(1) 3 次スプライン

```
     #include <stdio.h>
     #include <math.h>
S    #include <ksr/numarg.h>                                         /* パラレルリージョンを使用 */
     #include <ksr/ksr_timers.h>                                           /* 実行時間の計測 */

     typedef struct point{
           double  x;
           double  y;
       }element;
     element *point;                                                          /* ソースデータ */

     struct res{
           int  x;
           int  y;
       }result[12000];                                                          /* 結果データ */

S    int     THREADs;                                                     /* 並列実行スレッド数 */
     int     n, n2, count;
     double  a[3000][3000], *xd, *yd, *bx, *by, *u;
     FILE    *fout;                                                 /* 結果データ出力ファイル */

     void    worker();
     double  workU(int);
     void    workMATRIX(int, int);
     void    work1(int);
     void    work2(int);
     void    work3(int);
     void    work4(int);

     void main()
     {
S        int     TEAM_ID;
         int     i, j, k, p, type;
         double  sampX, as, us;

     /*----------------------------------------------------------------------------初期設定領域(始め)*/

S        printf(" input THREADs: ");
S        scanf("%d", &THREADs);                                        /* 実行スレッド数の取得 */

         printf(" input Number of Lines: ");
         scanf("%d", &n);                                       /* 計算を行う曲線の本数を取得 */

         type = 2;
         fout = fopen("ksrspl.txt", "w");
         fprintf(fout, "%d\n", type);

         k = 0;                                                       /* ソースデータの生成開始 */
         point = (element *)malloc(sizeof(element)*120*n);
         point[0].x = 0.0;
         point[0].y = 10.0;
         sampX = 1.0;
         for(i=0; i<n; i++){
             for(j=0; j<120; j++){
                 point[k+1].x = point[k].x + sampX * 10.0;
                 point[k+1].y = point[k].y + pow(-1.0, (double)j) * 3.0 * (double)(i+1);
                 k++;
             }
             sampX = sampX * (-1.0);
             point[k].y = point[k].y + 45.0;
         }                                                            /* ソースデータの生成終了 */

         n = n * 120;

         xd = (double *)malloc(sizeof(double)*n);
         yd = (double *)malloc(sizeof(double)*n);
         bx = (double *)malloc(sizeof(double)*n);
         by = (double *)malloc(sizeof(double)*n);
         u  = (double *)malloc(sizeof(double)*n);

         count = 0;

         n2 = n - 1;
         fprintf(fout, "%d\n", n2);

     /*--------------------------------------------------------------------------初期設定領域(終わり)*/
     /*----------------------------------------------------------------------------並列ブロック(始め)*/

         as = all_seconds();
         us = user_seconds();

S        TEAM_ID = pr_create_team(THREADs);                         /* 並列処理を行うチームの生成 */
S        pr_pcall(TEAM_ID, worker, copyargs());                          /* パラレルリージョン起動 */

     /*--------------------------------------------------------------------------並列ブロック(終わり)*/
     /*----------------------------------------------------------------------------結果出力領域(始め)*/

         printf(" CPU time = %f\n", user_seconds()-us);
         printf(" Elapsed time = %f\n", all_seconds()-as);

MP       for(p=0; p<count+1; p++)
             fprintf(fout, "%5d%5d", result[p].x, result[p].y);          /* 計算結果データの格納 */

         fclose(fout);

     /*--------------------------------------------------------------------------結果出力領域(終わり)*/
     }

     void worker()                                                          /* 並列処理を行う関数 */
     {
S        int  THREADno, offset, tmp;
N        int  loop_begin, loop_end;
         int  i;

     /*--------------------------------------------------------------------------タスク分割領域(始め)*/

S        THREADno = pr_mid();                                                 /* スレッド番号の取得 */
MS       offset = (n2+1)/THREADs;
MS       tmp    = (n2+1)%THREADs;

N        loop_begin = THREADno * offset;
N        loop_end   = loop_begin + offset - 1;
MS       if (THREADno < tmp) loop_end++;
N        if (THREADno == (THREADs-1)) loop_end--;

     /*------------------------------------------------------------------------タスク分割領域(終わり)*/
```

```
