Computational Methods for Kinetic Processes in Plasma Physics

Ken Nishikawa

Department of Physics/UAH

Main program 5

June 4, 2015

Main program (continued)

```
c  ************************************************************************
       subroutine Field_passing2(fx,fy,fz,mFx,mFy,mFz,mc,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 R,L
       integer requestx,requesty,requestz
       integer requestx1,requesty1,requestz1

       dimension fx(mFx,mFy,mFz),fy(mFx,mFy,mFz),fz(mFx,mFy,mFz)
       dimension fxs(mc,mrh),fys(mc,mrh),fzs(mc,mrh)
       dimension fxr(mc,mrh),fyr(mc,mrh),fzr(mc,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)
       do nguard = 1,2
c  pack fields to buffers
       if (n.eq.1) then
        R = FBDRx - nguard +1
                   mcount=(FBDRy+2)*(FBDRz+2)

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

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

        do i = FBDLx-2,FBDRx+2
   do j = FBDLy-2,FBDRy+2
    fxs(j,i) = fx(i,j,R)
    fys(j,i) = fy(i,j,R)
         fzs(j,i) = fz(i,j,R)
   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)                                     **
               L = FBDLx - nguard
               do k = FBDLz-2,FBDRz+2
                do j = FBDLy-2,FBDRy+2
         fx(L,j,k) = fxr(j,k)
         fy(L,j,k) = fyr(j,k)
                 fz(L,j,k) = fzr(j,k)
        end do
       end do

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

```
           else if (n.eq.3) then
               L = FBDLz - nguard
               do i = FBDLx-2,FBDRx+2
         do j = FBDLy-2,FBDRy+2
          fx(i,j,L) = fxr(j,i)
          fy(i,j,L) = fyr(j,i)
                 fz(i,j,L) = fzr(j,i)
         end do
        end do
       end if

       end do


c  send fields to the Left (or Front or Bottom)
       do nguard = 1,2
c  pack fields to buffers
       if (n.eq.1) then
       L = FBDLx + nguard -1
       mcount=(FBDRy+2)*(FBDRz+2)

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

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

        do i = FBDLx-2,FBDRx+2
   do j = FBDLy-2,FBDRy+2
    fxs(j,i) = fx(i,j,L)
    fys(j,i) = fy(i,j,L)
         fzs(j,i) = fz(i,j,L)
   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                    **
               R = FBDRx + nguard
               do k = FBDLz-2,FBDRz+2
                do j = FBDLy-2,FBDRy+2
         fx(R,j,k) = fxr(j,k)
         fy(R,j,k) = fyr(j,k)
                fz(R,j,k) = fzr(j,k)
        end do
       end do

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

```
   else if (n.eq.3) then
        R = FBDRz + nguard
        do i = FBDLx-2,FBDRx+2
  do j = FBDLy-2,FBDRy+2
   fx(i,j,R) = fxr(j,i)
   fy(i,j,R) = fyr(j,i)
          fz(i,j,R) = fzr(j,i)
  end do
 end do
 end if

 end do

end do

return
end
```

```
c  **************************************************************************
      subroutine E_Field_Passing_Add(ex,ey,ez,mFx,mFy,mFz,mcol,mrow,
     &           dims,coords,FBDRx,FBDRy,FBDRz,FBDLx,FBDLy,FBDLz,
     &                 nleft,nright,nfront,nrear,nbottom,ntop)
      include 'mpif.h'

      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 R,L
      integer requestx,requesty,requestz

      dimension ex(mFx,mFy,mFz),ey(mFx,mFy,mFz),ez(mFx,mFy,mFz)
      dimension fxs(mcol,mrow),fys(mcol,mrow),fzs(mcol,mrow)
      dimension fxr(mcol,mrow),fyr(mcol,mrow),fzr(mcol,mrow)

      common /pparms/ lgrp,comm3d

      Tag = 100
```

```
c  contributions from current deposit in guard cells are passed to the
c  appropriate domains (guard cells carry field elements from recent deposition
c  only, because they were cleared before in "clearlay")                    **

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 over "n"
c ** 2 "left" and 3 "right" layers must be passed in each dimension; this is **
c ** accomplished in loops over "nguard"

c !! note different buffer size compared to "Field_passing" subroutine !!
