Computational Methods for Kinetic Processes in Plasma Physics

Ken Nishikawa

Department of Physics/UAH

June 3, 2015

Context

- Structure of Tristan code

  Subroutines

- Initial settings (inputperpN15nSS.f)

  Jet simulations

  Reconnections

Structure of Tristan code: Subroutines

Main program
- main loop
- dump data for rerun and diagnostics
- accumulate data for radiation

input program (include)
- set parameters for simulations (density, jet velocity, magnetic field, etc)
- load particles as initial conditions

diagnostic program (include) (not active)

Initial settings

Jet simulations

```
c Version for the jet studies, based on 3D-periodic code for magnetic field
c generation studies at SNR shocks (warm beam case). Non-periodic boundary
c conditions for the jet direction (x-direction)

c DIFFERENCES with a serial version and an OpenMP version by Ken include:
c 1. Umeda 1-st order method for current deposition (can be easily changed
c    to Buneman-Villasenor method if required)
c 2. different output file format for file naming and data dump: now each
c    processor writes data to a separate file
c 3. data (double precision) coded with 16-bit integers, to save the disk
c    space - needs to be converted back to double prec. for post-processing
c 4. smoothing of currents is taken out of depsit routines and applied after
c    particle splitting
c 5. particle sorting in order they are stored initially in the memory is
c    applied for better performance
c number of processors
```

```
  parameter (Nproc=4)
  parameter (Npx=1,Npy=2,Npz=2)

c  VIRTUAL PARTICLE array is distributed among processes so that each proces
c  works on nFx*nFy*nFz cells (its subdomain);

c !!! y and z domain size MUST BE EQUAL in the present setup: nFy=nFz !!!

c  FIELD arrays have mFi (mFx,mFy,mFz) elements in each dimension:
c  nFi plus 3 ghost cells on the "Right" and 2 ghost cells on the "Left"
  parameter (mx=645,my=11,mz=11)

  parameter (nFx=(mx-5)/Npx,mFx=nFx+5)
  parameter (nFy=(my-5)/Npy,mFy=nFy+5)
  parameter (nFz=(mz-5)/Npz,mFz=nFz+5)
```

```
c depending the size of domain in a processor
      parameter (nptl=2000000)
      parameter (mb=nptl,mj=1800000)

c  size of particle communication buffer arrays
c ** size is set as for ambient particles that move in one general direction  **
c !! check for particle number moving perpendicular to jet flow in Left proc. !!
c **      mpass = nFy*nFz*c*DT*adens + 10%-20%                               **
c ** must be reset for different particle densities                       **

      parameter (mpass=500000)

      parameter (mdiag=2500)

c  2**15-1
      parameter (imax=32767)

      integer size,myid,ierror
      integer lgrp,comm3d
      integer topol

      integer dims(3),coords(3)
      logical isperiodic(3),reorder

      integer FBDRx,FBDRy,FBDRz
      integer FBDLx,FBDLy,FBDLz
      integer FBDRxe,FBDLxe
      integer FBDRxp,FBDLxp
      integer FBD_BRx,FBD_BRy,FBD_BRz
      integer FBD_BLx,FBD_BLy,FBD_BLz
      integer FBD_ERx,FBD_ERy,FBD_ERz
      integer FBD_ELx,FBD_ELy,FBD_ELz

c  temporary arrays in 16-bit integers for data dump
      integer*2 irho,ifx,ify,ifz,ixx,iyy,izz

      real mi,me
      character strb1*6,strb2*6,strb3*6,strb4*6,strb5*6,strb6*6
      character strj1*6,strj2*6,strj3*6,strj4*6,strj5*6,strj6*6
      character strfb*5,strfe*5
      character strin*8
      character dir*28,num*3,st01,st02,st03,st0*3,st*4,hyph
      character step*6,step1,step2,step3,step4,step5
      character soutb*3,soute*3,soutd*5,soutv*4
