```
/*--------------------------------------------------------------------*/
void cvgrbpush23lt(float part[], float fxy[], float bxy[], float qbm,
                   float dt, float dtc, float ci, float *ek, int idimp,
                   int nop, int npe, int nx, int ny, int nxv, int nyv,
                   int ipbc) {
/* for 2-1/2d code, this subroutine updates particle co-ordinates and
   velocities using leap-frog scheme in time and first-order linear
   interpolation in space, for relativistic particles with magnetic field
   Using the Boris Mover.
   vectorizable version using guard cells
   131 flops/particle, 4 divides, 2 sqrts, 25 loads, 5 stores
   input: all, output: part, ek
   momentum equations used are:
   px(t+dt/2) = rot(1)*(px(t-dt/2) + .5*(q/m)*fx(x(t),y(t))*dt) +
      rot(2)*(py(t-dt/2) + .5*(q/m)*fy(x(t),y(t))*dt) +
      rot(3)*(pz(t-dt/2) + .5*(q/m)*fz(x(t),y(t))*dt) +
      .5*(q/m)*fx(x(t),y(t))*dt)
   py(t+dt/2) = rot(4)*(px(t-dt/2) + .5*(q/m)*fx(x(t),y(t))*dt) +
      rot(5)*(py(t-dt/2) + .5*(q/m)*fy(x(t),y(t))*dt) +
      rot(6)*(pz(t-dt/2) + .5*(q/m)*fz(x(t),y(t))*dt) +
      .5*(q/m)*fy(x(t),y(t))*dt)
   pz(t+dt/2) = rot(7)*(px(t-dt/2) + .5*(q/m)*fx(x(t),y(t))*dt) +
      rot(8)*(py(t-dt/2) + .5*(q/m)*fy(x(t),y(t))*dt) +
      rot(9)*(pz(t-dt/2) + .5*(q/m)*fz(x(t),y(t))*dt) +
      .5*(q/m)*fz(x(t),y(t))*dt)
   where q/m is charge/mass, and the rotation matrix is given by:
      rot[0] = (1 - (om*dt/2)**2 + 2*(omx*dt/2)**2)/(1 + (om*dt/2)**2)
      rot[1] = 2*(omz*dt/2 + (omx*dt/2)*(omy*dt/2))/(1 + (om*dt/2)**2)
      rot[2] = 2*(-omy*dt/2 + (omx*dt/2)*(omz*dt/2))/(1 + (om*dt/2)**2)
      rot[3] = 2*(-omz*dt/2 + (omx*dt/2)*(omy*dt/2))/(1 + (om*dt/2)**2)
      rot[4] = (1 - (om*dt/2)**2 + 2*(omy*dt/2)**2)/(1 + (om*dt/2)**2)
      rot[5] = 2*(omx*dt/2 + (omy*dt/2)*(omz*dt/2))/(1 + (om*dt/2)**2)
      rot[6] = 2*(omy*dt/2 + (omx*dt/2)*(omz*dt/2))/(1 + (om*dt/2)**2)
      rot[7] = 2*(-omx*dt/2 + (omy*dt/2)*(omz*dt/2))/(1 + (om*dt/2)**2)
      rot[8] = (1 - (om*dt/2)**2 + 2*(omz*dt/2)**2)/(1 + (om*dt/2)**2)
   and om**2 = omx**2 + omy**2 + omz**2
   the rotation matrix is determined by:
   omx = (q/m)*bx(x(t),y(t))*gami, omy = (q/m)*by(x(t),y(t))*gami, and
   omz = (q/m)*bz(x(t),y(t))*gami,
   where gami = 1./sqrt(1.+(px(t)*px(t)+py(t)*py(t)+pz(t)*pz(t))*ci*ci)
   position equations used are:
   x(t+dt) = x(t) + px(t+dt/2)*dtg
   y(t+dt) = y(t) + py(t+dt/2)*dtg
   where dtg = dtc/sqrt(1.+(px(t+dt/2)*px(t+dt/2)+py(t+dt/2)*py(t+dt/2)+
   pz(t+dt/2)*pz(t+dt/2))*ci*ci)
   fx(x(t),y(t)), fy(x(t),y(t)), and fz(x(t),y(t))
   bx(x(t),y(t)), by(x(t),y(t)), and bz(x(t),y(t))
   are approximated by interpolation from the nearest grid points:
   fx(x,y) = (1-dy)*((1-dx)*fx(n,m)+dx*fx(n+1,m)) + dy*((1-dx)*fx(n,m+1)
      + dx*fx(n+1,m+1))
   where n,m = leftmost grid points and dx = x-n, dy = y-m
   similarly for fy(x,y), fz(x,y), bx(x,y), by(x,y), bz(x,y)
   part[0][n] = position x of particle n
   part[1][n] = position y of particle n
```

```
   part[2][n] = momentum px of particle n
   part[3][n] = momentum py of particle n
   part[4][n] = momentum pz of particle n
   fxy[k][j][0] = x component of force/charge at grid (j,k)
   fxy[k][j][1] = y component of force/charge at grid (j,k)
   fxy[k][j][2] = z component of force/charge at grid (j,k)
