# Computational Methods for Kinetic Processes in Plasma Physics

**Ken Nishikawa**

***Department of Physics/UAH***

Main program 4

June 4, 2015

# Main program (continued)

```
c  **************************************************************
      subroutine Smooth_Current(dex,dey,dez,mFx,mFy,mFz,sm,
     &                    FBDLx,FBDLy,FBDLz,FBDRx,FBDRy,FBDRz)

      integer FBDRx,FBDRy,FBDRz
      integer FBDLx,FBDLy,FBDLz

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

c  must zero these arrays - if not severe errors appear!!!
      do k = 1,mFz
       do j = 1,mFy
        do i = 1,mFx
              dextemp(i,j,k) = 0.0
              deytemp(i,j,k) = 0.0
              deztemp(i,j,k) = 0.0
        end do
      end do
      end do

c  filtering operates on cells in the "particle core" only
      do k = FBDLz,FBDRz
       do j = FBDLy,FBDRy
        do i = FBDLx,FBDRx
         do nz = -1,1
          do ny = -1,1
           do nx = -1,1
            dextemp(i+nx,j+ny,k+nz) = dextemp(i+nx,j+ny,k+nz)
     &                              +sm(nx,ny,nz)*dex(i,j,k)
            deytemp(i+nx,j+ny,k+nz) = deytemp(i+nx,j+ny,k+nz)
     &                              +sm(nx,ny,nz)*dey(i,j,k)
            deztemp(i+nx,j+ny,k+nz) = deztemp(i+nx,j+ny,k+nz)
     &                              +sm(nx,ny,nz)*dez(i,j,k)
           end do
          end do
         end do
        end do
        end do
      end do

c ** because of using temporary arrays "dextemp..." guard cells in "dex..." **
c ** contain contributions from the above applied filtering only -          **
c ** contributions from current deposition which have been already passed   **
c ** are automatically erased                                               **
      do k = 1,mFz
       do j = 1,mFy
        do i = 1,mFx
              dex(i,j,k) = dextemp(i,j,k)
              dey(i,j,k) = deytemp(i,j,k)
              dez(i,j,k) = deztemp(i,j,k)
        end do
       end do
      end do

      return
      end
```

```
c  **********************************************************************
       subroutine E_field_update(ex,ey,ez,dex,dey,dez,mFx,mFy,mFz,
     &                        FBDLx,FBDLy,FBDLz,FBDRx,FBDRy,FBDRz)

      integer FBDRx,FBDRy,FBDRz
      integer FBDLx,FBDLy,FBDLz

      dimension ex(mFx,mFy,mFz),ey(mFx,mFy,mFz),ez(mFx,mFy,mFz)
      dimension dex(mFx,mFy,mFz),dey(mFx,mFy,mFz),dez(mFx,mFy,mFz)

c ** indices range for cells in the "particle core" only - other elements     **
c ** provided by "Field_passing" subroutine; only i-index range is extended to**
c ** properly update fields in the guard cells of the Left and Right domains  **
      do k = FBDLz,FBDRz
       do j = FBDLy,FBDRy
        do i = FBDLx-1,FBDRx+1
           ex(i,j,k) = ex(i,j,k) + dex(i,j,k)
           ey(i,j,k) = ey(i,j,k) + dey(i,j,k)
           ez(i,j,k) = ez(i,j,k) + dez(i,j,k)
        end do
       end do
      end do
```

```
c  clear "current" arrays
      do k = 1,mFz
       do j = 1,mFy
        do i = 1,mFx
               dex(i,j,k) = 0.0
               dey(i,j,k) = 0.0
               dez(i,j,k) = 0.0
        end do
       end do
      end do

      return
      end
```

```
c  *****************************************************************

c  random number generator
c ** this is ran2 routine from Numerical Recepies (name changed to ran1) **
      real*8 FUNCTION ran1(idum)
      INTEGER idum,IM1,IM2,IMM1,IA1,IA2,IQ1,IQ2,IR1,IR2,NTAB,NDIV
