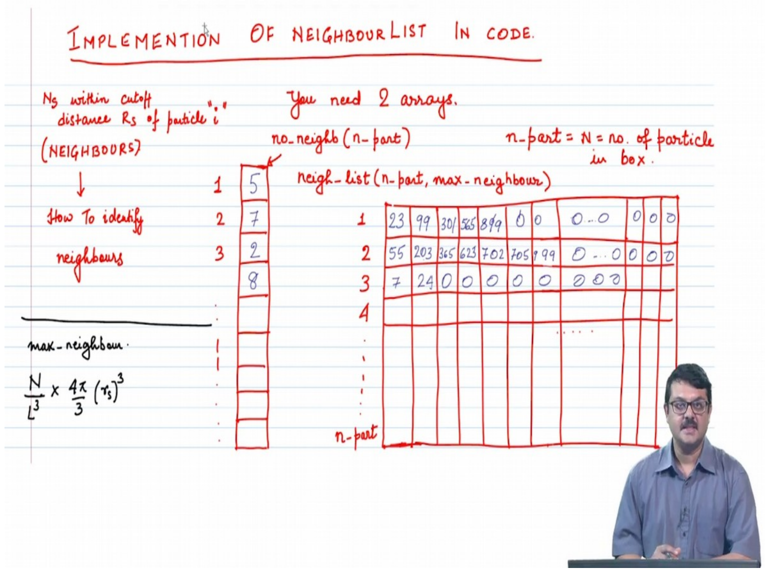
Computational Physics

Dr. Apratim Chatterji
Dr. Prasenjit Ghosh
Department of Physics
Indian Institute of Science Education and Research, Pune

Lecture - 54
Molecular Dynamics Neighbours Lists Part 02

(Refer Slide Time: 00:16)



IMPLEMENTION OF NEIGHBOUR LIST IN CODE.

N_s within cutoff distance R_s of particle "i"
(NEIGHBOURS)
↓
How To identify neighbours

You need 2 arrays.

no_neighb (n-part)

n-part = N = no. of particle in box.

neigh-list (n-part, max-neighbour)

	no_neighb
1	5
2	7
3	2
	8
⋮	

1	23	99	301	565	899	0	0	0 ... 0	0	0	0
2	55	203	365	623	702	705	999	0 ... 0	0	0	0
3	7	24	0	0	0	0	0	0 0 0			
4											
⋮								.....			
n-part											

max-neighbour

$$\frac{N}{L^3} \times \frac{4\pi}{3} (r_s)^3$$

Now let us discuss the implementation of neighbor list in the code. So, what do we have to identify? We have to identify the neighbors, what are neighbors? So, all the particles the N s particles which are within distance R s of particle i. So, they are the neighbors. So, we have to identify them and store them in an array. Now for the neighbor list to make a neighbor list you need actually 2 arrays. One array will store for each particle i. So, i runs from 1 to n part so, the length of this array will be 'npart'.

So, you initialize an array suppose no neighbor number of neighbors of length n part, where n part is essentially the number of particles in the box which we call n. When we discuss, but in the code you should have a bigger name like n part n number of particles write n; n is of 10 used as a dummy index in do loops, right. So, that is not a good name.

So, that is why in the code you could write 'npart' and that is equal to N the number of particles in the box, and you have to have an array suppose number of neighbors which says for each particle; particle 1 2 3 how many neighbors does particle 1 have. Suppose if particle number has 5 neighbors, particle number 2 has 7 neighbors, particle number 3 suppose has 2 neighbors which means you have 2 particles suppose particle number 3 is i is 3, suppose this i is 3.

And then particle number 3 has 2 particles with an distance R s; so, if you draw around the position of particle number 3 a sphere of radius R s then the number of particles neighboring particles which are within distance R s of the position of 3 is 2. And that is what 5 means it means there are 5 particles. So, N s is equal to 5 for when i equal to; when i equal to 1, right. So, you need 1 array number of neighbors which stores the number of neighbors of each particle, number of particles in the system is n part.

So, that why its length is n part and then the index of the neighbors is stored in another array called neigh list neigh for like neighbor list neigh list which will be a 2 dimensional array of length n part ok, the number for each particle you have to identify which other particles are it is neighbors. So, n part one side so, on one side it will be n part and on the other side it is max number of neighbors, what is max neighbor? It is the maximum number of neighbours that any particle could have, ok.