      character ndiag*6,nfield*7
      character num1,num2,num3
      character strpd*6

c  electric and magnetic field arrays
      dimension ex(mFx,mFy,mFz),ey(mFx,mFy,mFz),ez(mFx,mFy,mFz)
      dimension bx(mFx,mFy,mFz),by(mFx,mFy,mFz),bz(mFx,mFy,mFz)

c  electric field longitudinal increaments (currents) arrays
      dimension dex(mFx,mFy,mFz),dey(mFx,mFy,mFz),dez(mFx,mFy,mFz)

c  ambient ions and electrons positions and velocities arrays
      dimension xi(mb),yi(mb),zi(mb),ui(mb),vi(mb),wi(mb)
      dimension xe(mb),ye(mb),ze(mb),ue(mb),ve(mb),we(mb)
```

```
c  jet ions and electrons positions and velocities arrays
      dimension xij(mj),yij(mj),zij(mj),uij(mj),vij(mj),wij(mj)
      dimension xej(mj),yej(mj),zej(mj),uej(mj),vej(mj),wej(mj)

c  diagnostic arrays (for quantities recorded on the grid)
cWR      dimension flx(mFx,mFy,mFz),fly(mFx,mFy,mFz),flz(mFx,mFy,mFz)
cWR      dimension rho(mFx,mFy,mFz)

c  diagnostic arrays for velocity distribution
      dimension Cvpar(mdiag),Cvper(mdiag),xpos(mdiag)

c  integer*2 arrays
      dimension ifx(mFx,mFy,mFz),ify(mFx,mFy,mFz),ifz(mFx,mFy,mFz)
      dimension irho(mFx,mFy,mFz)
      dimension ixx(mb),iyy(mb),izz(mb)

c  temporary arrays for jet injection
c       dimension Cseli(msel),Csele(msel)

      dimension ipsend(4),iprecv(Nproc*4)
```

```
 cCOL embed "topology" calculation
      dimension itops(3),itopr(Nproc*3)
      dimension topol(0:(Nproc-1),3)

c  smoother array
      dimension sm(-1:1,-1:1,-1:1)
c  smoother arrays for combined digital filtering
      dimension sm1(-1:1,-1:1,-1:1),sm2(-1:1,-1:1,-1:1)
      dimension sm3(-1:1,-1:1,-1:1)
      dimension nfilt(16)

c  initialize MPI
c ** "size" must be equal "Nproc"; "myid" is a ID for each process **
      common /pparms/ lgrp,comm3d

      lgrp = MPI_COMM_WORLD

      call MPI_INIT(ierror)
      call MPI_COMM_SIZE(lgrp,size,ierror)
      call MPI_COMM_RANK(lgrp,myid,ierror)
```

```
c  define virtual topology - assign processes to the domains
c  3D Cartesian decomposition
c  number of processes in each direction
      dims(1)=Npx
      dims(2)=Npy
      dims(3)=Npz
c  indicate whether the processes at the "ends" are connected
c ** y and z directions are periodic **
cJET
      isperiodic(1)=.false.
      isperiodic(2)=.true.
      isperiodic(3)=.true.
c  changing "reorder" to .true. allows MPI to reorder processes for better
c  performance - in accordance to the underlying hardware topology
      reorder      =.false.
c  number of dimensions
      ndim=3
```

```
c  create Cartesian decomposition
c  new communicator "comm3d" created - it must be used for communication
c ** in test simulations processes in "comm3d" have the same rank as in **
c ** MPI_COMM_WORLD (is this feature portable to different computers?)  **
      call MPI_CART_CREATE(lgrp,ndim,dims,isperiodic,reorder,
     &                            comm3d,ierror)

c  find Cartesian coordinates of the process (one can use also "MPI_CART_GET")
c ** coordinates run from 0 to Npi-1 **
      call MPI_CART_COORDS(comm3d,myid,3,coords,ierror)
c     call MPI_CART_GET(comm3d,3,dims,isperiodic,coords,ierror)

c  find ID of the (6) neighbors to the process (through the surfaces)
c ** communication is set up in the way that it requires contacting 6 closest **
c ** neighbors only                                                       **
c ** second argument in "MPI_CART_SHIFT" is direction, third is shift=1 **
      call MPI_CART_SHIFT(comm3d,0,1,nleft,nright,ierror)
