rexpon_nonbonded.h/.cpp

vdW_Coulomb_Energy()

```
void vdW_Coulomb_Energy( rexpon_system *system, control_params *control,
                         simulation_data *data, storage *workspace,
                         rexpon_list **lists, output_controls *out_control
);
```

Parameters

system: The data structure containing all atoms (including x, v, f, ids of neighbors), and RexPoN parameters *rexpon_interaction* read from ffield description file.
control: Some global parameters. Only the following settings are involved in this function: *nonb_cut, lgflag, coulomb_off_flag, virial*
data: A data structure that records the calculated energies, virials, center of mass, etc of the system not individual atoms.
workspace: Holding temporary, intermediate calculation results. It also contains the parameters of the Taper function.
list: neighbor lists. The code will rely on these lists to find out neighbors of a given atom.
out_control: Not actually used in this function

Description

The main body of this function is a double-loop:

```
for( i = 0; i < natoms; ++i ) {
    for( pj = start_i; pj < end_i; ++pj ) {
        j = Get atom j from the neigbors of i
        /* Calculate the vdw interaction between atom i and j */
    }
}
```

The coordinates of i and j are obtained from system->my_atoms[i] and [j]. Note atom indexes i and j do NOT follow the order as appeared in the LAMMPS data file.
Within each loop, the following tasks will be performed:

1. Taper function values *Tap* and *dTap* are calculated based on atom distance *r_ij* and Taper parameters *workspace->Tap[]*. Note the Tap[] are already scaled by the cutoff distance *nonb_cut* (typically 12.0): *Tap[i] = Tap[i]*pow(nonb_cut,-i)*;
2. (New in RexPoN) a bool flag *flag_water* is added to judge whether atoms i,j are water atoms by test whether the atom names are "HW" and "OW". If *flag_water* is 1, then the vdw are calculated in a completely different way. Another flag *flag_QMMM* is also set to true if this is a water. Not clear about its meanings yet.

3. vdWaals interactions are calculated based on the value of *system->rexpon_param.gp.vdw_type*. As explained in *rexpon_ffield.cpp*, this flag is set based on atomic parameters *rcore2, acore2, and gamma_w*. But all the RexPoN codes reside in the `vdw_type == 2` clause*, i.e. with *"inner wall"* but without *"shielding"*.
4. Water and non-water are treated differently. Repulsive interactions are calculated first, then attractive interactions.
    - vdWaals in water are calculated by eqns in J.Chem.Phys. 149, 174502(2018) (The original RexPoN paper). The parameters are very confusing, see the following code, the comment shows the original name of parameters reported in Table II,III and eq. 6,7 of the above paper.

```
d_vdW = twbp->D;  // Aij
alf_vdW = twbp->beta; // -1 * alpha
re_vdW = twbp->alpha; // actually beta
bet_vdW = twbp->repa0; // actually gamma
gam_vdW = twbp->repr0; // actually nij
eta_vdW = twbp->repn;  // eta
del_vdW = twbp->repscal; // delta

exp1 = d_vdW * exp(-alf_vdW*r_ij+re_vdW); // eq 7 in paper
exp2 = exp(bet_vdW*pow(r_ij,gam_vdW)+eta_vdW*r_ij+del_vdW); // still eq 7

dexp1 = -alf_vdW * exp1; // calculates the derivatives
dexp2 = (bet_vdW*gam_vdW*pow(r_ij,(gam_vdW-1.0))+eta_vdW)*exp2;

e_vdW = exp1*exp2; // ENBrep(r) in eq 7
de_vdW = (dexp1*exp2 + exp1*dexp2)/r_ij; // and its derivative

data->my_en.e_vdW += Tap * e_vdW;
CEvd = Tap * de_vdW + dTap * e_vdW;
```

Therefore, the (repulsive only) energy are accumulated to *data->my_en.e_vdW*, and its derivate is temporarily stored in the variable *CEvd*
Shortly after the repulsive terms, the following codes calculate the attractive terms:

```
if(system->rexpon_param.gp.vdw_type==2 || system-
>rexpon_param.gp.vdw_type==3){
  /* unrelated codes from Reax */
  if (control->lgflag && flag_water) { // not active for non-water using
universal nonbond
      r_ij5 = pow( r_ij, 5.0 ); // r_ij is the atomic distance
      r_ij6 = pow( r_ij, 6.0 );
      re6 = pow( twbp->lgre, 6.0 ); // twbp->lgre is the RvdWij in Table
II
      e_lg = -(twbp->lgcij/( r_ij6 + re6 )); // This is eq 6. twbp->lgcij
is C6ij in Table II.
      data->my_en.e_vdW += Tap * e_lg;  // energy with taper
      de_lg = -6.0 * e_lg *  r_ij5 / ( r_ij6 + re6 ) ;
      CEvd += dTap * e_lg + Tap * de_lg / r_ij; // the derivative. CEvd
will be used later in force calcs.
```

```
    }
  }
```

