM);
16 :     dmat1 = init_dmatrix(DIM, DIM);
17 :     dmat2 = init_dmatrix(DIM, DIM);
18 :     for(i = 0; i < DIM; i++)
19 :     {
20 :         for(j = 0; j < DIM; j++)
21 :         {
22 :             set_dmatrix_ij(dmat1, i, j, (double)(i*DIM + j + 1));

23 :             set_dmatrix_ij(dmat2, i, j, (double)(DIM * DIM - (i*D
   IM +j)));
24 :         }
25 :     }
26 :
27 :     mul_dmatrix(dmat_ans, dmat1, dmat2);
28 :
29 :     printf("BNC:\n"); print_dmatrix(dmat_ans);
30 :
31 :     free_dmatrix(dmat_ans);
32 :     free_dmatrix(dmat1);
33 :     free_dmatrix(dmat2);
34 :
35 :     return EXIT_SUCCESS;
36 : }
```

これを多倍長化すると以下のようになる。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #define USE_GMP
 6 : #define USE_MPFR
 7 : #include "bnc.h"
 8 :
 9 : #define DIM 5
10 :
11 : int main(int argc, char *argv[])
12 : {
13 :     MPFMatrix dmat_ans, dmat1, dmat2;
14 :     long int i, j;
15 :     double start_wtime, end_wtime;
16 :
17 :     set_bnc_default_prec_decimal(50);
18 :
19 :     dmat_ans = init_mpfmatrix(DIM, DIM);
```

```
20 :     dmat1 = init_mpfmatrix(DIM, DIM);
21 :     dmat2 = init_mpfmatrix(DIM, DIM);
22 :     for(i = 0; i < DIM; i++)
23 :     {
24 :         for(j = 0; j < DIM; j++)
25 :         {
26 :             set_mpfmatrix_ij_d(dmat1, i, j, (double)(i*DIM + j +
   1));
27 :             set_mpfmatrix_ij_d(dmat2, i, j, (double)(DIM * DIM -
   (i*DIM +j)));
28 :         }
29 :     }
30 :
31 :     mul_mpfmatrix(dmat_ans, dmat1, dmat2);
32 :
33 :     printf("BNC:\n"); print_mpfmatrix(dmat_ans);
34 :
35 :     free_mpfmatrix(dmat_ans);
36 :     free_mpfmatrix(dmat1);
37 :     free_mpfmatrix(dmat2);
38 :
39 :     return EXIT_SUCCESS;
40 : }
41 :
```

9.2 行列積の並列分散アルゴリズム

各 PE へ正方行列を分割配置する方法は前章で解説した。その配置を利用して行列の積を並列分散化して実行するアルゴリズムを示す。ここでは行列 A, B の積 $C = AB$ を計算を行うこととする。

行列とベクトルの積とは異なり，行列同士の積は，ことに分散配置された行列 B の小行列を，PE 間を跨ぐ形で繰り返し利用する必要が出てくる。よって，積の計算は全て同じ PE に配置されている小行列同士でのみ行い，そこで必要となる行列 B の小行列はあらかじめ 1 対 1 通信を利用してその PE へ送り込んでおく。

使用する PE 数を 4 とした場合の行列積の実行の様子を図で示す (図 9.1〜9.3)。このアルゴリズムは Matrix Computations[9] に記述されているので詳細はそちらを参照されたい。

まず，図 9.1 のように，行列 A, B を MPIBNCpack 標準の方法で分割配置する。その後，行列 A の小行列も，2 行目の小行列を西方向へ 1 回シフト，3 行目を 2 回シフト，4 行目を 3 回シフトする。この場合は同じ PE に配置された小行列の順序をずらすだけなので，実際には添字のシフトのみ行えばよい。