     /*------------------------------------------------------------------------------計算領域（始め）*/

MP       for(i=loop_begin; i<loop_end+1; i++)  u[i] = workU(i);

N        if (THREADno == 0)  loop_begin = 1;

MP       for(i=loop_begin; i<loop_end+1; i++)  work1(i);
MP       for(i=loop_begin; i<loop_end+1; i++)  work2(i);

MP       workMATRIX(loop_begin, loop_end);

N        if (THREADno == 0)  xd[0] = (point[1].x - point[0].x)/(2.0*u[0]);
N        if (THREADno == 1)  yd[0] = (point[1].y - point[0].y)/(2.0*u[0]);
N        if (THREADs == 2){
N            if (THREADno == 0)  xd[n-1] = (point[n-1].x - point[n-2].x)/(2.0*u[n-2]);
N            if (THREADno == 1)  yd[n-1] = (point[n-1].y - point[n-2].y)/(2.0*u[n-2]);
         }
N        if ((THREADs > 3) && (THREADno == 2))
             xd[n-1] = (point[n-1].x - point[n-2].x)/(2.0*u[n-2]);
N        if ((THREADs > 3) && (THREADno == 3))
             yd[n-1] = (point[n-1].y - point[n-2].y)/(2.0*u[n-2]);

MP       for(i=loop_begin; i<loop_end+1; i++)  work3(i);
MP       for(i=loop_begin; i<loop_end+1; i++)  work4(i);

     /*----------------------------------------------------------------------------計算領域（終わり）*/
     }

     double workU(int i)                                                    /* 2 点間の距離を計算 */
     {
         double  u, sub1, sub2;

         sub1 = (point[i+1].x - point[i].x)*(point[i+1].x - point[i].x);
         sub2 = (point[i+1].y - point[i].y)*(point[i+1].y - point[i].y);
         u = sqrt(sub1 + sub2);
         return  u;
     }

     void work1(int  no)                                                           /* 行列の生成 */
     {
         int  j;

MP       for(j=1; j<no+1; j++)  a[no+1][j] = 0;
         a[no][no-1] = u[no-1]/3.0;
         a[no][no]   = 2.0*(u[no-1] + u[no])/3.0;
         a[no][no+1] = u[no]/3.0;
     }

     void work2(int  no)                                                /* 行列式の右辺の計算 */
     {
         bx[no] = u[no-1]*point[no+1].x/u[no] +
                  (u[no]/u[no-1] - u[no-1]/u[no])*point[no].x -
                  u[no]*point[no-1].x/u[no-1];
         by[no] = u[no-1]*point[no+1].y/u[no] +
                  (u[no]/u[no-1] - u[no-1]/u[no])*point[no].y -
                  u[no]*point[no-1].y/u[no-1];
     }
```

```
MP   void workMATRIX(int  min, int  max)                              /* 行列の連立方程式の計算 */
     {
         int     i;
         double  devide;

MP       for(i=min; i<max+1; i++){
             devide      = a[i+1][i]/a[i][i];
             a[i+1][0]   = a[i+1][0] - a[i][0]*devide;
             a[i+1][i]   = a[i+1][i] - a[i][i]*devide;
             a[i+1][i+1] = a[i+1][i+1] - a[i][i+1]*devide;
             bx[i+1]     = bx[i+1] - bx[i]*devide;
             by[i+1]     = by[i+1] - by[i]*devide;
         }
MP       for(i=max; i>min-1; i--){
             devide      = a[i-1][i]/a[i][i];
             a[i-1][0]   = a[i-1][0] - a[i][0]*devide;
             a[i-1][i]   = a[i-1][i] - a[i][i]*devide;
             a[i-1][max] = a[i-1][max] - a[i][max]*devide;
             bx[i-1]     = bx[i-1] - bx[i]*devide;
             by[i-1]     = by[i-1] - by[i]*devide;
         }
     }

     void work3(int  no)                                                      /* 微分値の計算 */
     {
         xd[no] = (bx[no] - (a[no][0]*xd[0] + a[no][n-1]*xd[n-1]))/a[no][no];
         yd[no] = (by[no] - (a[no][0]*yd[0] + a[no][n-1]*yd[n-1]))/a[no][no];
     }

     void work4(int  no)                                          /* 微分値から補間の計算を行う */
     {
         int     j;
         float   fx, fy;
         double  t, bx0, bx1, bx2, bx3, by0, by1, by2, by3;

         bx0 = point[no].x;
         bx1 = xd[no];
         bx2 = 3.0*(point[no+1].x - point[no].x)/(u[no]*u[no]) -
               (2.0*xd[no] + xd[no+1])/u[no];
         bx3 = 2.0*(point[no].x - point[no+1].x)/(u[no]*u[no]*u[no]) +
               (xd[no] + xd[no+1])/(u[no]*u[no]);

         by0 = point[no].y;
         by1 = yd[no];
         by2 = 3.0*(point[no+1].y - point[no].y)/(u[no]*u[no]) -
               (2.0*yd[no] + yd[no+1])/u[no];
         by3 = 2.0*(point[no].y - point[no+1].y)/(u[no]*u[no]*u[no]) +
               (yd[no] + yd[no+1])/(u[no]*u[no]);

         for(j=1; j<=100; j++){                                         /* 補間の座標値を計算 */
             t  = u[no]*(double)j/100.0;
             fx = (bx0 + t*(bx1 + t*(bx2 + t*bx3)));
             fy = (by0 + t*(by1 + t*(by2 + t*by3)));
