

```

!-----
! Skeleton 2-1/2D Electromagnetic GPU-MPI PIC code
! written by Viktor K. Decyk, UCLA
    program gpupbpic2
    use gpulib2_h
    use gpupbpush2_h
    use gpupfft2_h
    use pbpush2_h
    use pplib2      ! use with pplib2.f90
!   use pplib2_h    ! use with pplib2.f
    use gpplib2
    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
    complex, dimension(:,:), pointer :: fft
    integer, dimension(:), pointer :: mixup
    complex, dimension(:), pointer :: sct

```

```

        real, dimension(7) :: wtot, work
!
! declare arrays for MPI code
    real, dimension(:,:), pointer :: sbuf1, sbuf, rbuf1, rbuf
    real, dimension(:), pointer :: edges
    real, dimension(:), pointer :: scs, scr
    integer, dimension(:), pointer :: loc1
!
! declare arrays for GPU code
    integer, dimension(2) :: g_ge = 0.0
    integer, dimension(2) :: g_cue = 0.0, g_fxyze = 0.0, g_bxyze = 0.0
    integer, dimension(2) :: g_ffct = 0.0
    integer, dimension(2) :: g_mixup = 0.0, g_sct = 0.0
    integer, dimension(2) :: g_q = 0.0, g_cu = 0.0
    integer, dimension(2) :: g_qt = 0.0, g_cut = 0.0
    integer, dimension(2) :: g_fxyz = 0.0, g_hxyz = 0.0
    integer, dimension(2) :: g_fxyzt = 0.0, g_hxyzt = 0.0
    integer, dimension(2) :: g_exyzt = 0.0, g_bxyzt = 0.0
    integer, dimension(2) :: g_wke = 0.0, g_we = 0.0
    integer, dimension(2) :: g_wf = 0.0, g_wm = 0.0
    integer, dimension(2) :: g_ppart = 0.0, g_ppbuff = 0.0
    integer, dimension(2) :: g_kpic = 0.0
    integer, dimension(2) :: g_sbuf1 = 0.0, g_sbuf = 0.0
    integer, dimension(2) :: g_ncl = 0.0, g_ihole = 0.0
    integer, dimension(2) :: g_nc11 = 0.0, g_nclr = 0.0
    integer, dimension(2) :: g_bsm = 0.0, g_brm = 0.0
    integer, dimension(2) :: g_scs = 0.0
    integer, dimension(2) :: g_sum = 0.0
    integer, dimension(2) :: g_irc = 0.0
    complex, dimension(:,:), pointer :: qt
    complex, dimension(:,:,:), pointer :: fxyzt
    real, dimension(:,:,:), pointer :: ppart
    integer, dimension(:), pointer :: kpic
    integer, dimension(:,:), pointer :: nc11, nclr, mc11, mclr
    complex, dimension(:,:), pointer :: bsm, brm
!
! declare and initialize timing data
    real :: time
    integer, dimension(4) :: itime
    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
    double precision :: dtime
!
! 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
        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*myy1
! kxpp/kyp = actual size of GPU field partition
        kxpp = min(kxpd,max(0,nxhd-kxpd*idproc))
        kyp = 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 gpu_setgbsize(nblock)
    call init_cu(idev,irc(1))
    if (irc(1) /= 0) then
        write (*,*) kstrt,'CUDA initialization error!'
        call PPABORT()
        stop
    endif
! obtain compute capability
    mmcc = getmmcc()
    if (mmcc < 20) then
        write (*,*) kstrt, 'compute capability 2.x or higher required'
        call PPABORT()
        stop
    endif
! set cache size
    call gpu_set_cache_size(nscache)
! create asynchronous streams
    call gpu_initstream(1)
    call gpu_initstream(2)
    call gpu_initstream(3)
! allocate and initialize data for GPU code
    call gpu_fallocate(g_ge,nxe*nypmx,irc(1))
    call gpu_fallocate(g_cue,ndim*nxe*nypmx,irc(1))
    call gpu_fallocate(g_fxyze,ndim*nxe*nypmx,irc(1))
    call gpu_fallocate(g_bxyze,ndim*nxe*nypmx,irc(1))
    call gpu_callocate(g_ffct,nyh*kxpd,irc(1))
    call gpu_iallocate(g_mixup,nxhy,irc(1))
    call gpu_callocate(g_sct,nxyh,irc(1))
    call gpu_callocate(g_q,nxhd*kyp,irc(1))
    call gpu_callocate(g_cu,nxhd*ndim*kyp,irc(1))
    call gpu_callocate(g_qt,ny*kxpd,irc(1))
    call gpu_callocate(g_cut,ny*ndim*kxpd,irc(1))
    call gpu_callocate(g_fxyz,nxhd*ndim*kyp,irc(1))
    call gpu_callocate(g_hxyz,nxhd*ndim*kyp,irc(1))
    call gpu_callocate(g_fxyzt,ny*ndim*kxpd,irc(1))
    call gpu_callocate(g_hxyzt,ny*ndim*kxpd,irc(1))
    call gpu_callocate(g_exyzt,ny*ndim*kxpd,irc(1))
    call gpu_callocate(g_bxyzt,ny*ndim*kxpd,irc(1))
    call gpu_fallocate(g_wke,mxyp1,irc(1))
    call gpu_fallocate(g_we,kxpd,irc(1))
    call gpu_fallocate(g_wf,kxpd,irc(1))

```

```

    call gpu_fallocate(g_wm,kxpd,irc(1))
    call gpu_fallocate(g_sum,1,irc(1))
    if (irc(1) /= 0) then
        write (*,*) kstrt, 'GPU allocate error!'