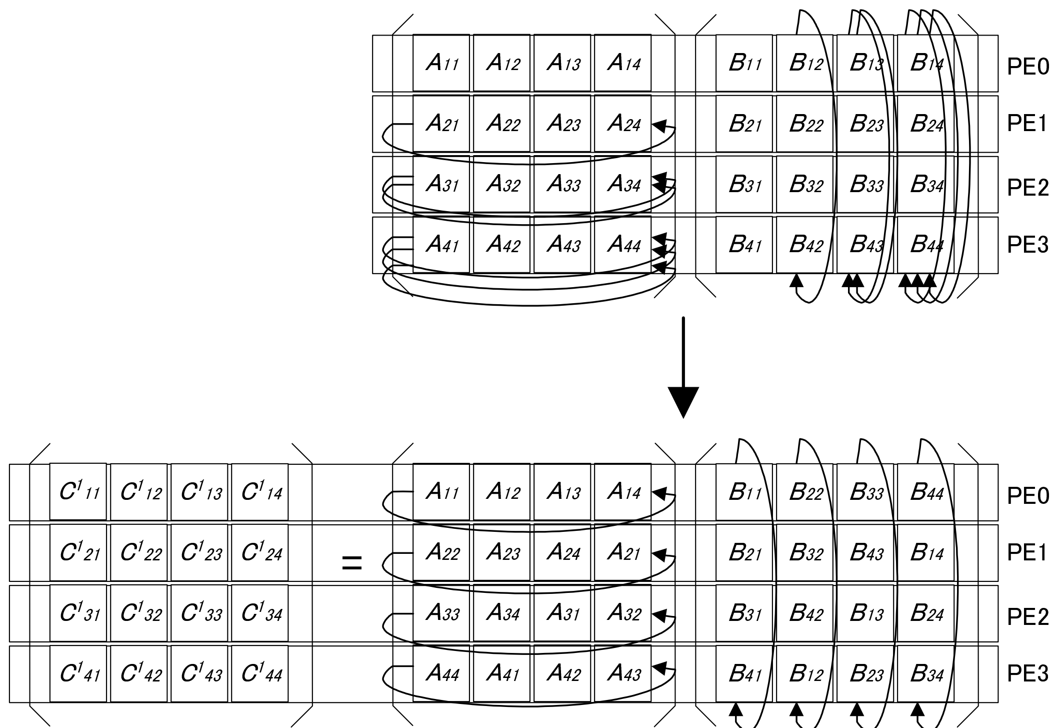

図 9.1: 行列積の計算 (1)

また，行列 B の 2 列目の小行列を北方向へ 1 回シフトする。3 列目は 2 回，4 列目は 3 回シフトしておく。この場合は PE 間を跨ぐ小行列の交換が必要になるための通信が発生する。

このように各行列のシフト作業が終了したら，同じ PE に配置された小行列の積を計算する。この時，添字がぴったり符合するようになる。この結果

$$
\begin{aligned}
C_{11}^{1} &= A_{11}B_{11},\ C_{12}^{1} = A_{11}B_{22},\ C_{13}^{1} = A_{13}B_{33},\ C_{14}^{1} = A_{14}B_{44} \\
C_{21}^{1} &= A_{22}B_{21},\ C_{22}^{1} = A_{23}B_{32},\ C_{23}^{1} = A_{24}B_{43},\ C_{24}^{1} = A_{21}B_{14} \\
C_{31}^{1} &= A_{33}B_{31},\ C_{32}^{1} = A_{34}B_{42},\ C_{33}^{1} = A_{31}B_{13},\ C_{34}^{1} = A_{32}B_{24} \\
C_{41}^{1} &= A_{44}B_{41},\ C_{42}^{1} = A_{41}B_{12},\ C_{43}^{1} = A_{42}B_{23},\ C_{44}^{1} = A_{43}B_{34}
\end{aligned}
$$

