```
/*--------------------------------------------------------------------*/
void cvgppushf2lt(float ppart[], float fxy[], int kpic[], int ncl[],
                  int ihole[], float qbm, float dt, float *ek,
                  int idimp, int nppmx, int nx, int ny, int mx, int my,
                  int nxv, int nyv, int mx1, int mxy1, int ntmax,
                  int *irc) {
/* 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 periodic boundary conditions.
   also determines list of particles which are leaving this tile
   vectorizable/OpenMP version using guard cells
   data read in tiles
   particles stored segmented array
   44 flops/particle, 12 loads, 4 stores
   input: all except ncl, ihole, irc, output: ppart, ncl, ihole, ek, irc
   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
   ppart[m][0][n] = position x of particle n in tile m
   ppart[m][1][n] = position y of particle n in tile m
   ppart[m][2][n] = velocity vx of particle n in tile m
   ppart[m][3][n] = velocity vy of particle n in tile m
   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
   kpic[k] = number of particles in tile k
   ncl[k][i] = number of particles going to destination i, tile k
   ihole[k][:][0] = location of hole in array left by departing particle
   ihole[k][:][1] = destination of particle leaving hole
   ihole[k][0][0] = ih, number of holes left (error, if negative)
   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
   nppmx = maximum number of particles in tile
   nx/ny = system length in x/y direction
   mx/my = number of grids in sorting cell in x/y
   nxv = second dimension of field arrays, must be >= nx+1
   nyv = third dimension of field arrays, must be >= ny+1
   mx1 = (system length in x direction - 1)/mx + 1
   mxy1 = mx1*my1, where my1 = (system length in y direction - 1)/my + 1
   ntmax = size of hole array for particles leaving tiles
   irc = maximum overflow, returned only if error occurs, when irc > 0
   optimized version
```

```
local data                                                            */
#define MXV             33
#define MYV             33
#define NPBLK             32
#define LVECT             4
   int noff, moff, npoff, npp, ipp, joff, nps;
   int i, j, k, m, ih, nh, nn, mm, lxv;
   float qtm, dxp, dyp, amx, amy;
   float x, y, dx, dy, vx, vy;
   float anx, any, edgelx, edgely, edgerx, edgery;
   float sfxy[2*MXV*MYV];
/* float sfxy[2*(mx+1)*(my+1)]; */
/* scratch arrays */
   int n[NPBLK];
   float s[NPBLK*LVECT], t[NPBLK*2];
   double sum1, sum2;
   lxv = mx + 1;
   qtm = qbm*dt;
   anx = (float) nx;
   any = (float) ny;
   sum2 = 0.0;
/* error if local array is too small */
/* if ((mx >= MXV) || (my >= MYV))   */
/*    return;                        */
/* loop over tiles */
#pragma omp parallel for \
private(i,j,k,m,noff,moff,npp,npoff,ipp,joff,nps,nn,mm,ih,nh,x,y,dxp, \
dyp,amx,amy,dx,dy,vx,vy,edgelx,edgely,edgerx,edgery,sum1,sfxy,n,s,t) \
reduction(+:sum2)
   for (k = 0; k < mxy1; k++) {
      noff = k/mx1;
      moff = my*noff;
      noff = mx*(k - mx1*noff);
      npp = kpic[k];
      npoff = idimp*nppmx*k;
      nn = nx - noff;
      nn = mx < nn ? mx : nn;
      mm = ny - moff;
      mm = my < mm ? my : mm;
      edgelx = noff;
      edgerx = noff + nn;
      edgely = moff;
      edgery = moff + mm;
      ih = 0;
      nh = 0;
      nn += 1;
      mm += 1;
/* load local fields from global array */
      for (j = 0; j < mm; j++) {
         for (i = 0; i < nn; i++) {
            sfxy[2*(i+lxv*j)] = fxy[2*(i+noff+nxv*(j+moff))];
            sfxy[1+2*(i+lxv*j)] = fxy[1+2*(i+noff+nxv*(j+moff))];
         }
      }
```

```
/* clear counters */
      for (j = 0; j < 8; j++) {
         ncl[j+8*k] = 0;
      }
      sum1 = 0.0;
/* loop over particles in tile */
      ipp = npp/NPBLK;
/* outer loop over number of full blocks */
      for (m = 0; m < ipp; m++) {
         joff = NPBLK*m;
/* inner loop over particles in block */
         for (j = 0; j < NPBLK; j++) {
/* find interpolation weights */
            x = ppart[j+joff+npoff];
            y = ppart[j+joff+nppmx+npoff];
            nn = x;
            mm = y;
            dxp = x - (float) nn;
            dyp = y - (float) mm;
            n[j] = nn - noff + lxv*(mm - moff);
            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 + lxv - 2;
            dx = 0.0f;
            dy = 0.0f;
#pragma ivdep
            for (i = 0; i < LVECT; i++) {
               if (i > 1)
                  nn = mm;
               dx += sfxy[2*(i+nn)]*s[j+NPBLK*i];
               dy += sfxy[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 = ppart[j+joff+2*nppmx+npoff];
            dyp = ppart[j+joff+3*nppmx+npoff];