c       REAL ran2,AM,EPS,RNMX
      REAL AM,EPS,RNMX
      PARAMETER (IM1=2147483563,IM2=2147483399,AM=1./IM1,IMM1=IM1-1,
     &           IA1=40014,IA2=40692,IQ1=53668,IQ2=52774,IR1=12211,
     &     IR2=3791,NTAB=32,NDIV=1+IMM1/NTAB,EPS=1.2e-7,RNMX=1.-EPS)
      INTEGER idum2,j,k,iv(NTAB),iy
      SAVE iv,iy,idum2
      DATA idum2/123456789/, iv/NTAB*0/, iy/0/

      if (idum.le.0) then
       idum=max(-idum,1)
       idum2=idum
```

```
        do j=NTAB+8,1,-1
        k=idum/IQ1
        idum=IA1*(idum-k*IQ1)-k*IR1
        if (idum.lt.0) idum=idum+IM1
        if (j.le.NTAB) iv(j)=idum
       enddo
       iy=iv(1)
      endif

      k=idum/IQ1
      idum=IA1*(idum-k*IQ1)-k*IR1
      if (idum.lt.0) idum=idum+IM1
      k=idum2/IQ2
      idum2=IA2*(idum2-k*IQ2)-k*IR2
      if (idum2.lt.0) idum2=idum2+IM2
      j=1+iy/NDIV
      iy=iv(j)-idum2
      iv(j)=idum
      if(iy.lt.1)iy=iy+IMM1
c       ran2=min(AM*iy,RNMX)
      ran1=min(AM*iy,RNMX)
      return
      END
```

```
c *************************************************************************
c                   PARTICLE SORTING SUBROUTINES
c *************************************************************************
      subroutine Sort_particles(ipar,x,y,z,u,v,w,mh,nFy,nFz,
     &                                    DHDx,DHDy,DHDz)

      integer isortZ(3:nFz+2),isortZY(3:nFz+2,3:nFy+2)

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

      dimension indx(mh),indx1(mh),indx2(mh)


      do j=1,ipar
       indx2(j)=j
       indx1(j)=j
       indx(j)=j
      end do
```

```
         do i = 3,nFz+2
         isortZ(i)=0
         do j = 3,nFy+2
          isortZY(i,j)=0
         end do
        end do

c  first sort in "k"
        call indexx(1,ipar,ipar,DHDz,z,indx2,indx2,indx)

c  how many elements has each k=int(z(indx(i))) of the sorted array?
c  e.g., k=3 has 10, k=4 has 2 etc. elements with j,i still to be sorted
c  array "isortZ" records: isortZ(3)=10,isortZ(4)=12 etc.
        inz=3
        ismax=0
        do i=1,ipar
         if (int(z(indx(i))-DHDz) .eq. inz) then
          isortZ(inz)=isortZ(inz)+1
                ismax=max(isortZ(inz),0)
```

```
           else
                do j = inz,nFz+1
                 inz = inz+1
                 if (int(z(indx(i))-DHDz) .eq. inz) then
                  isortZ(inz)=ismax+1
                  ismax=max(isortZ(inz),0)
                  goto 24
                 end if
        end do
       end if
24       Continue
      end do

c  then, for given "k" sort in "j"
      ismax=0
      do i = 3,nFz+2
       if ((isortZ(i)-ismax) .gt. 0) then
                call indexx(ismax+1,isortZ(i),ipar,DHDy,y,indx2,indx,indx1)
       end if
       ismax=max(isortZ(i),ismax)
      end do
```

```
inz=3
iny=3
ismax=0
do i=1,ipar
 if (int(z(indx(indx1(i)))-DHDz) .eq. inz) then
         if (int(y(indx(indx1(i)))-DHDy) .eq. iny) then
          isortZY(inz,iny)=isortZY(inz,iny)+1
          ismax=max(isortZY(inz,iny),0)
         else
          do j = iny,nFy+1
           iny = iny+1
           if (int(y(indx(indx1(i)))-DHDy) .eq. iny) then
            isortZY(inz,iny)=ismax+1
            ismax=max(isortZY(inz,iny),0)
            go to 25
           end if
          end do
         end if
```

```
   else
     iny=3
     do j = inz,nFz+1
      inz = inz+1
      if (int(z(indx(indx1(i)))-DHDz) .eq. inz) then
       if (int(y(indx(indx1(i)))-DHDy) .eq. iny) then
        isortZY(inz,iny)=ismax+1