      call MPI_CART_SHIFT(comm3d,1,1,nfront,nrear,ierror)
      call MPI_CART_SHIFT(comm3d,2,1,nbottom,ntop,ierror)
      if (nFy.ne.nFz) then
       if (myid.eq.1) print *,'Y and Z DOMAIN SIZES MUST BE EQUAL !!!'
       goto 9999
      end if

      call smoother(sm)

c  defining the characters for filenames
      strfb = 'fldb_'
      strfe = 'flde_'

      strb1 = 'pambx_'
      strb2 = 'pamby_'
      strb3 = 'pambz_'
      strb4 = 'pambu_'
      strb5 = 'pambv_'
      strb6 = 'pambw_'
      strj1 = 'pamjx_'
      strj2 = 'pamjy_'
      strj3 = 'pamjz_'
      strj4 = 'pamju_'
      strj5 = 'pamjv_'
      strj6 = 'pamjw_'

      strin = 'inparam_'

c  diagnostics and partial output files
      soutb = 'bf_'
      soute = 'ef_'
c       soutd = 'dens_'
      soutd = 'diag_'
      soutv = 'vel_'

c continued
```

```
cWR  now only this string for partial data files
      strpd = 'partd_'

      ndiag = 'number'
      nfield = 'bfield_'

cCOL      dir  = '/var/scratch/niemiec/beam11/'
cCOL      num = char(int(myid/10.)+48)//char(myid-int(myid/10.)*10+48)

      num1 = char(int(myid/100.)+48)
      num2 = char(int((myid-int(myid/100.)*100)/10.)+48)
      num3 = char(int(myid-int(myid/10.)*10)+48)
      num = num1//num2//num3

c  up to which time-step do calculations?
      last0 = 50
      last=2
c       last = 500
c       last = 1000
```

```
c      last = 2000
c      last = 3000
c      last = 4000
c      last = 4500
c      last = 5000
c      last = 8000
c      last = 10000
c      last = 37500
c    last = 12500   nst = 25
c      last = 15000
c       last = 19000
c       last = 20000
c      last = 24000
c      last = 25000
c      last = 26000
c      last = 28000
c       last = 30000
c      last = 32000
c      last = 36000
c      last = 40000
```

```
c       last = 44000
c       last = 48000
c       last = 52000
c       last = 56000
c       last = 60000
c       last = 64000


c  number of time-step at which last data was correctly dumped
c ** nst=0 for the start run, and appropriate number for continuation runs **
      nst = 0
c       nst = 1
c       nst = 5
c       nst = 8
c       nst = 10
c       nst = 14
c       nst = 19
c       nst = 20
c   somehow problem with partd_036_025
c       nst = 25
```

```
c      nst = 24
c      nst = 30
c       nst = 40
c      nst = 48
c      nst = 50
c      nst = 56
c       nst = 60
c      nst = 64
c      nst = 72
c      nst = 80
c      nst = 88

      hyph='_'
      st01 = char(int(nst/100.)+48)
      st02 = char(int((nst-int(nst/100.)*100)/10.)+48)
      st03 = char(int(nst-int(nst/10.)*10)+48)
      st0 = st01//st02//st03
```

```
c       st0 = char(int(nst/10.)+48)//char(nst-int(nst/10.)*10+48)
      st = hyph//st0

c  either initialize the parameters (new simulation) or read them from disk files
      if (last.gt.last0) goto 1024
c  informations on initial parameters and the simulation progress are written
c  to a single file accessed by the second (id=1) process only
       if (myid.eq.1) then
              open(1,file='out',status='new')
       end if

cCOL calculate topology
       do i = 1,3
        itops(1)=coords(1)
        itops(2)=coords(2)
        itops(3)=coords(3)
       end do
```

```
  call MPI_GATHER(itops,3,MPI_INTEGER,itopr,3,MPI_INTEGER,
&                 0,comm3d,ierror)

  if (myid .eq. 0) then
   k=0
   do i = 0,Nproc-1
    do j = 1,3
           k = k+1
           topol(i,j)=itopr(k)
          end do
   end do

   open(3,file="topology")
   do i = 0,Nproc-1
     write(3,*) i,topol(i,1),topol(i,2),topol(i,3)
