```
! <compile=optimized>
#include "copyright.h"
#include "../include/assert.fh"
#include "../include/dprec.fh"
#define VACUUM3 603

#ifdef LES
module genbornles
#else
module genborn
#endif

_REAL_, private, dimension(:), allocatable :: r2x,rjx,vectmp1,vectmp2, &
                              vectmp3,vectmp4,vectmp5,sumdeijda,psi
integer, private, dimension(:), allocatable :: temp_jj,k_vals,j_vals, neckidx
logical, private, dimension(:), allocatable :: skipv

#ifdef LES
_REAL_, private, dimension(:), allocatable ::
scalefac,lesscalefac,rbornlong,rix,vtemp7,vthi2
logical, private, dimension(:), allocatable :: longskipv,spreadfrc
integer, private, dimension(:), allocatable :: nradii,iridx,jridx
#endif


public allocate_gb, deallocate_gb, egb, igb7_init

contains

!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
!+ allocates scratch space for egb()
subroutine allocate_gb( natom,ncopy )

   implicit none
   integer, intent(in) :: natom, ncopy
   integer ier

#ifdef LES

   allocate( r2x(ncopy*natom), rjx(ncopy*natom), vectmp1(ncopy*natom), &
             vectmp2(ncopy*natom),   &
             vectmp3(ncopy*natom), vectmp4(ncopy*natom),
vectmp5(ncopy*natom), &
             sumdeijda(ncopy*natom), psi(ncopy*natom),
temp_jj(ncopy*natom),  &
             skipv(0:natom),k_vals(natom),j_vals(natom), neckidx(natom), &
             scalefac(natom*ncopy), spreadfrc(natom*ncopy), vthi2(ncopy), &
             longskipv(natom*ncopy), nradii(natom),lesscalefac(natom*ncopy),
&
             rbornlong(natom*ncopy), &
             rix(natom*ncopy),iridx(natom*ncopy),jridx(natom*ncopy), &
             vtemp7(ncopy),stat = ier )
#else
   allocate( r2x(natom), rjx(natom), vectmp1(natom), vectmp2(natom),   &
             vectmp3(natom), vectmp4(natom), vectmp5(natom), &
             sumdeijda(natom), psi(natom), temp_jj(natom),  &
             skipv(0:natom),k_vals(natom),j_vals(natom), neckidx(natom), &
             stat = ier )
#endif

   REQUIRE( ier == 0 )
   return

end subroutine allocate_gb

!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```



```
!+ deallocates scratch space for egb()
subroutine deallocate_gb( )

   implicit none
   integer ier

   ! assume that if r2x is allocated then all are allocated
   if ( allocated( r2x ) ) then
      deallocate( skipv, neckidx, j_vals, k_vals, temp_jj, psi, sumdeijda, &
            vectmp5, vectmp4, vectmp3, vectmp2, vectmp1, rjx, r2x, stat =
ier )
      REQUIRE( ier == 0 )
#ifdef LES
      deallocate( longskipv,scalefac,lesscalefac,nradii,rbornlong, rix, &
                  iridx,jridx,vtemp7,vthi2,spreadfrc, stat = ier )
      REQUIRE( ier == 0 )
#endif

   else
      REQUIRE( .false. )  ! cannot deallocate un-allocated array
   end if
   return

end subroutine deallocate_gb

!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
!+ Initialize table of indexes for GB neck lookup table.
subroutine igb7_init(natom, rborn)

  implicit none
  integer, intent(in) :: natom

  integer i

  _REAL_, intent(in) :: rborn(*)

  do i=1,natom
     neckidx(i) = nint((rborn(i)-1.0d0)*20d0)
     if (neckidx(i) < 0 .or. neckidx(i) > 20) then
        write (6,*) "Atom ",i," has radius ",rborn(i), &
                    " outside of allowed range"
        write (6,*) "of 1.0 to 2.0 angstroms for igb=7 or 8. Regenerate &
                    &prmtop file with bondi radii."
        call mexit(6,1)
     end if
  end do

end subroutine igb7_init

!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
!+ handles generalized Born functionality, plus reg. nonbon, plus surface
area
subroutine egb(x,f,rborn,fs,reff,onereff,charge,iac,ico,numex, &
      natex,dcharge,cut,ntypes,natom,natbel, &
      epol,eelt,evdw,esurf,dvdl,vdwrad,ineighbor,p1,p2,p3,p4, &
      ncopy &
#ifndef LES
      ,gbvalpha,gbvbeta,gbvgamma &
#endif
      )
      ! gbvalpha,gbvbeta,gbvgamma arrays for GB
      ! put all gbalpha, gbbeta, gbgamma for each atom in these 3 arrays
      ! (look at mdread.f for details)

   !------------------------------------------------------------------------
```

```
127     !     Compute nonbonded interactions with a generalized Born model,
128     !     getting the "effective" Born radii via the approximate pairwise
    method
129     !     Use Eqs 9-11 of Hawkins, Cramer, Truhlar, J. Phys. Chem. 100:19824
130     !     (1996).  Aside from the scaling of the radii, this is the same
131     !     approach developed in Schaefer and Froemmel, JMB 216:1045 (1990).
132     
133     !     The input coordinates are in the "x" array, and the forces in "f"
134     !     get updated; energy components are returned in "epol", "eelt" and
135     !     "evdw".
136     
137     !     Input parameters for the generalized Born model are "rborn(i)", the
138     !     intrinsic dielectric radius of atom "i", and "fs(i)", which is
139     !     set (in routine mdread) to (rborn(i) - offset)*si.
140     
141     !     Input parameters for the "gas-phase" electrostatic energies are
142     !     the charges, in the "charge()" array.
143     
144     !     Input parameters for the van der Waals terms are "cn1()" and
    "cn2()",
145     !     containing LJ 12-6 parameters, and "asol" and "bsol" containing
146     !     LJ 12-10 parameters.  (The latter are not used in 1994 are more
147     !     forcefields.)  The "iac" and "ico" arrays are used to point into
148     !     these matrices of coefficients.
149     
150     !     The "numex" and "natex" arrays are used to find "excluded" pairs of
151     !     atoms, for which gas-phase electrostatics and LJ terms are skipped;
152     !     note that GB terms are computed for all pairs of atoms.
153     
154     !     If gbsa=1, then an approximate surface-area dependent term is
155     !     computed, with the resulting energy placed into "esurf".  The
156     !     algorithm is from J. Weiser, P.S. Shenkin, and W.C. Still,
157     !     "Approximate atomic sufraces from linear combinations of pariwise
158     !     overlaps (LCPO)", J. Computat. Chem. 20:217 (1999).
159     
160     !     The code also supports a multiple-time-step facility:
161     
162     !       pairs closer than sqrt(cut_inner) are evaluated every nrespai
    steps;
163     !         "   between sqrt(cut_inner) and sqrt(cut) are evaluated
164     !                        every nrespa steps
165     !         "   beyond sqrt(cut) are ignored
166     
167     !       the forces arising from the derivatives of the GB terms with
    respect
168     !          to the effective Born radii are evaulated every nrespa steps
169     
170     !       the surface-area dependent term is evaluated every nrespa steps
171     
172     !       the effective radii are only updated every nrespai steps
173     
174     !     (Be careful with the above: what seems to work is dt=0.001,
175     !     nrespai=2, nrespa=4; anything beyond this seems dangerous.)
176     
177     !     Written 1999-2000, primarily by D.A. Case, with help from C. Brooks,
178     !       T. Simonson, R. Sinkovits  and V. Tsui.  The LCPO implementation
179     !       was written by V. Tsui.
180     
181     !     Vectorization and optimization 1999-2000, primarily by C. P. Sosa,
182     !       T. Hewitt, and D. A. Case.  Work presented at CUG Fall of 2000.
183     !--------------------------------------------------------------------------
184     
185     use icosasurf, only : icosa_init, icosa_sphere_approx
186     use decomp, only: decsasa, decpair
187     use qmmm_module, only : qmmm_nml,qmmm_struct,qm2_struct
188     use parms, only: cn1,cn2
```



```
189   #ifdef HAS_10_12
190      use parms, only: asol,bsol
191   #endif
192      use constants, only: zero, one, two, three, four, five, six, seven, &
193                           eight, nine, ten, eleven, twelve, half, third, &
194                           fourth, eighth, pi, fourpi, alpb_alpha, &
195                           AMBER_ELECTROSTATIC
196      use crg_reloc, only: ifcr, cr_add_dcdr_factor
197   
198   #ifdef LES
199      use les_data, only : elesp, lestmp, lfac, nlesty, lestyp, cnum
200      use pimd_vars, only: ipimd,nrg_all
201   #  ifdef MPI
202         use remd, only : rem
203   #  endif
204   #endif
205      implicit none
206   
207   #ifdef MPI
208   #  include "parallel.h"
209   #  ifdef MPI_DOUBLE_PRECISION
210   #    undef MPI_DOUBLE_PRECISION
211   #  endif
212      include 'mpif.h'
213   #  ifdef CRAY_PVP
214   #    define MPI_DOUBLE_PRECISION MPI_REAL8
215   #  endif
216   #endif
217   #  include "../include/md.h"
218   #  include "def_time.h"
219   #ifdef LES
220      _REAL_ lfaci,templfac,tempscale,tmp,addi,addj
221      logical first,spread
222      integer ijcnum,idx1,idx2,istrt,jstrt,icnum,jcnum,icopy
223      _REAL_  nrg_vdw_tmp,nrg_egb_tmp,nrg_ele_tmp
224   #else
225      _REAL_ temp7
226   #endif
227   
228      _REAL_ deel
229      logical onstep,onstepi,oncpstep,oncestep,doeel, dovdw
230      _REAL_ x,f,rborn,fs,reff,onereff,charge,cut, &
231             epol,eelt,evdw,esurf,vdwrad,p1,p2,p3,p4, &
232             dcharge
233      _REAL_ totsasa,extdieli,intdieli, &
234             lastxj,lastyj,lastzj,xi,yi,zi,ri,four_ri,ri1i,xij,yij,zij, &
235             dij1i,r2,dij,sj,sj2,frespa,si,sumaij,sumajk, &
236             sumaijajk,sumdaijddijdxi,sumdaijddijdyi,sumdaijddijdzi, &
237             sumdaijddijdxiajk,sumdaijddijdyiajk,sumdaijddijdziajk, &
238             xj,yj,zj,rij,tmpaij,aij,daijddij,daijddijdxj, daijddijdyj, &
239             daijddijdzj,sumajk2,sumdajkddjkdxj,sumdajkddjkdyj, &
240             sumdajkddjkdzj,p3p4aij,xk,yk,zk,rjk2,djk1i,rjk,vdw2dif, &
241             tmpajk,ajk,dajkddjk,dajkddjkdxj,dajkddjkdyj,dajkddjkdzj, &
242             daidxj,daidyj,daidzj,ai,daidxi,daidyi,daidzi,qi,dumx, &
243             dumy,dumz,de,rj,temp1,fgbi,rinv,r2inv,qiqj,fgbk,expmkf, &
244             dl,e,temp4,temp5,temp6,eel,r6inv,f6,f12, &
245             dedx,dedy,dedz,qi2h,dij2i,datmp,dij3i, &
246             qid2h,dvdl,thi,thi2, self_e,reff_j
247   !  _REAL_ intdiele_inv !variable for gas phase calculations (future fix by
    Dan Parkin)
248       _REAL_ :: alpb_beta, one_Arad_beta
249   #ifndef LES
250       _REAL_ :: gbvalpha(*),gbvbeta(*),gbvgamma(*) !add
    gbvalpha,gbvbeta,gbvgamma
251   #endif
252   
```

```
#ifdef HAS_10_12
   _REAL_ :: r10inv, f10
#endif

   integer count,count2,icount,ineighbor(*),max_count, iminus
   integer iac,ico,numex,natex,ntypes,natom,natbel,ncopy
   integer i,j,k,kk1,maxi,num_j_vals,jjj,count2_fin,num_k_vals, &
           iexcl,iaci,jexcl,jexcl_last,jjv,ic,kk
   integer j3
#ifdef MPI
   integer mpistart, ierr
#endif
   _REAL_ f_x,f_y,f_z,f_xi,f_yi,f_zi
   _REAL_ dumbo, tmpsd, rborn_i, psi_i
#ifndef LES
   _REAL_ gba_i, gbb_i, gbg_i ! temporary variables for GB (calculating GB
force)
#endif

   ! Variables for QMMM specific loops
   integer qm_temp_count
!Locals for link atoms
   _REAL_ :: forcemod(3)
   integer :: lnk_no, mm_no, qm_no


   ! variables needed for icosa surface area calculation
   integer ineighborpt

   ! variables needed for smooth integration cutoff in Reff:
   _REAL_ rgbmax2, rgbmax1i, rgbmax2i,rgbmaxpsmax2
   !     _REAL_  datmp2

   _REAL_ onekappa !1/kappa

   ! Scratch variables used for calculating neck correction
   _REAL_ mdist,mdist2,mdist3,mdist5,mdist6

#include "gbneck.h"

#ifdef LES
   dimension x(*),f(*),rborn(*),charge(*),iac(*), &
         ico(*),numex(*),natex(*),fs(*),reff(ncopy*natom),
onereff(ncopy*natom), &
         vdwrad(*),p1(*),p2(*),p3(*),p4(*), &
         dcharge(*)
#else
   dimension x(*),f(*),rborn(*),charge(*),iac(*), &
         ico(*),numex(*),natex(*),fs(*),reff(natom), onereff(natom), &
         vdwrad(*),p1(*),p2(*),p3(*),p4(*), &
         dcharge(*)

   integer ncopy2
#endif

   !   FGB taylor coefficients follow
   !   from A to H :
   !   1/3 , 2/5 , 3/7 , 4/9 , 5/11
   !   4/3 , 12/5 , 24/7 , 40/9 , 60/11

   _REAL_  ta
   _REAL_  tb
   _REAL_  tc
   _REAL_  td
   _REAL_  tdd
   _REAL_  te
```



```
   _REAL_  tf
   _REAL_  tg
   _REAL_  th
   _REAL_  thh
   parameter ( ta   = third )
   parameter ( tb   = two / five )
   parameter ( tc   = three / seven )
   parameter ( td   = four / nine )
   parameter ( tdd  = five / eleven )
   parameter ( te   = four / three )
   parameter ( tf   = twelve / five )
   parameter ( tg   = three * eight / seven )
   parameter ( th   = five * eight / nine )
   parameter ( thh  = three * four * five / eleven )

   qid2h = 0.d0
   dl = 0.d0
   f6 = 0.d0
   f12 = 0.d0

!============ QMMM ===========================================================
!  If ifqnt == True and igb /=0 and igb /= 6 then we will do EGB with QMMM.