を得る。**通常の行列積の手順とは異なることに注意！** この結果得られる行列をまとめて C^1 とする。

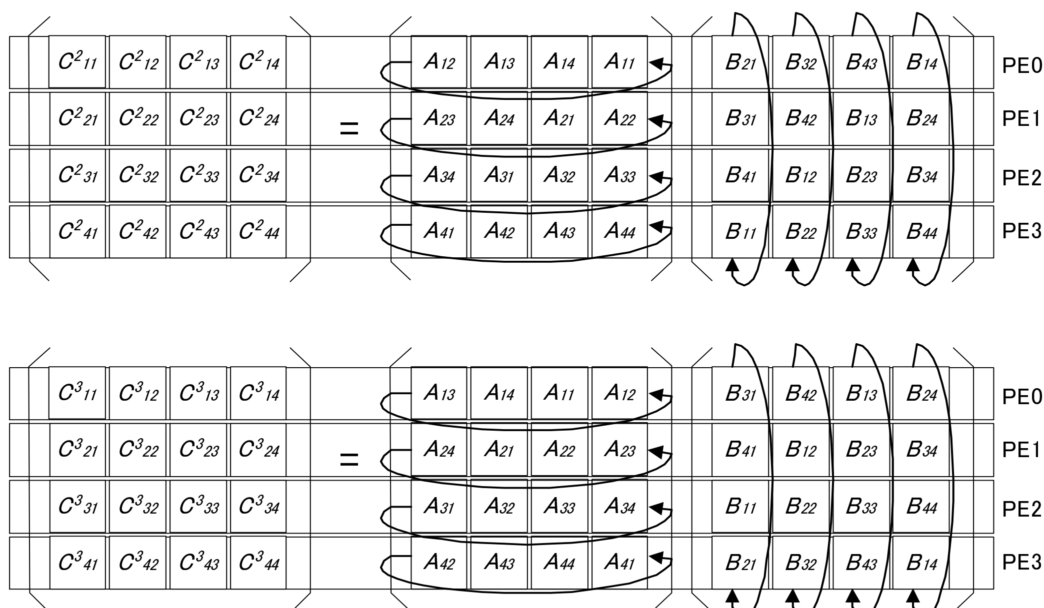

図 9.2: 行列積の計算 (2)

次に，行列 A は全ての行で西方向へ 1 回シフト，行列 B は全ての列で北方向に 1 回シフトを行い，先ほどと同様に，同じ PE において小行列の積を計算し，行列 C^2 を得る。更に同様にして行列 C^3，C^4 を得る。

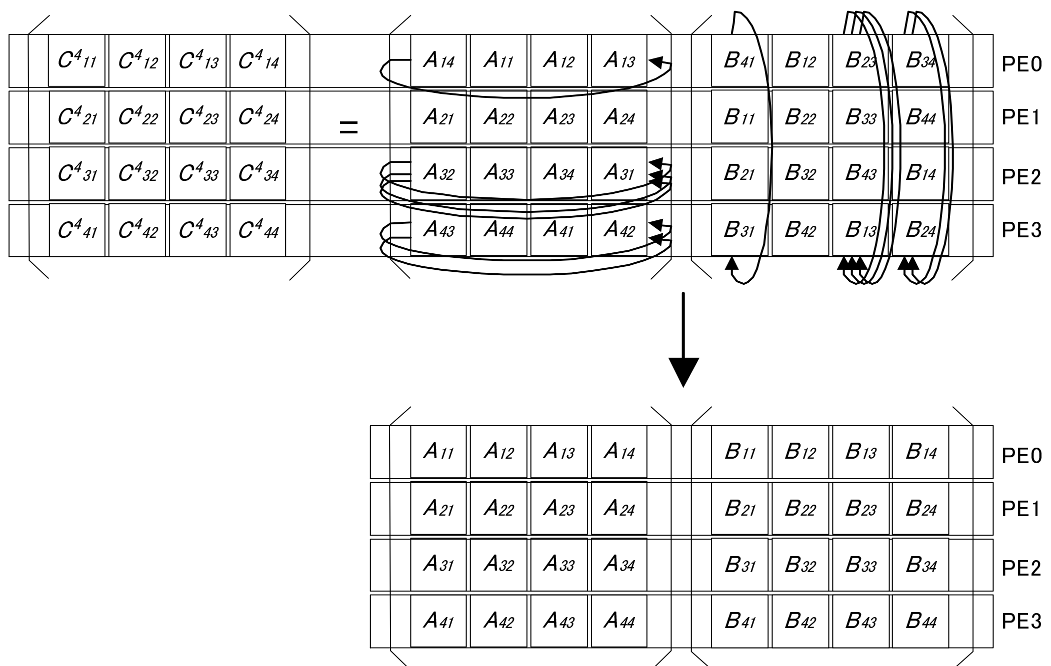

図 9.3: 行列積の計算 (3)

こうして得た各行列 C^1, C^2, C^3, C^4 を全て加えたものが行列 C となる。即ち

$$C = C^1 + C^2 + C^3 + C^4$$

となる。元の行列 A，B はそれぞれ適切な回数だけシフトを行ってもとの配置に戻しておくと良い。

MPIBNCpack に実装した _mpi_mul_dmatrix(倍精度用) 関数，及び _mpi_mul_mpfmatrix(多倍長用) 関数ではこのアルゴリズムに基づいて，正方行列の積を計算するようになっている。