   that is, convolution of electric field over particle shape
   bxy[k][j][0] = x component of magnetic field at grid (j,k)
   bxy[k][j][1] = y component of magnetic field at grid (j,k)
   bxy[k][j][2] = z component of magnetic field at grid (j,k)
   that is, the convolution of magnetic field over particle shape
   qbm = particle charge/mass ratio
   dt = time interval between successive calculations
   dtc = time interval between successive co-ordinate calculations
   ci = reciprocal of velocity of light
   kinetic energy/mass at time t is also calculated, using
   ek = gami*sum((px(t-dt/2) + .5*(q/m)*fx(x(t),y(t))*dt)**2 +
        (py(t-dt/2) + .5*(q/m)*fy(x(t),y(t))*dt)**2 +
        (pz(t-dt/2) + .5*(q/m)*fz(x(t),y(t))*dt)**2)/(1. + gami)
   idimp = size of phase space = 5
   nop = number of particles
   npe = first dimension of particle array
   nx/ny = system length in x/y direction
   nxv = second dimension of field arrays, must be >= nx+1
   nyv = third dimension of field arrays, must be >= ny+1
   ipbc = particle boundary condition = (0,1,2,3) =
   (none,2d periodic,2d reflecting,mixed reflecting/periodic)
local data                                                      */
#define NPBLK             32
#define LVECT             4
#define N                 4
   int i, j, k, ipp, joff, nps, nn, mm, nm;
   float qtmh, ci2, edgelx, edgely, edgerx, edgery, dxp, dyp, amx, amy;
   float dx, dy, dz, ox, oy, oz, acx, acy, acz, p2, gami, qtmg, dtg;
   float omxt, omyt, omzt, omt, anorm;
   float rot1, rot2, rot3, rot4, rot5, rot6, rot7, rot8, rot9;
   float x, y, vx, vy, vz;
/* scratch arrays */
   int n[NPBLK];
   float s1[NPBLK*LVECT], s2[NPBLK*LVECT], t[NPBLK*2];
   double sum1;
   qtmh = 0.5f*qbm*dt;
   ci2 = ci*ci;
   sum1 = 0.0;
/* set boundary values */
   edgelx = 0.0f;
   edgely = 0.0f;
   edgerx = (float) nx;
   edgery = (float) ny;
   if (ipbc==2) {
      edgelx = 1.0f;
      edgely = 1.0f;
      edgerx = (float) (nx-1);
      edgery = (float) (ny-1);
```

```
   }
   else if (ipbc==3) {
      edgelx = 1.0f;
      edgerx = (float) (nx-1);
   }
   ipp = nop/NPBLK;
/* outer loop over number of full blocks */
   for (k = 0; k < ipp; k++) {
      joff = NPBLK*k;
/* inner loop over particles in block */
      for (j = 0; j < NPBLK; j++) {
/* find interpolation weights */
         x = part[j+joff];
         y = part[j+joff+npe];
         nn = x;
         mm = y;
         dxp = x - (float) nn;
         dyp = y - (float) mm;
         n[j] = N*(nn + nxv*mm);
         amx = 1.0f - dxp;
         amy = 1.0f - dyp;
         s1[j] = amx*amy;
         s1[j+NPBLK] = dxp*amy;
         s1[j+2*NPBLK] = amx*dyp;
         s1[j+3*NPBLK] = dxp*dyp;
         t[j] = x;
         t[j+NPBLK] = y;
      }
/* find acceleration */
      for (j = 0; j < NPBLK; j++) {
         nn = n[j];
         mm = nn + N*(nxv - 2);
         dx = 0.0f;
         dy = 0.0f;
         dz = 0.0f;
         ox = 0.0f;
         oy = 0.0f;
         oz = 0.0f;
#pragma ivdep
         for (i = 0; i < LVECT; i++) {
            if (i > 1)
               nn = mm;
            dx += fxy[4*i+nn]*s1[j+NPBLK*i];
            dy += fxy[1+4*i+nn]*s1[j+NPBLK*i];
            dz += fxy[2+4*i+nn]*s1[j+NPBLK*i];
            ox += bxy[4*i+nn]*s1[j+NPBLK*i];
            oy += bxy[1+4*i+nn]*s1[j+NPBLK*i];