c ** information from all guard cells must be passed **

c  embedded here is "addlayer" subroutine from 1D parallel version (n=2,3)
      do n = 1,3

c  send fields to the Right (or Rear or Top)
c        do nguard = 1,3
       do nguard = 1,2
```

```
c  pack fields to buffers
       if (n.eq.1) then
        R = FBDRx + nguard
               mcount=mFy*mFz

               do k = 1,mFz
                do j = 1,mFy
         fxs(j,k) = ex(R,j,k)
         fys(j,k) = ey(R,j,k)
                 fzs(j,k) = ez(R,j,k)
         end do
         end do
         neighr = nright
               neighl = nleft

        else if (n.eq.2) then
         R = FBDRy + nguard
               mcount=mFz*mFx
```

```
 do i = 1,mFx
 do k = 1,mFz
  fxs(k,i) = ex(i,R,k)
  fys(k,i) = ey(i,R,k)
      fzs(k,i) = ez(i,R,k)
 end do
end do
neighr = nrear
neighl = nfront

else if (n.eq.3) then
    R = FBDRz + nguard
    mcount=mFy*mFx

    do i = 1,mFx
 do j = 1,mFy
  fxs(j,i) = ex(i,j,R)
  fys(j,i) = ey(i,j,R)
      fzs(j,i) = ez(i,j,R)
 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  update electric field elements
       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 **
cJET        if (n.eq.1) then
            L = FBDLx + nguard-1
            do k = 1,mFz
             do j = 1,mFy
         ex(L,j,k) = ex(L,j,k)+fxr(j,k)
         ey(L,j,k) = ey(L,j,k)+fyr(j,k)
              ez(L,j,k) = ez(L,j,k)+fzr(j,k)
        end do
        end do

        else if (n.eq.2) then
            L = FBDLy + nguard-1
```

```
  do i = 1,mFx
  do k = 1,mFz
   ex(i,L,k) = ex(i,L,k)+fxr(k,i)
   ey(i,L,k) = ey(i,L,k)+fyr(k,i)
          ez(i,L,k) = ez(i,L,k)+fzr(k,i)
  end do
  end do

 else if (n.eq.3) then
 L = FBDLz + nguard-1
        do i = 1,mFx
  do j = 1,mFy
   ex(i,j,L) = ex(i,j,L)+fxr(j,i)
   ey(i,j,L) = ey(i,j,L)+fyr(j,i)
          ez(i,j,L) = ez(i,j,L)+fzr(j,i)
  end do
 end do
 end if

end do
```

```
c  send fields to the Left (or Front or Bottom)
cU1       do nguard = 1,2
       do nguard = 2,2
c  pack fields to buffers
        if (n.eq.1) then
         L = FBDLx + nguard-3
                mcount=mFy*mFz

                do k = 1,mFz
                 do j = 1,mFy
          fxs(j,k) = ex(L,j,k)
          fys(j,k) = ey(L,j,k)
                  fzs(j,k) = ez(L,j,k)
         end do
         end do
         neighr = nright
                neighl = nleft

         else if (n.eq.2) then
         L = FBDLy + nguard-3
                mcount=mFz*mFx
```

```
 do i = 1,mFx
 do k = 1,mFz
  fxs(k,i) = ex(i,L,k)
  fys(k,i) = ey(i,L,k)
          fzs(k,i) = ez(i,L,k)
 end do
 end do
 neighr = nrear
        neighl = nfront

else if (n.eq.3) then
L = FBDLz + nguard-3
        mcount=mFy*mFx

        do i = 1,mFx
 do j = 1,mFy
  fxs(j,i) = ex(i,j,L)
  fys(j,i) = ey(i,j,L)
          fzs(j,i) = ez(i,j,L)
 end do
 end do
```

```
           neighr = ntop
           neighl = nbottom
      end if

      call MPI_IRECV(fxr,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+4,
     &                  comm3d,requestx,ierror)
      call MPI_IRECV(fyr,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+5,
