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

#define PARTICLES 1000000

typedef struct { float x, y, z, w; } vec4D;
vec4D pos[PARTICLES]; // particle positions
vec4D vel[PARTICLES]; // particle velocities
vec4D force; // current force being applied to the particles
float inv_mass[PARTICLES]; // inverse mass of the particles
float dt = 0.01f; // step in time
float temperature;

int main(int argc, char *argv[])
{
	float fTime; float dt_inv_mass;
	float endOfTime; int myid, numprocs;
	int i, j, k;
	double starttime, endtime; //sj
	int blocksize; //sj

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

	if (myid == 0)
	{
		printf("Please Enter end of time for integration: ");
		scanf("%f", &endOfTime);
	}

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

```
//The same initialization on all processors
            force.z = 0.01;
            force.x = 0.01;
            force.y = 0.01;
 //Here i, j, k are for each dimension of 3D cube (100x100x100 = 1000000)
        for (i=0;i<100;i++ ) {
         for (j=0;j<100;j++ ) {
          for (k=0;k<100; k++ ) {
                  pos[ (int) ( (i) * (100*100) + j*100 +k)].x= i*0.1;
                  pos[ (int) ( (i) * (100*100) + j*100 +k)].y= j*0.1;
                  pos[ (int) ( (i) * (100*100) + j*100 +k)].z= k*0.1;
          }
         }
        }

    for(i = 0; i < PARTICLES; i ++)
    {
          vel[i].x = rand() / (float)RAND_MAX;
          vel[i].y = rand() / (float)RAND_MAX;
          vel[i].z = rand() / (float)RAND_MAX;
          inv_mass[i] = 1.0f;
    }

 blocksize=PARTICLES/numprocs;

 //time the parallel computing part
 starttime = MPI_Wtime();
```

```
  for (fTime=0; fTime < endOfTime; fTime += dt)
  {
       mytemperature = 0;
// here i is from 0 to PARTICLES in 1D but increment by numproc. That is bad for performance
  for (i = myid; i < PARTICLES; i = i + numprocs)
// decompose into 4 blocks and i increments contiguously  which reduces memory access time
//   for (i = myid*blocksize; i < (myid+1)*blocksize; i++)
   {
   // 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;
     mytemperature += (1.0/PARTICLES)* (vel[i].x*vel[i].x + vel[i].y*vel[i].y +
                                vel[i].z*vel[i].z);
   }
  }
       //this is the temperature at the final time step
       MPI_Reduce(&mytemperature, &temperature, 1, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);

       endtime=MPI_Wtime();

       if(myid == 0)
       {
             printf("Total Seconds: %lf \n", endtime - starttime);
             printf("Temperature is: %f\n", temperature);
       }
       MPI_Finalize(); //before return (0)
       return (0);
}
```