            oz += bxy[2+4*i+nn]*s1[j+NPBLK*i];
         }
         s1[j] = dx;
         s1[j+NPBLK] = dy;
         s1[j+2*NPBLK] = dz;
         s2[j] = ox;
         s2[j+NPBLK] = oy;
```

```
         s2[j+2*NPBLK] = oz;
      }
/* new momentum */
      for (j = 0; j < NPBLK; j++) {
         x = t[j];
         y = t[j+NPBLK];
/* calculate half impulse */
         dx = qtmh*s1[j];
         dy = qtmh*s1[j+NPBLK];
         dz = qtmh*s1[j+2*NPBLK];
/* half acceleration */
         acx = part[j+joff+2*npe] + dx;
         acy = part[j+joff+3*npe] + dy;
         acz = part[j+joff+4*npe] + dz;
/* find inverse gamma */
         p2 = acx*acx + acy*acy + acz*acz;
         gami = 1.0f/sqrtf(1.0f + p2*ci2);
/* renormalize magnetic field */
         qtmg = qtmh*gami;
/* time-centered kinetic energy */
         sum1 += gami*p2/(1.0f + gami);
/* calculate cyclotron frequency */
         omxt = qtmg*s2[j];
         omyt = qtmg*s2[j+NPBLK];
         omzt = qtmg*s2[j+2*NPBLK];
/* calculate rotation matrix */
         omt = omxt*omxt + omyt*omyt + omzt*omzt;
         anorm = 2.0f/(1.0f + omt);
         omt = 0.5f*(1.0f - 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;
         rot3 = -omyt + rot7;
         rot7 += omyt;
         rot6 = omxt + rot8;
         rot8 -= omxt;
/* new momentum */
         vx = dx + (rot1*acx + rot2*acy + rot3*acz)*anorm;
         vy = dy + (rot4*acx + rot5*acy + rot6*acz)*anorm;
         vz = dz + (rot7*acx + rot8*acy + rot9*acz)*anorm;
/* update inverse gamma */
         p2 = vx*vx + vy*vy + vz*vz;
         dtg = dtc/sqrtf(1.0f + p2*ci2);
/* new position */
         s1[j] = x + vx*dtg;
         s1[j+NPBLK] = y + vy*dtg;
         s2[j] = vx;
         s2[j+NPBLK] = vy;
         s2[j+2*NPBLK] = vz;
```

```
      }
/* check boundary conditions */
      for (j = 0; j < NPBLK; j++) {
         dx = s1[j];
         dy = s1[j+NPBLK];
         vx = s2[j];
         vy = s2[j+NPBLK];
         vz = s2[j+2*NPBLK];
/* periodic boundary conditions */
         if (ipbc==1) {
            if (dx < edgelx) dx += edgerx;
            if (dx >= edgerx) dx -= edgerx;
            if (dy < edgely) dy += edgery;
            if (dy >= edgery) dy -= edgery;
         }
/* reflecting boundary conditions */
         else if (ipbc==2) {
            if ((dx < edgelx) || (dx >= edgerx)) {
               dx = t[j];
               vx = -vx;
            }
            if ((dy < edgely) || (dy >= edgery)) {
               dy = t[j+NPBLK];
               vy = -vy;
            }
         }
/* mixed reflecting/periodic boundary conditions */
         else if (ipbc==3) {
            if ((dx < edgelx) || (dx >= edgerx)) {
               dx = t[j];
               vx = -vx;
            }
            if (dy < edgely) dy += edgery;
            if (dy >= edgery) dy -= edgery;
         }
/* set new position */
         part[j+joff] = dx;
         part[j+joff+npe] = dy;
/* set new velocity */
         part[j+joff+2*npe] = vx;
         part[j+joff+3*npe] = vy;
         part[j+joff+4*npe] = vz;
      }
   }
   nps = NPBLK*ipp;
/* loop over remaining particles */
   for (j = nps; j < nop; j++) {
/* find interpolation weights */
      x = part[j];
      y = part[j+npe];
      nn = x;
      mm = y;
      dxp = x - (float) nn;
      dyp = y - (float) mm;
```

```
      nm = N*(nn + nxv*mm);
      amx = 1.0f - dxp;
