```
c-----------------------------------------------------------------------
      subroutine PPGRBPPUSHF23L(ppart,fxy,bxy,kpic,ncl,ihole,noff,nyp,qb
     1m,dt,dtc,ci,ek,idimp,nppmx,nx,ny,mx,my,nxv,nypmx,mx1,mxyp1,ntmax,i
     2rc)
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 with periodic boundary conditions.
c also determines list of particles which are leaving this tile
c OpenMP version using guard cells, for distributed data
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, ek, irc
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(1,n,m) = position x of particle n in partition in tile m
c ppart(2,n,m) = position y of particle n in partition in tile m
c ppart(3,n,m) = momentum vx of particle n in partition in tile m
c ppart(4,n,m) = momentum vy of particle n in partition in tile m
c ppart(5,n,m) = momentum vz of particle n in partition in tile m
c fxy(1,j,k) = x component of force/charge at grid (j,kk)
c fxy(2,j,k) = y component of force/charge at grid (j,kk)
c fxy(3,j,k) = z component of force/charge at grid (j,kk)
c that is, convolution of electric field over particle shape,
c where kk = k + noff - 1
c bxy(1,j,k) = x component of magnetic field at grid (j,kk)
c bxy(2,j,k) = y component of magnetic field at grid (j,kk)
c bxy(3,j,k) = z component of magnetic field at grid (j,kk)
c that is, the convolution of magnetic field over particle shape,
c where kk = k + noff - 1
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 noff = lowermost global gridpoint in particle partition.
c nyp = number of primary (complete) gridpoints in particle partition
c qbm = particle charge/mass ratio
c dt = time interval between successive calculations
c dtc = time interval between successive co-ordinate calculations
c ci = reciprical 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 = first dimension of field arrays, must be >= nx+1
c nypmx = maximum size of particle partition, including guard cells.
c mx1 = (system length in x direction - 1)/mx + 1
c mxyp1 = mx1*myp1, where myp1=(partition 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 noff, nyp, idimp, nppmx, nx, ny, mx, my, nxv, nypmx
      integer mx1, mxyp1, ntmax, irc
      real qbm, dt, dtc, ci, ek
      real ppart, fxy, bxy
      integer kpic, ncl, ihole
      dimension ppart(idimp,nppmx,mxyp1)
      dimension fxy(3,nxv,nypmx), bxy(3,nxv,nypmx)
      dimension kpic(mxyp1), ncl(8,mxyp1)
      dimension ihole(2,ntmax+1,mxyp1)
c local data
      integer MXV, MYV
```

```
      parameter(MXV=33,MYV=33)
      integer noffp, moffp, nppp
      integer mnoff, i, j, k, ih, nh, nn, mm
      real qtmh, ci2, 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 anx, any, edgelx, edgely, edgerx, edgery
      real x, y
      real sfxy, sbxy
      dimension sfxy(3,MXV,MYV), sbxy(3,MXV,MYV)
c     dimension sfxy(3,mx+1,my+1), sbxy(3,mx+1,my+1)
      double precision sum1, sum2
      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,noffp,moffp,nppp,nn,mm,ih,nh,mnoff,x,y,dxp,dyp,amx,
!$OMP& amy,dx,dy,dz,ox,oy,oz,acx,acy,acz,omxt,omyt,omzt,omt,anorm,rot1,
!$OMP& rot2,rot3,rot4,rot5,rot6,rot7,rot8,rot9,edgelx,edgely,edgerx,
!$OMP& edgery,p2,gami,qtmg,dtg,sum1,sfxy,sbxy)
!$OMP& REDUCTION(+:sum2)
      do 70 k = 1, mxyp1
      noffp = (k - 1)/mx1
      moffp = my*noffp
      noffp = mx*(k - mx1*noffp - 1)
      nppp = kpic(k)
      nn = min(mx,nx-noffp)
      mm = min(my,nyp-moffp)
      edgelx = noffp
      edgerx = noffp + nn
      edgely = noff + moffp
      edgery = noff + moffp + mm
      ih = 0