!  In this case the charge array should currently be zero for the QM atoms.
!  This corresponds to qmgb = 0 where we do only EGB(MM-MM) -- skipping
!  QM-MM and QM-QM interactions.

!  If qmgb==1 then we need to copy back the qm resp charges from
!  qm_resp_charges, so we can do EGB on everything. When done we need to
!  ensure we zero the QM charge array again.

!  Before doing the non-bonded electrostatics:
   if (qmmm_nml%ifqnt) then
      if (qmmm_nml%qmgb == 1) then

         call
qmmm_restore_mm_charges(qmmm_struct%nquant,qmmm_struct%qm_resp_charges,charge,
&
                                      qmmm_struct%scaled_mm_charges,
qmmm_struct%iqmatoms, &
qmmm_nml%chg_lambda,qmmm_struct%nlink,qmmm_struct%link_pairs, &
                                      qmmm_struct%mm_link_pair_resp_charges,
&
                                      .false.)

         ! Unlike ewald where we don't explicitly skip QM-MM interactions,
         ! in GB we do explicitly skip them so it doesn't matter that the
         ! QM charges are not set to zero in the charge array.

      else if (qmmm_nml%qmgb > 1 ) then

        ! In this case we need to fill the charge array with the Mulliken
        ! charges from the SCF.  Note if igb==2 we skip the radii
calculation
        ! below as this has already been done before the call to qm_mm.
        do qm_temp_count = 1, qmmm_struct%nquant_nlink
           charge(qmmm_struct%iqmatoms(qm_temp_count)) = &
                   qm2_struct%scf_mchg(qm_temp_count)*AMBER_ELECTROSTATIC
        end do
      end if

      !We need to replace the MM link pair coordinates with
      !the MM link atom coordinates.
      call adj_mm_link_pair_crd(x)
```

```
377         !We also need to zero the nlink part of dxyzqm so we can accumulate
    link atom forces in this routine.
378         qmmm_struct%dxyzqm(1:3,qmmm_struct%nquant+1:qmmm_struct%nquant_nlink)
    = zero
379
380      end if
381   !============ END QMMM
    ==================================================
382
383      epol = zero
384      eelt = zero
385      evdw = zero
386      esurf = zero
387      totsasa = zero
388      thi2 = zero
389      onekappa = zero
390      count2_fin = 0
391   #ifdef LES
392      elesp = zero
393   #endif
394      onstep = mod(irespa,nrespa) == 0
395      onstepi = mod(irespa,nrespai) == 0
396      if( .not.onstepi ) return
397
398      oncpstep = (icnstph == 1 .and. mod(irespa, ntcnstph) == 0) .or. icnstph
    == 2
399
400      oncestep = (icnste == 1 .and. mod(irespa, ntcnste) == 0) .or. icnste == 2
401
402   !     variable for gas phase calculations (future fix by Dan Parkin)
403   !  intdiele_inv = one/intdiel
404
405      if(alpb == 1) then
406   !     Sigalov Onufriev ALPB (epsilon-dependent GB):
407         alpb_beta = alpb_alpha*(intdiel/extdiel)
408         extdieli = one/(extdiel*(one + alpb_beta))
409         intdieli = one/(intdiel*(one + alpb_beta))
410         one_Arad_beta = alpb_beta/Arad
411         if (kappa/=zero) onekappa = one/kappa
412      else
413      !  Standard Still's GB - alpb=0
414         extdieli = one/extdiel
415         intdieli = one/intdiel
416         one_Arad_beta = zero
417      end if
418
419      ! Smooth "cut-off" in calculating GB effective radii.
420      ! Implementd by Andreas Svrcek-Seiler and Alexey Onufriev.
421      ! The integration over solute is performed up to rgbmax and includes
422      ! parts of spheres; that is an atom is not just "in" or "out", as
423      ! with standard non-bonded cut.  As a result, calclated effective
424      ! radii are less than rgbmax. This saves time, and there is no
425      ! discontinuity in dReff/drij.
426
427      ! Only the case rgbmax > 5*max(sij) = 5*fsmax ~ 9A is handled; this is
428      ! enforced in mdread().  Smaller values would not make much physical
429      ! sense anyway.
430
431      rgbmax2 = rgbmax*rgbmax
432      rgbmax1i = one/rgbmax
433      rgbmax2i = rgbmax1i*rgbmax1i
434      rgbmaxpsmax2 = (rgbmax+fsmax)**2
435
436   #ifdef LES
437   ! initialize some things for GB+LES
438
```



```
439   ! one over number of LES copies
440   ! GB+LES only works with 1 LES region so we can't have multiple
441   ! copy numbers like we can with PME
442
443      lfac = float(ncopy)
444      lfaci = one/(lfac)
445
446   #else
447      ncopy2 = ncopy
448   #endif
449
450      !------------------------------------------------------------------------
451      !      Step 1: loop over pairs of atoms to compute the effective Born
    radii.
452      !------------------------------------------------------------------------
453      !
454      ! The effective Born radii are now calculated via a call at the
455      ! beginning of force.
456      !
457      !------------------------------------------------------------------------
458
459      iexcl = 1 !moved to outside the index loop from original location in
    "step 2"
460
461   #ifdef MPI
462      do i=1,mytaskid
463         iexcl = iexcl + numex(i)
464      end do
465      mpistart = mytaskid+1
466   #endif
467
468      maxi = natom
469      if(natbel > 0) maxi = natbel
470
471      !------------------------------------------------------------------------
472      !
473      !     Step 2: loop over all pairs of atoms, computing the gas-phase
474      !             electrostatic energies, the LJ terms, and the off-diagonal
475      !             GB terms.  Also accumulate the derivatives of these off-
476      !             diagonal terms with respect to the inverse effective radii,
477      !             sumdeijda(k) will hold  sum over i,j>i ( deij/dak ),  where
478      !             "ak" is the inverse of the effective radius for atom "k".
479      !
480      !             Update the forces with the negative derivatives of the
481      !             gas-phase terms, plus the derivatives of the explicit
482      !             distance dependence in Fgb, i.e. the derivatives of the
483      !             GB energy terms assuming that the effective radii are
484      !             constant.
485      !
486      !------------------------------------------------------------------------
487
488   #ifdef LES
489      sumdeijda(1:natom*ncopy) = zero
490   #else
491      sumdeijda(1:natom) = zero
492   #endif
493
494      call timer_start(TIME_GBFRC)
495
496      !       Note: this code assumes that the belly atoms are the first natbel
497      !             atoms...this is checked in mdread.
498
499   #ifdef MPI
500      do i=mpistart,maxi,numtasks
501   #else
502
```

```
503     do i=1,maxi
504  #endif
505
506  #ifdef LES
507        lestmp = nlesty*(lestyp(i)-1)
508  #endif
509
510        xi = x(3*i-2)
511        yi = x(3*i-1)
512        zi = x(3*i  )
513        qi = charge(i)
514        ri = reff(i)
515        four_ri = four*reff(i)
516        iaci = ntypes * (iac(i) - 1)
517        jexcl = iexcl
518        jexcl_last = iexcl + numex(i) -1
519        dumx = zero
520        dumy = zero
521        dumz = zero
522
523  #if defined(LES)
524        icnum = cnum(i)
525        nrg_vdw_tmp = zero
526        nrg_ele_tmp = zero
527        nrg_egb_tmp = zero
528  #endif
529
530        !         -- check the exclusion list for eel and vdw:
531
532        do k=i+1,natom
533           skipv(k) = .false.
534        end do
535        do jjv=jexcl,jexcl_last
536           skipv(natex(jjv))=.true.
537        end do
538
539        ! QMMM:  We have 2 lists here:
540        !        atom_mask which is natom long and .true. for each qm atom
541        !        skipv which is natom long and is .true. if atom v should
542        !             be skipped for this atom
543
544        !Step 1 - is current atom i a QM atom - NOT link atoms?
545
546        if (qmmm_nml%ifqnt) then
547           if ( qmmm_struct%atom_mask(i) ) then  !yes it is
548             !step 2 - exclude all other QM atoms
549             do qm_temp_count = 1, qmmm_struct%nquant
550                skipv(qmmm_struct%iqmatoms(qm_temp_count)) = .true.
551                !The VDW between the link atom and the MM atoms is not
552                !calculated. Instead MM link pair atom with Real QM atoms is
    calculated.
553             end do
554           !Is atom i an MM link pair atom?
555           else if ( qmmm_struct%mm_link_mask(i) ) then !yes it is
556             !Exclude all interactions (eel and VDW) with this
557             !atom. We will do the MM link pair VDW terms manually
558             !at the end of this routine.
559             skipv(i+1:natom) = .true. !We can't just cycle here because we
    want the GB interactions.
560           end if
561        end if
562
563        ! END QMMM
564
565        icount = 0
566        do j=i+1,natom
```



```
567
568  #ifdef LES
569  ! skip pairs in different copies
570           if (icnum.ne.0.and.cnum(j).ne.0.and.icnum.ne.cnum(j)) cycle
571  #endif
572
573           xij = xi - x(3*j-2)
574           yij = yi - x(3*j-1)
575           zij = zi - x(3*j  )
576           r2 = xij*xij + yij*yij + zij*zij
577
578           if( r2 <= cut .and. (onstep .or. r2 <= cut_inner)) then
579  #ifdef LES
580  ! set up the loop over (possible) multiple effective radii for i and j
581
582           ijcnum=icnum+cnum(j)
583
584  ! tempscale is for normalizing the epol when we average over pairs of reff
     for i and j
585  ! default to only a single reff for each.
586  ! use a temp value because this may be assigned to several icount values if
     we have a loop
587  ! also default to no scaling factor for LES:LES pairs
588  ! tempscale is not one only for non-les:non-les pairs with multiple radii
589  ! and only used for epol.
590  ! templfac is not one only for les:les pairs and used for epol, eel and evdw
591  !
592           tempscale=one
593           templfac=one
594
595           spread = .false.
596
597           if( ijcnum == 0 ) then
598
599  ! both atoms are non-LES, see if we need to average
600
601             if (nradii(i).gt.1.or.nradii(j).gt.1) then
602
603  ! neither are LES, average over all ncopy pairs of reff
604  ! do NOT average over ncopy*ncopy pairs - only do pairs
605  ! with the same index.
606  ! do average even if only 1 has multiple reff, nothing to
607  ! be saved by using only the one reff for 1 atom if the other
608  ! has multiple reff
609
610
611                 idx1=1
612                 idx2=ncopy
613                 tempscale=lfaci
614             else
615
616
617  ! no need to average, use first reff for both (since all are average reff)
618  ! here is where we need to deal with the sumdeijda problem
619  ! if we use only 1 radius pair, we still need to divide the
620  ! interaction among the radii for the calculation of derivatives
621
622                 idx1=1
623                 idx2=1
624
625  ! set a flag saying whether the force for this interaction needs
626  ! to be spread across all of the sumdeijda for the atom
627  ! even if we use 1 reff, we need the force to arise from all atoms
628  ! or else the forces on the LES atoms will not be balanced
629  ! we only do this for non-LES:non-LES pairs
630
```

