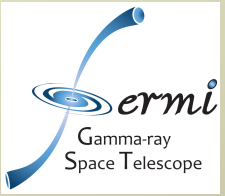
Computational Methods for Kinetic Processes in Plasma Physics



Ken Nishikawa

Department of Physics/UAH

Main program 3

June 4, 2015

Main program (continued)

```
c  ***********************************************************************
c  this subroutine is used in 3D version to sort particles that leave a processor
c  domain boundaries in all three directions; thus "iparL" stands for number
c  of particles leaving Left, Front, and Bottom boundary,
c  and "iparR" for Right, Rear, and Top boundary --> subroutine parameters
c  (identification of x,y,z and CRx,CRy,CRz etc. and boundaries) must be changed
c  accordingly upon calling
c  this method saves number of particle communication buffers (26-->12)
      subroutine Particle_sorting(iparR,iparL,ipar,mpass,mh,PBDL,PBDR,
     &                  CRx,CRy,CRz,CRu,CRv,CRw,
     &                  CLx,CLy,CLz,CLu,CLv,CLw,
     &                        x,y,z,u,v,w)

      dimension CRx(mpass),CRy(mpass),CRz(mpass)
      dimension CRu(mpass),CRv(mpass),CRw(mpass)
      dimension CLx(mpass),CLy(mpass),CLz(mpass)
      dimension CLu(mpass),CLv(mpass),CLw(mpass)
```

```
      dimension x(mh),y(mh),z(mh)
      dimension u(mh),v(mh),w(mh)

       iparR=0
       iparL=0

            if(ipar.eq.0)return

            k=0
130         k=k+1
            if (x(k).ge.PBDR) then
             iparR=iparR+1
             CRx(iparR)=x(k)
             CRy(iparR)=y(k)
             CRz(iparR)=z(k)

             CRu(iparR)=u(k)
             CRv(iparR)=v(k)
             CRw(iparR)=w(k)
```

```
c  replaced with particle from the top of the stock
                x(k)=x(ipar)
                y(k)=y(ipar)
                z(k)=z(ipar)

                u(k)=u(ipar)
                v(k)=v(ipar)
                w(k)=w(ipar)
                ipar=ipar-1
                k=k-1
                if (k.lt.ipar) goto 130
               end if

        if (k.eq.0) return
               if (x(k).lt.PBDL) then
                iparL=iparL+1
                CLx(iparL)=x(k)
                CLy(iparL)=y(k)
                CLz(iparL)=z(k)
```

```
            CLu(iparL)=u(k)
            CLv(iparL)=v(k)
            CLw(iparL)=w(k)

            x(k)=x(ipar)
            y(k)=y(ipar)
            z(k)=z(ipar)

            u(k)=u(ipar)
            v(k)=v(ipar)
            w(k)=w(ipar)
            ipar=ipar-1
            k=k-1
           end if

           if (k.lt.ipar) goto 130

c!!     print *, ipar,iparL,iparL,cRx(1),CLz(2)

      return
      end
```

```
c  ***************************************************************
c  clears E-field in ghost cells
c ** contributions to the electric field in ghost cells from the subsequent **
c ** current deposition ("split" subroutines) are passed and added to      **
c ** the corresponding E-field elements at neighboring processors          **
c ** ("E_Field_Passing_Add" subroutine)                                    **
      subroutine Clear_ghost(ex,ey,ez,mFx,mFy,mFz,dims,coords,
     &                 FBDRx,FBDRy,FBDRz,FBDLx,FBDLy,FBDLz)

      integer dims(3),coords(3)
      integer FBDRx,FBDRy,FBDRz
      integer FBDLx,FBDLy,FBDLz

      dimension ex(mFx,mFy,mFz),ey(mFx,mFy,mFz),ez(mFx,mFy,mFz)

c  all ghost cells (except as below) are cleared, since particles at the
c  boundaries deposit currents to all guard cells
```

```
c  clear Left and Right ghost cells (except in leftmost and rightmost domains)
      if(coords(1).ne.0)then
       do k = 1,mFz
        do j = 1,mFy
                do i = FBDLx-2,FBDLx-1
                 ex(i,j,k)=0.0
                 ey(i,j,k)=0.0
                 ez(i,j,k)=0.0
          end do
               end do
       end do
      end if

      if(coords(1).ne.(dims(1)-1))then
       do k = 1,mFz
        do j = 1,mFy
                do i = FBDRx+1,FBDRx+3
                 ex(i,j,k)=0.0
                 ey(i,j,k)=0.0
                 ez(i,j,k)=0.0
```

```
            end do
           end do
       end do
      end if

c  clear Front and Rear ghost cells
      do k = 1,mFz
       do i = 1,mFx
              do j = FBDLy-2,FBDLy-1
               ex(i,j,k)=0.0
               ey(i,j,k)=0.0
               ez(i,j,k)=0.0
        end do
       end do
      end do
```

```
      do k = 1,mFz
      do i = 1,mFx
            do j = FBDRy+1,FBDRy+3
             ex(i,j,k)=0.0
