```
/**************
 * Includes
 **************/

#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

/****************
 * Defines
 ****************/

#define PARTICLES 1000000

/*****************
 * Data Structures
 *****************/

typedef struct
{
      float x, y, z, w;
} vec4D;

/***************
 * Global Variables
 ***************/

// particle position and velocity
vec4D pos[PARTICLES];
vec4D vel[PARTICLES];

// current force being applied to the particles
vec4D force;

// inverse mass of the particles
float inv_mass[PARTICLES];

// step in time
float dt = 0.01f;

double temperature;

/*********************
 * Main Execution
 *********************/

int main( int argc, char *argv[] )
{
   int i, j, k;
   int numprocs, myid;
   float time;
   float dt_inv_mass;
   double mytemp = 0.0;
   int end_of_time = 0;
   double starttime, endtime;

   MPI_Init(&argc, &argv);

   // numprocs must evenly divide into 100
   MPI_Comm_size(MPI_COMM_WORLD, &numprocs);

   MPI_Comm_rank(MPI_COMM_WORLD, &myid);

   // No error or type checking is done!
   if (myid == 0)
   {
      printf("I am 0\n");

      printf("Enter end of time: ");
      scanf("%d", &end_of_time);
      printf("You entered: %d\n", end_of_time);
   }

   starttime = MPI_Wtime();
```

```
MPI_Bcast(&end_of_time, 1, MPI_INT, 0, MPI_COMM_WORLD);

// Initialize velocities with the random numbers between 0 and 1
for (i = myid; i < PARTICLES; i += numprocs)
{
   vel[i].x = drand48();
   vel[i].y = drand48();
   vel[i].z = drand48();
}

if (myid == 0)
{
   printf("Particle Velocities Initialized\n");
}

// Initialize mass
for (i = myid; i < PARTICLES; i += numprocs)
{
   inv_mass[i] = 1.0;
}

if (myid == 0)
{
   printf("Inv_Masses Initialized\n");
}

// Initialize the x, y, z force components of each particle with 0.01
force.x = 0.01;
force.y = 0.01;
force.z = 0.01;

if (myid == 0)
{
   printf("Force Initialized\n");
}

// Initialize Positions
for (i = myid; i < 100; i += numprocs)
{
   for (j = myid; j < 100; j += numprocs)
   {
      for (k = myid; k < 100; k += numprocs)
      {
         pos[i*(100*100) + j*100 + k].x = i * 0.1;
         pos[i*(100*100) + j*100 + k].y = j * 0.1;
         pos[i*(100*100) + j*100 + k].z = k * 0.1;
      }
   }
}

if (myid == 0)
{
   printf("Particle Positions Initialized\n");
}

// For each step in time
for (time = 0; time < end_of_time; time += dt)
{
   mytemp = 0;
   //temperature = 0;

   // For each particle
   // each processor handles 1/4th the particles per timeslice
   for (i = myid; i < PARTICLES; i += numprocs)
   {
      // Compute the new position and velocity
      // as acted upon by the force f.
      pos[i].x = vel[i].x * dt + pos[i].x;
      pos[i].y = vel[i].y * dt + pos[i].y;
      pos[i].z = vel[i].z * dt + pos[i].z;

      dt_inv_mass = dt * inv_mass[i];

      vel[i].x = dt_inv_mass * force.x + vel[i].x;
      vel[i].y = dt_inv_mass * force.y + vel[i].y;
      vel[i].z = dt_inv_mass * force.z + vel[i].z;
```

```
         //updated line
         mytemp += (1.0/PARTICLES) * (vel[i].x * vel[i].x
                                        + vel[i].y * vel[i].y
                                        + vel[i].z * vel[i].z);
      }

      // synchronize (sum up temp for timeslice)
      MPI_Reduce(&mytemp, &temperature, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
   }

   if (myid == 0)
   {
      printf("Finishing: %f\n", temperature);
   }

   endtime = MPI_Wtime();

   printf("Time: %f\n", endtime - starttime);

   MPI_Finalize();

   return (0);
}
```