```fortran
      subroutine GRJPOST2L(part,cu,qm,dt,ci,nop,idimp,nx,ny,nxv,nyv,ipbc
     1)
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 scalar 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(1,n) = position x of particle n
c part(2,n) = position y of particle n
c part(3,n) = x momentum of particle n
c part(4,n) = y momentum of particle n
c part(5,n) = 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 idimp = size of phase space = 5
c nx/ny = system length in x/y direction
c nxv = first dimension of current array, must be >= nx+1
c nyv = second 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)
      dimension part(idimp,nop), cu(3,nxv,nyv)
      ci2 = ci*ci
c set boundary values
      if (ipbc.eq.1) then
         edgelx = 0.
         edgely = 0.
         edgerx = float(nx)
         edgery = float(ny)
      else if (ipbc.eq.2) then
         edgelx = 1.
         edgely = 1.
         edgerx = float(nx-1)
         edgery = float(ny-1)
      else if (ipbc.eq.3) then
         edgelx = 1.
         edgely = 0.
         edgerx = float(nx-1)
         edgery = float(ny)
      endif
      do 10 j = 1, nop
c find interpolation weights
      nn = part(1,j)
      mm = part(2,j)
      dxp = qm*(part(1,j) - float(nn))
      dyp = part(2,j) - float(mm)
c find inverse gamma
      vx = part(3,j)
      vy = part(4,j)
```

```fortran
      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 + 1
      amx = qm - dxp
      mp = mm + 1
      amy = 1. - 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
      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 = 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
         if (dy.lt.edgely) dy = dy + edgery
         if (dy.ge.edgery) dy = dy - edgery
      endif
c set new position
      part(1,j) = dx
```

```
      part(2,j) = dy
   10 continue
      return
      end
```