```c
#include <stdio.h>
#include <math.h>

double f( double );

double f( double a )
{
   return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[])
{
   int n, i;
   double PI25DT = 3.141592653589793238462643;
   double pi, h, sum, x;

   scanf("&d",&n);

   h   = 1.0 / (double) n;
   sum = 0.0;
#pragma omp parallel for private(x) reduction(+:sum)
   for (i = 1; i <= n; i++){
      x = h * ((double)i - 0.5);
      sum += f(x);
   }
   pi = h * sum;

   printf("pi is approximately %.16f, Error is %.16f¥n",
         pi, fabs(pi - PI25DT));

   return 0;
}
```

```c
* cpi mpi version */
#include "mpi.h"
#include <stdio.h>
#include <math.h>

double f( double );

double f( double a )
{
   return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[])
{
   int done = 0, n, myid, numprocs, i;
   double PI25DT = 3.141592653589793238462643;
   double mypi, pi, h, sum, x;
   double startwtime = 0.0, endwtime;
   int  namelen;
   char processor_name[MPI_MAX_PROCESSOR_NAME];

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   MPI_Get_processor_name(processor_name,&namelen);

   fprintf(stderr,"Process %d on %s¥n", myid, processor_name);
   if(mypid == 0) scanf("%d",&n);
   startwtime = MPI_Wtime();
   MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);

   h   = 1.0 / (double) n;
   sum = 0.0;
   for (i = myid + 1; i <= n; i += numprocs){
      x = h * ((double)i - 0.5);
      sum += f(x);
   }
   mypi = h * sum;
   MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

   if (myid == 0){
      printf("pi is approximately %.16f, Error is %.16f¥n",
            pi, fabs(pi - PI25DT));
      endwtime = MPI_Wtime();
      printf("wall clock time = %f¥n",
            endwtime-startwtime);
   }
   MPI_Finalize();
   return 0;
}
```

```
/*
 * Laplace equation with explict method
 */
#include <stdio.h>
#include <math.h>

/* square region */
#define XSIZE 1000
#define YSIZE 1000

#define PI 3.1415927
#define NITER 100

double u[XSIZE+2][YSIZE+2],uu[XSIZE+2][YSIZE+2];

double time1,time2;
double second();

void initialize();
void lap_solve();

main()
{
   initialize();

   time1 = second();
   lap_solve();
   time2 = second();

   printf("time=%g¥n",time2-time1);
   exit(0);
}

void lap_solve()
{
   int x,y,k;
   double sum;

#pragma omp parallel private(k,x,y)
{
   for(k = 0; k < NITER; k++){
      /* old <- new */
#pragma omp for
      for(x = 1; x <= XSIZE; x++)
        for(y = 1; y <= YSIZE; y++)
          uu[x][y] = u[x][y];
      /* update */
#pragma omp for
      for(x = 1; x <= XSIZE; x++)
        for(y = 1; y <= YSIZE; y++)
          u[x][y] = (uu[x-1][y] + uu[x+1][y] + uu[x][y-1] + uu[x][y+1])/4.0;
   }
 }

/* check sum */
   sum = 0.0;
#pragma omp parallel for private(y) reduction(+:sum)
   for(x = 1; x <= XSIZE; x++)
      for(y = 1; y <= YSIZE; y++)
        sum += (uu[x][y]-u[x][y]);
   printf("sum = %g¥n",sum);
}

void initialize()
{
   int x,y;

   /* initalize */
   for(x = 1; x <= XSIZE; x++)
     for(y = 1; y <= YSIZE; y++)
       u[x][y]          =          sin((double)(x-1)/XSIZE*PI)          +
cos((double)(y-1)/YSIZE*PI);

   for(x = 0; x < (XSIZE+2); x++){
      u[x][0] = 0.0;
      u[x][YSIZE+1] = 0.0;
      uu[x][0] = 0.0;
      uu[x][YSIZE+1] = 0.0;
   }

   for(y = 0; y < (YSIZE+2); y++){
