```
!-----------------------------------------------------------------------
! Skeleton 2-1/2D Electromagnetic GPU-MPI PIC code
! written by Viktor K. Decyk, UCLA
      program gpufpbpic2
      use fgpulib2
      use fgpupbpush2
      use fgpupfft2
      use pbpush2_h
      use pplib2       ! use with pplib2.f90
!     use pplib2_h     ! use with pplib2.f
      use fgpplib2
      use dtimer_c
      implicit none
      integer, parameter :: indx =   9, indy =   9
      integer, parameter :: npx =  3072, npy =   3072
      integer, parameter :: ndim = 3
      real, parameter :: tend = 10.0, dt = 0.04, qme = -1.0
      real, parameter :: vtx = 1.0, vty = 1.0, vx0 = 0.0, vy0 = 0.0
      real, parameter :: vtz = 1.0, vz0 = 0.0
      real :: ax = .912871, ay = .912871, ci = 0.1
! idimp = dimension of phase space = 5
! relativity = (no,yes) = (0,1) = relativity is used
      integer :: idimp = 5, ipbc = 1, relativity = 1
! idps = number of partition boundaries
      integer :: idps = 2
      real :: wke = 0.0, we = 0.0, wf = 0.0, wm = 0.0, wt = 0.0
! sorting tiles
      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, nxh1, nxe, nye, nxeh
      integer :: nxyh, nxhy, ny1, mx1, ntime, nloop, isign, ierr
      real :: qbme, affp, dth
      real, dimension(1) :: ssum
      double precision :: np
!
! declare scalars for MPI code
      integer :: nvp, idproc, kstrt, npmax, kxp, kyp, nypmx, nypmn
      integer :: nyp, noff, npp, nps
      integer :: myp1, mxyp1, kxpp, kypp
!
! declare scalars for GPU code
      integer :: nblock = 128
! nscache = (0,1,2) = (no,small,big) cache size
      integer :: nscache = 1
      integer :: mmcc, nppmx, nppmx0, nbmaxp, ntmaxp, npbmx
      integer :: nxhd, kxpd, idev, ndev
      integer, dimension(1) :: irc
!
! declare arrays for standard code
      real, dimension(:,:), pointer :: part => null()
      complex, dimension(:,:), pointer :: ffct => null()
      integer, dimension(:), pointer :: mixup => null()
```

```
      complex, dimension(:), pointer :: sct => null()
      real, dimension(7) :: wtot, work
!
! declare arrays for MPI code
      real, dimension(:), pointer  :: edges => null()
      integer, dimension(:), pointer :: locl => null()
      complex, pinned, dimension(:), allocatable :: scs, scr
      real, pinned, dimension(:), allocatable :: sbufl, sbufr
      real, pinned, dimension(:), allocatable :: rbufl, rbufr
!
! declare arrays for GPU code
      real, device, dimension(:,:), allocatable :: g_qe
      real, device, dimension(:,:,:), allocatable :: g_cue
      real, device, dimension(:,:,:), allocatable :: g_fxyze, g_bxyze
      complex, device, dimension(:,:), allocatable :: g_q, g_qt
      complex, device, dimension(:,:,:), allocatable :: g_cu, g_cut
      complex, device, dimension(:,:,:), allocatable :: g_fxyz, g_hxyz
      complex, device, dimension(:,:,:), allocatable :: g_fxyzt, g_hxyzt
      complex, device, dimension(:,:,:), allocatable :: g_exyzt, g_bxyzt
      complex, device, dimension(:,:), allocatable :: g_ffct
      integer, device, dimension(:), allocatable :: g_mixup
      complex, device, dimension(:), allocatable :: g_sct
      real, device, dimension(:), allocatable:: g_wke, g_we, g_wf, g_wm
      real, device, dimension(:,:,:), allocatable :: g_ppart, g_ppbuff
      integer, device, dimension(:), allocatable :: g_kpic
      real, device, dimension(:), allocatable :: g_sbufl, g_sbufr
      integer, device, dimension(:,:), allocatable :: g_ncl
      integer, device, dimension(:,:,:), allocatable :: g_ihole
      integer, device, dimension(:,:), allocatable :: g_ncll, g_nclr
      complex, device, dimension(:,:), allocatable :: g_bsm, g_brm
      complex, device, dimension(:), allocatable :: g_scs
      real, device, dimension(:), allocatable :: g_sum
      integer, device, dimension(:), allocatable :: g_irc
      complex, dimension(:,:), pointer :: qt => null()
      complex, dimension(:,:,:), pointer :: fxyzt => null()
      real, dimension(:,:,:), pointer :: ppart => null()
      integer, dimension(:), pointer :: kpic => null()
      integer, dimension(:,:), pointer :: ncll => null(), nclr => null()
      integer, dimension(:,:), pointer :: mcll => null(), mclr => null()
      complex, pinned, dimension(:,:), allocatable :: bsm, brm
!