     &                  comm3d,requesty,ierror)
      call MPI_IRECV(fzr,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+6,
     &                  comm3d,requestz,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(requestx,istatus,ierror)
      call MPI_WAIT(requesty,istatus,ierror)
      call MPI_WAIT(requestz,istatus,ierror)
```

```
c  update electric field elements
       if (n.eq.1 .and. coords(1).ne.(dims(1)-1)) then
c              if (n.eq.1) then
               R = FBDRx + nguard-2
               do k = 1,mFz
                do j = 1,mFy
          ex(R,j,k) = ex(R,j,k)+fxr(j,k)
          ey(R,j,k) = ey(R,j,k)+fyr(j,k)
                ez(R,j,k) = ez(R,j,k)+fzr(j,k)
         end do
        end do

        else if (n.eq.2) then
        R = FBDRy + nguard-2
        do i = 1,mFx
         do k = 1,mFz
          ex(i,R,k) = ex(i,R,k)+fxr(k,i)
          ey(i,R,k) = ey(i,R,k)+fyr(k,i)
                ez(i,R,k) = ez(i,R,k)+fzr(k,i)
         end do
         end do
```

```
  else if (n.eq.3) then
 R = FBDRz + nguard-2
        do i = 1,mFx
  do j = 1,mFy
   ex(i,j,R) = ex(i,j,R)+fxr(j,i)
   ey(i,j,R) = ey(i,j,R)+fyr(j,i)
          ez(i,j,R) = ez(i,j,R)+fzr(j,i)
  end do
 end do
end if

end do
end do

99   Continue

return
end
```

```
c ************************************************************************
      subroutine Current_Passing(ex,ey,ez,mFx,mFy,mFz,mcol,mrow,
     &         dims,coords,FBDRx,FBDRy,FBDRz,FBDLx,FBDLy,FBDLz,
     &                 nleft,nright,nfront,nrear,nbottom,ntop)
      include 'mpif.h'

      integer lgrp,comm3d,ierror,Tag,istatus(MPI_STATUS_SIZE)
      integer dims(3),coords(3)
      integer FBDRx,FBDRy,FBDRz
      integer FBDLx,FBDLy,FBDLz
      integer R,L
      integer requestx,requesty,requestz

      dimension ex(mFx,mFy,mFz),ey(mFx,mFy,mFz),ez(mFx,mFy,mFz)
      dimension fxs(mcol,mrow),fys(mcol,mrow),fzs(mcol,mrow)
      dimension fxr(mcol,mrow),fyr(mcol,mrow),fzr(mcol,mrow)

      common /pparms/ lgrp,comm3d

      Tag = 100
```

```
c  the same subroutine as "E_Field_Passing_Add"
c  only 1 "left" and 1 "right" layer must be passed in each dimension -
c  loops over "nguard" are changed appropriately

     do n = 1,3

c  send fields to the Right (or Rear or Top)
      do nguard = 1,1
c  pack fields to buffers
       if (n.eq.1) then
        R = FBDRx + nguard
               mcount=mFy*mFz

               do k = 1,mFz
                do j = 1,mFy
         fxs(j,k) = ex(R,j,k)
         fys(j,k) = ey(R,j,k)
                 fzs(j,k) = ez(R,j,k)
          end do
        end do
       neighr = nright
       neighl = nleft

else if (n.eq.2) then
R = FBDRy + nguard
       mcount=mFz*mFx

       do i = 1,mFx
 do k = 1,mFz
  fxs(k,i) = ex(i,R,k)
  fys(k,i) = ey(i,R,k)
         fzs(k,i) = ez(i,R,k)
 end do
end do
neighr = nrear
neighl = nfront

else if (n.eq.3) then
       R = FBDRz + nguard
       mcount=mFy*mFx
```

```
     do i = 1,mFx
     do j = 1,mFy
      fxs(j,i) = ex(i,j,R)
      fys(j,i) = ey(i,j,R)
              fzs(j,i) = ez(i,j,R)