Why max neighbor? Now suppose you have Lennard-Jones interaction right at a particular temperature. So, there are attractive interactions so, suppose at a certain point in space you form a cluster of particles; cluster of Lennard-Jones particles. So, locally it is a very dense system some other part it is relatively less dense, ok. So, there can be density fluctuations because of interactions not that the entire box will be uniformly filled with Lennard-Jones particles right, because there would be attractions which could cluster particles.

So, you need an estimate of what could be the maximum number of neighbours right. Now suppose you have an estimate and how to estimate we will discuss this soon. Then basically you say particle number 1 has 5 neighbors which might be say 23 99 301 899. So, these are the 5 neighbors and rest of the array will be 0, right.

Now particle number 2 has suppose 7 neighbors; so, neigh list this array which is a 2 dimensional array, particle number 2 will have 7 neighbors say and what are the indices of those neighbors the identification of the neighbors suppose particle number 55 203 365 623 and so on so forth. So, there are 7 neighbors rest of the array is 0, now suppose particle number 3 has 2 neighbors say me they may be 7 and 24 say. Rest of the row is full of 0s right and so on so forth for each particle.

So, for each particle you basically save the number of neighbors in this array and you use this array, when you have to calculate force because now you have a smaller list of neighbors for each particle. So, for each particle you do not have to check over N number of particles in the box. So, you now just have to check over only this small set of particles N s; N s for each of these particles and you calculate the distance and if the distance is less than r c.

So, this; so, basically N s here would be 5 7 2 8 for a different values of i. And now you calculate so, over the next 100 iterations you calculate that how many of these particles are within; so, how many of these neighbours for a particular particle i as within r c and then you calculate the forces. What are you saving you are not checking over all the particles and every 100 iterations since the particles would have moved.

So, if suppose particle number 3 and so, around particle number 3, 7 is within r c at time t equal to 0, after the particles have moved 7 after say 70 iterations it is possible that particle number 7 is no longer within a distance of r c of particle number 3 maybe particle number 24 becomes, at a distance less than r c from the current position of particle number 3, right. And after 100 iterations when you update neighbor list of course, this all these will be rewrit10 and you have to write the current after 100 iterations the current at that time the neighbors you have to make a list of neighbors which will be the neighbors of each of these particles, ok. So, that is the broad idea, ok.

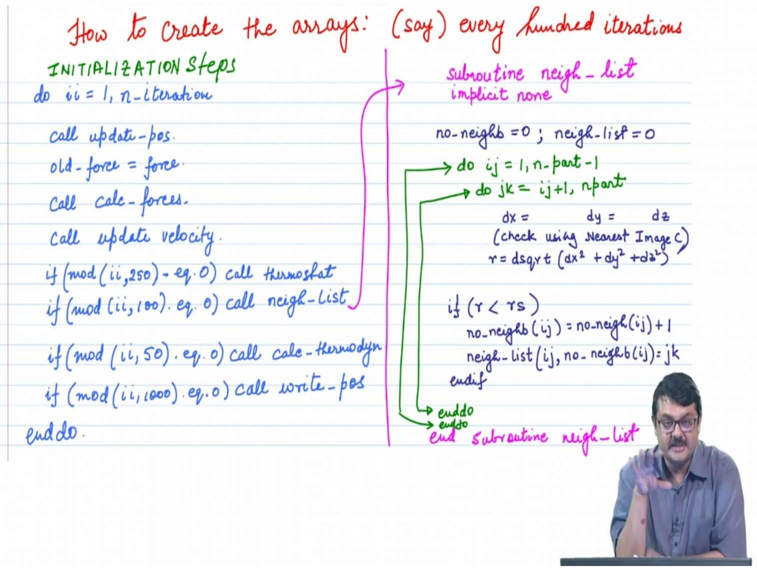
So, how do you estimate max neighbor, right. Now so that is discussed here. So, how would you have an estimate of max neighbor? Basically N by L cube; N is the total number of particles in the box and L cube is the volume of the box. So, N by L cube is essentially the number density, in an unit volume how many particles do you have that is what is number density right N by L cube. And if you multiply it by M, then you would have the actual density right mass of that entire system divided by volume that is the density N by L cube is the number density.

N by L cube is the number density into 4 pi by 3 r s cube is the average number of particles you would expect in a sphere of radius r s, right. But, as I said main due to interactions local density within a certain part of the box would be much higher it could be 3 times or 4 times that of the average density right. And some other part of the box could have lower density. So, you would typically choose max neighbor max inverse so, this quantity as 4 or 5 times this number.