             ey(i,j,k)=0.0
             ez(i,j,k)=0.0
       end do
      end do
     end do

c  clear Bottom and Top ghost cells
     do j = 1,mFy
      do i = 1,mFx
            do k = FBDLz-2,FBDLz-1
             ex(i,j,k)=0.0
             ey(i,j,k)=0.0
             ez(i,j,k)=0.0
       end do
      end do
     end do
```

```
do j = 1,mFy
 do i = 1,mFx
         do k = FBDRz+1,FBDRz+3
          ex(i,j,k)=0.0
          ey(i,j,k)=0.0
          ez(i,j,k)=0.0
  end do
 end do
end do

return
end
```

```
c  ***********************************************************
      subroutine Split_JET(ipar,x,y,z,u,v,w,mh,dex,dey,dez,
     &                  mFx,mFy,mFz,my,mz,q,DHDx,DHDy,DHDz,dt)

      dimension dex(mFx,mFy,mFz),dey(mFx,mFy,mFz),dez(mFx,mFy,mFz)
      dimension x(mh),y(mh),z(mh)
      dimension u(mh),v(mh),w(mh)

      k=0
50    k=k+1
c  previous particle position
      x0=x(k)-u(k)*dt
      y0=y(k)-v(k)*dt
      z0=z(k)-w(k)*dt

c  make particles which are out of the box (3,my-2)*(3,mz-2)
c  periodic in transverse dimensions with period my-5 (mz-5)
c ** in 3D version particles that were out of the virtual box transverse    **
c ** boundaries have been already passed to processes according to the      **
c ** periodic boundary conditions; here only their proper y and z-positions **
c ** in the new domain are calculated (e.g. y(k)=83.5 --> y(k)=3.5 (my=85)) **
```

```
cJET only periodicity applied
      per = sign(.5*(my-5.),y(k)-3.) + sign(.5*(my-5.),y(k)-my+2.)
      y(k) = y(k)-per
      y0   = y0  -per

      per = sign(.5*(mz-5.),z(k)-3.) + sign(.5*(mz-5.),z(k)-mz+2.)
      z(k) = z(k)-per
      z0   = z0  -per

cU1      call depsitUM2(x(k),y(k),z(k),x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,
cU1     &                                        q,DHDx,DHDy,DHDz)
      call depsitUM1(x(k),y(k),z(k),x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,
     &                                       q,DHDx,DHDy,DHDz)

53    if (k.lt.ipar) goto 50

      return
      end
```

```
c ***********************************************************
      subroutine Split_Ambient(ipar,x,y,z,u,v,w,mh,dex,dey,dez,
     &                mFx,mFy,mFz,mx,my,mz,q,DHDx,DHDy,DHDz,
     &              PVLeft,PVRght,PBFrnt,PBRear,PBBot,PBTop,
     &                    vthml,c,dt,refl,eloss,mass,is)

      real mass

      dimension dex(mFx,mFy,mFz),dey(mFx,mFy,mFz),dez(mFx,mFy,mFz)
      dimension x(mh),y(mh),z(mh)
      dimension u(mh),v(mh),w(mh)

      pi  = 4*atan(1.0d0)

c  variance of Maxwell distribution for individual velocity components
      sig = vthml/sqrt(2.)

      k=0
50    k=k+1
```

```
c  previous particle position needed for current deposition
      x0=x(k)-u(k)*dt
      y0=y(k)-v(k)*dt
      z0=z(k)-w(k)*dt

c4push
      xini=x(k)
      reflect=0.0

c  particles outside the box (3,mx-2) are reflected or eliminated from the
c  simulations
c  !!! now the boundaries are (PVLeft,PVRght) !!!
c ** only coords(1)=0 and coords(1)=Npx-1 processors calculate this part **
c      if (x(k).ge.3. .and. x(k).lt.mx-2.) goto 51
      if (x(k).ge.PVLeft .and. x(k).lt.PVRght) goto 51

c  which boundary has been crossed?
c       x(k)=amin1(amax1(x(k),3.),mx-2.000001)
       x(k)=amin1(amax1(x(k),PVLeft),PVRght-0.000001)
c  time at boundary crossing
       tx=(x(k)-x0)/u(k)
```

```
c  y,z at boundary crossing
c ** particles which are not reflected are stopped at the boundary at this **
c ** point and deposit current at this x-position                          **
      y(k)=y0+tx*v(k)
      z(k)=z0+tx*w(k)

c4push
      reflect=refl

c ** particle reflection according to the reflection rate "refl" **
c ** e.g., refl=0.8 -> ~20% particles reflected                    **
      rabs=ran1(is)

c  particles are reflected from stiff-wall x-boundary
c  now particles reflected have new set of inward velocities, as if a new
c  particle entered the box in place of the one which left the box
c ** this can cause numerical errors in a rare case in which a reflected     **
c ** particle crosses at the same time one of the other boundaries (y or z); **
c ** such a particle has been already passed to a new processor in          **
c ** "particle_passing" and its new location after a move with new velocity  **
```

```
c ** components must fit the domain boundaries after the periodic conditions **
