第10章　Krylov部分空間法への応用

基本線型計算の応用例として，Krylov 部分空間法 (Conjugate-Gradient 法 (CG 法), 積型 Krylov 部分空間法) を使って連立一次方程式を解いてみる。これは内積計算, ベクトルのスカラー倍，行列・ベクトル乗算のみを使ったアルゴリズムになっており，古くから並列化しやすい例として利用されてきた。また，丸め誤差の影響を受けやすく，多倍長計算の効果が発揮しやすいという特徴もあり，MPIBNCpack にはもってこいの題材である。

10.1　CG 法

共役勾配法 (Conjugate-Gradient，CG 法) は係数行列 $A \in \mathbb{R}^{n\times n}$ が正定値対称，即ち

$$A^T = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \vdots & \vdots & & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{nn} \end{bmatrix} = A$$

であるような n 次元の連立一次方程式

$$A\mathbf{x} = \mathbf{b} \ \ (\text{ここで } A \in \mathbb{R}^{n\times n},\ \mathbf{x}, \mathbf{b} \in \mathbb{R}^n) \tag{10.1}$$

を解くためのアルゴリズムである。勾配ベクトル $\mathbf{p}_k \in \mathbb{R}^n$ が，線形空間 $\mathbb{R}^n$ 内にある Krylov 部分空間 ($\mathbf{r}_0, A\mathbf{r}_0, ..., A^k\mathbf{r}_0$ が張る線形部分空間) に属するため，Krylov 部分空間法と呼ばれるカテゴリに属している。

この CG 法のアルゴリズムは以下の通りである。

1. 初期値 $\mathbf{x}_0 \in \mathbb{R}^n$ を決める。
2. $\mathbf{r}_0 := \mathbf{b} - A\mathbf{x}_0$，$\mathbf{p}_0 := \mathbf{r}_0$ とする。
3. $k = 0, 1, 2, \cdots$ に対して以下を計算する。
   (a) $\alpha_k := \dfrac{(\mathbf{r}_k, \mathbf{p}_k)}{(\mathbf{p}_k, A\mathbf{p}_k)}$

(b) $\mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$

(c) $\mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k A\mathbf{p}_k$(又は $:= \mathbf{b} - A\mathbf{x}_{k+1}$)

(d) $\beta_k := \frac{\|\mathbf{r}_{k+1}\|_2^2}{\|\mathbf{r}_k\|_2^2}$

(e) 収束判定: $\|\mathbf{r}_k\|_2 / \|\mathbf{r}_0\|_2 < \varepsilon_R$

(f) $\mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$

見ての通り，行列・ベクトル積 $A\mathbf{p}_k$ と内積およびベクトルの和とスカラー倍の計算のみから成立しているアルゴリズムであるため，並列化がきわめて容易である。

理論的には勾配ベクトル $\mathbf{p}_0, \mathbf{p}_1, ..., \mathbf{p}_k, ...$ が直交しているため，有限回 $\leq n$ で収束することが保証されている。しかし，実際には有限桁の FP 数を用いる限り，完全な直交性は維持できず，定常反復法 (Jacobi 反復法, Gauss-Seidel 法，SOR 法) と同じように収束判定をする必要がある。

10.2　逐次計算プログラム

使用する連立一次方程式 (10.1) の係数行列 (Frank 行列)A と解 $\mathbf{x}$ を

$$A = \begin{bmatrix} n & n-1 & \cdots & 1 \\ n-1 & n-1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ n-1 \end{bmatrix}$$

とする。次元数は 512 次元 ($n = 512$) とするが，以下に示すプログラムではどの次元数でも対応ができるようになっていることに注意せよ。

これを倍精度計算の CG 法で解くプログラムは以下のようになる。

cg.c

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #include "bnc.h"
 6 :
 7 : #define DIM 512
 8 :
 9 : void get_dproblem(DMatrix a, DVector b, DVector ans, long dim)
10 : {
```

```
11 :     long int i, j, k;
12 :     double tmp;
13 :
14 :     /* Frank Matrix */
15 :     for(i = 0; i < dim; i++)
16 :     {
17 :         for(j = 0; j < dim; j++)
18 :         {
19 :             if(i < j)
20 :                 set_dmatrix_ij(a, i, j, (double)(dim - j));
21 :             else
22 :                 set_dmatrix_ij(a, i, j, (double)(dim - i));
23 :         }
24 :     }
25 :
26 :     /* Answer */
27 :     for(i = 0; i < dim; i++)
28 :         set_dvector_i(ans, i, (double)i);
29 :
30 :     /* Make constant vector */
31 :     mul_dmatrix_dvec(b, a, ans);
32 : }
33 :
34 : int main(int argc, char *argv[])
35 : {
36 :     DMatrix da;
37 :     DVector db, dx, dans;
38 :     double start, dtime;
39 :
40 :     long int itimes_d;
41 :
42 :     /* initialize */
43 :     da = init_dmatrix(DIM, DIM);
44 :     db = init_dvector(DIM);
45 :     dx = init_dvector(DIM);
46 :     dans = init_dvector(DIM);
47 :
48 :     /* get problem */
49 :     get_dproblem(da, db, dans, DIM);
50 :
51 :     /* run DCG */
52 :     start = get_secv();
53 :     itimes_d = DCG(dx, da, db, 1.0e-13, 1.0e-99, DIM * 5);
54 :     dtime = get_secv() - start;
55 :
56 :     print_dvector(dx);
```