So, this is the total number of particles within this box you can choose it 4 or 5 times more. And I mean even if there is some in homogeneity you would not expect larger deviations, but that is not impossible and if you indeed have a larger number of neighbors. Then, you estimated initially by max neighbor then when you calculate neighbor list the program will show an error and then you must check whether basically for a particular particle in the neighbor list. The number of neighbors that you have for that is for that particle increase is more than max neighbor then you have to basically change the value of max neighbor to a higher value.

There is no point in giving it too higher value because then this array will become too large and if you have to handle large arrays the code becomes slow, because it takes time for we will discuss this later; why the code becomes slow if you have large arrays. But of course, if you have n part particles then you do have some large arrays like position velocity which is 3 into n-part, velocities 3 into n-part then the force array and the old force array each of these are 3 into n-part. But here this one which is a two-dimensional array, this typically of10 the largest array in the standard m d simulation because of the large value of max and neighbor, ok.

(Refer Slide Time: 10:46)



How to create the arrays: (say) every hundred iterations

```
INITIALIZATION steps
do ii = 1, n-iteration

   call update-pos.
   old-force = force.
   call calc-forces.
   call update velocity.
   if (mod(ii,250).eq.0) call thermostat
   if (mod(ii,100).eq.0) call neigh-List

   if (mod(ii,50).eq.0) call calc-thermodyn
   if (mod(ii,1000).eq.0) call write-pos
enddo.
```

```
subroutine neigh-list
implicit none

no-neighb = 0 ; neigh-list = 0
do ij = 1, n-part-1
  do jk = ij+1, npart
     dx =       dy =      dz
     (check using Nearest Image C)
     r = dsqrt (dx² + dy² + dz²)

  if (r < rs)
     no_neighb(ij) = no-neigh(ij) + 1
     neigh-list(ij, no-neighb(ij)) = jk
  endif
  enddo
enddo
end subroutine neigh-list
```

So, that is basically the broad idea how neighbor lists are generated. Now, let us go into the details and discuss even in even greater detail the algorithm how when you calculate it in the code what changes do you have to make from whatever we have discussed till previously.

So, now, we are trying to increase the efficiency of the code and believe me without neighbor list you can hardly do any molecular dynamics simulations as soon as the number of particles goes above 1000, you need neighbor list, right. And typically you do molecular dynamics with 10000 15000 depends upon the complexity of the interaction the length of the iteration. So, these decide how many particles you have you can even have up to million particles or 10 million particles interacting with each other, it depends upon your computational resources, right.

Now, let us talk about the details of how to create an neighbor list as soon as you have more than 1000 particles neighbor list becomes imperative. And suppose you have a you are calling the neighbor list you are updating their neighbor list every 100 iterations right. So, where at what point in the code should you call neighbor list? So, here I have given a schematic the flow chart or the schematic of the code.

So, when you have your molecular dynamics code at the beginning of course, you have your initialization, you would define all the variables, and then initialize position, initialize velocity, calc force. And this is the main body of the code where you have basically do i i being a dummy variable, i i equal to 1 to number of iterations depends how long you want to run the code which in turn depends upon what quantity you are calculating, then what is there in a standard m d, you have called update pause means updates the position. Then you stored the forces into an array called the old force, then you calculate the new force in this step right and then you update the velocity.

So, this is your main molecular dynamics code: update position, calculate new forces, update velocity, right. Then what do you do? Suppose you want to call the thermostat right, you call the thermostat suppose every 250 iterations. Suppose I mean you can call at 100 iterations you can call at 50 iterations, but just for the sake of it. Suppose you want to call the thermostat every 250 iterations then you write if mod i i equal to 250 equal to 0; mod essentially gives the remainder when you divide i i by 250. And if that remainder is equal to 0, then call thermostat which means the thermostat will be called every 250 iterations. If mod i i comma 100 equal to 0 call neighbor list, called the neighbor list every 100 iterations, right.

Then suppose every 50 iterations you want to call the subroutine which will calculate your statistical quantities from a particular microstate. Of course, you have to calculate average over these statistical quantities over many microstates to get a thermodynamic average. So, you could be calling that every 50 iterations say and suppose every 1000 iterations here in this step you could call a subroutine which will write down the positions of all the particles on the computer.

