**C语言项目报告**

**项目名称：\_**两种原子体系的动力学模拟**\_\_\_\_\_\_\_\_\_**

**项目成员：\_**吕佳潼**\_\_\_\_\_\_\_\_\_\_**

**填写日期：\_**2021.7.8**\_\_\_\_\_\_\_\_\_\_**

1. **摘要（Abstract）**

本项目是基于牛顿的经典力学模型，利用C语言编写了多粒子碰撞动力学模拟程序。程序实现了两种不同粒子在一定体积内短时间运动情况的模拟，并利用Ovito软件实现多粒子运动轨迹的可视化。

1. **问题描述（Problem Statement）**

大量分子在一定空间内的运动是混沌的，无法利用常规实验手段观测。由于分子运动可由经典牛顿运动方程和运动定律描述，因此可以利用分子动力学模拟分子在微观下的动力学行为。

1. **组内分工（Group Division）**

小组成员：吕佳潼12200404

1. **分析（Analysis）**

推导运动学方程，并在极短的时间间隔内计算出粒子的运动状态，最后循环过程，即可得到一段时间内分子的运动轨迹。

1. **设计（Design）**

直接在任何C语言环境中运行程序，再将输出的轨迹文件拖入Ovito软件中，即可实现分子动力学的可视化。

1. **实施（Implementation）**

具体代码：

// Molecular dynamics simulation linear code for binary Lennard-Jones liquid under NVE ensemble

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#include <math.h>

#include <sys/time.h>

//#include <cuda\_runtime.h>

// periodic boundary condition

struct float3 // structure to hold information

{

float3() : x(0.0), y(0.0), z(0.0){}

float3(float x0, float y0, float z0) : x(x0), y(y0), z(z0){}

float x;

float y;

float z;

};

struct float4 // structure to hold information

{

float4() : x(0.0), y(0.0), z(0.0), w(0.0){}

float4(float x0, float y0, float z0, float w0) : x(x0), y(y0), z(z0), w(w0){}

float x;

float y;

float z;

float w;

};

float pbc(float x, float box\_len) // implement periodic bondary condition

{

float box\_half = box\_len \* 0.5;

if (x > box\_half) x -= box\_len;

else if (x < -box\_half) x += box\_len;

return x;

}

// randome number generator [0.0-1.0)

float R2S()

{

int ran = rand();

float fran = (float)ran/(float)RAND\_MAX;

return fran;

}

// initially generate the position and mass of particles

void init(unsigned int np, float4\* r, float4\* v, float3 box, float min\_dis)

{

for (unsigned int i=0; i<np; i++)

{

bool find\_pos = false;

float4 ri;

while(!find\_pos)

{

ri.x = ( R2S() - 0.5 ) \* box.x;

ri.y = ( R2S() - 0.5 ) \* box.y;

ri.z = ( R2S() - 0.5 ) \* box.z;

find\_pos = true;

for(unsigned int j=0; j<i; j++)

{

float dx = pbc(ri.x - r[j].x, box.x);

float dy = pbc(ri.y - r[j].y, box.y);

float dz = pbc(ri.z - r[j].z, box.z);

float r = sqrt(dx\*dx + dy\*dy + dz\*dz);

if(r<min\_dis) // a minimum safe distance to avoid the overlap of LJ particles

{

find\_pos = false;

break;

}

}

}

if(R2S()>0.5) // randomly generate the type of particle, 1.0 represent type A and 2.0 represent type B

ri.w =1.0;

else

ri.w =2.0;

r[i] = ri;

v[i].w = 1.0;

}

}

// first step integration of velocity verlet algorithm

void first\_integration(unsigned int np, float dt, float3 box, float4\* r, float4\* v, float4\* f)

{

for (unsigned int i=0; i<np; i++)

{

float4 ri = r[i];

float mass = v[i].w;

v[i].x += 0.5 \* dt \* f[i].x / mass;

v[i].y += 0.5 \* dt \* f[i].y / mass;

v[i].z += 0.5 \* dt \* f[i].z / mass;

ri.x += dt \* v[i].x;

ri.y += dt \* v[i].y;

ri.z += dt \* v[i].z;

r[i].x = pbc(ri.x, box.x);

r[i].y = pbc(ri.y, box.y);

r[i].z = pbc(ri.z, box.z);