      u[0][y] = 0.0;
      u[XSIZE+1][y] = 0.0;
      uu[0][y] = 0.0;
      uu[XSIZE+1][y] = 0.0;
   }
}
```

```
/*
 * Laplace equation with explict method
 */
#include "mpi.h"
#include <stdio.h>
#include <math.h>

/* square region */
#define XSIZE 1000
#define YSIZE 1000

#define PI 3.1415927
#define NITER 100

double u[XSIZE+2][YSIZE+2],uu[XSIZE+2][YSIZE+2];

double time1,time2;
double second();

void initialize();
void lap_solve();

int myid, numprocs;
int  namelen;
char processor_name[MPI_MAX_PROCESSOR_NAME];

int xsize;

main(int argc, char *argv[])
{
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   MPI_Get_processor_name(processor_name,&namelen);

   fprintf(stderr,"Process %d on %s¥n", myid, processor_name);

   xsize = XSIZE/numprocs;
   if((XSIZE%numprocs) != 0){
      printf("bad number of processor, sorry!¥n");
      exit(1);
   }

   initialize();

   MPI_Barrier(MPI_COMM_WORLD);
   if(myid == 0) time1 = second();
   lap_solve();

   MPI_Barrier(MPI_COMM_WORLD);
   if(myid == 0) time2 = second();
   if(myid == 0) printf("time=%g¥n",time2-time1);

   MPI_Finalize();
   exit(0);
}

#define TAG_1 100
#define TAG_2 101

void lap_solve()
{
   int x,y,k;
   double sum;
   double t_sum;
   int x_start,x_end;
   MPI_Request req1,req2;
   MPI_Status status1,status2;

   x_start = 1 + xsize*myid;
   x_end  = 1 + xsize*(myid+1);

   for(k = 0; k < NITER; k++){
      /* old <- new */
      for(x = x_start; x < x_end; x++)
        for(y = 1; y <= YSIZE; y++)
          uu[x][y] = u[x][y];

      if(myid != 0)  /* recv from down */
         MPI_Irecv(&uu[x_start-1][0],YSIZE,MPI_DOUBLE,myid-1,TAG_1,
                   MPI_COMM_WORLD,&req1);
      if(myid != (numprocs -1)) /* recv from up */
         MPI_Irecv(&uu[x_end][0],YSIZE,MPI_DOUBLE,myid+1,TAG_2,
                   MPI_COMM_WORLD,&req2);

      if(myid != 0) /* send to down */
         MPI_Send(&u[x_start][0],YSIZE,MPI_DOUBLE,myid-1,TAG_2,
                  MPI_COMM_WORLD);
      if(myid != (numprocs-1)) /* send to up */
         MPI_Send(&u[x_end-1][0],YSIZE,MPI_DOUBLE,myid+1,TAG_1,
                  MPI_COMM_WORLD);

      if(myid != 0) MPI_Wait(&req1,&status1);
      if(myid != (numprocs -1)) MPI_Wait(&req2,&status2);

      /* update */
      for(x = x_start; x < x_end; x++)
         for(y = 1; y <= YSIZE; y++)
            u[x][y]                                                    =
(uu[x-1][y]+uu[x+1][y]+uu[x][y-1]+uu[x][y+1])/4.0;
```

```
    }

    /* check sum */
    sum = 0.0;
    for(x = x_start; x < x_end; x++)
        for(y = 1; y <= YSIZE; y++)
            sum += (uu[x][y]-u[x][y]);
    MPI_Reduce(&sum,&t_sum,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
    if(myid == 0) printf("sum = %g¥n",t_sum);
}

void initialize()
{
    int x,y;

    /* initalize */
    for(x = 1; x <= XSIZE; x++)
      for(y = 1; y <= YSIZE; y++)
        u[x][y]          =          sin((double)(x-1)/XSIZE*PI)          +
cos((double)(y-1)/YSIZE*PI);

    for(x = 0; x < (XSIZE+2); x++){
        u[x][0] = 0.0;
        u[x][YSIZE+1] = 0.0;
        uu[x][0] = 0.0;
        uu[x][YSIZE+1] = 0.0;
    }

    for(y = 0; y < (YSIZE+2); y++){
        u[0][y] = 0.0;
        u[XSIZE+1][y] = 0.0;
        uu[0][y] = 0.0;
        uu[XSIZE+1][y] = 0.0;
    }
}
```

