```
!-----------------------------------------------------------------------
! Skeleton 2D Electrostatic Vector PIC code
! written by Viktor K. Decyk, UCLA and Ricardo Fonseca, ISCTE
      program vpic2
      use sseflib2_h
      use ssepush2_h
      use vpush2_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
! sortime = number of time steps between standard electron sorting
      integer :: idimp = 4, ipbc = 1, sortime = 50
! wke/we/wt = particle kinetic/electric field/total energy
      real :: wke = 0.0, we = 0.0, wt = 0.0
! 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 :: npe, ny1, ntime, nloop, isign
      integer :: irc = 0
      real :: qbme, affp
!
! declare arrays for standard code:
! partt, partt2 = transposed particle arrays
      real, dimension(:,:), pointer :: partt, partt2, tpartt
! 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
! npicy = scratch array for reordering particles
      integer, dimension(:), pointer :: npicy
```

```
!
! 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
!
! 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); ny1 = ny + 1
! 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(mixup(nxhy),sct(nxyh))
!
! align memory for SSE
      npe = 4*((np - 1)/4 + 1)
      nxe = 4*((nxe - 1)/4 + 1)
      nxeh = nxe/2
      call sse_f2allocate(partt,npe,idimp,irc)
      if (sortime > 0) then
         call sse_f2allocate(partt2,npe,idimp,irc)
      endif
      call sse_f2allocate(qe,nxe,nye,irc)
      call sse_f3allocate(fxye,ndim,nxe,nye,irc)
      call sse_c2allocate(ffc,nxh,nyh,irc)
      call sse_i1allocate(npicy,ny1,irc)
      if (irc /= 0) then
         write (*,*) 'aligned allocation error: irc = ', irc
      endif
!
! prepare fft tables
      call WFFT2RINIT(mixup,sct,indx,indy,nxhy,nxyh)
! calculate form factors
      isign = 0
      call VPOIS22(qe,fxye,isign,ffc,ax,ay,affp,we,nx,ny,nxeh,nye,nxh,  &
     &nyh)
! initialize electrons
      call DISTR2T(partt,vtx,vty,vx0,vy0,npx,npy,idimp,npe,nx,ny,ipbc)
!
! * * * start main iteration loop * * *
!
  500 if (nloop <= ntime) go to 2000
!     write (*,*) 'ntime = ', ntime
!
! deposit charge with standard procedure: updates qe
```

```
      call dtimer(dtime,itime,-1)
      qe = 0.0
      if (kvec==1) then
         call VGPOST2LT(partt,qe,qme,np,npe,idimp,nxe,nye)
! SSE2 function
      else if (kvec==2) then
         call csse2gpost2lt(partt,qe,qme,np,npe,idimp,nxe,nye)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tdpost = tdpost + time
!
! add guard cells with standard procedure: 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 standard procedure: updates qe
      call dtimer(dtime,itime,-1)
      isign = -1
      if (kvec==1) then
         call WFFT2RVX(qe,isign,mixup,sct,indx,indy,nxeh,nye,nxhy,nxyh)
! SSE2 function
      else if (kvec==2) then
         call csse2wfft2rx(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 standard procedure:
! updates fxye
      call dtimer(dtime,itime,-1)
      isign = -1
      if (kvec==1) then
         call VPOIS22(qe,fxye,isign,ffc,ax,ay,affp,we,nx,ny,nxeh,nye,nxh&
     &,nyh)
! SSE2 function
      else if (kvec==2) then
         call csse2pois22(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 standard procedure: updates fxye
      call dtimer(dtime,itime,-1)
      isign = 1
      if (kvec==1) then
         call WFFT2RV2(fxye,isign,mixup,sct,indx,indy,nxeh,nye,nxhy,nxyh&
     &)
! SSE2 function
      else if (kvec==2) then
         call csse2wfft2r2(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 standard procedure: 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 standard procedure: updates part, wke
      wke = 0.0
      call dtimer(dtime,itime,-1)
      if (kvec==1) then
         call VGPUSH2LT(partt,fxye,qbme,dt,wke,idimp,np,npe,nx,ny,nxe,  &
     &nye,ipbc)
! SSE2 function
      else if (kvec==2) then
         call csse2gpush2lt(partt,fxye,qbme,dt,wke,idimp,np,npe,nx,ny,  &
     &nxe,nye,ipbc)
      endif
      call dtimer(dtime,itime,1)
      time = real(dtime)
      tpush = tpush + time
!
! sort particles by cell for standard procedure
      if (sortime > 0) then
         if (mod(ntime,sortime)==0) then
            call dtimer(dtime,itime,-1)
            if (kvec==1) then
               call DSORTP2YLT(partt,partt2,npicy,idimp,np,npe,ny1)
! SSE2 function
            else if (kvec==2) then
               call csse2dsortp2ylt(partt,partt2,npicy,idimp,np,npe,ny1)
            endif
! exchange pointers
            tpartt => partt
```

```
            partt => partt2
            partt2 => tpartt
            call dtimer(dtime,itime,1)
            time = real(dtime)
            tsort = tsort + time
         endif
      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
!
      call sse_deallocate(npicy); nullify(npicy)
      call sse_deallocate(ffc); nullify(ffc)
      call sse_deallocate(fxye); nullify(fxye)
      call sse_deallocate(qe); nullify(qe)
      if (sortime > 0) then
         call sse_deallocate(partt2); nullify(partt2)
      endif
      call sse_deallocate(partt); nullify(partt)
!
      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
!
```