Simulation Tutorials
Lecture 9

Instructor: Takeshi Kawasaki

D3 center, Osaka University

Last update: July 4, 2025

Contents

1. Acceleration of MD Calculations

Objective

- This Lecture focuses on accelerating the most computationally expensive part of MD calculations, which is the force computation.
- Especially when only short-range interactions are considered, it is pointless to compute distances between distant particles that do not interact.
- To omit such unnecessary calculations, we record the IDs of nearby particles (this record is called a list), and consider an algorithm that only calculates interactions between particles recorded in this list.
- The definition of nearby particles can be arbitrary; particles within a slightly extended distance from the potential cutoff can be considered nearby particles.
- This allows us to reuse the same list for a while.
- As time passes and particles move, some may leave the proximity region (or some may enter from outside), at which point the list needs to be updated.
- Below, I will explain how to create and update the list.

1.1. Verlet List Method

Here, we introduce the simplest method for creating a list: the Verlet list method [1, 2].

- First, nearby particles k of particle j are defined as those within a distance of $R_{cut} + \Delta$, as shown in Fig. **1**. Here, R_{cut} **is the potential cutoff length**, and Δ **is the skin (buffer) length**.
- In the basic Verlet list method, we first calculate the distance between all particles and then search for nearby particles k within $R_{cut} + \Delta$ from the base particle j.
- Once the neighbor list is determined, the same list is used for interaction calculations for a while (for n_{list} steps).
- It is necessary to find an optimal combination of Δ and n_{list}.
- A sample program of the Verlet list method is shown in Listing 1.
- The structure of the list array generated by this method is summarized in Listing 2.

1.1. Verlet List Method (2)

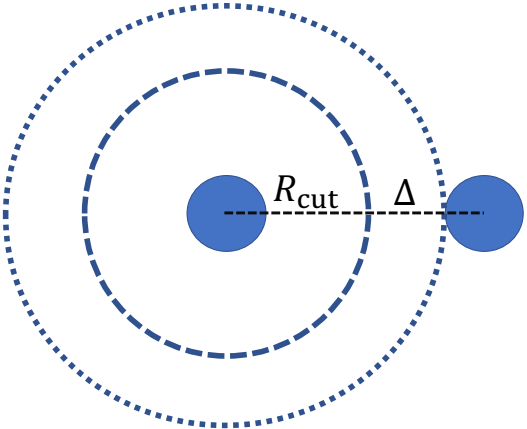


Fig. 1: A list of the nearest particles k to particle j, with R_{cut} as the cutoff length and Δ as the skin size.

1.1. Verlet List Method (3)

List 1: Sample Program (Subroutine)

```
1  #define Np 1024
2  #define Nn 100
3  #define L 40.0
4  #define dim 2
5  #define cut 3.0
6  #define skin 1.0
7
8  void list_verlet(int (*list)[Nn],double (*x)[dim]){
9    double dx,dy,dr2;
10   double thresh=cut+skin;
11   for(int i=0;i<Np;i++)
12     for(int j=0;j<Nn;j++)
13       list[i][j]=0;
14
15   for(int i=0;i<Np;i++)
16     for(int j=0;j<Np;j++){
17       if(j>i){
18         dx=x[i][0]-x[j][0];
19         dy=x[i][1]-x[j][1];
20         dx-=L*floor((dx+0.5*L)/L);
21         dy-=L*floor((dy+0.5*L)/L);
22         dr2=dx*dx+dy*dy;
```

1.1. Verlet List Method (4)

```
23          if(dr2<thresh*thresh){
24            list[i][0]++;
25            list[i][(int)list[i][0]]=j;
26          }
27        }
28      }
29  }
```

The list array will look as follows.

List 2: Structure of the List Array

```
1  list[i][0]: The number of the neighboring particles of particle i
2  list[i][1]: Neighboring particle id of particle i (1st)
3  list[i][2]: Neighboring particle id of particle i (2nd)
4  list[i][3]: Neighboring particle id of particle i (3rd)
5  ....
6  list[i][list[i][0]]: Neighboring particle id of particle i (list[i][0]th)
```

The following is the calculation of forces using the list array.