      nh = 0
      mnoff = moffp + noff - 1
c load local fields from global arrays
      do 20 j = 1, mm+1
      do 10 i = 1, nn+1
      sfxy(1,i,j) = fxy(1,i+noffp,j+moffp)
      sfxy(2,i,j) = fxy(2,i+noffp,j+moffp)
      sfxy(3,i,j) = fxy(3,i+noffp,j+moffp)
   10 continue
   20 continue
      do 40 j = 1, mm+1
      do 30 i = 1, nn+1
      sbxy(1,i,j) = bxy(1,i+noffp,j+moffp)
      sbxy(2,i,j) = bxy(2,i+noffp,j+moffp)
      sbxy(3,i,j) = bxy(3,i+noffp,j+moffp)
```

```
   30 continue
   40 continue
c clear counters
      do 50 j = 1, 8
      ncl(j,k) = 0
   50 continue
      sum1 = 0.0d0
c loop over particles in tile
      do 60 j = 1, nppp
c find interpolation weights
      x = ppart(1,j,k)
      y = ppart(2,j,k)
      nn = x
      mm = y
      dxp = x - real(nn)
      dyp = y - real(mm)
      nn = nn - noffp + 1
      mm = mm - mnoff
      amx = 1.0 - dxp
      amy = 1.0 - dyp
c find electric field
      dx = amx*sfxy(1,nn,mm)
      dy = amx*sfxy(2,nn,mm)
      dz = amx*sfxy(3,nn,mm)
      dx = amy*(dxp*sfxy(1,nn+1,mm) + dx)
      dy = amy*(dxp*sfxy(2,nn+1,mm) + dy)
      dz = amy*(dxp*sfxy(3,nn+1,mm) + dz)
      acx = amx*sfxy(1,nn,mm+1)
      acy = amx*sfxy(2,nn,mm+1)
      acz = amx*sfxy(3,nn,mm+1)
      dx = dx + dyp*(dxp*sfxy(1,nn+1,mm+1) + acx)
      dy = dy + dyp*(dxp*sfxy(2,nn+1,mm+1) + acy)
      dz = dz + dyp*(dxp*sfxy(3,nn+1,mm+1) + acz)
c find magnetic field
      ox = amx*sbxy(1,nn,mm)
      oy = amx*sbxy(2,nn,mm)
      oz = amx*sbxy(3,nn,mm)
      ox = amy*(dxp*sbxy(1,nn+1,mm) + ox)
      oy = amy*(dxp*sbxy(2,nn+1,mm) + oy)
      oz = amy*(dxp*sbxy(3,nn+1,mm) + oz)
      acx = amx*sbxy(1,nn,mm+1)
      acy = amx*sbxy(2,nn,mm+1)
      acz = amx*sbxy(3,nn,mm+1)
      ox = ox + dyp*(dxp*sbxy(1,nn+1,mm+1) + acx)
      oy = oy + dyp*(dxp*sbxy(2,nn+1,mm+1) + acy)
      oz = oz + dyp*(dxp*sbxy(3,nn+1,mm+1) + acz)
c calculate half impulse
      dx = qtmh*dx
      dy = qtmh*dy
      dz = qtmh*dz
c half acceleration
      acx = ppart(3,j,k) + dx
      acy = ppart(4,j,k) + dy
      acz = ppart(5,j,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
      dx = (rot1*acx + rot2*acy + rot3*acz)*anorm + dx
      dy = (rot4*acx + rot5*acy + rot6*acz)*anorm + dy
      dz = (rot7*acx + rot8*acy + rot9*acz)*anorm + dz
      ppart(3,j,k) = dx
      ppart(4,j,k) = dy
      ppart(5,j,k) = dz
c update inverse gamma
      p2 = dx*dx + dy*dy + dz*dz
      dtg = dtc/sqrt(1.0 + p2*ci2)
c new position
      dx = x + dx*dtg
      dy = y + dy*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(1,j,k) = dx
      ppart(2,j,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
      sum2 = sum2 + sum1
   60 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
   70 continue
!$OMP END PARALLEL DO
c normalize kinetic energy
      ek = ek + sum2
      return
      end
```

```
c-----------------------------------------------------------------------
      subroutine PPGPPOST2L(ppart,q,kpic,noff,qm,idimp,nppmx,mx,my,nxv,
     1nypmx,mx1,mxyp1)
c for 2d code, this subroutine calculates particle charge density
c using first-order linear interpolation, periodic boundaries
c OpenMP version using guard cells, for distributed data
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(1,n,m) = position x of particle n in partition in tile m
c ppart(2,n,m) = position y of particle n in partition in tile m
c q(j,k) = charge density at grid point (j,kk),
c where kk = k + noff - 1
c kpic = number of particles per tile
c noff = lowermost global gridpoint in particle partition.
