```
c-----------------------------------------------------------------------
      subroutine PPGRBPUSH23L(part,fxy,bxy,edges,npp,noff,ihole,qbm,dt,
     1dtc,ci,ek,nx,ny,idimp,npmax,nxv,nypmx,idps,ntmax,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 scalar version using guard cells, for distributed data
c also determines list of particles which are leaving this processor
c 129 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)), 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(1,n) = position x of particle n in partition
c part(2,n) = position y of particle n in partition
c part(3,n) = momentum px of particle n in partition
```

```
c part(4,n) = momentum py of particle n in partition
c part(5,n) = momentum pz of particle n in partition
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 edges(1:2) = lower:upper boundary of particle partition
c npp = number of particles in partition
c noff = lowermost global gridpoint in particle partition.
c ihole = location of hole left in particle arrays
c ihole(1) = 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 = 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 nx/ny = system length in x/y direction
c idimp = size of phase space = 5
c npmax = maximum number of particles in each partition
c nxv = first dimension of field arrays, must be >= nx+1
c nypmx = maximum size of particle partition, including guard cells.
c idps = number of partition boundaries
c ntmax = size of hole array for particles leaving processors
c ipbc = particle boundary condition = (0,1,2,3) =
c (none,2d periodic,2d reflecting,mixed reflecting/periodic)
      implicit none
      integer npp, noff, nx, ny, idimp, npmax, idps, ntmax, nxv, nypmx
      integer ipbc
      real qbm, dt, dtc, ci, ek
      real part, fxy, bxy, edges
      integer ihole
      dimension part(idimp,npmax), fxy(3,nxv,nypmx), bxy(3,nxv,nypmx)
      dimension edges(idps), ihole(ntmax+1)
c local data
      integer mnoff, j, nn, mm, np, mp, ih, nh
      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
      double precision sum1
      qtmh = .5*qbm*dt
      ci2 = ci*ci
      sum1 = 0.0d0
c set boundary values
      edgelx = 0.0
      edgely = 1.0
```

```
      edgerx = real(nx)
      edgery = real(ny-1)
      if ((ipbc.eq.2).or.(ipbc.eq.3)) then
         edgelx = 1.0
         edgerx = real(nx-1)
      endif
      mnoff = noff - 1
      ih = 0
      nh = 0
      do 10 j = 1, npp
c find interpolation weights
      nn = part(1,j)
      mm = part(2,j)
      dxp = part(1,j) - real(nn)
      dyp = part(2,j) - real(mm)
      nn = nn + 1
      mm = mm - mnoff
      amx = 1.0 - dxp
      mp = mm + 1
      amy = 1.0 - dyp
      np = nn + 1
c find electric field
      dx = dyp*(dxp*fxy(1,np,mp) + amx*fxy(1,nn,mp))
     1   + amy*(dxp*fxy(1,np,mm) + amx*fxy(1,nn,mm))
      dy = dyp*(dxp*fxy(2,np,mp) + amx*fxy(2,nn,mp))
     1   + amy*(dxp*fxy(2,np,mm) + amx*fxy(2,nn,mm))
      dz = dyp*(dxp*fxy(3,np,mp) + amx*fxy(3,nn,mp))
     1   + amy*(dxp*fxy(3,np,mm) + amx*fxy(3,nn,mm))
c calculate half impulse
      dx = qtmh*dx
      dy = qtmh*dy
      dz = qtmh*dz
c half acceleration
      acx = part(3,j) + dx
      acy = part(4,j) + dy
      acz = part(5,j) + dz
c find inverse gamma
      p2 = acx*acx + acy*acy + acz*acz
      gami = 1.0/sqrt(1.0 + p2*ci2)
c find magnetic field
      ox = dyp*(dxp*bxy(1,np,mp) + amx*bxy(1,nn,mp))
     1   + amy*(dxp*bxy(1,np,mm) + amx*bxy(1,nn,mm))
      oy = dyp*(dxp*bxy(2,np,mp) + amx*bxy(2,nn,mp))
     1   + amy*(dxp*bxy(2,np,mm) + amx*bxy(2,nn,mm))
      oz = dyp*(dxp*bxy(3,np,mp) + amx*bxy(3,nn,mp))
     1   + amy*(dxp*bxy(3,np,mm) + amx*bxy(3,nn,mm))
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. + omt)
      omt = 0.5*(1. - 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
      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
      part(3,j) = dx
      part(4,j) = dy
      part(5,j) = dz
c update inverse gamma
      p2 = dx*dx + dy*dy + dz*dz
      dtg = dtc/sqrt(1.0 + p2*ci2)
c new position
      dx = part(1,j) + dx*dtg
      dy = part(2,j) + dy*dtg
c periodic boundary conditions in x
