```
c-----------------------------------------------------------------------
      subroutine VGRBPPUSHF23LT(ppart,fxy,bxy,kpic,ncl,ihole,qbm,dt,dtc,
     1ci,ek,idimp,nppmx,nx,ny,mx,my,nxv,nyv,mx1,mxy1,ntmax,irc)
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 with periodic boundary conditions.
c Using the Boris Mover.
c also determines list of particles which are leaving this tile
c vectorizable/OpenMP version using guard cells
c data read in tiles
c particles stored segmented array
c 131 flops/particle, 4 divides, 2 sqrts, 25 loads, 5 stores
c input: all except ncl, ihole, irc, output: ppart, ncl, ihole, irc, 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 ppart(n,1,m) = position x of particle n in tile m
c ppart(n,2,m) = position y of particle n in tile m
c ppart(n,3,m) = momentum vx of particle n in tile m
c ppart(n,4,m) = momentum vy of particle n in tile m
c ppart(n,5,m) = momentum vz of particle n in tile m
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 kpic(k) = number of particles in tile k
c ncl(i,k) = number of particles going to destination i, tile k
c ihole(1,:,k) = location of hole in array left by departing particle
c ihole(2,:,k) = destination of particle leaving hole
c ihole(1,1,k) = ih, number of holes left (error, if negative)
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 nppmx = maximum number of particles in tile
c nx/ny = system length in x/y direction
c mx/my = number of grids in sorting cell in x/y
c nxv = second dimension of field arrays, must be >= nx+1
c nyv = third dimension of field arrays, must be >= ny+1
c mx1 = (system length in x direction - 1)/mx + 1
c mxy1 = mx1*my1, where my1 = (system length in y direction - 1)/my + 1
c ntmax = size of hole array for particles leaving tiles
c irc = maximum overflow, returned only if error occurs, when irc > 0
c optimized version
      implicit none
      integer idimp, nppmx, nx, ny, mx, my, nxv, nyv, mx1, mxy1, ntmax
      integer irc
      real qbm, dt, dtc, ci, ek
      real ppart, fxy, bxy
      integer kpic, ncl, ihole
      dimension ppart(nppmx,idimp,mxy1)
      dimension fxy(4,nxv*nyv), bxy(4,nxv*nyv)
      dimension kpic(mxy1), ncl(8,mxy1)
      dimension ihole(2,ntmax+1,mxy1)
c local data
      integer MXV, MYV
      parameter(MXV=33,MYV=33)
      integer npblk, lvect
      parameter(npblk=32,lvect=4)
      integer noff, moff, npp, ipp, joff, nps
      integer i, j, k, m, ih, nh, nn, mm, lxv
```

```
      real qtmh, ci2, dxp, dyp, amx, amy, dx, dy, dz, ox, oy, oz
      real acx, acy, acz, p2, gami, qtmg, dtg, omxt, omyt, omzt, omt
      real anorm, rot1, rot2, rot3, rot4, rot5, rot6, rot7, rot8, rot9
      real anx, any, edgelx, edgely, edgerx, edgery
      real x, y, vx, vy, vz
      real sfxy, sbxy
      dimension sfxy(4,MXV*MYV), sbxy(4,MXV*MYV)
c     dimension sfxy(4,(mx+1)*(my+1)), sbxy(4,(mx+1)*(my+1))
c scratch arrays
      integer n
      real s1, s2, t
      dimension n(npblk), s1(npblk,lvect), s2(npblk,lvect), t(npblk,2)
      double precision sum1, sum2
      lxv = mx + 1
      qtmh = 0.5*qbm*dt
      ci2 = ci*ci
      anx = real(nx)
      any = real(ny)
      sum2 = 0.0d0
c error if local array is too small
c     if ((mx.ge.MXV).or.(my.ge.MYV)) return
c loop over tiles
!$OMP PARALLEL DO
