```
c-----------------------------------------------------------------------
      subroutine VGRBPUSH23LT(part,fxy,bxy,qbm,dt,dtc,ci,ek,idimp,nop,
     1npe,nx,ny,nxv,nyv,ipbc)
c for 2-1/2d code, this subroutine updates particle co-ordinates and
c velocities using leap-frog scheme in time and first-order linear
c interpolation in space, for relativistic particles with magnetic field
c Using the Boris Mover.
c vectorizable version using guard cells
c 131 flops/particle, 4 divides, 2 sqrts, 25 loads, 5 stores
c input: all, output: part, ek
c momentum equations used are:
c px(t+dt/2) = rot(1)*(px(t-dt/2) + .5*(q/m)*fx(x(t),y(t))*dt) +
c    rot(2)*(py(t-dt/2) + .5*(q/m)*fy(x(t),y(t))*dt) +
c    rot(3)*(pz(t-dt/2) + .5*(q/m)*fz(x(t),y(t))*dt) +
c    .5*(q/m)*fx(x(t),y(t))*dt)
c py(t+dt/2) = rot(4)*(px(t-dt/2) + .5*(q/m)*fx(x(t),y(t))*dt) +
c    rot(5)*(py(t-dt/2) + .5*(q/m)*fy(x(t),y(t))*dt) +
c    rot(6)*(pz(t-dt/2) + .5*(q/m)*fz(x(t),y(t))*dt) +
c    .5*(q/m)*fy(x(t),y(t))*dt)
c pz(t+dt/2) = rot(7)*(px(t-dt/2) + .5*(q/m)*fx(x(t),y(t))*dt) +
c    rot(8)*(py(t-dt/2) + .5*(q/m)*fy(x(t),y(t))*dt) +
c    rot(9)*(pz(t-dt/2) + .5*(q/m)*fz(x(t),y(t))*dt) +
c    .5*(q/m)*fz(x(t),y(t))*dt)
c where q/m is charge/mass, and the rotation matrix is given by:
c    rot(1) = (1 - (om*dt/2)**2 + 2*(omx*dt/2)**2)/(1 + (om*dt/2)**2)
c    rot(2) = 2*(omz*dt/2 + (omx*dt/2)*(omy*dt/2))/(1 + (om*dt/2)**2)
c    rot(3) = 2*(-omy*dt/2 + (omx*dt/2)*(omz*dt/2))/(1 + (om*dt/2)**2)
c    rot(4) = 2*(-omz*dt/2 + (omx*dt/2)*(omy*dt/2))/(1 + (om*dt/2)**2)
c    rot(5) = (1 - (om*dt/2)**2 + 2*(omy*dt/2)**2)/(1 + (om*dt/2)**2)
c    rot(6) = 2*(omx*dt/2 + (omy*dt/2)*(omz*dt/2))/(1 + (om*dt/2)**2)
c    rot(7) = 2*(omy*dt/2 + (omx*dt/2)*(omz*dt/2))/(1 + (om*dt/2)**2)
c    rot(8) = 2*(-omx*dt/2 + (omy*dt/2)*(omz*dt/2))/(1 + (om*dt/2)**2)
c    rot(9) = (1 - (om*dt/2)**2 + 2*(omz*dt/2)**2)/(1 + (om*dt/2)**2)
c and om**2 = omx**2 + omy**2 + omz**2
c the rotation matrix is determined by:
c omx = (q/m)*bx(x(t),y(t))*gami, omy = (q/m)*by(x(t),y(t))*gami, and
c omz = (q/m)*bz(x(t),y(t))*gami,
c where gami = 1./sqrt(1.+(px(t)*px(t)+py(t)*py(t)+pz(t)*pz(t))*ci*ci)
c position equations used are:
c x(t+dt) = x(t) + px(t+dt/2)*dtg
c y(t+dt) = y(t) + py(t+dt/2)*dtg
c where dtg = dtc/sqrt(1.+(px(t+dt/2)*px(t+dt/2)+py(t+dt/2)*py(t+dt/2)+
c pz(t+dt/2)*pz(t+dt/2))*ci*ci)
c fx(x(t),y(t)), fy(x(t),y(t)), and fz(x(t),y(t))
c bx(x(t),y(t)), by(x(t),y(t)), and bz(x(t),y(t))
c are approximated by interpolation from the nearest grid points:
c fx(x,y) = (1-dy)*((1-dx)*fx(n,m)+dx*fx(n+1,m)) + dy*((1-dx)*fx(n,m+1)
c    + dx*fx(n+1,m+1))
c where n,m = leftmost grid points and dx = x-n, dy = y-m
c similarly for fy(x,y), fz(x,y), bx(x,y), by(x,y), bz(x,y)
c part(n,1) = position x of particle n
c part(n,2) = position y of particle n
c part(n,3) = momentum px of particle n
c part(n,4) = momentum py of particle n
```

```
c part(n,5) = momentum pz of particle n
c fxy(1,j,k) = x component of force/charge at grid (j,k)
c fxy(2,j,k) = y component of force/charge at grid (j,k)
c fxy(3,j,k) = z component of force/charge at grid (j,k)
c that is, convolution of electric field over particle shape