        call PPABORT()
        stop
    endif
!
! prepare fft tables
    call WPFFT2RINIT(mixup,sct,indx,indy,nxhy,nxyh)
! prepare NVIDIA ffts
    call gpupfft2rrcuinit(nx,kypp,ndim)
    call gpupfft2cuinit(kxpp,ny,ndim)
! calculate form factors
    isign = 0
    call PPOIS23T(qt,fxyz,tsign,ffct,ax,ay,affp,we,nx,ny,kstrt,ny, &
        &kxpd,nyh)
! copy in solver arrays to GPU
    call gpu_icopyin(mixup,g_mixup,nxhy)
    call gpu_ccopyin(sct,g_sct,nxyh)
    call gpu_ccopyin(ffct,g_ffct,nyh*kxpd)
! initialize electrons
    nps = 1
    npp = 0
    call PDISTR2H(part,edges,npp,nps,vtx,vty,vtz,vx0,vy0,vz0,npx,npj, &
        &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
    call gpu_zcmem(g_exyz,ny*ndim*kxpd)
    call gpu_zcmem(g_bxyz,ny*ndim*kxpd)
!
! find number of particles in each of mx, my tiles: updates kplic, nppmx
    call PPDBLKP2L(part,kpic,npp,noff,nppmx,idimp,npmax,mx,my,mx1, &
        &mxypl,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)
call gpu_fallocate(g_ppart,nppmx0*idimp*mxyp1,irc(1))
call gpu_fallocate(g_ppbuff,npbmx*idimp*mxyp1,irc(1))
call gpu_iallocate(g_kpic,mxyp1+1,irc(1))
call gpu_fallocate(g_sbuf1,nbmaxp*idimp,irc(1))
call gpu_fallocate(g_sbufr,nbmaxp*idimp,irc(1))
call gpu_iallocate(g_ncl,8*mxyp1,irc(1))
call gpu_iallocate(g_ihole,2*(ntmaxp+1)*mxyp1,irc(1))
call gpu_iallocate(g_nc11,3*mx1,irc(1))
call gpu_iallocate(g_nclr,3*mx1,irc(1))
call gpu_callocate(g_bsm,kxpd*ndim*kyp*nvp,irc(1))
call gpu_callocate(g_brm,kxpd*ndim*kyp*nvp,irc(1))
call gpu_fallocate(g_scs,nxe*ndim,irc(1))
call gpu_iallocate(g_irc,1,irc(1))
if (irc(1) /= 0) then
    write (*,*) kstrt, 'GPU allocate error!'
    call PPABORT()
    stop
endif
allocate(ppart(nppmx0,idimp,mxyp1))
allocate(nc11(3,mx1),nclr(3,mx1),mc11(3,mx1),mclr(3,mx1))

!
! allocate data for GPU-MPI buffers
!     allocate(scs(nxe*ndim),scr(nxe*ndim))
!     allocate(sbuf1(idimp,nbmaxp),sbufr(idimp,nbmaxp)
!     allocate(rbuf1(idimp,nbmaxp),rbufr(idimp,nbmaxp)
!     allocate(bsm(kxpd*ndim*kyp,nvp),brm(kxpd*ndim*kyp,nvp))
! allocate host page-locked memory for GPU-MPI buffers
call hpl_f1allocate(scs,nxe*ndim,irc(1))
call hpl_f1allocate(scr,nxe*ndim,irc(1))
call hpl_f2allocate(sbuf1,idimp,nbmaxp,irc(1))
call hpl_f2allocate(sbufr,idimp,nbmaxp,irc(1))
call hpl_f2allocate(rbuf1,idimp,nbmaxp,irc(1))
call hpl_f2allocate(rbufr,idimp,nbmaxp,irc(1))
call hpl_c2allocate(bsm,kxpd*ndim*kyp,nvp,irc(1))
call hpl_c2allocate(brm,kxpd*ndim*kyp,nvp,irc(1))
if (irc(1) /= 0) then
    write (*,*) kstrt, 'hpl allocate error, irc=', irc(1)
    call PPABORT()
    stop
endif

!
