```
      subroutine GRBPUSH22L(part,fxy,bz,qbm,dt,dtc,ci,ek,idimp,nop,nx,ny
     1,nxv,nyv,ipbc)
c for 2d code, this subroutine updates particle co-ordinates and
c velocities using leap-frog scheme in time and second-order spline
c interpolation in space, for relativistic particles with magnetic field
c in z direction, using the Boris Mover.
c scalar version using guard cells
c 74 flops/particle, 4 divides, 2 sqrts, 16 loads, 4 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    .5*(q/m)*fx(x(t),y(t))*dt)
c py(t+dt/2) = -rot(2)*(px(t-dt/2) + .5*(q/m)*fx(x(t),y(t))*dt) +
c    rot(1)*(py(t-dt/2) + .5*(q/m)*fy(x(t),y(t))*dt) +
c    .5*(q/m)*fy(x(t),y(t))*dt)
c where q/m is charge/mass, and the rotation matrix is given by:
c    rot(1) = (1 - (omz*dt/2)**2/(1 + (omz*dt/2)**2)
c    rot(2) = 2*(omz*dt/2)/(1 + (omz*dt/2)**2)
c where omz = (q/m)*bz(x(t),y(t))*gami,
c and gami = 1./sqrt(1.+(px(t)*px(t)+py(t)*py(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, where
c dtg = dtc/sqrt(1.+(px(t+dt/2)*px(t+dt/2)+py(t+dt/2)*py(t+dt/2))*ci*ci)
c fx(x(t),y(t)), fy(x(t),y(t)), 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), bz(x,y)
c part(1,n) = position x of particle n
c part(2,n) = position y of particle n
c part(3,n) = momentum px of particle n
c part(4,n) = momentum py 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 that is, convolution of electric field over particle shape
c bz(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)/(1. + gami)
c idimp = size of phase space = 4
c nop = number of particles
c nx/ny = system length in x/y direction
c nxv = first dimension of field arrays, must be >= nx+1
c nyv = second 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)
      double precision sum1
      dimension part(idimp,nop)
      dimension fxy(2,nxv,nyv), bz(nxv,nyv)
      qtmh = .5*qbm*dt
      ci2 = ci*ci
      sum1 = 0.0d0
```

```
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 = part(1,j) - float(nn)
      dyp = part(2,j) - float(mm)
      nn = nn + 1
      mm = mm + 1
      amx = 1. - dxp
      mp = mm + 1
      amy = 1. - dyp
      np = nn + 1
c find electric field
      dx = dyp*(dxp*fxy(1,np,mp) + amx*fxy(1,nn,mp)) + amy*(dxp*fxy(1,np
     1,mm) + amx*fxy(1,nn,mm))
      dy = dyp*(dxp*fxy(2,np,mp) + amx*fxy(2,nn,mp)) + amy*(dxp*fxy(2,np
     1,mm) + amx*fxy(2,nn,mm))
c calculate half impulse
      dx = qtmh*dx
      dy = qtmh*dy
c half acceleration
      acx = part(3,j) + dx
      acy = part(4,j) + dy
c find inverse gamma
      p2 = acx*acx + acy*acy
      gami = 1.0/sqrt(1.0 + p2*ci2)
c find magnetic field
      oz = dyp*(dxp*bz(np,mp) + amx*bz(nn,mp)) + amy*(dxp*bz(np,mm) + am
     1x*bz(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
      omzt = qtmg*oz
c calculate rotation matrix
      omt = omzt*omzt
      anorm = 2./(1. + omt)
      rot1 = .5*(1. - omt)
      rot2 = omzt
c new velocity
      dx = (rot1*acx + rot2*acy)*anorm + dx
      dy = (rot1*acy - rot2*acx)*anorm + dy
```

```
      part(3,j) = dx
      part(4,j) = dy
c update inverse gamma
      p2 = dx*dx + dy*dy
      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
      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
c normalize kinetic energy
      ek = ek + sum1
      return
      end
```