c bxy(1,j,k) = x component of magnetic field at grid (j,k)
c bxy(2,j,k) = y component of magnetic field at grid (j,k)
c bxy(3,j,k) = z component of magnetic field at grid (j,k)
c that is, the convolution of magnetic field over particle shape
c qbm = particle charge/mass ratio
c dt = time interval between successive calculations
c dtc = time interval between successive co-ordinate calculations
c ci = reciprocal of velocity of light
c kinetic energy/mass at time t is also calculated, using
c ek = gami*sum((px(t-dt/2) + .5*(q/m)*fx(x(t),y(t))*dt)**2 +
c      (py(t-dt/2) + .5*(q/m)*fy(x(t),y(t))*dt)**2 +
c      (pz(t-dt/2) + .5*(q/m)*fz(x(t),y(t))*dt)**2)/(1. + gami)
c idimp = size of phase space = 5
c nop = number of particles
c npe = first dimension of particle array
c nx/ny = system length in x/y direction
c nxv = second dimension of field arrays, must be >= nx+1
c nyv = third dimension of field arrays, must be >= ny+1
c ipbc = particle boundary condition = (0,1,2,3) =
c (none,2d periodic,2d reflecting,mixed reflecting/periodic)
      implicit none
      integer idimp, nop, npe, nx, ny, nxv, nyv, ipbc
      real qbm, dt, dtc, ci, ek
      real part, fxy, bxy
      dimension part(npe,idimp)
      dimension fxy(4,nxv*nyv), bxy(4,nxv*nyv)
c local data
      integer npblk, lvect
      parameter(npblk=32,lvect=4)
      integer i, j, k, ipp, joff, nps, nn, mm
      real qtmh, ci2, edgelx, edgely, edgerx, edgery, dxp, dyp, amx, amy
      real dx, dy, dz, ox, oy, oz, acx, acy, acz, p2, gami, qtmg, dtg
      real omxt, omyt, omzt, omt, anorm
      real rot1, rot2, rot3, rot4, rot5, rot6, rot7, rot8, rot9
      real x, y, vx, vy, vz
c scratch arrays
      integer n
      real s1, s2, t
      dimension n(npblk), s1(npblk,lvect), s2(npblk,lvect), t(npblk,2)
      double precision sum1
      qtmh = 0.5*qbm*dt
      ci2 = ci*ci
      sum1 = 0.0d0
c set boundary values
      edgelx = 0.0
      edgely = 0.0
      edgerx = real(nx)
      edgery = real(ny)
      if (ipbc.eq.2) then
```

```
         edgelx = 1.0
         edgely = 1.0
         edgerx = real(nx-1)
         edgery = real(ny-1)
      else if (ipbc.eq.3) then
         edgelx = 1.0
         edgerx = real(nx-1)
      endif
      ipp = nop/npblk
c outer loop over number of full blocks
      do 60 k = 1, ipp
      joff = npblk*(k - 1)
c inner loop over particles in block
      do 10 j = 1, npblk
c find interpolation weights
      x = part(j+joff,1)
      y = part(j+joff,2)
      nn = x
      mm = y
      dxp = x - real(nn)
      dyp = y - real(mm)
      n(j) = nn + nxv*mm
      amx = 1.0 - dxp
      amy = 1.0 - dyp
      s1(j,1) = amx*amy
      s1(j,2) = dxp*amy
      s1(j,3) = amx*dyp
      s1(j,4) = dxp*dyp
      t(j,1) = x
      t(j,2) = y
   10 continue
c find acceleration
      do 30 j = 1, npblk
      nn = n(j)
      mm = nn + nxv - 2
      dx = 0.0
      dy = 0.0
      dz = 0.0
      ox = 0.0
      oy = 0.0
      oz = 0.0
      do 20 i = 1, lvect
      if (i.gt.2) nn = mm
      dx = dx + fxy(1,i+nn)*s1(j,i)
      dy = dy + fxy(2,i+nn)*s1(j,i)
      dz = dz + fxy(3,i+nn)*s1(j,i)
      ox = ox + bxy(1,i+nn)*s1(j,i)
      oy = oy + bxy(2,i+nn)*s1(j,i)
      oz = oz + bxy(3,i+nn)*s1(j,i)
