```
/*--------------------------------------------------------------------*/
void cppgpush2l(float part[], float fxy[], float edges[], int npp,
                int noff, int ihole[], float qbm, float dt, float *ek,
                int nx, int ny, int idimp, int npmax, int nxv,
                int nypmx, int idps, int ntmax, 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
   also determines list of particles which are leaving this processor
   scalar version using guard cells, for distributed data
   42 flops/particle, 12 loads, 4 stores
   input: all except ihole, output: part, ihole, 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[n][0] = position x of particle n in partition
   part[n][1] = position y of particle n in partition
   part[n][2] = velocity vx of particle n in partition
   part[n][3] = velocity vy of particle n in partition
   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
   edges[0:1] = lower:upper boundary of particle partition
   npp = number of particles in partition
   noff = lowermost global gridpoint in particle partition.
   ihole = location of hole left in particle arrays
   ihole[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)
   nx/ny = system length in x/y direction
   idimp = size of phase space = 4
   npmax = maximum number of particles in each partition
   nxv = first dimension of field array, must be >= nx+1
   nypmx = maximum size of particle partition, including guard cells.
   idps = number of partition boundaries
   ntmax = size of hole array for particles leaving processors
   ipbc = particle boundary condition = (0,1,2,3) =
   (none,2d periodic,2d reflecting,mixed reflecting/periodic)
local data                                                 */
   int mnoff, j, nn, mm, np, mp, ih, nh, nxv2;
   float qtm, edgelx, edgely, edgerx, edgery, dxp, dyp, amx, amy;
   float dx, dy, vx, vy;
```

```
   double sum1;
   nxv2 = 2*nxv;
   qtm = qbm*dt;
   sum1 = 0.0;
/* set boundary values */
   edgelx = 0.0;
   edgely = 1.0;
   edgerx = (float) nx;
   edgery = (float) (ny-1);
   if ((ipbc==2) || (ipbc==3)) {
      edgelx = 1.0;
      edgerx = (float) (nx-1);
   }
   mnoff = noff;
   ih = 0;
   nh = 0;
   for (j = 0; j < npp; j++) {
/* find interpolation weights */
      nn = part[idimp*j];
      mm = part[1+idimp*j];
      dxp = part[idimp*j] - (float) nn;
      dyp = part[1+idimp*j] - (float) mm;
      nn = 2*nn;
      mm = nxv2*(mm - mnoff);
      amx = 1.0 - dxp;
      mp = mm + nxv2;
      amy = 1.0 - dyp;
      np = nn + 2;
/* find acceleration */
      dx = dyp*(dxp*fxy[np+mp] + amx*fxy[nn+mp])
         + amy*(dxp*fxy[np+mm] + amx*fxy[nn+mm]);
      dy = dyp*(dxp*fxy[1+np+mp] + amx*fxy[1+nn+mp])
         + amy*(dxp*fxy[1+np+mm] + amx*fxy[1+nn+mm]);
/* new velocity */
      vx = part[2+idimp*j];
      vy = part[3+idimp*j];
      dx = vx + qtm*dx;
      dy = vy + qtm*dy;
/* average kinetic energy */
      vx += dx;
      vy += dy;
      sum1 += vx*vx + vy*vy;
      part[2+idimp*j] = dx;
      part[3+idimp*j] = dy;
/* new position */
      dx = part[idimp*j] + dx*dt;
      dy = part[1+idimp*j] + dy*dt;
/* periodic boundary conditions in x */
      if (ipbc==1) {
         if (dx < edgelx) dx += edgerx;
         if (dx >= edgerx) dx -= edgerx;
      }
/* reflecting boundary conditions */
      else if (ipbc==2) {
```

```
         if ((dx < edgelx) || (dx >= edgerx)) {
            dx = part[idimp*j];
            part[2+idimp*j] = -part[2+idimp*j];
         }
         if ((dy < edgely) || (dy >= edgery)) {
            dy = part[1+idimp*j];
            part[3+idimp*j] = -part[3+idimp*j];
         }
      }
/* mixed reflecting/periodic boundary conditions */
      else if (ipbc==3) {
         if ((dx < edgelx) || (dx >= edgerx)) {
            dx = part[idimp*j];
            part[2+idimp*j] = -part[2+idimp*j];
         }
      }
/* find particles out of bounds */
      if ((dy < edges[0]) || (dy >= edges[1])) {
         if (ih < ntmax)
            ihole[ih+1] = j + 1;
         else
            nh = 1;
         ih += 1;
      }
/* set new position */
      part[idimp*j] = dx;
      part[1+idimp*j] = dy;
   }
/* set end of file flag */
      if (nh > 0)
         ih = -ih;
      ihole[0] = ih;
/* normalize kinetic energy */
   *ek += 0.125*sum1;
   return;
}
```

```
/*--------------------------------------------------------------------*/
void cppgpost2l(float part[], float q[], int npp, int noff, float qm,
                int idimp, int npmax, int nxv, int nypmx) {
/* for 2d code, this subroutine calculates particle charge density
   using first-order linear interpolation, periodic boundaries
   scalar version using guard cells, for distributed data
   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[n][0] = position x of particle n in partition
   part[n][1] = position y of particle n in partition
   q[k][j] = charge density at grid point (j,kk),
   where kk = k + noff
   npp = number of particles in partition
   noff = lowermost global gridpoint in particle partition.
   qm = charge on particle, in units of e
   idimp = size of phase space = 4
   npmax = maximum number of particles in each partition
   nxv = first dimension of charge array, must be >= nx+1
   nypmx = maximum size of particle partition, including guard cells.
local data                                                            */
   int  mnoff, j, nn, np, mm, mp;
   float dxp, dyp, amx, amy;
   mnoff = noff;
   for (j = 0; j < npp; j++) {
/* find interpolation weights */
      nn = part[idimp*j];
      mm = part[1+idimp*j];
      dxp = qm*(part[idimp*j] - (float) nn);
      dyp = part[1+idimp*j] - (float) mm;
      mm = nxv*(mm - mnoff);
      amx = qm - dxp;
      mp = mm + nxv;
      amy = 1.0 - dyp;
      np = nn + 1;
/* deposit charge */
      q[np+mp] += dxp*dyp;
      q[nn+mp] += amx*dyp;
      q[np+mm] += dxp*amy;
      q[nn+mm] += amx*amy;
   }
   return;
}
```