   end do
   close(3)
  end if
```

```
         open(25,file=ndiag//num,status='new')

cc       open(26,file=nfield//num,status='new')
cc       close(26)


       nstep = 0

c  INITIALIZE THE PARAMETERS
c  electron and ion charge
c      qe=-.005
c NewKen
       qe=-.01
       qi=-qe

c  electron and ion mass
c      me=0.25
c NewKen
       me=0.12
        mi=me*20.0
```

```
c NewKen   electron-positron plasma
c      mi=me*1.0

c  ratio charge/mass
       qme=qe/me
       qmi=qi/mi

c  speed of light (c satifies Courant condition for stability)
c       c=.5
c NewKen
       c=1.0

c  time-step (this must be set according to the jet velocity)
c       DT=0.25
c NewKen
       DT=0.1
c       DT=0.025
c for jitter
c       DT=0.005
```

```
c new attention the definition of jet vel needs to be here
cJET  jet velocity for electrons and ions
c      vijet = 0.9968*c
      vijet = 0.99778*c
      vejet = vijet

c  homogenous magnetic field component
      b0x=0.0*c
c new attention
      b0y=0.1*c
c      b0y=0.01*c
c new to make sure the basic structure is correct
      b0y=0.00*c
c new the minus sign is wrong?
      e0z=-vijet*b0y
c      e0z= vijet*b0y

c  number of filterings applied
      nsmooth = 13
cJsm        nsmooth = 1
```

```
c  smoother arrays for given filtering function (second argument)
c       call smoother1(sm1, 0.5)
c       call smoother1(sm2,-1./6)
c       call smoother1(sm3, 0.5)
      call smoother1(sm1, 1.0)
      call smoother1(sm2,-0.305)
      call smoother1(sm3, 0.5)
cJsm        call smoother1(sm1, 0.5)

c  combined filtering profile "nfilt"
c       do i=1,2
      do i=1,8
       nfilt(i)=1
      end do

c       do i=3,4
      do i=9,12
       nfilt(i)=2
      end do
      nfilt(13)=3
cJsm       nfilt(1)=1

c  initialize electric and magnetic fields
c new attention
      call Field_init(bx,by,bz,ex,ey,ez,dex,dey,dez,mFx,mFy,mFz,
     &   b0x,b0y,e0z,c)

c  particle position boundaries in each dimension
      PBLeft = coords(1)*nFx+3.0
      PBRght = (coords(1)+1)*nFx+3.0
      PBFrnt = coords(2)*nFy+3.0
      PBRear = (coords(2)+1)*nFy+3.0
      PBBot  = coords(3)*nFz+3.0
      PBTop  = (coords(3)+1)*nFz+3.0

c  global particle boundaries for x-direction (nonperiodic boundary conditions)
cJET
```

```
c4push
       if (dims(1).eq.1) then
        GBLeft = 1.0
        GBRght = mx
       else
       if (coords(1).eq.0) then
        GBLeft = 1.0
        GBRght = PBRght
       else if (coords(1).eq.(Npx-1)) then
        GBLeft = PBLeft
        GBRght = mx
       else
                   GBLeft = PBLeft
        GBRght = PBRght
       end if
       end if

c  offset from the particle's nearest grid point in a VIRTUAL array
       DHDx = PBLeft-3.0
       DHDY = PBFrnt-3.0
       DHDz = PBBot -3.0
```

```
c  boundaries for field arrays elements advanced in field pusher
c ** elements inside particle domain are calculated (not in ghost cells)     **
c ** the first (the last) processes advance the fields in their left (right) **
c ** ghost cells to provide appropriate field elements to MOVER and for      **
c ** proper handling of boundary conditions for the fields (SURFACE)         **
      FBD_BLx = 3
      FBD_ELx = 3
      FBD_BRx = nFx+2
      FBD_ERx = nFx+2

      FBD_BLy = 3
      FBD_ELy = 3
      FBD_BRy = nFy+2
      FBD_ERy = nFy+2

      FBD_BLz = 3
      FBD_ELz = 3
      FBD_BRz = nFz+2
      FBD_ERz = nFz+2
```

```
cJET
      if(coords(1).eq.0)then
                FBD_BLx = 1
                FBD_ELx = 2
      end if

      if(coords(1).eq.(Npx-1))then