!$OMP& PRIVATE(i,j,k,m,noff,moff,npp,nn,mm,ipp,joff,nps,ih,nh,x,y,vx,vy,
!$OMP& vz,dxp,dyp,amx,amy,dx,dy,dz,ox,oy,oz,acx,acy,acz,omxt,omyt,omzt,
!$OMP& omt,anorm,rot1,rot2,rot3,rot4,rot5,rot6,rot7,rot8,rot9,edgelx,
!$OMP& edgely,edgerx,edgery,p2,gami,qtmg,dtg,sum1,sfxy,sbxy,n,s1,s2,t)
!$OMP& REDUCTION(+:sum2)
      do 130 k = 1, mxy1
      noff = (k - 1)/mx1
      moff = my*noff
      noff = mx*(k - mx1*noff - 1)
      npp = kpic(k)
      nn = min(mx,nx-noff)
      mm = min(my,ny-moff)
      edgelx = noff
      edgerx = noff + nn
      edgely = moff
      edgery = moff + mm
      ih = 0
      nh = 0
c load local fields from global arrays
      do 20 j = 1, mm+1
      do 10 i = 1, nn+1
      sfxy(1,i+lxv*(j-1)) = fxy(1,i+noff+nxv*(j+moff-1))
      sfxy(2,i+lxv*(j-1)) = fxy(2,i+noff+nxv*(j+moff-1))
      sfxy(3,i+lxv*(j-1)) = fxy(3,i+noff+nxv*(j+moff-1))
   10 continue
   20 continue
      do 40 j = 1, mm+1
      do 30 i = 1, nn+1
      sbxy(1,i+lxv*(j-1)) = bxy(1,i+noff+nxv*(j+moff-1))
      sbxy(2,i+lxv*(j-1)) = bxy(2,i+noff+nxv*(j+moff-1))
      sbxy(3,i+lxv*(j-1)) = bxy(3,i+noff+nxv*(j+moff-1))
```

```
   30 continue
   40 continue
c clear counters
      do 50 j = 1, 8
      ncl(j,k) = 0
   50 continue
      sum1 = 0.0d0
      ipp = npp/npblk
c outer loop over number of full blocks
      do 110 m = 1, ipp
      joff = npblk*(m - 1)
c inner loop over particles in block
      do 60 j = 1, npblk
c find interpolation weights
      x = ppart(j+joff,1,k)
      y = ppart(j+joff,2,k)
      nn = x
      mm = y
      dxp = x - real(nn)
      dyp = y - real(mm)
      n(j) = nn - noff + lxv*(mm - moff)
      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
   60 continue
c find acceleration
      do 80 j = 1, npblk
      nn = n(j)
      mm = nn + lxv - 2
      dx = 0.0
      dy = 0.0
      dz = 0.0
      ox = 0.0
      oy = 0.0
      oz = 0.0
      do 70 i = 1, lvect
      if (i.gt.2) nn = mm
      dx = dx + sfxy(1,i+nn)*s1(j,i)
      dy = dy + sfxy(2,i+nn)*s1(j,i)
      dz = dz + sfxy(3,i+nn)*s1(j,i)
      ox = ox + sbxy(1,i+nn)*s1(j,i)
      oy = oy + sbxy(2,i+nn)*s1(j,i)
      oz = oz + sbxy(3,i+nn)*s1(j,i)
   70 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
   80 continue
c new momentum
      do 90 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 = ppart(j+joff,3,k) + dx
      acy = ppart(j+joff,4,k) + dy
      acz = ppart(j+joff,5,k) + 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 momentum
      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
```

```
   90 continue
! check boundary conditions
!dir$ novector
      do 100 j = 1, npblk
      dx = s1(j,1)
      dy = s1(j,2)
c find particles going out of bounds
      mm = 0
c count how many particles are going in each direction in ncl
c save their address and destination in ihole
c use periodic boundary conditions and check for roundoff error
c mm = direction particle is going
      if (dx.ge.edgerx) then
         if (dx.ge.anx) dx = dx - anx
         mm = 2
      else if (dx.lt.edgelx) then
         if (dx.lt.0.0) then
            dx = dx + anx
            if (dx.lt.anx) then
               mm = 1
            else
               dx = 0.0
            endif
         else
            mm = 1
         endif
      endif