```
 57 :
 58 :     /* end */
 59 :     free_dmatrix(da);
 60 :     free_dvector(db);
 61 :     free_dvector(dx);
 62 :     free_dvector(dans);
 63 :
 64 :     /* print itimes */
 65 :     printf("double         : %ld(%f)\n", itimes_d, dtime);
 66 : }
 67 :
```

多倍長計算の CG 法で解くプログラムは以下のようになる。

cg-gmp.c

```
  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 512
 10 :
 11 : void get_mpfproblem(MPFMatrix a, MPFVector b, MPFVector ans, long
      dim)
 12 : {
 13 :     long int i, j, k;
 14 :     mpf_t tmp, sqr2;
 15 :
 16 :     mpf_init2(tmp, prec_mpfvector(ans));
 17 :     mpf_init2(sqr2, prec_mpfvector(ans));
 18 :
 19 :     mpf_set_ui(sqr2, 2UL); mpf_sqrt(sqr2, sqr2);
 20 :
 21 :     /* Frank Matrix */
 22 :     for(i = 0; i < dim; i++)
 23 :     {
 24 :         for(j = 0; j < dim; j++)
 25 :         {
 26 :             if(i < j)
 27 :             {
 28 :                 mpf_set_si(tmp, dim - j);
 29 :                 set_mpfmatrix_ij(a, i, j, tmp);
```

```
30 :             }
31 :             else
32 :             {
33 :                 mpf_set_si(tmp, dim - i);
34 :                 set_mpfmatrix_ij(a, i, j, tmp);
35 :             }
36 :             mpf_mul(tmp, tmp, sqr2);
37 :             set_mpfmatrix_ij(a, i, j, tmp);
38 :         }
39 :     }
40 :
41 :     /* Answer */
42 :     for(i = 0; i < dim; i++)
43 :     {
44 :         mpf_set_si(tmp, i);
45 :         set_mpfvector_i(ans, i, tmp);
46 :     }
47 :
48 :     /* Make constant vector */
49 :     mul_mpfmatrix_mpfvec(b, a, ans);
50 :
51 :     mpf_clear(tmp);
52 :     mpf_clear(sqr2);
53 : }
54 :
55 : int main(int argc, char *argv[])
56 : {
57 :     double start, dtime, startwtime[2], endwtime[2];
58 :
59 :     MPFMatrix mpfa;
60 :     MPFVector mpfb, mpfx, mpfans;
61 :     mpf_t reps, aeps;
62 :     long int itimes_mpf;
63 :     double mpftime;
64 :
65 : #define MPF_PREC 128
66 :
67 :     /* initialize */
68 :     mpf_init(reps);
69 :     mpf_init(aeps);
70 :
71 :     mpfa  = init_mpfmatrix(DIM, DIM);
72 :     mpfb  = init_mpfvector(DIM);
73 :     mpfx  = init_mpfvector(DIM);
74 :     mpfans  = init_mpfvector(DIM);
75 :
```

```
 76 :     /* get problem */
 77 :     get_mpfproblem(mpfa, mpfb, mpfans, DIM);
 78 :
 79 :     /* run MPFFCG */
 80 :     mpf_set_d(reps, 1.0e-20);
 81 :     mpf_set_d(aeps, 1.0e-50);
 82 :
 83 :     start = get_secv();
 84 :     itimes_mpf = MPFCG(mpfx, mpfa, mpfb, reps, aeps, DIM * 5);
 85 :     mpftime = get_secv() - start;
 86 :
 87 :     print_mpfvector(mpfx);
 88 :
 89 :     free_mpfmatrix(mpfa);
 90 :     free_mpfvector(mpfb);
 91 :     free_mpfvector(mpfx);
 92 :     free_mpfvector(mpfans);
 93 :
 94 :     /* end */
 95 :     mpf_clear(reps); mpf_clear(aeps);
 96 :
 97 :     /* print itimes */
 98 :     printf("mpf_t(%d)  : %ld(%f)\n", MPF_PREC, itimes_mpf, mpftim
    e);
 99 :
100 :     return EXIT_SUCCESS;
101 : }
102 :
```

10.3　並列プログラム

倍精度 CG 法のプログラム (cg.c) を並列化したプログラムは以下のようになる。

mpi-cg.c

```
  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 512
```

```
10 :
11 :void get_dproblem(DMatrix a, DVector b, DVector ans, long dim)
12 :{
13 :    long int i, j, k;
14 :    double tmp;
15 :
16 :    /* Frank Matrix */
17 :    for(i = 0; i < dim; i++)
18 :    {
19 :        for(j = 0; j < dim; j++)
20 :        {
21 :            if(i < j)
22 :                set_dmatrix_ij(a, i, j, (double)(dim - j));
23 :            else
24 :                set_dmatrix_ij(a, i, j, (double)(dim - i));
25 :        }
26 :    }
27 :
28 :    /* Answer */
29 :    for(i = 0; i < dim; i++)
30 :        set_dvector_i(ans, i, (double)i);
31 :
32 :    /* Make constant vector */
33 :    mul_dmatrix_dvec(b, a, ans);
34 :}
35 :
36 :int main(int argc, char *argv[])
37 :{
38 :    int myid, numprocs;
39 :    int  namelen;
40 :    char processor_name[MPI_MAX_PROCESSOR_NAME];
41 :
42 :    long int d_ddim[MPI_GMP_MAXPROCS], local_dim;
43 :    DMatrix da, my_da[MPI_GMP_MAXPROCS];
44 :    DVector db, dx, dans, my_db, my_dx, my_dans;
45 :    double start, ftime, dtime, startwtime[2], endwtime[2];
46 :
47 :    long int itimes_f, itimes_d, itimes_dm;
48 :    long int i, j;
49 :
50 :    MPI_Init(&argc, &argv);
51 :    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
52 :    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
53 :    MPI_Get_processor_name(processor_name,&namelen);
54 :
55 :    fprintf(stdout,"Process %d of %d on %s\n",
```