   20 continue
      s1(j,1) = dx
      s1(j,2) = dy
      s1(j,3) = dz
      s2(j,1) = ox
      s2(j,2) = oy
      s2(j,3) = oz
   30 continue
c new momentum
      do 40 j = 1, npblk
      x = t(j,1)
      y = t(j,2)
c calculate half impulse
      dx = qtmh*s1(j,1)
      dy = qtmh*s1(j,2)
      dz = qtmh*s1(j,3)
c half acceleration
      acx = part(j+joff,3) + dx
      acy = part(j+joff,4) + dy
      acz = part(j+joff,5) + dz
c find inverse gamma
      p2 = acx*acx + acy*acy + acz*acz
      gami = 1.0/sqrt(1.0 + p2*ci2)
c renormalize magnetic field
      qtmg = qtmh*gami
c time-centered kinetic energy
      sum1 = sum1 + gami*p2/(1.0 + gami)
c calculate cyclotron frequency
      omxt = qtmg*s2(j,1)
      omyt = qtmg*s2(j,2)
      omzt = qtmg*s2(j,3)
c calculate rotation matrix
      omt = omxt*omxt + omyt*omyt + omzt*omzt
      anorm = 2.0/(1.0 + omt)
      omt = 0.5*(1.0 - omt)
      rot4 = omxt*omyt
      rot7 = omxt*omzt
      rot8 = omyt*omzt
      rot1 = omt + omxt*omxt
      rot5 = omt + omyt*omyt
      rot9 = omt + omzt*omzt
      rot2 = omzt + rot4
      rot4 = -omzt + rot4
      rot3 = -omyt + rot7
      rot7 = omyt + rot7
      rot6 = omxt + rot8
      rot8 = -omxt + rot8
c new velocity
      vx = (rot1*acx + rot2*acy + rot3*acz)*anorm + dx
      vy = (rot4*acx + rot5*acy + rot6*acz)*anorm + dy
      vz = (rot7*acx + rot8*acy + rot9*acz)*anorm + dz
c update inverse gamma
      p2 = vx*vx + vy*vy + vz*vz
      dtg = dtc/sqrt(1.0 + p2*ci2)
c new position
      s1(j,1) = x + vx*dtg
      s1(j,2) = y + vy*dtg
      s2(j,1) = vx
      s2(j,2) = vy
      s2(j,3) = vz
   40 continue
! check boundary conditions
!dir$ novector
      do 50 j = 1, npblk
      dx = s1(j,1)
      dy = s1(j,2)
      vx = s2(j,1)
      vy = s2(j,2)
      vz = s2(j,3)
c periodic boundary conditions
      if (ipbc.eq.1) then
         if (dx.lt.edgelx) dx = dx + edgerx
         if (dx.ge.edgerx) dx = dx - edgerx
         if (dy.lt.edgely) dy = dy + edgery
         if (dy.ge.edgery) dy = dy - edgery
c reflecting boundary conditions
      else if (ipbc.eq.2) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = t(j,1)
            vx = -vx
         endif
         if ((dy.lt.edgely).or.(dy.ge.edgery)) then
            dy = t(j,2)
            vy = -vy
         endif
c mixed reflecting/periodic boundary conditions
      else if (ipbc.eq.3) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = t(j,1)
            vx = -vx
         endif
         if (dy.lt.edgely) dy = dy + edgery
         if (dy.ge.edgery) dy = dy - edgery
      endif
c set new position
      part(j+joff,1) = dx
      part(j+joff,2) = dy
c set new velocity
      part(j+joff,3) = vx
      part(j+joff,4) = vy
      part(j+joff,5) = vz
   50 continue
   60 continue
      nps = npblk*ipp + 1
c loop over remaining particles
      do 70 j = nps, nop
c find interpolation weights
      x = part(j,1)
      y = part(j,2)
      nn = x
      mm = y
      dxp = x - real(nn)
      dyp = y - real(mm)
```

```
         nn = nn + nxv*mm + 1
         amx = 1.0 - dxp
         amy = 1.0 - dyp
c find electric field
         dx = amx*fxy(1,nn)
         dy = amx*fxy(2,nn)
         dz = amx*fxy(3,nn)