                FBD_BRx = nFx+4
                FBD_ERx = nFx+5
      end if

c  indices of B and E field arrays elements that are passed between processes
c  in "Field_passing" subroutine (only these needed by MOVER and field pushers)
      FBDLx = 3
      FBDRx = nFx+2
      FBDLy = 3
      FBDRy = nFy+2
      FBDLz = 3
      FBDRz = nFz+2

      adenj = float
```

```
c  indices needed in "E_Field_Passing_Add" subroutine
      FBDLxe = FBDLx
      FBDRxe = FBDRx

c  indices needed in "Field_passing2" subroutine
      FBDLxp = FBDLx
      FBDRxp = FBDRx
cJET  left in case 2-nd order shapes are used
c4push
      if (coords(1).eq.(Npx-1)) FBDRxp=nFx+3

      if (coords(1).eq.(Npx-1)) FBDRx=nFx+4
      if (coords(1).eq.0) FBDLx=2

c  bounds of buffer arrays in "Field_passing" subroutine
      mc  = FBDRy+1
      mrl = FBDLx-1
      mrh = max(FBDRy+1,FBDRx+1)
```

```
c  bounds of buffer arrays in "Field_passing2" subroutine
      mc2  = FBDRy+2
      mrh2 = max(FBDRy+2,FBDRxp+2)

c  bounds of buffer arrays in "E_Field_Passing_Add" subroutine
      mcol = mFy
      mrow = max(mFy,mFx)


c  Parameters for particle initialization and boundary conditions
      ions=0
      lecs=0
      ionj=0
      lecj=0

c  seed value for random number generator (different for each process)
c  ** must be negative integer **
      is = -(myid+1)
      isis=is
```

```
c  reflection rates for electrons and ions
c  ** all particles are reflected **
      refle = 0.0
      refli = 0.0

c  portion of particle population selected for diagnostics
c        rselect = 0.007
      rselect = 0.00125

c  ambient particle number density per cell per species
c       dens = 27.0
c NewKen
      dens = 12.0

cJET jet injection xboundary
      xinj = 25.0
c NewKen jet density
      densj = 8.0
```

```
c new attention the definition of jet vel needs to be before
cJET  jet velocity for electrons and ions
c      vijet = 0.9968*c
c      vijet = 0.99778*c
c      vejet = vijet
c new attention
c new Jacek
c!       e0z = -vejet*b0y

      gamjet = 1.0/sqrt(1.0-(vijet/c)**2)

c  ambient particle thermal velocities for the same el. and ion temperature
c ** thermal velocity as provided is the most common speed for the Maxwell   **
c ** velocity distribution: variance of a velocity component = vethml/sqrt(2)**
cJET
      vethml=0.05
      vithml=vethml/sqrt(mi/me)
```

```
cJET jet particles thermal dispersion
c ** multipled by sqrt(2.) to provide variance corresponding to Ken's setting **
      vethmj = 0.01*c*sqrt(2.)
      vithmj = vethmj/sqrt(mi/me)

c  ambient particle density setting for homogenous plasma
      lx = nFx
      ly = nFy
      lz = nFz

      denx = dens**(1./3)
      deny = denx
      denz = denx

      lx1 = lx*denx
      ly1 = ly*deny
      lz1 = lz*denz

      dlx = float(lx)/lx1
      dly = float(ly)/ly1
      dlz = float(lz)/lz1
```

```
c  adjusted ambient particle number density in cell
      adens = float(lx1*ly1*lz1)/(lx*ly*lz)

cJET
c now for jet particles (flat jet)
c  here assume homogenous density and injectioning every 3rd time-step
c       deljx = vijet*DT*3.0
c       deljy = deljx
c       deljz = deljx

c       denjx = 1.0/deljx
      denjx = densj**(1./3)
      denjy = denjx
      denjz = denjx
c NewKen revised
c       deljx = 1./denjx
c       deljy = deljx
c       deljz = deljx

      lxj1 = lx*denjx
      lyj1 = ly*denjy
      lzj1 = lz*denjz
      dlxj = float(lx)/lxj1
      dlyj = float(ly)/lyj1
      dlzj = float(lz)/lzj1

      njskip = int(dlxj/(vejet*DT))

