```
c-----------------------------------------------------------------------
      subroutine VGPPUSHF2LT(ppart,fxy,kpic,ncl,ihole,qbm,dt,ek,idimp,
     1nppmx,nx,ny,mx,my,nxv,nyv,mx1,mxy1,ntmax,irc)
c for 2d code, this subroutine updates particle co-ordinates and
c velocities using leap-frog scheme in time and first-order linear
c interpolation in space, with periodic boundary conditions.
c also determines list of particles which are leaving this tile
c vectorizable/OpenMP version using guard cells
c data read in tiles
c particles stored segmented array
c 44 flops/particle, 12 loads, 4 stores
c input: all except ncl, ihole, irc, output: ppart, ncl, ihole, ek, irc
c equations used are:
c vx(t+dt/2) = vx(t-dt/2) + (q/m)*fx(x(t),y(t))*dt,
c vy(t+dt/2) = vy(t-dt/2) + (q/m)*fy(x(t),y(t))*dt,
c where q/m is charge/mass, and
c x(t+dt) = x(t) + vx(t+dt/2)*dt, y(t+dt) = y(t) + vy(t+dt/2)*dt
c fx(x(t),y(t)) and fy(x(t),y(t)) are approximated by interpolation from
c the nearest grid points:
c fx(x,y) = (1-dy)*((1-dx)*fx(n,m)+dx*fx(n+1,m)) + dy*((1-dx)*fx(n,m+1)
c    + dx*fx(n+1,m+1))
c fy(x,y) = (1-dy)*((1-dx)*fy(n,m)+dx*fy(n+1,m)) + dy*((1-dx)*fy(n,m+1)
c    + dx*fy(n+1,m+1))
c where n,m = leftmost grid points and dx = x-n, dy = y-m
c ppart(n,1,m) = position x of particle n in tile m
c ppart(n,2,m) = position y of particle n in tile m
c ppart(n,3,m) = velocity vx of particle n in tile m
c ppart(n,4,m) = velocity vy of particle n in tile m
c fxy(1,j,k) = x component of force/charge at grid (j,k)
c fxy(2,j,k) = y component of force/charge at grid (j,k)
c that is, convolution of electric field over particle shape
c kpic(k) = number of particles in tile k
c ncl(i,k) = number of particles going to destination i, tile k
c ihole(1,:,k) = location of hole in array left by departing particle
c ihole(2,:,k) = destination of particle leaving hole
c ihole(1,1,k) = ih, number of holes left (error, if negative)
c qbm = particle charge/mass
c dt = time interval between successive calculations
c kinetic energy/mass at time t is also calculated, using
c ek = .125*sum((vx(t+dt/2)+vx(t-dt/2))**2+(vy(t+dt/2)+vy(t-dt/2))**2)
c idimp = size of phase space = 4
c nppmx = maximum number of particles in tile
c nx/ny = system length in x/y direction
c mx/my = number of grids in sorting cell in x/y
c nxv = second dimension of field arrays, must be >= nx+1
c nyv = third dimension of field arrays, must be >= ny+1
c mx1 = (system length in x direction - 1)/mx + 1
c mxy1 = mx1*my1, where my1 = (system length in y direction - 1)/my + 1
c ntmax = size of hole array for particles leaving tiles
c irc = maximum overflow, returned only if error occurs, when irc > 0
c optimized version
      implicit none
      integer idimp, nppmx, nx, ny, mx, my, nxv, nyv, mx1, mxy1, ntmax
      integer irc
      real qbm, dt, ek
      real ppart, fxy
      integer kpic, ncl, ihole
      dimension ppart(nppmx,idimp,mxy1), fxy(2,nxv*nyv)
      dimension kpic(mxy1), ncl(8,mxy1)
      dimension ihole(2,ntmax+1,mxy1)
c local data
      integer MXV, MYV
      parameter(MXV=33,MYV=33)
      integer npblk, lvect
      parameter(npblk=32,lvect=4)
      integer noff, moff, npp, ipp, joff, nps
      integer i, j, k, m, ih, nh, nn, mm, lxv
      real qtm, dxp, dyp, amx, amy
      real x, y, dx, dy, vx, vy
      real anx, any, edgelx, edgely, edgerx, edgery
      real sfxy
      dimension sfxy(2,MXV*MYV)
c     dimension sfxy(2,(mx+1)*(my+1))
c scratch arrays
      integer n
      real s, t
      dimension n(npblk), s(npblk,lvect), t(npblk,2)
      double precision sum1, sum2
      lxv = mx + 1
      qtm = qbm*dt
      anx = real(nx)
      any = real(ny)
      sum2 = 0.0d0