         dx = amy*(dxp*fxy(1,nn+1) + dx)
         dy = amy*(dxp*fxy(2,nn+1) + dy)
         dz = amy*(dxp*fxy(3,nn+1) + dz)
         acx = amx*fxy(1,nn+nxv)
         acy = amx*fxy(2,nn+nxv)
         acz = amx*fxy(3,nn+nxv)
         dx = dx + dyp*(dxp*fxy(1,nn+1+nxv) + acx)
         dy = dy + dyp*(dxp*fxy(2,nn+1+nxv) + acy)
         dz = dz + dyp*(dxp*fxy(3,nn+1+nxv) + acz)
c find magnetic field
         ox = amx*bxy(1,nn)
         oy = amx*bxy(2,nn)
         oz = amx*bxy(3,nn)
         ox = amy*(dxp*bxy(1,nn+1) + ox)
         oy = amy*(dxp*bxy(2,nn+1) + oy)
         oz = amy*(dxp*bxy(3,nn+1) + oz)
         acx = amx*bxy(1,nn+nxv)
         acy = amx*bxy(2,nn+nxv)
         acz = amx*bxy(3,nn+nxv)
         ox = ox + dyp*(dxp*bxy(1,nn+1+nxv) + acx)
         oy = oy + dyp*(dxp*bxy(2,nn+1+nxv) + acy)
         oz = oz + dyp*(dxp*bxy(3,nn+1+nxv) + acz)
c calculate half impulse
         dx = qtmh*dx
         dy = qtmh*dy
         dz = qtmh*dz
c half acceleration
         acx = part(j,3) + dx
         acy = part(j,4) + dy
         acz = part(j,5) + dz
c find inverse gamma
         p2 = acx*acx + acy*acy + acz*acz
         gami = 1.0/sqrt(1.0 + p2*ci2)
c renormalize magnetic field
         qtmg = qtmh*gami
c time-centered kinetic energy
         sum1 = sum1 + gami*p2/(1.0 + gami)
c calculate cyclotron frequency
         omxt = qtmg*ox
         omyt = qtmg*oy
         omzt = qtmg*oz
c calculate rotation matrix
         omt = omxt*omxt + omyt*omyt + omzt*omzt
         anorm = 2.0/(1.0 + omt)
         omt = 0.5*(1.0 - omt)
         rot4 = omxt*omyt
         rot7 = omxt*omzt
```

```
      rot8 = omyt*omzt
      rot1 = omt + omxt*omxt
      rot5 = omt + omyt*omyt
      rot9 = omt + omzt*omzt
      rot2 = omzt + rot4
      rot4 = -omzt + rot4
      rot3 = -omyt + rot7
      rot7 = omyt + rot7
      rot6 = omxt + rot8
      rot8 = -omxt + rot8
c new velocity
      vx = (rot1*acx + rot2*acy + rot3*acz)*anorm + dx
      vy = (rot4*acx + rot5*acy + rot6*acz)*anorm + dy
      vz = (rot7*acx + rot8*acy + rot9*acz)*anorm + dz
c update inverse gamma
      p2 = vx*vx + vy*vy + vz*vz
      dtg = dtc/sqrt(1.0 + p2*ci2)
c new position
      dx = x + vx*dtg
      dy = y + vy*dtg
c periodic boundary conditions
      if (ipbc.eq.1) then
         if (dx.lt.edgelx) dx = dx + edgerx
         if (dx.ge.edgerx) dx = dx - edgerx
         if (dy.lt.edgely) dy = dy + edgery
         if (dy.ge.edgery) dy = dy - edgery
c reflecting boundary conditions
      else if (ipbc.eq.2) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = t(j,1)
            vx = -vx
         endif
         if ((dy.lt.edgely).or.(dy.ge.edgery)) then
            dy = t(j,2)
            vy = -vy
         endif
c mixed reflecting/periodic boundary conditions
      else if (ipbc.eq.3) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = t(j,1)
            vx = -vx
         endif
         if (dy.lt.edgely) dy = dy + edgery
         if (dy.ge.edgery) dy = dy - edgery
      endif
c set new position
      part(j,1) = dx
      part(j,2) = dy
c set new velocity
      part(j,3) = vx
      part(j,4) = vy
      part(j,5) = vz
   70 continue
c normalize kinetic energy
```

```
ek = ek + sum1