9.3 MPIBNCpack による行列積と時間計測方法

正方行列の積を並列実行するプログラムは次のようになる。

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #include "mpi.h"
 6 :
 7 : #include "mpi_bnc.h"
 8 :
 9 : #define DIM 5
10 :
11 : int main(int argc, char *argv[])
12 : {
13 :     DMatrix my_dmat_ans[10], my_dmat1[10], my_dmat2[10];
14 :     DMatrix dmat_ans, dmat1, dmat2;
15 :
16 :     long int d_ddim[MPI_GMP_MAXPROCS];
17 :     long int i, j, local_dim;
18 :     int myrank, num_procs;
19 :     double start_wtime, end_wtime;
20 :
21 :     MPI_Init(&argc, &argv);
22 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
23 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
24 :
25 :     local_dim = _mpi_divide_dim(d_ddim, DIM, num_procs);
26 :
27 :     _mpi_init_dmatrix(my_dmat_ans, d_ddim, DIM, MPI_COMM_WORLD);
28 :     _mpi_init_dmatrix(my_dmat1, d_ddim, DIM, MPI_COMM_WORLD);
29 :     _mpi_init_dmatrix(my_dmat2, d_ddim, DIM, MPI_COMM_WORLD);
```

```
30 :
31 :     if(myrank == 0)
32 :     {
33 :         dmat_ans = init_dmatrix(local_dim * num_procs, local_dim
   * num_procs);
34 :         dmat1 = init_dmatrix(local_dim * num_procs, local_dim * n
   um_procs);
35 :         dmat2 = init_dmatrix(local_dim * num_procs, local_dim * n
   um_procs);
36 :         for(i = 0; i < DIM; i++)
37 :             for(j = 0; j < DIM; j++)
38 :             {
39 :                 set_dmatrix_ij(dmat1, i, j, (double)(i*DIM + j +
   1));
40 :                 set_dmatrix_ij(dmat2, i, j, (double)(DIM * DIM -
   (i*DIM +j)));
41 :             }
42 :     }
43 :
44 :     _mpi_divide_dmatrix(my_dmat1, d_ddim, dmat1, MPI_COMM_WORLD);

45 :     _mpi_divide_dmatrix(my_dmat2, d_ddim, dmat2, MPI_COMM_WORLD);

46 :
47 :     if(myrank == 0) start_wtime = MPI_Wtime();
48 :     _mpi_mul_dmatrix(my_dmat_ans, my_dmat1, my_dmat2, MPI_COMM_WO
   RLD);
49 :     if(myrank == 0) end_wtime = MPI_Wtime();
50 :
51 :     _mpi_collect_dmatrix(dmat_ans, d_ddim, my_dmat_ans, MPI_COMM_
   WORLD);
52 :
53 :     /* free */
54 :     _mpi_free_dmatrix(my_dmat_ans, MPI_COMM_WORLD);
55 :     _mpi_free_dmatrix(my_dmat1, MPI_COMM_WORLD);
56 :     _mpi_free_dmatrix(my_dmat2, MPI_COMM_WORLD);
57 :
58 :     if(myrank == 0)
59 :     {
60 :         printf("MPIBNC:\n"); print_dmatrix(dmat_ans);
61 :         printf("MPI_MUL_TIME: %f\n", end_wtime - start_wtime);
62 :
63 :         start_wtime = get_secv();
64 :         mul_dmatrix(dmat_ans, dmat1, dmat2);
65 :         end_wtime = get_secv();
66 :
```

```
67 :         printf("BNC:\n"); print_dmatrix(dmat_ans);
68 :         printf("BNC_MUL_TIME: %f\n", end_wtime - start_wtime);
69 :
70 :         free_dmatrix(dmat_ans);
71 :         free_dmatrix(dmat1);
72 :         free_dmatrix(dmat2);
73 :     }
74 :     MPI_Finalize();
75 :
76 :     return EXIT_SUCCESS;
77 : }
78 :
```

このプログラムは，行列積計算の時間を 2 箇所で計測している。一つは 53 行目〜55 行目であり，ここでは並列分散化した行列積の計算時間を，PE0 において MPI_Wtime 関数で計測している。もう一箇所は 69 行目〜71 行目で，1CPU における行列積の計算時間を BNCpack に備えられている get_secv 関数を使って計測している。どちらも呼び出した際の時間が倍精度実数で表現された秒 (second) で得られるため，その差が計算に要した時間ということになる。

多倍長化したものは以下の通り。前と同様に時間計測を 2 箇所で行っている。

```
 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 : #define DIM 5
12 :
13 : int main(int argc, char *argv[])
14 : {
15 :     MPFMatrix my_mpfmat_ans[10], my_mpfmat1[10], my_mpfmat2[10];
16 :     MPFMatrix mpfmat_ans, mpfmat1, mpfmat2;
17 :     mpf_t tmp;
18 :
19 :     long int d_ddim[10];
20 :     long int i, j, local_dim;
21 :     int myrank, num_procs;
22 :     double start_wtime, end_wtime;
23 :
24 :     MPI_Init(&argc, &argv);
```

