Consider a program designed to solve for the motion of multiple stars in a large cluster.

The objects all follow the equations of motion

Where is the mass of the star at position , G is Newton’s gravitational constant, , and , and and are the acceleration and velocity.

The following code solves for the motion of a random cluster.

Modify the code to run in parallel using MPI, and describe your performance improvement on up to 4 nodes with 8 cores each on puma.kean.edu. Submit your code as well as timings calculated using the MPI\_Wtime() command.

|  |
| --- |
| #include <stdio.h>  #include <math.h>  #include <stdlib.h>  void calc\_accelerations(int n, double \* ax, double \* ay, double \* az,  double \*x, double \*y, double \*z, double mG) {  int i,j;  double k;  double dx,dy,dz,dr,dr2,dr3i;  // calculate accelerations  for(i=0;i<n;i++) {  ax[i] = 0.0;  ay[i] = 0.0;  az[i] = 0.0;  }  for(i=0;i<n;i++) {  for(j=i+1;j<n;j++) {  dx = x[j]-x[i];  dy = y[j]-y[i];  dz = z[j]-z[i];  dr2 = dx\*dx+dy\*dy+dz\*dz;  dr=sqrt(dr2);  dr3i=1.0/(dr2\*dr);  k=mG\*dr3i;  ax[i]+=dx\*k;  ay[i]+=dy\*k;  az[i]+=dz\*k;  ax[j]-=dx\*k;  ay[j]-=dy\*k;  az[j]-=dz\*k;  }  }  }  void leapfrog(int n,  double \* x,double \*y,double \*z,  double \*vx,double \*vy,double \*vz,  double \*ax,double \*ay,double \*az, double time\_step) {  int i;  // leapfrog forward, update velocities then positions  for(i=0;i<n;i++) {  vx[i] += ax[i]\*time\_step;  vy[i] += ay[i]\*time\_step;  vz[i] += az[i]\*time\_step;  }  for(i=0;i<n;i++) {  x[i] += vx[i]\*time\_step;  y[i] += vy[i]\*time\_step;  z[i] += vz[i]\*time\_step;  }  }  int main(int argc, char \*\* argv) {  int n=200;  double \* x, \*y, \*z;  double \* vx, \*vy, \*vz;  double \* ax, \*ay, \*az;  double time=0.0;  double time\_final = 50.0;  double time\_step = 0.01;  int i,j;  double mG=0.001;  // allocate memory  x = (double \*)malloc(sizeof(double)\*n);  y = (double \*)malloc(sizeof(double)\*n);  z = (double \*)malloc(sizeof(double)\*n);  vx = (double \*)malloc(sizeof(double)\*n);  vy = (double \*)malloc(sizeof(double)\*n);  vz = (double \*)malloc(sizeof(double)\*n);  ax = (double \*)malloc(sizeof(double)\*n);  ay = (double \*)malloc(sizeof(double)\*n);  az = (double \*)malloc(sizeof(double)\*n);    // initialize memory  for(i=0;i<n;i++) {  x[i] = (double)rand()/(double)RAND\_MAX;  y[i] = (double)rand()/(double)RAND\_MAX;  z[i] = (double)rand()/(double)RAND\_MAX;  vx[i] = 0.0;  vy[i] = 0.0;  vz[i] = 0.0;  }  //time loop  while(time<time\_final) {  calc\_accelerations(n,ax,ay,az,x,y,z,mG);  leapfrog(n,x,y,z,vx,vy,vz,  ax,ay,az,time\_step);  // update time  time += time\_step;  }  // free memory  free(x);  free(y);  free(z);  free(vx);  free(vy);  free(vz);  free(ax);  free(ay);  free(az);  return 0;  } |