return
end
```

```
c-----------------------------------------------------------------------
      subroutine VGPOST2LT(part,q,qm,nop,npe,idimp,nxv,nyv)
c for 2d code, this subroutine calculates particle charge density
c using first-order linear interpolation, periodic boundaries
c vectorizable version using guard cells
c 17 flops/particle, 6 loads, 4 stores
c input: all, output: q
c charge density is approximated by values at the nearest grid points
c q(n,m)=qm*(1.-dx)*(1.-dy)
c q(n+1,m)=qm*dx*(1.-dy)
c q(n,m+1)=qm*(1.-dx)*dy
c q(n+1,m+1)=qm*dx*dy
c where n,m = leftmost grid points and dx = x-n, dy = y-m
c part(n,1) = position x of particle n
c part(n,2) = position y of particle n
c q(j,k) = charge density at grid point j,k
c qm = charge on particle, in units of e
c nop = number of particles
c npe = first dimension of particle array
c idimp = size of phase space = 4
c nxv = first dimension of charge array, must be >= nx+1
c nyv = second dimension of charge array, must be >= ny+1
      implicit none
      integer nop, npe, idimp, nxv, nyv
      real qm
      real part, q
      dimension part(npe,idimp), q(nxv*nyv)
c local data
      integer npblk, lvect
      parameter(npblk=32,lvect=4)
      integer i, j, k, ipp, joff, nps, nn, mm
      real x, y, dxp, dyp, amx, amy
c scratch arrays
      integer n
      real s
      dimension n(npblk), s(npblk,lvect)
      ipp = nop/npblk
c outer loop over number of full blocks
      do 40 k = 1, ipp
      joff = npblk*(k - 1)
c inner loop over particles in block
      do 10 j = 1, npblk
c find interpolation weights
      x = part(j+joff,1)
      y = part(j+joff,2)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
      dyp = y - real(mm)
      n(j) = nn + nxv*mm
      amx = qm - dxp
      amy = 1.0 - dyp
      s(j,1) = amx*amy
      s(j,2) = dxp*amy
      s(j,3) = amx*dyp
      s(j,4) = dxp*dyp
   10 continue
c deposit charge
      do 30 j = 1, npblk
      nn = n(j)
      mm = nn + nxv - 2
!dir$ ivdep
      do 20 i = 1, lvect
      if (i.gt.2) nn = mm
      q(i+nn) = q(i+nn) + s(j,i)
   20 continue
   30 continue
   40 continue
      nps = npblk*ipp + 1
c loop over remaining particles
      do 50 j = nps, nop
c find interpolation weights
      x = part(j,1)
      y = part(j,2)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
      dyp = y - real(mm)
      nn = nn + nxv*mm + 1
      amx = qm - dxp
      amy = 1.0 - dyp
c deposit charge
      x = q(nn) + amx*amy
      y = q(nn+1) + dxp*amy
      q(nn) = x
      q(nn+1) = y
      x = q(nn+nxv) + amx*dyp
      y = q(nn+nxv+1) + dxp*dyp
      q(nn+nxv) = x
      q(nn+nxv+1) = y
   50 continue
      return
      end
```

```
c-----------------------------------------------------------------------
      subroutine VGRJPOST2LT(part,cu,qm,dt,ci,nop,npe,idimp,nx,ny,nxv,
     1nyv,ipbc)
c for 2-1/2d code, this subroutine calculates particle current density
c using first-order linear interpolation for relativistic particles
c in addition, particle positions are advanced a half time-step
c vectorizable version using guard cells
c 47 flops/particle, 1 divide, 1 sqrt, 17 loads, 14 stores
c input: all, output: part, cu