             result[count].x = fx;  result[count].y = fy;
             count++;
         }
     }
```

(2) エッジ抽出

```
     #include <X11/Xlib.h>
     #include <X11/XWDFile.h>
     #include <stdio.h>
S    #include <ksr/numarg.h>                                   /* パラレルリージョンを使用 */
     #include <ksr/ksr_timers.h>                                         /* 実行時間の計測 */

     XColor                xcolor[256];
     static unsigned char  buff[307200], data[307200];
     unsigned long         size;
S    int                   THREADs;                                    /* 並列実行スレッド数 */
     int                   indata[480][640];                                /* ソースデータ */
     int                   outdata[480][640];                                 /* 結果データ */
     int                   ixm, iym, th;

     void worker();
     void edge();

     main()
     {
         XWDFileHeader         fh;
         static unsigned char  title[40], name[40];
         unsigned long         i, bytes_per_pixel;
S        int                   TEAM_ID;
         int                   flag, fx, fy, m, n;
         float                 fc;
         double                as, us;

         FILE                  *fin, *fout;

     /*----------------------------------------------------------------------------初期設定領域（始め）*/

S        printf(" input THREADs: ");
S        scanf("%d", &THREADs);                                      /* 実行スレッド数の取得 */

         fin = fopen("pansy.xwd", "r");                        /* 画像データの読み込み（開始）*/
         flag = fread(&fh, sizeof(XWDFileHeader), 1, fin);
         flag = fread(title, fh.header_size - 100, 1, fin);
         for(i=0; i<fh.ncolors; i++)
             flag = fread(&xcolor[i], sizeof(XColor), 1, fin);
         bytes_per_pixel = fh.bits_per_pixel / 8;
         size = fh.window_width * fh.window_height;
         flag = fread(buff, bytes_per_pixel, size, fin);
         fclose(fin);

         th = 25000;
         ixm = fh.window_width;
         iym = fh.window_height;                               /* 画像データの読み込み（終了）*/

     /*--------------------------------------------------------------------------初期設定領域（終わり）*/
     /*----------------------------------------------------------------------------並列ブロック（始め）*/

         as = all_seconds();
         us = user_seconds();

S        TEAM_ID = pr_create_team(THREADs);                    /* 並列処理を行うチームの生成 */
S        pr_pcall(TEAM_ID, worker, copyargs());                   /* パラレルリージョン起動 */

     /*--------------------------------------------------------------------------並列ブロック（終わり）*/

     /*------------------------------------------------------------------------------結果出力領域（始め）*/

         printf(" CPU time = %f\n", user_seconds()-us);
         printf(" Elapsed time = %f\n", all_seconds()-as);

         fout = fopen("edge.txt", "w");                          /* 計算結果の画像データの書き込み */
         fprintf(fout, "%d\n", size);
         for(i=0; i<size; i++)  fprintf(fout, "%5d", data[i]);
         fclose(fout);

     /*----------------------------------------------------------------------------結果出力領域（終わり）*/
     }

     void worker()                                                       /* 並列処理を行う関数 */
     {
S        int            THREADno;
S        int            tmp;
N        int            begin2, end2;
         int            m, n, gray;
S        unsigned long  offset;
N        unsigned long  begin1, end1;
         unsigned long  i, j, k;

     /*----------------------------------------------------------------------------タスク分割領域（始め）*/

S        THREADno = pr_mid();                                                /* スレッド番号の取得 */

S        offset = size/THREADs;
S        tmp    = size%THREADs;
N        if(THREADno<tmp){
N            begin1 = THREADno*offset + THREADno;
N            end1   = begin1 + offset + 1;
         }
N        else{
N            begin1 = THREADno*offset + tmp;
N            end1   = begin1 + offset;
         }

S        offset = iym/THREADs;
S        tmp    = iym%THREADs;
N        if(THREADno<tmp){
N            begin2 = THREADno*offset + THREADno;
N            end2   = begin2 + offset + 1;
         }
N        else{
N            begin2 = THREADno*offset + tmp;
N            end2   = begin2 + offset;
         }

     /*--------------------------------------------------------------------------タスク分割領域（終わり）*/
