Implementing Verlet Neighbour List algorithm in Molecular Dynamics

Course Project - CD61004

by

Shiv Sagar Sah (21CH10061)
Anish Roy (21PH10048)
Lakshya Bamne (20MA20029)

Submitted to
Dr. Sandeep Kumar Reddy

Course project for
High Performance Computing for Complex Physical System
(CD61004)
Indian Institute of Technology, Kharagpur

Contents

Chapter 1

Verlet Neighbour List Algorithm

1.1 Van der Waals forces

Potential Energy calculations between the molecules gives rise to the forces between these molecules and ultimately their motion. This potential energy is modeled by the Lennard Jones potential which gives rise to the Van der Waals forces acting on the molecules. But calculating the value for the Lennard Jones potential between every pair of molecules is computationally very expensive as well as unnecessary.

$$V(r) = 4\epsilon\big[(\frac{\sigma}{r})^{12} - (\frac{\sigma}{r})^{6}\big] \tag{1.1}$$

This is because after a **cut-off**[1] distance, the value of LJ Potential becomes negligible.

1.1.1 Truncating

When we use the cut-off distance to consider molecules for particle interactions, we achieve some level of optimization but we still calculate for $\frac{N(N-1)}{2}$ pairs which can be optimized using the Verlet Neighbour Lists.

1.1.2 Verlet Neighbour Lists

We use a data structure called the **Verlet Neighbour List** to achieve even more optimizations. Now the algorithm scales with $O(N * N_v)$ where N_v is the average number of molecules inside the Verlet radius around any molecule. The verlet radius is nothing but the **cut-off** radius along with a thin layer of **skin**.

$$r_{verlet} = r_{cutoff} + \delta s \tag{1.2}$$

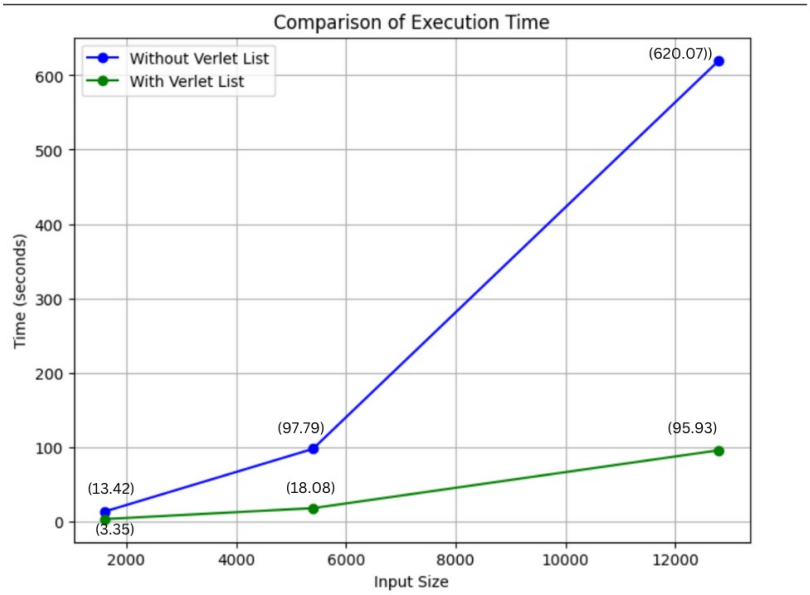
[1] Recommended cut-off is $2.5 * \sigma$

1.2 Implementation of Verlet Neighbour Lists

First we discuss the results obtained upon optimizing using the Verlet Neighbour Lists

1.2.1 Results

We have run the code for Molecular Dynamics simulations with (i) Truncation and with (ii) Verlet Neighbourhood Lists and the reulsts are visualised in the following graph.



Input Size	Time(without Verlet List)	Time(with Verlet List)
1600	13.42	3.35
5400	97.79	18.08
12800	620.07	95.93

Clearly the verlet neighbourhood lists provide a useful optimization with an acceptable tradeoff with solution accuracy and this difference in time is even more highlighted for very high input sizes.

1.2.2 Basic Idea of Verlet Lists

Basic idea is to maintain a data structure where the neighbours (molecules whose LJ potential is not-negligible) are stored and LJ and forces are only calculated for these group of neighbours, but this neighbour list is updated from time to time to account for molecules that move in and out of the verlet radius for a molecule.
Force calculations also need to be updated based on the neighbours, all the changes made to the code are listed in the next section.