c current density is approximated by values at the nearest grid points
c cu(i,n,m)=qci*(1.-dx)*(1.-dy)
c cu(i,n+1,m)=qci*dx*(1.-dy)
c cu(i,n,m+1)=qci*(1.-dx)*dy
c cu(i,n+1,m+1)=qci*dx*dy
c where n,m = leftmost grid points and dx = x-n, dy = y-m
c and qci = qm*pi*gami, where i = x,y,z
c where gami = 1./sqrt(1.+sum(pi**2)*ci*ci)
c part(n,1) = position x of particle n
c part(n,2) = position y of particle n
c part(n,3) = x momentum of particle n
c part(n,4) = y momentum of particle n
c part(n,5) = z momentum of particle n
c cu(i,j,k) = ith component of current density at grid point j,k
c qm = charge on particle, in units of e
c dt = time interval between successive calculations
c ci = reciprocal of velocity of light
c nop = number of particles
c npe = first dimension of particle array
c idimp = size of phase space = 5
c nx/ny = system length in x/y direction
c nxv = second dimension of current array, must be >= nx+1
c nyv = third dimension of current array, must be >= ny+1
c ipbc = particle boundary condition = (0,1,2,3) =
c (none,2d periodic,2d reflecting,mixed reflecting/periodic)
      implicit none
      integer nop, npe, idimp, nx, ny, nxv, nyv, ipbc
      real qm, dt, ci
      real part, cu
      dimension part(npe,idimp), cu(4,nxv*nyv)
c local data
      integer npblk, lvect
      parameter(npblk=32,lvect=4)
      integer i, j, k, ipp, joff, nps, nn, mm
      real ci2, edgelx, edgely, edgerx, edgery, dxp, dyp, amx, amy
      real x, y, dx, dy, vx, vy, vz, ux, uy, uz, p2, gami
c scratch arrays
      integer n
      real s1, s2, t
      dimension n(npblk), s1(npblk,lvect), s2(npblk,lvect), t(npblk,4)
      ci2 = ci*ci
      ipp = nop/npblk
c set boundary values
      edgelx = 0.0
      edgely = 0.0
      edgerx = real(nx)
      edgery = real(ny)
      if (ipbc.eq.2) then
         edgelx = 1.0
         edgely = 1.0
         edgerx = real(nx-1)
         edgery = real(ny-1)
      else if (ipbc.eq.3) then
         edgelx = 1.0
         edgerx = real(nx-1)
      endif
c outer loop over number of full blocks
      do 50 k = 1, ipp
      joff = npblk*(k - 1)
c inner loop over particles in block
      do 10 j = 1, npblk
c find interpolation weights
      x = part(j+joff,1)
      y = part(j+joff,2)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
      dyp = y - real(mm)
      n(j) = nn + nxv*mm
      amx = qm - dxp
      amy = 1.0 - dyp
      s1(j,1) = amx*amy
      s1(j,2) = dxp*amy
      s1(j,3) = amx*dyp
      s1(j,4) = dxp*dyp
      t(j,1) = x
      t(j,2) = y
c find inverse gamma
      ux = part(j+joff,3)
      uy = part(j+joff,4)
      uz = part(j+joff,5)
      p2 = ux*ux + uy*uy + uz*uz
      gami = 1.0/sqrt(1.0 + p2*ci2)
      s2(j,1) = ux*gami
      s2(j,2) = uy*gami
      s2(j,3) = uz*gami
      t(j,3) = ux
      t(j,4) = uy
   10 continue
c deposit current
      do 30 j = 1, npblk
      nn = n(j)
      mm = nn + nxv - 2
      vx = s2(j,1)
      vy = s2(j,2)
      vz = s2(j,3)
!dir$ ivdep
      do 20 i = 1, lvect
      if (i.gt.2) nn = mm
      cu(1,i+nn) = cu(1,i+nn) + vx*s1(j,i)
      cu(2,i+nn) = cu(2,i+nn) + vy*s1(j,i)
      cu(3,i+nn) = cu(3,i+nn) + vz*s1(j,i)
   20 continue
   30 continue
c advance position half a time-step