        ismax=max(isortZY(inz,iny),0)
        go to 25
       else
        do k = iny,nFy+1
         iny = iny+1
         if (int(y(indx(indx1(i)))-DHDy) .eq. iny) then
          isortZY(inz,iny)=ismax+1
          ismax=max(isortZY(inz,iny),0)
          go to 25
         end if
end do
       end if
      end if
     end do
```

```
      end if
25     Continue
      end do

c  finally, for given "k" and "j" sort in "i"
      ismax=0
      do i = 3,nFz+2
       do j = 3,nFy+2
            if ((isortZY(i,j)-ismax) .gt. 0) then
             call indexx(ismax+1,isortZY(i,j),ipar,DHDx,x,
     &                        indx,indx1,indx2)
            end if
            ismax=max(isortZY(i,j),ismax)
       end do
      end do

c  now, rearrange each array using index arrays (requires temporary array)
      do i = 1,ipar
       temp(i) = x(i)
      end do
```

```
do i = 1,ipar
 x(i) = temp(indx(indx1(indx2(i))))
end do
do i = 1,ipar
 temp(i) = y(i)
end do
do i = 1,ipar
 y(i) = temp(indx(indx1(indx2(i))))
end do
do i = 1,ipar
 temp(i) = z(i)
end do
do i = 1,ipar
 z(i) = temp(indx(indx1(indx2(i))))
end do
do i = 1,ipar
 temp(i) = u(i)
end do
do i = 1,ipar
 u(i) = temp(indx(indx1(indx2(i))))
end do
```

```
do i = 1,ipar
 temp(i) = v(i)
end do
do i = 1,ipar
 v(i) = temp(indx(indx1(indx2(i))))
end do
do i = 1,ipar
 temp(i) = w(i)
end do
do i = 1,ipar
 w(i) = temp(indx(indx1(indx2(i))))
end do

return
end
```

```
c ********************************************************************************

c  subroutine from Num. Recepies (Sec. 8.4) constructing index table for array
c  being sorted. Changes include: a) sorting in int(arr), b) additional index
c  arrays for sorting in k, j, i separately in a way to produce specific
c  order the arrays were at the start of simulation --> see "Paricle_init"

          SUBROUTINE indexx(n1,n2,n,DHD,arr,indx0,indx1,indx)
          INTEGER n1,n2,n,indx(n),indx0(n),indx1(n),M,NSTACK
cPIC      REAL arr(n)
          dimension arr(n)
          PARAMETER (M=7,NSTACK=500)
c  Indexes an array arr(1:n), i.e., outputs the array indx(1:n) such that
c  arr(indx(j)) is in ascending order for j = 1, 2, . . . ,N. The input
c  quantities n and arr are not changed.
          INTEGER i,indxt,ir,itemp,j,jstack,k,l,istack(NSTACK)
cPIC      REAL a
          integer a
```

```
cPIC      do j=1,n
          do j=n1,n2
           indx(j)=j
          end do
          jstack=0
cPIC      l=1
cPIC      ir=n
          l=n1
          ir=n2
1         if(ir-l.lt.M)then
           do j=l+1,ir
            indxt=indx(j)
cPIC        a=arr(indxt)
            a=arr(indx0(indx1(indxt)))-DHD
            do i=j-1,l,-1
cPIC         if(arr(indx(i)).le.a)goto 2
             if((arr(indx0(indx1(indx(i))))-DHD).le.a)goto 2
             indx(i+1)=indx(i)
            end do
            i=l-1
```

```
2          indx(i+1)=indxt
          end do
          if(jstack.eq.0)return
          ir=istack(jstack)
          l=istack(jstack-1)
          jstack=jstack-2
         else
          k=(l+ir)/2
          itemp=indx(k)
          indx(k)=indx(l+1)
          indx(l+1)=itemp
cPIC       if(arr(indx(l)).gt.arr(indx(ir)))then
           if(int(arr(indx0(indx1(indx(l))))-DHD).gt.
     &       int(arr(indx0(indx1(indx(ir))))-DHD))then
            itemp=indx(l)
            indx(l)=indx(ir)
            indx(ir)=itemp
           endif
cPIC       if(arr(indx(l+1)).gt.arr(indx(ir)))then
           if(int(arr(indx0(indx1(indx(l+1))))-DHD).gt.