      if (dy.ge.edgery) then
         if (dy.ge.any) dy = dy - any
         mm = mm + 6
      else if (dy.lt.edgely) then
         if (dy.lt.0.0) then
            dy = dy + any
            if (dy.lt.any) then
               mm = mm + 3
            else
               dy = 0.0
            endif
         else
            mm = mm + 3
         endif
      endif
c set new position
      ppart(j+joff,1,k) = dx
      ppart(j+joff,2,k) = dy
c set new momentum
      ppart(j+joff,3,k) = s2(j,1)
      ppart(j+joff,4,k) = s2(j,2)
      ppart(j+joff,5,k) = s2(j,3)
c increment counters
      if (mm.gt.0) then
         ncl(mm,k) = ncl(mm,k) + 1
         ih = ih + 1
         if (ih.le.ntmax) then
```

```
            ihole(1,ih+1,k) = j + joff
            ihole(2,ih+1,k) = mm
         else
            nh = 1
         endif
      endif
  100 continue
  110 continue
      nps = npblk*ipp + 1
c loop over remaining particles
      do 120 j = nps, npp
c find interpolation weights
      x = ppart(j,1,k)
      y = ppart(j,2,k)
      nn = x
      mm = y
      dxp = x - real(nn)
      dyp = y - real(mm)
      nn = nn - noff + 1 + lxv*(mm - moff)
      amx = 1.0 - dxp
      amy = 1.0 - dyp
c find electric field
      dx = amx*sfxy(1,nn)
      dy = amx*sfxy(2,nn)
      dz = amx*sfxy(3,nn)
      dx = amy*(dxp*sfxy(1,nn+1) + dx)
      dy = amy*(dxp*sfxy(2,nn+1) + dy)
      dz = amy*(dxp*sfxy(3,nn+1) + dz)
      acx = amx*sfxy(1,nn+lxv)
      acy = amx*sfxy(2,nn+lxv)
      acz = amx*sfxy(3,nn+lxv)
      dx = dx + dyp*(dxp*sfxy(1,nn+1+lxv) + acx)
      dy = dy + dyp*(dxp*sfxy(2,nn+1+lxv) + acy)
      dz = dz + dyp*(dxp*sfxy(3,nn+1+lxv) + acz)
c find magnetic field
      ox = amx*sbxy(1,nn)
      oy = amx*sbxy(2,nn)
      oz = amx*sbxy(3,nn)
      ox = amy*(dxp*sbxy(1,nn+1) + ox)
      oy = amy*(dxp*sbxy(2,nn+1) + oy)
      oz = amy*(dxp*sbxy(3,nn+1) + oz)
      acx = amx*sbxy(1,nn+lxv)
      acy = amx*sbxy(2,nn+lxv)
      acz = amx*sbxy(3,nn+lxv)
      ox = ox + dyp*(dxp*sbxy(1,nn+1+lxv) + acx)
      oy = oy + dyp*(dxp*sbxy(2,nn+1+lxv) + acy)
      oz = oz + dyp*(dxp*sbxy(3,nn+1+lxv) + acz)
c calculate half impulse
      dx = qtmh*dx
      dy = qtmh*dy
      dz = qtmh*dz
c half acceleration
      acx = ppart(j,3,k) + dx
      acy = ppart(j,4,k) + dy
```

```
      acz = ppart(j,5,k) + 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 momentum
      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 find particles going out of bounds
      mm = 0
c count how many particles are going in each direction in ncl
c save their address and destination in ihole
c use periodic boundary conditions and check for roundoff error
c mm = direction particle is going
      if (dx.ge.edgerx) then
         if (dx.ge.anx) dx = dx - anx
         mm = 2
      else if (dx.lt.edgelx) then
         if (dx.lt.0.0) then
            dx = dx + anx
            if (dx.lt.anx) then
               mm = 1
            else
               dx = 0.0
```

```
            endif
         else
            mm = 1
         endif
      endif
      if (dy.ge.edgery) then
         if (dy.ge.any) dy = dy - any
         mm = mm + 6
      else if (dy.lt.edgely) then
         if (dy.lt.0.0) then
            dy = dy + any
            if (dy.lt.any) then
               mm = mm + 3