      dlxj = vejet*DT*float(njskip)


c  buffer zone in x-direction must be larger when multiple filtering is applied
c  inject ambient particles few cells apart from left boundary and keep them
c  also few cells apart from right boundary - depending on number of smoothing
c  "nsmooth"; Virtual box particle x-boundaries are then changed, and "PVLeft"
c  and "PVRght" become parameters of "Split..." subroutines, where boundary
c  conditions for particles are applied
      if (nsmooth.eq.1) then
       napart = 0
      else
       napart = int((nsmooth-1)/dlx) + 1
      end if
```

```
cJET
      PVLeft = 3.0 + napart*dlx
      PVRght = mx - 2.0 - napart*dlx

      if (coords(1).eq.0) then
c4th         x0  = PBLeft - 0.5*dlx + napart*dlx
       x0  = PBLeft - 0.5*dlx + napart*dlx + 10.0
               lx1 = lx1 - napart
      else
       x0 = PBLeft - 0.5*dlx
              if (coords(1).eq.(Npx-1)) lx1 = lx1 - napart
      end if

      y0 = PBFrnt - 0.5*dly
      z0 = PBBot  - 0.5*dlz

c JET jet injection plane moved to the right depending on number of smoothing
c4push       xj0 = xinj + napart*dlx - 0.5*dlxj
c4push inject jet right in front of ambient plasma
      xj0 = x0 - 0.5*dlxj
      yj0 = PBFrnt - 0.5*dlyj
      zj0 = PBBot  - 0.5*dlzj
```

```
cJET ambient plasma rest frame
c  initialize ambient plasma particle positions and velocities
      call Particle_init(ions,lecs,mb,vithml,vethml,
cJsm     &                      PBLeft,PBRght,PBFrnt,PBRear,PBBot,PBTop,
     &                   PVLeft,PVRght,PBFrnt,PBRear,PBBot,PBTop,
     &                    x0,y0,z0,dlx,dly,dlz,lx1,ly1,lz1,
     &               xi,yi,zi,ui,vi,wi,xe,ye,ze,ue,ve,we,c,is)


         write(25,111) nstep,ions,lecs,ionj,ionj-ionj0,
     &                       lecj,lecj-lecj0
         close(25)
         ionj0=ionj
         lecj0=lecj


c  Miscellaneous useful parameters (written to "out")

c  electron and ion plasma frequency
      wpi=sqrt(qi*qi*adens/mi)
      wpe=sqrt(qe*qe*adens/me)
```

```
c  electron and ion cyclotron frequency in a regular field
c       wce=qe*b0x/(me*c)
c       wci=qi*b0x/(mi*c)
c new attention
c new Jacek
      b0 = sqrt(b0x**2+b0y**2)
      wce=qe*b0/me
      wci=qi*b0/mi

c  cyclotron-plasma frequency ratio
      wecp=wce/wpe
      wicp=wci/wpi

c  ion-electron mass ratio
      ratiom=mi/me

c  electron and ion temperature and temperature ratio
      te=0.5*me*vethml*vethml
      ti=0.5*mi*vithml*vithml
      tie=ti/te
```

```
c  electron Debye length
      debye=vethml/wpe

c  electron skin depth
      skind=c/wpe

c  Alfven velocity
c       valfc=wicp
c       valfc=c/(1.+1./wicp)
      valfnr = b0x*c/sqrt(adens*(me+mi))
cJET       valfnr = b0x*c/sqrt(adens*(mi+me)+adnrc*me)
c       valfc  = c/sqrt(1.0 + (c/valfnr)**2)
      valfc = valfnr

c  plasma betta cveth=c/vethml
      beta=2.0/((c/vethml)**2*wecp**2)
c new attention (if b0x = b0y = 0, wce = 0)
      rhoe=vethml/wce
      rhoi=vithml/wci
```

```
c  time step in units of plasma frequency
      deltm=wpe*DT

      if (myid.eq.1) then
       write(1,100) Nproc
             write(1,1009) Npx,Npy,Npz
       write(1,101) mx,my,mz,nFx,mFx,nFy,mFy,nFz,mFz
             write(1,1011) nsmooth
       write(1,102) 2*(mb+mj),mb,mj,mpass
c new attention
       write(1,103) b0x, b0y, e0z
       write(1,104) vejet, gamjet, vethmj, vithmj
cJET         write(1,104) vejet, gamjet
cJET         write(1,1044) vcre
c        write(1,105) avdenj, adenj, dlx, dly, dlz, x0-0.5*dlx, irank