! declare and initialize timing data
      real :: time
      type (timeval) :: itime
      double precision :: dtime
      real :: tdpost = 0.0, tguard = 0.0, tsort = 0.0
      real :: tfield = 0.0, tdjpost = 0.0, tpush = 0.0
      real :: tmov = 0.0
      real, dimension(2) :: tmsort = 0.0
      real, dimension(2) :: tfft = 0.0
!
! initialize scalars for standard code
      np =  dble(npx)*dble(npy)
      nx = 2**indx; ny = 2**indy; nxh = nx/2; nyh = ny/2
```

```
      nxh1 = nxh + 1
      nxe = nx + 2; nye = ny + 2; nxeh = nxe/2
      nxyh = max(nx,ny)/2; nxhy = max(nxh,ny); ny1 = ny + 1
      mx1 = (nx - 1)/mx + 1
      nloop = tend/dt + .0001; ntime = 0
      qbme = qme
      affp = dble(nx)*dble(ny)/np
      dth = 0.0
! set size for FFT arrays
      nxhd = nxh1
!
! 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
! kxpd = number of complex grids in fft partition in x direction
      kxp = (nxh - 1)/nvp + 1
! set size for FFT arrays
      kxpd = (nxh1 - 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
! kxpp/kypp = actual size of GPU field partition
      kxpp = min(kxpd,max(0,nxhd-kxpd*idproc))
      kypp = min(kyp,max(0,ny-kyp*idproc))
!
```

```
! allocate and initialize data for standard code
      allocate(part(idimp,npmax))
      allocate(ffct(nyh,kxpd))
      allocate(mixup(nxhy),sct(nxyh))
      allocate(kpic(mxyp1))
      allocate(qt(ny,kxpd),fxyzt(ny,ndim,kxpd))
!
! allocate and initialize data for MPI code
      allocate(locl(nvp))
!
! set up GPU
      irc = 0
! get unique GPU device ids
      call PPFNDGRP(locl,kstrt,nvp,idev,ndev)
      if (idev < 0) then
         write (*,*) kstrt,'GPU device id error!'
         call PPABORT()
         stop
      endif
      call fgpu_setgbsize(nblock)
      call init_cuf(idev,irc(1))
      if (irc(1) /= 0) then
         write (*,*) kstrt,'CUDA initialization error!'