            else
               dy = 0.0
            endif
         else
            mm = mm + 3
         endif
      endif
c set new position
      ppart(j,1,k) = dx
      ppart(j,2,k) = dy
c set new momentum
      ppart(j,3,k) = vx
      ppart(j,4,k) = vy
      ppart(j,5,k) = vz
c increment counters
      if (mm.gt.0) then
         ncl(mm,k) = ncl(mm,k) + 1
         ih = ih + 1
         if (ih.le.ntmax) then
            ihole(1,ih+1,k) = j
            ihole(2,ih+1,k) = mm
         else
            nh = 1
         endif
      endif
  120 continue
      sum2 = sum2 + sum1
c set error and end of file flag
c ihole overflow
      if (nh.gt.0) then
         irc = ih
         ih = -ih
      endif
      ihole(1,1,k) = ih
  130 continue
!$OMP END PARALLEL DO
c normalize kinetic energy
      ek = ek + sum2
      return
      end
```

```
c-----------------------------------------------------------------------
      subroutine VGPPOST2LT(ppart,q,kpic,qm,nppmx,idimp,mx,my,nxv,nyv,
     1mx1,mxy1)
c for 2d code, this subroutine calculates particle charge density
c using first-order linear interpolation, periodic boundaries
c vectorizable/OpenMP version using guard cells
c data deposited in tiles
c particles stored segmented array
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 ppart(n,1,m) = position x of particle n in tile m
c ppart(n,2,m) = position y of particle n in tile m
c q(j,k) = charge density at grid point j,k
c kpic = number of particles per tile
c qm = charge on particle, in units of e
c nppmx = maximum number of particles in tile
c idimp = size of phase space = 4
c mx/my = number of grids in sorting cell in x/y
c nxv = first dimension of charge array, must be >= nx+1
c nyv = second dimension of charge array, must be >= ny+1
c mx1 = (system length in x direction - 1)/mx + 1
c mxy1 = mx1*my1, where my1 = (system length in y direction - 1)/my + 1
      implicit none
      integer nppmx, idimp, mx, my, nxv, nyv, mx1, mxy1
      real qm
      real ppart, q
      integer kpic
      dimension ppart(nppmx,idimp,mxy1), q(nxv*nyv)
      dimension kpic(mxy1)
c local data
      integer MXV, MYV
      parameter(MXV=33,MYV=33)
      integer npblk, lvect
      parameter(npblk=32,lvect=4)
      integer noff, moff, npp, ipp, joff, nps
      integer i, j, k, m, nn, mm, lxv
      real x, y, dxp, dyp, amx, amy
      real sq
c     dimension sq(MXV*MYV)
      dimension sq((mx+1)*(my+1))
c scratch arrays
      integer n
      real s
      dimension n(npblk), s(npblk,lvect)
      lxv = mx + 1
c error if local array is too small
c     if ((mx.ge.MXV).or.(my.ge.MYV)) return
c loop over tiles
```

```
!$OMP PARALLEL DO
!$OMP& PRIVATE(i,j,k,m,noff,moff,npp,ipp,joff,nps,nn,mm,x,y,dxp,dyp,amx,
!$OMP& amy,sq,n,s)
      do 110 k = 1, mxy1
      noff = (k - 1)/mx1
      moff = my*noff
      noff = mx*(k - mx1*noff - 1)
      npp = kpic(k)
c zero out local accumulator
      do 10 j = 1, (mx+1)*(my+1)
      sq(j) = 0.0
   10 continue
c loop over particles in tile
      ipp = npp/npblk
c outer loop over number of full blocks
      do 50 m = 1, ipp
      joff = npblk*(m - 1)
c inner loop over particles in block
      do 20 j = 1, npblk
c find interpolation weights
      x = ppart(j+joff,1,k)