```
631                spread=.true.
632              endif
633            else
634
635  ! at least 1 is LES, no averaging, only use the cnum reff
636  ! for LES:LES, use the only reff that these have (cnum)
637  ! for LES:non-LES, use the reff for the non-LES atom that goes with this LES
     copy (cnum)
638  !
639              if (icnum.gt.0) then
640
641  ! i is LES (j may be too), use i's cnum (same as j's if j is LES or
642  ! else we would have skipped pair)
643
644                  idx1=icnum
645                  idx2=icnum
646
647  ! set les scaling fac for ele, vdw if both are LES
648
649                  if (cnum(i).eq.cnum(j)) templfac=lfac
650              else
651
652  ! i is not LES, j must be, so use j's cnum
653
654                  idx1=cnum(j)
655                  idx2=cnum(j)
656              endif
657            endif
658
659            istrt = ncopy * (i-1)
660            jstrt = ncopy * (j-1)
661
662
663            first=.true.
664            do k=idx1,idx2
665
666  ! non-LES uses just ri (reff(i)) in this loop, doesn't need to be vector
667
668              rix(icount+1)=reff(istrt+k)
669
670              reff_j=reff(jstrt+k)
671
672  ! set longskipv, which serves the purpose of telling if an atom is
673  ! excluded and also telling if this is not the first in the loop over
674
675  ! pairs of reff for an atom (only calculate nonbonds for the first pair of
     reff)
676  ! longskipv is only set for the icount atoms! (not all j like skipv)
677
678               if ((.not.first).or.skipv(j)) then
679                 longskipv(icount+1)=.true.
680               else
681                 longskipv(icount+1)=.false.
682               endif
683
684  ! for LES we need to know which reff this is when we calculate the sumdeija
685  ! so set a pointer here to tell us which reff this i,j pair are using
686  ! this points directly into correct spot in reff or sumdeijda
687
688               iridx(icount+1) = istrt+k
689               jridx(icount+1) = jstrt+k
690
691  ! set flag to tell whether sumdeijda needs to be dividing among components
     for non-LES atoms
692  ! temporary value was set above
693
```



```
694              spreadfrc(icount+1) = spread
695
696  #else
697  ! set for non-LES case so we can refer to reff_j not reff(j)
698  ! this is needed with LES since each atom j has multiple reff
699
700              reff_j = reff(j)
701  #endif
702
703              icount = icount + 1
704              temp_jj(icount) = j
705              r2x(icount) = r2
706
707  ! carlos changed this, we are STILL INSIDE LOOP OVER K for multiple LES
     reff_j
708  ! so we need to use reff_j not reff(j) (since LES may have multiple reff_j
     for atom j)
709
710              rjx(icount) = reff_j
711
712  #ifdef LES
713                 ! set scaling factor here where we know if we are doing
714                 ! averaging over multiple radii. in the loops below we don't
715                 ! have that info anymore without checking nradii
716                 ! we couldn't set these above since we didn't have an
717                 ! icount value yet
718
719                 scalefac(icount)=tempscale
720                 lesscalefac(icount)=templfac
721
722                 ! set first to F so that only the first of these i,j
723                 ! entries will have nonbonds
724
725                 first=.false.
726
727              enddo ! LOOP OVER K, pairs of reff for i and j
728  #endif
729         end if !r2 <= cut
730       end do
731
732       if( igb/=6 ) then
733
734  #ifdef LES
735           ! rix needs to be used instead of ri since i doesn't have just 1
     reff
736           vectmp1(1:icount) = four*rix(1:icount)*rjx(1:icount)
737  #else
738           vectmp1(1:icount) = four_ri*rjx(1:icount)
739  #endif
740
741           call vdinv( icount, vectmp1, vectmp1 ) !Invert things
742           vectmp1(1:icount) = -r2x(1:icount)*vectmp1(1:icount)
743           call vdexp( icount, vectmp1, vectmp1 )
744                ! [ends up with Exp(-rij^2/[4*ai*aj])]
745  #ifdef LES
746           ! ri is not the same for all of the icount!
747           vectmp3(1:icount) = r2x(1:icount) + &
748                   rjx(1:icount)*rix(1:icount)*vectmp1(1:icount) !ends up with
     fij
749  #else
750           vectmp3(1:icount) = r2x(1:icount) +
     rjx(1:icount)*ri*vectmp1(1:icount)
751                ! [ends up with fij]
752  #endif
753
754           ! CARLOS: LES
```

```
755           ! we now have two sets of vectors: one is 1 to last atom, the other
756           ! is 1 to icount
757           ! the atom one is for vdw/ele and for excl, the other is for gb
    offdiag
758           ! which do we loop over? unless we have a pointer we need to loop
759           ! over the bigger one, have a pointer j in it, and pull excl out
760           ! of that. but how to know to skip the nonbonds for all after the
761           ! first in a loop over a pair? maybe we should have the excl loop
762           ! done for icount, not atoms.
763           ! SKIPV PART OK, DO THE DISTANCE PART SO WE DON'T HAVE TO INV ALL
    RIJ
764           ! FOR NOW IT'S OK, JUST SLOW
765
766           call vdinvsqrt( icount, vectmp3, vectmp2 ) !vectmp2 = 1/fij
767
768           if( kappa /= zero ) then
769              call vdinv( icount, vectmp2, vectmp3 )
770              vectmp3(1:icount) = -kappa*vectmp3(1:icount)
771              call vdexp( icount, vectmp3, vectmp4 ) !exp(-kappa*fij)
772           end if
773
774        end if
775        call vdinvsqrt( icount, r2x, vectmp5 ) !1/rij
776
777        ! vectmp1 = Exp(-rij^2/[4*ai*aj])
778        ! vectmp2 = 1/fij
779        ! vectmp3 = -kappa*fij - if kappa/=zero, otherwise = fij
780        ! vectmp4 = exp(-kappa*fij)
781        ! vectmp5 = 1/rij - note with qmmm this contains the
782                            !distance to mm_link pairs not to QM link atoms.
783
784     !---- Start first outer loop ----
785   !!! !dir$ ivdep
786
787        do k=1,icount
788           j = temp_jj(k)
789
790           xij = xi - x(3*j-2)
791           yij = yi - x(3*j-1)
792           zij = zi - x(3*j  )
793           r2 = r2x(k)
794           qiqj = qi * charge(j)
795           if( igb/=6 ) then
796
797             if( kappa == zero ) then
798                fgbk = zero
799                expmkf = extdieli
800             else
801                expmkf = vectmp4(k)*extdieli
802
803
804                fgbk = vectmp3(k)*expmkf !-kappa*fij*exp(-kappa*fij)/Eout
805                if(alpb == 1) then ! Sigalov Onufriev ALPB:
806                   fgbk = fgbk+(fgbk*one_Arad_beta*(-vectmp3(k)*onekappa))
807
808                   ! (-kappa*fij*exp(-kappa*fij)(1 + fij*ab/A)/Eout)*(1/fij+ab/
    A)
809                   ! Note: -vectmp3(k)*onekappa = fij
810                end if
811             end if
812             dl = intdieli - expmkf
813
814             fgbi = vectmp2(k)  !1/fij
815
816   #ifdef LES
817             ! epol gets scaled with scalefac
```



```
818             ! the epol gets scaled for multiple reff pairs for i/j, but
819             ! ele is NOT scaled since the extras are skipped using longskipv)
820             ! the epol for LES-LES pairs gets scaled with lfac, and ele does
    too
821             ! so we can't mix the lfac and the reff pair scaling in a
822             ! single array; also can't pre-scale qiji since it is sometimes
823             ! used with scaling and sometimes not.
824
825             temp1 = -dl*(fgbi + one_Arad_beta)*scalefac(k)*lesscalefac(k)
826   #else
827             temp1 = -dl*(fgbi + one_Arad_beta)
828   #endif
829             e = qiqj*temp1
830             if ( ifcr /= 0 ) then
831                call cr_add_dcdr_factor( i, temp1*charge(j) )
832                call cr_add_dcdr_factor( j, temp1*qi )
833             end if
834             epol = epol + e
835
836   #ifdef LES
837             ! icnum is independant to j loop, so we cache the nrg here
838             nrg_egb_tmp = nrg_egb_tmp + e
839   #endif
840
841             if(idecomp == 1 .or. idecomp == 2) then
842                call decpair(1,i,j,e)
843             else if(idecomp == 3 .or. idecomp == 4) then
844                call decpair(-1,i,j,e)
845             end if
846   #ifdef MPI
847   #  ifdef LES
848             if(rem == 2) then
849               if(cnum(i) > 0 .or. cnum(j) > 0) then
850                 elesp = elesp + e
851               end if
852             end if
853   #  endif
854   #endif
855             if (oncpstep .or. oncestep) then
856                dvdl = dvdl - (dl*fgbi*dcharge(i)*dcharge(j)+e)
857             end if
858             temp4 = fgbi*fgbi*fgbi !1/fij^3
859
860             !   [here, and in the gas-phase part, "de" contains -(1/r)(dE/dr)]
861
862   #ifdef LES
863             temp6 = -qiqj*temp4*(dl + fgbk)*scalefac(k)*lesscalefac(k)
864   #else
865             temp6 = -qiqj*temp4*(dl + fgbk)
866   #endif
867
868             ! -qiqj/fij^3*[1/Ein - e(-Kfij)/Eout) -kappa*fij*
869             ! exp(-kappa*fij)(1 + fij*a*b/A ) /Eout]
870
871             temp1 = vectmp1(k) !Exp(-rij^2/[4*ai*aj])
872             de = temp6*(one - fourth*temp1)
873
874             rj = rjx(k)
875   #ifdef LES
876             ! modified to use rix vector entry instead of ri, since i
877             !has multiple reff
878             temp5 = half*temp1*temp6*(rix(k)*rj + fourth*r2)
879   #else
880             temp5 = half*temp1*temp6*(ri*rj + fourth*r2)
881   #endif
882
```

```
#ifdef LES
            ! use iridx and jridx to get the pointers into the longer
            ! sumdeijda arrays.  Need this for LES since we need to know
            ! which of i's reff this j uses

            ! check to see if the forces need to be spread across multiple
            ! reff for this atom pair. This would happen if they are both
            !non-LES and only a single reff pair was used

            if (spreadfrc(k)) then

               ! distribute the sumdeijda over multiple reff for these atoms
               ! iridx and jridx will be pointing to the first reff for each
atom

               addi = rix(k)*temp5*lfaci
               addj = rj*temp5*lfaci

               do icopy=0,ncopy-1
                  sumdeijda(iridx(k)+icopy) = sumdeijda(iridx(k)+icopy) + addi
                  sumdeijda(jridx(k)+icopy) = sumdeijda(jridx(k)+icopy) + addj
               enddo

            else

               ! only use this for this particular set of reff

               sumdeijda(iridx(k)) = sumdeijda(iridx(k)) + rix(k)*temp5
               sumdeijda(jridx(k)) = sumdeijda(jridx(k)) + rj*temp5
            endif

#else
            sumdeijda(i) = sumdeijda(i) + ri*temp5
            sumdeijda(j) = sumdeijda(j) + rj*temp5
#endif


         !    -- skip exclusions for remaining terms:
         else
           de = zero
         end if !if( igb/=6 )

#ifdef LES
         ! LES uses longskipv, which is for the icount atoms, and
         ! is T for the excluded atoms as well as the icount pairs
         ! that are not the first in the loop over reff pairs for i,j
         ! note that the index for longskipv is not the same as for skipv
         ! since we need multiple values for each i,j pair

         if( .not. longskipv(k) ) then
#else
         if( .not. skipv(j) ) then
#endif

            !   -- gas-phase Coulomb energy:

            ! Note: With QMMM we don't need to explicitly exclude QM-MM
            !       interactions here since the QM charges should
            !       be zero.  Unless qmgb/=0 in which case they are not
            !       zero so we need to exclude them.
            ! We also need to exclude the QM link atom interactions.

            doeel = .true.
            dovdw = .true.
            !  check if i or j is a quantum atom:
            if ( qmmm_nml%ifqnt ) then
```



```
               if ( qmmm_struct%atom_mask(i) .OR. qmmm_struct%atom_mask(j) )
then
                  doeel = .false.
               end if

               ! if i or j is an MM link pair atom we also need to exclude
it.
               ! We also need to skip the VDW term if either is a mm link
               ! pair atom
               if (qmmm_struct%mm_link_mask(i) .or.
qmmm_struct%mm_link_mask(j)) then
                 doeel = .false.
                 dovdw = .false.
               end if
            end if