c            write(1,105) adenj, dljx, dljy, dljz, xinj, rthml, rthmlj
             write(1,105) xinj, dncr, cmin
c       write(1,1055) dens, adens, dlx, dly, dlz,lx1,ly1,lz1
             write(1,1055) dens, adens, densj, adenj, njskip,
     1       xj0, dlx, dly, dlz,lx1,ly1,lz1,
     1       dlxj, dlyj, dlzj,lxj1,lyj1,lzj1
```

```
          write(1,1056) PVLeft,PVRght,napart
  end if

  if(myid.eq.0) print *,'after init'

  if (myid.eq.1) then
          write(1,106) qe,qi,me,mi,qme,qmi,ratiom,c,DT,deltm,
&                 refle,refli,rselect
   write(1,107) wpe, wpi, wce, wci, wecp, wicp
   write(1,108) vethml, vithml, valfc, beta
   write(1,109) debye, skind, rhoe, rhoi
   write(1,110) te,ti,tie
          write(1,*) '**************************************************'
          close(1)
          open(1,file='out',status='old',position='append')
  end if

 goto 1025
```

```
cCOL !!!
100   format('Nproc=',i4)
1009  format(/ 'Npx=',i3,2x,'Npy=',i3,2x,'Npz=',i3)
101   format(/ 'mx=',i4,2x,'my=',i4,2x,'mz=',i4
    &       / 'nFx=',i5,2X,'mFx=',i5
    &       / 'nFy=',i3,2X,'mFy=',i3
    &       / 'nFz=',i3,2X,'mFz=',i3)
1011  format(/ 'nsmooth=',i2)

102   format(/ 'nparticles=',i10
    &       / 'nambient=',i10,2x,'njet',i10,2x,'mpass=',i6)

c 103 format(/ 'b0x=',f8.5)
c new attention
103   format(/ 'b0x=',f8.5,'b0y=',f8.5,'e0z=',f8.5)

104   format(/1h 'vjet/c=',f9.6,2x,'gamma=',f9.6
    &       /1h 'vethmlj=',f7.5,2x,'vithmlj=',f8.6)
cJET104   format(/ 'vjet/c=',f9.6,2x,'gamma=',f9.6)
1044  format(/ 'vcr/c=',f9.6)
```

```
c105   format(/1h 'jet_dens=',f8.4
c     &       /1h 'dljx,dljy,dljz=',3(f8.5,2x)
c     &       /1h 'jet xinj=',f7.3,2x,'rthml=',f6.4,2x,'rthmlj=',f6.4)
105   format(/ 'CR xinj=',f7.3,2x,'dnCR=',f5.3,2x,'cmin=',f5.3)

cc1055   format(/1h 'amb_dens=',f8.4,2x,'adjusted dens=',f8.4
cc     &        /1h 'dlx,dly,dlz=',3(f8.5,2x))
c NewKen
1055   format(/ 'amb_dens=',f8.4,2x,'adjusted dens=',f8.4
     &       / 'jet_dens=',f8.4,2x,'adjusted denj=',f8.4
     &       / 'jet_skip=',i5,2x,'xj0=',f8.4
     &       / 'dlx,dly,dlz=',3(f8.5,2x)
     &       / 'lx1,ly1,lz1=',3(i6,2X)
     &       / 'dlxj,dlyj,dlzj=',3(f8.5,2x)
     &       / 'lx1j,ly1j,lz1j=',3(i6,2X))
1056   format(/ 'PVLeft=',f6.3,2x,'PVRght=',f9.3,2x,'napart=',i3)

106   format(/ 'qe=',f11.6,2x,'qi=',f11.6,2x,'me=',f9.4,2x,'mi=',f9.4
     &      / 'qme=',f9.4,2x,'qmi=',f9.4,2x,'mi/me=',f8.3,
     &      // 'c=',f5.2,2x,'DT=',f7.4,2x,'wpeDT=',f7.4
     &      // 'refle=',f5.2,2x,'refli=',f5.2,' rselect=',f7.4)
```

```
107   format(/ 'wpe=',f7.5,2x,'wpi=',f8.6,2x,'wce=',f8.5,2x,
    &           'wci=',f9.6 /1h 'wecp=',f9.6,2x,'wicp=',f9.6)

108   format(/ 'vethml=',f7.5,2x,'vithml=',f8.6
    &       / 'valfven=',f8.5,2x,'beta= ',f9.6)

109   format(/ 'debye=',f6.4,2x,'skind=',f7.3
    &       / 'rhoe=',f9.4,2x,'rhoi=',f9.4)

110   format(/ 'Te=',e12.4,2x,'Ti=',e12.4,2x,'Ti/Te=',f7.4)

c111   format(1h i4,2x,2(i7,2x),2(i7,2x,i5,2x,i5,2x))
c111   format(i5,2x,2(i7,2x),2(i7,2x,i5,2x))
111   format(i5,2x,2(i8,2x),2(i8,2x,i5,2x))

c  read parameters for continuation run
1024   continue
      if (myid.eq.1) then
            open(1,file='out',status='old',position='append')
      end if
      open(7,file=strpd//num//st,form='unformatted')

c attention ken
      c=1.0
c after nst = 20 (--041, --042, --043 , - - -)
c    read c
c somehow it does not work with reading read(7) c
      read(7) c
      read(7) ions,lecs,ionj,lecj
      read(7) PBLeft,PBRght,PBFrnt,PBRear,PBBot,PBTop
      read(7) bxmax,bxmin,bymax,bymin,bzmax,bzmin