     &  int(arr(indx0(indx1(indx(ir))))-DHD))then
```

```
              itemp=indx(l+1)
              indx(l+1)=indx(ir)
              indx(ir)=itemp
             endif
cPIC         if(arr(indx(l)).gt.arr(indx(l+1)))then
             if(int(arr(indx0(indx1(indx(l))))-DHD).gt.
    &      int(arr(indx0(indx1(indx(l+1))))-DHD))then
              itemp=indx(l)
              indx(l)=indx(l+1)
              indx(l+1)=itemp
             endif
             i=l+1
             j=ir
             indxt=indx(l+1)
cPIC         a=arr(indxt)
             a=arr(indx0(indx1(indxt)))-DHD
3            continue
         i=i+1
cPIC         if(arr(indx(i)).lt.a)goto 3
             if((arr(indx0(indx1(indx(i))))-DHD).lt.a)goto 3
```

```
4           continue
      j=j-1
cPIC        if(arr(indx(j)).gt.a)goto 4
            if(int(arr(indx0(indx1(indx(j))))-DHD).gt.a)goto 4
            if(j.lt.i)goto 5
            itemp=indx(i)
            indx(i)=indx(j)
            indx(j)=itemp

            goto 3
5           indx(l+1)=indx(j)
     indx(j)=indxt
            jstack=jstack+2
            if(jstack.gt.NSTACK)pause 'NSTACK too small in indexx'
            if(ir-i+1.ge.j-l)then
             istack(jstack)=ir
             istack(jstack-1)=i
      ir=j-1
```

```
 else
  istack(jstack)=j-1
  istack(jstack-1)=l
  l=i
 endif
endif
goto 1
END
```

```
c  ***************************************************************************
c                    COMMUNICATION SUBROUTINES
c  ***************************************************************************
      subroutine Field_passing(fx,fy,fz,mFx,mFy,mFz,mc,mrl,mrh,
     &              dims,coords,FBDLx,FBDLy,FBDLz,FBDRx,FBDRy,FBDRz,
     &                     nleft,nright,nfront,nrear,nbottom,ntop)
      include 'mpif.h'

c       integer myid,Nproc
      integer lgrp,comm3d,ierror,Tag,istatus(MPI_STATUS_SIZE)
      integer dims(3),coords(3)
      integer FBDRx,FBDRy,FBDRz
      integer FBDLx,FBDLy,FBDLz
      integer requestx,requesty,requestz
      integer requestx1,requesty1,requestz1

      dimension fx(mFx,mFy,mFz),fy(mFx,mFy,mFz),fz(mFx,mFy,mFz)
      dimension fxs(2:mc,mrl:mrh),fys(2:mc,mrl:mrh),fzs(2:mc,mrl:mrh)
      dimension fxr(2:mc,mrl:mrh),fyr(2:mc,mrl:mrh),fzr(2:mc,mrl:mrh)
```

```
        common /pparms/ lgrp,comm3d

       Tag = 100

c  communication is done separately for each dimension, so that number of
c  buffer zones is minimal; contributions from edge and corner cells are
c  automatically properly passed after the three loops

c  attention !!!
c  the present version allows for non-cubic domains - the domain sizes in
c  y and z-direction must be the same, x-size may vary
c ** to keep minimal number of buffer arrays and minimize communication (pass **
c ** only actual surface points) we change buffer counts in each dimension    **
c ** (for non-cubic domains); also data packing (and unpacking) to buffers    **
c ** is handled in a way to account for row-wise passing of arrays in MPI     **
c !! useful to check if array passing is row-wise when using different MPI !!
c !! implementation                                                         !!
