```fortran
!-----------------------------------------------------------------------
! Skeleton 2D Electrostatic MPI/OpenMP PIC code
! written by Viktor K. Decyk, UCLA
      program mppic2
      use mppush2_h
      use pplib2_h
      use omplib_h
      implicit none
      integer, parameter :: indx =   9, indy =   9
      integer, parameter :: npx =  3072, npy =   3072
      integer, parameter :: ndim = 2
      real, parameter :: tend = 10.0, dt = 0.1, qme = -1.0
      real, parameter :: vtx = 1.0, vty = 1.0, vx0 = 0.0, vy0 = 0.0
      real :: ax = .912871, ay = .912871
! idimp = dimension of phase space = 4
      integer :: idimp = 4, ipbc = 1
! idps = number of partition boundaries
      integer :: idps = 2
      real :: wke = 0.0, we = 0.0, wt = 0.0
! sorting tiles, should be less than or equal to 32
      integer :: mx = 16, my = 16
! fraction of extra particles needed for particle management
      real :: xtras = 0.2
! declare scalars for standard code
      integer :: nx, ny, nxh, nyh, nxe, nye, nxeh, nnxe, nxyh, nxhy
      integer :: mx1, ntime, nloop, isign, ierr
      real :: qbme, affp
      double precision :: np
!
! declare scalars for MPI code
      integer :: ntpose = 1
      integer :: nvp, idproc, kstrt, npmax, kxp, kyp, nypmx, nypmn
      integer :: nyp, noff, npp, nps, myp1, mxyp1
!
! declare scalars for OpenMP code
      integer :: nppmx, nppmx0, nbmaxp, ntmaxp, npbmx, irc
      integer :: nvpp

! declare arrays for standard code
      real, dimension(:,:), pointer :: part
      real, dimension(:,:), pointer :: qe
      real, dimension(:,:,:), pointer :: fxye
      complex, dimension(:,:), pointer :: qt
      complex, dimension(:,:,:), pointer :: fxyt
      complex, dimension(:,:), pointer :: ffc
      integer, dimension(:), pointer :: mixup
      complex, dimension(:), pointer :: sct
      real, dimension(4) :: wtot, work
!
! declare arrays for MPI code
      complex, dimension(:,:,:), pointer :: bs, br
      real, dimension(:,:), pointer :: sbufl, sbufr, rbufl, rbufr
      real, dimension(:), pointer  :: edges
      real, dimension(:), pointer  :: scs, scr
```

```
!
! declare arrays for OpenMP code
      real, dimension(:,:,:), pointer :: ppart, ppbuff
      integer, dimension(:), pointer :: kpic
      integer, dimension(:,:), pointer :: ncl
      integer, dimension(:,:,:), pointer :: iholep
      integer, dimension(:,:), pointer :: ncll, nclr, mcll, mclr
!
! declare and initialize timing data
      real :: time
      integer, dimension(4) :: itime
      real :: tdpost = 0.0, tguard = 0.0, ttp = 0.0, tfield = 0.0
      real :: tpush = 0.0, tsort = 0.0, tmov = 0.0
      real, dimension(2) :: tfft = 0.0
      double precision :: dtime
!
      irc = 0
! nvpp = number of shared memory nodes (0=default)
      nvpp = 0
!     write (*,*) 'enter number of nodes:'
!     read (5,*) nvpp
! initialize for shared memory parallel processing
      call INIT_OMP(nvpp)
!
! initialize scalars for standard code
      np =  dble(npx)*dble(npy)
      nx = 2**indx; ny = 2**indy; nxh = nx/2; nyh = ny/2
      nxe = nx + 2; nye = ny + 2; nxeh = nxe/2; nnxe = ndim*nxe
      nxyh = max(nx,ny)/2; nxhy = max(nxh,ny)
      mx1 = (nx - 1)/mx + 1
      nloop = tend/dt + .0001; ntime = 0
      qbme = qme
      affp = dble(nx)*dble(ny)/np
!
! nvp = number of distributed memory nodes
! initialize for distributed memory parallel processing
      call PPINIT2(idproc,nvp)
      kstrt = idproc + 1
! check if too many processors
      if (nvp > ny) then
         if (kstrt==1) then
         write (*,*) 'Too many processors requested: ny, nvp=', ny, nvp
         endif
         go to 3000
      endif
!
! initialize data for MPI code
      allocate(edges(idps))
! calculate partition variables: edges, nyp, noff, nypmx
! edges(1:2) = lower:upper boundary of particle partition
! nyp = number of primary (complete) gridpoints in particle partition
! noff = lowermost global gridpoint in particle partition
! nypmx = maximum size of particle partition, including guard cells
! nypmn = minimum value of nyp
```

```
      call PDICOMP2L(edges,nyp,noff,nypmx,nypmn,ny,kstrt,nvp,idps)
      if (nypmn < 1) then
         if (kstrt==1) then
            write (*,*) 'combination not supported nvp, ny =',nvp,ny
         endif
         go to 3000
      endif
!