```
 56 :         myid, numprocs, processor_name);
 57 :
 58 :     /* divide problem */
 59 :     local_dim = _mpi_divide_dim(d_ddim, DIM, numprocs);
 60 :     if(myid == 0)
 61 :     {
 62 :         /* initialize */
 63 :         da = init_dmatrix(DIM, DIM);
 64 :         db = init_dvector(DIM);
 65 :         dx = init_dvector(DIM);
 66 :         dans = init_dvector(DIM);
 67 :
 68 :         /* get problem */
 69 :         get_dproblem(da, db, dans, DIM);
 70 :
 71 ://         print_dmatrix(da);
 72 :     }
 73 :
 74 :
 75 :     my_db = _mpi_init_dvector(d_ddim, DIM, MPI_COMM_WORLD);
 76 :     my_dx = _mpi_init_dvector(d_ddim, DIM, MPI_COMM_WORLD);
 77 :     _mpi_init_dmatrix(my_da, d_ddim, DIM, MPI_COMM_WORLD);
 78 :
 79 :     _mpi_divide_dvector(my_db, d_ddim, db, MPI_COMM_WORLD);
 80 :     _mpi_divide_dmatrix(my_da, d_ddim, da, MPI_COMM_WORLD);
 81 :
 82 :     if(myid == 0) startwtime[0] = MPI_Wtime();
 83 :     itimes_dm = _mpi_DCG(my_dx, my_da, my_db, 1.0e-13, 1.0e-99, D
    IM * 5, DIM, MPI_COMM_WORLD);
 84 :     if(myid == 0) endwtime[0] = MPI_Wtime() - startwtime[0];
 85 ://    for(i = 0; i < local_dim; i++)
 86 ://        printf("%5ld %25.17e\n", i, get_dvector_i(my_dx, i));
 87 :     _mpi_collect_dvector(dx, d_ddim, my_dx, MPI_COMM_WORLD);
 88 :     if(myid == 0) print_dvector(dx);
 89 :
 90 :     if(myid == 0)
 91 :     {
 92 :
 93 :         /* run DCG */
 94 :         start = get_secv();
 95 :         itimes_d = DCG(dx, da, db, 1.0e-13, 1.0e-99, DIM * 5);
 96 :         dtime = get_secv() - start;
 97 :
 98 :         /* print */
 99 :         for(i = 0; i < DIM; i++)
100 :             printf("%5ld %25.17e %25.17e\n", i, get_dvector_i(dx,
```

```
     i), get_dvector_i(dans, i));
101 :
102 :         /* end */
103 :         free_dmatrix(da);
104 :         free_dvector(db);
105 :         free_dvector(dx);
106 :         free_dvector(dans);
107 :     }
108 :
109 :     MPI_Finalize();
110 :
111 :     if(myid == 0){
112 :     /* print itimes */
113 :     printf("Iterative Times\n");
114 :     printf("double(MPI)    : %ld(%f)\n", itimes_dm, endwtime[0]);

115 :     printf("double         : %ld(%f)\n", itimes_d, dtime);
116 :     }
117 :}
118 :
```

多倍長 CG 法のプログラム (cg-gmp.c) を並列化したプログラムは以下のようになる。

mpi-cg-gmp.c

```
  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 512
 12 :
 13 :void get_mpfproblem(MPFMatrix a, MPFVector b, MPFVector ans, long
     dim)
 14 :{
 15 :     long int i, j, k;
 16 :     mpf_t tmp, sqr2;
 17 :
 18 :     mpf_init2(tmp, prec_mpfvector(ans));
 19 :     mpf_init2(sqr2, prec_mpfvector(ans));
```

```
20 :
21 :     mpf_set_ui(sqr2, 2UL); mpf_sqrt(sqr2, sqr2);
22 :
23 :     /* Frank Matrix */
24 :     for(i = 0; i < dim; i++)
25 :     {
26 :         for(j = 0; j < dim; j++)
27 :         {
28 :             if(i < j)
29 :             {
30 :                 mpf_set_si(tmp, dim - j);
31 :                 set_mpfmatrix_ij(a, i, j, tmp);
32 :             }
33 :             else
34 :             {
35 :                 mpf_set_si(tmp, dim - i);
36 :                 set_mpfmatrix_ij(a, i, j, tmp);
37 :             }
38 :             mpf_mul(tmp, tmp, sqr2);
39 :             set_mpfmatrix_ij(a, i, j, tmp);
40 :         }
41 :     }
42 :
43 :     /* Answer */
44 :     for(i = 0; i < dim; i++)
45 :     {
46 :         mpf_set_si(tmp, i);
47 :         set_mpfvector_i(ans, i, tmp);
48 :     }
49 :
50 :     /* Make constant vector */
51 :     mul_mpfmatrix_mpfvec(b, a, ans);
52 :
53 :     mpf_clear(tmp);
54 :     mpf_clear(sqr2);
55 : }
56 :
57 : int main(int argc, char *argv[])
58 : {
59 :     int myid, numprocs;
60 :     int  namelen;
61 :     char processor_name[MPI_MAX_PROCESSOR_NAME];
62 :
63 :     long int d_ddim[MPI_GMP_MAXPROCS], local_dim;
64 :     double start, ftime, dtime, startwtime[2], endwtime[2];
65 :
```