            !we can use the cached values.
            rinv = vectmp5(k) !1/rij
            r2inv = rinv*rinv

            if( doeel ) then

               ! A future fix implemented by Dan Parkin from Waseda Uni.
               ! intdieli is not 1/Ein when alpb=1
               ! intdiel                : Ein
               ! intdieli (alpb==1)     : one/(intdiel*(one + alpb_beta))
               ! intdieli (alpb/=1)     : one/intdiel
               ! intdiele_inv (always) : one/intdiel [newly added variable]
#ifdef LES
               temp1 = intdieli*rinv*lesscalefac(k) !comment out to
implement Dan Parkin's fix
!              temp1 = intdiele_inv*rinv*lesscalefac(k) !add to implement
Dan Parkin's fix
#else
               temp1 = intdieli*rinv !comment out to implement Dan Parkin's
fix
!              temp1 = intdiele_inv*rinv !add to implement Dan Parkin's fix
#endif
               eel = qiqj*temp1
               if ( ifcr /= 0 ) then
                  call cr_add_dcdr_factor( i, temp1*charge(j) )
                  call cr_add_dcdr_factor( j, temp1*qi )
               end if
               deel=eel*r2inv
            else
               eel = zero
               deel = zero
            end if

            eelt = eelt + eel
#ifdef LES
            nrg_ele_tmp = nrg_ele_tmp + eel
#endif
            if(idecomp == 1 .or. idecomp == 2) then
               call decpair(2,i,j,eel)
            else if(idecomp == 3 .or. idecomp == 4) then
               call decpair(-2,i,j,eel)
            end if
            de = de + deel
            if (oncpstep .or. oncestep) then
               dvdl = dvdl + (intdieli*rinv*dcharge(i) &
                     *dcharge(j) - eel)

            end if

            !    -- van der Waals energy:
```

```
1008
1009              ic = ico( iaci + iac(j) )
1010              if( ic > 0 .and. dovdw ) then
1011                 !                                      6-12 potential:
1012                 r6inv = r2inv*r2inv*r2inv
1013  #ifdef LES
1014                 ! scale LES:LES interactions
1015                 f6 = cn2(ic)*r6inv* lesscalefac(k)
1016                 f12 = cn1(ic)*(r6inv*r6inv)* lesscalefac(k)
1017  #else
1018                 f6 = cn2(ic)*r6inv
1019                 f12 = cn1(ic)*(r6inv*r6inv)
1020  #endif /*LES*/
1021                 evdw = evdw + (f12 - f6)
1022                 if(idecomp == 1 .or. idecomp == 2) then
1023                    call decpair(3,i,j,f12-f6)
1024                 else if(idecomp == 3 .or. idecomp == 4) then
1025                    call decpair(-3,i,j,f12-f6)
1026                 end if
1027                 de = de + (twelve*f12 - six*f6)*r2inv
1028
1029  #ifdef HAS_10_12
1030
1031                 !    ---The following could be commented out if the Cornell
1032                 !       et al. force field was always used, since then all
     hbond
1033                 !       terms are zero.
1034
1035              else
1036                 !                                      10-12 potential:
1037                 r10inv = r2inv*r2inv*r2inv*r2inv*r2inv
1038                 f10 = bsol(-ic)*r10inv
1039                 f12 = asol(-ic)*r10inv*r2inv
1040                 evdw = evdw + f12 -f10
1041                 if(idecomp == 3 .or. idecomp == 2) then
1042                    call decpair(1,i,j,f12-f10)
1043                 else if(idecomp == 3 .or. idecomp == 4) then
1044                    call decpair(-1,i,j,f12-f10)
1045                 end if
1046                 de = de + (twelve*f12 - ten*f10)*r2inv
1047  #endif
1048
1049              end if  ! ( ic > 0 )
1050  #ifdef MPI
1051  #  ifdef LES
1052              if(rem == 2) then
1053                if(cnum(i) > 0 .or. cnum(j) > 0) then
1054                  elesp = elesp + eel + f12 - f6
1055                end if
1056              end if
1057  #  endif
1058  #endif
1059           end if  ! ( .not. skipv(j) )
1060
1061           !    -- derivatives:
1062           if( onstep .and. r2 > cut_inner ) then
1063              de = de*nrespa
1064           else
1065              de = de*nrespai
1066           end if
1067           dedx = de * xij
1068           dedy = de * yij
1069           dedz = de * zij
1070           dumx = dumx + dedx
1071           dumy = dumy + dedy
1072           dumz = dumz + dedz
```



```
1073
1074           if (qmmm_nml%ifqnt) then
1075             !Currently if j is a link atom then
1076             !dedxy,y,z are force on the link atom, not on the MM link pair.
1077             !we have to use the chain rule to put the forces back onto the MM
     link
1078             !pair atom.
1079             !For the moment we just accumulate the forces on the end of dxyzqm
1080             !this should have been zeroed at the beginning of this routine.
1081             if (qmmm_struct%mm_link_mask(j)) then
1082               !j is a link atom, instead of adding it to the main force
1083               !array, accumulate it in dxyzqm
1084               !Find the link id for this j
1085               do qm_temp_count = 1,qmmm_struct%nlink
1086                 if (qmmm_struct%link_pairs(1,qm_temp_count) == j) exit
1087               end do
1088               qm_temp_count = qm_temp_count + qmmm_struct%nquant
1089               qmmm_struct%dxyzqm(1,qm_temp_count) =
     qmmm_struct%dxyzqm(1,qm_temp_count) + dedx
1090               qmmm_struct%dxyzqm(2,qm_temp_count) =
     qmmm_struct%dxyzqm(2,qm_temp_count) + dedy
1091               qmmm_struct%dxyzqm(3,qm_temp_count) =
     qmmm_struct%dxyzqm(3,qm_temp_count) + dedz
1092             else
1093               !Not a link atom, can just add to main force array.
1094               f(3*j-2) = f(3*j-2) - dedx
1095               f(3*j-1) = f(3*j-1) - dedy
1096               f(3*j  ) = f(3*j  ) - dedz
1097             end if
1098           else
1099             f(3*j-2) = f(3*j-2) - dedx
1100             f(3*j-1) = f(3*j-1) - dedy
1101             f(3*j  ) = f(3*j  ) - dedz
1102           end if
1103        end do !k=1,icount
1104
1105  #ifdef LES
1106        if(ipimd>0) then
1107           nrg_all(icnum) = nrg_all(icnum) + nrg_vdw_tmp
1108           nrg_all(icnum) = nrg_all(icnum) + nrg_ele_tmp
1109           nrg_all(icnum) = nrg_all(icnum) + nrg_egb_tmp
1110        endif
1111  #endif
1112
1113    !---- End first outer loop ----
1114
1115        if (qmmm_nml%ifqnt) then
1116          if (qmmm_struct%mm_link_mask(i)) then
1117            do qm_temp_count = 1,qmmm_struct%nlink
1118              if (qmmm_struct%link_pairs(1,qm_temp_count) == i) exit
1119            end do
1120            qm_temp_count = qm_temp_count + qmmm_struct%nquant
1121            qmmm_struct%dxyzqm(1,qm_temp_count) =
     qmmm_struct%dxyzqm(1,qm_temp_count) - dumx
1122            qmmm_struct%dxyzqm(2,qm_temp_count) =
     qmmm_struct%dxyzqm(2,qm_temp_count) - dumy
1123            qmmm_struct%dxyzqm(3,qm_temp_count) =
     qmmm_struct%dxyzqm(3,qm_temp_count) - dumz
1124          else
1125            !Not a link atom, can just add to main force array.
1126            f(3*i-2) = f(3*i-2) + dumx
1127            f(3*i-1) = f(3*i-1) + dumy
1128            f(3*i  ) = f(3*i  ) + dumz
1129          end if
1130        else
1131          f(3*i-2) = f(3*i-2) + dumx
```

```
1132            f(3*i-1) = f(3*i-1) + dumy
1133            f(3*i  ) = f(3*i  ) + dumz
1134         end if
1135   #ifdef MPI
1136         do k=i,(min(i+numtasks-1,natom))
1137            iexcl = iexcl + numex(k)
1138         end do
1139   #else
1140         iexcl = iexcl + numex(i)
1141   #endif
1142      end do  !  i=1,maxi
1143      call timer_stop(TIME_GBFRC)
1144
1145      if( igb==6 ) goto VACUUM3
1146
1147      !------------------------------------------------------------------------
1148      !
1149      !    Step 3:  Finally, do the reduction over the sumdeijda terms:, adding
1150      !             into the forces those terms that involve derivatives of
1151      !             the GB terms (including the diagonal or "self" terms) with
1152      !             respect to the effective radii.  This is done by computing
1153      !             the vector dai/dxj, and using the chain rule with the
1154      !             previously-computed sumdeijda vector.
1155      !
1156      !             Also, compute a surface-area dependent term if igbsa=1
1157      !
1158      !             Do these terms only at "nrespa" multiple-time step
     intervals;
1159      !             (when igb=2 or 5, one may need to do this at every step)
1160      !
1161      !------------------------------------------------------------------------
1162
1163      if( onstep ) then
1164         count=0
1165         frespa = nrespa
1166   #ifdef MPI
1167
1168         !       -- first, collect all the sumdeijda terms:
1169
1170         call timer_start(TIME_GBRADDIST)
1171         if( numtasks > 1 ) then
1172   #ifdef LES
1173            k=natom*ncopy
1174   #else
1175            k=natom
1176   #endif
1177            ! carlos changed this to use k as set above, not natom
1178   #ifdef USE_MPI_IN_PLACE
1179            call mpi_allreduce(MPI_IN_PLACE,sumdeijda,k, &
1180                 MPI_DOUBLE_PRECISION,mpi_sum,commsander,ierr)
1181   #else
1182            call mpi_allreduce(sumdeijda,vectmp1,k, &
1183                 MPI_DOUBLE_PRECISION,mpi_sum,commsander,ierr)
1184            sumdeijda(1:k) = vectmp1(1:k)
1185   #endif
1186         end if
1187         call timer_stop(TIME_GBRADDIST)
1188   #endif
1189         call timer_start(TIME_GBRAD2)
1190
1191         !  -- diagonal epol term, plus off-diag derivs wrt alpha == reff^-1:
1192
1193   #ifdef MPI
1194
1195         do i=mpistart,maxi,numtasks
1196   #else
```



```
1197         do i=1,maxi
1198   #endif
1199
1200            f_xi = zero
1201            f_yi = zero
1202            f_zi = zero
1203            qi = charge(i)
1204   #ifdef LES
1205            icnum = cnum(i)
1206            nrg_egb_tmp = zero
1207   #endif
1208
1209   #ifdef LES
1210            ! here we need to loop over possible multiple effective radii for i
1211            ! and average the energy/force from each reff
1212
1213            self_e = zero
1214
1215            icnum = cnum(i)
1216            istrt = ncopy * (i-1)
1217            if (icnum.eq.0) then
1218
1219              ! non-LES atom, average over multiple reff by dividng by lfac
1220
1221              do k=1,ncopy
1222                 expmkf = exp( -kappa * reff(istrt+k) )*extdieli
1223                 dl = intdieli - expmkf
1224                 qi2h = half*qi*qi
1225                 qid2h = qi2h*dl
1226                 temp1 = (onereff(istrt+k) + one_Arad_beta) *lfaci
1227                 self_e = self_e + qid2h*temp1
1228                 if ( ifcr /= 0 ) &
1229                    call cr_add_dcdr_factor( i, -qi*temp1*dl )
1230
1231                 ! add in contribution with scaling factor for average over
1232                 ! reff : DECREASE using lfaci
1233
1234                 vtemp7(k) = - sumdeijda(istrt+k) &
1235                   + qid2h*lfaci - kappa*qi2h*lfaci*expmkf*reff(istrt+k)* &
1236                    (one+one_Arad_beta*reff(i))
1237              enddo
1238            else
1239
1240               ! LES atom: charge is scaled by N copies, so q^2 is scaled N^2.
1241               ! We need to scale UP by ncopy;  doesn't matter what copy # it
1242               ! is for scaling factor, but copy # determines reff index
1243
1244               expmkf = exp( -kappa * reff(istrt+icnum) )*extdieli
1245               dl = intdieli - expmkf
1246               qi2h = half*qi*qi
1247               qid2h = qi2h*dl
1248               temp1 = (onereff(istrt+icnum) + one_Arad_beta) *lfac
1249               self_e =  qid2h*temp1
1250               if ( ifcr /= 0 ) &
1251                  call cr_add_dcdr_factor( i, -qi*temp1*dl )
1252
1253               ! add in contribution with scaling factor for q^2 :
1254               ! INCREASE using lfac
1255
1256               vtemp7(icnum) = - sumdeijda(istrt+icnum) &
1257                 + qid2h*lfac - kappa*qi2h * lfac *expmkf*reff(istrt+icnum)* &
1258                  (one+one_Arad_beta*reff(i))
1259            endif
1260
1261   #else /*not LES*/
1262
```