So, if you have the position of all the particles every 1000 iterations you can essentially join them up and make a movie of your moving particles. You can visualize how your particles are moving whether they are form clusters, whether they are breaking up, whether they are moving, whatever they if you want to see a movie that is how you make it, ok.

So, but now we are discussing neighbor lists and basically every 100 steps iterations you are calling the neighbor list. What will be there in the subroutine neighbor lists? Of course, you have to start with subroutine neigh list, neigh for neighbor underscore list implicit none.

And of course, this is the end of the code end subroutine neigh list. Here you have to define all your variables whatever internal variables you need, but most importantly for the neighbor list what you need to define is number of neighbors this entire array right which stores how many neighbors each particle i has 1 2 3 4. How many neighbors each of these particles has that you set to 0 and then array in neigh list which says the name of the subroutine and this array has become the same. Just so, what you could do is just call this suppose neigh list 1 and this is neigh list 1 never mind.

So, this array neigh list you have to set it to 0. So, in this array you store the indices the number the identification of the neighbors for each particle, right. So, that is set to 0 then the loop goes like this suppose do i j is equal to 1 2 and part minus 1. So, basically nearly all the particles do j k equal to i j plus 1, i j is this i j plus just 1 second, i j plus 1 comma n part. So, that you are checking over each pair of particles in the system when you are calculating neighbor list you have to check over each pair of particles in the system.

And now basically i j is ones particle, j k is the index of another particle you calculate the distance displacement d x difference in positions in the x direction, in the y direction in the z direction, use nearest image convention to find out the smallest distance in each direction. Then calculate r the distance the scalar distance between two particles which is nothing, but d x square plus d y square plus d z square, d square root before that. And if this r, if the distance between particle i j and if particle i j and j k is less than r s, then you say ok, the number of neighbors of i j is incremented by 1 and this step.

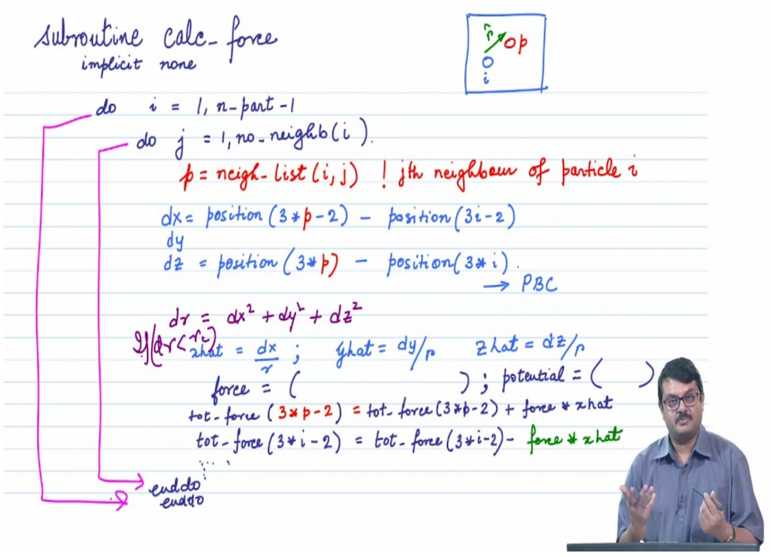
So, number of neighbors of i j is number of neighbors i j plus 1. So, previously if it has 0 number of neighbors, then it says that it has now 1 neighbor. Now as j k keeps on changing and for all the values of j k for which r is less than r s, this the number of neighbors will be incremented 1 2 3 4 5 6 up till say 7, right.

And what does this array store, this step? It says that for particle i j right, particle i j; the j th neighbor, so see here you have in you have incremented how many neighbors it has. So, that number equal to the particle index number the neighbor index number, right. So, if this distance is less than r s then in this array you say that the fifth neighbor; so, this is the fifth neighbor or the second neighbor of particle number i j is particle index g k right j k you are running over n part. And that is all that is there to it once this loop is completed every 100 iterations, you will have these two arrays which constitute your neighbor list it says each particle.

How many neighbors each particle has which is in this array and what is the particle index of each of these neighbors of particle i j that is stored in neigh list, ok. So, now, you have your neighbor list very good. Now, how do you use it; so, the way you use it is you have to use this neighbor list in subroutine calc force, right.

Previously in calc force you were checking for every pair of particle if the distance is less than r c and only then where you calculating, only then where you calculating the force, right. Here now there is n into n part calculation n into n minus 1 by 2 calculation is being done here in this neigh list which is being called every 100 iterations.