c error if local array is too small
c     if ((mx.ge.MXV).or.(my.ge.MYV)) return
c loop over tiles
!$OMP PARALLEL DO
!$OMP& PRIVATE(i,j,k,m,noff,moff,npp,ipp,joff,nps,nn,mm,ih,nh,x,y,dxp,
!$OMP& dyp,amx,amy,dx,dy,vx,vy,edgelx,edgely,edgerx,edgery,sum1,sfxy,n,
!$OMP& s,t)
!$OMP& REDUCTION(+:sum2)
      do 110 k = 1, mxy1
      noff = (k - 1)/mx1
      moff = my*noff
      noff = mx*(k - mx1*noff - 1)
      npp = kpic(k)
      nn = min(mx,nx-noff)
      mm = min(my,ny-moff)
      edgelx = noff
      edgerx = noff + nn
      edgely = moff
      edgery = moff + mm
      ih = 0
      nh = 0
c load local fields from global array
      do 20 j = 1, mm+1
      do 10 i = 1, nn+1
      sfxy(1,i+lxv*(j-1)) = fxy(1,i+noff+nxv*(j+moff-1))
      sfxy(2,i+lxv*(j-1)) = fxy(2,i+noff+nxv*(j+moff-1))
   10 continue
   20 continue
c clear counters
      do 30 j = 1, 8
      ncl(j,k) = 0
   30 continue
      sum1 = 0.0d0
c loop over particles in tile
      ipp = npp/npblk
c outer loop over number of full blocks
      do 90 m = 1, ipp
      joff = npblk*(m - 1)
c inner loop over particles in block
      do 40 j = 1, npblk
c find interpolation weights
      x = ppart(j+joff,1,k)
      y = ppart(j+joff,2,k)
      nn = x
      mm = y
      dxp = x - real(nn)
      dyp = y - real(mm)
      n(j) = nn - noff + lxv*(mm - moff)
      amx = 1.0 - dxp
      amy = 1.0 - dyp
      s(j,1) = amx*amy
      s(j,2) = dxp*amy
      s(j,3) = amx*dyp
      s(j,4) = dxp*dyp
      t(j,1) = x
      t(j,2) = y
   40 continue
c find acceleration
      do 60 j = 1, npblk
      nn = n(j)
      mm = nn + lxv - 2
      dx = 0.0
      dy = 0.0
!dir$ ivdep
      do 50 i = 1, lvect
      if (i.gt.2) nn = mm
      dx = dx + sfxy(1,i+nn)*s(j,i)
      dy = dy + sfxy(2,i+nn)*s(j,i)
   50 continue
      s(j,1) = dx
      s(j,2) = dy
   60 continue
c new velocity
      do 70 j = 1, npblk
      x = t(j,1)
      y = t(j,2)
      dxp = ppart(j+joff,3,k)
      dyp = ppart(j+joff,4,k)
      vx = dxp + qtm*s(j,1)
      vy = dyp + qtm*s(j,2)
c average kinetic energy
      dxp = dxp + vx
      dyp = dyp + vy
      sum1 = sum1 + (dxp*dxp + dyp*dyp)
c new position
      s(j,1) = x + vx*dt
      s(j,2) = y + vy*dt
      s(j,3) = vx
      s(j,4) = vy
   70 continue
c check boundary conditions
!dir$ novector
      do 80 j = 1, npblk
      dx = s(j,1)
      dy = s(j,2)
c find particles going out of bounds
      mm = 0
c count how many particles are going in each direction in ncl
c save their address and destination in ihole
c use periodic boundary conditions and check for roundoff error
c mm = direction particle is going
      if (dx.ge.edgerx) then
         if (dx.ge.anx) dx = dx - anx
         mm = 2
      else if (dx.lt.edgelx) then
         if (dx.lt.0.0) then
            dx = dx + anx
            if (dx.lt.anx) then
               mm = 1
            else
               dx = 0.0
            endif
         else
            mm = 1
         endif
      endif
      if (dy.ge.edgery) then
         if (dy.ge.any) dy = dy - any
         mm = mm + 6
      else if (dy.lt.edgely) then
         if (dy.lt.0.0) then
            dy = dy + any
            if (dy.lt.any) then
               mm = mm + 3
            else
               dy = 0.0
            endif
         else
            mm = mm + 3
         endif
      endif
c set new position
      ppart(j+joff,1,k) = dx
```

```
         ppart(j+joff,2,k) = dy
c set new velocity
         ppart(j+joff,3,k) = s(j,3)
         ppart(j+joff,4,k) = s(j,4)
c increment counters
         if (mm.gt.0) then
            ncl(mm,k) = ncl(mm,k) + 1
            ih = ih + 1
            if (ih.le.ntmax) then
               ihole(1,ih+1,k) = j + joff