     /*-----------------------------------------------------------------------------------計算領域（始め）*/

MP       for(i=begin1; i<end1; i++){                                              /* 入力データの２値化 */
             gray = (int)((float)xcolor[buff[i]].red   * 0.299 +
                          (float)xcolor[buff[i]].green * 0.587 +
                          (float)xcolor[buff[i]].blue  * 0.114);

             n = i/245;  m = i%245;
```

```
                if(gray<th)  indata[n][m] = 0;
                    else  indata[n][m] = 1;
            }

MP          edge(begin2, end2);

            th = 150;

MP          for(i=begin2; i<end2; i++)                                      /* 出力データの変換 */
                for(j=0; j<ixm; j++){
                    gray = outdata[i][j];
                    k = i*245 + j;
                    if(gray>=th)  data[k] = 0;
                        else  data[k] = 255;
                }

     /*------------------------------------------------------------------------------計算領域（終わり）*/
     }

MP   void edge(begin2, end2)                                                /* エッジ抽出の計算 */
N           int  begin2, end2;
     {
         int  i, j;

MP          for(i=begin2; i<end2; i++)
                for(j=0; j<ixm; j++)
                    if((indata[i][j]!=0) &&
                        (j!=0 || i!=0 || j!=ixm-1 || i!=iym-1) &&
                        (indata[i][j]==indata[i-1][j]) &&
                        (indata[i][j]==indata[i][j-1]) &&
                        (indata[i][j]==indata[i+1][j]) &&
                        (indata[i][j]==indata[i][j+1]))
                            outdata[i][j] = 255;
                    else  outdata[i][j] = 0;
     }
```

(3) 包装アルゴリズム

```
     #include <stdio.h>
S    #include <pthread.h>                                                    /* pthread システムコール */
S    #include <ksr/numarg.h>                                                /* パラレルリージョンを使用 */
     #include <ksr/ksr_timers.h>                                                     /* 実行時間の計測 */
     #define  N  80000

S    pthread_barrier_t  barrierA, barrierB;                                       /* バリア A/B の宣言 */

     typedef struct point{
           double  x;
           double  y;
       }element;

S    int      THREADs;                                                          /* 並列実行スレッド数 */
N    int      check_min[16];                                   /* 各スレッドの最小座標点番号（局所変数）*/
     int      N2, point_min;
     element  *point;                                                     /* ソースデータ，結果データ */
N    double   check_angle[16];                                       /* 各スレッドの最小角度（局所変数）*/
     double   *result_angle;                                                         /* 計算結果の角度 */

     void    worker();
     double  theta();
     void    input_data();

     main()
     {
         double  as, us;
         int     i;
S        int     TEAM_ID;

     /*----------------------------------------------------------------------------初期設定領域（始め）*/

S        printf(" input Num of THREADs: ");
S        scanf("%d", &THREADs);                                                 /* 実行スレッド数の取得 */

         point        = (element *)malloc(sizeof(element)*(N+1));
         result_angle = (double *)malloc(sizeof(double)*(N));

         input_data();
         printf(" *Input Data [%d] OK!\n\n", N);

S        pthread_barrier_init(&barrierA, pthread_barrierattr_default, THREADs);     /* バリア A の生成 */
S        pthread_barrier_init(&barrierB, pthread_barrierattr_default, THREADs);     /* バリア B の生成 */

     /*--------------------------------------------------------------------------初期設定領域（終わり）*/
     /*----------------------------------------------------------------------------並列ブロック（始め）*/

         as = all_seconds();
         us = user_seconds();

S        TEAM_ID = pr_create_team(THREADs);                               /* 並列処理を行うチームの作成 */
S        pr_pcall(TEAM_ID, worker, copyargs());                              /* パラレルリージョン起動 */

     /*--------------------------------------------------------------------------並列ブロック（終わり）*/
     /*----------------------------------------------------------------------------結果出力領域（始め）*/

         printf(" CPU time = %f\n", user_seconds()-us);
         printf(" Elapsed time = %f\n", all_seconds()-as);
```

```