c qm = charge on particle, in units of e
c idimp = size of phase space = 4
c nppmx = maximum number of particles in tile
c mx/my = number of grids in sorting cell in x/y
c nxv = first dimension of charge array, must be >= nx+1
c nypmx = maximum size of particle partition, including guard cells.
c mx1 = (system length in x direction - 1)/mx + 1
c mxyp1 = mx1*myp1, where myp1=(partition length in y direction-1)/my+1
      implicit none
      integer noff, idimp, nppmx, mx, my, nxv, nypmx, mx1, mxyp1
      real qm
      real ppart, q
      integer kpic
      dimension ppart(idimp,nppmx,mxyp1), q(nxv,nypmx), kpic(mxyp1)
c local data
      integer MXV, MYV
      parameter(MXV=33,MYV=33)
      integer noffp, moffp, nppp
      integer mnoff, i, j, k, nn, mm
      real x, y, dxp, dyp, amx, amy
      real sq
c     dimension sq(MXV,MYV)
      dimension sq(mx+1,my+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,noffp,moffp,nppp,mnoff,nn,mm,x,y,dxp,dyp,amx,amy,
!$OMP& sq)
      do 80 k = 1, mxyp1
      noffp = (k - 1)/mx1
      moffp = my*noffp
```

```
      noffp = mx*(k - mx1*noffp - 1)
      nppp = kpic(k)
      mnoff = moffp + noff - 1
c zero out local accumulator
      do 20 j = 1, my+1
      do 10 i = 1, mx+1
      sq(i,j) = 0.0
   10 continue
   20 continue
c loop over particles in tile
      do 30 j = 1, nppp
c find interpolation weights
      x = ppart(1,j,k)
      y = ppart(2,j,k)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
      dyp = y - real(mm)
      nn = nn - noffp + 1
      mm = mm - mnoff
      amx = qm - dxp
      amy = 1.0 - dyp
c deposit charge within tile to local accumulator
      x = sq(nn,mm) + amx*amy
      y = sq(nn+1,mm) + dxp*amy
      sq(nn,mm) = x
      sq(nn+1,mm) = y
      x = sq(nn,mm+1) + amx*dyp
      y = sq(nn+1,mm+1) + dxp*dyp
      sq(nn,mm+1) = x
      sq(nn+1,mm+1) = y
   30 continue
c deposit charge to interior points in global array
      nn = min(mx,nxv-noffp)
      mm = min(my,nypmx-moffp)
      do 50 j = 2, mm
      do 40 i = 2, nn
      q(i+noffp,j+moffp) = q(i+noffp,j+moffp) + sq(i,j)
   40 continue
   50 continue
c deposit charge to edge points in global array
      mm = min(my+1,nypmx-moffp)
      do 60 i = 2, nn
!$OMP ATOMIC
      q(i+noffp,1+moffp) = q(i+noffp,1+moffp) + sq(i,1)
      if (mm > my) then
!$OMP ATOMIC
         q(i+noffp,mm+moffp) = q(i+noffp,mm+moffp) + sq(i,mm)
      endif
   60 continue
      nn = min(mx+1,nxv-noffp)
      do 70 j = 1, mm
!$OMP ATOMIC
      q(1+noffp,j+moffp) = q(1+noffp,j+moffp) + sq(1,j)
```

```
      if (nn > mx) then
!$OMP ATOMIC
         q(nn+noffp,j+moffp) = q(nn+noffp,j+moffp) + sq(nn,j)
      endif
   70 continue
   80 continue
!$OMP END PARALLEL DO
      return
      end
```

```
c-----------------------------------------------------------------------