!dir$ novector
      do 40 j = 1, npblk
      x = t(j,1)
      y = t(j,2)
      vx = s2(j,1)
      vy = s2(j,2)
      ux = t(j,3)
      uy = t(j,4)
      dx = x + vx*dt
      dy = y + vy*dt
c periodic boundary conditions
      if (ipbc.eq.1) then
         if (dx.lt.edgelx) dx = dx + edgerx
         if (dx.ge.edgerx) dx = dx - edgerx
         if (dy.lt.edgely) dy = dy + edgery
         if (dy.ge.edgery) dy = dy - edgery
c reflecting boundary conditions
      else if (ipbc.eq.2) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = x
            part(j+joff,3) = -ux
         endif
         if ((dy.lt.edgely).or.(dy.ge.edgery)) then
            dy = y
            part(j+joff,4) = -uy
         endif
c mixed reflecting/periodic boundary conditions
      else if (ipbc.eq.3) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = x
            part(j+joff,3) = -ux
         endif
         if (dy.lt.edgely) dy = dy + edgery
         if (dy.ge.edgery) dy = dy - edgery
      endif
c set new position
      part(j+joff,1) = dx
      part(j+joff,2) = dy
   40 continue
   50 continue
      nps = npblk*ipp + 1
c loop over remaining particles
      do 60 j = nps, nop
c find interpolation weights
      x = part(j,1)
      y = part(j,2)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
      dyp = y - real(mm)
c find inverse gamma
      ux = part(j,3)
      uy = part(j,4)
      uz = part(j,5)
      p2 = ux*ux + uy*uy + uz*uz
      gami = 1.0/sqrt(1.0 + p2*ci2)
c calculate weights
      nn = nn + nxv*mm + 1
      amx = qm - dxp
      amy = 1.0 - dyp
c deposit current
      dx = amx*amy
      dy = dxp*amy
      vx = ux*gami
      vy = uy*gami
      vz = uz*gami
      cu(1,nn) = cu(1,nn) + vx*dx
      cu(2,nn) = cu(2,nn) + vy*dx
      cu(3,nn) = cu(3,nn) + vz*dx
      dx = amx*dyp
      cu(1,nn+1) = cu(1,nn+1) + vx*dy
      cu(2,nn+1) = cu(2,nn+1) + vy*dy
      cu(3,nn+1) = cu(3,nn+1) + vz*dy
      dy = dxp*dyp
      cu(1,nn+nxv) = cu(1,nn+nxv) + vx*dx
      cu(2,nn+nxv) = cu(2,nn+nxv) + vy*dx
      cu(3,nn+nxv) = cu(3,nn+nxv) + vz*dx
      cu(1,nn+1+nxv) = cu(1,nn+1+nxv) + vx*dy
      cu(2,nn+1+nxv) = cu(2,nn+1+nxv) + vy*dy
      cu(3,nn+1+nxv) = cu(3,nn+1+nxv) + vz*dy
c advance position half a time-step
      dx = x + vx*dt
      dy = y + vy*dt
c periodic boundary conditions
      if (ipbc.eq.1) then
         if (dx.lt.edgelx) dx = dx + edgerx
         if (dx.ge.edgerx) dx = dx - edgerx
         if (dy.lt.edgely) dy = dy + edgery
         if (dy.ge.edgery) dy = dy - edgery
c reflecting boundary conditions
      else if (ipbc.eq.2) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = x
            part(j,3) = -ux
         endif
         if ((dy.lt.edgely).or.(dy.ge.edgery)) then
            dy = y
            part(j,4) = -uy
         endif
c mixed reflecting/periodic boundary conditions
      else if (ipbc.eq.3) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
```

```
            dx = x
            part(j,3) = -ux
         endif
         if (dy.lt.edgely) dy = dy + edgery
         if (dy.ge.edgery) dy = dy - edgery
      endif
c set new position
      part(j,1) = dx
      part(j,2) = dy
   60 continue
      return
      end
```