}

}

// non-bonded force calculation

void force\_calculation(unsigned int np, float3 box, float3 epsilon, float3 sigma, float4\* r, float4\* f, float rcut)

{

for(unsigned int i=0; i<np; i++)

{

float4 force = float4(0.0, 0.0, 0.0, 0.0);

for(unsigned int j=0; j<np; j++)

{

/\* particles have no interactions with themselves \*/

if (i==j) continue;

/\* calculated the shortest distance between particle i and j \*/

float dx = pbc(r[i].x - r[j].x, box.x);

float dy = pbc(r[i].y - r[j].y, box.y);

float dz = pbc(r[i].z - r[j].z, box.z);

float type = r[i].w + r[j].w;

float r = sqrt(dx\*dx + dy\*dy + dz\*dz);

/\* compute force and energy if within cutoff \*/

if (r < rcut)

{

float epsilonij, sigmaij;

if(type==2.0) // i=1.0, j=1.0

{

epsilonij = epsilon.x;

sigmaij = sigma.x;

}

else if(type==3.0) // i=1.0, j=2.0; or i=2.0, j=1.0

{

epsilonij = epsilon.y;

sigmaij = sigma.y;

}

else if(type==4.0) // i=2.0, j=2.0

{

epsilonij = epsilon.z;

sigmaij = sigma.z;

}

float ffac = -4.0\*epsilonij\*(-12.0\*pow(sigmaij/r,12.0)/r + 6.0\*pow(sigmaij/r,6.0)/r); // force between particle i and j

float epot = 0.5\*4.0\*epsilonij\*(pow(sigmaij/r,12.0) - pow(sigmaij/r,6.0)); // potential between particle i and j

force.x += ffac\*dx/r;

force.y += ffac\*dy/r;

force.z += ffac\*dz/r;

force.w += epot;

}

}

f[i] = force;

}

}

// second step integration of velocity verlet algorithm

void second\_integration(unsigned int np, float dt, float4\* v, float4\* f)

{

for (unsigned int i=0; i<np; i++)

{

float mass = v[i].w;

v[i].x += 0.5 \* dt \* f[i].x / mass;

v[i].y += 0.5 \* dt \* f[i].y / mass;

v[i].z += 0.5 \* dt \* f[i].z / mass;

}

}

// system information collection for temperature, kinetic energy, potential and total energy

void compute\_info(unsigned int np, float4\* v, float4\* f, float\* info)

{

float ekin=0.0;

float potential;

for (unsigned int i = 0; i < np; i++)

{

float4 vi = v[i];

float mass = vi.w;

ekin += 0.5\*mass\*(vi.x\*vi.x + vi.y\*vi.y + vi.z\*vi.z);

potential += f[i].w;

}

unsigned int nfreedom = 3 \* np - 3;

float temp = 2.0\*ekin/float(nfreedom);

float energy = ekin + potential;

info[0] = temp;

info[1] = potential;

info[2] = energy;

}

// output system information and frame in XYZ formation which can be read by VMD

void output(FILE \*traj, unsigned int step, float\* info, float4\* r, unsigned int np)

{

float temp = info[0];

float potential = info[1];

float energy = info[2];

fprintf(traj,"%d\n step=%d temp=%20.8f pot=%20.8f ener=%20.8f\n", np, step, temp, potential, energy);

for (unsigned int i=0; i<np; i++)

{

float4 ri = r[i];

if (ri.w == 1.0)

fprintf(traj, "A %20.8f %20.8f %20.8f\n", ri.x, ri.y, ri.z);

else if (ri.w == 2.0)

fprintf(traj, "B %20.8f %20.8f %20.8f\n", ri.x, ri.y, ri.z);

}

}

// main function

int main(int argc, char \*\*argv)