         for(i=0; i<=point_min; i++)                                      /* 計算結果の出力 */
             printf(" x[%2d]=%11f,  y[%2d]=%11f,  angle=%11f\n",
                      i, point[i].x, i, point[i].y, result_angle[i]);

     /*-------------------------------------------------------------------------結果出力領域(終わり)*/
     }

     void worker()                                                        /* 並列処理を行う関数 */
     {
S        int      THREADno;
         int      i, num, min;
         double   angle, minangle;
         element  t;

S        THREADno = pr_mid();                                             /* スレッド番号の取得 */
         num = 0;
         result_angle[0] = 0.0;

     /*-----------------------------------------------------------------------繰り返し計算領域(始め)*/

         while(1){                                                 /* 凸包上の点を繰り返し求める */
S            pthread_barrier_checkin(&barrierA, THREADno);                 /* バリア A チェックイン */

MP           min = point_min;
             if(min==N){                                /* point[min] の座標点が始点と一致すれば終了 */
                 if(THREADno==0){
                     t = point[num];  point[num] = point[min];  point[min] = t;
                     point_min = num;
                 }
                 break;
             }

N            if(THREADno==0){
                 t = point[num];  point[num] = point[min];  point[min] = t;
             }

MP           angle = result_angle[num];
             minangle = 360.0;
MP           for(i=num+THREADno+1; i<=N; i=i+THREADs)
                             /* 直前に求めた線分と線分とのなす角より大きく，最小となる座標点を求める */
                 if(theta(point[num], point[i])>angle)
                     if(theta(point[num], point[i])<minangle){
                         min = i;
                         minangle = theta(point[num], point[min]);
                     }
N            check_angle[THREADno] = minangle;                /* 各スレッドの最小角度を局所変数に格納 */
N            check_min[THREADno]   = min;               /* 各スレッドの最小座標点番号を局所変数に格納 */

S            pthread_barrier_checkout(&barrierA, THREADno);              /* バリア A チェックアウト */

S            if(THREADno==0){                                 /* マスタースレッドが各スレッドの        */
N                minangle = check_angle[0];                   /*    局所変数から最小角度・座標点を決定 */
N                point_min = check_min[0];
N                for(i=1; i<THREADs; i++)
N                    if(check_angle[i]<minangle){
N                        minangle = check_angle[i];
N                        point_min = check_min[i];
                     }
                 result_angle[num+1] = minangle;
             }
             num++;

S            pthread_barrier_checkin(&barrierB, THREADno);                 /* バリア B チェックイン */

MP           min = point_min;
             if(min==N){
                 if(THREADno==0){
                     t = point[num];  point[num] = point[min];  point[min] = t;
                     point_min = num;
                 }
                 break;
             }

N            if(THREADno==0){
                 t = point[num];  point[num] = point[min];  point[min] = t;
             }

MP           angle = result_angle[num];
             minangle = 360.0;
MP           for(i=num+THREADno+1; i<=N; i=i+THREADs)
                 if(theta(point[num], point[i])>angle)
                     if(theta(point[num], point[i])<minangle){
                         min = i;
                         minangle = theta(point[num], point[min]);
                     }
N            check_angle[THREADno] = minangle;
N            check_min[THREADno]   = min;

S            pthread_barrier_checkout(&barrierB, THREADno);              /* バリア B チェックアウト */

S            if(THREADno==0){
N                minangle = check_angle[0];
N                point_min = check_min[0];
N                for(i=1; i<THREADs; i++)
N                    if(check_angle[i]<minangle){
N                        minangle = check_angle[i];
N                        point_min = check_min[i];
                     }
                 result_angle[num+1] = minangle;
             }
             num++;
         }

     /*---------------------------------------------------------------------繰り返し計算領域(終わり)*/
     }

     double theta(p1, p2)                                  /* 線分 p1p2 と水平正方向とのなす角 */
         element  p1, p2;
     {
         double  dx, dy, ax, ay, t;