            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];
/* find particles going out of bounds */
            mm = 0;
/* count how many particles are going in each direction in ncl   */
/* save their address and destination in ihole                   */
/* use periodic boundary conditions and check for roundoff error */
/* mm = direction particle is going                              */
            if (dx >= edgerx) {
               if (dx >= anx)
                  dx -= anx;
               mm = 2;
            }
            else if (dx < edgelx) {
               if (dx < 0.0f) {
                  dx += anx;
                  if (dx < anx)
                     mm = 1;
                  else
                     dx = 0.0;
               }
               else {
                  mm = 1;
               }
            }
            if (dy >= edgery) {
               if (dy >= any)
                  dy -= any;
               mm += 6;
            }
            else if (dy < edgely) {
               if (dy < 0.0) {
                  dy += any;
                  if (dy < any)
                     mm += 3;
                  else
                     dy = 0.0;
               }
               else {
                  mm += 3;
               }
            }
```

```
/* set new position */
            ppart[j+joff+npoff] = dx;
            ppart[j+joff+nppmx+npoff] = dy;
/* set new velocity */
            ppart[j+joff+2*nppmx+npoff] = s[j+2*NPBLK];
            ppart[j+joff+3*nppmx+npoff] = s[j+3*NPBLK];
/* increment counters */
            if (mm > 0) {
               ncl[mm+8*k-1] += 1;
               ih += 1;
               if (ih <= ntmax) {
                  ihole[2*(ih+(ntmax+1)*k)] = j + joff + 1;
                  ihole[1+2*(ih+(ntmax+1)*k)] = mm;
               }
               else {
                  nh = 1;
               }
            }
         }
      }
      nps = NPBLK*ipp;
/* loop over remaining particles */
      for (j = nps; j < npp; j++) {
/* find interpolation weights */
         x = ppart[j+npoff];
         y = ppart[j+nppmx+npoff];
         nn = x;
         mm = y;
         dxp = x - (float) nn;
         dyp = y - (float) mm;
         nn = 2*(nn - noff + lxv*(mm - moff));
         amx = 1.0f - dxp;
         amy = 1.0f - dyp;
/* find acceleration */
         dx = amx*sfxy[nn];
         dy = amx*sfxy[nn+1];
         dx = amy*(dxp*sfxy[nn+2] + dx);
         dy = amy*(dxp*sfxy[nn+3] + dy);
         nn += 2*lxv;
         vx = amx*sfxy[nn];
         vy = amx*sfxy[nn+1];
         dx += dyp*(dxp*sfxy[nn+2] + vx);
         dy += dyp*(dxp*sfxy[nn+3] + vy);
/* new velocity */
         dxp = ppart[j+2*nppmx+npoff];
         dyp = ppart[j+3*nppmx+npoff];
         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;
/* find particles going out of bounds */
         mm = 0;
/* count how many particles are going in each direction in ncl   */
/* save their address and destination in ihole                   */
/* use periodic boundary conditions and check for roundoff error */
/* mm = direction particle is going                              */
         if (dx >= edgerx) {
            if (dx >= anx)
               dx -= anx;
            mm = 2;
         }
         else if (dx < edgelx) {
            if (dx < 0.0f) {
               dx += anx;
               if (dx < anx)
                  mm = 1;
               else
                  dx = 0.0;
            }
            else {
               mm = 1;
            }
         }
         if (dy >= edgery) {
            if (dy >= any)
               dy -= any;
            mm += 6;
         }
         else if (dy < edgely) {
            if (dy < 0.0) {
               dy += any;
               if (dy < any)
                  mm += 3;
               else
                  dy = 0.0;
            }
            else {
               mm += 3;
            }
         }
/* set new position */
         ppart[j+npoff] = dx;
         ppart[j+nppmx+npoff] = dy;
/* set new velocity */
         ppart[j+2*nppmx+npoff] = vx;
         ppart[j+3*nppmx+npoff] = vy;
/* increment counters */
         if (mm > 0) {
            ncl[mm+8*k-1] += 1;
            ih += 1;
            if (ih <= ntmax) {
               ihole[2*(ih+(ntmax+1)*k)] = j + 1;
               ihole[1+2*(ih+(ntmax+1)*k)] = mm;
            }
            else {
               nh = 1;
            }
         }
      }
      sum2 += sum1;
/* set error and end of file flag */
/* ihole overflow */
      if (nh > 0) {
         *irc = ih;
         ih = -ih;
      }
      ihole[2*(ntmax+1)*k] = ih;
   }
/* normalize kinetic energy */
   *ek += 0.125f*sum2;
   return;
#undef LVECT
#undef NPBLK
#undef MXV