```
 66 :     MPFMatrix mpfa, my_mpfa[MPI_GMP_MAXPROCS];
 67 :     MPFVector mpfb, mpfx, mpfans;
 68 :     MPFVector my_mpfb, my_mpfx, my_mpfans;
 69 :     mpf_t reps, aeps;
 70 :     long int itimes_mpf, itimes_mpfm;
 71 :     double mpftime[3];
 72 :
 73 :     long int itimes_f, itimes_d, itimes_dm;
 74 :     long int i, j;
 75 :
 76 :     MPI_Init(&argc, &argv);
 77 :     MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
 78 :     MPI_Comm_rank(MPI_COMM_WORLD,&myid);
 79 :     MPI_Get_processor_name(processor_name,&namelen);
 80 :
 81 :     fprintf(stdout,"Process %d of %d on %s\n",
 82 :         myid, numprocs, processor_name);
 83 :
 84 : #define MPF_PREC 128
 85 :
 86 :     _mpi_set_bnc_default_prec(MPF_PREC, MPI_COMM_WORLD);
 87 :     commit_mpf(&(MPI_MPF), MPF_PREC, MPI_COMM_WORLD);
 88 :     create_mpf_op(&(MPI_MPF_SUM), _mpi_mpf_add, MPI_COMM_WORLD);
 89 :
 90 :     /* initialize */
 91 :     mpf_init(reps);
 92 :     mpf_init(aeps);
 93 :
 94 :     /* divide problem */
 95 :     local_dim = _mpi_divide_dim(d_ddim, DIM, numprocs);
 96 :     if(myid == 0)
 97 :     {
 98 :         mpfa  = init_mpfmatrix(DIM, DIM);
 99 :         mpfb  = init_mpfvector(DIM);
100 :         mpfx  = init_mpfvector(DIM);
101 :         mpfans  = init_mpfvector(DIM);
102 :
103 :         /* get problem */
104 :         get_mpfproblem(mpfa, mpfb, mpfans, DIM);
105 :
106 :         // print_mpfmatrix(mpfa);
107 :     }
108 :
109 :     /* run MPFFCG */
110 :     mpf_set_d(reps, 1.0e-20);
111 :     mpf_set_d(aeps, 1.0e-50);
```

```
112 :
113 :     my_mpfb = _mpi_init_mpfvector(d_ddim, DIM, MPI_COMM_WORLD);
114 :     my_mpfx = _mpi_init_mpfvector(d_ddim, DIM, MPI_COMM_WORLD);
115 :     _mpi_init_mpfmatrix(my_mpfa, d_ddim, DIM, MPI_COMM_WORLD);
116 :
117 :     _mpi_divide_mpfvector(my_mpfb, d_ddim, mpfb, MPI_COMM_WORLD);

118 :     _mpi_divide_mpfmatrix(my_mpfa, d_ddim, mpfa, MPI_COMM_WORLD);

119 :
120 :     if(myid == 0) startwtime[1] = MPI_Wtime();
121 :     itimes_mpfm = _mpi_MPFCG(my_mpfx, my_mpfa, my_mpfb, reps, aep
    s, DIM * 5, DIM, MPI_COMM_WORLD);
122 :     if(myid == 0) endwtime[1] = MPI_Wtime() - startwtime[1];
123 :
124 :/*    for(i = 0; i < local_dim; i++)
125 :     {
126 :         printf("%5ld ", i);
127 :         mpf_out_str(stdout, 10, 0, get_mpfvector_i(my_mpfx, i));
128 :         printf("\n");
129 :     }
130 :*/
131 :     _mpi_collect_mpfvector(mpfx, d_ddim, my_mpfx, MPI_COMM_WORLD)
    ;
132 ://    if(myid == 0) print_mpfvector(mpfx);
133 :     if(myid == 0)
134 :     {
135 :         i = 0; printf("%5d, ", i); mpf_out_str(stdout, 10, 0, gmp
    fvi(mpfx, i)); printf("\n");
136 :         i = DIM/2 - 1; printf("%5d, ", i); mpf_out_str(stdout, 10
    , 0, gmpfvi(mpfx, i)); printf("\n");
137 :         i = DIM - 1; printf("%5d, ", i); mpf_out_str(stdout, 10,
    0, gmpfvi(mpfx, i)); printf("\n");
138 :     }
139 :
140 :     if(myid == 0)
141 :     {
142 :         free_mpfmatrix(mpfa);
143 :         free_mpfvector(mpfb);
144 :         free_mpfvector(mpfx);
145 :         free_mpfvector(mpfans);
146 :     }
147 :
148 :     /* end */
149 :     mpf_clear(reps); mpf_clear(aeps);
150 :
```