     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  update electric field elements
      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 BUFFERS
cJET        if (n.eq.1) then
            L = FBDLx + nguard-1
            do k = 1,mFz
             do j = 1,mFy
        ex(L,j,k) = ex(L,j,k)+fxr(j,k)
```

```
          ey(L,j,k) = ey(L,j,k)+fyr(j,k)
          ez(L,j,k) = ez(L,j,k)+fzr(j,k)
 end do
end do

else if (n.eq.2) then
        L = FBDLy + nguard-1
        do i = 1,mFx
 do k = 1,mFz
  ex(i,L,k) = ex(i,L,k)+fxr(k,i)
  ey(i,L,k) = ey(i,L,k)+fyr(k,i)
          ez(i,L,k) = ez(i,L,k)+fzr(k,i)
 end do
 end do

else if (n.eq.3) then
L = FBDLz + nguard-1
        do i = 1,mFx
 do j = 1,mFy
  ex(i,j,L) = ex(i,j,L)+fxr(j,i)
  ey(i,j,L) = ey(i,j,L)+fyr(j,i)
```

```
                   ez(i,j,L) = ez(i,j,L)+fzr(j,i)
          end do
         end do
        end if

        end do

c  send fields to the Left (or Front or Bottom)
        do nguard = 2,2
c  pack fields to buffers
        if (n.eq.1) then
         L = FBDLx + nguard-3
                 mcount=mFy*mFz

                 do k = 1,mFz
                  do j = 1,mFy
           fxs(j,k) = ex(L,j,k)
           fys(j,k) = ey(L,j,k)
                   fzs(j,k) = ez(L,j,k)
          end do
          end do
        neighr = nright
        neighl = nleft

else if (n.eq.2) then
L = FBDLy + nguard-3
        mcount=mFz*mFx

        do i = 1,mFx
 do k = 1,mFz
  fxs(k,i) = ex(i,L,k)
  fys(k,i) = ey(i,L,k)
          fzs(k,i) = ez(i,L,k)
 end do
 end do
neighr = nrear
        neighl = nfront

else if (n.eq.3) then
L = FBDLz + nguard-3
        mcount=mFy*mFx
```

```
   do i = 1,mFx
   do j = 1,mFy
    fxs(j,i) = ex(i,j,L)
    fys(j,i) = ey(i,j,L)
           fzs(j,i) = ez(i,j,L)
   end do
  end do
         neighr = ntop
         neighl = nbottom
  end if

  call MPI_IRECV(fxr,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+4,
&                comm3d,requestx,ierror)
  call MPI_IRECV(fyr,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+5,
&                comm3d,requesty,ierror)
  call MPI_IRECV(fzr,mcount,MPI_DOUBLE_PRECISION,neighr,Tag+6,
&                comm3d,requestz,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(requestx,istatus,ierror)
        call MPI_WAIT(requesty,istatus,ierror)
        call MPI_WAIT(requestz,istatus,ierror)

c  update electric field elements
        if (n.eq.1 .and. coords(1).ne.(dims(1)-1)) then
cJET       if (n.eq.1) then
            R = FBDRx + nguard-2
            do k = 1,mFz
             do j = 1,mFy
          ex(R,j,k) = ex(R,j,k)+fxr(j,k)
          ey(R,j,k) = ey(R,j,k)+fyr(j,k)
              ez(R,j,k) = ez(R,j,k)+fzr(j,k)
         end do
        end do
```

```
else if (n.eq.2) then
R = FBDRy + nguard-2
do i = 1,mFx
 do k = 1,mFz
  ex(i,R,k) = ex(i,R,k)+fxr(k,i)
  ey(i,R,k) = ey(i,R,k)+fyr(k,i)
          ez(i,R,k) = ez(i,R,k)+fzr(k,i)
 end do
end do

else if (n.eq.3) then
R = FBDRz + nguard-2
        do i = 1,mFx
 do j = 1,mFy
  ex(i,j,R) = ex(i,j,R)+fxr(j,i)
  ey(i,j,R) = ey(i,j,R)+fyr(j,i)