! 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, &
&mxyp1,irc(1))

```

```

        if (irc(1) /= 0) then
            write (*,*) kstrt, 'PPPCHECK2LT error: irc=', irc(1)
            call PPABORT()
            stop
        endif
! copy to GPU
    call gpu_icopyin(irc,g_irc,1)
    call gpu_fcopyin(ppart,g_ppart,nppmx0*idimp*mxyp1)
    call gpu_icopyin(kpic,g_kpic,mxyp1)
    call gpu_zfmem(g_we,kxpd)
    call gpu_zfmem(g_wf,kxpd)
    call gpu_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 gpu_zfmem(g_cue,ndim*nxe*nypmx)
    if (relativity==1) then
        call cgpu2pprjppost2l(g_ppart,g_cue,g_kpic,noff,qme,dth,ci,      &
&nppmx0,idimp,nx,ny,mx,my,nxe,nypmx,mx1,mxyp1,ipbc)
    else
        call cgpu2ppjppost2l(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_sbuf1,g_sbufr,g_kpic,g_ncl,      &
&g_ihole,g_ncl1,g_nclr,sbuf1,sbufr,rbuf1,rbufr,ncl1,nclr,mcl1,mclr, &
&tmsort,noff,nyp,kstrt,nvp,idimp,nppmx0,nx,ny,mx,my,mx1,my1,npbm, &
&ntmaxp,nbmaxp,g_irc)
    tmsort = tmsort(1)
    tmov = tmsort(2)
!
! deposit charge with GPU code: updates g_ge
    call dtimer(dtime,itime,-1)
    call gpu_zfmem(g_ge,nxe*nypmx)
    call cgpu2ppgppost2l(g_ppart,g_ge,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 GPPCAGUARD2L(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 WAPPPFFT2RCS(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 WAPPPFFT2RCSN(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 cgpuppcuperp2t(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 cgpuippbpoisp23t(g_cut,g_bxyzt,g_ffct,ci,g_wm,nx,ny,kstrt,&
&ny,kxpd,nyh)
            wf = 0.0
            dth = 0.5*dt
        else
            call cgpuppmaxwel2t(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 cgpuppois23t(g_gt,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 cgpuppemfield2t(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 cgpuppemfield2t(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 WAPPPFFT2RCSN(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 WAPPPFFT2RCSN(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)
    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 cgpuppgbpush231(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,mryp1, &
&ipbc)
    else
        call cgpuppgbpush231(g_ppart,g_fxyze,g_bxyze,g_kpic,noff,nyp, &
&qbme,dt,dth,g_wke,idimp,nppmx0,nx,ny,mx,my,nxe,nypmx,mx1,mryp1, &
&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_sbuf1,g_sbuf,g_kpic,g_ncl, &
&g_ihole,g_ncl1,g_ncl,sbuf1,sbuf,rbuf1,rbuf,ncl1,ncl,mcl1,mcl, &
&tmsort,noff,nyp,kstrt,nvp,idimp,nppmx0,nx,ny,mx,my,mx1,mryp1,npbm, &
&ntmaxp,nbmaxp,g_irc)
    tmsort = tmsort(1)
    tmov = tmsort(2)
!