```
1263           expmkf = exp( -kappa * reff(i) )*extdieli
1264           dl = intdieli - expmkf
1265           qi2h = half*qi*qi
1266           qid2h = qi2h * dl
1267
1268           temp1 = (onereff(i) + one_Arad_beta)
1269           self_e = qid2h*temp1
1270           if ( ifcr /= 0 ) &
1271              call cr_add_dcdr_factor( i, -qi*temp1*dl )
1272
1273           temp7 = -sumdeijda(i) + qid2h - &
1274                 kappa*qi2h*expmkf*reff(i)*(one+one_Arad_beta*reff(i))
1275   #endif /*LES*/
1276
1277           !Ross Walker
1278           epol = epol - self_e
1279
1280   #ifdef LES
1281           nrg_egb_tmp = nrg_egb_tmp - self_e
1282   #endif
1283           if(idecomp == 1 .or. idecomp == 2) then
1284              call decpair(1,i,i,-qid2h*onereff(i))
1285           else if(idecomp == 3 .or. idecomp == 4) then
1286              call decpair(-1,i,i,-qid2h*onereff(i))
1287           end if
1288   #ifdef MPI
1289   #  ifdef LES
1290           ! local REMD
1291           if(rem == 2) then
1292             if(cnum(i) > 0) then
1293               elesp = elesp - qid2h*onereff(i)
1294               if ( ifcr /= 0 ) &
1295                  call cr_add_dcdr_factor( i, -qi*dl*onereff(i) )
1296             endif
1297           endif
1298   #  endif
1299   #endif
1300           if(oncpstep .or. oncestep) then
1301              dvdl = dvdl - (half*dl*dcharge(i)*dcharge(i)-qid2h)*onereff(i)
1302           end if
1303
1304           ! qi2h = qiqj/2
1305           ! qid2h = qiqj/2*[1/Ein - exp[-kappa*effbornrad]/Eout]
1306           ! temp7 = ... + qiqj/2*[1/Ein - exp[-kappa*effbornrad]/Eout]
1307           !         -kappa*qiqj/2*exp[-kappa*effbornrad]/Eout*effbornrad
1308           ! temp7 without the -sumdeijda part is the diagonal gradient.
1309
1310           ! carlos: moved temp7 calculation up into the LES region above
1311
1312           xi = x(3*i-2)
1313           yi = x(3*i-1)
1314           zi = x(3*i)
1315           ri = rborn(i)-offset
1316           ri1i = one/ri
1317           iaci = ntypes * (iac(i) - 1)
1318
1319           if( igb == 2 .or. igb == 5 .or. igb == 7 .or. igb == 8) then
1320
1321              !  --- new onufriev: we have to later scale values by a
1322              !      alpha,beta,gamma -dependent factor:
1323
1324              rborn_i = rborn(i)
1325              ri = rborn_i - offset
1326
1327   #ifdef LES
1328              if (icnum == 0) then
```



```
1329                 if (nradii(i) == 1) then
1330
1331                    ! get single reff and calc like non-les code, set flag
1332                    ! for distributing sumdei again?
1333
1334                    spread = .true.
1335                    psi_i = psi( (i-1)*ncopy + 1 )
1336                    thi = tanh( (gbalpha + gbgamma*psi_i*psi_i -
     gbbeta*psi_i )*psi_i )
1337                    thi2 = (gbalpha + three*gbgamma*psi_i*psi_i &
1338                       - two*gbbeta*psi_i )*(one - thi*thi)*ri/rborn_i
1339                 else
1340
1341                    ! loop over multiple psi values, store vector of thi2
     values
1342
1343                    do k=1,ncopy
1344                       psi_i = psi( (i-1)*ncopy + k )
1345                       thi = tanh( (gbalpha + gbgamma*psi_i*psi_i -
     gbbeta*psi_i )*psi_i )
1346
1347                       ! key change, thi2 is a vector due to multiple psi and
1348                       ! needing the thi2 below
1349
1350                       vthi2(k) = (gbalpha + three*gbgamma*psi_i*psi_i &
1351                       - two*gbbeta*psi_i )*(one - thi*thi)*ri/rborn_i
1352                       spread = .false.
1353                    enddo
1354                 endif
1355
1356              else !cnum>0, LES atom
1357
1358                    ! use the icnum psi value
1359
1360                    spread = .false.
1361                    psi_i = psi( (i-1)*ncopy + icnum )
1362                    thi = tanh( (gbalpha + gbgamma*psi_i*psi_i -
     gbbeta*psi_i )*psi_i )
1363                    thi2 = (gbalpha + three*gbgamma*psi_i*psi_i &
1364                       - two*gbbeta*psi_i )*(one - thi*thi)*ri/rborn_i
1365              endif
1366   #else
1367
1368           psi_i = psi(i)
1369           gba_i = gbvalpha(i) ! take gbalpha_i,gbbeta_i,gbgamma_i
1370           gbb_i = gbvbeta(i)
1371           gbg_i = gbvgamma(i)
1372           thi   = tanh( (gba_i + gbg_i*psi_i*psi_i - gbb_i*psi_i )*psi_i )
1373           thi2  = (gba_i + three*gbg_i*psi_i*psi_i - &
1374                     two*gbb_i*psi_i )*(one - thi*thi)*ri/rborn_i
1375   #endif
1376            end if
1377
1378            icount = 0
1379            do j=1,natom
1380               if( i /= j ) then
1381   #ifdef LES
1382                  jcnum = cnum(j)
1383
1384                  ! skip if both LES but not same copy
1385                  if((icnum.ne.0.and.jcnum.ne.0).and.icnum.ne.jcnum) cycle
1386
1387   #endif
1388
1389                   xij = xi - x(3*j-2)
1390                   yij = yi - x(3*j-1)
```

```
1391                 zij = zi - x(3*j  )
1392                 r2 = xij*xij + yij*yij + zij*zij
1393                 if( r2 <= rgbmaxpsmax2 ) then
1394                    ! pairlist contains only atoms within rgbmax + safety
     margin
1395                      icount = icount + 1
1396                       temp_jj(icount) = j
1397                       r2x(icount) = r2
1398                 end if ! r2 <= rgbmaxpsmax2
1399              end if !i/=j
1400           end do
1401           call vdinvsqrt( icount, r2x, vectmp1 )
1402
1403           kk1 = 0
1404           do k=1,icount
1405              j = temp_jj(k)
1406              r2 = r2x(k)
1407              sj =  fs(j)
1408
1409              dij1i = vectmp1(k)
1410              dij = r2*dij1i
1411              sj2 = sj * sj
1412
1413              if( dij <= four*sj ) then
1414                kk1 = kk1 + 1
1415                vectmp3(kk1) = dij + sj
1416                if( dij > ri+sj ) then
1417                   vectmp2(kk1) = r2 - sj2
1418                   vectmp4(kk1) = dij - sj
1419                else if ( dij > abs(ri-sj) ) then
1420                   vectmp2(kk1) = dij + sj
1421                   vectmp4(kk1) = ri
1422                else if ( ri < sj ) then
1423                   vectmp2(kk1) = r2 - sj2
1424                   vectmp4(kk1) = sj - dij
1425                else
1426                   vectmp2(kk1) = one
1427                   vectmp4(kk1) = one
1428                end if
1429              end if !dij <= four*sj
1430           end do
1431
1432           call vdinv( kk1, vectmp2, vectmp2 )
1433           call vdinv( kk1, vectmp3, vectmp3 )
1434           vectmp4(1:kk1) = vectmp4(1:kk1)*vectmp3(1:kk1)
1435           call vdln( kk1, vectmp4, vectmp4 )
1436
1437           kk1 = 0
1438           do k=1,icount
1439              j = temp_jj(k)
1440              j3 = 3*j
1441              r2 = r2x(k)
1442              xij = xi - x(j3-2)
1443              yij = yi - x(j3-1)
1444              zij = zi - x(j3  )
1445
1446              dij1i = vectmp1(k)
1447              dij = r2*dij1i
1448              sj =  fs(j)
1449              if (dij <= rgbmax +sj) then
1450                sj2 = sj * sj
1451
1452                !           datmp will hold (1/r)(dai/dr):
1453
1454                dij2i = dij1i*dij1i
1455                dij3i = dij2i*dij1i
```



```
1456
1457                if (dij > rgbmax - sj ) then
1458
1459                   temp1 = 1.0d0/(dij-sj)
1460                   datmp = eighth * dij3i * ((r2 + sj2) * &
1461                           (temp1*temp1 - rgbmax2i) - two * log(rgbmax*temp1))
1462
1463                else if( dij > four*sj ) then
1464
1465                   tmpsd = sj2*dij2i
1466                   dumbo = te+tmpsd* (tf+tmpsd* (tg+tmpsd* (th+tmpsd* thh)))
1467                   datmp = tmpsd*sj*dij2i*dij2i*dumbo
1468
1469                   !     ---check accuracy of above Taylor series:
1470                   !      kk1 = kk1 + 1
1471                   !      datmp2 = vectmp2(kk1)*sj*(-Half*dij2i +
     vectmp2(kk1)) +
1472                   !               Fourth*dij3i*vectmp4(kk1)
1473                   !      if( abs( datmp/datmp2 - 1.d0 ) .gt. 0.00001) &
1474                   !             write(6,*) i,j, datmp, datmp2
1475
1476                else if( dij > ri+sj ) then
1477
1478                   kk1 = kk1 + 1
1479                   datmp = vectmp2(kk1)*sj*(-half*dij2i + vectmp2(kk1)) + &
1480                         fourth*dij3i*vectmp4(kk1)
1481
1482                else if ( dij > abs(ri-sj) ) then
1483                   kk1 = kk1 + 1
1484                   datmp = -fourth*(-half*(r2 - ri*ri + sj2)*dij3i*ri1i*ri1i &
1485                         + dij1i*vectmp2(kk1)*(vectmp2(kk1) - dij1i) &
1486                         - dij3i*vectmp4(kk1) )
1487
1488                else if ( ri < sj ) then
1489                   kk1 = kk1 + 1
1490                   datmp = -half*(sj*dij2i*vectmp2(kk1) &
1491                         - two*sj*vectmp2(kk1)*vectmp2(kk1) &
1492                         - half*dij3i*vectmp4(kk1) )
1493
1494                else
1495                   kk1 = kk1 + 1
1496                   datmp = zero
1497                end if  ! ( dij > 4.d0*sj )
1498
1499                if( (igb == 7 .or. igb ==8) .and. dij < rborn(i) +rborn(j) +
     GBNECKCUT) then
1500
1501                   ! (no changes needed for LES)
1502
1503                   ! Derivative of neck with respect to dij is:
1504                   !                5
1505                   !            9 mdist
1506                   !   (2 mdist + --------) neckMaxVal gbneckscale
1507                   !                 5
1508                   ! -(-----------------------)
1509                   !                      6
1510                   !            2   3 mdist  2
1511                   !   (1 + mdist  + --------)
1512                   !                    10
1513
1514                   mdist = dij - neckMaxPos(neckidx(i),neckidx(j))
1515                   mdist2 = mdist * mdist
1516                   mdist3 = mdist2 * mdist
1517                   mdist5 = mdist2 * mdist3
1518                   mdist6 = mdist3 * mdist3
1519
```

```
1520                    ! temp1 will be divisor of above fraction * dij
1521                    ! (datmp is deriv * 1/r)
1522
1523                    temp1 = 1 + mdist2 + (three/ten)*mdist6
1524                    temp1 = temp1 * temp1 * dij
1525
1526                    ! (Note "+" means subtracting derivative, since above
1527                    !     expression has leading "-")
1528
1529                    datmp = datmp + ((2 * mdist + (nine/five) * mdist5) &
1530                         *neckMaxVal(neckidx(i),neckidx(j))*gbneckscale)/temp1
1531                 end if !f( (igb == 7 .or. igb == 8) .and. dij < rborn(i)
     +rborn(j) + GBNECKCUT)
1532
1533  #ifdef LES
1534                 ! loop over pairs of radii
1535                 ! SHOULD WE USE NRADII HERE?
1536
1537                 jcnum=cnum(j)
1538                 if( icnum == 0 ) then
1539                    ! i is not a LES copy
1540                    if( jcnum == 0 ) then
1541                       ! j not LES either, calculate force using all pairs of
     reff
1542                       idx1=1
1543                       idx2=ncopy
1544                    else
1545                       ! j is LES, add force using only one of the radii for
1546                       ! i and j (the one for j's cnum)
1547                       idx1=jcnum
1548                       idx2=jcnum
1549                    end if
1550                 else
1551                    ! i is LES, either j is not LES or is in same LES copy as
     i,
1552                    ! so add to force using i's cnum
1553                    idx1=icnum
1554                    idx2=icnum
1555                 endif
1556
1557                 ! save datmp since the loop modifies it
1558                 tmp=datmp
1559                 do icopy=idx1,idx2
1560
1561                    ! note that vtemp7 takes the place of temp7 used for non-
     LES
1562                    !simulations
1563                    datmp = -tmp*frespa*vtemp7(icopy)
1564
1565                    if( igb == 2 .or. igb == 5 .or. igb == 7 .or. igb == 8) then
1566
1567                       ! for nradii>1, we have to use the thi2 vector (vtemp7),
1568                       ! others have only 1 thi2 value
1569
1570                       if (nradii(i).gt.1) then
1571                            datmp = datmp*vthi2(icopy)
1572                       else
1573                            datmp = datmp*thi2
1574                       endif
1575                    endif
1576
1577                    f_x = xij*datmp
1578                    f_y = yij*datmp
1579                    f_z = zij*datmp
1580                    f(j3-2) = f(j3-2) + f_x
1581                    f(j3-1) = f(j3-1) + f_y
```



