```
/*--------------------------------------------------------------------*/
void cvgpush2lt(float part[], float fxy[], float qbm, float dt,
                float *ek, int idimp, int nop, int npe, int nx, int ny,
                int nxv, int nyv, int ipbc) {
/* for 2d code, this subroutine updates particle co-ordinates and
   velocities using leap-frog scheme in time and first-order linear
   interpolation in space, with various boundary conditions.
   vectorizable version using guard cells
   44 flops/particle, 12 loads, 4 stores
   input: all, output: part, ek
   equations used are:
   vx(t+dt/2) = vx(t-dt/2) + (q/m)*fx(x(t),y(t))*dt,
   vy(t+dt/2) = vy(t-dt/2) + (q/m)*fy(x(t),y(t))*dt,
   where q/m is charge/mass, and
   x(t+dt) = x(t) + vx(t+dt/2)*dt, y(t+dt) = y(t) + vy(t+dt/2)*dt
   fx(x(t),y(t)) and fy(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))
   fy(x,y) = (1-dy)*((1-dx)*fy(n,m)+dx*fy(n+1,m)) + dy*((1-dx)*fy(n,m+1)
      + dx*fy(n+1,m+1))
   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
   part[2][n] = velocity vx of particle n
   part[3][n] = velocity vy 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)
   that is, convolution of electric field over particle shape
   qbm = particle charge/mass
   dt = time interval between successive calculations
   kinetic energy/mass at time t is also calculated, using
   ek = .125*sum((vx(t+dt/2)+vx(t-dt/2))**2+(vy(t+dt/2)+vy(t-dt/2))**2)
   idimp = size of phase space = 4
   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
   int i, j, k, ipp, joff, nps, nn, mm;
   float qtm, edgelx, edgely, edgerx, edgery, dxp, dyp, amx, amy;
   float x, y, dx, dy, vx, vy;
/* scratch arrays */
   int n[NPBLK];
   float s[NPBLK*LVECT], t[NPBLK*2];
   double sum1;
   qtm = qbm*dt;
   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] = nn + nxv*mm;
         amx = 1.0f - 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;
         t[j] = x;
         t[j+NPBLK] = y;
      }
/* find acceleration */
      for (j = 0; j < NPBLK; j++) {
         nn = n[j];
         mm = nn + nxv - 2;
         dx = 0.0f;
         dy = 0.0f;
#pragma ivdep
         for (i = 0; i < LVECT; i++) {
            if (i > 1)
               nn = mm;
            dx += fxy[2*(i+nn)]*s[j+NPBLK*i];
            dy += fxy[1+2*(i+nn)]*s[j+NPBLK*i];
         }
         s[j] = dx;
         s[j+NPBLK] = dy;
      }
/* new velocity */
```

```
      for (j = 0; j < NPBLK; j++) {
         x = t[j];
         y = t[j+NPBLK];
         dxp = part[j+joff+2*npe];
         dyp = part[j+joff+3*npe];
         vx = dxp + qtm*s[j];
         vy = dyp + qtm*s[j+NPBLK];
/* average kinetic energy */
         dxp += vx;
         dyp += vy;
         sum1 += dxp*dxp + dyp*dyp;
/* new position */
         s[j] = x + vx*dt;
         s[j+NPBLK] = y + vy*dt;
         s[j+2*NPBLK] = vx;
         s[j+3*NPBLK] = vy;
      }
/* check boundary conditions */
#pragma novector
      for (j = 0; j < NPBLK; j++) {
         dx = s[j];
         dy = s[j+NPBLK];
         vx = s[j+2*NPBLK];
         vy = s[j+3*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;
      }
   }
   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;
      nn = 2*(nn + nxv*mm);
      amx = 1.0f - dxp;
      amy = 1.0f - dyp;
/* find acceleration */
      dx = amx*fxy[nn];
      dy = amx*fxy[nn+1];
      dx = amy*(dxp*fxy[nn+2] + dx);
      dy = amy*(dxp*fxy[nn+3] + dy);
      nn += 2*nxv;
      vx = amx*fxy[nn];
      vy = amx*fxy[nn+1];
      dx += dyp*(dxp*fxy[nn+2] + vx);
      dy += dyp*(dxp*fxy[nn+3] + vy);
/* new velocity */
      dxp = part[j+2*npe];
      dyp = part[j+3*npe];
      vx = dxp + qtm*dx;
      vy = dyp + qtm*dy;
/* average kinetic energy */
      dxp += vx;
      dyp += vy;
      sum1 += dxp*dxp + dyp*dyp;
/* new position */
      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;
            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;
   }
/* normalize kinetic energy */
   *ek += 0.125f*sum1;
   return;
#undef LVECT
#undef NPBLK
}
```

```
/*--------------------------------------------------------------------*/
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
}
```