#undef MYV
}
```

```
/*--------------------------------------------------------------------*/
void cvgppost2lt(float ppart[], float q[], int kpic[], float qm,
                 int nppmx, int idimp, int mx, int my, int nxv, int nyv,
                 int mx1, int mxy1) {
/* for 2d code, this subroutine calculates particle charge density
   using first-order linear interpolation, periodic boundaries
   vectorizable/OpenMP version using guard cells
   data deposited in tiles
   particles stored segmented array
   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
   ppart[m][0][n] = position x of particle n in tile m
   ppart[m][1][n] = position y of particle n in tile m
   q[k][j] = charge density at grid point j,k
   kpic = number of particles per tile
   qm = charge on particle, in units of e
   nppmx = maximum number of particles in tile
   idimp = size of phase space = 4
   mx/my = number of grids in sorting cell in x/y
   nxv = first dimension of charge array, must be >= nx+1
   nyv = second dimension of charge array, must be >= ny+1
   mx1 = (system length in x direction - 1)/mx + 1
   mxy1 = mx1*my1, where my1 = (system length in y direction - 1)/my + 1
local data                                                            */
#define MXV             33
#define MYV             33
#define NPBLK             32
#define LVECT             4
   int noff, moff, npoff, npp, ipp, joff, nps;
   int i, j, k, m, nn, mm, lxv;
   float x, y, dxp, dyp, amx, amy;
   float sq[MXV*MYV];
/* float sq[(mx+1)*(my+1)]; */
/* scratch arrays */
   int n[NPBLK];
   float s[NPBLK*LVECT];
   lxv = mx + 1;
/* error if local array is too small */
/* if ((mx >= MXV) || (my >= MYV))   */
/*    return;                        */
/* loop over tiles */
#pragma omp parallel for \
private(i,j,k,m,noff,moff,npp,npoff,ipp,joff,nps,nn,mm,x,y,dxp,dyp, \
amx,amy,sq,n,s)
   for (k = 0; k < mxy1; k++) {
      noff = k/mx1;
      moff = my*noff;
      noff = mx*(k - mx1*noff);
      npp = kpic[k];
      npoff = idimp*nppmx*k;
/* zero out local accumulator */
      for (j = 0; j < lxv*(my+1); j++) {
         sq[j] = 0.0f;
      }
/* loop over particles in tile */
      ipp = npp/NPBLK;
/* outer loop over number of full blocks */
      for (m = 0; m < ipp; m++) {
         joff = NPBLK*m;
/* inner loop over particles in block */
         for (j = 0; j < NPBLK; j++) {
/* find interpolation weights */
            x = ppart[j+joff+npoff];
            y = ppart[j+joff+nppmx+npoff];
            nn = x;
            mm = y;
            dxp = qm*(x - (float) nn);
            dyp = y - (float) mm;
            n[j] = nn - noff + lxv*(mm - moff);
            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 within tile to local accumulator */
         for (j = 0; j < NPBLK; j++) {
            nn = n[j];
            mm = nn + lxv - 2;
#pragma ivdep
            for (i = 0; i < LVECT; i++) {
               if (i > 1)
                  nn = mm;
               sq[i+nn] += s[j+NPBLK*i];
            }
         }
      }
      nps = NPBLK*ipp;
/* loop over remaining particles */
      for (j = nps; j < npp; j++) {
/* find interpolation weights */
         x = ppart[j+npoff];
         y = ppart[j+nppmx+npoff];
         nn = x;
         mm = y;
         dxp = qm*(x - (float) nn);
         dyp = y - (float) mm;
         nn = nn - noff + lxv*(mm - moff);
         amx = qm - dxp;
         amy = 1.0f - dyp;
/* deposit charge within tile to local accumulator */
```

```
         x = sq[nn] + amx*amy;
         y = sq[nn+1] + dxp*amy;
         sq[nn] = x;
         sq[nn+1] = y;
         nn += lxv;
         x = sq[nn] + amx*dyp;
         y = sq[nn+1] + dxp*dyp;
         sq[nn] = x;
         sq[nn+1] = y;
      }
/* deposit charge to interior points in global array */
      nn = nxv - noff;
      mm = nyv - moff;
      nn = mx < nn ? mx : nn;
      mm = my < mm ? my : mm;
      for (j = 1; j < mm; j++) {
         for (i = 1; i < nn; i++) {
            q[i+noff+nxv*(j+moff)] += sq[i+lxv*j];
         }
      }
/* deposit charge to edge points in global array */
      mm = nyv - moff;
      mm = my+1 < mm ? my+1 : mm;
      for (i = 1; i < nn; i++) {
#pragma omp atomic
         q[i+noff+nxv*moff] += sq[i];
         if (mm > my) {
#pragma omp atomic
            q[i+noff+nxv*(mm+moff-1)] += sq[i+lxv*(mm-1)];
         }
      }
      nn = nxv - noff;
      nn = mx+1 < nn ? mx+1 : nn;
      for (j = 0; j < mm; j++) {
#pragma omp atomic
         q[noff+nxv*(j+moff)] += sq[lxv*j];
         if (nn > mx) {
#pragma omp atomic
            q[nn+noff-1+nxv*(j+moff)] += sq[nn-1+lxv*j];
         }
      }
   }
   return;
#undef LVECT
#undef NPBLK
#undef MXV
#undef MYV
}
```