```
1582                    f(j3  ) = f(j3  ) + f_z
1583                    f_xi = f_xi - f_x
1584                    f_yi = f_yi - f_y
1585                    f_zi = f_zi - f_z
1586                 enddo
1587  #else /* not LES */
1588
1589                 datmp = -datmp*frespa*temp7
1590
1591                 if( igb == 2 .or. igb == 5 .or. igb == 7 .or. igb == 8) datmp
     = datmp*thi2
1592                 f_x = xij*datmp
1593                 f_y = yij*datmp
1594                 f_z = zij*datmp
1595                 if (qmmm_nml%ifqnt) then
1596                   if (qmmm_struct%mm_link_mask(j)) then
1597                     do qm_temp_count = 1,qmmm_struct%nlink
1598                       if (qmmm_struct%link_pairs(1,qm_temp_count) == j) exit
1599                     end do
1600                     qm_temp_count = qm_temp_count + qmmm_struct%nquant
1601                     qmmm_struct%dxyzqm(1,qm_temp_count) =
     qmmm_struct%dxyzqm(1,qm_temp_count) - f_x
1602                     qmmm_struct%dxyzqm(2,qm_temp_count) =
     qmmm_struct%dxyzqm(2,qm_temp_count) - f_y
1603                     qmmm_struct%dxyzqm(3,qm_temp_count) =
     qmmm_struct%dxyzqm(3,qm_temp_count) - f_z
1604                   else
1605                     !Not a link atom, can just add to main force array.
1606                     f(j3-2) = f(j3-2) + f_x
1607                     f(j3-1) = f(j3-1) + f_y
1608                     f(j3  ) = f(j3  ) + f_z
1609                   end if
1610                 else
1611                   f(j3-2) = f(j3-2) + f_x
1612                   f(j3-1) = f(j3-1) + f_y
1613                   f(j3  ) = f(j3  ) + f_z
1614                 end if
1615
1616                 f_xi = f_xi - f_x
1617                 f_yi = f_yi - f_y
1618                 f_zi = f_zi - f_z
1619
1620  #endif /*LES*/
1621
1622               end if ! (dij <= rgbmax +sj)
1623             end do  !  k=1,icount
1624
1625             if (qmmm_nml%ifqnt) then
1626               if (qmmm_struct%mm_link_mask(i)) then
1627                 do qm_temp_count = 1,qmmm_struct%nlink
1628                   if (qmmm_struct%link_pairs(1,qm_temp_count) == i) exit
1629                 end do
1630                 qm_temp_count = qm_temp_count + qmmm_struct%nquant
1631                 qmmm_struct%dxyzqm(1,qm_temp_count) =
     qmmm_struct%dxyzqm(1,qm_temp_count) - f_xi
1632                 qmmm_struct%dxyzqm(2,qm_temp_count) =
     qmmm_struct%dxyzqm(2,qm_temp_count) - f_yi
1633                 qmmm_struct%dxyzqm(3,qm_temp_count) =
     qmmm_struct%dxyzqm(3,qm_temp_count) - f_zi
1634               else
1635                 !Not a link atom, can just add to main force array.
1636                 f(3*i-2) = f(3*i-2) + f_xi
1637                 f(3*i-1) = f(3*i-1) + f_yi
1638                 f(3*i  ) = f(3*i  ) + f_zi
1639               end if
1640             else
```

```
1641              f(3*i-2) = f(3*i-2) + f_xi
1642              f(3*i-1) = f(3*i-1) + f_yi
1643              f(3*i  ) = f(3*i  ) + f_zi
1644           end if
1645
1646           !  --- Define neighbor list ineighbor for calc of LCPO areas ---
1647
1648           if ( gbsa > 0 ) then
1649              do k=1,icount
1650                 j = temp_jj(k)
1651                 dij = sqrt(r2x(k))
1652                 if ((vdwrad(i) + vdwrad(j)) > dij ) then
1653
1654                    ! Consider all atoms for icosa, only non-H's for LCPO:
1655
1656                    if ( (gbsa == 2) .or. &
1657                         (vdwrad(i) > 2.5).and.(vdwrad(j) > 2.5) ) then
1658                       count=count+1
1659                       ineighbor(count) = j
1660                    end if
1661                 end if
1662              end do
1663              count=count+1
1664              ineighbor(count)=0
1665           end if
1666
1667           if ( gbsa == 2 ) then
1668
1669              !  --- Calc surface area with icosasurf algo;
1670              !  does not provide forces ==> only for single point calculations
1671
1672   #ifdef MPI
1673              if(i == mpistart) then
1674                 ineighborpt = 1
1675                 call icosa_init(2, 3, zero)
1676              end if
1677
1678   #else
1679              if(i == 1) then
1680                 ineighborpt = 1
1681                 call icosa_init(2, 3, zero)
1682              end if
1683   #endif
1684
1685
1686            !  if(i == 1) then
1687            !     ineighborpt = 1
1688            !     call icosa_init(2, 3, zero)
1689            !  end if
1690              totsasa = totsasa + icosa_sphere_approx(i,x, &
1691                    vdwrad,ineighborpt,ineighbor,idecomp)
1692
1693           end if  !  ( gbsa == 2 )
1694
1695   #ifdef LES
1696         if(ipimd>0) nrg_all(icnum) = nrg_all(icnum) + nrg_egb_tmp
1697   #endif
1698
1699
1700         end do   ! end loop over atom i
1701
1702         call timer_stop(TIME_GBRAD2)
1703
1704         if ( gbsa == 1 ) then
1705
1706            !       --- calculate surface area by computing LCPO (Still) over
```



```
1707            !           all atoms ---
1708
1709   #        include "gbsa.h"
1710
1711         end if  !  ( gbsa == 1 )
1712         esurf = surften*totsasa
1713      end if  !  onstep
1714
1715
1716   VACUUM3 &
1717
1718   !======= QMMM ==========
1719      if (qmmm_nml%ifqnt) then
1720        call timer_start(TIME_QMMM)
1721        if (qmmm_nml%qmgb /= 0) then
1722
1723           ! We filled the main charge array with charges for QM atoms; make
      sure
1724           ! we zero it again so the 1-4's which are done outside of EGB will
      be
1725           ! correctly skipped for QM-MM on the next step. Note here we don't
      have the routine
1726           ! save the charges again since depending on the GB option they may
      have been filled
1727           ! with either RESP charges or mulliken charges.
1728           call qm_zero_charges(charge,qmmm_struct%scaled_mm_charges,.false.)
1729           if (qmmm_struct%nlink > 0 ) then
1730              !Don't save the charges here since they could be the resp charges.
1731              call
      qm_zero_mm_link_pair_main_chg(qmmm_struct%nlink,qmmm_struct%link_pairs,charge,
      &
1732
      qmmm_struct%scaled_mm_charges,.false.)
1733           end if
1734        end if
1735
1736        call timer_start(TIME_QMMMCOLLATEF)
1737        !We need to restore the MM link pair coordinates and then
1738        !Use the chain rule to put the link pair forces back onto the
1739        !QM and MM link pairs.
1740        call rst_mm_link_pair_crd(x)
1741        do i=1,qmmm_struct%nlink
1742          mm_no = 3*qmmm_struct%link_pairs(1,i)-2  !location of atom in x array
1743          lnk_no = qmmm_struct%link_pairs(2,i) !Nquant number of QM atom bound
      to link atom
1744          qm_no = 3*qmmm_struct%iqmatoms(lnk_no)-2
1745          !Note this routine uses the flink in the form -flink.
1746          call
      distribute_lnk_f(forcemod,qmmm_struct%dxyzqm(1:3,qmmm_struct%nquant+i),x(mm_no),
      &
1747                                  x(qm_no),qmmm_nml%lnk_dis)
1748
1749          !NOTE: forces are reversed in QM calc with respect to amber force
      array
1750          !so we subtract forcemod from MM atom and add it to QM atom.
1751          j = (qmmm_struct%link_pairs(1,i)-1)*3 !Natom number of MM link pair.
1752          !MM atom's new force = FMM(x,y,z) - FORCEMOD(x,y,z)
1753          f(j+1) = f(j+1) - forcemod(1)
1754          f(j+2) = f(j+2) - forcemod(2)
1755          f(j+3) = f(j+3) - forcemod(3)
1756
1757          j = (qmmm_struct%iqmatoms(lnk_no)-1)*3
1758          !QM atom's new force = FQM(x,y,z) - Flink(x,y,z) + FORCEMOD(x,y,z)
1759          !Note QM forces should be subtracted from sander F array to leave
      total force.
1760          f(j+1) = f(j+1) - qmmm_struct%dxyzqm(1,qmmm_struct%nquant+i) +
```

```
      forcemod(1)
1761         f(j+2) = f(j+2) - qmmm_struct%dxyzqm(2,qmmm_struct%nquant+i) +
      forcemod(2)
1762         f(j+3) = f(j+3) - qmmm_struct%dxyzqm(3,qmmm_struct%nquant+i) +
      forcemod(3)
1763       end do
1764       call timer_stop_start(TIME_QMMMCOLLATEF,TIME_QMMMENERGY)
1765       call timer_start(TIME_QMMMGBENERGY)
1766       !Finally we need to calculate the VDW terms
1767       !for the MM link pair atoms as they were skipped above.
1768
1769  !------------ START MM LINK PAIR VDW ------------------
1770       do i = 1, qmmm_struct%nlink
1771         skipv(1:natom) = .false.
1772         jexcl = 1
1773         mm_no = qmmm_struct%link_pairs(1,i)
1774         iminus = mm_no-1
1775         do j = 1, iminus
1776           !As we loop over each atom check if this atom is supposed to
      exclude this MM
1777           !link pair atom.
1778           jexcl_last = jexcl + numex(j) -1
1779           do jjv = jexcl, jexcl_last
1780             iexcl = natex(jjv)
1781             !Should this atom exclude the MM atom?
1782             if (iexcl == mm_no) skipv(j) = .true.
1783           end do
1784           jexcl = jexcl + numex(j)
1785         end do
1786         jexcl_last = jexcl + numex(mm_no) -1
1787         do jjv = jexcl, jexcl_last
1788           skipv(natex(jjv)) = .true.
1789         end do
1790         !We need to avoid double counting so we should exclude ourself and
1791         !all link atoms of lower number
1792         !Note this assumes the link atom list is numerically sorted...
1793         do k = 1, i !To i ensures we skip the self interaction.
1794           skipv(qmmm_struct%link_pairs(1,k)) = .true.
1795         end do
1796         xi = x(3*mm_no-2)
1797         yi = x(3*mm_no-1)
1798         zi = x(3*mm_no  )
1799
1800         iaci = ntypes * (iac(mm_no) - 1)
1801  #ifdef LES
1802         icnum=cnum(mm_no)
1803         nrg_vdw_tmp = zero
1804  #endif
1805         dumx = zero
1806         dumy = zero
1807         dumz = zero
1808         de = zero
1809  #ifdef MPI
1810         do j=mpistart,maxi,numtasks
1811  #else
1812         do j=1,maxi !1 to natom
1813  #endif
1814           !Do all atoms that are not excluded and
1815           !are within the cutoff for this i. Skip other
1816           !MM interactions that are less than i to avoid double
1817           !counting.
1818           if ( .not. skipv(j) ) then
1819             xij = xi - x(3*j-2)
1820             yij = yi - x(3*j-1)
1821             zij = zi - x(3*j  )
1822             r2 = xij*xij + yij*yij + zij*zij
```



```
1823             if ( r2 <= cut ) then
1824               r2inv = one/r2
1825               ic = ico( iaci + iac(j) )
1826               if (ic > 0) then
1827                 !6-12 potential:
1828                 r6inv = r2inv*r2inv*r2inv
1829                 f6 = cn2(ic)*r6inv
1830                 f12 = cn1(ic)*(r6inv*r6inv)
1831                 evdw = evdw + (f12 - f6)
1832
1833                 if(idecomp == 1 .or. idecomp == 2) then
1834                   call decpair(3,i,j,f12-f6)
1835                 else if(idecomp == 3 .or. idecomp == 4) then
1836                   call decpair(-3,i,j,f12-f6)
1837                 end if
1838                 de = (twelve*f12 - six*f6)*r2inv
1839  #ifdef HAS_10_12
1840               else
1841                 !10-12 potential:
1842                 r10inv = r2inv*r2inv*r2inv*r2inv*r2inv
1843                 f10 = bsol(-ic)*r10inv
1844                 f12 = asol(-ic)*r10inv*r2inv
1845                 evdw = evdw + f12 -f10
1846                 if(idecomp == 3 .or. idecomp == 2) then
1847                   call decpair(1,i,j,f12-f10)
1848                 else if(idecomp == 3 .or. idecomp == 4) then
1849                   call decpair(-1,i,j,f12-f10)
1850                 end if
1851                 de = (twelve*f12 - ten*f10)*r2inv
1852  #endif
1853               end if !if ic>0
1854               de = de*nrespai
1855               dedx = de * xij
1856               dedy = de * yij
1857               dedz = de * zij
1858               dumx = dumx + dedx
1859               dumy = dumy + dedy
1860               dumz = dumz + dedz
1861               f(3*j-2) = f(3*j-2) - dedx
1862               f(3*j-1) = f(3*j-1) - dedy
1863               f(3*j  ) = f(3*j  ) - dedz
1864             end if !if r2<=cut
1865           end if !if not skipv(i)
1866         end do
1867
1868  #ifdef LES
1869         if(ipimd>0) nrg_all(icnum) = nrg_all(icnum) + nrg_vdw_tmp
1870  #endif
1871         f(3*mm_no-2) = f(3*mm_no-2) + dumx
1872         f(3*mm_no-1) = f(3*mm_no-1) + dumy
1873         f(3*mm_no  ) = f(3*mm_no  ) + dumz
1874       end do ! i = 1, qmmm_struct%nlink
1875       call timer_stop(TIME_QMMMGBENERGY)
1876       call timer_stop(TIME_QMMMENERGY)
1877  !------------ END MM LINK PAIR VDW ---------------------
1878       call timer_stop(TIME_QMMM)
1879     end if !if ifqnt.
1880  !======== END QMMM ==========
1881
1882     return
1883  end subroutine egb
1884
1885  subroutine egb_calc_radii(igb,natom,x,fs,reff,onereff,fsmax,rgbmax, &
1886                            rborn, offset,rbornstat, &
1887                            rbave,rbfluct,rbmax,rbmin, gbneckscale, ncopy,
      rdt, &
```