         call PPABORT()
         stop
      endif
! obtain compute capability
      mmcc = fgetmmcc()
      if (mmcc < 20) then
         write (*,*) kstrt, 'compute capability 2.x or higher required'
         call PPABORT()
         stop
      endif
! set cache size
      call fgpu_set_cache_size(nscache)
! create asynchronous streams
      call fgpu_initstream(1)
      call fgpu_initstream(2)
      call fgpu_initstream(3)
! allocate and initialize data for GPU code
      allocate(g_qe(nxe,nypmx),g_cue(ndim,nxe,nypmx))
      allocate(g_fxyze(ndim,nxe,nypmx),g_bxyze(ndim,nxe,nypmx))
      allocate(g_ffct(nyh,kxpd),g_mixup(nxhy),g_sct(nxyh))
      allocate(g_q(nxhd,kyp),g_cu(nxhd,ndim,kyp))
      allocate(g_qt(ny,kxpd),g_cut(ny,ndim,kxpd))
      allocate(g_fxyz(nxhd,ndim,kyp),g_hxyz(nxhd,ndim,kyp))
      allocate(g_fxyzt(ny,ndim,kxpd),g_hxyzt(ny,ndim,kxpd))
      allocate(g_exyzt(ny,ndim,kxpd),g_bxyzt(ny,ndim,kxpd))
      allocate(g_wke(mxyp1),g_we(kxpd),g_wf(kxpd),g_wm(kxpd))
      allocate(g_sum(1))
!
! prepare fft tables
      call WPFFT2RINIT(mixup,sct,indx,indy,nxhy,nxyh)
! prepare NVIDIA ffts
```

```
      call fgpupfft2rrcuinit(nx,kypp,ndim)
      call fgpupfft2cuinit(kxpp,ny,ndim)
! calculate form factors
      isign = 0
      call PPOIS23T(qt,fxyzt,isign,ffct,ax,ay,affp,we,nx,ny,kstrt,ny,   &
     &kxpd,nyh)
! copy in solver arrays to GPU
      g_mixup = mixup
      g_sct = sct
      g_ffct = ffct
! initialize electrons
      nps = 1
      npp = 0
      call PDISTR2H(part,edges,npp,nps,vtx,vty,vtz,vx0,vy0,vz0,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
!
! initialize transverse electromagnetic fields
      g_exyzt = cmplx(0.0,0.0)
      g_bxyzt = cmplx(0.0,0.0)
!
! 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(1))
      if (irc(1) /= 0) then
         write (*,*) kstrt, 'PPDBLKP2L error, irc=', irc(1)
         call PPABORT()
         stop
      endif
! allocate vector particle data
      nppmx0 = (1.0 + xtras)*nppmx
      ntmaxp = 0.5*xtras*nppmx
      npbmx = 0.5*xtras*nppmx
      nbmaxp = 0.25*mx1*npbmx
! align data to warp size
      nppmx0 = 32*((nppmx0 - 1)/32 + 1)
      ntmaxp = 32*(ntmaxp/32 + 1)
      npbmx = 32*((npbmx - 1)/32 + 1)
      nbmaxp = 32*((nbmaxp - 1)/32 + 1)
      allocate(g_ppart(nppmx0,idimp,mxyp1),g_ppbuff(npbmx,idimp,mxyp1))
      allocate(g_kpic(mxyp1))
      allocate(g_sbufl(nbmaxp*idimp),g_sbufr(nbmaxp*idimp))
      allocate(g_ncl(8,mxyp1),g_ihole(2,ntmaxp+1,mxyp1))
      allocate(g_ncll(3,mx1),g_nclr(3,mx1))
      allocate(g_bsm(kxpd*ndim*kyp,nvp),g_brm(kxpd*ndim*kyp,nvp))
      allocate(g_scs(nxeh*ndim))
      allocate(g_irc(1))
      allocate(ppart(nppmx0,idimp,mxyp1))
```

```
      allocate(ncll(3,mx1),nclr(3,mx1),mcll(3,mx1),mclr(3,mx1))
!
! allocate host page-locked memory for GPU-MPI buffers
      allocate(scs(nxeh*ndim),scr(nxeh*ndim))
      allocate(sbufl(idimp*nbmaxp),sbufr(idimp*nbmaxp))
      allocate(rbufl(idimp*nbmaxp),rbufr(idimp*nbmaxp))
      allocate(bsm(kxpd*ndim*kyp,nvp),brm(kxpd*ndim*kyp,nvp))
!