c ** have been applied; because of that a loop in label "20" is necessary -  **
c ** this is a change compared with older version of this subroutine: note   **
c ** that the problem does not exist when reflection from stiff-wall boundary**
c ** is applied                                                              **
c new Jacek
      ntry = 0
      u0=u(k)
      v0=v(k)
      w0=w(k)

      if(rabs.gt.reflect) then
c       u(k)=-u(k)
c       ut=u(k)
       ut=u0
20      yt=y(k)
       zt=z(k)
       y0t=y0
       z0t=z0

       ntry = ntry + 1
```

```
c new Jacek
c ** stiff-wall boundary reflection is applied if loop 20 does not succeed after **
c ** 1000 trials; this in particular takes care of cases in which a particle     **
c ** crosses domain x-boundaries at a very oblique angle, so that y or z position**
c ** at boundary crossing is too much outside the domain boundaries and a       **
c ** selection of loop 20 must tune up to minus original velocity to place      **
c ** particle in right domain                                                   **
c4push       if (ntry.ge.1000) then
c4push  reflection method with new velocity components selection turned off
c4push  here because it lead to nonphysical accumulation of particles near
c4push  boundaries, stiff-wall reflection from left boundary and particle
c4push  leaking out of right boundary applied
       if (ntry.ge.1) then
        u(k)=-u0
        v(k)= v0
        w(k)= w0
        goto 21
       end if
```

```
18      r11 = ran1(is)
        if(r11.EQ.1.0) goto 18

        r1 = sqrt(-2.0*log(1.0-r11))
        if(sig*r1.ge.c) goto 18
        r2 = 2.0*pi*ran1(is)
        unew = sig*r1*cos(r2)
        vnew = sig*r1*sin(r2)

19      r33 = ran1(is)
        if(r33.EQ.1.0) goto 19

        r3 = sqrt(-2.0*log(1.0-r33))
        if(sig*r3.ge.c) goto 19
        r4 = 2.0*pi*ran1(is)
        wnew = sig*r3*cos(r4)

        u(k) = -sign(unew,ut)
        v(k) = vnew
        w(k) = wnew
```

```
21      yt=y(k)+(dt-tx)*v(k)
        zt=z(k)+(dt-tx)*w(k)

c apply the periodicity here and check the domain boundaries
        per = sign(.5*(my-5.),yt-3.) + sign(.5*(my-5.),yt-my+2.)
        yt = yt-per
        y0t= y0t -per

        per = sign(.5*(mz-5.),zt-3.) + sign(.5*(mz-5.),zt-mz+2.)
        zt = zt-per
        z0t= z0t -per

        if((yt.lt.PBFrnt) .or. (yt.ge.PBRear) .or.
     &     (zt.lt.PBBot ) .or. (zt.ge.PBTop ) .or.
     &     (u(k)**2+v(k)**2+w(k)**2 .ge. c**2)) goto 20

         x(k)=x(k)+(dt-tx)*u(k)
         y(k)=yt
         z(k)=zt
         y0=y0t
         z0=z0t
```

```
       endif
       go to 52

51     continue
      rabs=2.0
c52    continue

c  make particles which are out of the box (3,my-2)*(3,mz-2)
c  periodic in transversal dimensions with period my-5 (mz-5)
c ** in 3D version particles that are out of the virtual box transversal    **
c ** boundaries are passed to processes according to the periodic boundary  **
c ** conditions; here only their proper y and z-positions in the new domain **
c ** are calculated                                                        **
      per = sign(.5*(my-5.),y(k)-3.) + sign(.5*(my-5.),y(k)-my+2.)
      y(k) = y(k)-per
      y0   = y0  -per

      per = sign(.5*(mz-5.),z(k)-3.) + sign(.5*(mz-5.),z(k)-mz+2.)
      z(k) = z(k)-per
      z0   = z0  -per
```

```
c  split particles which cross cell boundaries and deposit currents
c ** Umeda's 2nd-order method **
cU1      call depsitUM2(x(k),y(k),z(k),x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,
cU1     &                                          q,DHDx,DHDy,DHDz)
52    call depsitUM1(x(k),y(k),z(k),x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,
     &                                          q,DHDx,DHDy,DHDz)



c  particles (in left or right processors) which are outside virtual box
c  x-boundaries (non-reflected particles) are eliminated and replaced by
c  particles from the top of the stack
c4push      if (rabs.ge.refl) goto 53
      if (rabs.gt.reflect .and. xini.ge.PVLeft) goto 53
c  checking kinetic energy lost
      eloss=eloss+0.5*mass*(u(k)**2+v(k)**2+w(k)**2)
```

```
      x(k)=x(ipar)
      y(k)=y(ipar)
      z(k)=z(ipar)
      u(k)=u(ipar)
      v(k)=v(ipar)
      w(k)=w(ipar)
      ipar=ipar-1
      k=k-1

53    if (k.lt.ipar) goto 50

      return
      end
```

```
c ***********************************************************
