```
#endif
   onstep = mod(irespa,nrespa) == 0
   onstepi = mod(irespa,nrespai) == 0
   if( .not.onstepi ) return

   oncpstep = (icnstph == 1 .and. mod(irespa, ntcnstph) == 0) .or. icnstph == 2


   if(alpb == 1) then
!     Sigalov Onufriev ALPB (epsilon-dependent GB):
      alpb_beta = alpb_alpha*(intdiel/extdiel)
      extdieli = one/(extdiel*(one + alpb_beta))
      intdieli = one/(intdiel*(one + alpb_beta))
      one_Arad_beta = alpb_beta/Arad
      if (kappa/=zero) onekappa = one/kappa
   else
!  Standard Still's GB - alpb=0
      extdieli = one/extdiel
      intdieli = one/intdiel
      one_Arad_beta = zero
   end if

   ! Smooth "cut-off" in calculating GB effective radii.
   ! Implementd by Andreas Svrcek-Seiler and Alexey Onufriev.
   ! The integration over solute is performed up to rgbmax and includes
   ! parts of spheres; that is an atom is not just "in" or "out", as
   ! with standard non-bonded cut.  As a result, calclated effective
   ! radii are less than rgbmax. This saves time, and there is no
   ! discontinuity in dReff/drij.

   ! Only the case rgbmax > 5*max(sij) = 5*fsmax ~ 9A is handled; this is
   ! enforced in mdread().  Smaller values would not make much physical
   ! sense anyway.

   rgbmax2 = rgbmax*rgbmax
   rgbmax1i = one/rgbmax
   rgbmax2i = rgbmax1i*rgbmax1i
   rgbmaxpsmax2 = (rgbmax+fsmax)**2

#ifdef LES
! initialize some things for GB+LES

! one over number of LES copies
! GB+LES only works with 1 LES region so we can't have multiple
! copy numbers like we can with PME

   lfac = float(ncopy)
   lfaci = one/(lfac)

#else
   ncopy2 = ncopy
#endif

   !------------------------------------------------------------------------------
   !     Step 1: loop over pairs of atoms to compute the effective Born radii.
   !------------------------------------------------------------------------------
   !
   ! The effective Born radii are now calculated via a call at the
   ! beginning of force.
   !
   !------------------------------------------------------------------------------

   iexcl = 1 !moved to outside the index loop from original location in "step 2"

#ifdef MPI
   do i=1,mytaskid
```



```
      iexcl = iexcl + numex(i)
   end do
   mpistart = mytaskid+1
#endif

   maxi = natom
   if(natbel > 0) maxi = natbel

   !------------------------------------------------------------------------------
   !
   !     Step 2: loop over all pairs of atoms, computing the gas-phase
   !             electrostatic energies, the LJ terms, and the off-diagonal
   !             GB terms.  Also accumulate the derivatives of these off-
   !             diagonal terms with respect to the inverse effective radii,
   !             sumdeijda(k) will hold  sum over i,j>i ( deij/dak ),  where
   !             "ak" is the inverse of the effective radius for atom "k".
   !
   !             Update the forces with the negative derivatives of the
   !             gas-phase terms, plus the derivatives of the explicit
   !             distance dependence in Fgb, i.e. the derivatives of the
   !             GB energy terms assuming that the effective radii are
   !             constant.
   !
   !------------------------------------------------------------------------------

#ifdef LES
   sumdeijda(1:natom*ncopy) = zero
#else
   sumdeijda(1:natom) = zero
#endif

   call timer_start(TIME_GBFRC)

   !       Note: this code assumes that the belly atoms are the first natbel
   !             atoms...this is checked in mdread.

#ifdef MPI
   do i=mpistart,maxi,numtasks
#else
   do i=1,maxi
#endif

#ifdef LES
      lestmp = nlesty*(lestyp(i)-1)
#endif

      xi = x(3*i-2)
      yi = x(3*i-1)
      zi = x(3*i  )
      qi = charge(i)
      ri = reff(i)
      four_ri = four*reff(i)
      iaci = ntypes * (iac(i) - 1)
      jexcl = iexcl
      jexcl_last = iexcl + numex(i) -1
      dumx = zero
      dumy = zero
      dumz = zero

#if defined(LES)
      icnum = cnum(i)
      nrg_vdw_tmp = zero
      nrg_ele_tmp = zero
      nrg_egb_tmp = zero
#endif
```

```
            endif
          endif

          istrt = ncopy * (i-1)
          jstrt = ncopy * (j-1)


          first=.true.