         int     dxx, dyy;

         dxx = (int)p2.x - (int)p1.x;    ax = abs(dxx);
         dyy = (int)p2.y - (int)p1.y;    ay = abs(dyy);

         dx = (double)dxx;  dy = (double)dyy;
```

```
    if((ax+ay)==0)  t = 0;
        else  t = dy/(ax+ay);
    if(dx<0)  t = 2-t;
        else if(dy<0)  t = 4+t;
    t = t*90.0;

    return  t;
}

void input_data()                                             /* 入力データの座標値の x, y を生成 */
{
    int     m, m1, a, b, p1, p2, q1, q2, multi, i, j, mod;
    double  random;

    m  = 100000000;
    m1 = 10000;
    a  = 1234567;
    b  = 31415821;
    point = (element *)malloc(sizeof(element)*(N+1));
    N2 = N*2;
    j = 0;
    for(i=0; i<N2; i++){                                          /* 入力データのランダム生成 */
        p1 = a/m1;  p2 = a%m1;
        q1 = b/m1;  q2 = b%m1;
        multi = (((p2*q1 + p1*q2)%m1)*m1 + p2*q2)%m;
        a = (multi+1)%m;
        random = a/100000.0;
        mod = i%2;
        if(mod==0)  point[j].x = random;
            else{
                point[j].y = random;
                j++;
            }
    }

    point_min = 0;
    for(i=1; i<N; i++)                                 /* y 座標が最小の座標点を求め，始点とする */
        if(point[i].y<point[point_min].y)  point_min = i;
    point[N] = point[point_min];                                /* 始点を point[N] に格納 */
    result_angle[0] = 0.0;
}
```

(4) 巡回セールスマン問題

```
     #include <stdio.h>
     #include <math.h>
S    #include <pthread.h>                                                    /* pthread システムコール */
S    #include <ksr/numarg.h>                                               /* パラレルリージョンを起動 */
     #include <ksr/ksr_timers.h>                                                  /* 実行時間の計測 */
     #define  N  324

     typedef struct link{
           int  a;                                                                  /* リンク先番号 */
           int  b;                                                          /* リンク先の距離: 1-9 */
       }element;

S    pthread_mutex_t  mu;                                                      /* ミューテックスの宣言 */
S    int  actThreads;                                                        /* アクティブスレッド数 */
S    int  THREADs;                                                                /* 実行スレッド数 */
S    int  *actBox;                                              /* アクティブ/アイドルスレッドのフラグ */
     element  data[N+1][4];                                 /* ソースデータ，データ数: N，枝番号: 0-3 */
     int  *result;                                                  /* 結果データ，探索結果の番号 */
MP   int  temp[16][N+2];                                                   /* 結果データの一時保管 */
MP   int  val[16][N+1];                                   /* 1-N: 探索マーク，既探索: 1，未探索: 0 */
MP   int  *id;                                                                      /* 探索の深さ */
     int  best;                                                                   /* 最短移動距離 */
N    int  *kBox;                                                                  /* 訪問点の番号 */
N    int  *lengthBox;                                                                 /* 移動距離 */

     void input_data();
     void worker();
     void visit();
     void result_check();

     main()
     {
         double  as, us;
         int     i, j;
S        int     TEAM_ID;

     /*---------------------------------------------------------------------------初期設定領域（始め）*/

S        printf(" Input Num. of THREADs: ");
S        scanf("%d", &THREADs);                                             /* 実行スレッド数の取得 */

S        actBox    = (int *)malloc(sizeof(int)*THREADs);
         result    = (int *)malloc(sizeof(int)*(N+2));
N        kBox      = (int *)malloc(sizeof(int)*THREADs);
N        lengthBox = (int *)malloc(sizeof(int)*THREADs);
N        id        = (int *)malloc(sizeof(int)*THREADs);

         input_data();
         printf(" *Input Data [%d] OK!\n", N);

MP       for(i=0; i<THREADs; i++)  val[i][0] = 1;                                /* 枝無しにマーク */
MP       for(i=0; i<THREADs; i++) for(j=1; j<N; j++)  val[i][j] = 0;          /* 探索マークの初期化 */
MP       for(i=0; i<THREADs; i++)  id[i] = 0;                                       /* 深さの初期化 */
         best = N*9+1;                               /* 最短移動距離の初期値として最大移動値を格納 */

S        pthread_mutex_init(&mu, pthread_mutexattr_default);                /* ミューテックスの生成 */

