```
!-----------------------------------------------------------------------
! Skeleton 2D Electrostatic OpenMP/Vector PIC code
! written by Viktor K. Decyk, UCLA and Ricardo Fonseca, ISCTE
      program vmpic2
      use sseflib2_h
      use ssempush2_h
      use vmpush2_h
      use omplib_h
      implicit none
! indx/indy = exponent which determines grid points in x/y direction:
! nx = 2**indx, ny = 2**indy.
      integer, parameter :: indx =   9, indy =   9
! npx/npy = number of electrons distributed in x/y direction.
      integer, parameter :: npx =  3072, npy =   3072
! ndim = number of velocity coordinates = 2
      integer, parameter :: ndim = 2
! tend = time at end of simulation, in units of plasma frequency.
! dt = time interval between successive calculations.
! qme = charge on electron, in units of e.
      real, parameter :: tend = 10.0, dt = 0.1, qme = -1.0
! vtx/vty = thermal velocity of electrons in x/y direction
! vx0/vy0 = drift velocity of electrons in x/y direction.
      real, parameter :: vtx = 1.0, vty = 1.0, vx0 = 0.0, vy0 = 0.0
! ax/ay = smoothed particle size in x/y direction
      real :: ax = .912871, ay = .912871
! idimp = number of particle coordinates = 4
! ipbc = particle boundary condition: 1 = periodic
      integer :: idimp = 4, ipbc = 1
! wke/we/wt = particle kinetic/electric field/total energy
      real :: wke = 0.0, we = 0.0, wt = 0.0
! mx/my = number of grids in x/y in sorting tiles
      integer :: mx = 16, my = 16
! xtras = fraction of extra particles needed for particle management
      real :: xtras = 0.2
! kvec = (1,2) = run (autovector,SSE2) version
      integer :: kvec = 1

! declare scalars for standard code
      integer :: np, nx, ny, nxh, nyh, nxe, nye, nxeh, nxyh, nxhy
      integer :: mx1, my1, mxy1, ntime, nloop, isign, lvect
      integer :: irc = 0
      real :: qbme, affp
!
! declare scalars for OpenMP code
      integer :: nppmx, nppmx0, ntmax, npbmx
      integer :: nvp
!
! declare arrays for standard code:
! part = original particle array
      real, dimension(:,:), pointer :: part
! qe = electron charge density with guard cells
      real, dimension(:,:), pointer :: qe
! fxye = smoothed electric field with guard cells
      real, dimension(:,:,:), pointer :: fxye
```

```
! ffc = form factor array for poisson solver
      complex, dimension(:,:), pointer :: ffc
! mixup = bit reverse table for FFT
      integer, dimension(:), pointer :: mixup
! sct = sine/cosine table for FFT
      complex, dimension(:), pointer :: sct
!
! declare arrays for OpenMP (tiled) code:
! ppartt = tiled particle array
! ppbuff = buffer array for reordering tiled particle array
      real, dimension(:,:,:), pointer :: ppartt, ppbuff
! kpic = number of particles in each tile
      integer, dimension(:), pointer :: kpic
! ncl = number of particles departing tile in each direction
      integer, dimension(:,:), pointer :: ncl
! ihole = location/destination of each particle departing tile
      integer, dimension(:,:,:), pointer :: ihole
! kp = original location of reordered particle
      integer, dimension(:,:), pointer :: kp
!
! declare and initialize timing data
      real :: time
      integer, dimension(4) :: itime
      real :: tdpost = 0.0, tguard = 0.0, tfft = 0.0, tfield = 0.0
      real :: tpush = 0.0, tsort = 0.0
      double precision :: dtime
!
      irc = 0
! nvp = number of shared memory nodes (0=default)
      nvp = 0
!     write (*,*) 'enter number of nodes:'
!     read (5,*) nvp
! initialize for shared memory parallel processing
      call INIT_OMP(nvp)
!
! initialize scalars for standard code
! np = total number of particles in simulation
! nx/ny = number of grid points in x/y direction
      np = npx*npy; nx = 2**indx; ny = 2**indy; nxh = nx/2; nyh = ny/2
      nxe = nx + 2; nye = ny + 1; nxeh = nxe/2
      nxyh = max(nx,ny)/2; nxhy = max(nxh,ny)
! mx1/my1 = number of tiles in x/y direction
      mx1 = (nx - 1)/mx + 1; my1 = (ny - 1)/my + 1; mxy1 = mx1*my1
! nloop = number of time steps in simulation
! ntime = current time step
      nloop = tend/dt + .0001; ntime = 0
      qbme = qme
      affp = real(nx*ny)/real(np)
!
! allocate data for standard code
      allocate(part(idimp,np))
      allocate(mixup(nxhy),sct(nxyh))
      allocate(kpic(mxy1))
!