```
151 :     free_mpf(&(MPI_MPF));
152 :     free_mpf_op(&(MPI_MPF_SUM));
153 :
154 : end:
155 :     MPI_Finalize();
156 :
157 :     if(myid == 0){
158 :         /* print itimes */
159 :         printf("Iterative Times\n");
160 :         printf("mpf_t(MPI, %d)  : %ld(%f)\n", MPF_PREC, itimes_mp
    fm, endwtime[1]);
161 :         printf("1 iter(millisec): %f milli-sec\n", 1000 * endwtim
    e[1] / (double)itimes_mpfm);
162 :     }
163 :
164 :     return EXIT_SUCCESS;
165 : }
166 :
```

10.4 最小計算時間の探索

更に，CG 法に限らず Krylov 部分空間法は丸め誤差の影響を受けやすいという性質が知られている。行列 A の固有値分布や初期値 $\mathbf{x}_0$(初期残差 $\mathbf{r}_0$) によってその挙動は一律ではないが，おおむね

$$\text{FP 数の桁数を増加させる} \Longrightarrow \text{反復回数が減る}$$

という傾向が見られる。多倍長 FP 数を用いることで反復回数をある程度まで減らすことができれば，CG 法が収束するまでに要する時間も減らすことが期待される。しかし，桁数を増やすと第 2 章で述べたように，一演算あたりの計算時間が増えるため，反復一回あたりの計算時間は必ず増加することになる。すると，どこかで最小の計算時間を得られる精度，というものが存在すると考えられる。つまり,「多倍長 FP 数演算を用いた CG 法の最小計算時間を求める」という問題は，一種の最適化問題として考えることができるのである (図 10.1)。

では，CG 法を並列化した場合の最小計算時間はどのようになるのだろうか？全体としては 1PE 計算時の時より短時間で実行できることは明らかであるが，通信時間を考えると PE 数だけ増やしてもどこかで打ち止めとなる。理論的には次元数 n と PE 数 p が等しい時が最小の計算時間となるが，通信時間の増加がこれを押し上げてしまう。

本節ではそれを実際に実行して確認してみることにする。

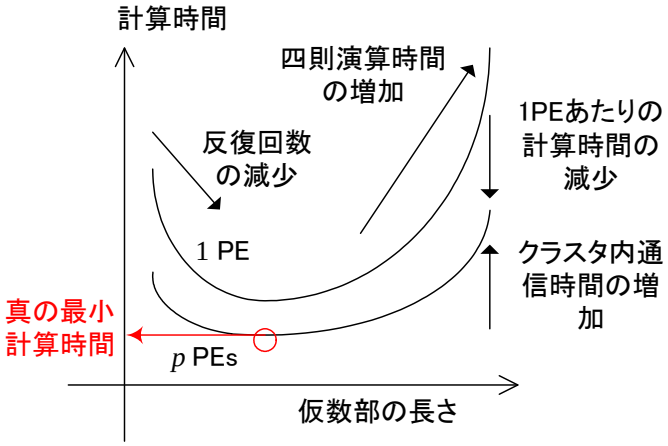


図 10.1: CG 法の最小計算時間の概念図

但しこれだと計算時間が短くなる (整数×多倍長 FP 数) ので，本節では

$$\sqrt{2} \cdot A$$

を係数行列として使用する。これによって連立一次方程式の数値的性質は変化することはない。次元数は $n = 512$ に固定して考える。

この Frank 行列を係数行列として持つ連立一次方程式 (10.1) を多倍長 CG 法で解くと，図 10.2 のような収束状況を示す。

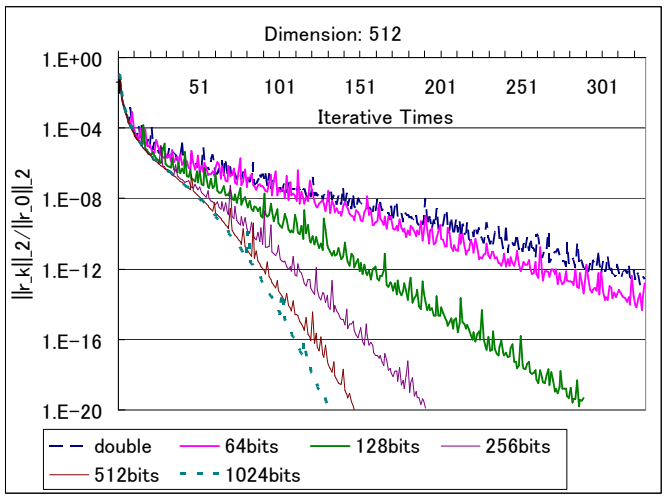
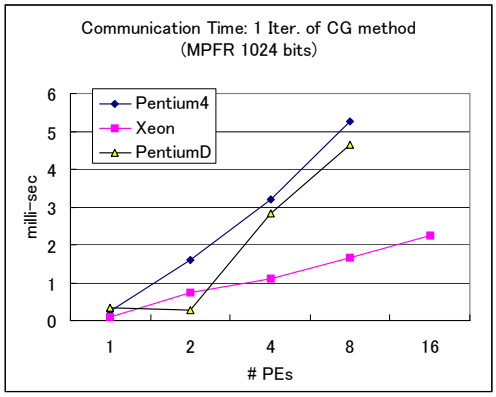


図 10.2: Frank 行列問題の収束状況

精度を増やすと残差ベクトルが次第に単調減少していくことがわかる。しかし，通信時間は図 10.3 のように増加する。



$n = 512$	行列	ベクトル	ベクトル/8
double	2 MB	4 KB	0.5 KB
64bits	7 (MB)	14 (KB)	0.33 (KB)
128bits	9	18	2.25
256bits	13	26	3.25
512bits	21	42	5.25
1024bits	37	74	9.25

図 10.3: CG 法の 1024bit 計算時の通信時間 (左) と記憶容量 (右)

このうち，PentiumD の通信時間が Xeon に比べて劣っているのは，転送されるベクトルの成分データ量の場合，Throughput が Xeon の半分程度になっているからである (図 10.4)。

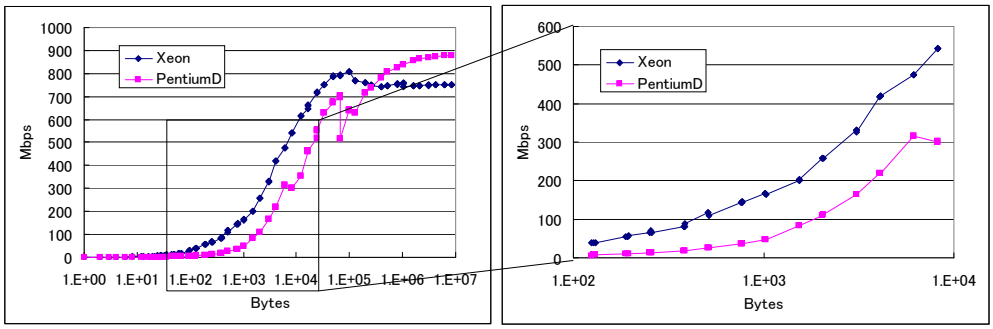


図 10.4: NetPIPE による MPI 通信の比較

この結果，1PE で計算した時の計算時間 (表 10.1) が，図 10.5 のように減少し，全て 128bit 計算時に最小計算時間を得ていることが分かる。

演習問題

積型 Krylov 部分空間法と呼ばれるアルゴリズムは，一般の連立一次方程式 (10.1) を解くためのものである。以下に示す BiCG, CGS, BiCGSTAB, GPBiCG 法を用い