- For non-water atoms, the Universal Nonbond as described in J.Chem.Phys. 151, 154111 (2019) is calculated.

```
    d_vdW = twbp->D;       // De in Table II
    re_vdW = twbp->r_vdW;  // Re in Table II
    le = twbp->alpha;      // L in Table II
    alf_vdW = twbp->repa0; // beta in Table III
    s0 = twbp->repr0;      // s0 ~ s5 are alpha0 ~ alpha5 in Table III
    s1 = twbp->repn;
    s2 = twbp->repscal;
    s3 = twbp->reps;
    s4 = twbp->lgre/2; //initially multiplied by 2
    s5 = twbp->lgcij;
    L2 = le*le; L3 = le*L2; L4 = L2*L2; L5 = le*L4;
    alf_vdW = alf_vdW/le;
    s1 = s1/le; s2 = s2/L2; s3 = s3/L3; s4 = s4/L4; s5 = s5/L5;
    dr = r_ij-re_vdW;
    dr2 = dr*dr;    dr3 = dr*dr2;   dr4 = dr*dr3;   r5 = dr*dr4;
    exp1 = -d_vdW * exp( -alf_vdW * dr); // first half of eq 8
    exp2 = s0 + s1*dr + s2*dr2 + s3*dr3 + s4*dr4 + s5*dr5 ; // 2nd half of
eq 8
    dexp1 = -alf_vdW * exp1;
    dexp2 = s1 + 2*s2*dr + 3*s3*dr2 + 4*s4*dr3 + 5*s5*dr4;
    e_vdW = exp1 * exp2;
    de_vdW = (dexp1*exp2 + exp1*dexp2)/r_ij;
    data->my_en.e_vdW += Tap * e_vdW;  // final energy with taper
    CEvd = Tap * de_vdW + dTap * e_vdW;  // derivative
```

There is a signficant flaw: s1 ~ s5 (alpha0 ~ alpha5) are fixed parameters of the UNB methods, and Re, De, L are properties of atoms. However, in this current implementation, all these parameters are provided as off-diagonal two-body parameters. 5. There is a *control->coulomb_off_flag*, which should be set to 0 to skip Coulomb interaction calculations (they are replaced by PQEq now). This flag is set in the lammps input file `pair_style rexpon NULL lgvdw yes coulomb_off yes checkqeq no`. This will make *CEclmb* and *e_ele* = 0.0 6. Finally, energy and force are "tallyed into per-atom energy". Most importantly, the forces on atoms are collected and saved to *workspace->f[i]* via

```
    rvec_ScaledAdd( workspace->f[i], -(CEvd + CEclmb), nbr_pj->dvec );
    rvec_ScaledAdd( workspace->f[j], +(CEvd + CEclmb), nbr_pj->dvec );
```

To understand the above codes, suppose that atoms i and j are interacting, and the total energy is a function of their distance only: $E=E(r_{ij})$, then the force on i equals:
$$ F_{on\ i} = -\nabla_{i}E(r_{ij})=-\frac{dE}{dr_{ij}}(\bold{r}_j-\bold{r}*i)$$*
In the above two lines, CEvd (and CEclmb, which is always zero since we do not include Coulombic term

here) is exactly $|frac{dE}{dr}{ij}}$, and *nbr_pj->dvec* is $(\bold{r}_j-\bold{r}_j)$. Therefore, what is accumulated in *workspace->f[i]* is in fact the force on atom i times -1. This can be confirmed by function `void PairRexPoN::read_rexpon_forces(int vflag)` defined in `pair_rexpon.cpp`, where the forces on each atom is read from RexPoN workspace:

```
    atom->f[i][0] += -workspace->f[i][0];
    atom->f[i][1] += -workspace->f[i][1];
    atom->f[i][2] += -workspace->f[i][2];
```

Other functions

There are 3 other functions defined in this module:

```
void Tabulated_vdW_Coulomb_Energy( rexpon_system *system,control_params
*control,
                                   simulation_data *data, storage
*workspace,
                                   rexpon_list **lists,
                                   output_controls *out_control );
void LR_vdW_Coulomb( rexpon_system *system, storage *workspace,
        control_params *control, int i, int j, double r_ij, LR_data *lr);
```

They come from the Reax codes and it seems like they have not been called in RexPoN cases (It is true?). Therefore they are not discussed here. Note that *LR_vdW_Coulomb* repeats the code within the inner loop of *vdW_Coulomb_Energy()*. Was it originally a subroutine of *vdW_Coulomb_Energy()* ?