```

```
      lvect = 4
! allocate vector field data
      nxe = lvect*((nxe - 1)/lvect + 1)
      nxeh = nxe/2
      call sse_f2allocate(qe,nxe,nye,irc)
      call sse_f3allocate(fxye,ndim,nxe,nye,irc)
      call sse_c2allocate(ffc,nxh,nyh,irc)
      if (irc /= 0) then
         write (*,*) 'aligned field allocation error: irc = ', irc
      endif
!
! prepare fft tables
      call WFFT2RINIT(mixup,sct,indx,indy,nxhy,nxyh)
! calculate form factors
      isign = 0
      call VMPOIS22(qe,fxye,isign,ffc,ax,ay,affp,we,nx,ny,nxeh,nye,nxh, &
     &nyh)
! initialize electrons
      call DISTR2(part,vtx,vty,vx0,vy0,npx,npy,idimp,np,nx,ny,ipbc)
!
! find number of particles in each of mx, my tiles: updates kpic, nppmx
      call DBLKP2L(part,kpic,nppmx,idimp,np,mx,my,mx1,mxy1,irc)
      if (irc /= 0) then
         write (*,*) 'DBLKP2L error, irc=', irc
         stop
      endif
! allocate vector particle data
      nppmx0 = (1.0 + xtras)*nppmx
      ntmax = xtras*nppmx
      npbmx = xtras*nppmx
! align data for Vector Processor
      nppmx0 = lvect*((nppmx0 - 1)/lvect + 1)
      ntmax = lvect*(ntmax/lvect + 1)
      npbmx = lvect*((npbmx - 1)/lvect + 1)
      call sse_f3allocate(ppartt,nppmx0,idimp,mxy1,irc)
      call sse_f3allocate(ppbuff,npbmx,idimp,mxy1,irc)
      allocate(ncl(8,mxy1))
      allocate(ihole(2,ntmax+1,mxy1))
      allocate(kp(nppmx0,mxy1))
      if (irc /= 0) then
         write (*,*) 'aligned particle allocation error: irc = ', irc
      endif
!
! copy ordered particle data for OpenMP: updates ppartt, kpic, and kp
      call PPMOVIN2LTP(part,ppartt,kpic,kp,nppmx0,idimp,np,mx,my,mx1,   &
     &mxy1,irc)
      if (irc /= 0) then
         write (*,*) 'PPMOVIN2LTP overflow error, irc=', irc
         stop
      endif
! sanity check
      call PPCHECK2LT(ppartt,kpic,idimp,nppmx0,nx,ny,mx,my,mx1,my1,irc)
      if (irc /= 0) then
         write (*,*) 'PPCHECK2LT error: irc=', irc
         stop
      endif
!
! * * * start main iteration loop * * *
!
  500 if (nloop <= ntime) go to 2000
!     write (*,*) 'ntime = ', ntime
!
! deposit charge with OpenMP: updates qe
      call dtimer(dtime,itime,-1)
      qe = 0.0
      if (kvec==1) then
         call VGPPOST2LT(ppartt,qe,kpic,qme,nppmx0,idimp,mx,my,nxe,nye, &
     &mx1,mxy1)
! SSE2 function
      else if (kvec==2) then
         call csse2gppost2lt(ppartt,qe,kpic,qme,nppmx0,idimp,mx,my,nxe, &
     &nye,mx1,mxy1)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tdpost = tdpost + time
!
! add guard cells with OpenMP: updates qe
      call dtimer(dtime,itime,-1)
      if (kvec==1) then
         call AGUARD2L(qe,nx,ny,nxe,nye)
! SSE2 function
      else if (kvec==2) then
         call csse2aguard2l(qe,nx,ny,nxe,nye)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tguard = tguard + time
!
! transform charge to fourier space with OpenMP: updates qe
      call dtimer(dtime,itime,-1)
      isign = -1
      if (kvec==1) then
         call WFFT2RVMX(qe,isign,mixup,sct,indx,indy,nxeh,nye,nxhy,nxyh)
! SSE2 function
      else if (kvec==2) then
         call csse2wfft2rmx(qe,isign,mixup,sct,indx,indy,nxeh,nye,nxhy, &
     &nxyh)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tfft = tfft + time
!
! calculate force/charge in fourier space with OpenMP: updates fxye, we
      call dtimer(dtime,itime,-1)
      isign = -1
      if (kvec==1) then
         call VMPOIS22(qe,fxye,isign,ffc,ax,ay,affp,we,nx,ny,nxeh,nye,  &
     &nxh,nyh)
! SSE2 function
      else if (kvec==2) then
         call csse2mpois22(qe,fxye,isign,ffc,ax,ay,affp,we,nx,ny,nxeh,  &
     &nye,nxh,nyh)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tfield = tfield + time
!
! transform force to real space with OpenMP: updates fxye
      call dtimer(dtime,itime,-1)
      isign = 1
      if (kvec==1) then
         call WFFT2RVM2(fxye,isign,mixup,sct,indx,indy,nxeh,nye,nxhy,   &
     &nxyh)
! SSE2 function
      else if (kvec==2) then
         call csse2wfft2rm2(fxye,isign,mixup,sct,indx,indy,nxeh,nye,nxhy&
     &,nxyh)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tfft = tfft + time
!