S        for(i=0; i<THREADs; i++) actBox[i] = 0;                      /* アクティブ: 1，アイドル: 0 */
```

```
S           actThreads = 1;                                          /* アクティブスレッド: 1個 */
S           actBox[0]  = 1;                                       /* マスタースレッド: active */
N           kBox[0] = 1;                                              /* 訪問点の番号の初期化 */
N           lengthBox[0] = 0;                                             /* 移動距離の初期化 */

        /*-------------------------------------------------------------------------初期設定領域 (終わり)*/
        /*---------------------------------------------------------------------------並列ブロック (始め)*/

            as = all_seconds();
            us = user_seconds();

S           TEAM_ID = pr_create_team(THREADs);                      /* 並列処理を行うチームの作成 */
S           pr_pcall(TEAM_ID, worker, copyargs());                    /* パラレルリージョンを起動 */

        /*-------------------------------------------------------------------------並列ブロック (終わり)*/
        /*---------------------------------------------------------------------------結果出力領域 (始め)*/

            printf(" CPU time = %f\n", user_seconds()-us);
            printf(" Elapsed time = %f\n", all_seconds()-as);

            result_check();

        /*-------------------------------------------------------------------------結果出力領域 (終わり)*/
        }

        void worker()                                                   /* 並列処理を行う関数 */
        {
S           int  THREADno;
S           int  flag;                                              /* タスクが来るのを待つフラグ */
N           int  a, b;

S           THREADno = pr_mid();                                            /* スレッド番号の取得 */
S           while (1){
S               flag = 0;
S               while((actThreads!=0)&&(flag==0)) flag = actBox[THREADno];      /* タスクが来るのを待つ */
S               if(actThreads==0) break;                         /* アクティブスレッドがなくなれば終了 */
N               a = kBox[THREADno];                                          /* 訪問点の番号を格納 */
N               b = lengthBox[THREADno];                                /* 現時点での移動距離を格納 */

MP              visit(a, b, THREADno);

S               pthread_mutex_lock(&mu);                                   /* ミューテックスのロック */
S               actThreads--;                                      /* アクティブスレッド数を減らす */
S               actBox[THREADno] = 0;                                        /* 自分をアイドルにする */
S               pthread_mutex_unlock(&mu);                              /* ミューテックスのアンロック */
            }
        }

MP      void visit(k, length, no)                                                 /* 訪問点の探索 */
            int  k;                                                         /* 訪問点のデータ番号 */
            int  length;                                                    /* 現時点での移動距離 */
        {
            int  i, j;                                                                /* カウンタ */
            int  t;                                                         /* リンク先番号を格納 */
S           int  idleNo;                                              /* アイドルスレッド番号を格納 */

MP          id[no]++;                                                     /* 深さのインクリメント */
MP          temp[no][id[no]] = k;                                /* 結果データ一時保管に現在点を格納 */
MP          val[no][k] = 1;                                                    /* 探索マークに (既) */
```

```
            for(i=0; i<4; i++){                                          /* 4方向への枝のループ制御 */
                t = data[k][i].a;                                              /* リンク先番号を格納 */
MP              if(val[no][t]==0){                              /* 経路探索を失敗していないかの判断制御 */
                    length = length + data[k][i].b;                              /* 現移動距離の計算 */
N                   pthread_mutex_lock(&mu);
                    if(length<best){                                /* 現時点での移動距離が最短となりうるか */
N                       pthread_mutex_unlock(&mu);
MP                      if((id[no]+1)==N){
                        if((data[t][0].a==1)||(data[t][3].a==1)){
                                                         /* 全ての点を通ってきたか，かつ始点と接しているか */
                            if(data[t][0].a==1) length = length + data[t][0].b;
                                                                          /* 最後の点への移動距離を計算 */