      y = ppart(j+joff,2,k)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
      dyp = y - real(mm)
      n(j) = nn - noff + lxv*(mm - moff)
      amx = qm - dxp
      amy = 1.0 - dyp
      s(j,1) = amx*amy
      s(j,2) = dxp*amy
      s(j,3) = amx*dyp
      s(j,4) = dxp*dyp
   20 continue
c deposit charge within tile to local accumulator
      do 40 j = 1, npblk
      nn = n(j)
      mm = nn + lxv - 2
!dir$ ivdep
      do 30 i = 1, lvect
      if (i.gt.2) nn = mm
      sq(i+nn) = sq(i+nn) + s(j,i)
   30 continue
   40 continue
   50 continue
      nps = npblk*ipp + 1
c loop over remaining particles
      do 60 j = nps, npp
c find interpolation weights
      x = ppart(j,1,k)
      y = ppart(j,2,k)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
```

```
      dyp = y - real(mm)
      nn = nn - noff + 1 + lxv*(mm - moff)
      amx = qm - dxp
      amy = 1.0 - dyp
c deposit charge within tile to local accumulator
      x = sq(nn) + amx*amy
      y = sq(nn+1) + dxp*amy
      sq(nn) = x
      sq(nn+1) = y
      x = sq(nn+lxv) + amx*dyp
      y = sq(nn+1+lxv) + dxp*dyp
      sq(nn+lxv) = x
      sq(nn+1+lxv) = y
   60 continue
c deposit charge to interior points in global array
      nn = min(mx,nxv-noff)
      mm = min(my,nyv-moff)
      do 80 j = 2, mm
      do 70 i = 2, nn
      q(i+noff+nxv*(j+moff-1)) = q(i+noff+nxv*(j+moff-1)) +
     1sq(i+lxv*(j-1))
   70 continue
   80 continue
c deposit charge to edge points in global array
      mm = min(my+1,nyv-moff)
      do 90 i = 2, nn
!$OMP ATOMIC
      q(i+noff+nxv*moff) = q(i+noff+nxv*moff) + sq(i)
      if (mm > my) then
!$OMP ATOMIC
         q(i+noff+nxv*(mm+moff-1)) = q(i+noff+nxv*(mm+moff-1)) +
     1sq(i+lxv*(mm-1))
      endif
   90 continue
      nn = min(mx+1,nxv-noff)
      do 100 j = 1, mm
!$OMP ATOMIC
      q(1+noff+nxv*(j+moff-1)) = q(1+noff+nxv*(j+moff-1)) +
     1sq(1+lxv*(j-1))
      if (nn > mx) then
!$OMP ATOMIC
         q(nn+noff+nxv*(j+moff-1)) = q(nn+noff+nxv*(j+moff-1)) +
     1sq(nn+lxv*(j-1))
      endif
  100 continue
  110 continue
!$OMP END PARALLEL DO
      return
      end
```

```
c-----------------------------------------------------------------------
      subroutine VGRJPPOSTF2LT(ppart,cu,kpic,ncl,ihole,qm,dt,ci,nppmx,
     1idimp,nx,ny,mx,my,nxv,nyv,mx1,mxy1,ntmax,irc)
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 with periodic boundary conditions.
c also determines list of particles which are leaving this tile
c vectorizable/OpenMP version using guard cells
c data deposited in tiles
c particles stored segmented array
c 47 flops/particle, 1 divide, 1 sqrt, 17 loads, 14 stores
c input: all except ncl, ihole, irc,
c output: ppart, cu, ncl, ihole, irc
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 ppart(n,1,m) = position x of particle n in tile m
c ppart(n,2,m) = position y of particle n in tile m
c ppart(n,3,m) = x momentum of particle n in tile m
c ppart(n,4,m) = y momentum of particle n in tile m
c ppart(n,5,m) = z momentum of particle n in tile m
c cu(i,j,k) = ith component of current density at grid point j,k
c kpic(k) = number of particles in tile k
c ncl(i,k) = number of particles going to destination i, tile k
c ihole(1,:,k) = location of hole in array left by departing particle
c ihole(2,:,k) = destination of particle leaving hole