      if (ipbc.eq.1) then
         if (dx.lt.edgelx) dx = dx + edgerx
         if (dx.ge.edgerx) dx = dx - edgerx
c reflecting boundary conditions
      else if (ipbc.eq.2) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = part(1,j)
            part(3,j) = -part(3,j)
         endif
         if ((dy.lt.edgely).or.(dy.ge.edgery)) then
            dy = part(2,j)
            part(4,j) = -part(4,j)
         endif
c mixed reflecting/periodic boundary conditions
      else if (ipbc.eq.3) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = part(1,j)
            part(3,j) = -part(3,j)
         endif
      endif
c find particles out of bounds
      if ((dy.lt.edges(1)).or.(dy.ge.edges(2))) then
         ih = ih + 1
         if (ih.le.ntmax) then
```

```
            ihole(ih+1) = j
         else
            nh = 1
         endif
      endif
c set new position
      part(1,j) = dx
      part(2,j) = dy
   10 continue
c set end of file flag
      if (nh.gt.0) ih = -ih
      ihole(1) = ih
c normalize kinetic energy
      ek = ek + sum1
      return
      end
```

```
c-----------------------------------------------------------------------
      subroutine PPGPOST2L(part,q,npp,noff,qm,idimp,npmax,nxv,nypmx)
c for 2d code, this subroutine calculates particle charge density
c using first-order linear interpolation, periodic boundaries
c and distributed data.
c scalar version using guard cells, for distributed data
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(1,n) = position x of particle n in partition
c part(2,n) = position y of particle n in partition
c q(j,k) = charge density at grid point (j,kk),
c where kk = k + noff - 1
c npp = number of particles in partition
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 npmax = maximum number of particles in each partition
c nxv = first dimension of charge array, must be >= nx+1
c nypmx = maximum size of particle partition, including guard cells.
      implicit none
      integer npp, noff, idimp, npmax, nxv, nypmx
      real qm
      real part, q
      dimension part(idimp,npmax), q(nxv,nypmx)
c local data
      integer mnoff, j, nn, np, mm, mp
      real dxp, dyp, amx, amy
      mnoff = noff - 1
      do 10 j = 1, npp
c find interpolation weights
      nn = part(1,j)
      mm = part(2,j)
      dxp = qm*(part(1,j) - real(nn))
      dyp = part(2,j) - real(mm)
      nn = nn + 1
      mm = mm - mnoff
      amx = qm - dxp
      mp = mm + 1
      amy = 1.0 - dyp
      np = nn + 1
c deposit charge
      q(np,mp) = q(np,mp) + dxp*dyp
      q(nn,mp) = q(nn,mp) + amx*dyp
      q(np,mm) = q(np,mm) + dxp*amy
      q(nn,mm) = q(nn,mm) + amx*amy
   10 continue
      return
      end
```

```
c-----------------------------------------------------------------------
      subroutine PPGRJPOST2L(part,cu,edges,npp,noff,ihole,qm,dt,ci,nx,ny
     1,idimp,npmax,nxv,nypmx,idps,ntmax,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 also determines list of particles which are leaving this processor
c scalar version using guard cells, for distributed data
c 45 flops/particle, 1 divide, 1 sqrt, 17 loads, 14 stores
c input: all except ihole, output: part, ihole, 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(1,n) = position x of particle n in partition
c part(2,n) = position y of particle n in partition
c part(3,n) = x momentum of particle n in partition
c part(4,n) = y momentum of particle n in partition
c part(5,n) = z momentum of particle n in partition
c cu(i,j,k) = ith component of current density at grid point (j,kk),
c where kk = k + noff - 1
c edges(1:2) = lower:upper boundary of particle partition
c npp = number of particles in partition
c noff = lowermost global gridpoint in particle partition.
c ihole = location of hole left in particle arrays
c ihole(1) = 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 = reciprical of velocity of light
c nx/ny = system length in x/y direction
c idimp = size of phase space = 5
c npmax = maximum number of particles in each partition
c nxv = first dimension of current array, must be >= nx+1
c nypmx = maximum size of particle partition, including guard cells.