! copy guard cells with OpenMP: updates fxye
      call dtimer(dtime,itime,-1)
      if (kvec==1) then
         call CGUARD2L(fxye,nx,ny,nxe,nye)
! SSE2 function
      else if (kvec==2) then
         call csse2cguard2l(fxye,nx,ny,nxe,nye)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tguard = tguard + time
!
! push particles with OpenMP:
      wke = 0.0
      call dtimer(dtime,itime,-1)
! updates ppartt, wke
!     if (kvec==1) then
!        call VGPPUSH2LT(ppartt,fxye,kpic,qbme,dt,wke,idimp,nppmx0,nx,ny&
!    &,mx,my,nxe,nye,mx1,mxy1,ipbc)
! SSE2 function
!     else if (kvec==2) then
!        call csse2gppush2lt(ppartt,fxye,kpic,qbme,dt,wke,idimp,nppmx0, &
!    &nx,ny,mx,my,nxe,nye,mx1,mxy1,ipbc)
!     endif
! updates ppartt, ncl, ihole, wke, irc
      if (kvec==1) then
         call VGPPUSHF2LT(ppartt,fxye,kpic,ncl,ihole,qbme,dt,wke,idimp, &
     &nppmx0,nx,ny,mx,my,nxe,nye,mx1,mxy1,ntmax,irc)
! SSE2 function
```

```
      else if (kvec==2) then
         call csse2gppushf2lt(ppartt,fxye,kpic,ncl,ihole,qbme,dt,wke,   &
     &idimp,nppmx0,nx,ny,mx,my,nxe,nye,mx1,mxy1,ntmax,irc)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tpush = tpush + time
      if (irc /= 0) then
         write (*,*) 'VGPPUSHF2LT error: irc=', irc
         stop
      endif
!
! reorder particles by tile with OpenMP:
      call dtimer(dtime,itime,-1)
! updates ppartt, ppbuff, kpic, ncl, ihole, and irc
!     if (kvec==1) then
!        call VPPORDER2LT(ppartt,ppbuff,kpic,ncl,ihole,idimp,nppmx0,nx, &
!    &ny,mx,my,mx1,my1,npbmx,ntmax,irc)
! SSE2 function
!     else if (kvec==2) then
!        call csse2pporder2lt(ppartt,ppbuff,kpic,ncl,ihole,idimp,nppmx0,&
!    &nx,ny,mx,my,mx1,my1,npbmx,ntmax,irc)
!     endif
! updates ppartt, ppbuff, kpic, ncl, and irc
      if (kvec==1) then
         call VPPORDERF2LT(ppartt,ppbuff,kpic,ncl,ihole,idimp,nppmx0,mx1&
     &,my1,npbmx,ntmax,irc)
! SSE2 function
      else if (kvec==2) then
         call csse2pporderf2lt(ppartt,ppbuff,kpic,ncl,ihole,idimp,nppmx0&
     &,mx1,my1,npbmx,ntmax,irc)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tsort = tsort + time
      if (irc /= 0) then
         write (*,*) 'VPPORDERF2LT error: ntmax, irc=', ntmax, irc
         stop
      endif
!
      if (ntime==0) then
         write (*,*) 'Initial Field, Kinetic and Total Energies:'
         write (*,'(3e14.7)') we, wke, wke + we
      endif
      ntime = ntime + 1
      go to 500
 2000 continue
!
! * * * end main iteration loop * * *
!
      write (*,*) 'ntime = ', ntime, 'kvec = ', kvec
      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 time = ', tfft
      write (*,*) 'push time = ', tpush
      write (*,*) 'sort time = ', tsort
      tfield = tfield + tguard + tfft
      write (*,*) 'total solver time = ', tfield
      time = tdpost + tpush + 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
      write (*,*)
!
      call sse_deallocate(ppartt); nullify(ppartt)
      call sse_deallocate(ppbuff); nullify(ppbuff)
      call sse_deallocate(ffc); nullify(ffc)
      call sse_deallocate(fxye); nullify(fxye)
      call sse_deallocate(qe); nullify(qe)
!
      stop
      end program
!
! Procedures to create Fortran90 pointers for data allocated in C.
! For details see V. K. Decyk, ACM Fortran Forum, vol. 27, no. 2 (2008).
      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 getf3cptr(cref,carray,nx,ny,nz)
! set reference to C data in 3d real Fortran pointer object
      implicit none
      integer :: nx, ny, nz
      real, dimension(nx,ny,nz), 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
!
      subroutine geti1cptr(cref,carray,nx)
! set reference to C data in 1d integer Fortran pointer object
      implicit none
      integer :: nx
      integer, dimension(nx), target :: carray
      integer, dimension(:), pointer :: cref
      cref => carray
      end subroutine
```