          ez(i,j,R) = ez(i,j,R)+fzr(j,i)
 end do
end do
end if
```

```
 end do
end do

return
end
```

```
c  *****************************************************************
      subroutine Particle_passing(ipar,mpass,mh,GBLeft,GBRght,
     &                                PBFrnt,PBRear,PBBot,PBTop,
     &                   nleft,nright,nfront,nrear,nbottom,ntop,
     &                                        x,y,z,u,v,w)

      include 'mpif.h'

      integer lgrp,comm3d,ierror,Tag,istatus(MPI_STATUS_SIZE)
      integer CP_sendR,CP_recvR,CP_sendL,CP_recvL
      integer requestp
      integer requestxl,requestyl,requestzl
      integer requestxr,requestyr,requestzr
      integer requestul,requestvl,requestwl
      integer requestur,requestvr,requestwr

      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 CP_Sx(mpass),CP_Sy(mpass),CP_Sz(mpass)
      dimension CP_Su(mpass),CP_Sv(mpass),CP_Sw(mpass)
      dimension CP_Rx(mpass),CP_Ry(mpass),CP_Rz(mpass)
      dimension CP_Ru(mpass),CP_Rv(mpass),CP_Rw(mpass)

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

      common /pparms/ lgrp,comm3d

      Tag = 100

c  communication is done separately for each dimension
c  embedded is particle sorting routine, also separately for left-right,
c  front-rear, and bottom-top directions: this way particles passing through
c  domain edges and corners are taken into account and communication pattern
c  is kept very simple; also the number of buffer arrays is minimized
c ** splitting of particle sorting does not require excess computation, for **
c ** each particle must be checked against crossing six surfaces separately **
```

```
c ** must zero these buffers if non-periodic conditions are used **
      CP_sendR=0
      CP_sendL=0
      CP_recvR=0
      CP_recvL=0

      do n = 1,3

       if (n.eq.1) then
c  sort particles and pick-up those that leave Left or Right particle boundary
c ** particles that are out of boundaries in y and z-direction (edges and      **
c ** corners on the left and right) are also sorted into send buffers          **
c ** subsequent sortings in y and z-directions pick-up these particles and     **
c ** ultimately they are passed to the appropriate domain without a need for   **
c ** a direct communication with up to 26 neighboring domains for each process **

c ** for sorting in x-direction "global" boundaries are used so that particles **
c ** outside the virtual box boundaries are not removed (packed to pass buffer)**
c ** at this point, but only after current deposition                          **
          call Particle_sorting(iparR,iparL,ipar,mpass,mh,GBLeft,GBRght,
     &                         CRx,CRy,CRz,CRu,CRv,CRw,
     &                                 CLx,CLy,CLz,CLu,CLv,CLw,
     &                                               x,y,z,u,v,w)
        neighr = nright
              neighl = nleft
       else if (n.eq.2) then
c  sort particles and pick-up those that leave Front or Rear particle boundary
              call Particle_sorting(iparR,iparL,ipar,mpass,mh,PBFrnt,PBRear,