               ihole(2,ih+1,k) = mm
            else
               nh = 1
            endif
         endif
   80 continue
   90 continue
      nps = npblk*ipp + 1
c loop over remaining particles
      do 100 j = nps, npp
c find interpolation weights
      x = ppart(j,1,k)
      y = ppart(j,2,k)
      nn = x
      mm = y
      dxp = x - real(nn)
      dyp = y - real(mm)
      nn = nn - noff + 1 + lxv*(mm - moff)
      amx = 1.0 - dxp
      amy = 1.0 - dyp
c find acceleration
      dx = amx*sfxy(1,nn)
      dy = amx*sfxy(2,nn)
      dx = amy*(dxp*sfxy(1,nn+1) + dx)
      dy = amy*(dxp*sfxy(2,nn+1) + dy)
      vx = amx*sfxy(1,nn+lxv)
      vy = amx*sfxy(2,nn+lxv)
      dx = dx + dyp*(dxp*sfxy(1,nn+1+lxv) + vx)
      dy = dy + dyp*(dxp*sfxy(2,nn+1+lxv) + vy)
c new velocity
      dxp = ppart(j,3,k)
      dyp = ppart(j,4,k)
      vx = dxp + qtm*dx
      vy = dyp + qtm*dy
c average kinetic energy
      dxp = dxp + vx
      dyp = dyp + vy
      sum1 = sum1 + (dxp*dxp + dyp*dyp)
c new position
      dx = x + vx*dt
      dy = y + vy*dt
c find particles going out of bounds
      mm = 0
c count how many particles are going in each direction in ncl
```

```
c save their address and destination in ihole
c use periodic boundary conditions and check for roundoff error
c mm = direction particle is going
      if (dx.ge.edgerx) then
         if (dx.ge.anx) dx = dx - anx
         mm = 2
      else if (dx.lt.edgelx) then
         if (dx.lt.0.0) then
            dx = dx + anx
            if (dx.lt.anx) then
               mm = 1
            else
               dx = 0.0
            endif
         else
            mm = 1
         endif
      endif
      if (dy.ge.edgery) then
         if (dy.ge.any) dy = dy - any
         mm = mm + 6
      else if (dy.lt.edgely) then
         if (dy.lt.0.0) then
            dy = dy + any
            if (dy.lt.any) then
               mm = mm + 3
            else
               dy = 0.0
            endif
         else
            mm = mm + 3
         endif
      endif
c set new position
      ppart(j,1,k) = dx
      ppart(j,2,k) = dy
c set new velocity
      ppart(j,3,k) = vx
      ppart(j,4,k) = vy
c increment counters
      if (mm.gt.0) then
         ncl(mm,k) = ncl(mm,k) + 1
         ih = ih + 1
         if (ih.le.ntmax) then
            ihole(1,ih+1,k) = j
            ihole(2,ih+1,k) = mm
         else
            nh = 1
         endif
      endif
  100 continue
      sum2 = sum2 + sum1
c set error and end of file flag
c ihole overflow
```

```
      if (nh.gt.0) then
         irc = ih
         ih = -ih
      endif
      ihole(1,1,k) = ih
  110 continue
!$OMP END PARALLEL DO
c normalize kinetic energy
      ek = ek + 0.125*sum2
      return
      end
```

```
c-----------------------------------------------------------------------
      subroutine VGPPOST2LT(ppart,q,kpic,qm,nppmx,idimp,mx,my,nxv,nyv,
     1mx1,mxy1)
c for 2d code, this subroutine calculates particle charge density
c using first-order linear interpolation, periodic boundaries
c vectorizable/OpenMP version using guard cells
c data deposited in tiles
c particles stored segmented array
c 17 flops/particle, 6 loads, 4 stores
c input: all, output: q
c charge density is approximated by values at the nearest grid points
c q(n,m)=qm*(1.-dx)*(1.-dy)
c q(n+1,m)=qm*dx*(1.-dy)
c q(n,m+1)=qm*(1.-dx)*dy
c q(n+1,m+1)=qm*dx*dy
c where n,m = leftmost grid points and dx = x-n, dy = y-m
c ppart(n,1,m) = position x of particle n in tile m
c ppart(n,2,m) = position y of particle n in tile m
c q(j,k) = charge density at grid point j,k
c kpic = number of particles per tile
c qm = charge on particle, in units of e
c nppmx = maximum number of particles in tile