      subroutine xsplit(x,y,z,x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                  DHDx,DHDy,DHDz)

      dimension dex(mFx,mFy,mFz),dey(mFx,mFy,mFz),dez(mFx,mFy,mFz)
      dimension sm(-1:1,-1:1,-1:1)

      if(ifix(x).ne.ifix(x0).and.((x-x0).ne.0.))go to 1
      call ysplit(x,y,z,x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                  DHDx,DHDy,DHDz)
      return
1     x1=.5*(1+ifix(x)+ifix(x0))
      y1=y0+(y-y0)*((x1-x0)/(x-x0))
      z1=z0+(z-z0)*((x1-x0)/(x-x0))
      call ysplit(x,y,z,x1,y1,z1,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                  DHDx,DHDy,DHDz)
      call ysplit(x1,y1,z1,x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                  DHDx,DHDy,DHDz)
      return
      end
```

```
c ***********************************************************
      subroutine ysplit(x,y,z,x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                  DHDx,DHDy,DHDz)

      dimension dex(mFx,mFy,mFz),dey(mFx,mFy,mFz),dez(mFx,mFy,mFz)
      dimension sm(-1:1,-1:1,-1:1)

      if(ifix(y).ne.ifix(y0).and.((y-y0).ne.0.))go to 1
      call zsplit(x,y,z,x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                              DHDx,DHDy,DHDz)
      return
1     y1=.5*(1+ifix(y)+ifix(y0))
      z1=z0+(z-z0)*((y1-y0)/(y-y0))
      x1=x0+(x-x0)*((y1-y0)/(y-y0))
      call zsplit(x,y,z,x1,y1,z1,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                              DHDx,DHDy,DHDz)
      call zsplit(x1,y1,z1,x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                DHDx,DHDy,DHDz)
      return
      end
```

```
c ***********************************************************
      subroutine zsplit(x,y,z,x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                  DHDx,DHDy,DHDz)

      dimension dex(mFx,mFy,mFz),dey(mFx,mFy,mFz),dez(mFx,mFy,mFz)
      dimension sm(-1:1,-1:1,-1:1)

      if(ifix(z).ne.ifix(z0).and.((z-z0).ne.0.))go to 1
      call depsit(x,y,z,x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                  DHDx,DHDy,DHDz)
      return
1     z1=.5*(1+ifix(z)+ifix(z0))
      x1=x0+(x-x0)*((z1-z0)/(z-z0))
      y1=y0+(y-y0)*((z1-z0)/(z-z0))
      call depsit(x,y,z,x1,y1,z1,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                  DHDx,DHDy,DHDz)
      call depsit(x1,y1,z1,x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                  DHDx,DHDy,DHDz)
      return
      end
```

```
c ***********************************************************
      subroutine depsit(x,y,z,x0,y0,z0,dex,dey,dez,mFx,mFy,mFz,q,sm,
     &                                  DHDx,DHDy,DHDz)

      dimension dex(mFx,mFy,mFz),dey(mFx,mFy,mFz),dez(mFx,mFy,mFz)
      dimension sm(-1:1,-1:1,-1:1)

c  cell indices of half-way point:
      i=.5*(x+x0) - DHDx
      j=.5*(y+y0) - DHDy
      k=.5*(z+z0) - DHDz
c  displacements in cell of half-way point:
      dx=.5*(x+x0) - i - DHDx
      dy=.5*(y+y0) - j - DHDy
      dz=.5*(z+z0) - k - DHDz
c  current elements:
      qu=q*(x-x0)
      qv=q*(y-y0)
      qw=q*(z-z0)
      delt=.08333333*qu*(y-y0)*(z-z0)
```

```
c  *** DEPOSIT CURRENT and SMOOTHING ***
c  If one desires NO smoothing (risking the presence of alias-prone
c  high harmonics), one can replace the loops by taking nx=ny=nz=0
c  and boost the value of  q  by a factor 8.
c      do nz = -1,1
c       do ny = -1,1
c        do nx = -1,1
c         sd = sm(nx,ny,nz)*delt
c              su = sm(nx,ny,nz)*qu

c !!! NO SMOOTHING IS APPLIED HERE !!!
              nx = 0
              ny = 0
              nz = 0
              sd = delt
              su = qu
              dex(i+nx,j+1+ny,k+1+nz)= dex(i+nx,j+1+ny,k+1+nz)
     &                         -su*dy*dz         -sd
        dex(i+nx,j+ny,k+1+nz)  = dex(i+nx,j+ny,k+1+nz)
     &                         -su*(1.-dy)*dz     +sd
```

```
      dex(i+nx,j+1+ny,k+nz)  = dex(i+nx,j+1+ny,k+nz)
     &                        -su*dy*(1.-dz)     +sd
      dex(i+nx,j+ny,k+nz)    = dex(i+nx,j+ny,k+nz)
     &                        -su*(1.-dy)*(1.-dz)-sd

c       sv = sm(nx,ny,nz)*qv
            sv = qv
      dey(i+1+nx,j+ny,k+1+nz)= dey(i+1+nx,j+ny,k+1+nz)
     &                  -sv*dz*dx         -sd
      dey(i+1+nx,j+ny,k+nz)  = dey(i+1+nx,j+ny,k+nz)
     &                  -sv*(1.-dz)*dx     +sd