      read(7) ifx,ify,ifz
      call F_reconv(bx,by,bz,ifx,ify,ifz,mFx,mFy,mFz,imax,
     &              bxmax,bxmin,bymax,bymin,bzmax,bzmin)
      read(7) exmax,exmin,eymax,eymin,ezmax,ezmin
      read(7) ifx,ify,ifz
      call F_reconv(ex,ey,ez,ifx,ify,ifz,mFx,mFy,mFz,imax,
     &              exmax,exmin,eymax,eymin,ezmax,ezmin)
c  read ambient ions
      read(7) (ixx(i),i=1,ions),(iyy(i),i=1,ions),(izz(i),i=1,ions)
      call X_reconv(ions,xi,yi,zi,ixx,iyy,izz,mb,mb,imax,
     &          PBLeft,PBRght,PBFrnt,PBRear,PBBot,PBTop)
```

```
      read(7) umax,umin,vmax,vmin,wmax,wmin
      read(7) (ixx(i),i=1,ions),(iyy(i),i=1,ions),(izz(i),i=1,ions)
cNewJacek       call V_reconv(ions,ui,vi,wi,ixx,iyy,izz,mb,mb,imax,
cNewJacek     &                     umax,umin,vmax,vmin,wmax,wmin)
cNewJacek  also below
      call P_reconv(ions,ui,vi,wi,ixx,iyy,izz,mb,mb,imax,
     &                     umax,umin,vmax,vmin,wmax,wmin,c)
c  read ambient electrons
      read(7) (ixx(i),i=1,lecs),(iyy(i),i=1,lecs),(izz(i),i=1,lecs)
      call X_reconv(lecs,xe,ye,ze,ixx,iyy,izz,mb,mb,imax,
     &          PBLeft,PBRght,PBFrnt,PBRear,PBBot,PBTop)
ccPconv_corrected
c attention ken inorder to rerun temporary c is defined
c      read(7) c
c attention ken
c      c = 1.0
      read(7) umax,umin,vmax,vmin,wmax,wmin
      read(7) (ixx(i),i=1,lecs),(iyy(i),i=1,lecs),(izz(i),i=1,lecs)
      call P_reconv(lecs,ue,ve,we,ixx,iyy,izz,mb,mb,imax,
     &                     umax,umin,vmax,vmin,wmax,wmin,c)
```

```
c  read CR ions
      if (ionj.gt.0) then
       read(7) (ixx(i),i=1,ionj),(iyy(i),i=1,ionj),(izz(i),i=1,ionj)
       call X_reconv(ionj,xij,yij,zij,ixx,iyy,izz,mj,mb,imax,
     &              PBLeft,PBRght,PBFrnt,PBRear,PBBot,PBTop)
ccPconv_corrected
c attention ken inorder to rerun temporary c is defined
c      read(7) c
c      c = 1.0
       read(7) umax,umin,vmax,vmin,wmax,wmin
       read(7) (ixx(i),i=1,ionj),(iyy(i),i=1,ionj),(izz(i),i=1,ionj)
       call P_reconv(ionj,uij,vij,wij,ixx,iyy,izz,mj,mb,imax,
     &                        umax,umin,vmax,vmin,wmax,wmin,c)
      end if
c  read CR electrons
      if (lecj.gt.0) then
       read(7) (ixx(i),i=1,lecj),(iyy(i),i=1,lecj),(izz(i),i=1,lecj)
       call X_reconv(lecj,xej,yej,zej,ixx,iyy,izz,mj,mb,imax,
     &              PBLeft,PBRght,PBFrnt,PBRear,PBBot,PBTop)
       read(7) umax,umin,vmax,vmin,wmax,wmin
       read(7) (ixx(i),i=1,lecj),(iyy(i),i=1,lecj),(izz(i),i=1,lecj)
      call P_reconv(lecj,uej,vej,wej,ixx,iyy,izz,mj,mb,imax,
     &                         umax,umin,vmax,vmin,wmax,wmin,c)
      end if

      read(7) c,DT,qi,qe,mi,me,qmi,qme,vithml,vethml,vijet,vejet,
     &         vithmj,vethmj,refli,refle,rselect,xj0,yj0,zj0,b0x,
c new Jacek
     &         b0y,e0z,
     &         dlxj,dlyj,dlzj,lyj1,lzj1,
     &         mc,mrl,mrh,mc2,mrh2,mcol,mrow,isis
c NewKen
     &          ,njskip

      read(7) GBLeft,GBRght,DHDx,DHDy,DHDz,FBD_BLx,FBD_BRx,
     &         FBD_BLy,FBD_BRy,FBD_BLz,FBD_BRz,FBD_ELx,FBD_ERx,
     &         FBD_ELy,FBD_ERy,FBD_ELz,FBD_ERz,FBDLx,FBDLy,FBDLz,
     &         FBDRx,FBDRy,FBDRz,FBDLxe,FBDRxe,FBDLxp,FBDRxp,
     &         PVLeft,PVRght,nsmooth,sm1,sm2,sm3,nfilt,nstep


      close(7)
```

```
c  seed value for random number generator changed here
c ** "last" must be larger than "Nproc" to have each time different seed values **
cCOL       is = -(myid+1+last)
      is = -(myid+1+Nproc + 3)
      isis=is

      ionj0=ionj
      lecj0=lecj

1025  Continue

c end of the program
```