```

```
c  periodic boundary conditions for B-fields, that were imposed in 1D version
c  in "copylayr" subroutine are now automatically embedded here by making
c  grid topology periodic
     do n = 1,3

c  send fields to the Right (or Rear or Top)
c  pack fields to buffers
      if (n.eq.1) then
       mcount=FBDRy*FBDRz

c ** cell indices passed between domains range from 2 to nFi+3 (nFi=nFx,nFy,..)**
c ** however for leftmost and rightmost domains limits are changed in x-direct.**
c ** because this direction is not periodic: i=1,nFx+3 in leftmost domains and **
c ** i=2,nFx+5 in rightmost domains; because of that "mrl" and "mrh" in buffer **
c ** arrays change depending on position along x-dir.; this works well for     **
c ** communication in y and z-dir (loop n=2,3), but problem arises for leftmost**
c ** domains in communication with right neighbors, because buffer arrays      **
c ** indexing differs between them and rows are sent starting from k=1 but     **
c ** but received from k=2; that's why sent buffers for leftmost domains below **
```

```
c ** copy field arrays with a shift in "k", then they are unpacked properly    **
c ** from the receive buffers on the right; this method also ensures proper    **
c ** counts for passing!!                                                       **
               if (coords(1).eq.0) then
                do k = FBDLz-1,FBDRz+1
                 do j = FBDLy-1,FBDRy+1
          fxs(j,k-1) = fx(FBDRx,j,k)
          fys(j,k-1) = fy(FBDRx,j,k)
                  fzs(j,k-1) = fz(FBDRx,j,k)
         end do
        end do
               else
                do k = FBDLz-1,FBDRz+1
                 do j = FBDLy-1,FBDRy+1
          fxs(j,k) = fx(FBDRx,j,k)
          fys(j,k) = fy(FBDRx,j,k)
                  fzs(j,k) = fz(FBDRx,j,k)
         end do
        end do
               end if
        neighr = nright
        neighl = nleft
else if (n.eq.2) then
        mcount=FBDRz*(FBDRx-FBDLx+3)

        do i = FBDLx-1,FBDRx+1
  do k = FBDLz-1,FBDRz+1
   fxs(k,i) = fx(i,FBDRy,k)
   fys(k,i) = fy(i,FBDRy,k)
          fzs(k,i) = fz(i,FBDRy,k)
  end do
 end do
 neighr = nrear
        neighl = nfront
else if (n.eq.3) then
 mcount=FBDRy*(FBDRx-FBDLx+3)

        do i = FBDLx-1,FBDRx+1
  do j = FBDLy-1,FBDRy+1
```

```
     fxs(j,i) = fx(i,j,FBDRz)
     fys(j,i) = fy(i,j,FBDRz)
            fzs(j,i) = fz(i,j,FBDRz)
    end do
   end do
          neighr = ntop
          neighl = nbottom
  end if

  call MPI_IRECV(fxr,mcount,MPI_DOUBLE_PRECISION,neighl,Tag+1,
&               comm3d,requestx,ierror)
  call MPI_IRECV(fyr,mcount,MPI_DOUBLE_PRECISION,neighl,Tag+2,
&               comm3d,requesty,ierror)
  call MPI_IRECV(fzr,mcount,MPI_DOUBLE_PRECISION,neighl,Tag+3,
&               comm3d,requestz,ierror)

  call MPI_SEND(fxs,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+1,
&              comm3d,ierror)
  call MPI_SEND(fys,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+2,
&              comm3d,ierror)
```

```
       call MPI_SEND(fzs,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+3,
     &               comm3d,ierror)

       call MPI_WAIT(requestx,istatus,ierror)
       call MPI_WAIT(requesty,istatus,ierror)
       call MPI_WAIT(requestz,istatus,ierror)

c  unpack buffers
       if (n.eq.1 .and. coords(1).ne.0) then
c !!! do the same for other directions if they are non-periodic !!!