     &                                 CRy,CRx,CRz,CRu,CRv,CRw,
     &                                 CLy,CLx,CLz,CLu,CLv,CLw,
     &                                               y,x,z,u,v,w)
              neighr = nrear
              neighl = nfront
       else if (n.eq.3) then
c  sort particles and pick-up those that leave Bottom or Top particle boundary
              call Particle_sorting(iparR,iparL,ipar,mpass,mh,PBBot,PBTop,
     &                               CRz,CRy,CRx,CRu,CRv,CRw,
     &                               CLz,CLy,CLx,CLu,CLv,CLw,
     &                                             z,y,x,u,v,w)
        neighr = ntop
              neighl = nbottom
       end if
```

```
c  send particles to the Right (or Rear or Top)
       do i = 1,iparR
        CP_Sx(i)=CRx(i)
        CP_Sy(i)=CRy(i)
        CP_Sz(i)=CRz(i)
        CP_Su(i)=CRu(i)
        CP_Sv(i)=CRv(i)
        CP_Sw(i)=CRw(i)
       end do

c  first send information on how many particles is to be passed
c * CP_sendR is received as CP_recvR *
       CP_sendR = iparR

c ** if there is no neighbor to send or receive information, MPI_PROC_NULL     **
c ** is used by MPI instead of a source or destination rank (MPI_CART_CREATE); *
c ** then SEND and RECEIVE succeed and return and receive buffer is not changed**
c ** because of that the buffers must be cleared after passing which is done   **
c ** by using separate buffer counts for right-left communication and clearing **
```

```
c ** them at the end of the main "do"-loop                              **
       call MPI_IRECV(CP_recvR,1,MPI_INTEGER,neighl,Tag+1,
     &                comm3d,requestp,ierror)
       call MPI_SEND(CP_sendR,1,MPI_INTEGER,neighr,Tag+1,
     &               comm3d,ierror)

       call MPI_WAIT(requestp,istatus,ierror)

c  then send particles
       call MPI_IRECV(CP_Rx,CP_recvR,MPI_DOUBLE_PRECISION,
     &                neighl,Tag+2,comm3d,requestxr,ierror)
       call MPI_IRECV(CP_Ry,CP_recvR,MPI_DOUBLE_PRECISION,
     &                neighl,Tag+3,comm3d,requestyr,ierror)
       call MPI_IRECV(CP_Rz,CP_recvR,MPI_DOUBLE_PRECISION,
     &                neighl,Tag+4,comm3d,requestzr,ierror)
       call MPI_IRECV(CP_Ru,CP_recvR,MPI_DOUBLE_PRECISION,
     &                neighl,Tag+5,comm3d,requestur,ierror)
       call MPI_IRECV(CP_Rv,CP_recvR,MPI_DOUBLE_PRECISION,
     &                neighl,Tag+6,comm3d,requestvr,ierror)
       call MPI_IRECV(CP_Rw,CP_recvR,MPI_DOUBLE_PRECISION,
     &                neighl,Tag+7,comm3d,requestwr,ierror)
```

```
  call MPI_SEND(CP_Sx,CP_sendR,MPI_DOUBLE_PRECISION,neighr,Tag+2,
&             comm3d,ierror)
  call MPI_SEND(CP_Sy,CP_sendR,MPI_DOUBLE_PRECISION,neighr,Tag+3,
&             comm3d,ierror)
  call MPI_SEND(CP_Sz,CP_sendR,MPI_DOUBLE_PRECISION,neighr,Tag+4,
&             comm3d,ierror)
  call MPI_SEND(CP_Su,CP_sendR,MPI_DOUBLE_PRECISION,neighr,Tag+5,
&             comm3d,ierror)
  call MPI_SEND(CP_Sv,CP_sendR,MPI_DOUBLE_PRECISION,neighr,Tag+6,
&             comm3d,ierror)
  call MPI_SEND(CP_Sw,CP_sendR,MPI_DOUBLE_PRECISION,neighr,Tag+7,