c ihole(1,1,k) = ih, number of holes left (error, if negative)
c qm = charge on particle, in units of e
c dt = time interval between successive calculations
c ci = reciprocal of velocity of light
c nppmx = maximum number of particles in tile
c idimp = size of phase space = 5
c nx/ny = system length in x/y direction
c mx/my = number of grids in sorting cell in x/y
c nxv = second dimension of current array, must be >= nx+1
c nyv = third dimension of current array, must be >= ny+1
c mx1 = (system length in x direction - 1)/mx + 1
c mxy1 = mx1*my1, where my1 = (system length in y direction - 1)/my + 1
c ntmax = size of hole array for particles leaving tiles
c irc = maximum overflow, returned only if error occurs, when irc > 0
c optimized version
      implicit none
      integer nppmx, idimp, nx, ny, mx, my, nxv, nyv, mx1, mxy1, ntmax
      integer irc
      real qm, dt, ci
      real ppart, cu
      integer kpic, ncl, ihole
      dimension ppart(nppmx,idimp,mxy1), cu(4,nxv*nyv)
```

```
      dimension kpic(mxy1), ncl(8,mxy1)
      dimension ihole(2,ntmax+1,mxy1)
c local data
      integer MXV, MYV
      parameter(MXV=33,MYV=33)
      integer npblk, lvect
      parameter(npblk=32,lvect=4)
      integer noff, moff, npp, ipp, joff, nps
      integer i, j, k, m, ih, nh, nn, mm, lxv
      real ci2, dxp, dyp, amx, amy
      real x, y, dx, dy, vx, vy, vz, p2, gami
      real anx, any, edgelx, edgely, edgerx, edgery
      real scu
      dimension scu(4,MXV*MYV)
c     dimension scu(4,(mx+1)*(my+1))
c scratch arrays
      integer n
      real s1, s2, t
      dimension n(npblk), s1(npblk,lvect), s2(npblk,lvect), t(npblk,2)
      lxv = mx + 1
      ci2 = ci*ci
      anx = real(nx)
      any = real(ny)
c error if local array is too small
c     if ((mx.ge.MXV).or.(my.ge.MYV)) return
c loop over tiles
!$OMP PARALLEL DO
!$OMP& PRIVATE(i,j,k,m,noff,moff,npp,ipp,joff,nps,nn,mm,ih,nh,x,y,dxp,
!$OMP& dyp,amx,amy,dx,dy,vx,vy,vz,edgelx,edgely,edgerx,edgery,p2,gami,
!$OMP& scu,n,s1,s2,t)
      do 130 k = 1, mxy1
      noff = (k - 1)/mx1
      moff = my*noff
      noff = mx*(k - mx1*noff - 1)
      npp = kpic(k)
      nn = min(mx,nx-noff)
      mm = min(my,ny-moff)
      edgelx = noff
      edgerx = noff + nn
      edgely = moff
      edgery = moff + mm
      ih = 0
      nh = 0
c zero out local accumulator
      nn = lxv*(my + 1)
      do 10 i = 1, nn
      scu(1,i) = 0.0
      scu(2,i) = 0.0
      scu(3,i) = 0.0
   10 continue
c clear counters
      do 20 j = 1, 8
      ncl(j,k) = 0
   20 continue
```

```
      ipp = npp/npblk
c outer loop over number of full blocks
      do 70 m = 1, ipp
      joff = npblk*(m - 1)
c inner loop over particles in block
      do 30 j = 1, npblk
c find interpolation weights
      x = ppart(j+joff,1,k)
      y = ppart(j+joff,2,k)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
      dyp = y - real(mm)
      n(j) = nn - noff + lxv*(mm - moff)
      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
      vx = ppart(j+joff,3,k)
      vy = ppart(j+joff,4,k)
      vz = ppart(j+joff,5,k)
      p2 = vx*vx + vy*vy + vz*vz
      gami = 1.0/sqrt(1.0 + p2*ci2)
      s2(j,1) = vx*gami
      s2(j,2) = vy*gami
      s2(j,3) = vz*gami
   30 continue
c deposit current
      do 50 j = 1, npblk