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/*
 * PSC2001 sample program (OpenMP version)
 */
/* prototypes */
void psc2001_get_config(int problem_no,
                    int * size_p, int * n_steps_p, double * dt_p);
void psc2001_get_data(int problem_no, double * a);
void psc2001_verify(int problem_no, double * a);

/* bug fix 4/13 */
double round(double x){
    if(x < 0.0) return -floor(-x+0.5);
    else return floor(x+0.5);
}

typedef struct my_particle {
    double m;
    double x,y,z;
    double vx,vy,vz;
    double ax,ay,az;
} particle;

double DT;
int n_steps;
int n_particles;
particle *particles;

void do_step();

int main(int argc, char ** argv)
{
    int problem_no,it,i;
    double *a,*ap;
    particle *p;

    if(argc != 2){
        fprintf(stderr,"bad args...¥n");
        exit(1);
    }
    problem_no = atoi(argv[1]);

    psc2001_get_config(problem_no,&n_particles,&n_steps,&DT);

    printf("problem_no=%d: n_particle=%d, n_steps=%d, DT=%e¥n",
          problem_no, n_particles, n_steps, DT);
```

```
    a = (double *)malloc(sizeof(double)*7*n_particles);
    particles = (particle *)malloc(sizeof(particle) * n_particles);
    if(a == NULL || particles == NULL){
        fprintf(stderr,"no memory...¥n");
        exit(1);
    }

    psc2001_get_data(problem_no,a);

    ap = a;
    for(i = 0; i < n_particles; i++) {
        p = &particles[i];
        p->m = *ap++;
        p->x = *ap++;
        p->y = *ap++;
        p->z = *ap++;
        p->vx = *ap++;
        p->vy = *ap++;
        p->vz = *ap++;
    }

#pragma omp parallel private(it)
{
    for(it = 0; it < n_steps; it++){
        do_step();
    }
}

    ap = a;
    for(i = 0; i < n_particles; i++) {
        p = &particles[i];
        *ap++ = p->m;
        *ap++ = p->x;
        *ap++ = p->y;
        *ap++ = p->z;
        *ap++ = p->vx;
        *ap++ = p->vy;
        *ap++ = p->vz;
    }

    psc2001_verify(problem_no,(double *)a);

    exit(0);
}

void do_step()
{
    int i,j;
    double a2 = 256.0;
    double b2 = 1024.0;
    double ax,ay,az,dx,dy,dz,X,f;
    particle *p,*q;

#pragma omp for
    for(i = 0; i < n_particles; i++) {
        p = &particles[i];
        ax = 0.0;
        ay = 0.0;
        az = 0.0;
        for(j = 0; j < n_particles; j++){
            if(i == j) continue;
            q = &particles[j];
            dx = p->x - q->x;
            dy = p->y - q->y;
            dz = p->z - q->z;

            dx = round(dx);
            dy = round(dy);
            dz = round(dz);

            X = dx * dx + dy * dy + dz * dz;
            if (X < b2) {
                f = q->m * (X - a2) * (X - b2);
                ax += f * dx;
                ay += f * dy;
                az += f * dz;
            }
        }
        p->ax = ax;
        p->ay = ay;
        p->az = az;
    }