{

//running parameters

unsigned int np = 2700; // the number of particles

unsigned int nsteps = 500; // the number of time steps

float dt = 0.001; // integration time step

float rcut = 3.0; // the cutoff radius of interactions

// float temperature = 1.0; // target temperature

unsigned int nprint = 100; // period for data output

timeval start; // start time

timeval end; // end time

float3 box =float3(15.0, 15.0, 15.0); // box size in x, y, and z directions

float3 epsilon = float3(1.0, 0.5, 1.0); // epsilon.x for type 1.0 and 1.0; epsilon.y for type 1.0 and 2.0; epsilon.z for type 1.0 and 2.0

float3 sigma = float3(1.0, 1.0, 1.0); // sigma.x for type 1.0 and 1.0; sigma.y for type 1.0 and 2.0; sigma.z for type 1.0 and 2.0

float min\_dis = sigma.x\*0.9; // the minimum distance between particles for system generation

//memory allocation

float4\* r = (float4 \*)malloc(np\*sizeof(float4)); // rx, ry, rz, type(0, 1, 2 ...)

float4\* v = (float4 \*)malloc(np\*sizeof(float4)); // vx, vy, vz, mass

float4\* f = (float4 \*)malloc(np\*sizeof(float4)); // fx, fy, fz, potential

float\* info = (float \*)malloc(16\*sizeof(float)); // temperature, potential, energy ...

FILE \*traj=fopen("traj.xyz","w"); // trajectory file in XYZ format that can be open by VMD

/\* generate system information \*/

printf("Starting simulation with %d atoms for %d steps.\n", np, nsteps);

printf("Generating system.\n", np, nsteps);

init(np, r, v, box, min\_dis);

gettimeofday(&start,NULL); //get start time

/\* main MD loop \*/

printf("Running simulation.\n", np, nsteps);

for(unsigned int step =0; step <= nsteps; step++) //running simulation loop

{

/\* first integration for velverlet \*/

first\_integration(np, dt, box, r, v, f);

/\* force calculation \*/

force\_calculation(np, box, epsilon, sigma, r, f, rcut);

/\* compute temperature and potential \*/

compute\_info(np, v, f, info);

/\* second integration for velverlet \*/

second\_integration(np, dt, v, f);

/\* write output frames and system information, if requested \*/

if ((step % nprint) == 0)

{

output(traj, step, info, r, np);

printf("time step %d \n", step);

}

}

gettimeofday(&end,NULL); // get end time

long timeusr=(end.tv\_sec-start.tv\_sec)\*1000000+(end.tv\_usec-start.tv\_usec);

printf("time is %ld microseconds\n",timeusr); // the spending time on simulation in microseconds

fclose(traj);

free(r);

free(v);

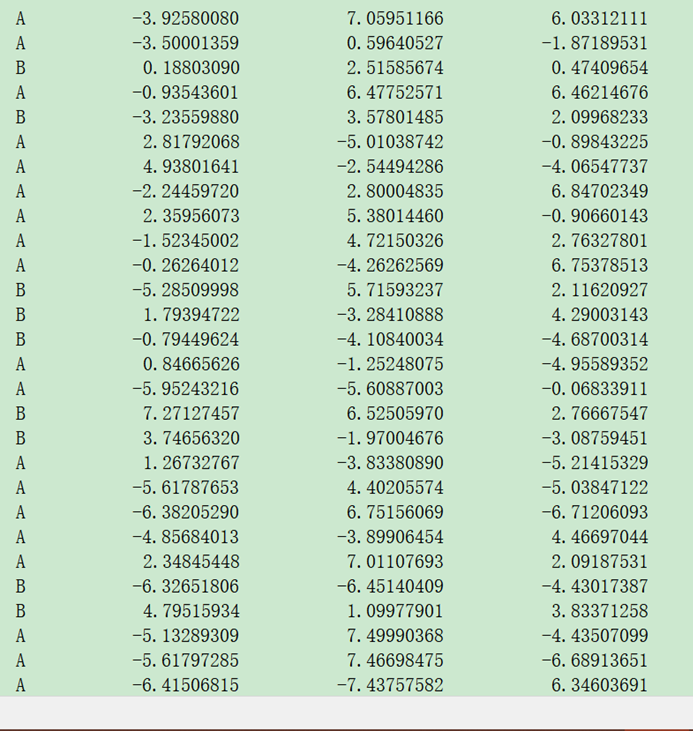
free(f);

return 0;

}

1. **测试（Test）**

部分轨迹展示：

****

可视化结果展示：

