```
/*--------------------------------------------------------------------*/
void cppgppushf2l(float ppart[], float fxy[], int kpic[], int ncl[],
                  int ihole[], int noff, int nyp, float qbm, float dt,
                  float *ek, int nx, int ny, int mx, int my, int idimp,
                  int nppmx, int nxv, int nypmx, int mx1, int mxyp1,
                  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
   OpenMP version using guard cells, for distributed data
   data read in tiles
   particles stored segmented array
   42 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][n][0] = position x of particle n in partition in tile m
   ppart[m][n][1] = position y of particle n in partition in tile m
   ppart[m][n][2] = velocity vx of particle n in partition in tile m
   ppart[m][n][3] = velocity vy of particle n in partition in tile m
   fxy[k][j][0] = x component of force/charge at grid (j,kk)
   fxy[k][j][1] = y component of force/charge at grid (j,kk)
   in other words, fxy are the convolutions of the electric field
   over the particle shape, where kk = k + noff
   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)
   noff = lowermost global gridpoint in particle partition.
   nyp = number of primary (complete) gridpoints in particle partition
   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)
   nx/ny = system length in x/y direction
   mx/my = number of grids in sorting cell in x/y
   idimp = size of phase space = 4
   nppmx = maximum number of particles in tile
   nxv = first dimension of field array, must be >= nx+1
   nypmx = maximum size of particle partition, including guard cells.
   mx1 = (system length in x direction - 1)/mx + 1
   mxyp1 = mx1*myp1, where myp1=(partition 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
   int noffp, moffp, npoff, nppp;
   int mnoff, i, j, k, ih, nh, nn, mm, mxv;
   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)]; */
   double sum1, sum2;
   mxv = 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,noffp,moffp,nppp,npoff,nn,mm,ih,nh,mnoff,x,y,dxp,dyp, \
amx,amy,dx,dy,vx,vy,edgelx,edgely,edgerx,edgery,sum1,sfxy) \
reduction(+:sum2)
   for (k = 0; k < mxyp1; k++) {
      noffp = k/mx1;
      moffp = my*noffp;
      noffp = mx*(k - mx1*noffp);
      nppp = kpic[k];
      npoff = nppmx*k;
      nn = nx - noffp;
      nn = mx < nn ? mx : nn;
      mm = nyp - moffp;
      mm = my < mm ? my : mm;
      edgelx = noffp;
      edgerx = noffp + nn;
      edgely = noff + moffp;
      edgery = noff + moffp + mm;
      ih = 0;
      nh = 0;
      nn += 1;
      mm += 1;
      mnoff = moffp + noff;
/* load local fields from global array */
      for (j = 0; j < mm; j++) {
         for (i = 0; i < nn; i++) {
            sfxy[2*(i+mxv*j)] = fxy[2*(i+noffp+nxv*(j+moffp))];
            sfxy[1+2*(i+mxv*j)] = fxy[1+2*(i+noffp+nxv*(j+moffp))];
         }
      }
/* clear counters */
```

```
      for (j = 0; j < 8; j++) {
         ncl[j+8*k] = 0;
      }
      sum1 = 0.0;
/* loop over particles in tile */
      for (j = 0; j < nppp; j++) {
/* find interpolation weights */
         x = ppart[idimp*(j+npoff)];
         y = ppart[1+idimp*(j+npoff)];
         nn = x;
         mm = y;
         dxp = x - (float) nn;
         dyp = y - (float) mm;
         nn = 2*(nn - noffp) + 2*mxv*(mm - mnoff);
         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*mxv;
         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 */
         vx = ppart[2+idimp*(j+npoff)];
         vy = ppart[3+idimp*(j+npoff)];
         dx = vx + qtm*dx;
         dy = vy + qtm*dy;
/* average kinetic energy */
         vx += dx;
         vy += dy;
         sum1 += vx*vx + vy*vy;
         ppart[2+idimp*(j+npoff)] = dx;
         ppart[3+idimp*(j+npoff)] = dy;
/* new position */
         dx = x + dx*dt;
         dy = y + dy*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[idimp*(j+npoff)] = dx;
         ppart[1+idimp*(j+npoff)] = dy;
/* 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 MXV
#undef MYV
}
```

```
/*--------------------------------------------------------------------*/
void cppgppost2l(float ppart[], float q[], int kpic[], int noff,
                 float qm, int idimp, int nppmx, int mx, int my,
                 int nxv, int nypmx, int mx1, int mxyp1) {
/* for 2d code, this subroutine calculates particle charge density
   using first-order linear interpolation, periodic boundaries
   OpenMP version using guard cells, for distributed data
   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][n][0] = position x of particle n in partition in tile m
   ppart[m][n][1] = position y of particle n in partition in tile m
   q[k][j] = charge density at grid point (j,kk),
   where kk = k + noff
   kpic = number of particles per tile
   noff = lowermost global gridpoint in particle partition.