&             comm3d,ierror)

  call MPI_WAIT(requestxr,istatus,ierror)
  call MPI_WAIT(requestyr,istatus,ierror)
  call MPI_WAIT(requestzr,istatus,ierror)
  call MPI_WAIT(requestur,istatus,ierror)
  call MPI_WAIT(requestvr,istatus,ierror)
  call MPI_WAIT(requestwr,istatus,ierror)
```

```
      do i = 1,CP_recvR
            ipar=ipar+1
            x(ipar)=CP_Rx(i)
            y(ipar)=CP_Ry(i)
            z(ipar)=CP_Rz(i)
            u(ipar)=CP_Ru(i)
            v(ipar)=CP_Rv(i)
            w(ipar)=CP_Rw(i)
      end do

c  send particles to the Left (or Front or Bottom)
      do i = 1,iparL
       CP_Sx(i)=CLx(i)
       CP_Sy(i)=CLy(i)
       CP_Sz(i)=CLz(i)
       CP_Su(i)=CLu(i)
       CP_Sv(i)=CLv(i)
       CP_Sw(i)=CLw(i)
      end do
      CP_sendL = iparL

      call MPI_IRECV(CP_recvL,1,MPI_INTEGER,neighr,Tag+1,
     &                comm3d,requestp,ierror)
      call MPI_SEND(CP_sendL,1,MPI_INTEGER,neighl,Tag+1,
     &               comm3d,ierror)

      call MPI_WAIT(requestp,istatus,ierror)

c  then send particles
      call MPI_IRECV(CP_Rx,CP_recvL,MPI_DOUBLE_PRECISION,
     &                neighr,Tag+2,comm3d,requestxl,ierror)
      call MPI_IRECV(CP_Ry,CP_recvL,MPI_DOUBLE_PRECISION,
     &                neighr,Tag+3,comm3d,requestyl,ierror)
      call MPI_IRECV(CP_Rz,CP_recvL,MPI_DOUBLE_PRECISION,
     &                neighr,Tag+4,comm3d,requestzl,ierror)
      call MPI_IRECV(CP_Ru,CP_recvL,MPI_DOUBLE_PRECISION,
     &                neighr,Tag+5,comm3d,requestul,ierror)
      call MPI_IRECV(CP_Rv,CP_recvL,MPI_DOUBLE_PRECISION,
     &                neighr,Tag+6,comm3d,requestvl,ierror)
```

```
 call MPI_IRECV(CP_Rw,CP_recvL,MPI_DOUBLE_PRECISION,
&                neighr,Tag+7,comm3d,requestwl,ierror)

 call MPI_SEND(CP_Sx,CP_sendL,MPI_DOUBLE_PRECISION,neighl,Tag+2,
&               comm3d,ierror)
 call MPI_SEND(CP_Sy,CP_sendL,MPI_DOUBLE_PRECISION,neighl,Tag+3,
&               comm3d,ierror)
 call MPI_SEND(CP_Sz,CP_sendL,MPI_DOUBLE_PRECISION,neighl,Tag+4,
&               comm3d,ierror)
 call MPI_SEND(CP_Su,CP_sendL,MPI_DOUBLE_PRECISION,neighl,Tag+5,
&               comm3d,ierror)
 call MPI_SEND(CP_Sv,CP_sendL,MPI_DOUBLE_PRECISION,neighl,Tag+6,
&               comm3d,ierror)
 call MPI_SEND(CP_Sw,CP_sendL,MPI_DOUBLE_PRECISION,neighl,Tag+7,
&               comm3d,ierror)

 call MPI_WAIT(requestxl,istatus,ierror)
 call MPI_WAIT(requestyl,istatus,ierror)
 call MPI_WAIT(requestzl,istatus,ierror)
 call MPI_WAIT(requestul,istatus,ierror)
call MPI_WAIT(requestvl,istatus,ierror)
      call MPI_WAIT(requestwl,istatus,ierror)

      do i = 1,CP_recvL
              ipar=ipar+1
              x(ipar)=CP_Rx(i)
              y(ipar)=CP_Ry(i)
              z(ipar)=CP_Rz(i)
              u(ipar)=CP_Ru(i)
              v(ipar)=CP_Rv(i)
              w(ipar)=CP_Rw(i)
      end do

      CP_sendR=0
      CP_sendL=0
      CP_recvR=0
      CP_recvL=0
     end do
```

```
 return
      end
c   **********************************************************************

      include 'diagperp15n.f'
```