      amy = 1.0f - dyp;
/* find electric field */
      nn = nm;
      dx = amx*fxy[nn];
      dy = amx*fxy[nn+1];
      dz = amx*fxy[nn+2];
      mm = nn + N;
      dx = amy*(dxp*fxy[mm] + dx);
      dy = amy*(dxp*fxy[mm+1] + dy);
      dz = amy*(dxp*fxy[mm+2] + dz);
      nn += N*nxv;
      acx = amx*fxy[nn];
      acy = amx*fxy[nn+1];
      acz = amx*fxy[nn+2];
      mm = nn + N;
      dx += dyp*(dxp*fxy[mm] + acx);
      dy += dyp*(dxp*fxy[mm+1] + acy);
      dz += dyp*(dxp*fxy[mm+2] + acz);
/* find magnetic field */
      nn = nm;
      ox = amx*bxy[nn];
      oy = amx*bxy[nn+1];
      oz = amx*bxy[nn+2];
      mm = nn + N;
      ox = amy*(dxp*bxy[mm] + ox);
      oy = amy*(dxp*bxy[mm+1] + oy);
      oz = amy*(dxp*bxy[mm+2] + oz);
      nn += N*nxv;
      acx = amx*bxy[nn];
      acy = amx*bxy[nn+1];
      acz = amx*bxy[nn+2];
      mm = nn + N;
      ox += dyp*(dxp*bxy[mm] + acx);
      oy += dyp*(dxp*bxy[mm+1] + acy);
      oz += dyp*(dxp*bxy[mm+2] + acz);
/* calculate half impulse */
      dx *= qtmh;
      dy *= qtmh;
      dz *= qtmh;
/* half acceleration */
      acx = part[j+2*npe] + dx;
      acy = part[j+3*npe] + dy;
      acz = part[j+4*npe] + dz;
/* find inverse gamma */
      p2 = acx*acx + acy*acy + acz*acz;
      gami = 1.0f/sqrtf(1.0f + p2*ci2);
/* renormalize magnetic field */
      qtmg = qtmh*gami;
/* time-centered kinetic energy */
      sum1 += gami*p2/(1.0f + gami);
/* calculate cyclotron frequency */
      omxt = qtmg*ox;
```

```
      omyt = qtmg*oy;
      omzt = qtmg*oz;
/* calculate rotation matrix */
      omt = omxt*omxt + omyt*omyt + omzt*omzt;
      anorm = 2.0f/(1.0f + omt);
      omt = 0.5f*(1.0f - 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;
      rot3 = -omyt + rot7;
      rot7 += omyt;
      rot6 = omxt + rot8;
      rot8 -= omxt;
/* new momentum */
      vx = dx + (rot1*acx + rot2*acy + rot3*acz)*anorm;
      vy = dy + (rot4*acx + rot5*acy + rot6*acz)*anorm;
      vz = dz + (rot7*acx + rot8*acy + rot9*acz)*anorm;
/* update inverse gamma */
      p2 = vx*vx + vy*vy + vz*vz;
      dtg = dtc/sqrtf(1.0f + p2*ci2);
/* new position */
      dx = x + vx*dtg;
      dy = y + vy*dtg;
/* periodic boundary conditions */
      if (ipbc==1) {
         if (dx < edgelx) dx += edgerx;
         if (dx >= edgerx) dx -= edgerx;
         if (dy < edgely) dy += edgery;
         if (dy >= edgery) dy -= edgery;
      }
/* reflecting boundary conditions */
      else if (ipbc==2) {
         if ((dx < edgelx) || (dx >= edgerx)) {
            dx = x;
            vx = -vx;
         }
         if ((dy < edgely) || (dy >= edgery)) {
            dy = y;
            vy = -vy;
         }
      }
/* mixed reflecting/periodic boundary conditions */
      else if (ipbc==3) {
         if ((dx < edgelx) || (dx >= edgerx)) {
            dx = x;
            vx = -vx;
         }
         if (dy < edgely) dy += edgery;
         if (dy >= edgery) dy -= edgery;
```

```
      }
/* set new position */
      part[j] = dx;
      part[j+npe] = dy;
/* set new velocity */
      part[j+2*npe] = vx;
      part[j+3*npe] = vy;
      part[j+4*npe] = vz;
   }
/* normalize kinetic energy */
   *ek += sum1;
   return;
#undef LVECT
#undef NPBLK
#undef N
}
```

```
/*--------------------------------------------------------------------*/