! initialize additional scalars for MPI code
! kxp = number of complex grids in each field partition in x direction
      kxp = (nxh - 1)/nvp + 1
! kyp = number of complex grids in each field partition in y direction
      kyp = (ny - 1)/nvp + 1
! npmax = maximum number of electrons in each partition
      npmax = (np/nvp)*1.25
! myp1 = number of tiles in y direction
      myp1 = (nyp - 1)/my + 1; mxyp1 = mx1*myp1
!
! allocate and initialize data for standard code
      allocate(part(idimp,npmax))
      allocate(qe(nxe,nypmx),fxye(ndim,nxe,nypmx))
      allocate(qt(nye,kxp),fxyt(ndim,nye,kxp))
      allocate(ffc(nyh,kxp),mixup(nxhy),sct(nxyh))
      allocate(kpic(mxyp1))
!
! allocate and initialize data for MPI code
      allocate(bs(ndim,kxp,kyp),br(ndim,kxp,kyp))
      allocate(scs(nxe*ndim),scr(nxe*ndim))
!
! prepare fft tables
      call WPFFT2RINIT(mixup,sct,indx,indy,nxhy,nxyh)
! calculate form factors
      isign = 0
      call MPPOIS22(qt,fxyt,isign,ffc,ax,ay,affp,we,nx,ny,kstrt,nye,kxp,&
     &nyh)
! initialize electrons
      nps = 1
      npp = 0
      call PDISTR2(part,edges,npp,nps,vtx,vty,vx0,vy0,npx,npy,nx,ny,    &
     &idimp,npmax,idps,ipbc,ierr)
! check for particle initialization error
      if (ierr /= 0) then
         if (kstrt==1) then
            write (*,*) 'particle initialization error: ierr=', ierr
         endif
         go to 3000
      endif
!
! find number of particles in each of mx, my tiles: updates kpic, nppmx
      call PPDBLKP2L(part,kpic,npp,noff,nppmx,idimp,npmax,mx,my,mx1,    &
     &mxyp1,irc)
      if (irc /= 0) then
         write (*,*) 'PPDBLKP2L error, irc=', irc
         stop
      endif
! allocate vector particle data
      nppmx0 = (1.0 + xtras)*nppmx
      ntmaxp = xtras*nppmx
      npbmx = xtras*nppmx
      nbmaxp = 0.25*mx1*npbmx
      allocate(sbufl(idimp,nbmaxp),sbufr(idimp,nbmaxp))
      allocate(rbufl(idimp,nbmaxp),rbufr(idimp,nbmaxp))
      allocate(ppart(idimp,nppmx0,mxyp1))
      allocate(ppbuff(idimp,npbmx,mxyp1))
      allocate(ncl(8,mxyp1))
      allocate(iholep(2,ntmaxp+1,mxyp1))
      allocate(ncll(3,mx1),nclr(3,mx1),mcll(3,mx1),mclr(3,mx1))
!
! copy ordered particle data for OpenMP
      call PPPMOVIN2L(part,ppart,kpic,npp,noff,nppmx0,idimp,npmax,mx,my,&
     &mx1,mxyp1,irc)
      if (irc /= 0) then
         write (*,*) kstrt, 'PPPMOVIN2L overflow error, irc=', irc
         call PPABORT()
         stop
      endif
! sanity check
      call PPPCHECK2L(ppart,kpic,noff,nyp,idimp,nppmx0,nx,mx,my,mx1,myp1&
     &,irc)
      if (irc /= 0) then
         write (*,*) kstrt, 'PPPCHECK2L error: irc=', irc
         call PPABORT()
         stop
      endif
!
! * * * start main iteration loop * * *
!
  500 if (nloop <= ntime) go to 2000
!     if (kstrt==1) write (*,*) 'ntime = ', ntime
!
! deposit charge with standard procedure: updates qe
      call dtimer(dtime,itime,-1)
      qe = 0.0
      call PPGPPOST2L(ppart,qe,kpic,noff,qme,idimp,nppmx0,mx,my,nxe,    &
     &nypmx,mx1,mxyp1)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tdpost = tdpost + time
!
! add guard cells with standard procedure: updates qe
      call dtimer(dtime,itime,-1)
      call PPAGUARD2XL(qe,nyp,nx,nxe,nypmx)
      call PPNAGUARD2L(qe,scr,nyp,nx,kstrt,nvp,nxe,nypmx)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tguard = tguard + time
!
! transform charge to fourier space with standard procedure: updates qt
! modifies qe
      call dtimer(dtime,itime,-1)
      isign = -1
      call WPPFFT2RM(qe,qt,bs,br,isign,ntpose,mixup,sct,ttp,indx,indy,  &
     &kstrt,nvp,nxeh,nye,kxp,kyp,nypmx,nxhy,nxyh)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tfft(1) = tfft(1) + time
      tfft(2) = tfft(2) + ttp
!
! calculate force/charge in fourier space with standard procedure:
! updates fxyt
      call dtimer(dtime,itime,-1)
      isign = -1
      call MPPOIS22(qt,fxyt,isign,ffc,ax,ay,affp,we,nx,ny,kstrt,nye,kxp,&
     &nyh)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tfield = tfield + time
!