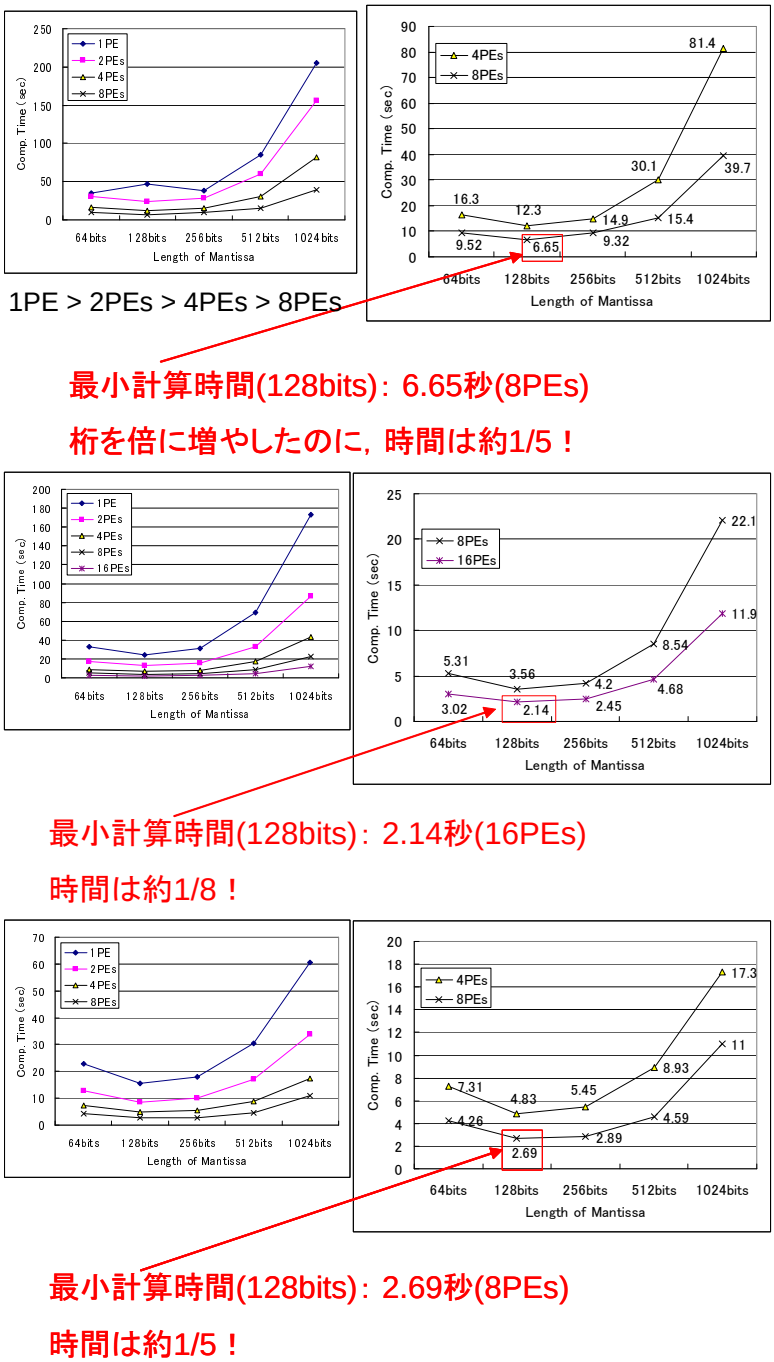


図 10.5: 最小計算時間: Pentium4(上), Xeon(中), PentiumD

表 10.1: 1PE で計算した時の反復回数と計算時間 (sec)

#bits	#Iteration	Pentium4	Xeon	PentiumD
64	506〜513	34.4	16.9	22.9
128	286〜294	47.3	24.2	15.6
256	191	38.4	31.2	17.9
512	147	85.3	69.3	30.4
1024	131	205	173	60.6

て，次の行列と解 $\mathbf{x}=[0\ 1\ \cdots\ n-1]^T$ を持つ連立一次方程式を解き，最小計算時間を求めよ。

$$A=\begin{bmatrix} n & n-1 & & & & \\ n-1 & n-1 & n-2 & & & \\ n-2 & n-2 & n-2 & n-3 & & \\ \vdots & \vdots & \vdots & \ddots & \ddots & \\ 2 & 2 & 2 & \cdots & 2 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix}$$

BiCG 法

$\mathbf{x}_0$: 初期値

$\mathbf{r}_0$: 初期残差 ($\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$)

$\widetilde{\mathbf{r}_0}$: $(\mathbf{r}_0, \widetilde{\mathbf{r}_0}) \neq 0$ を満たす任意のベクトル．例えば $\widetilde{\mathbf{r}_0} = \mathbf{r}_0$.

K: 前処理行列 (前処理なしの場合は $K = I$)

for $i = 1, 2, ...$

$K\mathbf{w}_{i-1} = \mathbf{r}_{i-1}$ を解いて $\mathbf{w}_{i-1}$ を求める.

$K^T\ \widetilde{\mathbf{w}_{i-1}} = \widetilde{\mathbf{r}_{i-1}}$ を解いて $\widetilde{\mathbf{w}_{i-1}}$ を求める.

$\rho_{i-1} = (\widetilde{\mathbf{w}_{i-1}}, \mathbf{w}_{i-1})$

if $\rho_{i-1} = 0$ **then** 終了.

if $i = 1$ **then**

$\mathbf{p}_1 = \mathbf{w}_0$

$\widetilde{\mathbf{p}_1} = \widetilde{\mathbf{w}_0}$

else

$\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$

$\mathbf{p}_i = \mathbf{w}_{i-1} + \beta_{i-1}\mathbf{p}_{i-1}$

$\widetilde{\mathbf{p}_i} = \widetilde{\mathbf{w}_i} + \beta_{i-1}\widetilde{\mathbf{p}_{i-1}}$

end if

$\mathbf{z}_i = A\mathbf{p}_i$

$\widetilde{\mathbf{z}_i} = A\widetilde{\mathbf{p}_i}$

$\alpha_i = \rho_{i-1}/(\widetilde{\mathbf{p}_i}, \mathbf{z}_i)$

$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\mathbf{p}_i$

$\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i\mathbf{z}_i$

$\widetilde{\mathbf{r}_i} = \widetilde{\mathbf{r}_{i-1}} - \alpha_i\widetilde{\mathbf{z}_i}$

収束判定

end for

CGS 法

$\mathbf{x}_0$: 初期値

$\mathbf{r}_0$: 初期残差 ($\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$)

$\widetilde{\mathbf{r}}$: $(\mathbf{r}_0, \widetilde{\mathbf{r}}) \neq 0$ を満足する任意ベクトル．例えば $\widetilde{\mathbf{r}} = \mathbf{r}_0$.