void cvgpost2lt(float part[], float q[], float qm, int nop, int npe,
                int idimp, int nxv, int nyv) {
/* for 2d code, this subroutine calculates particle charge density
   using first-order linear interpolation, periodic boundaries
   vectorizable version using guard cells
   17 flops/particle, 6 loads, 4 stores
   input: all, output: q
   charge density is approximated by values at the nearest grid points
   q(n,m)=qm*(1.-dx)*(1.-dy)
   q(n+1,m)=qm*dx*(1.-dy)
   q(n,m+1)=qm*(1.-dx)*dy
   q(n+1,m+1)=qm*dx*dy
   where n,m = leftmost grid points and dx = x-n, dy = y-m
   part[0][n] = position x of particle n
   part[1][n] = position y of particle n
   q[k][j] = charge density at grid point j,k
   qm = charge on particle, in units of e
   nop = number of particles
   npe = first dimension of particle array
   idimp = size of phase space = 4
   nxv = first dimension of charge array, must be >= nx+1
   nyv = second dimension of charge array, must be >= ny+1
local data                                                          */
#define NPBLK             32
#define LVECT             4
   int i, j, k, ipp, joff, nps, nn, mm;
   float x, y, dxp, dyp, amx, amy;
/* scratch arrays */
   int n[NPBLK];
   float s[NPBLK*LVECT];
   ipp = nop/NPBLK;
/* outer loop over number of full blocks */
   for (k = 0; k < ipp; k++) {
      joff = NPBLK*k;
/* inner loop over particles in block */
      for (j = 0; j < NPBLK; j++) {
/* find interpolation weights */
         x = part[j+joff];
         y = part[j+joff+npe];
         nn = x;
         mm = y;
         dxp = qm*(x - (float) nn);
         dyp = y - (float) mm;
         n[j] = nn + nxv*mm;
         amx = qm - dxp;
         amy = 1.0f - dyp;
         s[j] = amx*amy;
         s[j+NPBLK] = dxp*amy;
         s[j+2*NPBLK] = amx*dyp;
         s[j+3*NPBLK] = dxp*dyp;
     }
/* deposit charge */
      for (j = 0; j < NPBLK; j++) {
```

```
         nn = n[j];
         mm = nn + nxv - 2;
#pragma ivdep
         for (i = 0; i < LVECT; i++) {
            if (i > 1)
               nn = mm;
            q[i+nn] += s[j+NPBLK*i];
         }
      }
   }
   nps = NPBLK*ipp;
/* loop over remaining particles */
   for (j = nps; j < nop; j++) {
/* find interpolation weights */
      x = part[j];
      y = part[j+npe];
      nn = x;
      mm = y;
      dxp = qm*(x - (float) nn);
      dyp = y - (float) mm;
      nn = nn + nxv*mm;
      amx = qm - dxp;
      amy = 1.0f - dyp;
/* deposit charge */
      x = q[nn] + amx*amy;
      y = q[nn+1] + dxp*amy;
      q[nn] = x;
      q[nn+1] = y;
      nn += nxv;
      x = q[nn] + amx*dyp;
      y = q[nn+1] + dxp*dyp;
      q[nn] = x;
      q[nn+1] = y;
   }
   return;
#undef LVECT
#undef NPBLK
}
```

```
/*--------------------------------------------------------------------*/
void cvgrjpost2lt(float part[], float cu[], float qm, float dt,
                  float ci, int nop, int npe, int idimp, int nx, int ny,
                  int nxv, int nyv, int ipbc) {
/* for 2-1/2d code, this subroutine calculates particle current density
   using first-order linear interpolation for relativistic particles
   in addition, particle positions are advanced a half time-step
   vectorizable version using guard cells
   47 flops/particle, 1 divide, 1 sqrt, 17 loads, 14 stores
   input: all, output: part, cu