      subroutine PPGRJPPOSTF2L(ppart,cu,kpic,ncl,ihole,noff,nyp,qm,dt,ci
     1,nppmx,idimp,nx,ny,mx,my,nxv,nypmx,mx1,mxyp1,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 OpenMP version using guard cells, for distributed data
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(1,n,m) = position x of particle n in partition in tile m
c ppart(2,n,m) = position y of particle n in partition in tile m
c ppart(3,n,m) = x momentum of particle n in partition in tile m
c ppart(4,n,m) = y momentum of particle n in partition in tile m
c ppart(5,n,m) = z momentum of particle n in partition in tile m
c cu(i,j,k) = ith component of current density at grid point (j,kk),
c where kk = k + noff - 1
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 noff = lowermost global gridpoint in particle partition.
c nyp = number of primary (complete) gridpoints in particle partition
c qm = charge on particle, in units of e
c dt = time interval between successive calculations
c ci = reciprical 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 = first dimension of current array, must be >= nx+1
c nypmx = maximum size of particle partition, including guard cells.
c mx1 = (system length in x direction - 1)/mx + 1
c mxyp1 = mx1*myp1, where myp1=(partition 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 noff, nyp, nppmx, idimp, nx, ny, mx, my, nxv, nypmx, mx1
      integer mxyp1, ntmax, irc
      real qm, dt, ci
```

```
      real ppart, cu
      integer kpic, ncl, ihole
      dimension ppart(idimp,nppmx,mxyp1), cu(3,nxv,nypmx)
      dimension kpic(mxyp1), ncl(8,mxyp1)
      dimension ihole(2,ntmax+1,mxyp1)
c local data
      integer MXV, MYV
      parameter(MXV=33,MYV=33)
      integer noffp, moffp, nppp
      integer mnoff, i, j, k, nn, mm, ih, nh
      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(3,MXV,MYV)
c     dimension scu(3,mx+1,my+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,noffp,moffp,nppp,nn,mm,ih,nh,mnoff,x,y,dxp,dyp,amx,
!$OMP& amy,dx,dy,vx,vy,vz,edgelx,edgely,edgerx,edgery,p2,gami,scu)
      do 90 k = 1, mxyp1
      noffp = (k - 1)/mx1
      moffp = my*noffp
      noffp = mx*(k - mx1*noffp - 1)
      nppp = kpic(k)
      nn = min(mx,nx-noffp)
      mm = min(my,nyp-moffp)
      edgelx = noffp
      edgerx = noffp + nn
      edgely = noff + moffp
      edgery = noff + moffp + mm
      ih = 0
      nh = 0
      mnoff = moffp + noff - 1
c zero out local accumulator
      do 20 j = 1, my+1
      do 10 i = 1, mx+1
      scu(1,i,j) = 0.0
      scu(2,i,j) = 0.0
      scu(3,i,j) = 0.0
   10 continue
   20 continue
c clear counters
      do 30 j = 1, 8
      ncl(j,k) = 0
   30 continue
c loop over particles in tile
      do 40 j = 1, nppp
c find interpolation weights
```

```
         x = ppart(1,j,k)
         y = ppart(2,j,k)
         nn = x
         mm = y
         dxp = qm*(x - real(nn))