K: 前処理行列 (前処理なしの場合は $K = I$)

for $i = 1, 2, ...$

$\rho_{i-1} = (\widetilde{\mathbf{r}}, \mathbf{r}_{i-1})$

if $\rho_{i-1} = 0$ **then** 終了.

if $i = 1$ **then**

$\mathbf{u}_1 = \mathbf{r}_0$

$\mathbf{p}_1 = \mathbf{u}_1$

else

$\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$

$\mathbf{u}_i = \mathbf{r}_{i-1} + \beta_{i-1}\mathbf{q}_{i-1}$

$\mathbf{p}_i = \mathbf{u}_i + \beta_{i-1}(\mathbf{q}_{i-1} + \beta_{i-1}\mathbf{p}_{i-1})$

end if

$K\,\widehat{\mathbf{p}} = \mathbf{p}_i$ を解いて $\widehat{\mathbf{p}}$ を求める.

$\widehat{\mathbf{v}} = A\widehat{\mathbf{p}_i}$

$\alpha_i = \rho_{i-1}/(\widetilde{\mathbf{r}}, \widehat{\mathbf{v}})$

$\mathbf{q}_i = \mathbf{u}_i - \alpha_i\widehat{\mathbf{u}}$

$\widehat{\mathbf{u}}$ を $K\,\widehat{\mathbf{u}} = \mathbf{u}_i + \mathbf{q}_i$ を解いて求める.

$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\widehat{\mathbf{u}}$

$\mathbf{r}_i = \mathbf{r}_{i-1} - \alpha_i A\widehat{\mathbf{u}}$

収束判定

end for

BiCGSTAB 法

$\mathbf{x}_0$: 初期値

$\mathbf{r}_0$: 初期残差 ($\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$)

$\widetilde{\mathbf{r}}$: $(\mathbf{r}_0, \widetilde{\mathbf{r}}) \neq 0$ を満足する任意ベクトル. 例えば $\widetilde{\mathbf{r}} = \mathbf{r}_0$.

K: 前処理行列 (前処理なしの場合は $K = I$)

for $i = 1, 2, ...$

$\rho_{i-1} = (\widetilde{\mathbf{r}}, \mathbf{r}_{i-1})$

if $\rho_{i-1} = 0$ **then** 終了.

if $i = 1$ **then**

$\mathbf{p}_1 = \mathbf{r}_0$

else

$\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$

$\mathbf{p}_i = \mathbf{r}_i + \beta_{i-1}(_{i-1} - \omega_{i-1}\mathbf{v}_{i-1})$

end if

$\widehat{\mathbf{p}}$ を $K\widehat{\mathbf{p}} = \mathbf{p}_i$ を解いて求める.

$\widehat{\mathbf{v}_i} = A\widehat{\mathbf{p}}$

$\alpha_i = \rho_{i-1}/(\widetilde{\mathbf{r}}, \mathbf{v}_i)$

$\mathbf{s} = \mathbf{r}_{i-1} - \alpha_i \mathbf{v}_i$

if $\|s\|$ が十分に小さい **then**

$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \widehat{\mathbf{p}}$

終了.

end if

$\widehat{\mathbf{s}}$ を $K\widehat{\mathbf{s}} = \mathbf{s}$ を解いて求める.

$\mathbf{t} = A\widehat{\mathbf{s}}$

$\omega_i = (\mathbf{t}, \mathbf{s})/(\mathbf{t}, \mathbf{t})$