   qm = charge on particle, in units of e
   idimp = size of phase space = 4
   nppmx = maximum number of particles in tile
   mx/my = number of grids in sorting cell in x/y
   nxv = first dimension of charge array, must be >= nx+1
   nypmx = maximum size of particle partition, including guard cells.
   mx1 = (system length in x direction - 1)/mx + 1
   mxyp1 = mx1*myp1, where myp1=(partition length in y direction-1)/my+1
local data                                                            */
#define MXV             33
#define MYV             33
   int noffp, moffp, npoff, nppp, mxv;
   int mnoff, i, j, k, nn, mm;
   float x, y, dxp, dyp, amx, amy;
   float sq[MXV*MYV];
/* float sq[(mx+1)*(my+1)]; */
   mxv = 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,noffp,moffp,nppp,npoff,mnoff,nn,mm,x,y,dxp,dyp,amx,amy, \
sq)
   for (k = 0; k < mxyp1; k++) {
      noffp = k/mx1;
      moffp = my*noffp;
      noffp = mx*(k - mx1*noffp);
      nppp = kpic[k];
      npoff = nppmx*k;
      mnoff = moffp + noff;
```

```
/* zero out local accumulator */
      for (j = 0; j < my+1; j++) {
         for (i = 0; i < mx+1; i++) {
            sq[i+mxv*j] = 0.0f;
         }
      }
/* loop over particles in tile */
      for (j = 0; j < nppp; j++) {
/* find interpolation weights */
         x = ppart[idimp*(j+npoff)];
         y = ppart[1+idimp*(j+npoff)];
         nn = x;
         mm = y;
         dxp = qm*(x - (float) nn);
         dyp = y - (float) mm;
         nn = nn - noffp + mxv*(mm - mnoff);
         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 += mxv;
         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 - noffp;
      mm = nypmx - moffp;
      nn = mx < nn ? mx : nn;
      mm = my < mm ? my : mm;
      for (j = 1; j < mm; j++) {
         for (i = 1; i < nn; i++) {
            q[i+noffp+nxv*(j+moffp)] += sq[i+mxv*j];
         }
      }
/* deposit charge to edge points in global array */
      mm = nypmx - moffp;
      mm = my+1 < mm ? my+1 : mm;
      for (i = 1; i < nn; i++) {
#pragma omp atomic
         q[i+noffp+nxv*moffp] += sq[i];
         if (mm > my) {
#pragma omp atomic
            q[i+noffp+nxv*(mm+moffp-1)] += sq[i+mxv*(mm-1)];
         }
      }
      nn = nxv - noffp;
      nn = mx+1 < nn ? mx+1 : nn;
      for (j = 0; j < mm; j++) {
#pragma omp atomic
```

```
         q[noffp+nxv*(j+moffp)] += sq[mxv*j];
         if (nn > mx) {
#pragma omp atomic
            q[nn+noffp-1+nxv*(j+moffp)] += sq[nn-1+mxv*j];
         }
      }
   }
   return;
#undef MXV
#undef MYV
}
```