      nn = n(j)
      mm = nn + lxv - 2
      vx = s2(j,1)
      vy = s2(j,2)
      vz = s2(j,3)
!dir$ ivdep
      do 40 i = 1, lvect
      if (i.gt.2) nn = mm
      scu(1,i+nn) = scu(1,i+nn) + vx*s1(j,i)
      scu(2,i+nn) = scu(2,i+nn) + vy*s1(j,i)
      scu(3,i+nn) = scu(3,i+nn) + vz*s1(j,i)
   40 continue
   50 continue
c advance position half a time-step
!dir$ novector
      do 60 j = 1, npblk
      dx = t(j,1) + s2(j,1)*dt
      dy = t(j,2) + s2(j,2)*dt
c find particles going out of bounds
      mm = 0
```

```
c count how many particles are going in each direction in ncl
c save their address and destination in ihole
c use periodic boundary conditions and check for roundoff error
c mm = direction particle is going
      if (dx.ge.edgerx) then
         if (dx.ge.anx) dx = dx - anx
         mm = 2
      else if (dx.lt.edgelx) then
         if (dx.lt.0.0) then
            dx = dx + anx
            if (dx.lt.anx) then
               mm = 1
            else
               dx = 0.0
            endif
         else
            mm = 1
         endif
      endif
      if (dy.ge.edgery) then
         if (dy.ge.any) dy = dy - any
         mm = mm + 6
      else if (dy.lt.edgely) then
         if (dy.lt.0.0) then
            dy = dy + any
            if (dy.lt.any) then
               mm = mm + 3
            else
               dy = 0.0
            endif
         else
            mm = mm + 3
         endif
      endif
c set new position
      ppart(j+joff,1,k) = dx
      ppart(j+joff,2,k) = dy
c increment counters
      if (mm.gt.0) then
         ncl(mm,k) = ncl(mm,k) + 1
         ih = ih + 1
         if (ih.le.ntmax) then
            ihole(1,ih+1,k) = j + joff
            ihole(2,ih+1,k) = mm
         else
            nh = 1
         endif
      endif
   60 continue
   70 continue
      nps = npblk*ipp + 1
c loop over remaining particles
      do 80 j = nps, npp
c find interpolation weights
```

```
      x = ppart(j,1,k)
      y = ppart(j,2,k)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
      dyp = y - real(mm)
c find inverse gamma
      vx = ppart(j,3,k)
      vy = ppart(j,4,k)
      vz = ppart(j,5,k)
      p2 = vx*vx + vy*vy + vz*vz
      gami = 1.0/sqrt(1.0 + p2*ci2)
c calculate weights
      nn = nn - noff + 1 + lxv*(mm - moff)
      amx = qm - dxp
      amy = 1.0 - dyp
c deposit current
      dx = amx*amy
      dy = dxp*amy
      vx = vx*gami
      vy = vy*gami
      vz = vz*gami
      scu(1,nn) = scu(1,nn) + vx*dx
      scu(2,nn) = scu(2,nn) + vy*dx
      scu(3,nn) = scu(3,nn) + vz*dx
      dx = amx*dyp
      scu(1,nn+1) = scu(1,nn+1) + vx*dy
      scu(2,nn+1) = scu(2,nn+1) + vy*dy
      scu(3,nn+1) = scu(3,nn+1) + vz*dy
      dy = dxp*dyp
      scu(1,nn+lxv) = scu(1,nn+lxv) + vx*dx
      scu(2,nn+lxv) = scu(2,nn+lxv) + vy*dx
      scu(3,nn+lxv) = scu(3,nn+lxv) + vz*dx
      scu(1,nn+1+lxv) = scu(1,nn+1+lxv) + vx*dy
      scu(2,nn+1+lxv) = scu(2,nn+1+lxv) + vy*dy
      scu(3,nn+1+lxv) = scu(3,nn+1+lxv) + vz*dy
c advance position half a time-step
      dx = x + vx*dt
      dy = y + vy*dt
c find particles going out of bounds
      mm = 0
c count how many particles are going in each direction in ncl
c save their address and destination in ihole
c use periodic boundary conditions and check for roundoff error