      dey(i+nx,j+ny,k+1+nz)  = dey(i+nx,j+ny,k+1+nz)
     &                  -sv*dz*(1.-dx)     +sd
      dey(i+nx,j+ny,k+nz)    = dey(i+nx,j+ny,k+nz)
     &                  -sv*(1.-dz)*(1.-dx)-sd

c       sw = sm(nx,ny,nz)*qw
            sw = qw
      dez(i+1+nx,j+1+ny,k+nz)= dez(i+1+nx,j+1+ny,k+nz)
     &                  -sw*dx*dy         -sd
```

```
      dez(i+nx,j+1+ny,k+nz)  = dez(i+nx,j+1+ny,k+nz)
     &                    -sw*(1.-dx)*dy     +sd
      dez(i+1+nx,j+ny,k+nz)  = dez(i+1+nx,j+ny,k+nz)
     &                         -sw*dx*(1.-dy)     +sd
      dez(i+nx,j+ny,k+nz)    = dez(i+nx,j+ny,k+nz)
     &                         -sw*(1.-dx)*(1.-dy)-sd
c      end do
c     end do
c    end do

     return
     end
```

```
c ***********************************************************
      subroutine depsitUM1(x2,y2,z2,x1,y1,z1,dex,dey,dez,mFx,mFy,mFz,
     &                                   q,DHDx,DHDy,DHDz)

      dimension dex(mFx,mFy,mFz),dey(mFx,mFy,mFz),dez(mFx,mFy,mFz)


      i1=x1 - DHDx
      j1=y1 - DHDy
      k1=z1 - DHDz

      i2=x2 - DHDx
      j2=y2 - DHDy
      k2=z2 - DHDz

      xr = min(min(i1*1.0,i2*1.0)+1.0,max(max(i1*1.0,i2*1.0),
     &                          0.5*(x1+x2)-DHDx))
      yr = min(min(j1*1.0,j2*1.0)+1.0,max(max(j1*1.0,j2*1.0),
     &                          0.5*(y1+y2)-DHDy))
      zr = min(min(k1*1.0,k2*1.0)+1.0,max(max(k1*1.0,k2*1.0),
     &                          0.5*(z1+z2)-DHDz))
```

```
c      if (i1.eq.i2) then
c       xr = 0.5*(x1+x2)-DHDx
c      else
c       xr = max(i1,i2)
c      end if

c      if (j1.eq.j2) then
c       yr = 0.5*(y1+y2)-DHDy
c      else
c       yr = max(j1,j2)
c      end if

c      if (k1.eq.k2) then
c       zr = 0.5*(z1+z2)-DHDz
c      else
c       zr = max(k1,k2)
c      end if

      qu1=q*(xr-x1+DHDx)
      qv1=q*(yr-y1+DHDy)
      qw1=q*(zr-z1+DHDz)
```

```
      qu2=q*(x2-xr-DHDx)
      qv2=q*(y2-yr-DHDy)
      qw2=q*(z2-zr-DHDz)

      dx1=0.5*(x1+xr-DHDx) - i1
      dy1=0.5*(y1+yr-DHDy) - j1
      dz1=0.5*(z1+zr-DHDz) - k1

      dx2=0.5*(x2+xr-DHDx) - i2
      dy2=0.5*(y2+yr-DHDy) - j2
      dz2=0.5*(z2+zr-DHDz) - k2

c  *** DEPOSIT CURRENT and SMOOTHING ***

c !!! NO SMOOTHING IS APPLIED HERE !!!

       dex(i1,j1+1,k1+1)= dex(i1,j1+1,k1+1) - qu1*dy1*dz1
       dex(i1,j1,k1+1)  = dex(i1,j1,k1+1)   - qu1*(1.-dy1)*dz1
       dex(i1,j1+1,k1)  = dex(i1,j1+1,k1)   - qu1*dy1*(1.-dz1)
       dex(i1,j1,k1)    = dex(i1,j1,k1)     - qu1*(1.-dy1)*(1.-dz1)
```

```
dey(i1+1,j1,k1+1)= dey(i1+1,j1,k1+1) - qv1*dx1*dz1
dey(i1+1,j1,k1)  = dey(i1+1,j1,k1)   - qv1*dx1*(1.-dz1)
dey(i1,j1,k1+1)  = dey(i1,j1,k1+1)   - qv1*(1.-dx1)*dz1
dey(i1,j1,k1)    = dey(i1,j1,k1)     - qv1*(1.-dx1)*(1.-dz1)

dez(i1+1,j1+1,k1)= dez(i1+1,j1+1,k1) - qw1*dx1*dy1
dez(i1,j1+1,k1)  = dez(i1,j1+1,k1)   - qw1*(1.-dx1)*dy1
dez(i1+1,j1,k1)  = dez(i1+1,j1,k1)   - qw1*dx1*(1.-dy1)
dez(i1,j1,k1)    = dez(i1,j1,k1)     - qw1*(1.-dx1)*(1.-dy1)


dex(i2,j2+1,k2+1)= dex(i2,j2+1,k2+1) - qu2*dy2*dz2
dex(i2,j2,k2+1)  = dex(i2,j2,k2+1)   - qu2*(1.-dy2)*dz2
dex(i2,j2+1,k2)  = dex(i2,j2+1,k2)   - qu2*dy2*(1.-dz2)
dex(i2,j2,k2)    = dex(i2,j2,k2)     - qu2*(1.-dy2)*(1.-dz2)

dey(i2+1,j2,k2+1)= dey(i2+1,j2,k2+1) - qv2*dx2*dz2
dey(i2+1,j2,k2)  = dey(i2+1,j2,k2)   - qv2*dx2*(1.-dz2)
dey(i2,j2,k2+1)  = dey(i2,j2,k2+1)   - qv2*(1.-dx2)*dz2
dey(i2,j2,k2)    = dey(i2,j2,k2)     - qv2*(1.-dx2)*(1.-dz2)
```

```
      dez(i2+1,j2+1,k2)= dez(i2+1,j2+1,k2) - qw2*dx2*dy2
      dez(i2,j2+1,k2)  = dez(i2,j2+1,k2)   - qw2*(1.-dx2)*dy2
      dez(i2+1,j2,k2)  = dez(i2+1,j2,k2)   - qw2*dx2*(1.-dy2)
      dez(i2,j2,k2)    = dez(i2,j2,k2)     - qw2*(1.-dx2)*(1.-dy2)

      return
      end
c *****************************************************************
      subroutine depsitUM2(x2,y2,z2,x1,y1,z1,dex,dey,dez,mFx,mFy,mFz,
     &                                  q,DHDx,DHDy,DHDz)

      dimension dex(mFx,mFy,mFz),dey(mFx,mFy,mFz),dez(mFx,mFy,mFz)

      dimension wi1(-1:1),wj1(-1:1),wk1(-1:1)
      dimension wi2(-1:1),wj2(-1:1),wk2(-1:1)

      i1=nint(x1) - DHDx
      j1=nint(y1) - DHDy
      k1=nint(z1) - DHDz
```

```
i2=nint(x2) - DHDx
j2=nint(y2) - DHDy
k2=nint(z2) - DHDz

if (i1.eq.i2) then
 xr = 0.5*(x1+x2)-DHDx
else
 xr = 0.5*(i1+i2)
end if

if (j1.eq.j2) then
 yr = 0.5*(y1+y2)-DHDy
else
 yr = 0.5*(j1+j2)
end if

if (k1.eq.k2) then
 zr = 0.5*(z1+z2)-DHDz
else
 zr = 0.5*(k1+k2)
end if
```