! transform force to real space with standard procedure: updates fxye
! modifies fxyt
      call dtimer(dtime,itime,-1)
      isign = 1
      call WPPFFT2RM2(fxye,fxyt,bs,br,isign,ntpose,mixup,sct,ttp,indx,  &
     &indy,kstrt,nvp,nxeh,nye,kxp,kyp,nypmx,nxhy,nxyh)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tfft(1) = tfft(1) + time
      tfft(2) = tfft(2) + ttp
!
! copy guard cells with standard procedure: updates fxye
      call dtimer(dtime,itime,-1)
      call PPNCGUARD2L(fxye,nyp,kstrt,nvp,nnxe,nypmx)
      call PPCGUARD2XL(fxye,nyp,nx,ndim,nxe,nypmx)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tguard = tguard + time
!
! push particles: updates part, wke, and ihole
      call dtimer(dtime,itime,-1)
      wke = 0.0
      call PPGPPUSHF2L(ppart,fxye,kpic,ncl,iholep,noff,nyp,qbme,dt,wke, &
     &nx,ny,mx,my,idimp,nppmx0,nxe,nypmx,mx1,mxyp1,ntmaxp,irc)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tpush = tpush + time
      if (irc /= 0) then
         write (*,*) kstrt, 'PPGPPUSHF2L error: irc=', irc
         call PPABORT()
         stop
      endif
!
! reorder particles by tile with OpenMP
! first part of particle reorder on x and y cell with mx, my tiles:
! updates ppart, ppbuff, ncl, iholep, irc, sbufl, sbufr, ncll, nclr
      call dtimer(dtime,itime,-1)
      call PPPORDERF2LA(ppart,ppbuff,sbufl,sbufr,ncl,iholep,ncll,nclr,  &
     &idimp,nppmx0,mx1,myp1,npbmx,ntmaxp,nbmaxp,irc)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tsort = tsort + time
      if (irc /= 0) then
         write (*,*) kstrt,'PPPORDERF2LA error: ntmaxp, irc=',ntmaxp,irc
         call PPABORT()
         stop
      endif
! move particles into appropriate spatial regions:
! updates rbufr, rbufl, mcll, mclr
      call dtimer(dtime,itime,-1)
      call PPPMOVE2(sbufr,sbufl,rbufr,rbufl,ncll,nclr,mcll,mclr,kstrt,  &
     &nvp,idimp,nbmaxp,mx1)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tmov = tmov + time
! second part of particle reorder on x and y cell with mx, my tiles:
! updates ppart, kpic
      call dtimer(dtime,itime,-1)
      call PPPORDER2LB(ppart,ppbuff,rbufl,rbufr,kpic,ncl,iholep,mcll,   &
     &mclr,idimp,nppmx0,mx1,myp1,npbmx,ntmaxp,nbmaxp,irc)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tsort = tsort + time
      if (irc /= 0) then
         write (*,*) kstrt,'PPPORDER2LB error: nppmx0, irc=',nppmx0,irc
         call PPABORT()
         stop
      endif
!
! energy diagnostic
      wtot(1) = we
      wtot(2) = wke
      wtot(3) = 0.0
      wtot(4) = we + wke
      call PPSUM(wtot,work,4)
      we = wtot(1)
      wke = wtot(2)
      if (ntime==0) then
         if (kstrt==1) then
            write (*,*) 'Initial Field, Kinetic and Total Energies:'
            write (*,'(3e14.7)') we, wke, wke + we
         endif
      endif
      ntime = ntime + 1
      go to 500
 2000 continue
!
! * * * end main iteration loop * * *
```

```
!
      if (kstrt==1) then
         write (*,*) 'ntime = ', ntime
         write (*,*) 'MPI nodes nvp = ', nvp
         write (*,*) 'Final Field, Kinetic and Total Energies:'
         write (*,'(3e14.7)') we, wke, wke + we
!
         write (*,*)
         write (*,*) 'deposit time = ', tdpost
         write (*,*) 'guard time = ', tguard
         write (*,*) 'solver time = ', tfield
         write (*,*) 'fft and transpose time = ', tfft(1), tfft(2)
         write (*,*) 'push time = ', tpush
         write (*,*) 'particle move time = ', tmov
         write (*,*) 'sort time = ', tsort
         tfield = tfield + tguard + tfft(1)
         write (*,*) 'total solver time = ', tfield
         time = tdpost + tpush + tmov + tsort
         write (*,*) 'total particle time = ', time
         wt = time + tfield
         write (*,*) 'total time = ', wt
         write (*,*)
!
         wt = 1.0e+09/(real(nloop)*real(np))
         write (*,*) 'Push Time (nsec) = ', tpush*wt
         write (*,*) 'Deposit Time (nsec) = ', tdpost*wt
         write (*,*) 'Sort Time (nsec) = ', tsort*wt
         write (*,*) 'Total Particle Time (nsec) = ', time*wt
      endif
!
 3000 continue
      call PPEXIT()
      end program
```