! copy ordered particle data for GPU code: updates ppart, kpic
      call PPPMOVIN2LT(part,ppart,kpic,npp,noff,nppmx0,idimp,npmax,mx,my&
     &,mx1,mxyp1,irc(1))
      if (irc(1) /= 0) then
         write (*,*) kstrt, 'PPPMOVIN2LT overflow error, irc=', irc(1)
         call PPABORT()
         stop
      endif
! sanity check
      call PPPCHECK2LT(ppart,kpic,noff,nyp,idimp,nppmx0,nx,mx,my,mx1,   &
     &myp1,irc(1))
      if (irc(1) /= 0) then
         write (*,*) kstrt, 'PPPCHECK2LT error: irc=', irc(1)
         call PPABORT()
         stop
      endif
! copy to GPU
      g_irc = irc
      g_ppart = ppart
      g_kpic = kpic
      call fgpu_zfmem(g_we,kxpd)
      call fgpu_zfmem(g_wf,kxpd)
      call fgpu_zfmem(g_wm,kxpd)
!
      if (dt > 0.45*ci) then
         if (kstrt==1) then
            write (*,*) 'Warning: Courant condition may be exceeded!'
         endif
      endif
!
! * * * start main iteration loop * * *
!
  500 if (nloop <= ntime) go to 2000
!     if (kstrt==1) write (*,*) 'ntime = ', ntime
!
! deposit current with GPU code: updates g_ppart, g_cue
      call dtimer(dtime,itime,-1)
      call fgpu_zfmem(g_cue,ndim*nxe*nypmx)
      if (relativity==1) then
         call fgpu2pprjppost2l(g_ppart,g_cue,g_kpic,noff,qme,dth,ci,    &
     &nppmx0,idimp,nx,ny,mx,my,nxe,nypmx,mx1,mxyp1,ipbc)
      else
         call fgpu2ppjppost2l(g_ppart,g_cue,g_kpic,noff,qme,dth,nppmx0, &
     &idimp,nx,ny,mx,my,nxe,nypmx,mx1,mxyp1,ipbc)
      endif
      call dtimer(dtime,itime,1)
```

```
      time = real(dtime)
      tdjpost = tdjpost + time
!
! reorder particles by tile with GPU code:
! updates g_ppart, g_ppbuff, g_kpic, g_ncl, g_ihole, and g_irc,
! as well as various buffers
      call GPPORDER2L(g_ppart,g_ppbuff,g_sbufl,g_sbufr,g_kpic,g_ncl,    &
     &g_ihole,g_ncll,g_nclr,sbufl,sbufr,rbufl,rbufr,ncll,nclr,mcll,mclr,&
     &tmsort,noff,nyp,kstrt,nvp,idimp,nppmx0,nx,ny,mx,my,mx1,myp1,npbmx,&
     &ntmaxp,nbmaxp,g_irc)
      tsort = tmsort(1)
      tmov = tmsort(2)
!
! deposit charge with GPU code: updates g_qe
      call dtimer(dtime,itime,-1)
      call fgpu_zfmem(g_qe,nxe*nypmx)
      call fgpu2ppgppost2l(g_ppart,g_qe,g_kpic,noff,qme,idimp,nppmx0,mx,&
     &my,nxe,nypmx,mx1,mxyp1)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tdpost = tdpost + time
!
! add and copy guard cells with GPU code: updates g_cu, g_q
      call dtimer(dtime,itime,-1)
      call GPPCACGUARD2L(g_cu,g_cue,g_scs,scs,scr,nx,nyp,kstrt,nvp,ndim,&
     &nxe,nypmx,nxhd,kyp)
      call GPPCAGUARD2L(g_q,g_qe,g_scs,scs,scr,nx,nyp,kstrt,nvp,nxe,    &
     &nypmx,nxhd,kyp)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tguard = tguard + time
!