qu1=q*(xr-x1+DHDx)
qv1=q*(yr-y1+DHDy)
qw1=q*(zr-z1+DHDz)

qu2=q*(x2-xr-DHDx)
qv2=q*(y2-yr-DHDy)
qw2=q*(z2-zr-DHDz)

dx1=0.5*(x1+xr-DHDx) - i1
dy1=0.5*(y1+yr-DHDy) - j1
dz1=0.5*(z1+zr-DHDz) - k1

dx2=0.5*(x2+xr-DHDx) - i2
dy2=0.5*(y2+yr-DHDy) - j2
dz2=0.5*(z2+zr-DHDz) - k2

wi1(-1)= 0.5*(0.5-dx1)*(0.5-dx1)
wi1(0) = 0.75-dx1*dx1
wi1(1) = 0.5*(0.5+dx1)*(0.5+dx1)

wj1(-1)= 0.5*(0.5-dy1)*(0.5-dy1)
wj1(0) = 0.75-dy1*dy1
wj1(1) = 0.5*(0.5+dy1)*(0.5+dy1)

wk1(-1)= 0.5*(0.5-dz1)*(0.5-dz1)
wk1(0) = 0.75-dz1*dz1
wk1(1) = 0.5*(0.5+dz1)*(0.5+dz1)

wi2(-1)= 0.5*(0.5-dx2)*(0.5-dx2)
wi2(0) = 0.75-dx2*dx2
wi2(1) = 0.5*(0.5+dx2)*(0.5+dx2)

wj2(-1)= 0.5*(0.5-dy2)*(0.5-dy2)
wj2(0) = 0.75-dy2*dy2
wj2(1) = 0.5*(0.5+dy2)*(0.5+dy2)

wk2(-1)= 0.5*(0.5-dz2)*(0.5-dz2)
wk2(0) = 0.75-dz2*dz2
wk2(1) = 0.5*(0.5+dz2)*(0.5+dz2)

```
qu1m = qu1*(0.5-dx1)
qu1p = qu1*(0.5+dx1)
qv1m = qv1*(0.5-dy1)
qv1p = qv1*(0.5+dy1)
qw1m = qw1*(0.5-dz1)
qw1p = qw1*(0.5+dz1)

qu2m = qu2*(0.5-dx2)
qu2p = qu2*(0.5+dx2)
qv2m = qv2*(0.5-dy2)
qv2p = qv2*(0.5+dy2)
qw2m = qw2*(0.5-dz2)
qw2p = qw2*(0.5+dz2)


c  *** DEPOSIT CURRENT and SMOOTHING ***

c !!! NO SMOOTHING IS APPLIED HERE !!!

      dex(i1-1,j1-1,k1-1)= dex(i1-1,j1-1,k1-1) -qu1m*wj1(-1)*wk1(-1)
      dex(i1-1,j1,k1-1)  = dex(i1-1,j1,k1-1)   -qu1m*wj1(0)*wk1(-1)
```

```
dex(i1-1,j1+1,k1-1)= dex(i1-1,j1+1,k1-1) -qu1m*wj1(1)*wk1(-1)
dex(i1-1,j1-1,k1)  = dex(i1-1,j1-1,k1)   -qu1m*wj1(-1)*wk1(0)
dex(i1-1,j1,k1)    = dex(i1-1,j1,k1)     -qu1m*wj1(0)*wk1(0)
dex(i1-1,j1+1,k1)  = dex(i1-1,j1+1,k1)   -qu1m*wj1(1)*wk1(0)
dex(i1-1,j1-1,k1+1)= dex(i1-1,j1-1,k1+1) -qu1m*wj1(-1)*wk1(1)
dex(i1-1,j1,k1+1)  = dex(i1-1,j1,k1+1)   -qu1m*wj1(0)*wk1(1)
dex(i1-1,j1+1,k1+1)= dex(i1-1,j1+1,k1+1) -qu1m*wj1(1)*wk1(1)

dex(i1,j1-1,k1-1)= dex(i1,j1-1,k1-1) -qu1p*wj1(-1)*wk1(-1)
dex(i1,j1,k1-1)  = dex(i1,j1,k1-1)   -qu1p*wj1(0)*wk1(-1)
dex(i1,j1+1,k1-1)= dex(i1,j1+1,k1-1) -qu1p*wj1(1)*wk1(-1)
dex(i1,j1-1,k1)  = dex(i1,j1-1,k1)   -qu1p*wj1(-1)*wk1(0)
dex(i1,j1,k1)    = dex(i1,j1,k1)     -qu1p*wj1(0)*wk1(0)
dex(i1,j1+1,k1)  = dex(i1,j1+1,k1)   -qu1p*wj1(1)*wk1(0)
dex(i1,j1-1,k1+1)= dex(i1,j1-1,k1+1) -qu1p*wj1(-1)*wk1(1)
dex(i1,j1,k1+1)  = dex(i1,j1,k1+1)   -qu1p*wj1(0)*wk1(1)
dex(i1,j1+1,k1+1)= dex(i1,j1+1,k1+1) -qu1p*wj1(1)*wk1(1)

dey(i1-1,j1-1,k1-1)= dey(i1-1,j1-1,k1-1) -qv1m*wi1(-1)*wk1(-1)
dey(i1  ,j1-1,k1-1)= dey(i1  ,j1-1,k1-1) -qv1m*wi1(0)*wk1(-1)
dey(i1+1,j1-1,k1-1)= dey(i1+1,j1-1,k1-1) -qv1m*wi1(1)*wk1(-1)
```

```
dey(i1-1,j1-1,k1)  = dey(i1-1,j1-1,k1)   -qv1m*wi1(-1)*wk1(0)
dey(i1  ,j1-1,k1)  = dey(i1  ,j1-1,k1)   -qv1m*wi1(0)*wk1(0)
dey(i1+1,j1-1,k1)  = dey(i1+1,j1-1,k1)   -qv1m*wi1(1)*wk1(0)
dey(i1-1,j1-1,k1+1)= dey(i1-1,j1-1,k1+1) -qv1m*wi1(-1)*wk1(1)
dey(i1  ,j1-1,k1+1)= dey(i1  ,j1-1,k1+1) -qv1m*wi1(0)*wk1(1)
dey(i1+1,j1-1,k1+1)= dey(i1+1,j1-1,k1+1) -qv1m*wi1(1)*wk1(1)

dey(i1-1,j1,k1-1)= dey(i1-1,j1,k1-1) -qv1p*wi1(-1)*wk1(-1)
dey(i1  ,j1,k1-1)= dey(i1  ,j1,k1-1) -qv1p*wi1(0)*wk1(-1)
dey(i1+1,j1,k1-1)= dey(i1+1,j1,k1-1) -qv1p*wi1(1)*wk1(-1)
dey(i1-1,j1,k1)  = dey(i1-1,j1,k1)   -qv1p*wi1(-1)*wk1(0)
dey(i1  ,j1,k1)  = dey(i1  ,j1,k1)   -qv1p*wi1(0)*wk1(0)
dey(i1+1,j1,k1)  = dey(i1+1,j1,k1)   -qv1p*wi1(1)*wk1(0)
dey(i1-1,j1,k1+1)= dey(i1-1,j1,k1+1) -qv1p*wi1(-1)*wk1(1)
dey(i1  ,j1,k1+1)= dey(i1  ,j1,k1+1) -qv1p*wi1(0)*wk1(1)
dey(i1+1,j1,k1+1)= dey(i1+1,j1,k1+1) -qv1p*wi1(1)*wk1(1)

dez(i1-1,j1-1,k1-1)= dez(i1-1,j1-1,k1-1) -qw1m*wi1(-1)*wj1(-1)
dez(i1  ,j1-1,k1-1)= dez(i1  ,j1-1,k1-1) -qw1m*wi1(0)*wj1(-1)
dez(i1+1,j1-1,k1-1)= dez(i1+1,j1-1,k1-1) -qw1m*wi1(1)*wj1(-1)
```