1.1. Verlet List Method (5)

List 3: Force Calculation Using the List Array

```
1  #define Np 1024
2  #define Nn 100
3  #define L 40.0
4  #define dim 2
5  #define cut 3.0
6  #define skin 1.0
7
8  void calc_force(double (*x)[dim],double (*f)[dim],double *a,double *U,int (*list)[Nn]){
9    double dx,dy,dr2,dUr,w2,w6,w12,aij;
10   double Ucut=1./pow(cut,12);
11   ini_array(f);
12   *U=0;
13   for(int i=0;i<Np;i++)
14     for(int j=1;j<=list[i][0];j++){
15       dx=x[i][0]-x[list[i][j]][0];
16       dy=x[i][1]-x[list[i][j]][1];
17       dx-=L*floor((dx+0.5*L)/L);
18       dy-=L*floor((dy+0.5*L)/L);
19       dr2=dx*dx+dy*dy;
20       if(dr2<cut*cut){
21         aij=0.5*(a[i]+a[list[i][j]]);
22         w2=aij*aij/dr2;
```

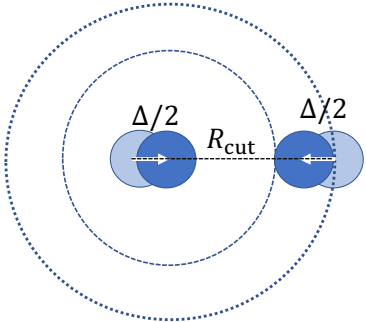
1.1. Verlet List Method (6)

```
23          w6=w2*w2*w2;
24          w12=w6*w6;
25          dUr=-12.*w12/dr2;
26          f[i][0]-=dUr*dx;
27          f[list[i][j]][0]+=dUr*dx;
28          f[i][1]-=dUr*dy;
29          f[list[i][j]][1]+=dUr*dy;
30          *U+=w12-Ucut;
31        }
32      }
33  }
```

1.2 Automatic List Update Method

- By introducing the skin length, the list of neighboring particles can be reused for a while, even if the particle positions change slightly from the initial list creation.
- The timing for updating the list is when any particle moves by **half the skin length**.
- This is because, if two particles move in opposite directions by a distance of $skin/2$, it becomes possible (though rare) for particles to enter the potential cutoff region from outside, making the previous list inaccurate (see Fig. **2**).
- Therefore, the list is reused as long as the maximum displacement of particles is within $skin/2$, and the following algorithm automatically updates the list when this threshold is exceeded.

1.2 Automatic List Update Method (2)



帳簿の前更新時から粒子が最大 $\Delta/2$動くと外部からカットオフ
領域内への粒子の侵入の可能性が生じる（レアではあるが）

Fig. 2: If two particles move in opposite directions by a distance of $\Delta/2$ ($= skin/2$), it becomes possible for them to enter the potential cutoff region from outside, making the previous list inaccurate.

1.2 Automatic List Update Method (3)

List 4: Algorithm for Automatic List Update

```
1  #define Np 1024
2  #define Nn 100
3  #define L 40.0
4  #define dim 2
5  #define cut 3.0
6  #define skin 1.0
7
8  void update(double (*x_update)[dim],double (*x)[dim])
9  {
10   for(int i=0;i<Np;i++)
11     for(int j=0;j<dim;j++)
12       x_update[i][j]=x[i][j];
13 }
14
15 void calc_disp_max(double *disp_max,double (*x)[dim],double (*x_update)[dim])
16 {
17   double dx,dy;
18   double disp;
19   for(int i=0;i<Np;i++){
20     dx=x[i][0]-x_update[i][0];
21     dy=x[i][1]-x_update[i][1];
22     dx-=L*floor((dx+0.5*L)/L);
23     dy-=L*floor((dy+0.5*L)/L);
24     disp = dx*dx+dy*dy;
25     if(disp > *disp_max)
26       *disp_max =disp;
```

1.2 Automatic List Update Method (4)

```
27    }
28  }
29
30  void auto_list_update(double *disp_max,double (*x)[dim],double (*x_update)[dim],int (*list)[Nn]){
31    static int count=0;
32    count++;
33    calc_disp_max(&(*disp_max),x,x_update);
34    if(*disp_max > skin*skin*0.25){
35      list_verlet(list,x);
36      update(x_update,x);
37      //     std::cout<<"update"<<*disp_max<<" "<<count<<std::endl;
38      *disp_max=0.0;
39      count=0;
40    }
41  }
```

- The code for the molecular dynamics method (md.cpp) from Lecture 7, modified with the list method (automatic update), is available in the GitHub repository at **[Link]**.
- The code for the molecular dynamics method (mc.cpp) from Lecture 8, modified with the list method (automatic update), is available in the GitHub repository at **[Link]**.

Note For further acceleration, there is a method called the **cell list method**, which is an extension of the Verlet list method. Depending on the system size, it can be about 1.5 times faster than the Verlet list method. However, the Verlet list method itself provides sufficient speedup (more than 10 times faster than traditional methods), so it is sufficient if you can achieve this level of performance.

Contents

References and Websites

[1] Allen MP, Tildesley DJ (2017) Computer Simulation of Liquids: Second Edition. (Oxford University Press).

[2] Frenkel D, Smit B (2001) Understanding Molecular Simulation: From Algorithms to Applications. (Elsevier).