```
                           gbalpha,gbbeta,gbgamma &
#ifdef MPI
                           ,mpistart &
#endif
                          )

!don't have igb = 8 for sander.LES. Thus, instead of using gbavalpha,
!gbvbeta and gbvgamma array, I still keep original gbalpha, gbbeta, gbgamma
!(igb1,2,5,7) for LES part to get correct force

! Calculates effective GB radii and puts result in reff;
! onereff contains 1.0d0/reff.

  use constants, only : zero, eighth, fourth, third, half, &
                        one, two, three, four, five, seven, eight, nine, &
                        eleven, twelve, thirtieth
#ifdef LES
   use les_data, only : cnum
#endif

  implicit none

#include "gbneck.h"

#ifdef MPI
# include "parallel.h"
   include 'mpif.h'
  integer, intent(in) :: mpistart
  integer :: ierr
#endif

   ! Scratch variables used for calculating neck correction
   _REAL_ mdist,mdist2,mdist3,mdist6,neck

  integer, intent(in) :: igb, natom, rbornstat, ncopy
  _REAL_, intent(in) :: x(3*natom), fs(natom), rdt

#ifdef LES
   integer istrt,iend,k1
   integer jstrt
! copy number for atoms i and j
   integer icnum,jcnum
   _REAL_ rmin,rmax
  _REAL_, intent(out) :: reff(ncopy*natom), onereff(ncopy*natom)
#else
  _REAL_, intent(out) :: reff(natom), onereff(natom)
#endif

  _REAL_, intent(in) :: fsmax, rgbmax, gbneckscale
  _REAL_, intent(in) :: rborn(natom),offset

#ifdef LES
  _REAL_, intent(in) :: gbalpha, gbbeta, gbgamma
#else
  _REAL_, intent(in) :: gbalpha(natom), gbbeta(natom), gbgamma(natom)
  _REAL_ :: reff_i
  integer :: ncopy2
#endif
  _REAL_, intent(out) :: rbave(natom), rbfluct(natom),
rbmax(natom),rbmin(natom)

! Local:
  integer :: i, icount, iplus, kk1, kk2, k, j, vecend
  _REAL_ :: xi,yi,zi,xij, yij, zij,r2,ri,ri1i,rj1i,si,si2,sj, sj2
  _REAL_ :: dij1i, dij2i, dij, rj, uij, tmpsd, dumbo, theta
  _REAL_ :: rgbmax1i, rgbmax2i, rgbmaxpsmax2
```



```
  _REAL_ :: temp3,temp4

   !   FGB taylor coefficients follow
   !   from A to D :
   !   1/3 , 2/5 , 3/7 , 4/9 , 5/11
   _REAL_  ta
   _REAL_  tb
   _REAL_  tc
   _REAL_  td
   _REAL_  tdd
   parameter ( ta   = third )
   parameter ( tb   = two / five )
   parameter ( tc   = three / seven )
   parameter ( td   = four / nine )
   parameter ( tdd  = five / eleven )

   rgbmax1i = one/rgbmax
   rgbmax2i = rgbmax1i*rgbmax1i
   rgbmaxpsmax2 = (rgbmax+fsmax)**2

#ifdef LES
! need to expand range of onereff that are intialized
   onereff(1:natom*ncopy) = zero
! initialize rbornlong, see egb()
   k=0
   do i=1,natom
      do j=1,ncopy
         k=k+1
         rbornlong(k)=rborn(i)
      enddo
   enddo
#else
   onereff(1:natom) = zero
   ncopy2 = ncopy ! BUGBUG -- only here to suppress warnings
#endif

#ifdef MPI
   do i=mpistart,natom,numtasks
#else
   do i=1,natom
#endif
      xi = x(3*i-2)
      yi = x(3*i-1)
      zi = x(3*i)

#ifdef LES
! copy # for atom i
      icnum = cnum(i)

! due to the multiple reff we will not use reff_i but loop over array
directly
! pointers to entries in expanded reff array for atom i
! reff and onereff can't use atom index as pointer

      istrt = ncopy * (i-1)
#else
      reff_i = onereff(i)
#endif

#ifdef LES
! these next variables are not affected by LES (rborn, fs)
#endif

      ri = rborn(i)-offset
      ri1i = one/ri
      si = fs(i)
```

```
2018         si2 = si*si
2019
2020         !  Here, reff_i will sum the contributions to the inverse effective
2021         !  radius from all of the atoms surrounding atom "i"; later the
2022         !  inverse of its own intrinsic radius will be added in
2023
2024         icount = 0
2025         iplus=i+1
2026
2027         do j=iplus,natom
2028
2029   #ifdef LES
2030            jcnum = cnum(j)
2031            ! LES atoms in different copies do not descreen each other
2032            if((icnum.ne.0.and.jcnum.ne.0).and.icnum.ne.jcnum) cycle
2033   #endif
2034
2035            xij = xi - x(3*j-2)
2036            yij = yi - x(3*j-1)
2037            zij = zi - x(3*j  )
2038            r2 = xij*xij + yij*yij + zij*zij
2039
2040            if( r2 <= rgbmaxpsmax2 ) then
2041              icount = icount + 1
2042              temp_jj(icount) = j
2043              r2x(icount) = r2
2044            end if
2045
2046         end do
2047
2048         call vdinvsqrt( icount, r2x, vectmp1 )
2049
2050         kk1 = 0
2051         kk2 = 0
2052   !!!      !dir$ ivdep
2053         do k = 1, icount
2054
2055            j = temp_jj(k)
2056            r2 = r2x(k)
2057            sj =  fs(j)
2058
2059   !        don't fill the remaining vectmp arrays if atoms don't see each
         other:
2060            dij1i = vectmp1(k) !1/sqrt(r^2)
2061            dij = r2*dij1i != rij
2062            if (dij <= rgbmax+si .or. dij <= rgbmax+sj) then
2063              rj = rborn(j) - offset
2064
2065              if( dij <= four*sj) then
2066                 kk1 = kk1 + 1
2067                 vectmp2(kk1) = dij + sj
2068                 if( dij > ri+sj ) then
2069                    vectmp4(kk1) = dij - sj
2070                 else if ( dij > abs(ri-sj) ) then
2071                    vectmp4(kk1) = ri
2072                 else if ( ri < sj ) then
2073                    vectmp4(kk1) = sj - dij
2074                 else
2075                    vectmp4(kk1) = one
2076                 end if
2077              end if
2078
2079              if( dij <= four*si) then
2080                 kk2 = kk2 + 1
2081                 vectmp3(kk2) = dij + si
2082                 if( dij > rj+si) then
```



```
2083                    vectmp5(kk2) = dij - si
2084                 else if ( dij > abs(rj-si) ) then
2085                    vectmp5(kk2) = rj
2086                 else if ( rj < si ) then
2087                    vectmp5(kk2) = si - dij
2088                 else
2089                    vectmp5(kk2) = one
2090                 end if
2091              end if
2092            end if ! (dij <= rgbmax+si .or. dij <= rgbmax+sj)
2093         end do  !  k = 1, icount
2094
2095   #ifdef LES
2096   ! these arrays for vector calls are not changed by LES
2097   #endif
2098
2099         call vdinv( kk1, vectmp2, vectmp2 )
2100         call vdinv( kk2, vectmp3, vectmp3 )
2101         vectmp4(1:kk1) = vectmp2(1:kk1)*vectmp4(1:kk1)
2102         vectmp5(1:kk2) = vectmp3(1:kk2)*vectmp5(1:kk2)
2103         call vdln( kk1, vectmp4, vectmp4 )
2104         call vdln( kk2, vectmp5, vectmp5 )
2105
2106         kk1 = 0
2107         kk2 = 0
2108         do k = 1, icount
2109
2110            j = temp_jj(k)
2111   #ifdef LES
2112            jstrt = ncopy * (j-1)
2113            jcnum=cnum(j)
2114   #endif
2115
2116            r2 = r2x(k)
2117
2118            rj = rborn(j) - offset
2119            rj1i = one/rj
2120            sj =  fs(j)
2121
2122            sj2 = sj * sj
2123
2124            xij = xi - x(3*j-2)
2125            yij = yi - x(3*j-1)
2126            zij = zi - x(3*j  )
2127
2128            dij1i = vectmp1(k)
2129            dij = r2*dij1i
2130
2131            temp3 = zero
2132            temp4 = zero
2133
2134            if (dij <= rgbmax + sj) then
2135
2136               if ((dij > rgbmax - sj)) then
2137                  uij = 1.0d0/(dij -sj)
2138
2139   ! carlos: store descreening contrib in temp3, this makes it easier to
2140   ! apply to multiple atoms for LES
2141
2142                  temp3 = temp3  - eighth * dij1i * (one + two * dij *uij + &
2143                        rgbmax2i * (r2 - four * rgbmax * dij - sj2) + &
2144                        two * log((dij-sj)*rgbmax1i))
2145
2146               else if( dij > four*sj ) then
2147
2148                  dij2i = dij1i*dij1i
```

```
2149                  tmpsd = sj2*dij2i
2150                  dumbo = ta+tmpsd* (tb+tmpsd* (tc+tmpsd* (td+tmpsd* tdd)))
2151
2152                  temp3 = temp3 - tmpsd*sj*dij2i*dumbo
2153
2154               !     ---following are from the Appendix of Schaefer and
     Froemmel,
2155               !        J. Mol. Biol. 216:1045-1066, 1990, divided by (4*Pi):
2156
2157               else if( dij > ri+sj ) then
2158
2159                  kk1 = kk1 + 1
2160                  temp3 = temp3 - half*( sj/(r2-sj2) + half*dij1i*vectmp4(kk1) )
2161
2162               !-----------------------------------------------------------------
2163
2164               else if ( dij > abs(ri-sj) ) then
2165
2166                  kk1 = kk1 + 1
2167                  theta = half*ri1i*dij1i*(r2 + ri*ri -sj2)
2168                  temp3 = temp3- fourth*( ri1i*(two-theta) &
2169                     - vectmp2(kk1) + dij1i*vectmp4(kk1) )
2170
2171               !-----------------------------------------------------------------
2172
2173               else if ( ri < sj ) then
2174                  kk1 = kk1 + 1
2175                  temp3 = temp3- half*sj/(r2-sj2) + ri1i    &
2176                     + fourth*dij1i*vectmp4(kk1)
2177
2178               !-----------------------------------------------------------------
2179
2180               else
2181                  kk1 = kk1 + 1
2182               end if  ! ( dij > 4.d0*sj )
2183
2184               if ( (igb == 7 .or. igb ==8) .and. dij < rborn(i) +rborn(j) +
     GBNECKCUT) then
2185                  mdist = dij - neckMaxPos(neckidx(i),neckidx(j))
2186                  mdist2 = mdist *mdist
2187                  mdist3 = mdist2 * mdist
2188                  mdist6 = mdist3 *mdist3
2189                  neck = neckMaxVal(neckidx(i),neckidx(j))/ &
2190                        (one + mdist2 +0.3d0*mdist6)
2191                  temp3 = temp3 - gbneckscale * neck
2192               end if
2193
2194            end if
2195
2196            !   --- Now the same thing, but swap i and j and use temp4 for
     descreening contrib:
2197
2198            if (dij <= rgbmax +si) then
2199
2200              if (dij > rgbmax - si) then
2201                 uij = 1.0d0/(dij -si)
2202                 temp4 = temp4 - eighth * dij1i * (one + two * dij *uij + &
2203                          rgbmax2i * (r2 - four * rgbmax * dij - si2) + &
2204                          two * log((dij-si)*rgbmax1i))
2205              else if( dij > four*si ) then
2206                 dij2i = dij1i*dij1i
2207                 tmpsd = si2*dij2i
2208                 dumbo = ta+tmpsd* (tb+tmpsd* (tc+tmpsd* (td+tmpsd* tdd)))
2209                 temp4 = temp4 - tmpsd*si*dij2i*dumbo
2210              else if( dij > rj+si ) then
2211                 kk2 = kk2 + 1
```