dez(i1-1,j1 ,k1-1)= dez(i1-1,j1 ,k1-1) -qw1m*wi1(-1)*wj1(0)
dez(i1 ,j1 ,k1-1)= dez(i1 ,j1 ,k1-1) -qw1m*wi1(0)*wj1(0)
dez(i1+1,j1 ,k1-1)= dez(i1+1,j1 ,k1-1) -qw1m*wi1(1)*wj1(0)
dez(i1-1,j1+1,k1-1)= dez(i1-1,j1+1,k1-1) -qw1m*wi1(-1)*wj1(1)
dez(i1 ,j1+1,k1-1)= dez(i1 ,j1+1,k1-1) -qw1m*wi1(0)*wj1(1)
dez(i1+1,j1+1,k1-1)= dez(i1+1,j1+1,k1-1) -qw1m*wi1(1)*wj1(1)

dez(i1-1,j1-1,k1)= dez(i1-1,j1-1,k1) -qw1p*wi1(-1)*wj1(-1)
dez(i1 ,j1-1,k1)= dez(i1 ,j1-1,k1) -qw1p*wi1(0)*wj1(-1)
dez(i1+1,j1-1,k1)= dez(i1+1,j1-1,k1) -qw1p*wi1(1)*wj1(-1)
dez(i1-1,j1 ,k1)= dez(i1-1,j1 ,k1) -qw1p*wi1(-1)*wj1(0)
dez(i1 ,j1 ,k1)= dez(i1 ,j1 ,k1) -qw1p*wi1(0)*wj1(0)
dez(i1+1,j1 ,k1)= dez(i1+1,j1 ,k1) -qw1p*wi1(1)*wj1(0)
dez(i1-1,j1+1,k1)= dez(i1-1,j1+1,k1) -qw1p*wi1(-1)*wj1(1)
dez(i1 ,j1+1,k1)= dez(i1 ,j1+1,k1) -qw1p*wi1(0)*wj1(1)
dez(i1+1,j1+1,k1)= dez(i1+1,j1+1,k1) -qw1p*wi1(1)*wj1(1)