! transform charge to fourier space with GPU code: updates g_q, g_qt,
! as well as various buffers
      isign = -1
      call WAPPFFT2RCS(g_q,g_qt,g_bsm,g_brm,bsm,brm,isign,g_mixup,g_sct,&
     &tfft,indx,indy,kstrt,nvp,kxpd,kyp,nxhd,ny,kyp,nxhy,nxyh)
! NVIDIA fft
!     call GPUPPFFT2RRCU(g_q,g_qt,g_bsm,g_brm,bsm,brm,isign,tfft,indx,  &
!    &indy,kstrt,nvp,kxpd,kyp,nxhd,ny,kyp)
!
! transform current to fourier space with GPU code: updates g_cu, g_cut,
! as well as various buffers
      isign = -1
      call WAPPFFT2RCSN(g_cu,g_cut,g_bsm,g_brm,bsm,brm,isign,g_mixup,   &
     &g_sct,tfft,indx,indy,kstrt,nvp,ndim,kxpd,kyp,nxhd,ny,kyp,nxhy,nxyh&
     &)
! NVIDIA fft
!     call GPUPPFFT2RRCUN(g_cu,g_cut,g_bsm,g_brm,bsm,brm,isign,tfft,indx&
!    &,indy,kstrt,nvp,ndim,kxpd,kyp,nxhd,ny,kyp)
!
! take transverse part of current with GPU code: updates g_cut
      call dtimer(dtime,itime,-1)
      call fgpuppcuperp2t(g_cut,nx,ny,kstrt,ny,kxpd)
```

```
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tfield = tfield + time
!
! calculate electromagnetic fields in fourier space with GPU code:
! updates g_exyzt, g_bxyzt, g_wf, g_wm
      call dtimer(dtime,itime,-1)
      if (ntime==0) then
         call fgpuippbpoisp23t(g_cut,g_bxyzt,g_ffct,ci,g_wm,nx,ny,kstrt,&
     &ny,kxpd,nyh)
         wf = 0.0
         dth = 0.5*dt
      else
         call fgpuppmaxwel2t(g_exyzt,g_bxyzt,g_cut,g_ffct,affp,ci,dt,   &
     &g_wf,g_wm,nx,ny,kstrt,ny,kxpd,nyh)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tfield = tfield + time
!
! calculate force/charge in fourier space with GPU code:
! updates g_fxyzt, g_we
      call dtimer(dtime,itime,-1)
      call fgpuppois23t(g_qt,g_fxyzt,g_ffct,g_we,nx,ny,kstrt,ny,kxpd,nyh&
     &)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tfield = tfield + time
!
! add longitudinal and transverse electric fields with with GPU code:
! updates g_fxyzt
      call dtimer(dtime,itime,-1)
      isign = 1
      call fgpuppemfield2t(g_fxyzt,g_exyzt,g_ffct,isign,nx,ny,kstrt,ny, &
     &kxpd,nyh)
! copy magnetic field with GPU code: updates g_hxyzt
      isign = -1
      call fgpuppemfield2t(g_hxyzt,g_bxyzt,g_ffct,isign,nx,ny,kstrt,ny, &
     &kxpd,nyh)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tfield = tfield + time
!
! transform force to real space with GPU code: updates g_fxyz, g_fxyzt
! as well as various buffers
      isign = 1
      call WAPPFFT2RCSN(g_fxyz,g_fxyzt,g_bsm,g_brm,bsm,brm,isign,g_mixup&
     &,g_sct,tfft,indx,indy,kstrt,nvp,ndim,kxpd,kyp,nxhd,ny,kyp,nxhy,   &
     &nxyh)
! NVIDIA fft
!     call GPUPPFFT2RRCUN(g_fxyz,g_fxyzt,g_bsm,g_brm,bsm,brm,isign,tfft,&
!    &indx,indy,kstrt,nvp,ndim,kxpd,kyp,nxhd,ny,kyp)
!