c idimp = size of phase space = 4
c mx/my = number of grids in sorting cell in x/y
c nxv = first dimension of charge array, must be >= nx+1
c nyv = second dimension of charge array, must be >= ny+1
c mx1 = (system length in x direction - 1)/mx + 1
c mxy1 = mx1*my1, where my1 = (system length in y direction - 1)/my + 1
      implicit none
      integer nppmx, idimp, mx, my, nxv, nyv, mx1, mxy1
      real qm
      real ppart, q
      integer kpic
      dimension ppart(nppmx,idimp,mxy1), q(nxv*nyv)
      dimension kpic(mxy1)
c local data
      integer MXV, MYV
      parameter(MXV=33,MYV=33)
      integer npblk, lvect
      parameter(npblk=32,lvect=4)
      integer noff, moff, npp, ipp, joff, nps
      integer i, j, k, m, nn, mm, lxv
      real x, y, dxp, dyp, amx, amy
      real sq
c     dimension sq(MXV*MYV)
      dimension sq((mx+1)*(my+1))
c scratch arrays
      integer n
      real s
      dimension n(npblk), s(npblk,lvect)
      lxv = mx + 1
c error if local array is too small
c     if ((mx.ge.MXV).or.(my.ge.MYV)) return
c loop over tiles
```

```
!$OMP PARALLEL DO
!$OMP& PRIVATE(i,j,k,m,noff,moff,npp,ipp,joff,nps,nn,mm,x,y,dxp,dyp,amx,
!$OMP& amy,sq,n,s)
      do 110 k = 1, mxy1
      noff = (k - 1)/mx1
      moff = my*noff
      noff = mx*(k - mx1*noff - 1)
      npp = kpic(k)
c zero out local accumulator
      do 10 j = 1, (mx+1)*(my+1)
      sq(j) = 0.0
   10 continue
c loop over particles in tile
      ipp = npp/npblk
c outer loop over number of full blocks
      do 50 m = 1, ipp
      joff = npblk*(m - 1)
c inner loop over particles in block
      do 20 j = 1, npblk
c find interpolation weights
      x = ppart(j+joff,1,k)
      y = ppart(j+joff,2,k)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
      dyp = y - real(mm)
      n(j) = nn - noff + lxv*(mm - moff)
      amx = qm - dxp
      amy = 1.0 - dyp
      s(j,1) = amx*amy
      s(j,2) = dxp*amy
      s(j,3) = amx*dyp
      s(j,4) = dxp*dyp
   20 continue
c deposit charge within tile to local accumulator
      do 40 j = 1, npblk
      nn = n(j)
      mm = nn + lxv - 2
!dir$ ivdep
      do 30 i = 1, lvect
      if (i.gt.2) nn = mm
      sq(i+nn) = sq(i+nn) + s(j,i)
   30 continue
   40 continue
   50 continue
      nps = npblk*ipp + 1
c loop over remaining particles
      do 60 j = nps, npp
c find interpolation weights
      x = ppart(j,1,k)
      y = ppart(j,2,k)
      nn = x
      mm = y
      dxp = qm*(x - real(nn))
      dyp = y - real(mm)
      nn = nn - noff + 1 + lxv*(mm - moff)
      amx = qm - dxp
      amy = 1.0 - dyp
c 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
      x = sq(nn+lxv) + amx*dyp
      y = sq(nn+1+lxv) + dxp*dyp
      sq(nn+lxv) = x
      sq(nn+1+lxv) = y
   60 continue
c deposit charge to interior points in global array
      nn = min(mx,nxv-noff)
      mm = min(my,nyv-moff)
      do 80 j = 2, mm
      do 70 i = 2, nn
      q(i+noff+nxv*(j+moff-1)) = q(i+noff+nxv*(j+moff-1)) +
     1sq(i+lxv*(j-1))
   70 continue
   80 continue
c deposit charge to edge points in global array
      mm = min(my+1,nyv-moff)
      do 90 i = 2, nn
!$OMP ATOMIC
      q(i+noff+nxv*moff) = q(i+noff+nxv*moff) + sq(i)
      if (mm > my) then
!$OMP ATOMIC
         q(i+noff+nxv*(mm+moff-1)) = q(i+noff+nxv*(mm+moff-1)) +
     1sq(i+lxv*(mm-1))
      endif
   90 continue
      nn = min(mx+1,nxv-noff)
      do 100 j = 1, mm
!$OMP ATOMIC
      q(1+noff+nxv*(j+moff-1)) = q(1+noff+nxv*(j+moff-1)) +
     1sq(1+lxv*(j-1))
      if (nn > mx) then
!$OMP ATOMIC
         q(nn+noff+nxv*(j+moff-1)) = q(nn+noff+nxv*(j+moff-1)) +
     1sq(nn+lxv*(j-1))
      endif
  100 continue
  110 continue
!$OMP END PARALLEL DO
      return
      end
```