dex(i2-1,j2-1,k2-1)= dex(i2-1,j2-1,k2-1) -qu2m*wj2(-1)*wk2(-1)
dex(i2-1,j2,k2-1) = dex(i2-1,j2,k2-1)   -qu2m*wj2(0)*wk2(-1)
dex(i2-1,j2+1,k2-1)= dex(i2-1,j2+1,k2-1) -qu2m*wj2(1)*wk2(-1)

```
dex(i2-1,j2-1,k2)  = dex(i2-1,j2-1,k2)   -qu2m*wj2(-1)*wk2(0)
dex(i2-1,j2,k2)    = dex(i2-1,j2,k2)     -qu2m*wj2(0)*wk2(0)
dex(i2-1,j2+1,k2)  = dex(i2-1,j2+1,k2)   -qu2m*wj2(1)*wk2(0)
dex(i2-1,j2-1,k2+1)= dex(i2-1,j2-1,k2+1) -qu2m*wj2(-1)*wk2(1)
dex(i2-1,j2,k2+1)  = dex(i2-1,j2,k2+1)   -qu2m*wj2(0)*wk2(1)
dex(i2-1,j2+1,k2+1)= dex(i2-1,j2+1,k2+1) -qu2m*wj2(1)*wk2(1)

dex(i2,j2-1,k2-1)= dex(i2,j2-1,k2-1) -qu2p*wj2(-1)*wk2(-1)
dex(i2,j2,k2-1)  = dex(i2,j2,k2-1)   -qu2p*wj2(0)*wk2(-1)
dex(i2,j2+1,k2-1)= dex(i2,j2+1,k2-1) -qu2p*wj2(1)*wk2(-1)
dex(i2,j2-1,k2)  = dex(i2,j2-1,k2)   -qu2p*wj2(-1)*wk2(0)
dex(i2,j2,k2)    = dex(i2,j2,k2)     -qu2p*wj2(0)*wk2(0)
dex(i2,j2+1,k2)  = dex(i2,j2+1,k2)   -qu2p*wj2(1)*wk2(0)
dex(i2,j2-1,k2+1)= dex(i2,j2-1,k2+1) -qu2p*wj2(-1)*wk2(1)
dex(i2,j2,k2+1)  = dex(i2,j2,k2+1)   -qu2p*wj2(0)*wk2(1)
dex(i2,j2+1,k2+1)= dex(i2,j2+1,k2+1) -qu2p*wj2(1)*wk2(1)

dey(i2-1,j2-1,k2-1)= dey(i2-1,j2-1,k2-1) -qv2m*wi2(-1)*wk2(-1)
dey(i2  ,j2-1,k2-1)= dey(i2  ,j2-1,k2-1) -qv2m*wi2(0)*wk2(-1)
dey(i2+1,j2-1,k2-1)= dey(i2+1,j2-1,k2-1) -qv2m*wi2(1)*wk2(-1)
```

```
dey(i2-1,j2-1,k2)  = dey(i2-1,j2-1,k2)   -qv2m*wi2(-1)*wk2(0)
dey(i2  ,j2-1,k2)  = dey(i2  ,j2-1,k2)   -qv2m*wi2(0)*wk2(0)
dey(i2+1,j2-1,k2)  = dey(i2+1,j2-1,k2)   -qv2m*wi2(1)*wk2(0)
dey(i2-1,j2-1,k2+1)= dey(i2-1,j2-1,k2+1) -qv2m*wi2(-1)*wk2(1)
dey(i2  ,j2-1,k2+1)= dey(i2  ,j2-1,k2+1) -qv2m*wi2(0)*wk2(1)
dey(i2+1,j2-1,k2+1)= dey(i2+1,j2-1,k2+1) -qv2m*wi2(1)*wk2(1)

dey(i2-1,j2,k2-1)= dey(i2-1,j2,k2-1) -qv2p*wi2(-1)*wk2(-1)
dey(i2  ,j2,k2-1)= dey(i2  ,j2,k2-1) -qv2p*wi2(0)*wk2(-1)
dey(i2+1,j2,k2-1)= dey(i2+1,j2,k2-1) -qv2p*wi2(1)*wk2(-1)
dey(i2-1,j2,k2)  = dey(i2-1,j2,k2)   -qv2p*wi2(-1)*wk2(0)
dey(i2  ,j2,k2)  = dey(i2  ,j2,k2)   -qv2p*wi2(0)*wk2(0)
dey(i2+1,j2,k2)  = dey(i2+1,j2,k2)   -qv2p*wi2(1)*wk2(0)
dey(i2-1,j2,k2+1)= dey(i2-1,j2,k2+1) -qv2p*wi2(-1)*wk2(1)
dey(i2  ,j2,k2+1)= dey(i2  ,j2,k2+1) -qv2p*wi2(0)*wk2(1)
dey(i2+1,j2,k2+1)= dey(i2+1,j2,k2+1) -qv2p*wi2(1)*wk2(1)

dez(i2-1,j2-1,k2-1)= dez(i2-1,j2-1,k2-1) -qw2m*wi2(-1)*wj2(-1)
dez(i2  ,j2-1,k2-1)= dez(i2  ,j2-1,k2-1) -qw2m*wi2(0)*wj2(-1)
dez(i2+1,j2-1,k2-1)= dez(i2+1,j2-1,k2-1) -qw2m*wi2(1)*wj2(-1)
```

```
dez(i2-1,j2  ,k2-1)= dez(i2-1,j2  ,k2-1) -qw2m*wi2(-1)*wj2(0)
dez(i2  ,j2  ,k2-1)= dez(i2  ,j2  ,k2-1) -qw2m*wi2(0)*wj2(0)
dez(i2+1,j2  ,k2-1)= dez(i2+1,j2  ,k2-1) -qw2m*wi2(1)*wj2(0)
dez(i2-1,j2+1,k2-1)= dez(i2-1,j2+1,k2-1) -qw2m*wi2(-1)*wj2(1)
dez(i2  ,j2+1,k2-1)= dez(i2  ,j2+1,k2-1) -qw2m*wi2(0)*wj2(1)
dez(i2+1,j2+1,k2-1)= dez(i2+1,j2+1,k2-1) -qw2m*wi2(1)*wj2(1)

dez(i2-1,j2-1,k2)= dez(i2-1,j2-1,k2) -qw2p*wi2(-1)*wj2(-1)
dez(i2  ,j2-1,k2)= dez(i2  ,j2-1,k2) -qw2p*wi2(0)*wj2(-1)
dez(i2+1,j2-1,k2)= dez(i2+1,j2-1,k2) -qw2p*wi2(1)*wj2(-1)
dez(i2-1,j2  ,k2)= dez(i2-1,j2  ,k2) -qw2p*wi2(-1)*wj2(0)
dez(i2  ,j2  ,k2)= dez(i2  ,j2  ,k2) -qw2p*wi2(0)*wj2(0)
dez(i2+1,j2  ,k2)= dez(i2+1,j2  ,k2) -qw2p*wi2(1)*wj2(0)
dez(i2-1,j2+1,k2)= dez(i2-1,j2+1,k2) -qw2p*wi2(-1)*wj2(1)
dez(i2  ,j2+1,k2)= dez(i2  ,j2+1,k2) -qw2p*wi2(0)*wj2(1)
dez(i2+1,j2+1,k2)= dez(i2+1,j2+1,k2) -qw2p*wi2(1)*wj2(1)

return
end
```