```
25 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
26 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
27 :
28 :     _mpi_set_bnc_default_prec_decimal(50, MPI_COMM_WORLD);
29 :     commit_mpf(&(MPI_MPF), ceil(50/log10(2.0)), MPI_COMM_WORLD);
30 :     create_mpf_op(&(MPI_MPF_SUM), _mpi_mpf_add, MPI_COMM_WORLD);
31 :
32 :     local_dim = _mpi_divide_dim(d_ddim, DIM, num_procs);
33 :
34 :     _mpi_init_mpfmatrix(my_mpfmat_ans, d_ddim, DIM, MPI_COMM_WORL
   D);
35 :     _mpi_init_mpfmatrix(my_mpfmat1, d_ddim, DIM, MPI_COMM_WORLD);

36 :     _mpi_init_mpfmatrix(my_mpfmat2, d_ddim, DIM, MPI_COMM_WORLD);

37 :
38 :     if(myrank == 0)
39 :     {
40 :         mpfmat_ans = init_mpfmatrix(local_dim * num_procs, local_
   dim * num_procs);
41 :         mpfmat1 = init_mpfmatrix(local_dim * num_procs, local_dim
    * num_procs);
42 :         mpfmat2 = init_mpfmatrix(local_dim * num_procs, local_dim
    * num_procs);
43 :         for(i = 0; i < DIM; i++)
44 :             for(j = 0; j < DIM; j++)
45 :             {
46 :                 set_mpfmatrix_ij_ui(mpfmat1, i, j, (unsigned long
   )(i*DIM + j + 1));
47 :                 set_mpfmatrix_ij_ui(mpfmat2, i, j, (unsigned long
   )(DIM * DIM - (i*DIM + j)));
48 :             }
49 :     }
50 :
51 :     _mpi_divide_mpfmatrix(my_mpfmat1, d_ddim, mpfmat1, MPI_COMM_W
   ORLD);
52 :     _mpi_divide_mpfmatrix(my_mpfmat2, d_ddim, mpfmat2, MPI_COMM_W
   ORLD);
53 :
54 :     if(myrank == 0) start_wtime = MPI_Wtime();
55 :     _mpi_mul_mpfmatrix(my_mpfmat_ans, my_mpfmat1, my_mpfmat2, MPI
   _COMM_WORLD);
56 :     if(myrank == 0) end_wtime = MPI_Wtime();
57 :
58 :     _mpi_collect_mpfmatrix(mpfmat_ans, d_ddim, my_mpfmat_ans, MPI
   _COMM_WORLD);
```

```
59 :
60 :     /* free */
61 :     _mpi_free_mpfmatrix(my_mpfmat_ans, MPI_COMM_WORLD);
62 :     _mpi_free_mpfmatrix(my_mpfmat1, MPI_COMM_WORLD);
63 :     _mpi_free_mpfmatrix(my_mpfmat2, MPI_COMM_WORLD);
64 :
65 :     if(myrank == 0)
66 :     {
67 :         printf("MPIBNC:\n"); print_mpfmatrix(mpfmat_ans);
68 :         printf("MPI_MUL_TIME: %f\n", end_wtime - start_wtime);
69 :
70 :         start_wtime = get_secv();
71 :         mul_mpfmatrix(mpfmat_ans, mpfmat1, mpfmat2);
72 :         end_wtime = get_secv();
73 :
74 :         printf("BNC:\n"); print_mpfmatrix(mpfmat_ans);
75 :         printf("BNC_MUL_TIME: %f\n", end_wtime - start_wtime);
76 :
77 :         free_mpfmatrix(mpfmat_ans);
78 :         free_mpfmatrix(mpfmat1);
79 :         free_mpfmatrix(mpfmat2);
80 :     }
81 :
82 :     free_mpf_op(&(MPI_MPF_SUM));
83 :     free_mpf(&(MPI_MPF));
84 :     MPI_Finalize();
85 :
86 :     return EXIT_SUCCESS;
87 : }
88 :
```

結果については省略する。自ら実行して確認されたい。

演習問題

1. mpi-mat5.c を用いて，ベンチマークテストを実行せよ。その際には表計算ソフトウェアで次のような表を作っておくと便利である。

次元数	mat5.c	1PE	2PEs	4PEs	8PEs
128					
256					
512					
1024					

2. mpi-mat5-gmp.c を用いて，ベンチマークテストを実行せよ。その際には上記の表を参考にして，更に精度を 50 桁, 100 桁, 200 桁とした場合の計算時間もそれぞれ計測せよ。