(Refer Slide Time: 20:14)



```
subroutine calc_force
    implicit none

    do  i = 1, n-part -1
        do j = 1, no-neighb(i).
            p = neigh-list(i, j)  ! jth neighbour of particle i
            dx = position(3*p-2) - position(3i-2)
            dy
            dz = position(3*p) - position(3*i).
                                            -> PBC
            dr = dx^2 + dy^2 + dz^2
         if(dr<rc) xhat = dx/r ;  yhat = dy/r   zhat = dz/r
            force = (                 ) ; potential = (    )
            tot-force(3*p-2) = tot-force(3*p-2) + force * xhat
            tot-force(3*i-2) = tot-force(3*i-2) - force * xhat
            ...
        enddo
    enddo
```

And in the calc force in the subroutine calculation of forces what do you do? You say do i equal to 1 2 n part minus 1 because you want to calculate the force on each particle, why n part minus 1? Because the last particle has to have some neighbor and that will already be calculated when you calculate force you just do it once right. If some force is acted on particle j, an equal and opposite force is being acted on particle I, right.

So, you do not need to calculate it twice; so, do i equal to 1 comma n part minus 1, do j equal to 1 to number of neighbors of i, number of neighbors of i right. Then p and in this step you are identifying which particle; particle number p is neigh list i j. So, j is the so, first neighbors, second neighbors are basically j changes from first neighbor, second neighbor, third neighbor, forth neighbor, fifth neighbor, seventh neighbor whatever.

So, this p is basically the j th neighbor of particle number i which are which is within distance r s, right. Then what then you calculate the distance d x position 3 p minus 2. Because p is the one; p is the particle index with which i is interacting. So, 3 p minus 2 minus position 3 i minus 2 similarly you calculate for d y and similarly you calculate for d z position 3 into p minus position 3 into i. Do PBC, i mean correct for PBC use the nearest image convention calculate dr the distance between 2 particles and basically if dr is less than rc right.

Only then you can calculate x hat, y hat, z hat which is nothing, but d x by r you have already corrected for the correct sign of d s using the nearest image convention. Then, y hat is the basically y cap which is the unit vector x hat, y hat and z hat is the unit vector and this is how you calculate that and this vector basically points from i to p, right. So, this is schematic diagram this is your 3 p minus 2 to 3 i minus 2, so, the vector is pointing from i to p, we have discussed this in 1 of the previous classes if you are confused please go back and have a look there.

So, you calculate the unit vectors then you calculate the magnitude of the force you have r, if r is less than r c then you calculate the magnitude of the force and then the expression of the po10tial we have discussed all of this and then the total force hitting. On particle number i the x component 3 into p minus 2 is the x component equal to total force, which has already been acted upon may be on other particles plus force into x hat, right.

And, what is the force acting on the i th particle. So, you will write tot force into 3 into i minus 2, the 3 i minus 2 stores of forces acting on particle i due to other particles minus here it was plus, here it is minus because equal and opposite if the force acting on p. And now, you are calculating a force acting on i which will be an opposite direction minus force x hat, right, and that is all that is there to it and then of course, this loop.

So, this loop is basically ends here you have an end do and you have an end do and that is the standard thing right that is all that is there to calculation of forces so, i using the neighbor list. And now you will see quite around 10 times faster or 10 20 times faster than your previous molecular dynamics code.

To end this class I want to tell you I really taught you the most simplest neighbor list, the actual labor lists which are used in a molecular dynamics calculations are much more advanced version of this.

But in this course, we are not trying to make you an expert of molecular dynamics right, we are just trying we are teaching you 5 or 6 different methods computational methods. So, that you have an exposure to different types of calculation techniques and through this you are learning how to develop algorithms which is our main focus, right. So, for molecular dynamics you need to do such checks with the periodic boundary condition how do you implement that, then we discussed about thermostat how do you implement that, cutting of forces how do you implement that what is the logic.

And here we discuss neighbor list this is the simplest one, if you really have to do neighbor molecular dynamics for your research of whatever company job that you do then of course, you will have more advanced techniques. Which you can learn from the book and then just like here you are going to slowly develop the algorithm you will be able I hope that you will be able to do that right. Here we discuss the algorithm, the implementation, the logic in great detail, ok. With this we will come to the end of this class and discussion shall go on to the next class.

Thank you.