                            if(data[t][3].a==1) length = length + data[t][3].b;
                                                                              /* 始点への移動距離の計算 */
N                           pthread_mutex_lock(&mu);
                            if(length<best){                                  /* 移動距離がベストより短いか */
                                best = length;                                  /* 移動距離のベストを更新 */
N                               pthread_mutex_unlock(&mu);
MP                              temp[no][id[no]+1] = t;
                                                                  /* 最後の訪問点を結果データ一時保管に格納 */
MP                              temp[no][id[no]+2] = 1;
                                                                          /* 始点を結果データ一時保管に格納 */
N                               pthread_mutex_lock(&mu);
MP                              for(j=1; j<=N+1; j++)  result[j] = temp[no][j];
                                                                  /* 結果データ一時保管を結果データに更新 */
                            }
N                           pthread_mutex_unlock(&mu);
                        }}
S                       idleNo = 0;
S                       pthread_mutex_lock(&mu);
S                       if(actThreads<THREADs){                                 /* アイドルスレッドがあれば */
S                           actThreads++;                          /* アクティブスレッド数のインクリメント */
S                           while((idleNo<THREADs)&&(actBox[idleNo]==1))  idleNo++;
                                                                              /* アイドルスレッドの探索 */
N                           kBox[idleNo] = t;                                      /* 訪問点番号の受渡し */
N                           lengthBox[idleNo] = length;                              /* 移動距離の受渡し */
N                           id[idleNo] = id[no];                                         /* 深さの受渡し */
N                           for(j=1; j<=id[no]; j++)  temp[idleNo][j] = temp[no][j];
                                                                                  /* 探索経路の受渡し */
N                           for(j=1; j<=N; j++) val[idleNo][j] = val[no][j];       /* 探索マークの受渡し */
S                           actBox[idleNo] = 1;                                /* スレッドをアクティブにする */
S                           pthread_mutex_unlock(&mu);
                        }
                        else {
S                           pthread_mutex_unlock(&mu);
                            visit(t, length, no);                                    /* リンク先への探索 */
                        }
                    }
N                   else pthread_mutex_unlock(&mu);
                }
            }
MP          id[no]--;                          /* 探索に失敗し，リンクを戻っていく時の深さのデクリメント */
MP          val[no][k] = 0;           /* 探索に失敗し，リンクを戻っていく時に探索マーク (既) を (未) に変換 */
        }

        void input_data()                                                     /* ソースデータの生成 */
        {
            int  base, i, loop1, loop2, weight, edge;
```

```
    base = (int)sqrt((float)N);
    loop1 = 1;  loop2 = base;
    for(i=loop1; i<=loop2; i++) data[i][0].a = 0;
    loop1 = base;  loop2 = N;
    for(i=loop1; i<=loop2; i=i+base) data[i][1].a = 0;
    loop1 = base*(base-1)+1;  loop2 = N;
    for(i=loop1; i<=loop2; i++) data[i][2].a = 0;
    loop1 = 1;  loop2 = base*(base-1)+1;
    for(i=loop1; i<=loop2; i=i+base) data[i][3].a = 0;

    loop1 = base+1;  loop2 = N;
    for(i=loop1; i<=loop2; i++){
        data[i][0].a = i-base;
        data[i-base][2].a = i;
        weight = rand()%10;
        if(weight==0){
            weight = i%10;
            if(weight==0) weight = 3;
        }
        data[i][0].b = weight;
        data[i-base][2].b = weight;
    }

    loop1 = 1;  loop2 = N-1;
    for(i=loop1; i<=loop2; i++){
        edge = i%base;
        if(edge!=0){
            data[i][1].a = i+1;
            data[i+1][3].a = i;
            weight = rand()%10;
            if(weight==0){
                weight = i%10;
                if(weight==0) weight = 3;
            }
            data[i][1].b = weight;
            data[i+1][3].b = weight;
        }
    }
}

void result_check()                                                           /* 結果のチェック */
{
    int  i;

    printf(" *Print Result\n");
    printf("   Best Weight: %d\n", best);
    for(i=1; i<=N+1; i++) printf("  %2d", result[i]);
    printf("\n");
}
```