```
2212                  temp4 = temp4 - half*( si/(r2-si2) + &
2213                        half*dij1i*vectmp5(kk2) )
2214               !-----------------------------------------------------------------
2215               else if ( dij > abs(rj-si) ) then
2216                  kk2 = kk2 + 1
2217                  theta = half*rj1i*dij1i*(r2 + rj*rj -si2)
2218                  temp4 = temp4 - fourth*( rj1i*(two-theta) &
2219                        - vectmp3(kk2) + dij1i*vectmp5(kk2) )
2220               !-----------------------------------------------------------------
2221               else if ( rj < si ) then
2222                  kk2 = kk2 + 1
2223                  temp4 = temp4 - half*si/(r2-si2) + rj1i &
2224                        + fourth*dij1i*vectmp5(kk2)
2225               !-----------------------------------------------------------------
2226               else
2227                  kk2 = kk2 + 1
2228               end if  ! ( dij > 4.d0*si )
2229
2230               if ((igb == 7 .or. igb ==8) .and. dij < rborn(j) +rborn(i)
     +GBNECKCUT) then
2231                  mdist = dij - neckMaxPos(neckidx(j),neckidx(i))
2232                  mdist2 = mdist * mdist
2233                  mdist3 = mdist2 * mdist
2234                  mdist6 = mdist3 * mdist3
2235                  neck = neckMaxVal(neckidx(j),neckidx(i))/ &
2236                          (one + mdist2 +0.3d0*mdist6)
2237                  temp4 = temp4 - gbneckscale *neck
2238               end if
2239
2240            end if !(dij <= rgbmax +si)
2241   ! now add the calculated descreening component to onereff
2242
2243   #ifdef LES
2244
2245               ! recall that reff and onereff do not use atom index as pointer
2246               ! but instead we need istrt and jstrt
2247
2248               if( icnum == 0 ) then
2249
2250                  ! i is not a LES copy
2251
2252                  if( jcnum == 0 ) then
2253
2254                     ! j not LES either, add temp3 to all copies of i's reff
2255
2256                     do k1= 1,ncopy
2257
2258                        onereff(istrt+k1) = onereff(istrt+k1)+temp3
2259                        onereff(jstrt+k1) = onereff(jstrt+k1)+temp4
2260                     end do
2261                  else
2262
2263                     ! j is LES, add only to one of the radii for i and j
2264                     !  (the one for j's cnum)
2265
2266                     onereff(istrt+jcnum) = onereff(istrt+jcnum)+temp3
2267                     onereff(jstrt+jcnum) = onereff(jstrt+jcnum)+temp4
2268                  end if
2269
2270               else
2271
2272                  ! i is LES, either j is not LES or is in same LES copy as i,
2273                  ! so add to reff for i's cnum
2274
2275                  onereff(istrt+icnum) = onereff(istrt+icnum)+temp3
2276                  onereff(jstrt+icnum) = onereff(jstrt+icnum)+temp4
```

```
2277              endif
2278
2279  #else /* not LES */
2280
2281              reff_i = reff_i + temp3
2282              onereff(j) = onereff(j) + temp4
2283  #endif
2284
2285        end do                    !  k = 1, icount
2286
2287
2288        ! we are ending the do-i-loop, reassign the scalar to the original
      array:
2289
2290  #ifdef LES
2291        ! LES used onereff() directly so we don't need to reassign onereff(i)
2292        ! based on reff_i
2293  #else
2294        onereff(i) = reff_i
2295  #endif
2296
2297     end do  !  i = 1,natom
2298
2299  #ifdef MPI
2300     call timer_stop_start(TIME_GBRAD1,TIME_GBRADDIST)
2301
2302     !       collect the (inverse) effective radii from other nodes:
2303
2304     if( numtasks > 1 ) then
2305  #  ifdef LES
2306           ! LES has more reff
2307  #   ifdef USE_MPI_IN_PLACE
2308           call mpi_allreduce( MPI_IN_PLACE, onereff, ncopy*natom, &
2309              MPI_DOUBLE_PRECISION,mpi_sum,commsander,ierr)
2310  #   else
2311           call mpi_allreduce( onereff, vectmp1, ncopy*natom, &
2312              MPI_DOUBLE_PRECISION,mpi_sum,commsander,ierr)
2313           onereff(1:ncopy*natom) = vectmp1(1:ncopy*natom)
2314  #   endif
2315  #  else
2316  #   ifdef USE_MPI_IN_PLACE
2317           call mpi_allreduce( MPI_IN_PLACE, onereff, natom, &
2318              MPI_DOUBLE_PRECISION,mpi_sum,commsander,ierr)
2319  #   else
2320           call mpi_allreduce( onereff, vectmp1, natom, &
2321              MPI_DOUBLE_PRECISION,mpi_sum,commsander,ierr)
2322           onereff(1:natom) = vectmp1(1:natom)
2323  #   endif
2324  #  endif
2325
2326     end if
2327     call timer_stop_start(TIME_GBRADDIST,TIME_GBRAD1)
2328  #endif
2329
2330  ! set end variable since LES arrays are longer than normal
2331  ! this will make the changes for LES code simpler to understand and modify
2332
2333  #ifdef LES
2334     vecend = natom*ncopy
2335  #else
2336     vecend = natom
2337  #endif
2338
2339     if( igb == 2 .or. igb == 5 .or. igb == 7 .or. igb == 8) then
2340
2341        ! --- apply the new Onufriev "gbalpha, gbbeta, gbgamma" correction:
```



```
2342
2343
2344  #ifdef LES
2345  ! use rbornlong for these vectors, not rborn
2346        vectmp1(1:vecend) = rbornlong(1:vecend)
2347  #else
2348        vectmp1(1:vecend) = rborn(1:vecend)
2349  #endif
2350        vectmp2(1:vecend) = vectmp1(1:vecend)-offset
2351        call vdinv(vecend, vectmp1, vectmp1) !1.0d0/rborn
2352
2353  ! use of onereff here means that we need changes for LES to accomodate
      multiple onereff per atom
2354
2355        psi(1:vecend) = -vectmp2(1:vecend)*onereff(1:vecend)
2356        call vdinv(vecend, vectmp2, vectmp2) !1.0d0/(rborn-offset)
2357
2358  #ifdef LES
2359   !Using gbalpha, gbbeta, gbgamma
2360   !Not use igb=8 for LES
2361        vectmp3(1:vecend) = ((gbalpha+gbgamma*psi(1:vecend)*   &
2362                                psi(1:vecend)-
      gbbeta*psi(1:vecend))*psi(1:vecend))
2363  #else
2364    !if not LES: use gbalpha, gbgamma, gbbeta arrays instead
2365        vectmp3(1:vecend) = ((gbalpha(1:vecend)
      +gbgamma(1:vecend)*psi(1:vecend)*   &
2366                                psi(1:vecend)-
      gbbeta(1:vecend)*psi(1:vecend))*psi(1:vecend))
2367  #endif
2368        call vdtanh(vecend, vectmp3, vectmp3)
2369
2370        ! vectmp1 = 1.0d0/rborn
2371        ! vectmp2 = 1.0d0/(rborn-offset)
2372        ! vectmp3 = tanh( (gbalpha + gbgamma*psi_i*psi_i -
      gbbeta*psi_i )*psi_i )
2373
2374        onereff(1:vecend) = vectmp2(1:vecend) - &
2375                                (vectmp3(1:vecend)*vectmp1(1:vecend))
2376        do j =1,vecend
2377           if ( onereff(j) < zero ) onereff(j) = thirtieth
2378        end do
2379        call vdinv(vecend, onereff, reff)
2380     else
2381
2382        !       "standard" GB, including the "diagonal" term here:
2383
2384  #ifdef LES
2385  ! use rbornlong for these vectors, not rborn
2386        vectmp1(1:vecend) = rbornlong(1:vecend)-offset
2387  #else
2388        vectmp1(1:vecend) = rborn(1:vecend)-offset
2389  #endif
2390
2391        call vdinv(vecend, vectmp1, vectmp1)
2392        onereff(1:vecend) = onereff(1:vecend) + vectmp1(1:vecend)
2393        call vdinv(vecend, onereff, reff)
2394     end if
2395
2396  #ifdef LES
2397
2398      ! use RDT - determine if any of the non-LES atoms
2399      ! need multiple reff to be used
2400
2401      do k=1,natom
2402
```

```
2403         ! loop over the multiple effective radii for the atom
2404
2405         if (cnum(k).eq.0) then
2406
2407            ! non-LES atoms have multiple radii, compare range to rdt
2408
2409            if (rdt.gt.0.) then
2410
2411               istrt = ncopy * (k-1) +1
2412               iend = ncopy * (k)
2413               rmax=-10.
2414               rmin=999.
2415               do i=istrt,iend
2416                  if ( rmax <= reff(i) ) rmax = reff(i)
2417                  if ( rmin >= reff(i) ) rmin = reff(i)
2418               enddo
2419               if ((rmax-rmin).gt.rdt) then
2420                  nradii(k)=ncopy
2421               else
2422                  nradii(k)=1
2423               endif
2424            else
2425               nradii(k)=ncopy
2426            endif
2427         else
2428            ! LES atom
2429            nradii(k)=1
2430         endif
2431      enddo
2432   #else
2433      REQUIRE( rdt == 0 )
2434   #endif
2435
2436
2437      if ( rbornstat == 1 ) then
2438
2439         do k=1,natom
2440   #ifdef LES
2441            ! loop over the multiple effective radii for the atom
2442
2443            if (cnum(i).eq.0) then
2444
2445               ! non-LES atoms have multiple radii
2446
2447               istrt = ncopy * (i-1) +1
2448               iend = ncopy * (i)
2449            else
2450
2451            ! LES atoms only have one reff
2452
2453               istrt = ncopy * (i-1) +1
2454               iend = ncopy * (i-1) +1
2455            endif
2456            do i=istrt,iend
2457   #else
2458            i=k
2459   #endif
2460               rbave(i) = rbave(i) + reff(i)
2461               rbfluct(i) = rbfluct(i) + reff(i)*reff(i)
2462               if ( rbmax(i) <= reff(i) ) rbmax(i) = reff(i)
2463               if ( rbmin(i) >= reff(i) ) rbmin(i) = reff(i)
2464   #ifdef LES
2465            enddo
2466   #endif
2467         end do
2468      end if
```



```
2469
2470      return
2471
2472   end subroutine egb_calc_radii
2473
2474   ! checking if an atom belongs to nucleic or not
2475   ! make sure to add new DNA/RNA residue name to this list
2476   ! anyway not to hardcoded nucnamenum? (=60)
2477   ! tag: gbneck2nu
2478   ! TODO: use DNA/RNA some where?
2479   subroutine isnucat(nucat,atomindex,nres,nucnamenum,ipres,lbres)
2480       implicit none
2481       integer, intent(in) :: atomindex,nres,nucnamenum,ipres(nres)
2482       integer :: j,k,nucat
2483       character(4) :: nucname(nucnamenum),lbres(nres)
2484       !nucnamenum = 60
2485       nucat = 0
2486       nucname = (/ &
2487        "A   ", & !1
2488        "A3  ", &
2489        "A5  ", &
2490        "AN  ", &
2491        "C   ", & !5
2492        "C3  ", &
2493        "C5  ", &
2494        "CN  ", &
2495        "DA  ", &
2496        "DA3 ", & !10
2497        "DA5 ", &
2498        "DAN ", &
2499        "DC  ", &
2500        "DC3 ", &
2501        "DC5 ", & !15
2502        "DCN ", &
2503        "DG  ", &
2504        "DG3 ", &
2505        "DG5 ", &
2506        "DGN ", & !20
2507        "DT  ", &
2508        "DT3 ", &
2509        "DT5 ", &
2510        "DTN ", &
2511        "G   ", & !25
2512        "G3  ", &
2513        "G5  ", &
2514        "8OG ", &
2515        "GN  ", &
2516        "U   ", & !30
2517        "U3  ", &
2518        "U5  ", &
2519        "UN  ", &
2520        "AP  ", &
2521        "DAP ", & !35
2522        "CP  ", &
2523        "AF2 ", &
2524        "AF5 ", &
2525        "AF3 ", &
2526        "GF2 ", & !40
2527        "GF3 ", &
2528        "GF5 ", &
2529        "CF2 ", &
2530        "CFZ ", &
2531        "CF3 ", & !45
2532        "CF5 ", &
2533        "UF2 ", &
2534        "UF5 ", &
```

```
2535        "UF3 ", &
2536        "UFT ", & !50
2537        "OMC ", &
2538        "OMG ", &
2539        "MC3 ", &
2540        "MC5 ", &
2541        "MG3 ", & !55
2542        "MG5 ", &
2543        "SIC ", &
2544        "NAM ", &
2545        "NM5 ", &
2546        "CAP "  /)!60
2547
2548      do j=1,nres-1
2549          !find residue to which atom i belongs
2550          if (ipres(j)  <= atomindex .and. atomindex < ipres(j+1)) then
2551              do k=1,nucnamenum
2552                  !if atom i belongs to nucleic, return 1
2553                  if (lbres(j)(1:4) == nucname(k)(1:4)) then
2554                      nucat = 1
2555                  end if
2556              end do
2557          end if
2558      end do
2559      if (atomindex >= ipres(nres)) then
2560         do k=1,nucnamenum
2561             !if atom i belongs to nucleic, return 1
2562             if (lbres(j)(1:4) == nucname(k)(1:4)) then
2563                 nucat = 1
2564             end if
2565         end do
2566      end if
2567  end subroutine isnucat
2568
2569  #ifdef LES
2570  end module genbornles
2571  #else
2572  end module genborn
2573  #endif
```