          do k=idx1,idx2

! non-LES uses just ri (reff(i)) in this loop, doesn't need to be vector

              rix(icount+1)=reff(istrt+k)

              reff_j=reff(jstrt+k)

! set longskipv, which serves the purpose of telling if an atom is
! excluded and also telling if this is not the first in the loop over

! pairs of reff for an atom (only calculate nonbonds for the first pair of reff)
! longskipv is only set for the icount atoms! (not all j like skipv)

              if ((.not.first).or.skipv(j)) then
                longskipv(icount+1)=.true.
              else
                longskipv(icount+1)=.false.
              endif

! for LES we need to know which reff this is when we calculate the sumdeija
! so set a pointer here to tell us which reff this i,j pair are using
! this points directly into correct spot in reff or sumdeijda

              iridx(icount+1) = istrt+k
              jridx(icount+1) = jstrt+k

! set flag to tell whether sumdeijda needs to be dividing among components for non-
LES atoms
! temporary value was set above

              spreadfrc(icount+1) = spread

#else
! set for non-LES case so we can refer to reff_j not reff(j)
! this is needed with LES since each atom j has multiple reff

              reff_j = reff(j)
#endif

              icount = icount + 1
              temp_jj(icount) = j
              r2x(icount) = r2

! carlos changed this, we are STILL INSIDE LOOP OVER K for multiple LES reff_j
! so we need to use reff_j not reff(j) (since LES may have multiple reff_j for
atom j)

              rjx(icount) = reff_j

#ifdef LES
                ! set scaling factor here where we know if we are doing
                ! averaging over multiple radii. in the loops below we don't
                ! have that info anymore without checking nradii
                ! we couldn't set these above since we didn't have an
                ! icount value yet

                scalefac(icount)=tempscale
```



```
                lesscalefac(icount)=templfac

                ! set first to F so that only the first of these i,j
                ! entries will have nonbonds

                first=.false.

              enddo ! LOOP OVER K, pairs of reff for i and j
#endif
            end if !r2 <= cut
          end do

          if( igb/=6 ) then

#ifdef LES
            ! rix needs to be used instead of ri since i doesn't have just 1 reff
            vectmp1(1:icount) = four*rix(1:icount)*rjx(1:icount)
#else
            vectmp1(1:icount) = four_ri*rjx(1:icount)
#endif

            call vdinv( icount, vectmp1, vectmp1 ) !Invert things
            vectmp1(1:icount) = -r2x(1:icount)*vectmp1(1:icount)
            call vdexp( icount, vectmp1, vectmp1 )
                ! [ends up with Exp(-rij^2/[4*ai*aj])]
#ifdef LES
            ! ri is not the same for all of the icount!
            vectmp3(1:icount) = r2x(1:icount) + &
                    rjx(1:icount)*rix(1:icount)*vectmp1(1:icount) !ends up with fij
#else
            vectmp3(1:icount) = r2x(1:icount) + rjx(1:icount)*ri*vectmp1(1:icount)
                ! [ends up with fij]
#endif

            ! CARLOS: LES
            ! we now have two sets of vectors: one is 1 to last atom, the other
            ! is 1 to icount
            ! the atom one is for vdw/ele and for excl, the other is for gb offdiag
            ! which do we loop over? unless we have a pointer we need to loop
            ! over the bigger one, have a pointer j in it, and pull excl out
            ! of that. but how to know to skip the nonbonds for all after the
            ! first in a loop over a pair? maybe we should have the excl loop
            ! done for icount, not atoms.
            ! SKIPV PART OK, DO THE DISTANCE PART SO WE DON'T HAVE TO INV ALL RIJ
            ! FOR NOW IT'S OK, JUST SLOW

            call vdinvsqrt( icount, vectmp3, vectmp2 ) !vectmp2 = 1/fij

            if( kappa /= zero ) then
               call vdinv( icount, vectmp2, vectmp3 )
               vectmp3(1:icount) = -kappa*vectmp3(1:icount)
               call vdexp( icount, vectmp3, vectmp4 ) !exp(-kappa*fij)
            end if

          end if
          call vdinvsqrt( icount, r2x, vectmp5 ) !1/rij

          ! vectmp1 = Exp(-rij^2/[4*ai*aj])
          ! vectmp2 = 1/fij.
          ! vectmp3 = -kappa*fij - if kappa/=zero, otherwise = fij
          ! vectmp4 = exp(-kappa*fij)
          ! vectmp5 = 1/rij - note with qmmm this contains the
                            !distance to mm_link pairs not to QM link atoms.

        !---- Start first outer loop ----
!!! !dir$ ivdep
```