   current density is approximated by values at the nearest grid points
   cu(i,n,m)=qci*(1.-dx)*(1.-dy)
   cu(i,n+1,m)=qci*dx*(1.-dy)
   cu(i,n,m+1)=qci*(1.-dx)*dy
   cu(i,n+1,m+1)=qci*dx*dy
   where n,m = leftmost grid points and dx = x-n, dy = y-m
   and qci = qm*pi*gami, where i = x,y,z
   where gami = 1./sqrt(1.+sum(pi**2)*ci*ci)
   part[0][n] = position x of particle n
   part[1][n] = position y of particle n
   part[2][n] = x momentum of particle n
   part[3][n] = y momentum of particle n
   part[4][n] = z momentum of particle n
   cu[k][j][i] = ith component of current density at grid point j,k
   qm = charge on particle, in units of e
   dt = time interval between successive calculations
   ci = reciprocal of velocity of light
   nop = number of particles
   npe = first dimension of particle array
   idimp = size of phase space = 5
   nx/ny = system length in x/y direction
   nxv = second dimension of current array, must be >= nx+1
   nyv = third dimension of current array, must be >= ny+1
   ipbc = particle boundary condition = (0,1,2,3) =
   (none,2d periodic,2d reflecting,mixed reflecting/periodic)
local data                                                          */
#define NPBLK             32
#define LVECT             4
#define N                 4
   int i, j, k, ipp, joff, nps, nn, mm;
   float ci2, edgelx, edgely, edgerx, edgery, dxp, dyp, amx, amy;
   float x, y, dx, dy, vx, vy, vz, ux, uy, uz, p2, gami;
/* scratch arrays */
   int n[NPBLK];
   float s1[NPBLK*LVECT], s2[NPBLK*LVECT], t[NPBLK*4];
   ci2 = ci*ci;
/* set boundary values */
   edgelx = 0.0;
   edgely = 0.0;
   edgerx = (float) nx;
   edgery = (float) ny;
   if (ipbc==2) {
      edgelx = 1.0;
      edgely = 1.0;
```

```
      edgerx = (float) (nx-1);
      edgery = (float) (ny-1);
   }
   else if (ipbc==3) {
      edgelx = 1.0;
      edgerx = (float) (nx-1);
   }
   ipp = nop/NPBLK;
/* outer loop over number of full blocks */
   for (k = 0; k < ipp; k++) {
      joff = NPBLK*k;
/* inner loop over particles in block */
      for (j = 0; j < NPBLK; j++) {
/* find interpolation weights */
         x = part[j+joff];
         y = part[j+joff+npe];
         nn = x;
         mm = y;
         dxp = qm*(x - (float) nn);
         dyp = y - (float) mm;
         n[j] = N*(nn + nxv*mm);
         amx = qm - dxp;
         amy = 1.0f - dyp;
         s1[j] = amx*amy;
         s1[j+NPBLK] = dxp*amy;
         s1[j+2*NPBLK] = amx*dyp;
         s1[j+3*NPBLK] = dxp*dyp;
         t[j] = x;
         t[j+NPBLK] = y;
/* find inverse gamma */
         ux = part[j+joff+2*npe];
         uy = part[j+joff+3*npe];
         uz = part[j+joff+4*npe];
         p2 = ux*ux + uy*uy + uz*uz;
         gami = 1.0f/sqrtf(1.0f + p2*ci2);
         s2[j] = ux*gami;
         s2[j+NPBLK] = uy*gami;
         s2[j+2*NPBLK] = uz*gami;
         t[j+2*NPBLK] = ux;
         t[j+3*NPBLK] = uy;
      }
/* deposit current */
      for (j = 0; j < NPBLK; j++) {
         nn = n[j];
         mm = nn + N*(nxv - 2);
         vx = s2[j];
         vy = s2[j+NPBLK];
         vz = s2[j+2*NPBLK];
#pragma ivdep
         for (i = 0; i < LVECT; i++) {
            if (i > 1)
               nn = mm;
            cu[4*i+nn] += vx*s1[j+NPBLK*i];
            cu[1+4*i+nn] += vy*s1[j+NPBLK*i];
```

```
            cu[2+4*i+nn] += vz*s1[j+NPBLK*i];
         }
      }