c mm = direction particle is going
      if (dx.ge.edgerx) then
         if (dx.ge.anx) dx = dx - anx
         mm = 2
      else if (dx.lt.edgelx) then
         if (dx.lt.0.0) then
            dx = dx + anx
            if (dx.lt.anx) then
               mm = 1
            else
```

```
               dx = 0.0
            endif
         else
            mm = 1
         endif
      endif
      if (dy.ge.edgery) then
         if (dy.ge.any) dy = dy - any
         mm = mm + 6
      else if (dy.lt.edgely) then
         if (dy.lt.0.0) then
            dy = dy + any
            if (dy.lt.any) then
               mm = mm + 3
            else
               dy = 0.0
            endif
         else
            mm = mm + 3
         endif
      endif
c set new position
      ppart(j,1,k) = dx
      ppart(j,2,k) = dy
c increment counters
      if (mm.gt.0) then
         ncl(mm,k) = ncl(mm,k) + 1
         ih = ih + 1
         if (ih.le.ntmax) then
            ihole(1,ih+1,k) = j
            ihole(2,ih+1,k) = mm
         else
            nh = 1
         endif
      endif
   80 continue
c deposit current to interior points in global array
      nn = min(mx,nxv-noff)
      mm = min(my,nyv-moff)
      do 100 j = 2, mm
      do 90 i = 2, nn
      cu(1,i+noff+nxv*(j+moff-1)) = cu(1,i+noff+nxv*(j+moff-1))
     1 + scu(1,i+lxv*(j-1))
      cu(2,i+noff+nxv*(j+moff-1)) = cu(2,i+noff+nxv*(j+moff-1))
     1 + scu(2,i+lxv*(j-1))
      cu(3,i+noff+nxv*(j+moff-1)) = cu(3,i+noff+nxv*(j+moff-1))
     1 + scu(3,i+lxv*(j-1))
   90 continue
  100 continue
c deposit current to edge points in global array
      mm = min(my+1,nyv-moff)
      do 110 i = 2, nn
!$OMP ATOMIC
      cu(1,i+noff+nxv*moff) = cu(1,i+noff+nxv*moff) + scu(1,i)
```

```
!$OMP ATOMIC
      cu(2,i+noff+nxv*moff) = cu(2,i+noff+nxv*moff) + scu(2,i)
!$OMP ATOMIC
      cu(3,i+noff+nxv*moff) = cu(3,i+noff+nxv*moff) + scu(3,i)
      if (mm > my) then
!$OMP ATOMIC
         cu(1,i+noff+nxv*(mm+moff-1)) = cu(1,i+noff+nxv*(mm+moff-1))
     1    + scu(1,i+lxv*(mm-1))
!$OMP ATOMIC
         cu(2,i+noff+nxv*(mm+moff-1)) = cu(2,i+noff+nxv*(mm+moff-1))
     1    + scu(2,i+lxv*(mm-1))
!$OMP ATOMIC
         cu(3,i+noff+nxv*(mm+moff-1)) = cu(3,i+noff+nxv*(mm+moff-1))
     1    + scu(3,i+lxv*(mm-1))
      endif
  110 continue
      nn = min(mx+1,nxv-noff)
      do 120 j = 1, mm
!$OMP ATOMIC
      cu(1,1+noff+nxv*(j+moff-1)) = cu(1,1+noff+nxv*(j+moff-1))
     1 + scu(1,1+lxv*(j-1))
!$OMP ATOMIC
      cu(2,1+noff+nxv*(j+moff-1)) = cu(2,1+noff+nxv*(j+moff-1))
     1 + scu(2,1+lxv*(j-1))
!$OMP ATOMIC
      cu(3,1+noff+nxv*(j+moff-1)) = cu(3,1+noff+nxv*(j+moff-1))
     1 + scu(3,1+lxv*(j-1))
      if (nn > mx) then
!$OMP ATOMIC
         cu(1,nn+noff+nxv*(j+moff-1)) = cu(1,nn+noff+nxv*(j+moff-1))
     1    + scu(1,nn+lxv*(j-1))
!$OMP ATOMIC
         cu(2,nn+noff+nxv*(j+moff-1)) = cu(2,nn+noff+nxv*(j+moff-1))
     1    + scu(2,nn+lxv*(j-1))
!$OMP ATOMIC
         cu(3,nn+noff+nxv*(j+moff-1)) = cu(3,nn+noff+nxv*(j+moff-1))
     1    + scu(3,nn+lxv*(j-1))
      endif
  120 continue
c set error and end of file flag
c ihole overflow
      if (nh.gt.0) then
         irc = ih
         ih = -ih
      endif
      ihole(1,1,k) = ih
  130 continue
!$OMP END PARALLEL DO
      return
      end
```