#pragma omp for
    for(i = 0; i < n_particles; i++){
        p = &particles[i];
        p->x += p->vx * DT;
        p->y += p->vy * DT;
        p->z += p->vz * DT;
        p->vx += p->ax * DT;
        p->vy += p->ay * DT;
        p->vz += p->az * DT;
    }
}
```

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include "mpi.h"

#define MAX_PROC 64

/*
 * PSC2001 sample program (MPI version)
 */
/* prototypes */
void psc2001_get_config(int problem_no,
                    int * size_p, int * n_steps_p, double * dt_p);
void psc2001_get_data(int problem_no, double * a);
void psc2001_verify(int problem_no, double * a);

/* bug fix 4/13 */
double round(double x){
   if(x < 0.0) return -floor(-x+0.5);
   else return floor(x+0.5);
}

double *m;
double *x,*y,*z;
double *vx,*vy,*vz;
double *ax,*ay,*az;

double DT;
int n_steps;
int n_particles;

void do_step();

int numprocs;
int myid;
int my_start,my_end,my_count;

int starts[MAX_PROC];
int counts[MAX_PROC];

int main(int argc, char ** argv)
{
   int problem_no,it,i,j,s;
   double *a,*ap;

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

   if(myid == 0){
      if(argc != 2){
         fprintf(stderr,"bad args...¥n");
         exit(1);
      }
      problem_no = atoi(argv[1]);

      psc2001_get_config(problem_no,&n_particles,&n_steps,&DT);

      printf("problem_no=%d: n_particle=%d, n_steps=%d, DT=%e¥n",
            problem_no, n_particles, n_steps, DT);
   }
   MPI_Bcast(&n_particles, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&n_steps, 1, MPI_INT, 0, MPI_COMM_WORLD);
   MPI_Bcast(&DT, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

   s = (n_particles+numprocs-1)/numprocs;
   j = 0;
   for(i = 0; i < numprocs; i++){
      starts[i] = j;
      j += s;
      if(j > n_particles) j = n_particles;
      counts[i] = j - starts[i];
   }
   my_start = starts[myid];
   my_count = counts[myid];
   my_end = my_start + my_count;

   a = (double *)malloc(sizeof(double)*7*n_particles);
   x = (double *)malloc(sizeof(double)*n_particles);
   m = (double *)malloc(sizeof(double)*n_particles);
   y = (double *)malloc(sizeof(double)*n_particles);
   z = (double *)malloc(sizeof(double)*n_particles);
   vx = (double *)malloc(sizeof(double)*n_particles);
   vy = (double *)malloc(sizeof(double)*n_particles);
   vz = (double *)malloc(sizeof(double)*n_particles);
   ax = (double *)malloc(sizeof(double)*n_particles);
   ay = (double *)malloc(sizeof(double)*n_particles);
   az = (double *)malloc(sizeof(double)*n_particles);

   if(myid == 0){
      psc2001_get_data(problem_no,a);
      ap = a;
      for(i = 0; i < n_particles; i++) {
         m[i] = *ap++;
         x[i] = *ap++;
         y[i] = *ap++;
         z[i] = *ap++;
         vx[i] = *ap++;
         vy[i] = *ap++;
         vz[i] = *ap++;
```

```
        }
    }

    MPI_Bcast(m, n_particles, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(x, n_particles, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(y, n_particles, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(z, n_particles, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(vx, n_particles, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(vy, n_particles, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(vz, n_particles, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    for(it = 0; it < n_steps; it++){
        do_step();
    }

    /* gather velocity */
    MPI_Gatherv(vx+my_start,my_count,MPI_DOUBLE,
              vx,counts,starts,MPI_DOUBLE,0,MPI_COMM_WORLD);
    MPI_Gatherv(vy+my_start,my_count,MPI_DOUBLE,
              vy,counts,starts,MPI_DOUBLE,0,MPI_COMM_WORLD);
    MPI_Gatherv(vz+my_start,my_count,MPI_DOUBLE,
              vz,counts,starts,MPI_DOUBLE,0,MPI_COMM_WORLD);

    if(myid == 0){
        ap = a;
        for(i = 0; i < n_particles; i++){
            *ap++ = m[i];    /* not used */
            *ap++ = x[i];
            *ap++ = y[i];
            *ap++ = z[i];
            *ap++ = vx[i];
            *ap++ = vy[i];
            *ap++ = vz[i];
        }
        psc2001_verify(problem_no,(double *)a);
    }

    MPI_Finalize();

    exit(0);
}

void do_step()
{
    int i,j;
    double a2 = 256.0;
    double b2 = 1024.0;
    double ax0,ay0,az0,dx,dy,dz,X,f;

    for(i = my_start; i < my_end; i++) {
        ax0 = 0.0;
        ay0 = 0.0;
        az0 = 0.0;
        for(j = 0; j < n_particles; j++){
            if(i == j) continue;
            dx = x[i] - x[j];
            dy = y[i] - y[j];
            dz = z[i] - z[j];

            dx = round(dx);
            dy = round(dy);
            dz = round(dz);

            X = dx * dx + dy * dy + dz * dz;
            if (X < b2) {
                f = m[j] * (X - a2) * (X - b2);
                ax0 += f * dx;
                ay0 += f * dy;
                az0 += f * dz;
            }
        }
        ax[i] = ax0;
        ay[i] = ay0;
        az[i] = az0;
    }

    for(i = my_start; i < my_end; i++){
        x[i] += vx[i] * DT;
        y[i] += vy[i] * DT;
        z[i] += vz[i] * DT;
        vx[i] += ax[i] * DT;
        vy[i] += ay[i] * DT;
        vz[i] += az[i] * DT;
    }

    /* update all position */
    MPI_Allgatherv(x+my_start,my_count,MPI_DOUBLE,
                x,counts,starts,MPI_DOUBLE,MPI_COMM_WORLD);
    MPI_Allgatherv(y+my_start,my_count,MPI_DOUBLE,
                y,counts,starts,MPI_DOUBLE,MPI_COMM_WORLD);
    MPI_Allgatherv(z+my_start,my_count,MPI_DOUBLE,
                z,counts,starts,MPI_DOUBLE,MPI_COMM_WORLD);
}
```