         dyp = y - real(mm)
c find inverse gamma
         vx = ppart(3,j,k)
         vy = ppart(4,j,k)
         vz = ppart(5,j,k)
         p2 = vx*vx + vy*vy + vz*vz
         gami = 1.0/sqrt(1.0 + p2*ci2)
c calculate weights
         nn = nn - noffp + 1
         mm = mm - mnoff
         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,mm) = scu(1,nn,mm) + vx*dx
         scu(2,nn,mm) = scu(2,nn,mm) + vy*dx
         scu(3,nn,mm) = scu(3,nn,mm) + vz*dx
         dx = amx*dyp
         scu(1,nn+1,mm) = scu(1,nn+1,mm) + vx*dy
         scu(2,nn+1,mm) = scu(2,nn+1,mm) + vy*dy
         scu(3,nn+1,mm) = scu(3,nn+1,mm) + vz*dy
         dy = dxp*dyp
         scu(1,nn,mm+1) = scu(1,nn,mm+1) + vx*dx
         scu(2,nn,mm+1) = scu(2,nn,mm+1) + vy*dx
         scu(3,nn,mm+1) = scu(3,nn,mm+1) + vz*dx
         scu(1,nn+1,mm+1) = scu(1,nn+1,mm+1) + vx*dy
         scu(2,nn+1,mm+1) = scu(2,nn+1,mm+1) + vy*dy
         scu(3,nn+1,mm+1) = scu(3,nn+1,mm+1) + 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(1,j,k) = dx
      ppart(2,j,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
   40 continue
c deposit current to interior points in global array
      nn = min(mx,nxv-noffp)
      mm = min(my,nypmx-moffp)
      do 60 j = 2, mm
      do 50 i = 2, nn
      cu(1,i+noffp,j+moffp) = cu(1,i+noffp,j+moffp) + scu(1,i,j)
      cu(2,i+noffp,j+moffp) = cu(2,i+noffp,j+moffp) + scu(2,i,j)
      cu(3,i+noffp,j+moffp) = cu(3,i+noffp,j+moffp) + scu(3,i,j)
   50 continue
   60 continue
c deposit current to edge points in global array
      mm = min(my+1,nypmx-moffp)
      do 70 i = 2, nn
!$OMP ATOMIC
      cu(1,i+noffp,1+moffp) = cu(1,i+noffp,1+moffp) + scu(1,i,1)
!$OMP ATOMIC
      cu(2,i+noffp,1+moffp) = cu(2,i+noffp,1+moffp) + scu(2,i,1)
```

```
!$OMP ATOMIC
      cu(3,i+noffp,1+moffp) = cu(3,i+noffp,1+moffp) + scu(3,i,1)
      if (mm > my) then
!$OMP ATOMIC
         cu(1,i+noffp,mm+moffp) = cu(1,i+noffp,mm+moffp) + scu(1,i,mm)
!$OMP ATOMIC
         cu(2,i+noffp,mm+moffp) = cu(2,i+noffp,mm+moffp) + scu(2,i,mm)
!$OMP ATOMIC
         cu(3,i+noffp,mm+moffp) = cu(3,i+noffp,mm+moffp) + scu(3,i,mm)
      endif
   70 continue
      nn = min(mx+1,nxv-noffp)
      do 80 j = 1, mm
!$OMP ATOMIC
      cu(1,1+noffp,j+moffp) = cu(1,1+noffp,j+moffp) + scu(1,1,j)
!$OMP ATOMIC
      cu(2,1+noffp,j+moffp) = cu(2,1+noffp,j+moffp) + scu(2,1,j)
!$OMP ATOMIC
      cu(3,1+noffp,j+moffp) = cu(3,1+noffp,j+moffp) + scu(3,1,j)
      if (nn > mx) then
!$OMP ATOMIC
         cu(1,nn+noffp,j+moffp) = cu(1,nn+noffp,j+moffp) + scu(1,nn,j)
!$OMP ATOMIC
         cu(2,nn+noffp,j+moffp) = cu(2,nn+noffp,j+moffp) + scu(2,nn,j)
!$OMP ATOMIC
         cu(3,nn+noffp,j+moffp) = cu(3,nn+noffp,j+moffp) + scu(3,nn,j)
      endif
   80 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
   90 continue
!$OMP END PARALLEL DO
      return
      end
```