! transform magnetic field to fourier space with GPU code:
```

```
! updates g_hxyz, g_hxyzt, as well as various buffers
      isign = 1
      call WAPPFFT2RCSN(g_hxyz,g_hxyzt,g_bsm,g_brm,bsm,brm,isign,g_mixup&
     &,g_sct,tfft,indx,indy,kstrt,nvp,ndim,kxpd,kyp,nxhd,ny,kyp,nxhy,   &
     &nxyh)
! NVIDIA fft
!     call GPUPPFFT2RRCUN(g_hxyz,g_hxyzt,g_bsm,g_brm,bsm,brm,isign,tfft,&
!    &indx,indy,kstrt,nvp,ndim,kxpd,kyp,nxhd,ny,kyp)
!
! copy guard cells with GPU code: updates g_fxyze, g_bxyze
      call dtimer(dtime,itime,-1)
      call GPPCBGUARD2L(g_fxyz,g_fxyze,g_scs,scs,scr,nx,nyp,kstrt,nvp,  &
     &ndim,nxe,nypmx,nxhd,kyp)
      call GPPCBGUARD2L(g_hxyz,g_bxyze,g_scs,scs,scr,nx,nyp,kstrt,nvp,  &
     &ndim,nxe,nypmx,nxhd,kyp)
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tguard = tguard + time
!
! push particles with GPU code: updates g_ppart, g_wke
      call dtimer(dtime,itime,-1)
      if (relativity==1) then
         call fgpuppgrbppush23l(g_ppart,g_fxyze,g_bxyze,g_kpic,noff,nyp,&
     &qbme,dt,dth,ci,g_wke,idimp,nppmx0,nx,ny,mx,my,nxe,nypmx,mx1,mxyp1,&
     &ipbc)
      else
         call fgpuppgbppush23l(g_ppart,g_fxyze,g_bxyze,g_kpic,noff,nyp, &
     &qbme,dt,dth,g_wke,idimp,nppmx0,nx,ny,mx,my,nxe,nypmx,mx1,mxyp1,   &
     &ipbc)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tpush = tpush + time
!
! reorder particles by tile with GPU code:
! updates g_ppart, g_ppbuff, g_kpic, g_ncl, g_ihole, and g_irc,
! as well as various buffers
      call GPPORDER2L(g_ppart,g_ppbuff,g_sbufl,g_sbufr,g_kpic,g_ncl,    &
     &g_ihole,g_ncll,g_nclr,sbufl,sbufr,rbufl,rbufr,ncll,nclr,mcll,mclr,&
     &tmsort,noff,nyp,kstrt,nvp,idimp,nppmx0,nx,ny,mx,my,mx1,myp1,npbmx,&
     &ntmaxp,nbmaxp,g_irc)
      tsort = tmsort(1)
      tmov = tmsort(2)
!
! sanity check
      irc = g_irc(1)
      if (irc(1) /= 0) then
         write (*,*) kstrt, 'GPPORDER2L error: irc=', irc(1)
         call PPABORT()
         stop
      endif
!
! energy diagnostic
      if (ntime==0) then
```

```
         call fgpu_zfmem(g_sum,1)
         call fgpusum2(g_we,g_sum,kxpd)
         we = g_sum(1)
         call fgpu_zfmem(g_sum,1)
         call fgpusum2(g_wf,g_sum,kxpd)
         wf = g_sum(1)
         call fgpu_zfmem(g_sum,1)
         call fgpusum2(g_wm,g_sum,kxpd)
         wm = g_sum(1)
         call fgpu_zfmem(g_sum,1)
         call fgpusum2(g_wke,g_sum,mxyp1)
         wke = g_sum(1)
         wt = we + wf + wm
         wtot(1) = wt
         wtot(2) = wke
         wtot(3) = 0.0
         wtot(4) = wke + wt
         wtot(5) = we
         wtot(6) = wf
         wtot(7) = wm
         call PPSUM(wtot,work,7)
         wke = wtot(2)
         we = wtot(5)
         wf = wtot(6)
         wm = wtot(7)
         if (kstrt==1) then
            wt = we + wf + wm
            write (*,*) 'Initial Total Field, Kinetic and Total Energies&
     &:'
            write (*,'(3e14.7)') wt, wke, wke + wt
            write (*,*) 'Initial Electrostatic, Transverse Electric and &
     &Magnetic Field Energies:'
            write (*,'(3e14.7)') we, wf, wm
         endif
       endif
       ntime = ntime + 1
       go to 500
 2000 continue
!