$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \widehat{\mathbf{p}} + \omega_i \widehat{\mathbf{s}}$

収束判定

$\mathbf{r}_i = \mathbf{s} - \omega_i \mathbf{t}$

$\omega_i \neq 0$ であれば反復続行.

end for

GPBiCG 法

$\mathbf{x}_0$: 初期値

$\mathbf{r}_0$: 初期残差 ($\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$)

$\widetilde{\mathbf{r}}$: $(\mathbf{r}_0, \widetilde{\mathbf{r}}) \neq 0$ を満足する任意ベクトル．例えば $\widetilde{\mathbf{r}} = \mathbf{r}_0$.

$\mathbf{u} = \mathbf{z} = 0$

for $i = 1, 2, ...$

$\rho_{i-1} = (\widetilde{\mathbf{r}}, \mathbf{r}_{i-1})$

if $\rho_{i-1} = 0$ **then** 終了.

if $i = 1$ **then**

$\mathbf{p} = \mathbf{r}_0$

$\mathbf{q} = A\mathbf{p}$

$\alpha_i = \rho_{i-1}/(\widetilde{r}, \mathbf{q})$

$\mathbf{t} = \mathbf{r}_{i-1} - \alpha_i\mathbf{q}$

$\mathbf{v} = A\mathbf{t}$

$\mathbf{y} = \alpha_i\mathbf{q} - \mathbf{r}_{i-1}$

$\mu_2 = (\mathbf{v}, \mathbf{t})$

$\mu_5 = (\mathbf{v}, \mathbf{v})$

$\zeta = \mu_2/\mu_5$

$\eta = 0$

else

$\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\zeta)$

$\mathbf{w} = \mathbf{v} + \beta_{i-1}\mathbf{q}$

$\mathbf{p} = \mathbf{r}_{i-1} + \beta_{i-1}(\mathbf{p} - \mathbf{u})$

$\mathbf{q} = A\mathbf{p}$

$\alpha_i = \rho_{i-1}/(\widetilde{r}, \mathbf{q})$

$\mathbf{s} = \mathbf{t} - \mathbf{r}_{i-1}$

$\mathbf{t} = \mathbf{r}_{i-1} - \alpha_i\mathbf{q}$

$\mathbf{v} = A\mathbf{t}$

$\mathbf{y} = \mathbf{s} - \alpha_i(\mathbf{w} - \mathbf{q})$

$\mu_1 = (\mathbf{y}, \mathbf{y})$

$\mu_2 = (\mathbf{v}, \mathbf{t})$

$\mu_3 = (\mathbf{y}, \mathbf{t})$

$\mu_4 = (\mathbf{v}, \mathbf{y})$

$\mu_5 = (\mathbf{v}, \mathbf{v})$

$\tau = \mu_5\mu_1 - \overline{\mu_4}\mu_4$

$\zeta = (\mu_1\mu_2 - \mu_3\mu_4)/\tau$

$\eta = (\mu_5\mu_3 - \overline{\mu_4}\mu_2)/\tau$

end if

$\mathbf{u} = \zeta\mathbf{q} + \eta(\mathbf{s} + \beta_{i-1}\mathbf{u})$

$\mathbf{z} = \zeta\mathbf{r}_{i-1} + \zeta\mathbf{z} - \alpha_i\mathbf{u}$

$\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i \mathbf{p} + \mathbf{z}$

収束判定

$\mathbf{r}_i = \mathbf{t} - \eta \mathbf{y} - \zeta \mathbf{u}$

$\zeta \neq 0$ であれば反復続行.

end for