! sanity check
    call gpu_icopyout(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 gpu_zfmem(g_sum,1)
        call cgpusum2(g_we,g_sum,kxpd)
        call gpu_fcopyout(ssum,g_sum,1); we = ssum(1)
        call gpu_zfmem(g_sum,1)
        call cgpusum2(g_wf,g_sum,kxpd)
        call gpu_fcopyout(ssum,g_sum,1); wf = ssum(1)
        call gpu_zfmem(g_sum,1)
        call cgpusum2(g_wm,g_sum,kxpd)
        call gpu_fcopyout(ssum,g_sum,1); wm = ssum(1)
        call gpu_zfmem(g_sum,1)
        call cgpusum2(g_wke,g_sum,mryp1)
        call gpu_fcopyout(ssum,g_sum,1); wke = ssum(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 gpu_zfmem(g_sum,1)
    call cgpsum2(g_we,g_sum,kxpd)
    call gpu_fcyyout(ssum,g_sum,1); we = ssum(1)
    call gpu_zfmem(g_sum,1)
    call cgpsum2(g_wf,g_sum,kxpd)
    call gpu_fcyyout(ssum,g_sum,1); wf = ssum(1)
    call gpu_zfmem(g_sum,1)
    call cgpsum2(g_wm,g_sum,kxpd)
    call gpu_fcyyout(ssum,g_sum,1); wm = ssum(1)
    call gpu_zfmem(g_sum,1)
    call cgpsum2(g_wke,g_sum,mxyp1)
    call gpu_fcyyout(ssum,g_sum,1); wke = ssum(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 gpupfft2cudel()
    call gpupfft2rrcudel()
! deallocate memory on GPU
    call gpu_deallocate(g_irc,irc(1))
    call gpu_deallocate(g_scs,irc(1))
    call gpu_deallocate(g_brm,irc(1))
    call gpu_deallocate(g_bsm,irc(1))
    call gpu_deallocate(g_nclr,irc(1))
    call gpu_deallocate(g_nc11,irc(1))
    call gpu_deallocate(g_ihole,irc(1))
    call gpu_deallocate(g_ncl,irc(1))
    call gpu_deallocate(g_sbufr,irc(1))
    call gpu_deallocate(g_sbufl,irc(1))
    call gpu_deallocate(g_kpic,irc(1))
    call gpu_deallocate(g_ppbuff,irc(1))
    call gpu_deallocate(g_ppart,irc(1))
    call gpu_deallocate(g_sum,irc(1))
    call gpu_deallocate(g_wm,irc(1))
    call gpu_deallocate(g_wf,irc(1))
    call gpu_deallocate(g_we,irc(1))
    call gpu_deallocate(g_wke,irc(1))
    call gpu_deallocate(g_bxyzt,irc(1))
    call gpu_deallocate(g_exyzt,irc(1))

```

```

    call gpu_deallocate(g_hxyz,irc(1))
    call gpu_deallocate(g_fxzyt,irc(1))
    call gpu_deallocate(g_hxyz,irc(1))
    call gpu_deallocate(g_fxzy,irc(1))
    call gpu_deallocate(g_cut,irc(1))
    call gpu_deallocate(g_qt,irc(1))
    call gpu_deallocate(g_cu,irc(1))
    call gpu_deallocate(g_q,irc(1))
    call gpu_deallocate(g_sct,irc(1))
    call gpu_deallocate(g_mixup,irc(1))
    call gpu_deallocate(g_ffct,irc(1))
    call gpu_deallocate(g_bxyze,irc(1))
    call gpu_deallocate(g_fxzye,irc(1))
    call gpu_deallocate(g_cue,irc(1))
    call gpu_deallocate(g_ge,irc(1))
! deallocate host page-locked memory
    call hpl_deallocate(scs,irc(1)); nullify(scs)
    call hpl_deallocate(scr,irc(1)); nullify(scr)
    call hpl_deallocate(sbuf1,irc(1)); nullify(sbuf1)
    call hpl_deallocate(sbufr,irc(1)); nullify(sbufr)
    call hpl_deallocate(rbuf1,irc(1)); nullify(rbuf1)
    call hpl_deallocate(rbufr,irc(1)); nullify(rbufr)
    call hpl_deallocate(bsm,irc(1)); nullify(bsm)
    call hpl_deallocate(brm,irc(1)); nullify(brm)
3000 continue
!
! delete asynchronous streams
    call gpu_delstream(3)
    call gpu_delstream(2)
    call gpu_delstream(1)
! close down GPU
    call end_cu()
! close down MPI
    call PPEXIT()
!
    stop
    end program
!
    subroutine getfcptr(cref,carray,nx)
! set reference to C data in 1d real Fortran pointer object
    implicit none
    integer :: nx
    real, dimension(nx), target :: carray
    real, dimension(:), pointer :: cref
    cref => carray
    end subroutine
!
    subroutine getf2cptr(cref,carray,nx,ny)
! set reference to C data in 2d real Fortran pointer object
    implicit none
    integer :: nx, ny
    real, dimension(nx,ny), target :: carray
    real, dimension(:,,:), pointer :: cref
    cref => carray

```

```
        end subroutine
!
    subroutine getc2cptr(cref,carray,nx,ny)
! set reference to C data in 2d complex Fortran pointer object
    implicit none
    integer :: nx, ny
    complex, dimension(nx,ny), target :: carray
    complex, dimension(:,:), pointer :: cref
    cref => carray
    end subroutine
```