c idps = number of partition boundaries
c ntmax =  size of hole array for particles leaving processors
c ipbc = particle boundary condition = (0,1,2,3) =
c (none,2d periodic,2d reflecting,mixed reflecting/periodic)
      implicit none
      integer npp, noff, nx, ny, idimp, npmax, idps, ntmax, nxv, nypmx
      integer ipbc
      real qm, dt, ci
      real part, cu, edges
      integer ihole
      dimension part(idimp,npmax), cu(3,nxv,nypmx)
      dimension edges(idps), ihole(ntmax+1)
c local data
      integer mnoff, j, nn, mm, np, mp, ih, nh
      real ci2, edgelx, edgely, edgerx, edgery, dxp, dyp, amx, amy
      real dx, dy, vx, vy, vz, p2, gami
```

```
      ci2 = ci*ci
c set boundary values
      edgelx = 0.0
      edgely = 1.0
      edgerx = real(nx)
      edgery = real(ny-1)
      if ((ipbc.eq.2).or.(ipbc.eq.3)) then
         edgelx = 1.0
         edgerx = real(nx-1)
      endif
      mnoff = noff - 1
      ih = 0
      nh = 0
      do 10 j = 1, npp
c find interpolation weights
      nn = part(1,j)
      mm = part(2,j)
      dxp = qm*(part(1,j) - real(nn))
      dyp = part(2,j) - real(mm)
c find inverse gamma
      vx = part(3,j)
      vy = part(4,j)
      vz = part(5,j)
      p2 = vx*vx + vy*vy + vz*vz
      gami = 1.0/sqrt(1.0 + p2*ci2)
c calculate weights
      nn = nn + 1
      mm = mm - mnoff
      amx = qm - dxp
      mp = mm + 1
      amy = 1.0 - dyp
      np = nn + 1
c deposit current
      dx = dxp*dyp
      dy = amx*dyp
      vx = vx*gami
      vy = vy*gami
      vz = vz*gami
      cu(1,np,mp) = cu(1,np,mp) + vx*dx
      cu(2,np,mp) = cu(2,np,mp) + vy*dx
      cu(3,np,mp) = cu(3,np,mp) + vz*dx
      dx = dxp*amy
      cu(1,nn,mp) = cu(1,nn,mp) + vx*dy
      cu(2,nn,mp) = cu(2,nn,mp) + vy*dy
      cu(3,nn,mp) = cu(3,nn,mp) + vz*dy
      dy = amx*amy
      cu(1,np,mm) = cu(1,np,mm) + vx*dx
      cu(2,np,mm) = cu(2,np,mm) + vy*dx
      cu(3,np,mm) = cu(3,np,mm) + vz*dx
      cu(1,nn,mm) = cu(1,nn,mm) + vx*dy
      cu(2,nn,mm) = cu(2,nn,mm) + vy*dy
      cu(3,nn,mm) = cu(3,nn,mm) + vz*dy
c advance position half a time-step
      dx = part(1,j) + vx*dt
```

```
      dy = part(2,j) + vy*dt
c periodic boundary conditions in x
      if (ipbc.eq.1) then
         if (dx.lt.edgelx) dx = dx + edgerx
         if (dx.ge.edgerx) dx = dx - edgerx
c reflecting boundary conditions
      else if (ipbc.eq.2) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = part(1,j)
            part(3,j) = -part(3,j)
         endif
         if ((dy.lt.edgely).or.(dy.ge.edgery)) then
            dy = part(2,j)
            part(4,j) = -part(4,j)
         endif
c mixed reflecting/periodic boundary conditions
      else if (ipbc.eq.3) then
         if ((dx.lt.edgelx).or.(dx.ge.edgerx)) then
            dx = part(1,j)
            part(3,j) = -part(3,j)
         endif
      endif
c find particles out of bounds
      if ((dy.lt.edges(1)).or.(dy.ge.edges(2))) then
         ih = ih + 1
         if (ih.le.ntmax) then
            ihole(ih+1) = j
         else
            nh = 1
         endif
      endif
c set new position
      part(1,j) = dx
      part(2,j) = dy
   10 continue
c set end of file flag
      if (nh.gt.0) ih = -ih
      ihole(1) = ih
      return
      end
```