Chapter 2

List of Change

2.1 Changes made in main.f90

```
!! Change-1 Declaration for the verlet list data structure and the total neighbours in the list
integer::max_neigh
integer, dimension(:,:), allocatable :: vL
```

```
!! Change-2 we have approximated that maximum neighbours of a single molecule will be a tenth of the total molecules
max_neigh=TotAtom/10

allocate(r(TotAtom,3),rm(TotAtom,3),v(TotAtom,3))
allocate(Force(TotAtom,3))
allocate(AtomLabel(TotAtom))

!! change-3 allocating memory for the verlet list
allocate(vL(TotAtom, TotAtom))


call initialize(TotAtom,CoorFileName,Temp,Mass,Box,r,v,AtomLabel)     ! get initial coordinates and velocities

!! Change-4 Initializing the verlet list using the initial positions for the molecules
call update_verlet_list(TotAtom, r, Rcut, vL, Box)


!! Change-5 signature for the subroutine force_calc is changes to accomodate the verlet_list
call force_calc(TotAtom,Box,Rcut,r,Sig,Eps,Force,PE,vL)
```

```
!! Change-6 we have to update the Verlet list after 10 md steps
if(MOD(md_step, 10)==0)then
  call update_verlet_list(TotAtom, r, Rcut, vL, Box)
end if
```

2.2 Changes made in force.f90

```
!! Change-5 signature for the subroutine force_calc is changed
subroutine force_calc(TotAtom,Box,Rcut,r,Sig,Eps,Force,PE,vL)
```

```
!! Initialization for the verlet list and index variables
integer, intent(inout) :: vL(TotAtom,TotAtom)
integer::i,j,neighbor_index
```

```
do i = 1, TotAtom

    !! Change-7 Inner loop is changed to only calculate LJ Potential for the molecules in the verlet list
    !! -> neighbor_index is used as an index which is a change
    !! -> force accumulation is also changed
    do j = 1, vL(i, 1)  ! Assuming vL(i, 1) stores the number of neighbors for atom i
        neighbor_index = vL(i, j + 1)  ! Adding 1 to skip the first element which stores the number of neighbors
        dr = r(i, :) - r(neighbor_index, :)
        ! Apply periodic boundary conditions
        dr = dr - Box * anint(dr / Box)
        r2 = dot_product(dr, dr)
        ! Check if the neighbor is within the cutoff radius
        if (r2 <= R2cut) then
            r2 = 1.0_dp / r2
            fac2 = r2 * Sig * Sig
            fac6 = fac2 * fac2 * fac2
            df = 48.0_dp * Eps * r2 * fac6 * (fac6 - 0.5_dp)
            fc = df * dr
            ! Accumulate forces
            Force(i, :) = Force(i, :) + fc
            Force(neighbor_index, :) = Force(neighbor_index, :) - fc
            ! Accumulate potential energy
            PE = PE + 4.0_dp * Eps * fac6 * (fac6 - 1.0_dp) - Ecut ! Shifted to zero at cutoff
        end if
    end do
end do
```

2.3 Changes made in integrate.f90

```
!! Change-8 signature for the subroutine integrate is changed to accomodate the verlet list
subroutine integrate(t, EQMDStep, TotAtom, Mass, Box, Temp, Rcut,Sig, Eps, AtomLabel, TimeStep, r, v, Force, KE, PE, vL)
```

```
!! Initializing the verlet list
integer, intent(inout) :: vL(TotAtom,TotAtom)
```

```
!! Force calculation is done using the updated subroutine accomodating the verlet lists
call force_calc(TotAtom,Box,Rcut,r,Sig,Eps,Force,PE,vL)
```

2.4 Changes made in verletList.f90

verletList.f90

```
!! We have defined a subroutine to update the verlet lists after some md steps
subroutine update_verlet_list(TotAtom, r, Rcut, vL,Box)
    implicit none
    integer, parameter :: dp=kind(0.d0)
    integer, intent(in) :: TotAtom
    real(kind=dp), intent(in) :: r(TotAtom, 3)
    real(kind=dp), intent(in) :: Rcut, Box
    integer, intent(inout) :: vL(TotAtom,TotAtom)

    integer :: i, j, count
    real(kind=dp) :: dr(3), r2, rv

    rv = Rcut + 2.0_dp  ! Adjust the buffer distance for the Verlet list

    ! Loop through particles and update Verlet list
    do i = 1, TotAtom
        ! Reset Verlet list for particle i
        count = 0

        ! Check distances from other particles and update Verlet list
        do j = i+1, TotAtom

            dr = r(i, :) - r(j, :)
            dr = dr - Box * anint(dr / Box)
            r2 = dot_product(dr, dr)

            if (r2 <= rv**2) then
                count = count + 1
                vL(i, count+1) = j

            endif
        end do

        ! Store the count in the first column of the vL array
        vL(i, 1) = count
    end do
end subroutine update_verlet_list
```

Chapter 3

Conclusions

3.1 Work distribution

This was a fairly complicated project but we managed to complete it as a team with equal efforts, but more specific work distribution is also provided

- Shiv Sagar Sah (21CH10061) : Coding, debugging and result visualization.
- Anish Roy (21PH10048) : Theory, Coding and debugging.
- Lakshya Bamne (20MA20029) : Coding, Report making and overall management.