! * * * end main iteration loop * * *
!
! energy diagnostic
       call fgpu_zfmem(g_sum,1)
       call fgpusum2(g_we,g_sum,kxpd)
       we = g_sum(1)
       call fgpu_zfmem(g_sum,1)
       call fgpusum2(g_wf,g_sum,kxpd)
       wf = g_sum(1)
       call fgpu_zfmem(g_sum,1)
       call fgpusum2(g_wm,g_sum,kxpd)
       wm = g_sum(1)
       call fgpu_zfmem(g_sum,1)
       call fgpusum2(g_wke,g_sum,mxyp1)
       wke = g_sum(1)
```

```
      wt = we + wf + wm
      wtot(1) = wt
      wtot(2) = wke
      wtot(3) = 0.0
      wtot(4) = wke + wt
      wtot(5) = we
      wtot(6) = wf
      wtot(7) = wm
      call PPSUM(wtot,work,7)
      wke = wtot(2)
      we = wtot(5)
      wf = wtot(6)
      wm = wtot(7)
      if (kstrt==1) then
         write (*,*) 'ntime, relativity = ', ntime, relativity
         write (*,*) 'MPI nodes nvp = ', nvp, ', GPUs per host = ', ndev
         wt = we + wf + wm
         write (*,*) 'Final Total Field, Kinetic and Total Energies:'
         write (*,'(3e14.7)') wt, wke, wke + wt
         write (*,*) 'Final Electrostatic, Transverse Electric and Magne&
     &tic Field Energies:'
         write (*,'(3e14.7)') we, wf, wm
         write (*,*)
!
         write (*,*) 'deposit time = ', tdpost
         write (*,*) 'current deposit time = ', tdjpost
         tdpost = tdpost + tdjpost
         write (*,*) 'total deposit time = ', tdpost
         write (*,*) 'guard time = ', tguard
         write (*,*) 'solver time = ', tfield
         write (*,*) 'fft times = ', sum(tfft), tfft
         write (*,*) 'push time = ', tpush
         write (*,*) 'move time = ', tmov
         write (*,*) 'sort time = ', tsort
         tfield = tfield + tguard + sum(tfft)
         write (*,*) 'total solver time = ', tfield
         time = tdpost + tpush + tsort + tmov
         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 (*,*) 'Move Time (nsec) = ', tmov*wt
         write (*,*) 'Total Particle Time (nsec) = ', time*wt
      endif
!
! close down NVIDIA fft
      call fgpupfft2cudel()
      call fgpupfft2rrcudel()
! deallocate memory on GPU
```

```
      deallocate(g_irc,g_scs,g_brm,g_bsm,g_nclr,g_ncll,g_ihole,g_ncl)
      deallocate(g_sbufr,g_sbufl,g_kpic,g_ppbuff,g_ppart)
      deallocate(g_sum,g_wm,g_wf,g_we,g_wke)
      deallocate(g_bxyzt,g_exyzt,g_hxyzt,g_fxyzt,g_hxyz,g_fxyz,g_cut)
      deallocate(g_qt,g_cu,g_q,g_sct,g_mixup,g_ffct,g_bxyze,g_fxyze)
      deallocate(g_cue,g_qe)
! deallocate host page-locked memory
      deallocate(scs,scr,sbufl,sbufr,rbufl,rbufr,bsm,brm)
 3000 continue
!
! delete asynchronous streams
      call fgpu_delstream(3)
      call fgpu_delstream(2)
      call fgpu_delstream(1)
! close down GPU
      call end_cuf()
! close down MPI
      call PPEXIT()
!
      stop
      end program
```