/* advance position half a time-step */
      for (j = 0; j < NPBLK; j++) {
         x = t[j];
         y = t[j+NPBLK];
         vx = s2[j];
         vy = s2[j+NPBLK];
         ux = t[j+2*NPBLK];
         uy = t[j+3*NPBLK];
         dx = x + vx*dt;
         dy = y + vy*dt;
/* periodic boundary conditions */
         if (ipbc==1) {
            if (dx < edgelx) dx += edgerx;
            if (dx >= edgerx) dx -= edgerx;
            if (dy < edgely) dy += edgery;
            if (dy >= edgery) dy -= edgery;
         }
/* reflecting boundary conditions */
         else if (ipbc==2) {
            if ((dx < edgelx) || (dx >= edgerx)) {
               dx = x;
               part[j+joff+2*npe] = -ux;
            }
            if ((dy < edgely) || (dy >= edgery)) {
               dy = y;
               part[j+joff+3*npe] = -uy;
            }
         }
/* mixed reflecting/periodic boundary conditions */
         else if (ipbc==3) {
            if ((dx < edgelx) || (dx >= edgerx)) {
               dx = x;
               part[j+joff+2*npe] = -ux;
            }
            if (dy < edgely) dy += edgery;
            if (dy >= edgery) dy -= edgery;
          }
/* set new position */
         part[j+joff] = dx;
         part[j+joff+npe] = dy;
      }
   }
   nps = NPBLK*ipp;
/* loop over remaining particles */
   for (j = nps; j < nop; j++) {
/* find interpolation weights */
      x = part[j];
      y = part[j+npe];
      nn = x;
      mm = y;
      dxp = qm*(x - (float) nn);
```

```
      dyp = y - (float) mm;
/* find inverse gamma */
      ux = part[j+2*npe];
      uy = part[j+3*npe];
      uz = part[j+4*npe];
      p2 = ux*ux + uy*uy + uz*uz;
      gami = 1.0/sqrtf(1.0 + p2*ci2);
/* calculate weights */
      nn = N*(nn + nxv*mm);
      amx = qm - dxp;
      amy = 1.0f - dyp;
/* deposit current */
      dx = amx*amy;
      dy = dxp*amy;
      vx = ux*gami;
      vy = uy*gami;
      vz = uz*gami;
      cu[nn] += vx*dx;
      cu[nn+1] += vy*dx;
      cu[nn+2] += vz*dx;
      dx = amx*dyp;
      mm = nn + N;
      cu[mm] += vx*dy;
      cu[mm+1] += vy*dy;
      cu[mm+2] += vz*dy;
      dy = dxp*dyp;
      nn += N*nxv;
      cu[nn] += vx*dx;
      cu[nn+1] += vy*dx;
      cu[nn+2] += vz*dx;
      mm = nn + N;
      cu[mm] += vx*dy;
      cu[mm+1] += vy*dy;
      cu[mm+2] += vz*dy;
/* advance position half a time-step */
      dx = x + vx*dt;
      dy = y + vy*dt;
/* periodic boundary conditions */
      if (ipbc==1) {
         if (dx < edgelx) dx += edgerx;
         if (dx >= edgerx) dx -= edgerx;
         if (dy < edgely) dy += edgery;
         if (dy >= edgery) dy -= edgery;
      }
/* reflecting boundary conditions */
      else if (ipbc==2) {
         if ((dx < edgelx) || (dx >= edgerx)) {
            dx = x;
            part[j+2*npe] = -ux;
         }
         if ((dy < edgely) || (dy >= edgery)) {
            dy = y;
            part[j+3*npe] = -uy;
         }
```

```
      }
/* mixed reflecting/periodic boundary conditions */
      else if (ipbc==3) {
         if ((dx < edgelx) || (dx >= edgerx)) {
            dx = x;
            part[j+2*npe] = -ux;
         }
         if (dy < edgely) dy += edgery;
         if (dy >= edgery) dy -= edgery;
      }
/* set new position */
      part[j] = dx;
      part[j+npe] = dy;
   }
   return;
#undef LVECT
#undef NPBLK
#undef N
}
```