c ** because boundary conditions in x-direction are non-periodic, processes **
c ** without neighbors send or receive information to (from) MPI_PROC_NULL  **
c ** upon which receive buffers are not changed; DON'T UNPACK THESE
c      BUFFERS **
c ** (buffers from right shift to processes on the left are zeroed arrays   **
c ** and buffers in the left shift to processes on the right are the same   **
c ** as for the right shift)                                                **
               do k = FBDLz-1,FBDRz+1
                do j = FBDLy-1,FBDRy+1
         fx(FBDLx-1,j,k) = fxr(j,k)
         fy(FBDLx-1,j,k) = fyr(j,k)
```

```
           fz(FBDLx-1,j,k) = fzr(j,k)
  end do
 end do

else if (n.eq.2) then
         do i = FBDLx-1,FBDRx+1
  do k = FBDLz-1,FBDRz+1
   fx(i,FBDLy-1,k) = fxr(k,i)
   fy(i,FBDLy-1,k) = fyr(k,i)
           fz(i,FBDLy-1,k) = fzr(k,i)
  end do
 end do

else if (n.eq.3) then
         do i = FBDLx-1,FBDRx+1
  do j = FBDLy-1,FBDRy+1
   fx(i,j,FBDLz-1) = fxr(j,i)
   fy(i,j,FBDLz-1) = fyr(j,i)
           fz(i,j,FBDLz-1) = fzr(j,i)
  end do
 end do
```

```
         end if


c  send fields to the Left (or Front or Bottom)
c  pack fields to buffers
       if (n.eq.1) then
        mcount=FBDRy*FBDRz

               do k = FBDLz-1,FBDRz+1
                do j = FBDLy-1,FBDRy+1
          fxs(j,k) = fx(FBDLx,j,k)
          fys(j,k) = fy(FBDLx,j,k)
                 fzs(j,k) = fz(FBDLx,j,k)
         end do
        end do
        neighr = nright
               neighl = nleft
       else if (n.eq.2) then
               mcount=FBDRz*(FBDRx-FBDLx+3)
        do i = FBDLx-1,FBDRx+1
  do k = FBDLz-1,FBDRz+1
   fxs(k,i) = fx(i,FBDLy,k)
   fys(k,i) = fy(i,FBDLy,k)
         fzs(k,i) = fz(i,FBDLy,k)
  end do
 end do
 neighr = nrear
        neighl = nfront
else if (n.eq.3) then
        mcount=FBDRy*(FBDRx-FBDLx+3)

        do i = FBDLx-1,FBDRx+1
  do j = FBDLy-1,FBDRy+1
   fxs(j,i) = fx(i,j,FBDLz)
   fys(j,i) = fy(i,j,FBDLz)
         fzs(j,i) = fz(i,j,FBDLz)
  end do
 end do
```

```
         neighr = ntop
         neighl = nbottom
  end if

  call MPI_IRECV(fxr,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+4,
&                comm3d,requestx1,ierror)
  call MPI_IRECV(fyr,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+5,
&                comm3d,requesty1,ierror)
  call MPI_IRECV(fzr,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+6,
&                comm3d,requestz1,ierror)

  call MPI_SEND(fxs,mcount,MPI_DOUBLE_PRECISION,neighl,Tag+4,
&               comm3d,ierror)
  call MPI_SEND(fys,mcount,MPI_DOUBLE_PRECISION,neighl,Tag+5,
&               comm3d,ierror)
  call MPI_SEND(fzs,mcount,MPI_DOUBLE_PRECISION,neighl,Tag+6,
&               comm3d,ierror)

  call MPI_WAIT(requestx1,istatus,ierror)
  call MPI_WAIT(requesty1,istatus,ierror)
  call MPI_WAIT(requestz1,istatus,ierror)
```

```
c  unpack buffers
       if (n.eq.1 .and. coords(1).ne.(dims(1)-1)) then
c ** situation analogous to the right send: field elements sent to leftmost **
c ** processes starting from k=2 are received from k=1                       **
               if (coords(1).eq.0) then
                do k = FBDLz-1,FBDRz+1
                 do j = FBDLy-1,FBDRy+1
           fx(FBDRx+1,j,k) = fxr(j,k-1)
           fy(FBDRx+1,j,k) = fyr(j,k-1)
                  fz(FBDRx+1,j,k) = fzr(j,k-1)
          end do
         end do
               else
                do k = FBDLz-1,FBDRz+1
                 do j = FBDLy-1,FBDRy+1
           fx(FBDRx+1,j,k) = fxr(j,k)
           fy(FBDRx+1,j,k) = fyr(j,k)
                  fz(FBDRx+1,j,k) = fzr(j,k)
          end do
         end do
end if

else if (n.eq.2) then
        do i = FBDLx-1,FBDRx+1
  do k = FBDLz-1,FBDRz+1
   fx(i,FBDRy+1,k) = fxr(k,i)
   fy(i,FBDRy+1,k) = fyr(k,i)
          fz(i,FBDRy+1,k) = fzr(k,i)
  end do
 end do

else if (n.eq.3) then
        do i = FBDLx-1,FBDRx+1
  do j = FBDLy-1,FBDRy+1
   fx(i,j,FBDRz+1) = fxr(j,i)
   fy(i,j,FBDRz+1) = fyr(j,i)
          fz(i,j,FBDRz+1) = fzr(j,i)
  end do
 end do
end if
```

```
 end do

return
end
```